UNIVERSIDADE TÉCNICA
DE LISBOA

INSTITUTO SUPERIOR TÉCNICO

CARNEGIE MELLON UNIVERSITY

Carnegie
Mellon
University

The Geometry of Constrained Structured Prediction: Applications to Inference and Learning of Natural Language Syntax

André Filipe Torres Martins

Thesis Committee:

Prof. Mário Alexandre Teles Figueiredo,	*Instituto Superior Técnico*
Prof. Noah Ashton Smith,	*Carnegie Mellon University*
Prof. Pedro Manuel Quintas Aguiar,	*Instituto Superior Técnico*
Prof. Eric Poe Xing,	*Carnegie Mellon University*
Prof. Isabel Maria Martins Trancoso,	*Instituto Superior Técnico*
Prof. Carlos Ernesto Guestrin,	*Carnegie Mellon University*
Prof. Michael Collins,	*Columbia University*

Thesis specifically prepared to obtain the Dual PhD Degree in Language Technology

Resumo

Esta tese propõe novos modelos e algoritmos para a predição de variáveis estruturadas, com ênfase em aplicações no processamento de linguagem natural. Apresentam-se contribuições em duas frentes: no problema de inferência, cujo objectivo é fazer uma predição a partir de um modelo estatístico, e no problema de aprendizagem, onde o modelo é treinado a partir dos dados.

No primeiro problema, propõe-se uma mudança de paradigma, considerando-se modelos expressivos com características e restrições globais, representáveis como modelos gráficos restritos. Introduz-se um novo algoritmo de inferência aproximada, o qual ignora os efeitos globais causados pelos ciclos do grafo. Esta metodologia é depois aplicada ao problema da análise sintáctica de texto, originando novos analisadores de dependências a que chamamos "turbo parsers", os quais atingem resultados ao nível do estado-da-arte.

No segundo problema, considera-se uma família de funções de custo que inclui os campos aleatórios condicionais, as máquinas de vectores de suporte, e o perceptrão estruturado, para a qual apresentamos novos algoritmos em linha que dispensam a especificação de uma taxa de aprendizagem. Em seguida, debruçamo-nos sobre o regularizador, o qual é utilizado para promover esparsidade estruturada e para aprender classificadores estruturados com múltiplos núcleos. Para o efeito, introduzem-se novos algoritmos do tipo gradiente-proximal, os quais têm a capacidade de explorar grandes espaços de características de forma eficiente, com reduzido consumo de memória. Como resultado, obtém-se um novo procedimento para a selecção de características-padrão, o qual conduz a modelos compactos e precisos.

Palavras-chave: Aprendizagem estatística, classificação estruturada, processamento de linguagem natural, sintaxe de dependências, modelos gráficos probabilísticos, programação linear inteira, decomposição dual, método das direcções alternadas, algoritmos gradiente-proximal, esparsidade estruturada.

Abstract

This thesis proposes new models and algorithms for structured output prediction, with an emphasis on natural language processing applications. We advance in two fronts: in the inference problem, whose aim is to make a prediction given a model, and in the learning problem, where the model is trained from data.

For inference, we make a paradigm shift, by considering rich models with global features and constraints, representable as constrained graphical models. We introduce a new approximate decoder that ignores global effects caused by the cycles of the graph. This methodology is then applied to syntactic analysis of text, yielding a new framework which we call "turbo parsing," with state-of-the-art results.

For learning, we consider a family of loss functions encompassing conditional random fields, support vector machines and the structured perceptron, for which we provide new online algorithms that dispense with learning rate hyperparameters. We then focus on the regularizer, which we use for promoting structured sparsity and for learning structured predictors with multiple kernels. We introduce online proximal-gradient algorithms that can explore large feature spaces efficiently, with minimal memory consumption. The result is a new framework for feature template selection yielding compact and accurate models.

Keywords: Machine learning, structured classification, natural language processing, dependency parsing, probabilistic graphical models, integer linear programming, dual decomposition, alternating directions method of multipliers, proximal gradient algorithms, structured sparsity.

Acknowledgments

This thesis would not exist without the help of many.

First of all, I would like to thank my quadruplet of advisors at IST and CMU, who have always pointed me not just to the North, but to all four cardinal directions: Mário Figueiredo, Pedro Aguiar, Noah Smith, and Eric Xing. Thanks to their diverse areas of expertise, I could expand horizons and learn about many different topics, which ended up cross-fertilizing and producing this thesis. I have been working with Mário and Pedro since the beginning of my research career. They have always provided truly insightful advice, whether the topic is computer vision, kernel methods, or optimization algorithms. Noah has always been there for enthusiastic discussions, either physically at CMU, or via Skype meetings while I was at IST. It was him who introduced me the problem of dependency parsing and taught me all I know about statistical NLP. I'm grateful to Eric for pointing out structured prediction as an exciting research topic. He has a tremendous insight about what are the open problems in machine learning that are worthwhile for pursuing research, and has been very encouraging and supportive of my research.

Thanks also to Carlos Guestrin and Michael Collins for agreeing being in my thesis committee. I have greatly benefited from discussions with both of them that materialized in this thesis. Carlos' work in optimization and graphical models has been a source of inspiration, and so has been the work done by Mike and his group on dual decomposition and statistical parsing.

During my two-year stay at CMU, I was very fortunate to participate in the ARK and SAILING research groups. I learned a lot from uncountable discussions with Shay Cohen, Dipanjan Das, Kevin Gimpel, Mike Heilman, Tae Yano, Nathan Schneider, Chris Dyer, Brendan O'Connor, Dani Yogatama, Jacob Eisenstein, Mladen Kolar, Kriti Puniyani, Amr Ahmed, Le Song, and Pradipta Ray, among others. Special thanks to Dipanjan and Kevin for collaborating with me in several projects. It was also a remarkable experience to interact with several other people in the LTI: Alan Black, Alon Lavie, Lori Levin, Stephen Vogel, William Cohen, Gopala Anumanchipalli, Sourish Chaudhuri, Ni Lao, Vítor Carvalho, Andreas Zollmann, Luis Marujo, Matthew Marge, and Konstantin Salomatin are just some of them. I am thankful to Bob Frederking and José Moura for helping to clarify several questions about the CMU-Portugal program, allowing me to make the most of this two-year experience. Thanks also to Dana Houston, Stacey Young, Radha Rao, Michelle Martin, Diane Stidle, Lori Spears, Nicole Hillard-Hudson, and remaining staff at LTI, MLD and ICTI. Special thanks to all my friends for the good times we have spent in Pittsburgh, in particular Vasco and Maria Calais Pedro, Luis Pedro Coelho, Rita Reis, the corresponding families, and the remaining Portuguese crew.

During the period of my PhD in Portugal, I had the opportunity of interacting with

many colleagues working in the fields of machine learning and NLP. At IST, I learned a lot from discussions with João Xavier, José Bioucas Dias, Ana Fred, Mariana Sá Correia, André Lourenço, João Graça, Ana Mendes, João Mota, Manya Afonso, and many others. João Paulo Costeira, Victor Barroso and Isabel Trancoso answered several questions about the CMU-Portugal program. Special thanks to João Graça and Luis Sarmento for co-organizing with me the first LxMLS (Lisbon Machine Learning School), a remarkable experience! I am also very grateful to Paulo Almeida, for teaching me so much about mathematics, geometry, and language. Now I realize that I need to learn much more.

I was fortunate to spend a semester interning at Google Research in New York. I am very thankful to my host, Slav Petrov, as well as the remaining people in the NLP group: Ryan McDonald, Kuzman Ganchev, Keith Hall, and Hao Zhang, with whom I had so many interesting discussions. During that period I was also lucky to interact with Sasha Rush, who was interning, Yoav Goldberg and Percy Liang, who were doing a post-doc, Afshin Rostamizadeh and Umar Syed, who had just started as research scientists, and Stephen Boyd, who was visiting.

Along the five years of my PhD, I benefited from many interesting discussions with researchers from other labs and universities, such as Jason Eisner, David Smith, David Sontag, Ben Taskar, Xavier Carreras, Terry Koo, David Hall, Taylor Berg-Kirkpatrick, John Blitzer, John DeNero, Sebastian Riedel, James Clarke, Ming-Wei Chang, Scott Yih, Kristina Toutanova, Hoifung Poon, Chris Meek, Chris Quirk, Partha Talukdar, Guy Lebanon, Cheng Soon Ong, Manuele Bicego, Geoff Gordon, Ben Recht, Reza Zadeh, Ofer Meshi, and Gholamreza Haffiri.

I would like to thank Priberam for all the support during the PhD years, in particular Afonso Mendes, who encouraged me to apply to the PhD program, and also Carlos Amaral and João Prieto for believing that research is an added value. Priberam has also sponsored the machine learning seminars at IST which I have been organizing since 2009.

My PhD studies were funded by a grant from Fundação para a Ciência e Tecnologia (FCT) and Information and Communication Technology Institute (ICTI) through the CMU-Portugal Program (SFRH/BD/36737/2007). Some work was partially supported by the FET programme (EU FP7), under the SIMBAD project (contract 213250), and by a FCT grant PTDC/EEA-TEL/72572/2006.

On a personal note, I am deeply grateful to my parents Teresa and Leonel Martins, who have always motivated me to study since my first day of school, an opportunity which they never had in their lifetimes. I am also thankful to my brother Luis. Obrigado por tudo!

Finally, I thank my loving and supportive family for sharing this journey with me. In particular, my wife Sarah, who encouraged me to apply to this PhD program and has made unconditional sacrifices to join me in Pittsburgh. Thank you for all your love and patience! I dedicate this thesis to my son Samuel, who gave his first steps in our house in Greenfield, and to my daugther Mariana, who was born after we came back, just a couple of days before the first LxMLS.

Contents

Table of Notation

$\mathbb{R}$	The set of real numbers
$\mathbb{R}_+$	The set of non-negative real numbers
$\mathbb{R}_{++}$	The set of strictly positive real numbers
ε	The empty string
Σ	An alphabet
Σ^*	The Kleene closure of Σ, $\Sigma^* = \{\varepsilon\} \cup \Sigma \cup \Sigma^2 \cup \dots$
$[\![\pi]\!]$	1 if predicate π is true, 0 otherwise
$\mathcal{A}, \mathcal{B}, \mathcal{C}, \dots$	Arbitrary sets
$\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \dots$	Arbitrary vectors of real numbers
$\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots$	Arbitrary matrices over the reals
$\mathcal{X}$	The input set
$\mathcal{Y}$	The output set
$h : \mathcal{X} \to \mathcal{Y}$	A classifier
$\mathcal{Y}(x)$	The set of admissible outputs for $x \in \mathcal{X}$
$\boldsymbol{f}(x, y)$	A joint feature vector
$\boldsymbol{w}$	The weight vector
$\boldsymbol{f}_p(x, y_p)$	A local feature vector at a place p
$Z(\boldsymbol{w}, x)$	The partition function of $P_{\boldsymbol{w}}(y \mid x)$
$\theta_p(y_p)$	A local score for a part (p, y_p)
$\boldsymbol{\theta}$	A score vector, corresponding to the canonical parameters
$\boldsymbol{\mu}$	A marginal vector, corresponding to the mean paramaters
$\mathcal{G} := (\mathcal{V}, \mathcal{F}, \mathcal{E})$	A factor graph with variable nodes $\mathcal{V}$, factor nodes $\mathcal{F}$, and edges $\mathcal{E}$
$\mathrm{MARG}(\mathcal{G})$	The marginal polytope of a factor graph $\mathcal{G}$
$\mathrm{LOCAL}(\mathcal{G})$	The local polytope of a factor graph $\mathcal{G}$

Chapter 1

Introduction

The last years have witnessed a remarkable progress in language technologies, well patent in the ever-increasing quality of search engines, question-answering systems, speech recognizers, and automatic translators. All these systems perform tasks that have to deal with structured and ambiguous textual data, where the "structure" emanates from the complex network of syntactic and semantic phenomena that underlies natural language. Despite the technological advances, we are far from the point where those problems can be declared "solved." To make computers capable of understanding and extracting meaning from text, a new generation of methods, models and algorithms is likely to be necessary.

So, what is the driving force behind the aforementioned progress? Essentially, it is the alliance of two important factors: the massive amount of *data* that became available with the advent of the Web, and the success of *machine learning* techniques to extract statistical models from the data (Mitchell, 1997; Manning and Schütze, 1999; Schölkopf and Smola, 2002; Bishop, 2006; Smith, 2011). As a consequence, a new paradigm has emerged in the last couple of decades, which directs attention to the data itself, as opposed to the explicit representation of knowledge (Abney, 1996; Pereira, 2000; Halevy et al., 2009). This data-centric paradigm has been extremely fruitful in *natural language processing* (NLP), and came to replace the classic knowledge representation methodology which was prevalent until the 1980s, based on symbolic rules written by experts. The increasing availability of human annotated corpora—of which the Penn Treebank (Marcus et al., 1993) is perhaps the most famous example—opened the door for the use of supervised machine learning methods. While earlier statistical models, such as hidden Markov models (Jelinek, 1997; Rabiner, 1989) and probabilistic context-free grammars (Charniak, 1996b; Baker, 1979) were over-simplistic, relying on too stringent independence assumptions, significant progress has been made recently using feature-based models for structured prediction. As a consequence, increasingly accurate models have been devised, with applications to syntax, semantics, and machine translation.

Structured prediction (Bakır et al., 2007) is a machine learning framework that provides a unified treatment for dealing with structured output variables, rooted in seminal work by Lafferty et al. (2001); Collins (2002a); Taskar et al. (2003); Altun et al. (2003); Tsochantaridis et al. (2004), among others. This framework paved the way for the development of increasingly sophisticated statistical language models, with fewer independence assumptions than earlier models, at the cost of more expensive estimation procedures. Today, there is a clear demand for more powerful models, motivated by the perception, from the scientific com-

munity, that there is a long tail of linguistic phenomena which can only be captured by incorporating *global* features and constraints into the model.[1] A current limitation is that the inclusion of this sort of features and constraints has a negative effect on the simplicity and efficiency of inference algorithms. For this reason, the progress in this area has been shaped by the ability of extending the existing algorithms, rather than being oriented by the design of features that can have impact on the quality of the model, as would be desirable. That is, the model design is typically *subjugated* by algorithmic considerations.

This thesis contributes to a paradigm shift by embracing principled forms of *approximate inference*. A general framework is proposed, based on the formalism of graphical models, which allows the easy incorporation of global features and declarative constraints as factors in the model. Then, new inference algorithms are proposed that are tailored to solve a linear relaxation of the original problem, while exploiting the structure of the graphical model. This framework is compatible with recently proposed formalisms that ally the robustness and semantics of statistical models and the expressive power of first order logic (Roth and Yih, 2004; Richardson and Domingos, 2006), bridging the gap between statistical and symbolic methods.

More broadly, we contribute new methods, models, and algorithms for statistical learning and inference with structured data. We give particular emphasis to applications in NLP, such as syntactic parsing, text segmentation, and recognition of named entities, all of which involve predicting structured output variables, such as trees or sequences. For modeling the syntax of languages, we introduce rich models with global features and constraints and embrace approximate inference through a new dual decomposition algorithm that is suitable for handling declarative constraints. We also consider the structure in the feature space, through the proposal of new learning algorithms that select feature templates and identify relevant patterns in the data.

Note, however, that structured data is by no means exclusive of natural language. There is a wide range of important problems, in areas such as computer vision, communication theory, computational biology, signal processing, and others, which are also characterized by the need of handling structured data and identifying complex interactions. The methods that we propose are quite general; it is thus likely that the contributions presented in this thesis will benefit other areas dealing with structured prediction.

1.1 Motivation and Related Work

There are several aspects that characterize structured prediction problems: the representational framework and its interplay with the inference procedure, the learning procedure that is employed for training the model, and the particular properties of the application that is being tackled. We next describe our motivations in each of these aspects and briefly pinpoint some of the related work; a more thorough description of prior work will appear in subsequent chapters, when we discuss each aspect in full detail. We will also highlight some of the open problems that this thesis addresses.

[1] By "global," in this context, we mean features capable of modeling long-range dependencies, which are not easy to incorporate in low-order Markov models.

1.1.1 Inference and Graphical Models

Many problems in NLP, computer vision, coding theory, computational biology, and other areas require predicting output variables which are strongly interdependent. Probabilistic graphical models (Pearl, 1988; Wainwright and Jordan, 2008; Koller and Friedman, 2009) are a framework that allows the unified treatment of such problems, combining tools from graph theory, statistics, and optimization.

Many advances were made recently by the graphical models community regarding approximate inference algorithms, both for marginal and MAP inference, based on sampling, heuristic search, or variational approximations. A line of research that has stimulated a lot of recent interest is the linear programming relaxation of Schlesinger (1976), which in this thesis we designate by "LP-MAP inference." Many message-passing and dual decomposition algorithms have been proposed recently to address this problem (Wainwright et al., 2005a; Kolmogorov, 2006; Komodakis et al., 2007; Globerson and Jaakkola, 2008; Rush et al., 2010). The advantage over other approximate algorithms is that the underlying optimization problem is well-understood and the algorithms are convergent or provide certain guarantees. Moreover, there are ways of tightening the relaxation toward the exact solution (Sontag et al., 2008).

Despite these advances, the case of *constrained* graphical models has barely been addressed, and it is not well understood how the approximate algorithms behave in the presence of declarative constraints. For example, it is not clear whether marginal and MAP inference are equally difficult, or one is more amenable to approximations than the other. Some previous work applied message-passing algorithms under global constraints, but it was tailored to specific applications, such as bipartite matching (Duchi et al., 2007) and dependency parsing (Smith and Eisner, 2008). A formal characterization of constrained graphical models is still missing, in particular one that is compatible with the kind of declarative constraints that can be encoded through symbolic rules. This lack of theoretical understanding contrasts with a number of practical achievements, brought by new representation formalisms such as probabilistic relational networks (Friedman et al., 1999), constrained conditional models (Roth and Yih, 2004), and Markov logic networks (Richardson and Domingos, 2006). It is likely that consistent advances in the field can only be achieved through a better understanding of the underlying approximations and their geometry.

Some open questions are:

- How to ally the graphical model machinery and the use of declarative constraints, *e.g.*, in first-order logic?
- Can we construct faster approximate algorithms, more amenable to the presence of constraints?
- Can we interpret those approximate algorithms geometrically, and write down the optimization problem that they are addressing?

1.1.2 Learning Structured Predictors

Given a graphical model for representing a structured output, a fundamental problem is that of learning its parameters. This is the main concern of structured prediction. In this thesis,

we focus on discriminative structured linear models, introduced in a string of seminal work starting with conditional random fields (Lafferty et al., 2001), and followed by the structured perceptron algorithm (Collins, 2002a), and structured support vector machines (Taskar et al., 2003; Altun et al., 2003; Tsochantaridis et al., 2004).

In the formalisms above, the learning problem corresponds to the minimization of a regularized empirical risk functional:

$$\text{minimize} \quad \Omega(\boldsymbol{w}) + \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{w}; x^n, y^n) \quad \text{with respect to } \boldsymbol{w}, \tag{1.1}$$

where $((x^n, y^n))_{n=1}^N$ is the training data, Ω is a regularizer and L is a loss function, parameterized by $\boldsymbol{w}$. Despite the progress in this area, many important questions remain unanswered.

- What is the "right" loss function for each problem? Is it worth interpolating between some of the losses that have been proposed (such as the logistic loss, hinge loss, *etc.*)?
- Which algorithms are more suitable for a large-scale setting (large N)? Currently, online algorithms such as the structured passive-aggressive algorithm (Crammer et al., 2006) or stochastic gradient descent (Bottou, 1991a; Vishwanathan et al., 2006) appear to be the most suitable, but can we design better ones?
- And what if we use an approximate inference algorithm as a subroutine—what would be the impact in the model that is learned?

Regarding the regularizer Ω, even more is left to be done. Most work so far either does not regularize, or uses a regularizer with the sole goal of controlling overfitting or producing a compact model. However, as recent work in sparse modeling suggests (Zhao et al., 2009; Jenatton et al., 2009; Bach et al., 2011), a clever design of the regularizer is an opportunity for exploiting rich prior knowledge, or for discovering patterns in the data. Very little has been done toward this goal in structured prediction and in NLP, where sparse modeling is still taking its first steps. The following questions remain unanswered:

- How to deal with the structure of the feature space? Can we promote structural patterns or discover patterns in the data through the regularizer?
- If so, how can we explore large feature spaces efficiently, without having to touch every feature? How to adapt online algorithms to solve the resulting optimization problem and exploit the sparsity?

1.1.3 Natural Language Processing

This thesis applies structured prediction models and algorithms to problems in NLP. Our preferred application is syntactic parsing, which has seen considerable progress in the last two decades. The development of treebanks for several languages, of which the Penn Treebank is an example (Marcus et al., 1993), made possible approaching the problem with supervised learning techniques. These treebanks contain tens of thousands of sentences paired with the corresponding syntactic parse trees, annotated by humans.

In this thesis, we adopt *dependency syntax* as the grammar formalism of choice. The historical origins of dependency syntax go back to the Sanskrit grammar of Pāṇini in the 4th

century BC, and its modern treatment is due to the seminal work of Tesnière (1959); Hudson (1984); Mel'čuk (1988), among others. It is a lightweight formalism which, due to its lexical motivation, relative ease of annotation, and the fact that it does not require the explicit construction of a grammar, presents computational advantages over other formalisms, such as phrase-structure grammars (Chomsky, 1965). Statistical dependency parsers have been first developed by Eisner (1996); McDonald et al. (2005b), and have since then gained prominence in NLP, with applications to relation extraction (Culotta and Sorensen, 2004), machine translation (Ding and Palmer, 2005), and question answering (Wang et al., 2007).

Increasingly accurate models for dependency parsing have been proposed lately, some of them while this thesis was in preparation (McDonald and Pereira, 2006; Smith and Eisner, 2008; Koo and Collins, 2010; Koo et al., 2010; Huang and Sagae, 2010). These models become more accurate by incorporating more global features, for example through an increase of horizontal or vertical Markov context. We will review this related work in detail in Chapter 2, with pointers to the literature.

Some open questions are:

- How can we incorporate global features that promote certain properties (such as nearly projective parse trees) when they are observed in the data?
- Are horizontal and vertical Markovian assumptions adequate for free-order languages? How can we still do inference beyond those assumptions?
- How can we inject prior knowledge, such as declarative constraints, without having to redesign inference algorithms from scratch?

1.2 Previous Publications

To make this dissertation a coherent piece, we omit some of the research work carried out during the doctoral studies. This includes work in information-theoretic kernels for structured inputs (Martins et al., 2008c,b, 2009a), later applied to image classification and pattern recognition problems (Bicego et al., 2010a; Martins et al., 2010a; Bicego et al., 2010b). Some work on stacked learning of dependency parsers (Martins et al., 2008a), and on document summarization through a joint model for sentence extraction and compression (Martins and Smith, 2009) was also omitted. Finally, we only mention very briefly some of our work on characterizing learning through approximate inference (Martins et al., 2009c).

1.3 Contributions and Thesis Statement

We next summarize the main contributions of this thesis, addressing some of the open problems mentioned in the previous section.

1. **We make graphical models capable of handling declarative constraints by introducing a simple set of logic factors.** Despite their simplicity, these factors constitute the building blocks for expressing arbitrary logic constraints. We derive analytical expressions for their messages, marginals, entropies, and marginal polytopes. In a global sense, we provide a formal characterization of constrained graphical models and their

geometry, extending the unconstrained case. Our results provide a unified treatment of isolated cases studied in some of the aforementioned previous work.

2. **We propose AD^3, a new dual decomposition algorithm for LP-MAP inference.** AD^3 is particularly suitable for models that cannot be lightly decomposed, typical when dealing with declarative constraints. It has the same modular architecture of previous dual decomposition algorithms, but it is faster to reach consensus, and it is suitable for embedding in a branch-and-bound procedure toward the true MAP. We derive efficient procedures for handling the above logic factors, turning AD^3 into a powerful tool for constrained structured prediction. We also introduce a new active set procedure for dealing with dense, large or combinatorial factors, whose only interface with the factor is an oracle for computing the local MAP.

3. **We cast dependency parsing as a concise integer program and introduce turbo parsers.** Through a multi-commodity flow formulation, we obtain a *polynomial* number of constraints in the integer linear program (ILP), contrasting with previous formulations, where this number was exponential. We incorporate rich global features and constraints into the model, leading to substantial performance gains. We show how the linear relaxation of the ILP, as well as other parsers recently proposed, are all instances of *turbo parsers*: parsers that perform approximate inference in loopy graphical models, ignoring global effects caused by the loops.[2]

4. **We generalize the MIRA algorithm for a wide family of loss functions.** This family includes support vector machines, conditional random fields, and the structured perceptron. The resulting algorithms are similar to online and stochastic gradient descent, but have the advantage of self-adjusting their stepsizes, dispensing the specification of a learning rate. They have an interpretation as dual coordinate ascent algorithms.

5. **We propose new online proximal-gradient algorithms for kernel learning and structured sparsity.** This addresses one of the open questions posed above, regarding the role of regularizers in the learning problem. We employ group-structured regularizers for learning combinations of multiple kernels and for identifying relevant feature templates. Our regularizers are designed to take into account the structure of the feature space and the desired sparsity patterns. Our approach can also handle overlapping groups of features, through sequential proximal steps. The online algorithms allow exploring large feature spaces with fast runtime and minimal memory requirements.

Thesis Statement. Our claim is that the structure of natural language can be effectively modeled with the machinery of structured prediction and graphical models. A vanilla application of the existing methods, however, is not sufficient. *First*, there is a long tail of linguistic phenomena that can only be captured by rich models with global features and constraints. Approximate algorithms based on relaxations are a principled way for making predictions with these rich models, liberating the model designer from the handcuffs of algorithmic concerns. *Second*, we argue that an effective exploration of large model spaces is possible without sacrificing computational efficiency of learning or accuracy, by endowing the learner with a regularizer that promotes structured sparsity.

[2] This strategy finds a parallel in error-correcting coding theory, where it underlies the famous turbo-codes.

1.4 Organization of the Thesis

This thesis is divided into four parts, which we describe below.

Part I: Background. The first part systematizes the previous work in structured prediction and NLP which is relevant for this thesis, providing the necessary background for better exposing our contributions in remaining chapters. It includes the following chapters:

- **Chapter 2** describes the NLP applications addressed in this thesis. All of them involve predicting some kind of structure.
- **Chapter 3** provides background on supervised learning and structured prediction. We describe structured linear models and online algorithms for learning them from data.
- **Chapter 4** describes inference in graphical models. We discuss in detail some approximate inference algorithms and the relaxed problems that they address.

The reader who is familiar with this background material might prefer to skip the aforementioned chapters in a first reading, and proceed to the second part.

Part II: Inference. This part addresses models and algorithms for *inference* with structured outputs, in which we present novel contributions. It is comprised of the following chapters:

- **Chapter 5** extends structured predictors and graphical models to deal with *constrained* outputs. It presents an expressive set of logic factors, along with closed-form expressions for computing messages, marginals, and other quantities of interest.
- **Chapter 6** introduces the AD^3 algorithm, analyzes its convergence properties, and shows how to deal with the logic factors introduced in Chapter 5. It also provides an active set method for arbitrary factors, presenting experiments in benchmark datasets.
- **Chapter 7** presents turbo parsers, along with a concise ILP formulation for non-projective dependency parsing and the corresponding constrained factor graph with logic factors. Detailed experiments are presented for 14 languages.

Part III: Learning. This part is about methods and algorithms for *learning* structured predictors. The following chapters are its constitution:

- **Chapter 8** introduces a new class of online learning algorithms that can deal with a wide family of structured loss functions. The algorithms are evaluated in named entity recognition and dependency parsing tasks.
- **Chapter 9** introduces a new online algorithm for learning structure predictors with multiple kernels, establishing its regret and convergence properties.
- **Chapter 10** proposes a new structured sparsity approach for selecting feature templates in structured prediction problems, adapting the algorithm introduced in Chapter 9. Experiments are presented in various sequence labeling tasks and in dependency parsing.

Part IV: Conclusions. This part concludes, by providing, in Chapter 11, a summary of contributions and drawing possible directions of future work.

Part I

Background

Chapter 2

Structure in Natural Language

This chapter provides a brief overview about the structured problems in natural language processing that this thesis addresses.

At the heart of this thesis is the word "structure," which derives from Latin *structura*, a form of the verb *struere* ("to build"). According to the Oxford dictionary, "structure" can refer to either the arrangement of parts that form something complex, or the actual object that is constructed from the several parts. We will adopt primarily the former sense, and talk about the object with structure as a "structured object."[1] Furthermore, we will designate by "structured set" a discrete set of objects that can be formed by arranging parts in different ways; and by "structured problem" a computational problem where the goal is to output a structured object chosen from a structured set. In Chapter 3, we will provide a more formal definition of "structured prediction" in terms of statistical dependencies between the parts of an object.

Structured problems abound in natural language processing. For example, machine translation and speech recognition both involve generating *text* as output—a structured object formed by a sequential arrangement of words. In syntactic parsing, one is given a sentence and the goal is to find a tree-structured arrangement of words or phrases, which is also a structured object. Semantic parsing aims to represent text through logical forms or other structured representations. There are several textbooks devoted to statistical approaches to these problems: for example, Charniak (1996a); Manning and Schütze (1999); Jurafsky and Martin (2000); Koehn (2010); Smith (2011).

In this thesis, we focus on a few paradigmatic and well-defined structured problems, all involving some *input object* or *observation* $x \in \mathcal{X}$, given which we want to predict a *structured output* $y \in \mathcal{Y}$. To do so, we have some *model* at our disposal, which has an associated prediction rule $h : \mathcal{X} \to \mathcal{Y}$. Typical prediction rules for structured problems are built of local score functions $\theta_p(x, y_p)$ where each p is a *place*, and y_p is an output assignment at that place; the goal is then to predict the global assignment $\widehat{y}$ that maximizes the sum of local scores:

$$\widehat{y} := \arg\max_{y \in \mathcal{Y}} \left(\sum_p \theta_p(x, y_p) \right). \tag{2.1}$$

Sometimes, instead of places, it is more convenient to talk about the "parts of the structure,"

[1] Named after the word "structure" are doctrines and disciplines such as *structuralism* or *structural linguistics*; this is *not* the sense of the word that we will be using throughout.

and to see y as a collection of parts. By *part* we mean a pair $r := (p, y_p)$ comprised of a place and an output assignment at that place. According to the above, each part $r = (p, y_p)$ will have a score $\phi_r(x) := \theta_p(x, y_p)$. The prediction rule (2.1) can as well be written as

$$\widehat{y} := \arg\max_{y \in \mathcal{Y}} \left(\sum_{r \in y} \phi_r(x) \right). \tag{2.2}$$

This chapter is organized as follows: in Section 2.1, we describe tagging and segmentation problems. Sections 2.2 and 2.3 provide a background on syntax and parsing, where the target structure is a tree. Specifically, we address phrase-structure grammars in Section 2.2, and spend Section 2.3 on dependency parsing, which is the syntactic formalism most prominently discussed in this thesis.

2.1 Tagging and Segmentation

Many tasks involving natural language have a *sequential* nature. This is not surprising: languages are sets of strings, *i.e.*, sequences of characters in an alphabet. Common needs are that of *segmenting* such sequences, or *tagging* each character with a particular label. We next discuss tagging, without losing generality—any segmentation task can be reduced to tagging by specifying labels denoting the beginning and ending of segments.

Depending on the application, there are several possible levels of granularity regarding what a "character" should mean: it can be a *letter* or *digit* (this is the case, for example, in handwriting recognition, in which we want to map from image representations of handwritten text characters to their textual representation), a *phoneme* (as in speech recognition), or a *word* (for example, in part-of-speech tagging and named entity recognition).

2.1.1 Sequence Models

We describe two statistical sequence models commonly used in NLP: *hidden Markov models* (HMMs) and *conditional random fields* (CRFs).

Hidden Markov models. HMMs (Jelinek, 1997; Rabiner, 1989) are a probabilistic analogue of finite state machines. They are are generative models that regard $\boldsymbol{X} := (X_1, \ldots, X_L)$ as a sequence of random variables describing *observations*, and $\boldsymbol{Y} := (Y_1, \ldots, Y_L)$ as a sequence of random variables describing *states*. Each state Y_i emits a symbol X_i according to an emission probability $P(X_i|Y_i)$, and the state sequence forms a Markov chain with transition probabilities $P(Y_i|Y_{i-1})$. We have, in addition, special "start" and "stop" states for modeling the beginning and ending of sequences. We denote by $\Lambda := \{\lambda_1, \ldots, \lambda_{|\Lambda|}\}$ the set of possible states. Figure 2.1 shows a graphical representation. With this setup, the joint probability distribution $P(X, Y)$ factorizes as follows:

$$P(X, Y) = \left(\prod_{i=1}^{L} P(X_i|Y_i) \right) P(Y_1|\text{start}) \left(\prod_{i=2}^{L} P(Y_i|Y_{i-1}) \right) P(\text{stop}|Y_L). \tag{2.3}$$

Given an HMM and a particular observation sequence x, an important problem is that of searching for the most probable Y given $X = x$. Defining edge scores $\theta_{i,i+1}(x, y_i, y_{i+1}) :=$

$\log P(y_{i+1}|y_i)$, and node scores

$$\theta_i(x, y_i) := \begin{cases} \log P(x_1|y_1) + \log P(y_1|\text{start}) & \text{if } i = 1 \\ \log P(x_L|y_L) + \log P(\text{stop}|y_L) & \text{if } i = L \\ \log P(x_i|y_i) & \text{otherwise,} \end{cases} \tag{2.4}$$

this boils down to solving an optimization problem of the following form:

$$\widehat{y} = \arg\max_{y_1,\ldots,y_L} \left(\sum_{i=1}^{L} \theta_i(x, y_i) + \sum_{i=1}^{L-1} \theta_{i,i+1}(x, y_i, y_{i+1}) \right). \tag{2.5}$$

Eq. 2.5 expresses a common characteristic of structured prediction problems: computing a prediction corresponds to solving a combinatorial optimization problem. Observe that this problem has the general form in Eq. 2.1, where the set of places is $\{1, \ldots, L\} \cup \{(i, i+1)\}_{i=1}^{L-1}$. Such problem is often called *maximum* a posteriori *(MAP) inference* or *MAP decoding*. Another important inference problem is that of computing the posterior marginal distributions $P\{Y_i|X = x\}$ for each $i = 1, \ldots, L$; this is called *marginal inference* or *marginal decoding*. Both problems are tractable in HMMs thanks to dynamic programming: MAP inference can be carried out in time $O(L\Lambda^2)$ with the *Viterbi algorithm* (Viterbi, 1967; Forney, 1973); for marginal inference, the *forward-backward algorithm* (Baum and Petrie, 1966) has the same runtime complexity.[2]

The model just exposed is a bigram sequence model—*i.e.*, scores depend only on context windows of size 2 (in other words, the state sequence is first-order Markov). More generally, one can think of K-gram models with $K \geq 2$. Such models can be reduced to bigram models by redefining the state space, with a corresponding increase in the computational complexity, which becomes $O(L\Lambda^K)$.

Another important problem in HMMs regards the estimation of the emission and transition probabilities; this is commonly refered as the *learning* or *training* problem. If learning is supervised, *i.e.*, if there is training data containing paired observation and state sequences, these probabilities can be easily fit by the maximum likelihood criterion, which have a closed form solution, involving counting and normalizing transition and emission events. Unsupervised learning of HMMs is more involved; the most popular procedure is the Expectation-Maximization (EM) algorithm (Baum et al., 1970; Dempster et al., 1977).[3]

Conditional Random Fields. CRFs (Lafferty et al., 2001) are *discriminative* rather than generative: they model the *conditional* distribution $P(Y|X)$ instead of the joint $P(X, Y)$. The distribution takes the form:

$$P(y|x) := \frac{1}{Z(\boldsymbol{\theta}, x)} \exp\left(\sum_{i=1}^{L} \theta_i(x, y_i) + \sum_{i=1}^{L-1} \theta_{i,i+1}(x, y_i, y_{i+1}) \right), \tag{2.6}$$

where $Z(\boldsymbol{\theta}, x)$ is a normalization factor ($Z(., x)$ is called the *partition function*). Unlike HMMs, the score functions in CRFs need not correspond to log-probabilities. Rather, for each place

[2] We assume the reader is familiar with both these algorithms; if not, a good tutorial is Rabiner (1989). We will discuss the subject of MAP and marginal inference in more depth in Chapter 4, in the context of probabilistic graphical models.

[3] Again, we refer to Rabiner (1989) for further details.

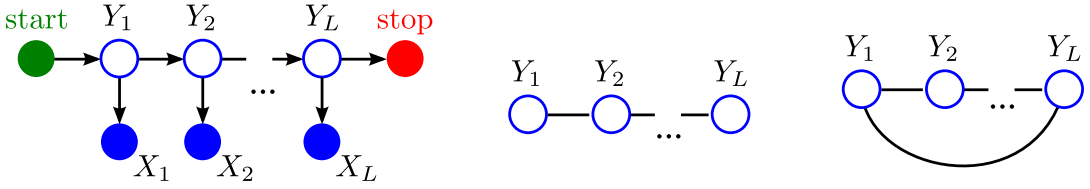


Figure 2.1: Typical sequence models in natural language tasks. Left: a hidden Markov model, drawn as a directed graphical model. Middle: a conditional random field, drawn as an undirected graphical model. Right: a skip-chain conditional random field, obtained by adding an edge modeling a long-range dependency.

p, the score function θ_p is defined as the inner product of a *weight vector* $\boldsymbol{w} \in \mathbb{R}^D$, which parameterizes the model, with a *joint feature vector* $\boldsymbol{f}_p(x, y_p) \in \mathbb{R}^D$, that is,

$$\theta_p(x, y_p) := \boldsymbol{w} \cdot \boldsymbol{f}_p(x, y_p). \tag{2.7}$$

We describe this kind of parameterization in more depth in Section 3.2. An important advantage of CRFs over HMMs is that each score function θ_p may depend *globally* on the observation sequence x (in HMMs, the score $\theta_i(x, y_i)$ depends on x only via x_i, and $\theta_{i,i+1}(x, y_i)$ does not depend on x at all.) In spite of this, the MAP inference problem for CRFs is identical to that of HMMs (both correspond to Eq. 2.5), and marginal inference can also be carried out with the same forward-backward algorithm, which can be used as a by-product to evaluate the partition function $Z(\boldsymbol{\theta}, x)$. On the other hand, the training of CRFs is more involved, and requires numerical optimization algorithms. We will discuss this issue further in Section 3.4. CRFs are represented as *undirected* graphical models; see Figure 2.1 (middle).

As in HMMs, increasing the context of the model to K-grams has a cost which is exponential in K. This highlights the fact that the tractability of inference is intimately linked to the factorization assumptions made by the model. As an example, *skip-chain CRFs* (Sutton and McCallum, 2006; Galley, 2006)—which are CRFs that contain additional pairwise scores for non-consecutive states (see Figure 2.1, right) are in general intractable to decode, even if those additional scores are spurious. This is because the dynamic programming algorithms need to exponentially increase the number of states to accommodate such pairwise scores. For these kinds of models, we must consider *approximate* inference algorithms. This subject will play a major role in this thesis.

2.1.2 Tagging Problems in Natural Language Processing

We next enumerate the natural language processing applications addressed in this thesis that involve tagging or segmentation.

Handwriting Recognition. In handwritten text recognition, we are given as input a sequence of images of alphanumeric characters and the goal is to determine their textual representation (Figure 2.2). Intuitively, we want our model to be able to capture interactions between consecutive characters: that way, even if some characters are hard to read (such as the fifth letter in Figure 2.2, which could as well be a c or a e), we may "guess" the correct

Figure 2.2: A sequence of images of alphanumeric characters, extracted from the OCR dataset of Taskar et al. (2003), publicly available at http://www.cis.upenn.edu/~taskar/ocr. The correct output sequence is the word "braces."

Input:	Meanwhile	,	overall	evidence	on	the	economy	remains	fairly	clouded	.
	RB	,	JJ	NN	IN	DT	NN	VBZ	RB	VBN	.
Output:	B-ADVP	O	B-NP	I-NP	B-PP	B-NP	I-NP	B-VP	B-ADJP	I-ADJP	O

Figure 2.3: A sentence, a predicted sequence of part-of-speech tags (both provided as input), and a segmentation into phrase chunks (example taken from the CoNLL-2000 shared task dataset, available at http://www.cnts.ua.ac.be/conll2000/chunking/). Words in the same segment are represented with the same colors.

character by looking at the other possible characters appearing in the context. We address handwriting recognition in Chapter 9.

Text Chunking. Text chunking is a sentence segmentation task. It consists of dividing a text in syntactically correlated parts of words, called *phrases*; an example is provided in Figure 2.3. Text chunking can be seen as a very shallow form of parsing (see Section 2.2), that ignores phrase-structure recursion. The typical way of transforming segmentation into a sequence labeling task is by defining B-I-O tags, which stand for *beginning of a segment* (B-x), *inside a segment* (I-x), and *outside any segment* (O). Once such a set of tags is defined, the same kind of sequence models seen above can be used to produce a segmentation. It is often useful to have part-of-speech tagging as a preprocessing step (hence providing additional part-of-speech *inputs*, as shown in Figure 2.3), so that unigram and bigram scores may take into consideration these tags. This is an example of how several structured tasks can be used in a pipeline model. We address text chunking in Chapter 10.

Named Entity Recognition. Information extraction aims to automatically extract structured information from text in natural language. An important step is that of identifying and classifying *named entities*; these are names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, *etc.* Like text chunking, this can also be framed as a segmentation task; see Figure 2.4. We address named entity recognition in Chapters 8 and 10.

2.2 Phrase-Structure Grammar

Probabilistic sequence models, which we reviewed last section, are a stochastic version of finite-state machines, the class of automata that generates regular languages. A step higher in the hierarchy of languages leads us to *context-free grammars* and *push-down automata* (Chomsky, 1956, 1965). The stochastic variant of these machines is more powerful than hidden

Input:	Only	France	and	Britain	backed	Fischler	's	proposal	.
	RB	NNP	CC	NNP	VBD	NNP	POS	NN	.
Output:	O	B-LOC	O	B-LOC	O	B-PER	O	O	.

Figure 2.4: A sentence, a predicted sequence of part-of-speech tags (both provided as input), and extracted named entities (example taken from the CoNLL-2003 shared task dataset, available at http://www.cnts.ua.ac.be/conll2003/ner/). Words in the same segment are represented with the same colors.

Markov models, and is especially amenable for modeling the syntax of natural languages.[4]

Definition 2.1 (Chomsky 1965) *A* phrase-structure grammar *or* context-free grammar *is a tuple* $\mathcal{G} = (\Lambda, \Sigma, \mathcal{P}, S)$ *where:*

1. Λ *is a finite set of* non-terminal *symbols. Elements of* Λ *are denoted by upper case letters* $(X, Y, Z, \ldots)$*. Each non-terminal symbol is a syntactic category: it represents a different type of phrase or clause in the sentence.*

2. Σ *is a finite set of* terminal *symbols (disjoint from* Γ*). This is the* alphabet *of the language defined by the grammar. Elements of* Σ *are denoted by lower case letters* $(a, b, c, \ldots)$*.*

3. $\mathcal{P}$ *is a set of* production rules, i.e., *a finite relation from* Λ *to* $(\Sigma \cup \Lambda)^*$*. We represent a production rule as* $\langle X \to \alpha \rangle$*, where* $X \in \Lambda$ *and* $\alpha \in (\Sigma \cup \Lambda)^*$*.* $\mathcal{G}$ *is said to be in* Chomsky normal form *(CNF) if any production rule in* $\mathcal{P}$ *is either of the form* $X \to YZ$ *or* $X \to a$*.*

4. S *is a* start symbol*, used to represent the whole sentence. It must be an element of* Λ*.*

By starting from S and applying some of the production rules in $\mathcal{P}$, a sentence can be derived that is composed only of terminal symbols. Figure 2.5 depicts an example of a simple grammar along with a *parse tree* for a particular sentence.

Any context-free grammar can be transformed to be in CNF without loosing any expressive power in terms of the language it generates. We henceforth assume that $\mathcal{G}$ is in CNF without loss of generality.

A fundamental characteristic of natural languages is *ambiguity*; Figure 2.6 illustrates three plausible interpretations for a sentence, yielding different parse trees. The ambiguity is caused by the several places to which the prepositional phrase could be attached. This kind of syntactic ambiguity (*PP-attachment*) is very frequent in natural language.

Weighted context-free grammars. Weighted CFGs model the *uncertainty* in interpretations of natural language sentences (Booth and Thompson, 1973; Baker, 1979). In early probabilistic models, one defines a conditional probability $\psi_\pi := P(\alpha|X)$ for each production rule $\pi \in \mathcal{P}$ of the form $\langle X \to \alpha \rangle$. These probabilities can be estimated from a corpora of sentences with annotated parse trees (a *treebank*, see Figure 2.7) by simply counting events and

[4] This does not mean that natural languages are context free. There is an immense body of work on grammar formalisms that relax the "context-free" assumption, and those formalisms have been endowed with a probabilistic framework as well. Examples are: lexical functional grammars, head-driven phrase structured grammars, combinatorial categorial grammars, tree adjoining grammars, etc. Some of these formalisms are *mildly context sensitive*, a relaxation of the "context-free" assumption which still allows polynomial parsing algorithms. There is also equivalence in expressive power among several of these formalisms.

```
S --> NP VP
NP --> Det Adj N
VP --> V NP Adv
Adj --> minimal
Adv --> here
Det --> a
N --> logic
N --> role
V --> plays
```

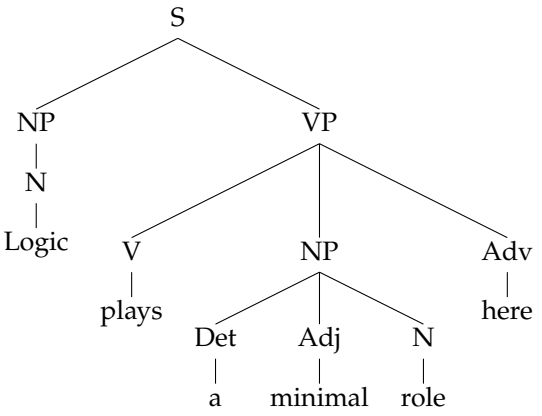


Figure 2.5: Left: a simple grammar. Right: a parse tree for the sentence *Logic plays a minimal role here* (extracted from the Penn Treebank), derived according to that grammar.

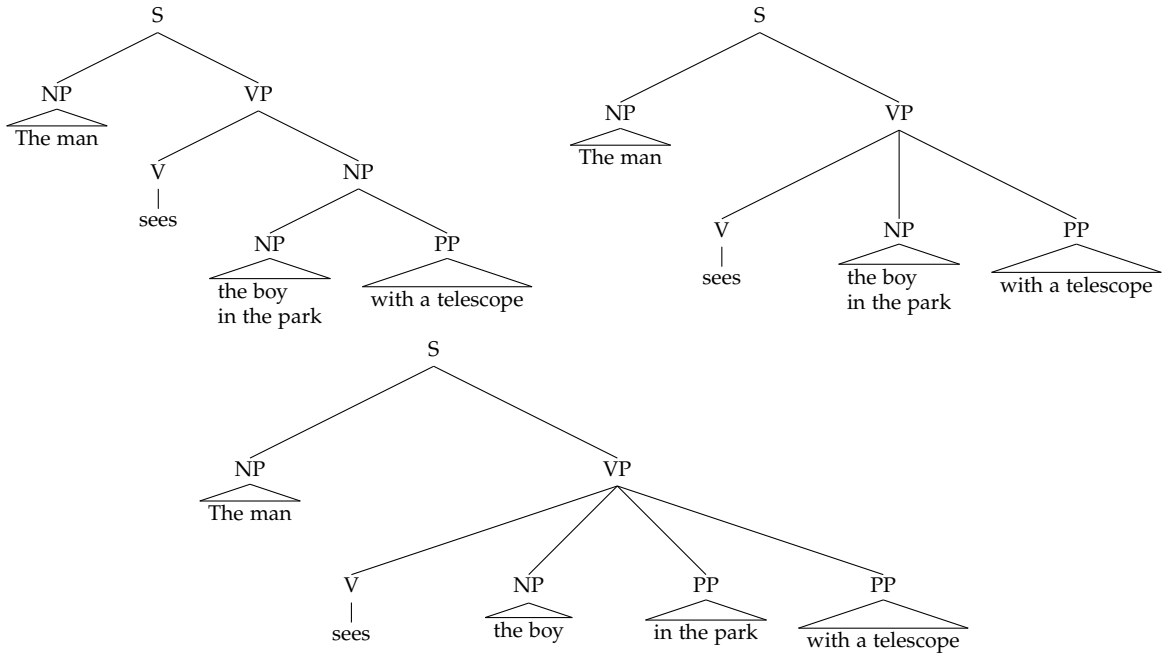


Figure 2.6: Three possible interpretations for the sentence *The man sees the boy in the park with a telescope.* Top left: the boy is in a park and he has a telescope. Top right: the boy is in a park, and the man sees him using a telescope as an instrument. Bottom: the man is in the park and he has a telescope, through which he sees a boy somewhere.

```
( (S
    (NP-SBJ (NNP BELL) (NNP INDUSTRIES) (NNP Inc.) )
    (VP (VBD increased)
      (NP (PRP$ its) (NN quarterly) )
      (PP-DIR (TO to)
        (NP (CD 10) (NNS cents) ))
      (PP-DIR (IN from)
        (NP
          (NP (CD seven) (NNS cents) )
          (NP-ADV (DT a) (NN share) ))))
    (. .) ))
```

Figure 2.7: Example of an annotated sentence in one of the most widely used treebanks, the *Penn Treebank* (Marcus et al., 1993). See http://www.cis.upenn.edu/~treebank/ for further information.

normalizing, analogously to the case of HMMs described in the previous section. In such models, the joint probability of a sentence x and parse tree y factors as

$$P(x, y) = \prod_{\pi \in \mathcal{P}} \psi_\pi^{n_\pi(x,y)}, \tag{2.8}$$

where $n_\pi(x, y)$ is the number of times production rule π is applied in the derivation. For example, for the sentence in Figure 2.5 this probability would be

$$\begin{aligned} P(x, y) \;=\; & P(\texttt{NP VP}|\texttt{S}) \times P(\texttt{N}|\texttt{NP}) \times P(\texttt{Logic}|\texttt{N}) \times P(\texttt{V NP Adv}|\texttt{VP}) \times P(\texttt{plays}|\texttt{V}) \\ & \times P(\texttt{Det Adj N}|\texttt{NP}) \times P(\texttt{a}|\texttt{Det}) \times P(\texttt{minimal}|\texttt{Adj}) \\ & \times P(\texttt{role}|\texttt{N}) \times P(\texttt{here}|\texttt{Adv}). \end{aligned} \tag{2.9}$$

When a sentence is ambiguous, the most likely parse tree can be obtained by maximizing this quantity with respect to y. This maximization can also be written in the form (2.2), by specifying a decomposition into parts. To see this, let us first introduce the following sets: a set of possible spans, *i.e.*, phrase constituents anchored on the input sentence,

$$\mathcal{R}_\text{s} := \left\{ \langle X, i, j \rangle \;\middle|\; \begin{array}{c} X \in \Lambda, \\ 1 \leq i \leq j \leq L \end{array} \right\}, \tag{2.10}$$

and a set of anchored production rules,

$$\mathcal{R}_\text{p} := \left\{ \langle X, Y, Z, i, j, k \rangle \;\middle|\; \begin{array}{c} \langle X \to YZ \rangle \in \mathcal{P}, \\ 1 \leq i \leq k < j \leq L \end{array} \right\} \bigcup \left\{ \langle X, i \rangle \;\middle|\; \begin{array}{c} \langle X \to x_i \rangle \in \mathcal{P}, \\ 1 \leq i \leq L \end{array} \right\}. \tag{2.11}$$

These are our sets of "parts," the building blocks that may or may not participate in the parse tree y. In other words, any parse tree y can be represented as a *subset* of $\mathcal{R} := \mathcal{R}_\text{s} \cup \mathcal{R}_\text{p}$, and this subset is always unique.[5] If we assign a score $\theta_r(x)$ to each part $r \in \mathcal{R}$, we can then cast the search for the most likely parse tree $\widehat{y}$ (the MAP inference problem) as the problem

[5] Note, however, that the converse is not true: there may be subsets in $\mathcal{R}$ that do not correspond to valid parse trees—namely, subsets for which productions and spans are inconsistent or cannot be combined to build up a tree. We will see later that this kind of problem can be addressed by deriving a set of *consistency equations* that work as constraints in a linear optimization problem.

of obtaining the combination of rules which maximizes the overall score:

$$\widehat{y} = \arg\max_{y \in \mathcal{Y}(x)} \left(\sum_{r \in y} \theta_r(x) \right), \tag{2.12}$$

where $\mathcal{Y}(x)$ denotes the set of valid parse trees for sentence x.

Given an anchored production rule $r \in \mathcal{R}_{\text{p}}$, denote by $\pi(r) \in \mathcal{P}$ the corresponding production rule; the correspondent scores in Eq. 2.12 for the probabilistic context-free model defined above (which is a *generative* model) are $\theta_r(x) := \log \psi_\pi$ for each $r \in \mathcal{R}_{\text{p}}$, and $\theta_r(x) := 0$ for each $r \in \mathcal{R}_{\text{s}}$. Alternatives are discriminative log-linear models (the analogue of CRFs for context-free parsing, Finkel et al. 2008), and max-margin models (Taskar et al., 2004b); both result in a problem of the form (2.12), for the same decomposition into parts.

Inference in Weighted CFGs. Let us turn to the problem of *solving* Eq. 2.12. Since the number of possible parse trees grows exponentially with the sentence length, it is computationally prohibitive to carry out this maximization directly. However, we can make use of dynamic programming to carry out this computation efficiently, through the Cocke-Kasami-Younger (CKY) algorithm (Kasami, 1966; Younger, 1967; Cocke and Schwartz, 1970), which has runtime $O(|\mathcal{P}|L^3)$, where L is the length of the sentence, and $|\mathcal{P}|$ is a grammar constant, specifying the number of production rules in the grammar.

The other important inference problem is that of computing the *posterior marginals*, $\Pr\{r \in y|x\}$, for each constituent phrase span $r \in \mathcal{R}_{\text{s}}$. This can be solved with the *inside-outside algorithm*, which is the "sum-product variant" of the CKY algorithm. We assume the reader is familiar with both the CKY and the inside-outside algorithms; if not, good references are Charniak (1996a) and Manning and Schütze (1999). Essentially, these algorithms can be seen as the counterparts of the Viterbi and the forward-backward algorithms of probabilistic sequence models to context-free parsing models.

Further Refinements: Parent Annotation, Lexicalization, Latent Variables. The construction of natural language parsers from text corpora goes back to the early 1990s, through the seminal works of Pereira and Schabes (1992); Black et al. (1993); Magerman (1995). The earliest attempts to use probabilistic context-free grammars in a supervised manner, estimating their parameters via a large treebank, go back to Charniak (1996b). The performance of these models, though, is very far from the current state of the art, and the reason is that they make too strong independence assumptions. A number of refinements has been made since then toward more accurate parsers. One important refinement is *parent annotation*, the analogue of increasing the Markov context in sequence models. This strategy splits each non-terminal symbol in the grammar (*e.g.* Z) by annotating it with all its possible parents (*e.g.* creates nodes $Z^X, Z^Y, \ldots$ every time production rules like $X \to Z\cdot$, $X \to \cdot Z$, $Y \to Z\cdot$, or $Y \to \cdot Z$ exist in the original grammar). This increases the vertical Markovian length of the model, hence weakening the independence assumptions. Parent annotation was initiated by Johnson (1998) and carried on in the unlexicalized parsers of Klein and Manning (2003) and follow-up works.

In a different line of research, a major improvement was achieved thanks to *lexicalization* (Magerman, 1995; Eisner, 1996; Charniak, 1997; Collins, 1999), which allows to exploit word

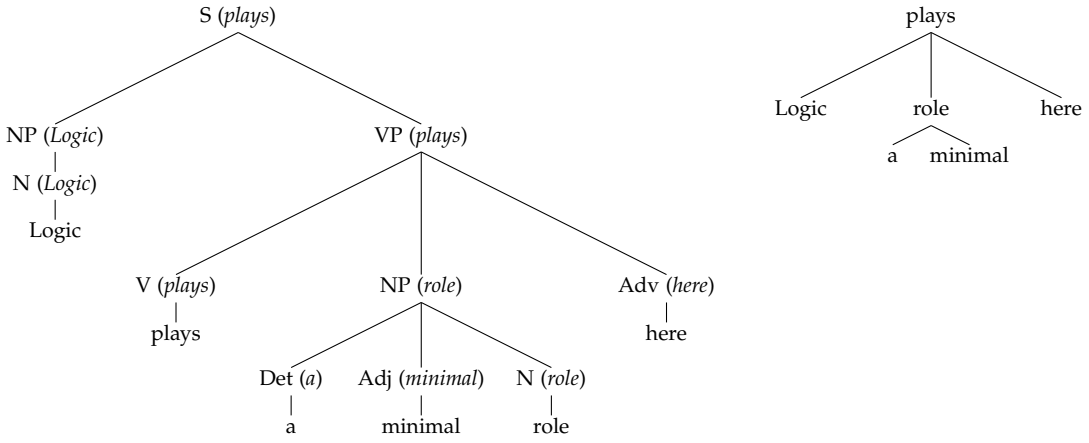


Figure 2.8: Left: a lexicalized parse tree for the sentence *Logic plays a minimal role here*. Right: a dependency tree for the same sentence, obtained by dropping all constituents and only keeping the lexical annotations.

context when predicting non-terminal node spans. This technique annotates each phrase node with the lexical item (word) which governs that phrase: this is called the *head* word of the phrase. Figure 2.8 shows an example of a lexicalized parse tree. To account for lexicalization, each non-terminal symbol in the grammar (*e.g.* Z) is split into many symbols, each annotated with a word that may govern that phrase (*e.g.* $Z^{w_1}, Z^{w_2}, \ldots$). This greatly increases the size of the grammar, but it has a significant impact on performance. We will return to the topic of lexicalization in the next section, as it is intimately related with dependency grammars.

As in sequence models, a significant progress was achieved by moving from generative to discriminative models, as the latter allow the inclusion of features that depend arbitrarily on the input sentence, with an impact on performance. Discriminative parsers have been devised by Taskar et al. (2004b) and Finkel et al. (2008).

Splitting the variables in the grammar by introducing latent variables appears as an alternative to lexicalization and parent annotation. There is a string of work concerning latent variable grammars, both for the generative and discriminative cases (Matsuzaki et al., 2005; Dreyer and Eisner, 2006; Petrov and Klein, 2007, 2008a,b). Some related work also considers coarse-to-fine parsing, which iteratively applies more and more refined models (Charniak et al., 2006; Petrov, 2009).

Finally, there is a totally different line of work which models parsers as a sequence of greedy shift-reduce decisions made by a push-down automaton (Ratnaparkhi, 1999; Henderson, 2003). When discriminative models are used, arbitrary conditioning can be done on past decisions made by the automaton, allowing to include features that are difficult to handle by the other parsers. This comes at the price of greediness in the decisions taken, which implies suboptimality in maximizing the desired objective function.

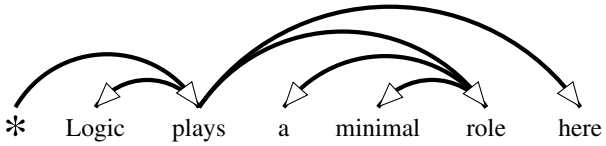


Figure 2.9: A dependency tree for the sentence *Logic plays a minimal role here*, drawn with the `drawdeptree.pl` visualization tool developed by Terry Koo. Note the additional dummy root symbol ($*$) which is included for convenience.

2.3 Dependency Parsing

We now move to a diferent syntactic formalism, *dependency grammar*. Consider again the sentence *Logic plays a minimal role here*, along with the lexicalized parse tree displayed in Figure 2.8 (left). If we drop the phrase constituents and keep only the head words, the parse tree would become as depicted in Figure 2.8 (right). This representation is called a *dependency tree*; it can be alternatively represented as shown in Figure 2.9.

Dependency trees retain the lexical relationships involved in lexicalized phrase-based parse trees. However, they drop phrasal constituents, which render non-terminal nodes unnecessary. This has computational advantages (the grammar constant involved in the complexity of the parsing algorithms becomes much smaller) as well as design advantages (no grammar is necessary, and treebank annotations are much simpler, since no internal constituents need to be annotated). It also shifts the focus from internal syntactic structures and generative grammars (Chomsky, 1965) to lexical and transformational grammars (Tesnière, 1959; Hudson, 1984; Mel'čuk, 1988; Covington, 1990). Dependency grammars are also related with link grammars (Lafferty et al., 1992) and head-automata (Alshawi, 1996). Lately, this formalism has been used as an alternative to phrase-based parsing for a variety of tasks, ranging from machine translation (Ding and Palmer, 2005) to relation extraction (Culotta and Sorensen, 2004) and question answering (Wang et al., 2007), to name just a few.

We proceed to a formal definition of a dependency tree:[6]

Definition 2.2 (Dependency tree.) *Let $x = (x_0, x_1, \ldots, x_L)$ be a string, where $x_0 := *$ is a special symbol. We denote the tokens in x by integers $0, 1, \ldots, L$. A* dependency tree *for x is a directed tree y rooted at 0 that spans $\{0, 1, \ldots, L\}$.*

We represent y by its set of arcs, called *dependency arcs* or *dependency links*. In an arc (h, m), the source token h is called the *head* and the target token m is called the *modifier*. The *span* of (h, m) is the set of tokens between h and m, $\text{span}(h, m) := \{k \in \mathbb{N} \mid \min\{h, m\} \leq k \leq \max\{h, m\}\}$. The tree structure induces a partial order: we write $a \preceq_y d$ if there is a directed path in y from token a to token d, in which case a is called an *ancestor* of d and d is a *descendant* of a.

Given our early depiction of a dependency tree as the outcome of a lexicalized context-free phrase-structured tree, one might wonder if any dependency tree can be constructed this way. The answer turns out to be negative: the ones that can be constructed like this are the *single-rooted*, *projective* dependency trees, in which there is only one arc whose head is 0, and

[6] Some authors provide an alternative definition, which corresponds to what we call a *projective* dependency tree (see Definition 2.4 in the sequel).

dependency arcs are nested (cannot cross each other); this is made formal in Definition 2.4 and Propositions 2.1–2.2. We start by the definition of a projective *arc*,[7] and then move on to projective *trees*.

Definition 2.3 (Projective arc.) *Given a dependency tree y, an arc $(h, m) \in y$ is called* projective *if all tokens in its span descend from h. It is called* non-projective *otherwise.*

Definition 2.4 (Projective dependency tree.) *A dependency tree y is* projective *if all its arcs are projective.*

We say that arcs (h, m) and (h', m') cross each other if they are not nested ($\text{span}(h, m) \nsubseteq \text{span}(h', m')$ and $\text{span}(h', m') \nsubseteq \text{span}(h, m)$) but their spans have non-empty intersection ($\text{span}(h, m) \cap \text{span}(h', m') \neq \varnothing$). A dependency tree is said to be *single-rooted* if there is a unique arc departing from 0.

Proposition 2.1 *A dependency tree is projective if and only if no arcs cross each other.*

Proof. See Appendix A.1. ■

Proposition 2.2 *A dependency tree can be constructed from a lexicalized context-free phrase-structured tree if and only if it is projective and single-rooted.*

Proof. See Gaifman (1965). ■

One of the reasons why it is attractive to consider the general class of dependency trees as defined in Definition 2.2 is that in many languages (*e.g.*, those which are free-order) the projectivity assumption is too stringent. Even in languages with fixed word order (such as English) there are syntactic phenomena which are awkward to characterize using projective trees arising from the context-free assumption, such as wh-movement and general filler-gap constructions. Such phenomena are commonly characterized with additional linguistic artifacts (*e.g.*, inserting a trace symbol in the gap position, distinguishing between deep and surface structure, *etc.*).[8] The allowance of non-projective arcs dispenses with these artifacts. An example is the sentence (extracted from the Penn Treebank)

> We learned a lesson in 1987 about volatility.

There, the prepositional phrase *in 1987* should be attached to the verb phrase headed by *learned* (since this is *when* we learned the lesson), but the other prepositional phrase *about volatility* should be attached to the noun phrase headed by *lesson* (since the *lesson* was about volatility). To explain such phenomena, context-free grammars need to use additional machinery which allows words to be scrambled (in this case, via a movement transformation and the consequent insertion of a trace). In the dependency-based formalism, we can get rid of all those artifacts by allowing *non-projective* parse trees; an example for the sentence above is shown in Figure 2.10.

[7] Other definitions of "projective arc" have been proposed in the literature; this one is due to Kahane et al. (1998).

[8] The best way of analyzing wh-movement and similar phenomena is a subject of intense debate in syntax theories (Chomsky, 1993, 1995; Sag et al., 1999; Bresnan, 2001).

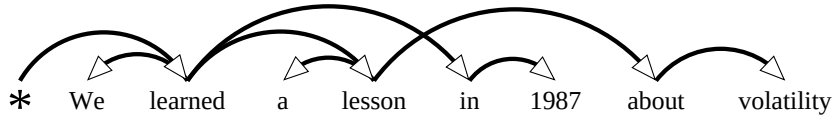


Figure 2.10: A parse tree which is not projective. The arc *lesson* → *about* is non-projective.

In the remaining part of this section, we discuss models and algorithms for projective and non-projective dependency parsing. Before doing so, we shall mention that dependency trees are often presented with label annotations on their arcs, which provide more detailed syntactic information. For example, the arc *enjoys* → *She* could be labeled as `SUBJ` to denote that the modifier *She* has a subject function, and the arc *enjoys* → *school* could be labeled as `OBJ` to denote that the modifier *school* has an object function. In our thesis, we consider dependency parsing as the problem of finding an *unlabeled* tree, conveying the "bare bones" of the syntactic structure.[9]

Arc-Factored Models. We start by considering a simple kind of models which are called *arc-factored*. These models assign a score $\theta_{h,m}(x)$ to each possible arc (h, m) with $h \neq m$ and $m \neq 0$. In a probabilistic model, the conditional probability of a parse tree y is given by:

$$P(y|x) \propto \prod_{(h,m)\in y} \exp(\theta_{h,m}(x)). \tag{2.13}$$

As usual, from the point of view of the parsing algorithm, it does not matter whether the scores come from a generative or discriminative model, or which features were used to compute the scores. The three important inference tasks are:

1. Obtain the most likely dependency tree (the MAP inference problem),

$$\widehat{y} = \arg\max_{y} \sum_{(h,m)\in y} \theta_{h,m}(x). \tag{2.14}$$

   Note that, once again, this takes the form in Eq. 2.1.

2. Compute the partition function (for a log-linear model),

$$Z(\boldsymbol{\theta}, x) = \sum_{y} \prod_{(h,m)\in y} \exp(\theta_{h,m}(x)). \tag{2.15}$$

3. Compute the posterior marginals for all the possible arcs (for a log-linear model),

$$\Pr_{\boldsymbol{\theta}}\{(h, m) \in y|x\}, \qquad \text{for every } (h, m). \tag{2.16}$$

As we will see, all these three problems can be solved efficiently if the model is arc-factored, both for the projective and non-projective case.

[9]Several possibilities exist to extend a bare bone parser to output dependency labels: one is to use a joint model for inferring the dependencies and labels altogether (McDonald and Pereira, 2006); another is to follow a two-stage approach that first gets the dependencies, and then the labels (McDonald et al., 2006).

Unfortunately, the arc-factored assumption is often too severe, as it fails to capture "interactions" between arcs. These interactions are usually important: for example, two tokens m and s, when considered individually, may both be strong candidates for dependents of a verb h, for example because both look like plausible subjects for that verb. In accordance, an arc-factored model could well predict a dependency parse tree containing both (h, m) and (h, s). However, it is usually unlikely that a verb accepts more than one subject, hence at least one of those arcs would be an incorrect prediction. Consider now a more powerful model that includes a score (say $\theta_{h,m,s}$) for the *simultaneous* inclusion of (h, m) and (h, s). Such a model would disfavor the presence of both arcs and could well pick only one of them, eventually achieving a correct prediction where h gets only one subject.

Pairs of arcs as above, of the form (h, m) and (h, s), are usually called *siblings* (because they render tokens m and s as siblings in the tree). Another important kind of arc pairs is those of the form (g, h) and (h, m), which are commonly called *grandparents* (since they render g grandparent of m). We next discuss how inference under these more powerful models can be done, both for the projective and non-projective cases.

Projective Dependency Parsing. Given the strict relationship between projective dependency trees and lexicalized phrase-structured trees, it is not surprising that the *projective* case can be addressed by what are essentially variations of the CKY algorithm, albeit with a lexical flavour.[10]

For arc-factored models, the three tasks listed above can be solved in $O(L^3)$ time with Eisner's dynamic programming algorithm (Eisner, 1996, 2000), a clever adaptation of the CKY algorithm that makes use of incomplete spans. A max-product variant of the algoritm is what is necessary for the first task (MAP inference), while the last two tasks (computing the partition function and the marginals) can be solved with the sum-product variant.

Eisner's algorithm can be extended for models that include scores for *consecutive sibling arcs* on the same side of the head,[11] while keeping the $O(L^3)$ runtime (Eisner, 1996). Models that include such scores (in addition to scores for individual arcs) are said to use *second-order horizontal Markov context*.

More recently, Carreras (2007) addressed *second-order vertical Markovization* by extending Eisner's algorithm to handle scores for a restricted kind of *grandparent arcs*, albeit increasing the asymptotic runtime to $O(L^4)$. Even more recently, Koo and Collins (2010) introduced *third-order* dependency parsers that use grand-siblings and tri-siblings, along with an algorithm which still runs in $O(L^4)$ time.

Like in phrase-structure parsing, there is a totally different line of work which models parsers as a sequence of greedy shift-reduce decisions (Nivre et al., 2006; Huang and Sagae, 2010)—these are called *transition-based parsers*. These parsers achieve a very appealing speed-accuracy trade-off: empirically, they have been reported to exhibit expected linear runtime,

[10] Note, however, that making these variations *efficient* is not an obvious step and was only accomplished by Eisner (1996). The process of lexicalizing a context-free grammar consists in decorating the symbols in each production rule with lexical information: for example $X \to YZ$ is multiplied into several rules of the form $X[h] \to Y[h]Z[m]$ and $X[h] \to Y[m]Z[h]$, where h and m are lexical items, *i.e.*, terminal symbols. When parsing a string, there are two additional indices involved in each of these rules, hence a naïve adaptation of the CKY algorithm would render $O(L^5)$ runtime. The trick is to use *incomplete* spans (Eisner and Satta, 1999; Eisner, 2000), which reduce runtime to $O(L^4)$ in general bilexical phrase-structured grammars, and to $O(L^3)$ bilexical *split* grammars.

[11] These are pairs of sibling arcs, (h, m) and (h, s) such that no arc (h, r) exists with r between m and s.

and are only slightly less accurate than the state of the art.

Non-Projective Dependency Parsing. What about the case of *non-projective* dependency parsing—where the trees are not constrained to be projective? For arc-factored models, efficient algorithms are also available for solving the three problems in Eqs. 2.14–2.16. Interestingly, these algorithms are not related with dynamic programming.

The first problem, *i.e.*, that of computing the most likely dependency tree (MAP inference), corresponds to finding a maximum weighted directed spanning tree (also called a maximum weighted arborescence) in a directed graph. This fact has first been noted by McDonald et al. (2005b). This problem is well known in combinatorics and can be solved in $O(L^3)$ using Chu-Liu-Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967).[12]

The second and third problems—computing the partition function and the marginals—can be solved by invoking another important result in combinatorics, a weighted version of Kirchhoff's matrix-tree theorem (Kirchhoff, 1847; Tutte, 1984). This fact has been noticed independently by Smith and Smith (2007); Koo et al. (2007); McDonald and Satta (2007). The cost is that of computing a determinant and inverting a matrix, which can be done in time $O(L^3)$.[13] The procedure is as follows. We first consider the directed weighted graph formed by including all the possible dependency links (h, m) (including the ones departing from the start symbol, for which $h = 0$ by convention), along with weights $\psi_{h,m} := \exp(\theta_{h,m}(x))$. We then compute its $(L+1)$-by-$(L+1)$ Laplacian matrix $\mathbf{L}$ whose entries are:

$$L_{hm} := \begin{cases} \sum_{h'=0}^{L} \psi_{h',m} & \text{if } h = m, \\ -\psi_{h,m} & \text{otherwise.} \end{cases} \tag{2.17}$$

Denote by $\hat{\mathbf{L}}$ the $(0,0)$-minor of $\mathbf{L}$, *i.e.*, the matrix obtained from $\mathbf{L}$ by removing the first row and column. Consider the determinant $\det \hat{\mathbf{L}}$ and the inverse matrix $\hat{\mathbf{L}}^{-1}$. Then:

- the partition function is given by

$$Z(\boldsymbol{\theta}, x) = \det \hat{\mathbf{L}}; \tag{2.18}$$

- the posterior marginals are given by

$$\Pr_{\boldsymbol{\theta}}\{(h, m) \in y | x\} = \begin{cases} \psi_{h,m} \times ([\hat{\mathbf{L}}^{-1}]_{mm} - [\hat{\mathbf{L}}^{-1}]_{mh}) & \text{if } h \neq 0 \\ \psi_{0,m} \times [\hat{\mathbf{L}}^{-1}]_{mm} & \text{otherwise.} \end{cases} \tag{2.19}$$

Unfortunately, extensions beyond the arc-factored model have a much greater impact on runtime in the non-projective case than they have in the projective case: horizontal and vertical Markovization renders MAP inference NP-hard (McDonald and Pereira, 2006; McDonald and Satta, 2007). Yet, approximate algorithms have been proposed to handle "second-order models" that seem to work well: a projective parser followed by hill-climbing (McDonald et al., 2006), loopy belief propagation (Smith and Eisner, 2008), and a dual decomposition

[12] Asymptotically faster algorithms exist, *e.g.*, Tarjan (1977) and Gabow et al. (1986) propose algorithms that solve the same problem in $O(L^2)$ and are even faster when only some candidate arcs are considered.

[13] Again, there are faster asymptotic algorithms for carrying out this computation; e.g., an adaptation of Coppersmith-Winograd algorithm would take $O(L^{2.376})$ (Coppersmith and Winograd, 1990). However, it is likely that the constants hidden in the O notation are too large to be useful in practice.

	$\mathcal{Y}$	Parts
Sequence labeling	L-length strings (Λ^L)	Unigram and bigram assignments
Phrase structure parsing	Phrase structure trees	Anchored phrase spans and production rules (maybe lexicalized and/or parent annotated)
Dependency parsing	Dependency trees	Candidate arcs (maybe siblings, grandparents, *etc.*)

Table 2.1: Summary of the sets of outputs and parts for the tasks discussed in this chapter.

method (Koo et al., 2010). In this thesis, we present an integer programming formulation with a polynomial number of constraints, along with an efficient algorithm for solving the linear programming relaxation (Chapter 7). Transition-based parsers have also been adapted to handle non-projective parsing, albeit with a slower runtime than transition-based projective parsers (Nivre, 2009).

2.4 Conclusion

In this chapter, we have described several structured problems in NLP: handwriting recognition, text chunking, named entity recognition, phrase-structure parsing, and dependency parsing. We have focused on statistical models to tackle such problems, such as hidden Markov models, conditional random fields, probabilistic context-free grammars, and other statistical parsers. We have discussed some of the existing work on this area.

There are a number of common ingredients in all these problems:

- A *structured* output set $\mathcal{Y}$ from which we want to pick a prediction $\widehat{y}$;
- A *set of parts* $\mathcal{R}$ which are the building blocks of the elements of $\mathcal{Y}$;
- Some sort of *constraints* that tell which combinations of parts produce valid outputs;
- A statistical model that defines a *global* score function, which decomposes into *local* scores defined on the parts.

What the parts are and how they are entangled is what characterizes each of the problems. Table 2.1 summarizes what these are in the several tasks discussed in this chapter. The next chapters (Chapters 3 and 4) will make things more formal, by covering the inference and learning problems in structured prediction.

Chapter 3

Structured Prediction and Learning Algorithms

The models and algorithms described in this thesis are applicable to general *structured prediction* problems. These are problems with a strong interdependence among the output variables, often with sequential, graphical, or combinatorial structure. Problems of this kind arise in natural language processing, computer vision, robotics, and computational biology. The latest years have witnessed a considerable progress towards a unified formalism for tackling this kind of problems, most notably due to the seminal works of Lafferty et al. (2001); Collins (2002a); Altun et al. (2003); Taskar et al. (2003); Tsochantaridis et al. (2004).

In this chapter, we synthesize key foundational prior work in structured prediction, providing a brief overview of the tools and concepts that will be necessary for presenting the main contributions of this thesis.[1] This chapter is organized as follows. Section 3.1 presents the basics of *supervised learning*. Section 3.2 describes *linear models*, which is the kind of models that we use throughout, along with a brief review of discriminative methods. In Section 3.3, we define *structured prediction* rather informally, as a distinctive set of problems that deserve special attention. The remaining pair of sections is devoted to the problem of training structured predictors. We discuss the underlying optimization problems (Section 3.4) and focus on online learning algorithms (Section 3.5).

3.1 Supervised Learning

Humans act and make decisions on a daily basis. Part of the inference procedure that guides these decisions is about observing the state x of some object (the *input*) and using our current understanding of the world (our *model*, learned from past experience) to predict some hidden property y (the *output*). Hopefully, we will learn to predict better and better as we get to see more data. This intuition carries out to machines.

We are broadly interested in learning a map $h : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$ is an input set and $\mathcal{Y}$ is an output set. Given $x \in \mathcal{X}$, we call $\widehat{y} := h(x)$ a *prediction*. In the sequel, $\mathcal{Y}$ will always be discrete; we refer to its elements as *labels* or *classes*. Simple examples are binary classification, where $\mathcal{Y} := \{0, 1\}$, and multi-class classification, where $\mathcal{Y}$ is finite and $|\mathcal{Y}| \geq 2$. In structured prediction (to be introduced in Section 3.3), $\mathcal{Y}$ can be very (possibly infinitely) large. A map

[1] Additional background on this subject can be found in Bakır et al. (2007) and the references therein.

h as just defined is called a *hypothesis* or a *classifier*. We want h to yield accurate predictions in a sense to be made precise below.

In the supervised learning paradigm, we are given access to *training data* $\mathcal{D}$ consisting of input-output pairs (called *examples* or *instances*):

$$\mathcal{D} := \Big((x^1, y^1), \ldots, (x^N, y^N)\Big) \in (\mathcal{X} \times \mathcal{Y})^N. \tag{3.1}$$

We follow the common assumption that there is an underlying joint probability distribution $P(X, Y)$ that governs the generation of data, and that $\mathcal{D}$ is a sample drawn i.i.d. from this distribution. The machine only gets to see the sample; the true distribution $P(X, Y)$ is unknown. The goal of supervised machine learning is to "learn" h from the data in $\mathcal{D}$. To this end, one considers a *hypothesis class* $\mathcal{H} \subseteq \{h : \mathcal{X} \to \mathcal{Y}\}$.[2] The precise choice of $\mathcal{H}$ is usually driven by our prior knowledge about the problem at hand, as well as eventual computational limitations. This involves one or more of the following decisions: choosing the represention of inputs and outputs, designing features or a kernel function, deciding on the number of hidden layers in a neural network, choosing the number of latent variables in a probabilistic model, making factorization assumptions, *etc*. We are then led to the following problem:

- Given the hypothesis class $\mathcal{H}$, how to pick $h \in \mathcal{H}$?

Obviously, one wants h to generalize well to unseen data: given a new instance $(x, y) \sim P(X, Y)$ from which only x is observed (a *test* example), the prediction $\widehat{y} = h(x)$ should be "close" (in expectation) to the true output y. This can be formalized as follows: let $\rho : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a non-negative *cost function*, which vanishes on the diagonal (*i.e.*, such that $\rho(y, y) = 0$ for every $y \in \mathcal{Y}$). The value $\rho(\widehat{y}, y)$ represents the cost incurred when the true output is y and the machine predicts $\widehat{y}$. An example is the 0/1 cost:

$$\rho_{0/1}(\widehat{y}, y) := \begin{cases} 1 & \text{if } \widehat{y} \neq y \\ 0 & \text{otherwise,} \end{cases} \tag{3.2}$$

which can be written compactly as $\rho_{0/1}(\widehat{y}, y) = [\![\widehat{y} \neq y]\!]$, where $[\![\pi]\!]$ is 1 if predicate π is true, and 0 otherwise. One wants a classifier h that incurs as little cost as possible on unseen data. This is captured through the following notion:

Definition 3.1 (Expected and empirical risk.) *Let* $\mathbb{E}_{X,Y}$ *denote the expectation operator under* $P(X, Y)$. *The* expected risk *of a classifier* h *(with respect to a cost function* ρ*) is the quantity*

$$\text{risk}(h; \rho) := \mathbb{E}_{X,Y}[\rho(h(X), Y)]. \tag{3.3}$$

Given training data $\mathcal{D}$, *the* empirical risk *of* h *is*

$$\text{risk}_{\mathcal{D}}(h; \rho) := \mathbb{E}_{\mathcal{D}}[\rho(h(X), Y)] = \frac{1}{N} \sum_{n=1}^{N} \rho(h(x^n), y^n). \tag{3.4}$$

[2] There is a variety of learning algorithms which are characterized by the different kinds of hypothesis classes they operate on; examples are nearest neighbor classifiers, decision trees, multilayer neural networks, or linear models. In this thesis we will focus on the last—cf. Section 3.2. There are numerous textbooks that provide a background on several of the learning formalisms mentioned above, *e.g.*, Mitchell (1997); Duda et al. (2001); Schölkopf and Smola (2002); Bishop (2006).

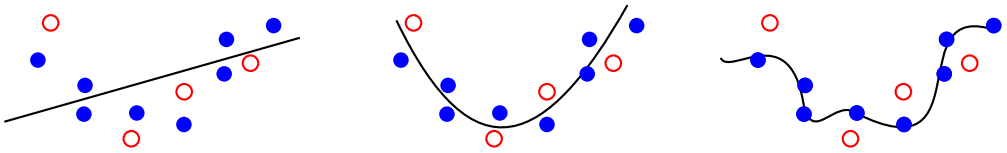


Figure 3.1: A regression problem with N training examples, drawn as blue circles. Test examples are drawn as white circles. Our hypothesis classes are polynomials up to some degree D. As we increase the complexity of the hypothesis class (*i.e.*, as we increase D), we obtain classifiers that fit the training data better, thus reducing the empirical risk—in particular, with polynomials up to degree $D = N - 1$ zero empirical risk can always be achieved. However, this increases the danger of *overfitting* the training data, leading to poor generalization (right). This is in contrast with a too small D, which would lead to *underfitting* (left). A general machine learning problem is to find an equilibrium between these two extremes (middle).

When the cost function is clear from the context, we omit the dependency of the expected and empirical risk on ρ, writing these quantities simply as $\text{risk}(h)$ and $\text{risk}_{\mathcal{D}}(h)$. An optimal decision would be to pick a classifier $h^* \in \mathcal{H}$ which minimizes the expected risk. However, since the true distribution $P(X, Y)$ is unknown, this risk cannot be assessed. The best we have is the *empirical* risk; yet, optimizing this risk directly is usually problematic for a couple of reasons:

1. For many cost functions (*e.g.*, the $0/1$ cost) empirical risk minimization leads to a difficult combinatorial problem, which becomes intractable even for "simple" hypothesis classes (such as linear models).

2. If the training set $\mathcal{D}$ is too small and/or the hypothesis class $\mathcal{H}$ has too large a "complexity," then the classifier minimizing the empirical risk may *overfit* the data. This means that the gap between the empirical and expected risks can be too large, so that the former is no longer a good indicator of the quality of the classifier. Typical bounds on this gap grow as fast as $O(\sqrt{D/N})$, where D is a measure of complexity[3] of $\mathcal{H}$ and $N = |\mathcal{D}|$. Figure 3.1 illustrates this point.

The first problem is commonly sidestepped by replacing $\text{risk}_{\mathcal{D}}(h)$ by a surrogate *loss function* more amenable to optimization. The second problem is avoided by introducing a *regularizer* that penalizes the "complexity" of a hypothesis. This will be made clear in the next section, where we discuss linear models.

3.2 Linear Models

When one is faced with the choice of a model for supervised learning, the class of *linear models*[4] appears at the front line. Linear models have several virtues: simplicity, ease of

[3]There are many ways we can assess the complexity of a hypothesis class from the point of view of statistical learning theory. Examples are the Vapnik-Chervonenkis (VC) dimension (Vapnik, 1998) and the Rademacher complexity (Anthony and Bartlett, 2009). See Schölkopf and Smola (2002) for further details on VC-theory and statistical learning.

[4]These models are also commonly referred to as *generalized linear models* (Hastie et al., 2001). The term *generalized* comes from the fact that, in regression, they generalize standard linear regression models through the use of a link function.

interpretation, and the ability of being rendered non-linear through the use of a kernel function. We consider a *joint feature map* $\boldsymbol{f} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ that represents input-output pairs as vectors of *features* in a D-dimensional Euclidean space. The actual choice of features is usually driven by practical considerations and it is where the practicioner should exploit her knowledge about the problem at hand. Features can take multiple forms:

- *integer values* associated with counts of events observed in $(x, y) \in \mathcal{X} \times \mathcal{Y}$;
- *binary-valued* indicators that some predicate π holds in (x, y);
- *categorical* features, usually converted to an array of binary values (as many as the number of categories);
- *real-valued* statistics, measurements, or confidence values (such as log-probability estimates) output by another model run at a previous stage.

In linear models, the hypothesis class $\mathcal{H} := \{h_{\boldsymbol{w}} : \mathcal{X} \to \mathcal{Y} \mid \boldsymbol{w} \in \mathbb{R}^D\}$ is comprised of linear discriminant functions of the form:[5]

$$h_{\boldsymbol{w}}(x) := \arg\max_{y \in \mathcal{Y}} \ \boldsymbol{w} \cdot \boldsymbol{f}(x, y), \tag{3.5}$$

where $\boldsymbol{w} \in \mathbb{R}^D$ is a *parameter vector* and $\cdot$ denotes the standard inner product in $\mathbb{R}^D$, $\boldsymbol{w} \cdot \boldsymbol{f}(x, y) := \sum_{d=1}^{D} w_d f_d(x, y)$. Intuitively, each parameter w_d is a *weight* for feature $f_d(x, y)$ (and therefore $\boldsymbol{w}$ is also commonly called a *weight vector*).[6]

The problem in Eq. 3.5 is called *inference* or *decoding*. In binary and multi-class classification with few classes, this problem is easily solved by enumerating all the scores $\theta(y) := \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$ for each $y \in \mathcal{Y}$, and picking the class with the largest score. The case is different in structured prediction, however, due to the *hardness-of-enumeration* assumption, to be described in Section 3.3.

We now turn to the problem of *learning* the model parameters $\boldsymbol{w}$. Following up on the last section, this can be formulated as an optimization problem of the form[7]

$$\begin{array}{ll} \text{minimize} & \Omega(\boldsymbol{w}) + \frac{1}{N} \sum_{n=1}^{N} L(\boldsymbol{w}; x^n, y^n) \\ \text{w.r.t.} & \boldsymbol{w} \in \mathbb{R}^D, \end{array} \tag{3.6}$$

where $\Omega : \mathbb{R}^D \to \mathbb{R}$ is a *regularizer* and $L : \mathbb{R}^D \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a *loss function*. Problem 3.6 is called *learning* or *training*. One of the nice aspects of linear models is that it is simple to formulate learning as a *convex* optimization problem, once we make a suitable choice of Ω and L (namely, both need to be convex functions). This brings two great benefits: first, we can use the machinery of convex optimization algorithms to solve Eq. 3.6 efficiently. Second, we do not need to worry about local optima (since every local minimum will be a global minimum). This is an important advantage over more complex models, such as multilayer neural networks, or some probabilistic models with latent variables.

[5] This notation is sloppy because there may be multiple arguments maximizing the objective. We assume that ties are broken arbitrarily using some predefined fixed rule.

[6] These models can be "kernelized" by letting $\mathcal{H}$ be a reproducing kernel Hilbert space with kernel $K : (\mathcal{X} \times \mathcal{Y})^2 \to \mathbb{R}$. We will see instances in Chapter 9.

[7] This is commonly referred in the literature as *regularized empirical risk minimization*. We note however that the term "empirical risk" enclosed in this expression does not usually refer to the original cost function (*e.g.*, the 0/1 cost) but to a surrogate function which is tractable to optimize.

3.2.1 Regularization

A natural choice for the regularizer Ω is a norm, possibly raised to a power $q \geq 1$: this function is always guaranteed to be convex (see, *e.g.*, Boyd and Vandenberghe 2004, pp. 73, 86), and penalizes the magnitude of the weight vector, which can be interpreted as a penalization of the complexity of the classifier. The most typical choices are:[8]

- L_2-regularization: $\Omega_\lambda^{L_2}(\boldsymbol{w}) := \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$;
- L_1-regularization: $\Omega_\tau^{L_1}(\boldsymbol{w}) := \tau\|\boldsymbol{w}\|_1$,

where λ and τ are non-negative scalars—called *regularization coefficients*—that control the intensity of regularization. These coefficients are typically tuned with held-out data or cross-validation. Both regularizers have first been studied in the context of ill-posed problems. The use of L_2-regularization goes back to Tikhonov (1943) and, in the context of regression, is commonly referred as *ridge regularization*. In NLP, it has been used in classification tasks by Chen and Rosenfeld (1999, 2000), among others, for regularizing maximum entropy models. While L_2-regularization is often robust to the inclusion of irrelevant features, such features typically get nonzero weights (leading to a dense weight vector $\boldsymbol{w}$ as a solution of Eq. 3.6). L_1-regularization was popularized by Tibshirani (1996) in the context of sparse regression, being called *Lasso regularization*.[9] It has been used by Kazama and Tsujii (2003) and Goodman (2004) as an alternative to L_2-regularization in maximum entropy models. Unlike the L_2 case, this usually leads to a sparse weight vector $\boldsymbol{w}$ as a solution of Eq. 3.6, where some weights become exactly zero—consequently, the corresponding features can be discarded from the model. There are a couple of reasons why this can be desirable:

- It yields a more *compact* model, with less memory requirements;
- It allows pinpointing the relevant features, yielding model *interpretability*.

An empirical comparison between these two regularization strategies has been presented by Gao et al. (2007). Other related regularizers have been proposed, such as *elastic nets* (Zou and Hastie, 2005), which are an interpolation of L_1 and L_2. These have been used by Lavergne et al. (2010) to regularize conditional random fields. In Chapter 10, we will introduce other regularizers, that are able to promote *structured sparsity*.

3.2.2 Loss Functions for Discriminative Learning

Let us now discuss the loss function L in Eq. 3.6. In linear models, it is assumed that $L(\boldsymbol{w}; x, y)$ depends on $\boldsymbol{w}$ only via the inner product $\boldsymbol{w} \cdot \boldsymbol{f}(x, y)$.

Most learning methods are either *generative* or *discriminative*. The ones of the first kind design the loss function through a "generative story" about how data is generated; for example, they may model the joint probability $P(X, Y)$ and use that to define the loss function

[8] Both regularizers have a Bayesian interpretation. In probabilistic models, the loss term L usually corresponds to the negative log-likelihood of a (joint or conditional) probability distribution parameterized by the weight vector $\boldsymbol{w}$. Bayesians would rather see $\boldsymbol{w}$ as another random variable instead of a "parameter vector," and do so by defining a prior belief $P(\boldsymbol{w})$ on the weights. In that case, the regularizer Ω can be interpreted as the negative logarithm of this prior distribution, and the solution of Eq. 3.6 as the maximum *a posteriori* estimate of $\boldsymbol{w}$. Under this lens, $\Omega_\lambda^{L_2}$ corresponds to using independent zero mean Gaussian priors for each feature weight $w_d \sim \mathcal{N}(0, \lambda^{-1})$, and $\Omega_\tau^{L_1}$ corresponds to zero-mean Laplacian priors, $p(w_d) \propto \exp(\tau|w_d|)$.

[9] For *least absolute shrinkage and selection operator*.

(this is the procedure that underlies, *e.g.*, the naïve Bayes method). In contrast, discriminative methods shift the focus from data generation to output prediction. Comparisons between generative and discriminative methods for different problems and regimes have appeared several times in the literature (Ng and Jordan, 2002; Liang and Jordan, 2008), and fusions between the two kinds of methods have also been proposed (Jaakkola and Haussler, 1998; Jebara et al., 2004; Martins et al., 2009a). Since the focus of this thesis is on discriminative methods, we describe only those.

Logistic Regression. A *log-linear model* is a probabilistic model that takes the form:

$$P_{\boldsymbol{w}}(y|x) := \frac{1}{Z(\boldsymbol{w}, x)} \exp(\boldsymbol{w} \cdot \boldsymbol{f}(x, y)), \tag{3.7}$$

where $Z(\boldsymbol{w}, x) := \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y')\right)$ is a normalization factor. This model can be fit to the data by maximizing the conditional log-likelihood. This procedure leads to the following *logistic loss* function, which is plugged in Eq. 3.6:

$$L_{\text{LR}}(\boldsymbol{w}; x, y) := -\boldsymbol{w} \cdot \boldsymbol{f}(x, y) + \log \sum_{y' \in \mathcal{Y}} \exp\left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y')\right). \tag{3.8}$$

These are also called *maximum entropy models*, since the dual optimization problem of Eq. 3.6 with $L = L_{\text{LR}}$ becomes that of picking a distribution $P(Y|X)$ with maximal entropy subject to first-order moment matching constraints (*i.e.*, constraints of the kind $\mathbb{E}_{Y|x}(\boldsymbol{f}(x, Y)) = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{f}(x^n, y^n)$), up to some slack which depends on the regularizer. This is a manifestation of the *conjugate duality* relation that we revisit in Section 4.3.1.

Support Vector Machines. If all we want is a model for prediction, probabilistic models are not a real necessity, and we may well go without them.[10] Support vector machines are purely discriminative classifiers based on the principle of maximizing the *margin of separation*. We describe here the multi-class formulation put forth by Crammer and Singer (2002), which addresses the following learning problem:

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \xi_n \\ \text{w.r.t.} \quad & \boldsymbol{w} \in \mathbb{R}^D, \xi \in \mathbb{R}^N_+ \\ \text{s.t.} \quad & \boldsymbol{w} \cdot \boldsymbol{f}(x_n, y_n) \geq \boldsymbol{w} \cdot \boldsymbol{f}(x_n, y'_n) + \rho(y'_n, y_n) - \xi_n, \quad \forall n \in \{1, \ldots, N\}, \forall y'_n \in \mathcal{Y}. \end{aligned} \tag{3.9}$$

To better understand what Eq. 3.9 is optimizing, let us regard the variables ξ_n as *slack variables*. The quantity $1/\|\boldsymbol{w}\|$ corresponds to the *separation margin*, which we seek to maximize. With $\xi_n = 0$, the constraints require that the parameters $\boldsymbol{w}$ are such that the score of the true output y_n exceeds that of any competitor $y'_n \neq y_n$ by an amount of at least the value of the

[10] A good reason for doing so is that, from the viewpoint of statistical learning theory, estimating a probability distribution is a more complex problem than that of training a classifier with good generalization capability. It may not be wise to embrace such a more complex problem as an intermediate step (Vapnik, 2000). However, there might be reasons for considering probabilistic models, since they have a very appealing semantics, well-defined procedures for dealing with hidden variables, and are able to complement predictions with well-founded measures of confidence (such as a posterior distribution over outputs).

cost $\rho(y'_n, y_n)$. When $\xi_n > 0$, some violations of these constraints are permitted, but they pay a linear penalty (cf. the term $\sum_{n=1}^{N} \xi_n$ in the objective).

We can rewrite Eq. 3.9 in the form of Eq. 3.6 by letting Ω be a L_2-regularizer, and defining the following *hinge loss* function:

$$L_{\text{SVM}}(\boldsymbol{w}; x, y) := -\boldsymbol{w} \cdot \boldsymbol{f}(x, y) + \max_{y' \in \mathcal{Y}} \left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y') + \rho(y', y) \right). \tag{3.10}$$

Comparing Eqs. 3.8 and 3.10, we observe that the logistic and hinge losses have resemblances: where the former uses a soft-max, the latter uses a max; and the hinge loss explicitly includes the cost function in the scope of this max. We will revisit this comparison in Chapter 8. Finally, it is worth mentioning that, even though support vector machines are most commonly used with L_2-regularization, that is not a strict necessity.[11] We may arbitrarily cross the loss functions described in the last and current sections with the regularizers seen in Section 3.2.1.

3.3 What is Structured Prediction?

In Chapter 2, we described several instances of *structured prediction* problems. The term is usually employed to denote classification problems where $\mathcal{Y}$ is a very large set, endowed with some sort of *structure*. But what does "structure" mean in this context? Why does structured prediction deserve special treatment? What is there that precludes a straightforward adaptation of the tools and techniques that already exist for multi-class classification?

We address these questions by explicitly formulating the assumptions that, in the scope of this thesis, characterize structured prediction problems.

- **First assumption: input-specific outputs.** When there are just a few classes, we predict y by searching from the entire output set $\mathcal{Y}$; this is commonly inappropriate in structured prediction. For example, in sequence labeling problems (Section 2.1), some outputs are structurally impossible: if x is a sequence of length L, then the corresponding y must be of the same size. In parsing tasks (Sections 2.2–2.3), we know that the yield of a parse tree must be the input sentence.

  To take this into consideration, we denote by $\mathcal{Y}(x)$ the set of *admissible outputs* for a given $x \in \mathcal{X}$. We define $\mathcal{Y} = \bigcup_{x \in \mathcal{X}} \mathcal{Y}(x)$. We assume that the true distribution $P(X, Y)$ vanishes on non-admissible pairs: that is, we must have $P(x, y) = 0$ if $y \notin \mathcal{Y}(x)$.

- **Second assumption: hardness of enumeration.** We assume that the output set $\mathcal{Y}(x)$ is so large that it is intractable to enumerate its elements. In sequence labeling, $|\mathcal{Y}(x)|$ grows exponentially with the length of the input string x; in dependency parsing, the number of valid parses for a sentence with L words is $(L+1)^{L-1}$, which is even worse.[12]

[11] See Bradley and Mangasarian (1998) for L_1-regularized variants of support vector machines. However, note that those variants cannot be kernelized: the Representer Theorem of Kimeldorf and Wahba (1971) hinges on L_2-regularization. For more details, see Schölkopf and Smola (2002).

[12] This is a consequence of Cayley's formula (Borchardt, 1860; Cayley, 1889), which states that the number of undirected spanning trees of a complete graph on N vertices is N^{N-2}. The result follows from observing that a non-projective dependency tree spans $L+1$ words (including the dummy root word) and the orientation of the edges is automatically determined given this root word.

We assume that $|\mathcal{Y}(x)|$ grows superpolynomially with some meaningful measure of "length" for the input x (for example, the number of bits necessary to encode x in a machine). For every $x \in \mathcal{X}$, we assume that there is an integer $L(x)$ such that $\mathcal{Y}(x)$ can be represented as a subset of a product set $\mathcal{Y}_1 \times \ldots \times \mathcal{Y}_{L(x)}$, where each $\mathcal{Y}_i$ is finite. Hence, outputs $y \in \mathcal{Y}(x)$ can be seen as *tuples of atomic components*, $y := (y_i)_{i=1}^{L(x)}$.

- **Third assumption: global coupling.** There are *strong statistical dependencies* among the atomic components. In other words, we assume that the true joint distribution cannot be accurately approximated as $P(X,Y) \approx P(X)\prod_{i=1}^{L(x)} P(Y_i|X)$—otherwise, the *global* problem could be trivially split as a sequence of *local* atomic ones. Besides this, in some tasks (such as parsing) there may be *global structural constraints* regarding which tuples are valid representations of elements of $\mathcal{Y}(x)$.

  To sum up, we assume that the atomic output variables $Y_1, \ldots, Y_{L(x)}$ are "globally" coupled through structural constraints, statistical interdependencies, or both.

Putting together all aspects above, we arrive at the following informal definition:

Definition 3.2 (Structured prediction.) *Structured prediction refers to a classification problem in which the three assumptions above are met.*

According to this definition, the "structure" emanates from the global coupling assumption; and the inadequacy of standard multi-class classification methods stems from the hardness of enumeration assumption, which, among other things, implies that the inference problem in Eq. 3.5 cannot be solved efficiently by brute force.

We distinguish between two kinds of approaches that have been taken to address structured prediction problems: *greedy methods* and *global optimization methods*.

Greedy methods. These methods aim to approximate the inference problem in Eq. 3.5 by performing a sequence of greedy decisions. Typically, atomic outputs are predicted one at the time, and predictions made in the past are included as features to help make later decisions—these are commonly called *history-based features*. These approaches have been pursued in sequence labeling (Giménez and Marquez, 2004), phrase-structure parsing (Black et al., 1993; Magerman, 1995; Ratnaparkhi, 1999), and transition-based dependency parsing (Yamada and Matsumoto, 2003; Nivre et al., 2006; Huang and Sagae, 2010; Goldberg and Elhadad, 2010). The main advantages of greedy methods are: (i) their ability to incorporate non-local contextual features, and (ii) their computational efficiency. If we ignore the time necessary to evaluate features and compute scores, the prediction runtimes of greedy methods are essentially the same as if the global coupling assumption were ignored. They also permit efficient learning, once the data instances are mapped into a sequence of actions.[13] It all boils down to training one or several binary of multi-class classifiers. The great bottleneck of greedy methods is *error propagation*: if a mistake is made early on, then all future decisions will be severely affected—this leads to *burst errors*, which are often undesirable. Recently, search-based methods (*e.g.*, Pal et al. 2006; Daumé et al. 2009) were proposed that integrate learning and search, while reducing greediness by using beam search, A^* search,

[13] It is worth noting, however, that this mapping is usually one-to-many, *i.e.*, different actions may lead to the same instance. Heuristics are commonly used to pick a canonical sequence of actions.

or other combinatorial search algorithms that are able to backtrack. While these techniques can alleviate error propagation and they can "learn" to do better inference, this comes at the cost of increasing runtime, and there is usually a trade-off between these two.

Global optimization methods. These methods attack directly the optimization problem in Eq. 3.5. Solving this problem exactly (*exact decoding*) usually requires designing the model in a clever way, by making certain *factorization assumptions* that limit the interactions between atomic outputs. In sequence labeling and parsing tasks, well-known examples are hidden Markov models (Rabiner, 1989), conditional random fields (Lafferty et al., 2001), stochastic context-free grammars (Magerman, 1995; Eisner, 1996; Charniak, 1997; Johnson, 1998; Collins, 1999), and arc-factored dependency parsers (McDonald et al., 2005b); all these were reviewed in Chapter 2 and correspond to some decomposition of the form

$$h_{\boldsymbol{w}}(x) = \arg\max_{y \in \mathcal{Y}(x)} \sum_{p \in \mathcal{P}} \boldsymbol{w} \cdot \boldsymbol{f}_p(x, y_p), \tag{3.11}$$

where $\mathcal{P}$ is a set of *places*. A suitable way of representing the model factorization is via graphical models, as will be seen in detail in Chapter 4; there, the places correspond to the factors of a factor graph. The main advantage of global optimization methods over greedy methods is that they do not suffer from error propagation. The price to pay is that the model factorizations that permit tractable inference are often too stringent. Typically, designing and extending the model are tasks that have to obey algorithmic considerations. It is often worthwhile to embrace *approximate inference* techniques (non-greedy ones) to tackle those problems. All this will be discussed in Sections 4.5–4.6.

The approaches investigated in this thesis are based on *global optimization*. In particular, we devise approximate inference algorithms that can handle the sort of *non-local features* that are typically incorporated in greedy methods, hence alleviating what is seen as the main drawback of global optimization methods for structured prediction.

3.4 Discriminative Learning of Structured Predictors

We now adapt the supervised learning framework described in Sections 3.1–3.2 for learning *structured predictors*. We assume a linear model where the features decompose as

$$\boldsymbol{f}(x, y) := \sum_{p \in \mathcal{P}} \boldsymbol{f}_p(x, y_p), \tag{3.12}$$

where each $p \in \mathcal{P}$ is called a *place*, and each pair (p, y_p) is called a *part*. The partial output y_p takes values in a set $\mathcal{Y}_p$, which is typically a product set of atomic outputs (for example, in sequence models, places refer to unigrams and bigrams). We assume that the set of places $\mathcal{P}$ includes the atomic places $i \in \{1, \ldots, L(x)\}$ mentioned above. This will be made more formal in Chapter 4, when we talk about factor graphs. With this decomposition, the inference problem becomes that of Eq. 3.11.

3.4.1 Structured Cost Functions

Let us now discuss the cost functions that are suitable for structured prediction problems. One thing to note is that the 0/1 cost function, which is the typical choice in multi-class classification problems, is in general inadequate in structured prediction, because it does not give partial credit for getting most of the structure right. Toward this end, we consider *decomposable* cost functions.[14]

Definition 3.3 (Decomposable cost.) *A cost function* $\rho : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ *is* decomposable *if it can be written as*

$$\rho(\widehat{y}, y) = \sum_{p \in \mathcal{P}} \rho_p(\widehat{y}_p, y_p), \tag{3.13}$$

for some choice of "local" cost functions $\rho_p : \mathcal{Y}_p \times \mathcal{Y}_p \to \mathbb{R}_+$.

Example 3.1 (Hamming cost.) *Let* $\widehat{y} = (\widehat{y}_i)_{i=1}^{L(x)}$ *and* $y = (y_i)_{i=1}^{L(x)}$. *The Hamming cost is defined as:*

$$\rho_H(\widehat{y}, y) = \sum_{i=1}^{L(x)} [\![\widehat{y}_i \neq y_i]\!] \tag{3.14}$$

In words, $\rho_H(\widehat{y}, y)$ *returns the number of mismatched atomic outputs. Assuming, as stated above, that the set of places include the atomic places, then we have that the Hamming cost is decomposable, with* $\rho_p \equiv 0$ *for every non-atomic place* p, *and* $\rho_{\{i\}} \equiv \rho_{0/1}$ *for every* $i \in \{1, \ldots, L(x)\}$.

We point out, however, that decomposable cost functions are not always the most suitable for some problems—while the Hamming cost gives a measure of error rate regarding how much of the structure was predicted correctly, for some problems metrics based on precision, recall, or F_β-measure are the most suitable. While precision and recall can both be captured with decomposable costs, that is not the case with F_β.

3.4.2 Conditional Random Fields

A *conditional random field* or CRF (Lafferty et al., 2001) is a generalization of a logistic regression model for structured prediction. One considers a conditional probabilistic model that takes the form in Eq. 3.7, but where $\boldsymbol{f}(x, y)$ has a decomposition as in Eq. 3.12.[15] This yields:

$$P_{\boldsymbol{w}}(y|x) \quad := \quad \frac{1}{Z(\boldsymbol{w}, x)} \exp\left(\sum_{p \in \mathcal{P}(x)} \boldsymbol{w} \cdot \boldsymbol{f}_p(x, y_p) \right), \tag{3.15}$$

where $Z(\boldsymbol{w}, x)$ is the partition function.

[14] To accommodate the fact that sets of admissible outputs may be input dependent, we assume that any cost function $\rho : \mathcal{Y} \times \mathcal{Y}$ is infinity-valued for pairs $(\widehat{y}, y)$ for which there is no x such that both $\widehat{y}$ and y belong to $\mathcal{Y}(x)$; in what follows we abbreviate and define ρ as if its domain was $\mathcal{Y}(x) \times \mathcal{Y}(x)$.

[15] Strictly speaking, in a CRF the decomposition is with respect to the cliques of a *Markov network* (which will be defined in Chapter 4), hence $P(Y|X)$ forms a Markov random field (hence the name). However, it is common to say "conditional random field" even when the underlying structure is not described by a Markov network, as soon as it decomposes as in Eq. 3.12. An example is "CRF parsing" (Finkel et al., 2008), where the structure comes from a context-free grammar.

The loss function of conditional random fields (called *structured logistic loss*) corresponds to the negative conditional log-likelihood, which can be written as the soft-max (log-sum-exp) of a term which decomposes into parts:

$$\begin{aligned} L_{\text{CRF}}(\boldsymbol{w}; x, y) &:= -\boldsymbol{w} \cdot \boldsymbol{f}(x, y) + \log \sum_{y' \in \mathcal{Y}(x)} \exp\left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y')\right) \\ &= \log \sum_{y' \in \mathcal{Y}(x)} \exp\left(\sum_{p \in \mathcal{P}} \boldsymbol{w} \cdot \delta \boldsymbol{f}_p(x, y_p, y'_p)\right) \end{aligned} \tag{3.16}$$

where $\delta \boldsymbol{f}_p(x, y_p, y'_p) := \boldsymbol{f}_p(x, y'_p) - \boldsymbol{f}_p(x, y_p)$. Given a model trained by minimizing the (regularized) structured logistic loss, there has been two decoding strategies proposed in the literature: *MAP decoding* and *minimum risk decoding*.

MAP decoding. MAP decoding (also called *Viterbi decoding*) seeks the configuration with maximal probability according to the model:

$$\widehat{y} := \arg\max_{y \in \mathcal{Y}(x)} \Pr_{\boldsymbol{w}}\{Y = y | x\}. \tag{3.17}$$

Minimum risk decoding. Minimum risk decoding is a decoding rule that can be used in any probabilistic model, and which is commonly used with CRFs. It corresponds to predicting the output $\widehat{y} \in \mathcal{Y}(x)$ which minimizes the decision risk $\mathbb{E}_{\boldsymbol{w}}\left(\rho(\widehat{y}, Y)\right)$. Under a 0/1 cost, it reverts to the MAP decoding rule. When a Hamming cost is used, the objective to minimize becomes:

$$\begin{aligned} \mathbb{E}_{\boldsymbol{w}}\left(\rho_H(y, Y)\right) &= \mathbb{E}_{\boldsymbol{w}}\left(\sum_{i=1}^{L(x)} \rho_{0/1}(y_i, Y_i)\right) = \sum_{i=1}^{L(x)} \mathbb{E}_{\boldsymbol{w}}\left(\rho_{0/1}(y_i, Y_i)\right) \\ &= \sum_{i=1}^{L(x)} (1 - \Pr_{\boldsymbol{w}}\{Y_i = y_i | x\}), \end{aligned} \tag{3.18}$$

hence under a Hamming cost, the minimum risk decoding rule becomes:

$$\widehat{y} := \arg\max_{y \in \mathcal{Y}(x)} \sum_{i=1}^{L(x)} \Pr_{\boldsymbol{w}}\{Y_i = y_i | x\}, \tag{3.19}$$

in other words, one needs to compute the posterior marginals $\Pr_{\boldsymbol{w}}\{Y_i = y_i | x\}$ and then obtain the configuration that maximizes the sum of marginals. When the outputs are unconstrained, *i.e.*, $\mathcal{Y}(x) \cong \prod_{i=1}^{L(x)} \mathcal{Y}_i$, this rule amounts to maximizing the posterior marginals componentwise, $\widehat{y}_i = \arg\max_{y_i \in \mathcal{Y}_i} \Pr_{\boldsymbol{w}}\{Y_i = y_i | x\}$.

3.4.3 Structured Support Vector Machines

Support vector machines can be also generalized for structured prediction, as shown by Taskar et al. (2003); Altun et al. (2003); Tsochantaridis et al. (2004). We present here the formulation of Taskar et al. (2003), which takes the name *max-margin Markov networks*. Rather than the likelihood of a probabilistic model, the goal is now to maximize the separation

margin, with respect to a cost function ρ, which we assume to be decomposable. The corresponding loss function is called *structured hinge loss* and is written as:

$$\begin{aligned} L_{\text{SSVM}}(\boldsymbol{w}; x, y) &:= -\boldsymbol{w} \cdot \boldsymbol{f}(x, y) + \max_{y' \in \mathcal{Y}(x)} \left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y') + \rho(y', y) \right) \\ &= \max_{y' \in \mathcal{Y}(x)} \left(\sum_{p \in \mathcal{P}(x)} \left(\boldsymbol{w} \cdot \delta \boldsymbol{f}_p(x, y_p, y'_p) + \rho_p(y'_p, y_p) \right) \right) \end{aligned} \tag{3.20}$$

where $\delta \boldsymbol{f}_p(x, y_p, y'_p)$ is defined similarly as in Eq. 3.16. Comparing Eq. 3.16 with Eq. 3.20, we observe only two differences:

- The soft-max operator in Eq. 3.16 is replaced by a max operator in Eq. 3.20;
- Eq. 3.20 includes the cost function in the scope of the maximization.

We will come back to this issue in Chapter 8, where we will devise a family of loss functions that interpolate between conditional random fields and structured support vector machines.

3.5 Online and Stochastic Learning Algorithms

Now that we have discussed different learning formalisms for structured prediction, expressing them as optimization problems, it is time to describe *learning algorithms* for tackling these problems. Recall the general optimization problem of Eq. 3.6:

$$\begin{array}{ll} \text{minimize} & \Omega(\boldsymbol{w}) + \frac{1}{N} \sum_{n=1}^{N} L(\boldsymbol{w}; x^n, y^n) \\ \text{w.r.t.} & \boldsymbol{w} \in \mathbb{R}^D. \end{array} \tag{3.21}$$

An important requirement is that algorithms are *scalable*, *i.e.*, they must have reasonable memory consumption needs and manageable runtime when N and D are large. This requirement rules out some of the fast optimizers, such as interior-point methods, which have superlinear convergence guarantees. On the other hand, algorithms with slower convergence rates, such as the gradient method or its stochastic/online counterparts, look suitable for large-scale machine learning: even though they are usually slow to converge asymptotically, they are fast to reach a region close to the optimum, hence can be stopped early and retrieve a model with good generalization properties. Bottou and Bousquet (2007) describe in more detail this tradeoff between *approximation*, *estimation*, and *optimization* error;[16] the inclusion of the latter is distinctive of large-scale settings.

The development of algorithms for learning structured predictors is a very active area of research. We limit our discussion to the algorithms that will be used in our thesis; we omit others, such as the *exponentiated gradient* (Collins et al., 2008) and the primal and dual *extragradient* (Taskar et al., 2006a,b), which are amongst to the most competitive for this task.

We review in Appendix B some basic concepts of convex analysis that are used throughout the remaining sections.

[16] *Approximation error* is the difference between the minimum expected risk (for some oracle $h : \mathcal{X} \to \mathcal{Y}$ not necessarily in the hypothesis class $\mathcal{H}$) and the best achievable one in $\mathcal{H}$; *estimation error* is the difference between the best achievable expected risk and that of the hypothesis minimizing Eq. 3.21; *optimization error* is the difference between the latter term and the actual model which is returned by an algorithm optimizing Eq. 3.21 up to a tolerance.

3.5.1 Stochastic Gradient Descent

For each instance n, define $F_n(\boldsymbol{w}) := \Omega(\boldsymbol{w}) + L(\boldsymbol{w}; x^n, y^n)$, and $F(\boldsymbol{w}) := \frac{1}{N}\sum_{n=1}^{N} F_n(\boldsymbol{w})$. The learning problem in Eq. 3.21 corresponds to an unconstrained minimization of $F(\boldsymbol{w})$. Let us assume for a moment that $F : \mathbb{R}^D \to \mathbb{R}$ is differentiable, and denote by $\nabla F(\boldsymbol{w})$ the gradient of F at $\boldsymbol{w}$, which is given by $\nabla F(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \nabla F_n(\boldsymbol{w})$. A gradient method for solving this problem would iteratively pick a stepsize η_t and apply an update rule of the form

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta_t \nabla F(\boldsymbol{w}^t). \tag{3.22}$$

A proper choice of stepsizes ensures a convergence rate of $O(1/\epsilon)$, in the sense that at most $T = O(1/\epsilon)$ iterations are necessary to achieve an objective value which differs no more than ϵ from the optimum (this difference is the *optimization error*). With accelerated gradient techniques (Nesterov, 1983) it is even possible to achieve a faster convergence rate of $O(1/\sqrt{\epsilon})$. However, a single iteration of these methods requires processing an entire batch of data for computing the gradient $\nabla F(\boldsymbol{w})$. This makes the gradient method unsatisfying in a large-scale setting, where N is large. One possibility to sidestep this issue is to parallelize the computation; another, which we describe next, is to use stochastic gradients.

Stochastic gradient methods (Robbins and Monro, 1951; Bottou, 1991a, 2004) replace each gradient evaluation with a "noisy" estimate built from a single instance. In other words, at each iteration t (called *round*) they pick an instance $n(t)$ uniformly at random, with replacement; then the following update is made based only on that instance:

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t - \eta_t \nabla F_{n(t)}(\boldsymbol{w}^t), \tag{3.23}$$

When this choice is not done at random but instead cycles over the data through $n(t) = t - N\lfloor t/N \rfloor$, the method is called *online gradient* (or *incremental gradient*, in the optimization literature).[17] In stochastic and online gradient methods, one can obtain convergence for a suitable choice of the stepsize sequence $(\eta_t)_{t\in\mathbb{N}}$, which in this context is commonly called the *learning rate*. In general, by choosing $\eta_t = O(1/\sqrt{t})$, one can guarantee $O(1/\epsilon^2)$ convergence (in a PAC[18] setting), for arbitrary convex loss functions and regularizers. This result can be established following the analysis of Zinkevich (2003) for online convex programming, and adopting a simple online-to-batch conversion that averages the iterates (Cesa-Bianchi et al., 2004). That convergence rate can be improved if the objective function is λ-strongly convex (which always happens if we use L_2-regularization, $\Omega := \Omega_\lambda^{L_2}$, provided the loss function L is convex), in which case we have $O(1/(\lambda\epsilon))$ convergence with $\eta_t := O(1/(\lambda t))$. The analysis is due to Hazan et al. (2007) and is essentially the same that underlies the Pegasos algorithm (Shalev-Shwartz et al., 2007, 2010).

Stochastic gradient descent algorithms have been proposed as a way of training L_2-regularized CRFs in large-scale settings (Vishwanathan et al., 2006), achieving in practice faster progress than more sophisticated batch optimization methods that were previously used, such as conjugate gradient and L-BFGS (Sha and Pereira, 2003). With $\Omega := \Omega_\lambda^{L_2}$, the

[17] There are also methods that use mini-batches instead of a single instance. The theoretical analysis of such methods is very similar to that of stochastic/online gradient descent.

[18] "Probably approximately correct" (Valiant, 1984).

Algorithm 1 Stochastic Gradient Descent for L_2-regularized CRFs/SVMs

1: **input:** $\mathcal{D}$, λ, loss function L, number of rounds T, learning rate sequence $(\eta_t)_{t=1,\dots,T}$
2: initialize $\boldsymbol{w}^1 := \mathbf{0}$
3: **for** $t = 1$ **to** T **do**
4: choose $n = n(t)$ randomly and take training pair (x^n, y^n)
5: computing the gradient or a subgradient $\boldsymbol{g} := \widetilde{\nabla}_{\boldsymbol{w}} L(\boldsymbol{w}^t, x^n, y^n)$ (Eqs. 3.25 and 3.27)
6: update the model:

$$\boldsymbol{w}^{t+1} \leftarrow (1 - \lambda\eta_t)\boldsymbol{w}^t - \eta_t \boldsymbol{g}$$

7: **end for**
8: **output:** the last weight vector $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{T+1}$, or the average $\widehat{\boldsymbol{w}} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}^{t+1}$

gradient of F_n at $\boldsymbol{w}$ becomes $\nabla F_n(\boldsymbol{w}) = \lambda \boldsymbol{w} - \nabla_{\boldsymbol{w}} L(\boldsymbol{w}, x^n, y^n)$, hence the updates become

$$\boldsymbol{w}^{t+1} \leftarrow (1 - \lambda\eta_t)\boldsymbol{w}^t + \eta_t \nabla_{\boldsymbol{w}} L(\boldsymbol{w}, x^n, y^n). \tag{3.24}$$

This yields Algorithm 1. When applied to CRF training, one needs to compute the gradient of the structured logistic loss, which is[19]

$$\begin{aligned} \nabla L_{\text{CRF}}(\boldsymbol{w}, x, y) &= -\boldsymbol{f}(x, y) + \nabla_{\boldsymbol{w}} \log Z(\boldsymbol{w}, x) \\ &= -\boldsymbol{f}(x, y) + \mathbb{E}_{\boldsymbol{w}}(\boldsymbol{f}(x, Y)), \end{aligned} \tag{3.25}$$

where $\mathbb{E}_{\boldsymbol{w}}(\boldsymbol{f}(x, Y)) = \sum_{p \in \mathcal{P}} \sum_{y_p \in \mathcal{Y}_p(x)} \mu_p(y_p) \boldsymbol{f}_p(x, y_p)$ is the vector of feature expectations; to compute this vector, all that is necessary is the set of posterior marginals:

$$\mu_p(y_p) := \Pr_{\boldsymbol{w}}\{Y_p = y_p | x\}. \tag{3.26}$$

Hence, each round of the Algorithm 1 requires performing a *marginal inference step* in order to compute the gradient. This is the step that usually requires the most computational effort. In Chapter 8, we will consider problems for which marginal inference is not tractable and has to be approximated. We will also introduce an algorithm which does not require the specification of a learning rate.

3.5.2 Stochastic Subgradient Descent

An analogous procedure can be applied for learning structured support vector machines. There is a small twist though, which is the fact that the structured hinge loss is not differentiable. However, the same algorithm works with *subgradients*, as shown by Ratliff et al. (2006) and Shalev-Shwartz et al. (2007, 2010). Let $\partial F(\boldsymbol{w}_0)$ be the subdifferential of F at a point $\boldsymbol{w}_0$, and denote by $\widetilde{\nabla} F(\boldsymbol{w}_0) \in \partial F(\boldsymbol{w}_0)$ a particular subgradient; define analogously $\partial L_{\text{SSVM}}(\boldsymbol{w}_0)$ and $\widetilde{\nabla} L_{\text{SSVM}}(\boldsymbol{w}_0)$ for the loss function L_{SSVM}. From Danskin's theorem (see Appendix B), we have that

$$\boldsymbol{f}(x, \widehat{y}) - \boldsymbol{f}(x, y) \in \partial L_{\text{SSVM}}(\boldsymbol{w}, x, y), \tag{3.27}$$

[19] We use a result to be derived in Chapter 4 that states that the gradient of the log-partition function equals the vector of feature expectations (cf. Eq. 4.30).

where $\widehat{y}$ is a solution of the *cost-augmented inference problem*:

$$\widehat{y} = \arg\max_{y' \in \mathcal{Y}(x)} \left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y') + \rho(y', y) \right). \tag{3.28}$$

The update equations become:

$$\boldsymbol{w}^{t+1} \leftarrow (1 - \eta_t \lambda)\boldsymbol{w}^t + \eta_t \left(\boldsymbol{f}(x^{n(t)}, y^{n(t)}) - \boldsymbol{f}(x^{n(t)}, \widehat{y}^{n(t)}) \right). \tag{3.29}$$

The same rule as above applies for the learning rates: by choosing $\eta_t = O(1/\sqrt{t})$, one can guarantee $O(1/\epsilon^2)$ convergence; since we are considering L_2-regularization, a better asymptotic convergence of $O(1/(\lambda\epsilon))$ can be achieved with a faster decaying schedule $\eta_t = O(1/(\lambda t))$. The latter is essentially the Pegasos algorithm (Shalev-Shwartz et al., 2007, 2010).[20] The similarity with the CRF case is striking: at each round, instead of computing the feature vector expectation using the current model, we compute the feature vector associated with the cost-augmented prediction using the current model. If the cost function is decomposable, then this step can be solved with an inference algorithm for computing the MAP assignment: all that is necessary is to redefine the scores for each part by adding to $\boldsymbol{w} \cdot \boldsymbol{f}_p(x, y'_p)$ the quantity $\rho_p(y'_p, y_p)$, and then call a MAP decoder. Throughout this thesis, we will see multiple problems for which this step is not tractable, and approximate MAP inference is necessary. We defer details for Chapters 5, 7 and 8. This is also the step that usually requires the most computational effort.

3.5.3 The Perceptron Algorithm

Perhaps the oldest algorithm for training a linear classifier is the *perceptron algorithm* (Rosenblatt, 1958), depicted as Algorithm 2. The perceptron was first applied to structured prediction by Collins (2002a). The algorithm works as follows: at each round, it takes an input datum, and uses the current model to make a prediction. If the prediction is correct, nothing happens (the update in line 6 will be vacuous in that case). Otherwise (*i.e.*, if the model made a *mistake*), a correction will take place by adding to $\boldsymbol{w}$ the feature vector with respect to the correct output and subtracting the feature vector with respect to the predicted (wrong) output. Algorithm 2 is remarkably simple; yet it often reaches a very good performance, usually not much worse than that obtained with CRFs or SSVMs. The most important property of the perceptron algorithm is its ability to find a separating hyperplane, as we will see next.

Definition 3.4 (Separability.) *We say that the training data $\mathcal{D}$ is separable with margin $\gamma > 0$ (or simply γ-*separable*) if there is a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ with unit norm ($\|\boldsymbol{w}\| = 1$) such that*

$$\boldsymbol{w} \cdot \boldsymbol{f}(x^n, y^n) > \boldsymbol{w} \cdot \boldsymbol{f}(x^n, y') + \gamma, \quad \forall n \in \{1, \ldots, N\}, \forall y' \in \mathcal{Y}(x^n) \setminus \{y^n\}, \tag{3.30}$$

in which case any vector of the form $c\boldsymbol{w}$ with $c > 0$ is said to separate $\mathcal{D}$. We say that $\mathcal{D}$ is separable *if the set $\{\gamma > 0 \mid \mathcal{D}$ is γ-separable$\}$ is non-empty, in which case its supremum is called the* margin of separation *of $\mathcal{D}$.*

[20] The original version of Pegasos includes a projection onto the L_2-ball, but convergence is not affected by omitting that projection, as established by Shalev-Shwartz et al. (2010).

Algorithm 2 Structured Perceptron

1: **input:** data $\mathcal{D}$, number of rounds T
2: initialize $\boldsymbol{w}^1 := \mathbf{0}$
3: **for** $t = 1$ **to** T **do**
4: choose $n = n(t)$ randomly
5: take training pair (x^n, y^n) and predict using the current model $\boldsymbol{w}^t$:

$$\widehat{y} := \arg \max_{y' \in \mathcal{Y}(x^n)} \boldsymbol{w}^t \cdot \boldsymbol{f}(x^n, y')$$

6: update the model:

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \boldsymbol{f}(x^n, y^n) - \boldsymbol{f}(x^n, \widehat{y})$$

7: **end for**
8: **output:** the last weight vector $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{T+1}$, or the average $\widehat{\boldsymbol{w}} \leftarrow \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}^{t+1}$

Proposition 3.1 (Rosenblatt 1958; Collins 2002a) *Let $\mathcal{D}$ be γ-separable, and let R be such that $\|\boldsymbol{f}(x^n, y^n) - \boldsymbol{f}(x^n, y')\| \leq R$ for any $n \in \{1, \ldots, N\}$ and $y' \in \mathcal{Y}(x^n)$. Then the perceptron algorithm converges to a separating weight vector, after committing at most $\lfloor R^2/\gamma^2 \rfloor$ mistakes.*

We can regard the structured perceptron as a subgradient method for solving a learning problem of the form in Eq. 3.21 (albeit without regularization), with $L := L_{\text{SP}}$ defined as:

$$\begin{aligned} L_{\text{SP}}(\boldsymbol{w}; x, y) &:= -\boldsymbol{w} \cdot \boldsymbol{f}(x, y) + \max_{y' \in \mathcal{Y}(x)} \left(\boldsymbol{w} \cdot \boldsymbol{f}(x, y')\right) \\ &= \max_{y' \in \mathcal{Y}(x)} \left(\sum_{r \in \mathcal{R}(x)} \boldsymbol{w} \cdot \delta \boldsymbol{f}_r(x, y_r, y'_r) \right) \end{aligned} \tag{3.31}$$

where $\delta \boldsymbol{f}_r(x, y_r, y'_r)$ is defined similarly as in Eq. 3.16. We call this the *structured perceptron* loss; note the resemblance with the structured hinge loss (Eq. 3.20), the difference being the lack of the cost term. To guarantee convergence, though, we would need to add a suitable stepsize for the updates in Algorithm 2.

3.5.4 Online Passive-Aggressive Algorithms

Online passive-aggressive algorithms (Crammer and Singer, 2003; Crammer et al., 2006) are a refinement of the perceptron algorithm, outperforming the latter in a number of tasks. They are related with the previously proposed Margin Infused Relaxed Algorithm (MIRA), in particular they correspond to what has been called 1-best MIRA in McDonald et al. (2005a). Here, we talk solely about the so-called max-loss variant of PA-I (in the terminology of Crammer et al. 2006), which very much resembles the stochastic subgradient algorithm for SVMs, but with an automatic mechanism for setting the stepsize.

At each round t, the passive aggressive algorithm updates the weight vector by setting it

Algorithm 3 Online Passive Aggressive (max-loss, PA-I)

input: data $\mathcal{D}$, parameter λ, number of rounds T
initialize $\boldsymbol{w}^1 := \mathbf{0}$
for $t = 1$ **to** T **do**
  choose $n = n(t)$ randomly
  take training pair (x^n, y^n) and compute a cost-augmented prediction using the current model $\boldsymbol{w}^t$:

$$\widehat{y} := \arg\max_{y' \in \mathcal{Y}(x^n)} \boldsymbol{w}^t \cdot \boldsymbol{f}(x^n, y') + \rho(y', y^n)$$

  if $\widehat{y} = y^n$ **then**
    keep the model unchanged, $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t$
  else
    compute loss: $\ell^t = \boldsymbol{w}^t \cdot \boldsymbol{f}(x^n, \widehat{y}) - \boldsymbol{w}^t \cdot \boldsymbol{f}(x^n, y^n) + \rho(\widehat{y}, y^n)$
    compute stepsize: $\eta_t = \min\left\{\lambda^{-1}, \frac{\ell^t}{\|\boldsymbol{f}(x^n, y^n) - \boldsymbol{f}(x^n, \widehat{y})\|^2}\right\}$
    update the model: $\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \eta_t(\boldsymbol{f}(x^n, y^n) - \boldsymbol{f}(x^n, \widehat{y}))$
  end if
end for
output: the last weight vector $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{T+1}$, or the average $\widehat{\boldsymbol{w}} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}^{t+1}$

to the solution of the following optimization problem, involving the $n := n(t)$th instance:

$$\begin{array}{ll} \text{minimize} & \frac{\lambda}{2}\|\boldsymbol{w} - \boldsymbol{w}^t\|^2 + \xi \\ \text{w.r.t.} & \boldsymbol{w} \in \mathbb{R}^D, \quad \xi \geq 0 \\ \text{s.t.} & \boldsymbol{w} \cdot \boldsymbol{f}(x^n, y^n) \geq \boldsymbol{w} \cdot \boldsymbol{f}(x^n, \widehat{y}) + \rho(\widehat{y}, y^n) - \xi, \end{array} \tag{3.32}$$

where

$$\widehat{y} := \arg\max_{y' \in \mathcal{Y}(x)} \boldsymbol{w}^t \cdot \boldsymbol{f}(x^n, y') + \rho(y', y^n) \tag{3.33}$$

is the "cost-augmented prediction." By inspecting Eq. 3.32 we see that the passive-aggressive scheme attempts to achieve a tradeoff between *conservativeness*—by penalizing large changes from the previous weight vector via the term $\frac{\lambda}{2}\|\boldsymbol{w} - \boldsymbol{w}^t\|^2$—and *correctness*—by requiring, through the constraints, that the new model $\boldsymbol{w}^{t+1}$ "separates" the true output from the prediction with a margin (although slack $\xi \geq 0$ is allowed). The intuition for this large margin separation is the same for support vector machines. Note that if the cost-augmented prediction matches the true output ($\widehat{y} = y^m$), which means that the current model already achieves a large margin on this example, then the solution of the problem Eq. 3.32 leaves the weight vector unchanged ($\boldsymbol{w}^{t+1} = \boldsymbol{w}^t$). Otherwise, this quadratic programming problem has a closed form solution:

$$\boldsymbol{w}^{t+1} \leftarrow \boldsymbol{w}^t + \eta_t(\boldsymbol{f}(x^m, y^m) - \boldsymbol{f}(x^m, \widehat{y})), \tag{3.34}$$

with

$$\eta_t = \min\left\{\lambda^{-1}, \frac{\boldsymbol{w}^t \cdot \boldsymbol{f}(x^m, \widehat{y}) - \boldsymbol{w}^t \cdot \boldsymbol{f}(x^m, y^m) + \rho(\widehat{y}, y^m)}{\|\boldsymbol{f}(x^m, y^m) - \boldsymbol{f}(x^m, \widehat{y})\|^2}\right\}. \tag{3.35}$$

The resulting algorithm is depicted as Algorithm 3. For other variants of the passive-aggressive scheme, see Crammer et al. (2006).

Passive-aggressive algorithms can be interpreted as dual coordinate ascent algorithms for learning support vector machines in an online setting (Shalev-Shwartz and Singer, 2006; Kakade and Shalev-Shwartz, 2008). We will elaborate on this issue in Chapter 8, where we propose new variants of coordinate ascent algorithms to tackle other loss functions besides the structured hinge loss.

Chapter 4

Graphical Models and Inference Algorithms

In this chapter, we provide background on *probabilistic graphical models*, describing their role in modeling inference problems within the scope of structured prediction.

Graphical models enable compact representations of probability distributions and are widely used in computer vision, natural language processing, and computational biology (Pearl, 1988; Lauritzen, 1996; Koller and Friedman, 2009). Two kinds of models have been proposed in the literature: *directed* and *undirected*. Directed models express causal relationships, being called *Bayesian networks*. Their undirected counterparts—*Markov networks*—model dependencies among variables, rather than causality. There are ways of converting Bayes networks into Markov networks and vice-versa. In this thesis, undirected graphical models are our formalism of choice; in particular, we use intermediate representations called *factor graphs* (Tanner, 1981; Kschischang et al., 2001), which are more informative than Markov networks since they explicitly represent the factors of a distribution.

This chapter summarizes prior work in graphical models, with a focus on models and algorithms for approximate inference. We restrict discussion to *unconstrained* models. Later, in Chapters 5 and 6, we address the issue of hard constraints and present original contributions for constrained structured prediction.

4.1 Undirected Graphical Models

In this section, we describe Markov networks and factor graphs, two undirected graphical model formalisms. Before doing so, we briefly review some basic terminology of graph theory that will be used throughout.

An *undirected graph* is a tuple $\mathcal{G} := (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes, or vertices, and $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$ is a set of edges.[1] For convenience, we write uv to denote an edge $\{u, v\}$. Two nodes u and v are said to be *adjacent* or *neighbors* if $uv \in \mathcal{E}$. We denote by $\mathcal{N}(u)$ the *neighborhood* of a node u, $\mathcal{N}(u) := \{v \in \mathcal{V} \mid uv \in \mathcal{E}\}$. The cardinality of this set is called the *degree* of u, $\deg(u) := |\mathcal{N}(u)|$. A *clique* in $\mathcal{G}$ is a subset of nodes $\alpha \subseteq \mathcal{V}$ which are pairwise adjacent, *i.e.*, for any $u, v \in \alpha$ with $u \neq v$ we have $uv \in \mathcal{E}$. By definition, any singleton v is a clique. A clique is *maximal* if it is not contained in any other clique. A *path* in $\mathcal{G}$ is a sequence of nodes

[1] $\binom{\mathcal{V}}{2}$ denotes the set of all unordered pairs of distinct vertices.

$(v_0, \dots, v_n)$ such that $v_{i-1}v_i \in \mathcal{E}$ for every $i \in \{1, \dots, n\}$ and $v_1, \dots, v_n$ are distinct. Such a path is said to connect v_0 and v_n. If $v_0 = v_n$ the path is called a *cycle*. A graph is *connected* if any two nodes are connected by a path. A *tree* is a connected graph that does not contain any cycles, and a *forest* is a graph whose connected components are trees.

4.1.1 Markov Random Fields

A *Markov network* is an undirected graph $\mathscr{H} := (\mathcal{V}, \mathcal{E})$ whose nodes index a structured random variable $\boldsymbol{Y} := (Y_i)_{i \in \mathcal{V}}$ and whose cliques correspond to the factors of the distribution $P(\boldsymbol{Y})$. Formally, we have

$$\Pr\{\boldsymbol{Y} = \boldsymbol{y}\} \propto \prod_{\alpha \in \mathcal{C}(\mathscr{H})} \psi_\alpha(\boldsymbol{y}_\alpha), \tag{4.1}$$

where $\mathcal{C}(\mathscr{H})$ is a set of cliques of $\mathscr{H}$—not necessarily maximal—and $\psi_\alpha : \mathcal{Y}_\alpha \to \mathbb{R}_+$ are non-negative *potential functions*. A structured random variable $\boldsymbol{Y}$ which factors this way is called a *Markov random field* (MRF). In Eq. 4.1 and throughout, we use Greek subscripts $(\alpha, \beta, \gamma, \dots)$ to denote a subset of indices. For example, $\boldsymbol{Y}_\alpha := (Y_i)_{i \in \alpha}$, with $\alpha \subseteq \mathcal{V}$, denotes the tuple of variables that are indexed by the elements of α. As usual, uppercase letters denote random variables and lowercase letters denote values of those variables. We consider only discrete fields and assume that each Y_i takes values in a finite set $\mathcal{Y}_i$.[2] Each $\boldsymbol{Y}_\alpha$ takes values on the Cartesian product set $\mathcal{Y}_\alpha := \prod_{i \in \alpha} \mathcal{Y}_i$, and $\boldsymbol{Y}$ takes values on the full product set $\mathcal{Y} := \prod_{i \in \mathcal{V}} \mathcal{Y}_i$.

The reason why Markov networks are appealing representations is that many conditional independence properties can be immediately read from the graph $\mathscr{H}$. For example, every variable is conditionally independent from of all other variables given its neighbours (the *local Markov property*); and two subsets of variables are conditionally independent given a separating subset (the *global Markov property*).[3] We refer to Koller and Friedman (2009) for a thorough exposition of these and other properties of Markov networks.

Pairwise Markov networks. A *pairwise* MRF is one whose potentials only express up to pairwise interactions, *i.e.*, in which the cliques in $\mathcal{C}(\mathscr{H})$ are composed of nodes and edges. In this case, Eq. 4.1 becomes

$$P(y) \propto \prod_{i \in \mathcal{V}} \psi_i(y_i) \prod_{ij \in \mathcal{E}} \psi_{ij}(y_i, y_j). \tag{4.2}$$

Many models can be represented as pairwise Markov networks, most noticeably sequence models for the tasks described in Section 2.1, grid models for image segmentation, or graphs for relational learning. Some of these are illustrated in Figure 4.1. In fact, *any Markov network can be converted to an equivalent one which is pairwise*; see Wainwright and Jordan (2008, Appendix E) for a construction. While of great conceptual interest—as it allows to transfer

[2] Many formalisms discussed in this section extend naturally to discrete or continuous domains where $\mathcal{Y}_i$ are infinite sets; however these are out of the scope of this thesis.

[3] The converse statement—that a distribution satisfying these Markov properties must factorize according to Eq. 4.1—does not hold in general unless one makes additional assumptions, *e.g.*, that $P(\boldsymbol{Y})$ has full support. This important result is known as the Hammersley-Clifford theorem (Hammersley and Clifford, 1971; Besag, 1974). In this thesis we generally work with distributions that do not have full support (due to hard constraints), hence the theorem will not apply. This is not a reason for concern, however, as this converse statement is not needed in any place.

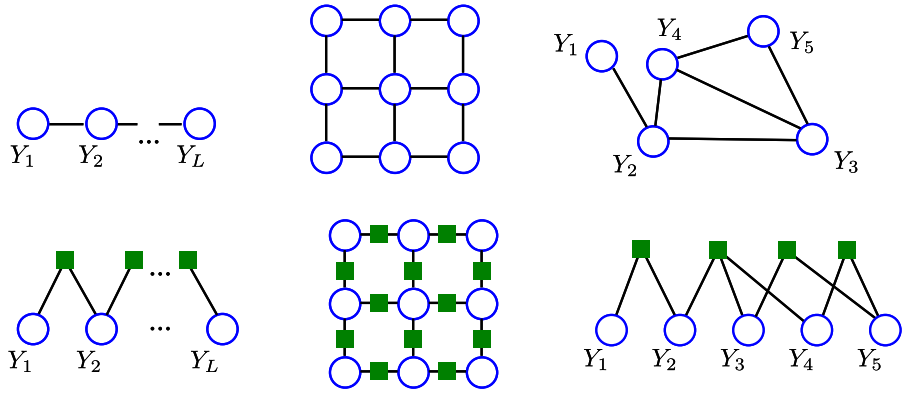


Figure 4.1: Top: several pairwise Markov random fields (a sequence, a grid, and an arbitrary graph). Bottom: factor graph representations for the same models.

properties of pairwise Markov networks to general networks—its practical consequences are limited by the fact that the construction may lead to a blow-up in the number of states.

4.1.2 Factor Graphs

An alternative representation of probability distributions is through *factor graphs* (Kschischang et al., 2001), which explicitly represent the factors of the distribution (*i.e.*, the actual cliques that are used in Eq. 4.1). A *factor graph* is a bipartite graph $\mathcal{G} := (\mathcal{V} \cup \mathcal{F}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{F}$ are disjoint sets of nodes (respectively called *variable nodes* and *factor nodes*) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{F}$ is a set of edges, each connecting a variable node to a factor node. We say that $P(\boldsymbol{Y})$ factors according to $\mathcal{G}$ if it has the form

$$P(\boldsymbol{y}) \propto \prod_{i \in \mathcal{V}} \psi_i(y_i) \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{y}_\alpha), \tag{4.3}$$

where each $\psi_i : \mathcal{Y}_i \to \mathbb{R}_+$ and $\psi_\alpha : \mathcal{Y}_\alpha \to \mathbb{R}_+$ are non-negative *potential functions*, respectively defined on the variable nodes and on the factor nodes.[4]

We use Latin letters $i, j, \ldots$ to denote variable nodes and Greek letters $\alpha, \beta, \ldots$ for factor nodes. For notational convenience, we often represent the latter by the set of indices of variable nodes that are linked to it (without loss of generality, we assume that there are no distinct factor nodes with the same neighborhood set). The bottom part of Figure 4.1 shows examples of factor graphs.

Given a factor graph $\mathcal{G}$ along with all the potential functions—which give a full specification of the distribution $P(\boldsymbol{Y})$—we want to solve *inference problems* in such a graph. The three tasks that will be our main concern throughout are:

1. **MAP inference.** Computing an assignment with maximal probability (called a *maxi-*

[4]Some authors omit the variable potential functions, writing the factorization above as $P(\boldsymbol{y}) \propto \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{y}_\alpha)$. That is not less general, as one can always define unary factors α_i such that $\mathcal{N}(\alpha_i) = \{i\}$ whose potentials play the same role as our variable potentials ($\psi_{\alpha_i} \equiv \psi_i$). We choose to explicitly use variable potentials for two reasons: (i) in most cases unary potentials are used anyways, hence we just avoid having to define unary factors all the time; (ii) in the sequel we will see that there is a certain "symmetry" between potentials and posterior marginals that is preserved if we consider variable potentials.

mum a posteriori or MAP assignment):

$$\widehat{\boldsymbol{y}} := \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}). \tag{4.4}$$

2. **Partition function evaluation.** Computing the normalizing factor of Eq. 4.3, which corresponds to evaluating the *partition function* Z:

$$Z(\boldsymbol{\psi}) := \sum_{y \in \mathcal{Y}} \left(\prod_{i \in \mathcal{V}} \psi_i(y_i) \prod_{\alpha \in \mathcal{F}} \psi_\alpha(\boldsymbol{y}_\alpha) \right) \tag{4.5}$$

3. **Marginal inference.** Computing posterior marginals for "local" variable assignments (by "local" we mean "at the variable and factor nodes of $\mathcal{G}$"):

$$\mu_i(y_i) := P(Y_i = y_i) \quad \text{and} \quad \mu_\alpha(\boldsymbol{y}_\alpha) := P(\boldsymbol{Y}_\alpha = \boldsymbol{y}_\alpha). \tag{4.6}$$

4.2 Message-Passing Algorithms

If $\mathcal{G}$ has no cycles (*i.e.*, if it is a tree or a forest), the three inference problems stated above can all be solved with dynamic programming. The MAP assignment can be obtained via the *max-product algorithm*, a generalization of the Viterbi algorithm for trees (Viterbi, 1967; Forney, 1973); the normalization factor and the marginals can be computed through the *sum-product algorithm*, which generalizes the forward-backward algorithm (Baum and Petrie, 1966).[5] In both cases, the runtime is linear in $|\mathcal{E}|$ and (in general) exponential in the largest factor degree.[6] These algorithms work by passing information through the graph, and are equivalent to the *belief propagation algorithm* of Pearl (1988)—which we next describe—for a particular scheduling of the messages.

The belief propagation (BP) algorithm iteratively passes messages between variables and factors reflecting their local "beliefs." In sum-product BP, the messages take the form:

$$\textbf{Variable to factor:} \qquad M_{i \to \alpha}(y_i) \propto \psi_i(y_i) \prod_{\substack{\beta \in \mathcal{N}(i) \\ \beta \neq \alpha}} M_{\beta \to i}(y_i) \tag{4.7}$$

$$\textbf{Factor to variable:} \qquad M_{\alpha \to i}(y_i) \propto \sum_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{\substack{j \in \mathcal{N}(\alpha) \\ j \neq i}} M_{j \to \alpha}(y_j). \tag{4.8}$$

In Eq. 4.8, we employ the standard $\sim$ notation, where a summation/maximization indexed by $\boldsymbol{y}_\alpha \sim y_i$ means that it is over all $\boldsymbol{y}_\alpha$ with the i-th component held fixed and set to y_i. The order by which the messages are applied is called its *schedule*. In the sequel, we assume a *flooding schedule*, according to which one iterates between computing all messages of the kind in Eq. 4.7, and then all messages of the kind in Eq. 4.8.

[5] Note that query-oriented algorithms such as the variable elimination algorithm (Zhang and Poole, 1994; Dechter, 1999) are not suitable for computing *all* marginals, since they are tailored to answer a single "query" at the time.

[6] In some sense, the two algorithms (max-product and sum-product) are isomorphic and are both special cases of the generalized sum-product algorithm for semirings (Shafer and Shenoy, 1990; Dawid, 1992; Goodman, 1999; Aji and McEliece, 2000; Mohri, 2002).

Upon convergence, variable and factor beliefs are computed as:

$$\textbf{Variable beliefs:} \quad b_i(y_i) \propto \psi_i(y_i) \prod_{\alpha \in \mathcal{N}(i)} M_{\alpha \to i}(y_i), \tag{4.9}$$

$$\textbf{Factor beliefs:} \quad b_\alpha(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} M_{i \to \alpha}(y_i). \tag{4.10}$$

If the BP algorithm attains a fixed point, *i.e.*, if all messages are left unchanged after completing a full round of the iterations (4.7–4.8), then the resulting beliefs must satisfy the following *calibration equations*, for every $i \in \mathcal{V}$ and $y_i \in \mathcal{Y}_i$:[7]

$$b_i(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} b_\alpha(\boldsymbol{y}_\alpha). \tag{4.11}$$

If $\mathcal{G}$ has no cycles, then BP always converges and these beliefs are precisely the posterior marginals in Eq. 4.6 (*i.e.*, we have $\mu_i(y_i) = b_i(y_i)$ and $\mu_\alpha(\boldsymbol{y}_\alpha) = b_\alpha(\boldsymbol{y}_\alpha)$). Given the marginals, we have the following reparametrization of the distribution $P(\boldsymbol{Y})$:

$$P(\boldsymbol{y}) = \frac{\prod_{\alpha \in \mathcal{F}} \mu_\alpha(\boldsymbol{y}_\alpha)}{\prod_{i \in \mathcal{V}} \mu_i(y_i)^{\deg(i)-1}}. \tag{4.12}$$

A distribution in the form (4.12) is said to be *calibrated*. Given a calibrated distribution, one can evaluate probabilities, and by choosing a particular assignment, it is straighforward to evaluate the partition function (Eq. 4.3).

In max-product BP, the summations in Eqs. 4.8 and 4.11 are replaced by maximizations,

$$M_{\alpha \to i}(y_i) \propto \max_{\boldsymbol{y}_\alpha \sim y_i} \left(\psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j \neq i} M_{j \to \alpha}(y_j) \right), \tag{4.13}$$

$$b_i(y_i) = \max_{\boldsymbol{y}_\alpha \sim y_i} b_\alpha(\boldsymbol{y}_\alpha); \tag{4.14}$$

and upon convergence, one can compute the exact MAP variable-wise from the beliefs (Eqs. 4.9–4.10) through:

$$\widehat{y}_i \propto \arg\max_{y_i \in \mathcal{Y}_i} b_i(y_i), \quad \forall i \in \mathcal{V}. \tag{4.15}$$

As mentioned above, the runtime of the belief propagation algorithm is exponential in the largest factor degree; this follows from observing that the computation of the factor-to-variable messages (Eqs. 4.8 and 4.13) involves (in general) enumerating all $|\mathcal{Y}_\alpha|$ configurations. Linearity in $|\mathcal{E}|$ comes from the fact that convergence is achieved after at most $|\mathcal{E}|$ message passing iterations.

4.2.1 Graphs with Cycles and Approximate Inference

The correctness of the BP algorithm falls apart when $\mathcal{G}$ has cycles, which leads to nuisance effects due to some messages being double counted during the updates. Exact inference is

[7] The proof is immediate, by observing that the messages and the beliefs are related through $M_{i \to \alpha}(y_i) \propto M_{\alpha \to i}(y_i)^{-1} b_i(y_i)$ and $M_{\alpha \to i}(y_i) \propto M_{i \to \alpha}(y_i)^{-1} \sum_{\boldsymbol{y}_\alpha \sim y_i} b_\alpha(\boldsymbol{y}_\alpha)$.

still possible via the *junction tree algorithm*, which constructs a new graph from $\mathcal{G}$ (called a *junction tree*) by collapsing some of the nodes into supernodes, and then runs the belief propagation algorithm on the junction tree (see Lauritzen 1996; Koller and Friedman 2009 for details). This however leads to a runtime which grows exponentially with a parameter called the *treewidth* of $\mathcal{G}$,[8] rendering the procedure intractable for graphs with a large treewidth. This should come as no surprise: it can be shown that exact MAP inference in a general graph (even one with only pairwise interactions) is NP-complete, and computing the normalization factor is #P-hard. For details and pointers to the literature, see, *e.g.*, Sontag (2010, Chapter 2).

A variety of approximate inference methods have been proposed: some based on sampling (Geman and Geman, 1984; Marroquin et al., 1987; Gelfand and Smith, 1990; Casella and Robert, 1999; Finkel et al., 2005), others on approximate search (Zhang and Clark, 2008; Daumé et al., 2009), yet others on variational inference (see Wainwright and Jordan 2008 and references therein). In this thesis, we focus on the last kind. One instance is the *loopy belief propagation algorithm* (loopy BP), which is exactly what we have described, but applied to a general graph $\mathcal{G}$ with cycles. In other words, loopy BP "pretends" that $\mathcal{G}$ is a tree by ignoring the global effects caused by the cycles. Of course, as would be expected, it is not guaranteed to compute the exact quantities, or even to converge.[9] Nevertheless, it often works well in practice for some graph topologies.

4.3 The Geometry of Graphical Models

Inference in graphical models is better understood by characterizing their *geometry*. We next highlight a few crucial geometric aspects, some of which inspire approximate inference algorithms in loopy graphs, as we shall see. The two main ingredients are:

- The *marginal polytope*, which characterizes the structure of the graphical model;
- A duality relationship of exponential families (*Legendre-Fenchel duality*) which allows expressing marginal inference as a variational problem.

In this section, we briefly review the main concepts. Additional details can be found in the monograph by Wainwright and Jordan (2008).

4.3.1 Graphical Models as Exponential Families

An *exponential family*[10] $\mathcal{E} := \{P_{\boldsymbol{\theta}}(.) \mid \boldsymbol{\theta} \in \mathbb{R}^R\}$ is a family of distributions, each of the following form:

$$P_{\boldsymbol{\theta}}(y) \;=\; \frac{1}{Z(\boldsymbol{\theta})} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(y)) Q(y), \tag{4.16}$$

where $\boldsymbol{\theta} \in \mathbb{R}^R$ is a *parameter vector*, $\boldsymbol{\phi}(y) \in \mathbb{R}^R$ is a vector of *sufficient statistics* (or *features*), $Z : \mathbb{R}^R \to \mathbb{R}_+$ is the *partition function*, and Q is a base measure on $\mathcal{Y}$. In this section, we

[8] The treewidth is the size of the largest clique of the triangulated Markov network representation of $\mathcal{G}$, minus one. See Lauritzen (1996) or Koller and Friedman (2009) for details.

[9] Although it is provably correct or at least convergent for some special classes of graphs which are not trees: see Weiss and Freeman (2001b,a); Bayati et al. (2005); Huang and Jebara (2007).

[10] Also called a *family of Gibbs distributions* in statistical physics, and a *family of log-linear distributions* in natural language processing and machine learning.

assume that Q is a constant-valued function ($Q \equiv 1$ without loss of generality), whence it can be dropped from Eq. 4.16. The partition function takes the form:

$$Z(\boldsymbol{\theta}) := \sum_{y \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(y)). \tag{4.17}$$

Exponential families are extremely general; their asymptotic and geometric properties have been widely studied in statistics (Darmois, 1935; Pitman, 1936; Koopman, 1936) and information geometry (Murray and Rice, 1993; Amari and Nagaoka, 2001).

Factor graphs with strictly positive potentials can be regarded as exponential families. Indeed, we can let the parameters $\boldsymbol{\theta}$ be the *log-potentials*,

$$\theta_i(y_i) := \log \psi_i(y_i), \qquad \theta_\alpha(\boldsymbol{y}_\alpha) := \log \psi_\alpha(\boldsymbol{y}_\alpha), \tag{4.18}$$

and define the sufficient statistics $\boldsymbol{\phi}(\boldsymbol{y}) := \boldsymbol{\chi}(\boldsymbol{y})$ as the indicator vector of local configurations (on variable and factor nodes) induced by the output $\boldsymbol{y} \in \mathcal{Y}$:

$$\chi_{i,y_i'}(\boldsymbol{y}) := [\![y_i = y_i']\!], \qquad \chi_{\alpha,\boldsymbol{y}_\alpha'}(\boldsymbol{y}) := [\![\boldsymbol{y}_\alpha = \boldsymbol{y}_\alpha']\!]. \tag{4.19}$$

The dimension of the vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}(\boldsymbol{y})$ is $R := \sum_{i \in \mathcal{V}} |\mathcal{Y}_i| + \sum_{\alpha \in \mathcal{F}} |\mathcal{Y}_\alpha|$. We write Eq. 4.16 as

$$P_{\boldsymbol{\theta}}(\boldsymbol{y}) \;=\; \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{\alpha \in \mathcal{F}} \theta_\alpha(\boldsymbol{y}_\alpha) \right), \tag{4.20}$$

and Eq. 4.17 as

$$Z(\boldsymbol{\theta}) := \sum_{y \in \mathcal{Y}} \exp\left(\sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{\alpha \in \mathcal{F}} \theta_\alpha(\boldsymbol{y}_\alpha) \right). \tag{4.21}$$

This is equivalent to Eq. 4.3 when all potentials are strictly positive. This parametrization of a factor graph, through $\boldsymbol{\theta}$, is called the *canonical overcomplete parametrization* (Wainwright and Jordan, 2008).[11]

Example 4.1 (Pairwise MRFs: Boltzmann, Ising, and Potts models.) *The canonical overcomplete parametrization of a pairwise MRF is*

$$P_{\boldsymbol{\theta}}(\boldsymbol{y}) \;\propto\; \exp\left(\sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{ij \in \mathcal{E}} \theta_{ij}(y_i, y_j) \right). \tag{4.22}$$

A binary pairwise MRF *is one in which all variables are binary, $\mathcal{Y}_i := \{0, 1\}$. Such models are also called* Ising models.[12] *In that case, there are $2|\mathcal{V}| + 4|\mathcal{E}|$ parameters in Eq. 4.22. The same family*

[11] The name "overcomplete" comes from the fact that the map $\boldsymbol{\theta} \mapsto P_{\boldsymbol{\theta}}$ is not injective: in fact, if we define $\boldsymbol{\theta}'$ such that $\theta_i'(y_i) = \theta_i(y_i) + c_i$ and $\theta_\alpha'(\boldsymbol{y}_\alpha) = \theta_\alpha(\boldsymbol{y}_\alpha) + c_\alpha$, where c_i and c_α are arbitrary real constants, we have that $\boldsymbol{\theta}'$ and $\boldsymbol{\theta}$ parameterize the same distribution.

[12] Named after Ernst Ising (1900–1998), who first proposed these models in statistical physics to model interaction between particles (Ising, 1925). These models are also called *Boltzmann machine models* in the artificial neural networks literature (Ackley et al., 1985).

can be parameterized with only $|\mathcal{V}| + |\mathcal{E}|$ *parameters through:*

$$P_{\boldsymbol{s}}(\boldsymbol{y}) \propto \exp\left(\sum_{i\in\mathcal{V}} s_i y_i + \sum_{ij\in\mathcal{E}} s_{ij} y_i y_j\right). \tag{4.23}$$

This reparametrizarion does not lose expressive power. Given $\boldsymbol{s}$*, we can define* $\boldsymbol{\theta}$ *as*

$$\theta_i(1) := s_i, \quad \theta_i(0) := 0, \qquad \theta_{ij}(1,1) := s_{ij}, \quad \theta_{ij}(0,0) = \theta_{ij}(0,1) = \theta_{ij}(1,0) := 0, \tag{4.24}$$

and vice-versa:

$$s_i = \theta_i(1) - \theta_i(0) + \sum_{j\in N(i)} (\theta_{ij}(1,0) - \theta_{ij}(0,0)), \quad s_{ij} = \theta_{ij}(1,1) - \theta_{ij}(0,1) - \theta_{ij}(1,0) + \theta_{ij}(0,0). \tag{4.25}$$

Of particular interest are ferromagnetic Ising models*, in which all edge couplings are "attractive"* (i.e., $s_{ij} \geq 0$ *for all* $ij \in \mathcal{E}$)*.*[13] *In such models, neighboring nodes "prefer" to be in the same state. Ferromagnetic Ising models permit tractable MAP inference through graph cut algorithms (Greig et al., 1989); in general Ising models, MAP inference is NP-hard. A* Potts model *is a generalization of an Ising model for* $K > 2$ *states,* i.e., $\mathcal{Y}_i := \{1, \ldots, K\}$ *for every* $i \in \mathcal{V}$*. Potts models are extensively used in computer vision (for image segmentation and stereo vision problems) and relational network modeling. It is common to define* attractive *log-potentials as* $\theta_{ij}(y_i, y_j) := \alpha_{ij} \times [\![y_i = y_j]\!]$*, where* $\alpha_{ij} \geq 0$ *is a constant, which corresponds to the assumption that neighbors prefer to be in the same state. Markov networks with these property are called* associative *(Taskar et al., 2004a).*

Example 4.2 (Conditional random fields.) *We have described CRFs in Section 3.4.2, defined via a set of places* $\mathcal{P}$*, local feature vectors* $\boldsymbol{f}_p(x, y_p) \in \mathbb{R}^D$ *and a weight vector* $\boldsymbol{w} \in \mathbb{R}^D$*:*

$$P_{\boldsymbol{w}}(y|x) := \frac{1}{Z(\boldsymbol{w}, x)} \exp\left(\sum_{p\in\mathcal{P}} \boldsymbol{w} \cdot \boldsymbol{f}_p(x, y_p)\right). \tag{4.26}$$

We can regard the "places" as the variable and factor nodes in a factor graph $\mathcal{G} := (\mathcal{V}, \mathcal{F})$*, in which case we may rewrite Eq. 4.26 as*

$$P_{\boldsymbol{w}}(y|x) := \frac{1}{Z(\boldsymbol{w}, x)} \exp\left(\sum_{i\in\mathcal{V}} \boldsymbol{w} \cdot \boldsymbol{f}_i(x, y_i) + \sum_{\alpha\in\mathcal{F}} \boldsymbol{w} \cdot \boldsymbol{f}_\alpha(x, \boldsymbol{y}_\alpha)\right). \tag{4.27}$$

This is a discriminative linear model in which the joint feature map decomposes as

$$\boldsymbol{f}(x, y) := \sum_{i\in\mathcal{V}} \boldsymbol{f}_i(x, y_i) + \sum_{\alpha\in\mathcal{F}} \boldsymbol{f}_\alpha(x, y_\alpha). \tag{4.28}$$

Note that Eq. 4.28 allows features to be shared across variables and factors. For each $x \in \mathcal{X}$*, such models form an exponential family* $\mathscr{E}(x)$ *parameterized by* $\boldsymbol{w} \in \mathbb{R}^D$*. This family is a subset of the canonically parameterized family in Eq. 4.20: there is an affine transformation that transforms the feature weights* $\boldsymbol{w} \in \mathbb{R}^D$ *into canonical overcomplete parameters* $\boldsymbol{\theta} \in \mathbb{R}^R$*:*

$$\theta_i(y_i) = \boldsymbol{w} \cdot \boldsymbol{f}_i(x, y_i), \qquad \theta_\alpha(\boldsymbol{y}_\alpha) = \boldsymbol{w} \cdot \boldsymbol{f}_\alpha(x, \boldsymbol{y}_\alpha). \tag{4.29}$$

[13] Equivalently, the potentials are "submodular," as commonly refered in the computer vision literature.

We can write this compactly as $\boldsymbol{\theta} = \mathbf{F}(x)^\top \boldsymbol{w}$*, where* $\mathbf{F}(x)$ *is a D-by-R feature matrix, whose columns are the local feature vectors* $\boldsymbol{f}_i(x, y_i)$ *and* $\boldsymbol{f}_\alpha(x, \boldsymbol{y}_\alpha)$*.*

4.3.2 Mean Parametrization and Marginal Polytope

Consider a particular distribution $P_{\boldsymbol{\theta}} \in \mathscr{E}$, and its vector of *expected sufficient statistics* $\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}(\boldsymbol{Y})]$.[14] A fundamental property of exponential families is that the gradient of the log-partition function equals that quantity:

$$\nabla_{\boldsymbol{\theta}} \log Z(\boldsymbol{\theta}) = \frac{\nabla_{\boldsymbol{\theta}} Z(\boldsymbol{\theta})}{Z(\boldsymbol{\theta})} = \frac{\sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{y})) \boldsymbol{\phi}(\boldsymbol{y})}{Z(\boldsymbol{\theta})} = \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}(\boldsymbol{Y})]. \tag{4.30}$$

Recall that with a canonical overcomplete parametrization, the sufficient statistics are simply indicator functions of variable and factor configurations, *i.e.*, we have $\boldsymbol{\phi}(\boldsymbol{y}) := \boldsymbol{\chi}(\boldsymbol{y})$ (cf. Eq. 4.19); hence the expected sufficient statistics are precisely the *posterior marginals on local configurations* (cf. 4.6), *i.e.*, we have

$$\mu_i(y_i) = \mathbb{E}_{\boldsymbol{\theta}}[\chi_{i,y_i}(\boldsymbol{Y})], \qquad \mu_\alpha(\boldsymbol{y}_\alpha) = \mathbb{E}_{\boldsymbol{\theta}}[\chi_{\alpha,\boldsymbol{y}_\alpha}(\boldsymbol{Y})]. \tag{4.31}$$

We stack these marginals together into a *marginal vector* $\boldsymbol{\mu} := \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\chi}(\boldsymbol{Y})] \in \mathbb{R}^R$. This yields an alternative parametrization of the exponential family $\mathscr{E}$, called the *mean parameterization*. To make this formal, we need to specify the range of these parameters, *i.e.*, the set of marginals which are *realizable* by some arbitrary distribution $P(\boldsymbol{Y})$:

$$\begin{aligned} \mathrm{MARG}(\mathscr{G}) : &= \left\{ \boldsymbol{\mu} \in \mathbb{R}^R \mid \exists P(\boldsymbol{Y}) \text{ s.t. } \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{Y} \sim P(\boldsymbol{Y})}[\boldsymbol{\chi}(\boldsymbol{Y})] \right\} \\ &= \mathrm{conv}\{\boldsymbol{\chi}(\boldsymbol{y}) \mid \boldsymbol{y} \in \mathcal{Y}\}. \end{aligned} \tag{4.32}$$

The second equality is by definition of convex hull (cf. Appendix B), since a well-defined distribution $P(\boldsymbol{Y})$ is one which satisfies $\sum_{\boldsymbol{y}} P(\boldsymbol{y}) = 1$ and $P(\boldsymbol{y}) \geq 0$ for every $\boldsymbol{y} \in \mathcal{Y}$. Being the convex hull of a finite set, $\mathrm{MARG}(\mathscr{G})$ is a polytope—it is called the *marginal polytope* of $\mathscr{G}$ (Wainwright and Jordan, 2008). The marginal polytope has two important properties:

1. Each vertex of $\mathrm{MARG}(\mathscr{G})$ corresponds to an output $\boldsymbol{y} \in \mathcal{Y}$;

2. Each point in $\mathrm{MARG}(\mathscr{G})$ corresponds to a vector of marginal probabilities that is realizable by some distribution.

This is represented schematically in Figure 4.2. By definition of marginal vector, we have that the map $\boldsymbol{\theta} \mapsto \boldsymbol{\mu}$ is well defined (*i.e.*, for every $\boldsymbol{\theta} \in \mathbb{R}^R$ there is one and only one $\boldsymbol{\mu}(\boldsymbol{\theta}) \in \mathrm{MARG}(\mathscr{G})$); from Eq. 4.30, we have that it is precisely the gradient map $\nabla \log Z : \mathbb{R}^M \to \mathrm{MARG}(\mathscr{G})$. Note however that this map is not injective, since the $\boldsymbol{\theta}$-parametrization is overcomplete. Yet, there is a bijection between $\mathscr{E}$ and the relative interior[15] of the marginal polytope, as the next proposition asserts.[16]

[14] We denote $\mathbb{E}_{\boldsymbol{\theta}} \equiv \mathbb{E}_{\boldsymbol{Y} \sim P_{\boldsymbol{\theta}}(\boldsymbol{Y})}$ to make explicit the dependency on the parameters $\boldsymbol{\theta}$.

[15] See Appendix B for the definition of *relative interior*.

[16] Theorem 3.3 in Wainwright and Jordan (2008) is stated for a minimal parametrization; however they discuss its extension to the overcomplete case in their Appendix B.1.

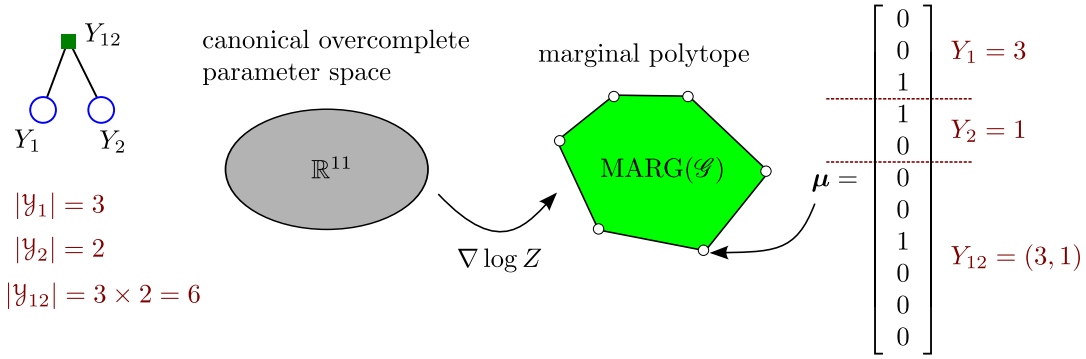


Figure 4.2: Representation of the marginal polytope for a simple factor graph connecting two nodes and one factor. Each point in the polytope is a realizable marginal vector, and each vertex correspond to a possible assignment; we illustrate, on the right, one of those vertices, corresponding to the configuration $Y_1 = 3$ and $Y_2 = 1$. The gradient map $\nabla \log Z$ maps the canonical overcomplete parameters to the mean parameters.

Proposition 4.1 (Wainwright and Jordan 2008, Th. 3.3) *Let $\boldsymbol{\mu} \in \text{relint}(\text{MARG}(\mathscr{G}))$. Then there exists some $\boldsymbol{\theta} \in \mathbb{R}^R$ such that $\boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\chi}(\boldsymbol{Y})]$. Furthermore, all such $\boldsymbol{\theta}$ are equivalent in the sense that they all parameterize the same distribution $P_{\boldsymbol{\theta}}(\boldsymbol{Y})$.*

4.3.3 MAP Inference as a Linear Program

Given a distribution $P_{\boldsymbol{\theta}} \in \mathscr{E}$, MAP inference (Eq. 4.4) is a combinatorial optimization problem which can be written as:

$$\begin{array}{ll} \text{maximize} & \sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{\alpha \in \mathcal{F}} \theta_\alpha(\boldsymbol{y}_\alpha) \\ \text{w.r.t.} & \boldsymbol{y} \in \mathcal{Y}. \end{array} \tag{4.33}$$

Let $\boldsymbol{\theta} \cdot \boldsymbol{\mu} = \sum_{i \in \mathcal{V}} \sum_{y_i \in \mathcal{Y}_i} \theta_i(y_i)\mu_i(y_i) + \sum_{\alpha \in \mathcal{F}} \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha)\mu_\alpha(\boldsymbol{y}_\alpha)$. Given that there is a one-to-one correspondence between the vertices of MARG($\mathscr{G}$) and the elements in $\mathcal{Y}$, we can transform Eq. 4.33 into the following linear program:

$$\boxed{\begin{array}{lll} \textbf{MAP:} & \text{maximize} & \boldsymbol{\theta} \cdot \boldsymbol{\mu} \\ & \text{w.r.t.} & \boldsymbol{\mu} \in \text{MARG}(\mathscr{G}). \end{array}} \tag{4.34}$$

The equivalence between the two formulations is due to the fact that any linear program whose constraint set is non-empty and bounded—which is the case of MARG($\mathscr{G}$)—has a solution which is attained at a vertex. Unfortunately, for a general graph $\mathscr{G}$, reformulating the non-convex combinatorial problem in Eq. 4.33 as the continuous and convex one in Eq. 4.34 does not make it easier. Although a fundamental result in convex analysis, the Minkowsky-Weyl theorem (Rockafellar, 1970), guarantees that the marginal polytope MARG($\mathscr{G}$) has a representation in terms of a finite number of linear inequalities, it is widely believed that this number grows superpolynomially with the size of the graph $\mathscr{G}$.[17]

[17] If this statement turns out to be false, it will rule out important conjectures in complexity theory, such as $P \neq NP$. To see this, note that the well-known problem of finding a maximum cut in a graph is equivalent to MAP inference in a Ising model, and hence can be reduced to the linear program in Eq. 4.34. If there was a

Yet, the linear programming formulation in Eq. 4.34 is very useful, since it sheds light on the MAP inference problem, and opens up ways of devising *approximate* inference algorithms by considering approximations of $\mathrm{MARG}(\mathscr{G})$. We will see specific examples later on.

4.3.4 Marginal Inference and Conjugate Duality

We now turn to the problem of computing the marginals and evaluating the partition function. The relationship between the two had appeared in Eq. 4.30, which expressed the marginals as the gradient of the log-partition function. This is part of a bigger picture that involves an important form of duality (called *Legendre-Fenchel duality*). To display the whole picture, we need to introduce another ingredient, the Shannon entropy of a distribution:

$$H(P(\boldsymbol{Y})) := \mathbb{E}_{\boldsymbol{Y}}[-\log P(\boldsymbol{Y})]. \tag{4.35}$$

From Proposition 4.1, we have a bijection $P(\boldsymbol{Y}) \mapsto \boldsymbol{\mu}$, and therefore we may express the entropy in terms of the $\boldsymbol{\mu}$-parameters, $H : \mathrm{MARG}(\mathscr{G}) \to \mathbb{R}$, where we write (abusing notation) $H(\boldsymbol{\mu})$ to denote the entropy of the distribution $P(\boldsymbol{Y})$ which is parameterized by $\boldsymbol{\mu}$.

Proposition 4.2 (Wainwright and Jordan 2008, Th. 3.4) *The log-partition function,* $\log Z : \mathbb{R}^R \to \mathbb{R}$*, and the negative entropy,* $-H : \mathrm{MARG}(\mathscr{G}) \to \mathbb{R}$*, are conjugate dual of each other. As a consequence, the following variational representation for the log-partition function holds:*

$$\log Z(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \mathrm{MARG}(\mathscr{G})} \boldsymbol{\theta} \cdot \boldsymbol{\mu} + H(\boldsymbol{\mu}). \tag{4.36}$$

Furthermore, the solution of Eq. 4.36 is attained at $\boldsymbol{\mu}(\boldsymbol{\theta})$*,* i.e.*, the marginal vector corresponding to the distribution* $P_{\boldsymbol{\theta}}$*. Conversely, the entropy function also has a variational representation:*

$$-H(\boldsymbol{\mu}) = \max_{\boldsymbol{\theta} \in \mathbb{R}^R} \boldsymbol{\theta} \cdot \boldsymbol{\mu} - \log Z(\boldsymbol{\theta}), \tag{4.37}$$

with a solution being attained for any $\boldsymbol{\theta}$ *such that* $\boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{\theta})$*. Therefore:*

$$H(P_{\boldsymbol{\theta}}) = -\boldsymbol{\theta} \cdot \boldsymbol{\mu}(\boldsymbol{\theta}) + \log Z(\boldsymbol{\theta}), \tag{4.38}$$

for any $\boldsymbol{\theta}$ *that parameterizes* $P_{\boldsymbol{\theta}}$*.*

The duality expressed in Proposition 4.2 is intimately related with the well-known equivalence between conditional log-likelihood maximization in logistic regression models and entropy maximization subject to first moment constraints. Note also that by applying Danskin's theorem to Eq. 4.36, we recover the fact that $\nabla \log Z(\boldsymbol{\theta}) = \boldsymbol{\mu}(\boldsymbol{\theta})$.

Like in the case of MAP inference, Eq. 4.36 allows us to formulate marginal inference as a convex optimization problem over the marginal polytope:

$$\boxed{\begin{array}{lll} \textbf{Marginals:} & \text{maximize} & \boldsymbol{\theta} \cdot \boldsymbol{\mu} + H(\boldsymbol{\mu}) \\ & \text{w.r.t.} & \boldsymbol{\mu} \in \mathrm{MARG}(\mathscr{G}). \end{array}} \tag{4.39}$$

polynomial number of constraints in Eq. 4.34, then the linear program (and hence max-cut) would be solvable in polynomial time.

Note that the only difference with respect to the MAP inference case (Eq. 4.34) is the inclusion of the entropic term $H(\boldsymbol{\mu})$ in the objective, which breaks the linearity. Albeit non-linear, this term is convex, hence Eq. 4.34 is still a convex program. But the entropic term introduces another complication, which adds to the already known difficulty in characterizing the marginal polytope $\text{MARG}(\mathcal{G})$: when expressed in terms of $\boldsymbol{\mu}$, the entropy lacks a closed form (in general) and is difficult to characterize.

4.4 Local Polytope Approximation

We mentioned in Section 4.3.3 that, for a general graph $\mathcal{G}$ with cycles, the marginal polytope $\text{MARG}(\mathcal{G})$ is difficult to characterize as a set of linear inequalities. The reason is that the cycles induce *global consistency constraints* which are hard to specify. There are, however, weaker constraints that all realizable marginals must satisfy to ensure *local consistency*:

1. **Non-negativity.** All local marginals $\mu_i(y_i)$ and $\mu_\alpha(\boldsymbol{y}_\alpha)$ must be non-negative.

2. **Normalization.** We must have $\sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) = 1$ for every $i \in \mathcal{V}$, and $\sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \mu_\alpha(\boldsymbol{y}_\alpha) = 1$ for every $\alpha \in \mathcal{F}$, to ensure that at a local level all marginals normalize properly.

3. **Marginalization.** A variable participating in a factor must have a marginal which is consistent with the factor marginals, *i.e.*, we must have $\mu_i(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha)$ for every $i \in \mathcal{V}$, $y_i \in \mathcal{Y}_i$ and $\alpha \in \mathcal{N}(i)$.

Putting everything together, we form the so-called *local polytope*:

$$\text{LOCAL}(\mathcal{G}) = \left\{ \boldsymbol{\mu} \in \mathbb{R}^R \;\middle|\; \begin{array}{ll} \sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) = 1, & \forall i \in \mathcal{V} \\ \mu_i(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha), & \forall i \in \mathcal{V},\ y_i \in \mathcal{Y}_i,\ \alpha \in \mathcal{N}(i) \\ \mu_\alpha(\boldsymbol{y}_\alpha) \geq 0, & \forall \alpha \in \mathcal{F},\ \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha \end{array} \right\}. \tag{4.40}$$

The number of constraints that define $\text{LOCAL}(\mathcal{G})$ is *linear* in R, rather than superpolynomial.The elements of $\text{LOCAL}(\mathcal{G})$ are called *pseudo-marginals*. The "local" character of $\text{LOCAL}(\mathcal{G})$ may be emphasized by noting that it can be equivalently expressed as the intersection of smaller marginal polytopes, "lifted"[18] to the ambient space $\mathbb{R}^R$:

$$\text{LOCAL}(\mathcal{G}) := \left\{ \boldsymbol{\mu} \in \mathbb{R}^R \;\middle|\; \boldsymbol{\mu}|_\alpha \in \text{MARG}(\mathcal{G}|_\alpha) \quad \text{for every } \alpha \in \mathcal{F} \right\}, \tag{4.41}$$

where we denote by $\mathcal{G}|_\alpha$ the subgraph formed by a single factor α and their neighbors, and by $\boldsymbol{\mu}|_\alpha$ the restriction of $\boldsymbol{\mu}$ to the entries $\mu_\alpha(.)$ and $\mu_i(.)$ for every $i \in \mathcal{N}(\alpha)$. Since any true marginal vector must satisfy the constraints above, $\text{LOCAL}(\mathcal{G})$ is an *outer approximation*:

$$\text{MARG}(\mathcal{G}) \subseteq \text{LOCAL}(\mathcal{G}). \tag{4.42}$$

The next proposition states that the approximation is tight for tree-structured graphs.

Proposition 4.3 *If $\mathcal{G}$ does not have cycles, then* $\text{MARG}(\mathcal{G}) = \text{LOCAL}(\mathcal{G})$.

[18] See Appendix B for a definition of *lifting*.

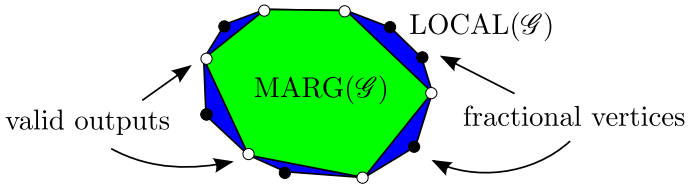


Figure 4.3: Marginal polytope (in green) and its outer aproximation, the local polytope (in blue). Each element of the marginal polytope corresponds to a distribution $P(\boldsymbol{Y})$, and each vertex corresponds to a configuration $y \in \mathcal{Y}$, having coordinates in $\{0,1\}$. The local polytope may have additional fractional vertices, with coordinates in $[0,1]$.

Proof sketch. It suffices to show that $\text{LOCAL}(\mathscr{G}) \subseteq \text{MARG}(\mathscr{G})$. Given $\boldsymbol{\mu} \in \text{LOCAL}(\mathscr{G})$, define $P(\boldsymbol{Y})$ through Eq. 4.12; then show that marginalizing $P(\boldsymbol{Y})$ over variables and factors recovers $\boldsymbol{\mu}$, *i.e.*, that $\sum_{\boldsymbol{y} \sim y_i} P(\boldsymbol{y}) = \mu_i(y_i)$ and $\sum_{\boldsymbol{y} \sim \boldsymbol{y}_\alpha} P(\boldsymbol{y}) = \mu_\alpha(\boldsymbol{y}_\alpha)$. This follows from the marginalization constraints that define $\text{LOCAL}(\mathscr{G})$, along with the fact that for any acyclic $\mathscr{G}$, no two distinct factors α and β share more than one variable (which would cause a cycle).

Unfortunately, the inclusion $\text{MARG}(\mathscr{G}) \subseteq \text{LOCAL}(\mathscr{G})$ is proper even for very small graphs with cycles (see *e.g.* Wainwright and Jordan 2008, Example 4.1). All we have is:

Proposition 4.4 *The integral points of* $\text{LOCAL}(\mathscr{G})$ *are the vertices of* $\text{MARG}(\mathscr{G})$. *Hence:*

$$\text{MARG}(\mathscr{G}) = \text{conv}(\text{LOCAL}(\mathscr{G}) \cap \mathbb{Z}^R). \tag{4.43}$$

Proof. Since $\text{LOCAL}(\mathscr{G})$ is contained in the unit cube $[0,1]^R$, its integral points are binary-valued. From the normalization constraints, each integral point $\boldsymbol{\mu} \in \text{LOCAL}(\mathscr{G})$ must indicate a single assignment at each factor and variable node, and the marginalization constraints force these to agree. Hence $\boldsymbol{\mu} = \boldsymbol{\chi}(\boldsymbol{y})$ for some $\boldsymbol{y} \in \mathcal{Y}$, and therefore we have $\{\boldsymbol{\chi}(\boldsymbol{y}) \mid \boldsymbol{y} \in \mathcal{Y}\} = \text{LOCAL}(\mathscr{G}) \cap \mathbb{Z}^R$. The result follows from taking the convex hull on both sides. ∎

We depict in Figure 4.3 a schematic representation of the local polytope. We will see in the next two sections that this approximation plays a crucial role in approximate inference algorithms, both for marginal inference (Section 4.5) and for MAP inference (Section 4.6).

4.5 Bethe's Variational Principle and Loopy Belief Propagation

We now describe how the loopy BP algorithm is related to an approximation of the variational expression of Eq. 4.39. For convenience, we will use in the sequel the following vector notation for the local marginals,

$$\boldsymbol{\mu}_i := (\mu_i(y_i))_{y_i \in \mathcal{Y}_i}, \quad \boldsymbol{\mu}_\alpha := (\mu_\alpha(\boldsymbol{y}_\alpha))_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha}. \tag{4.44}$$

As mentioned above, for a general graph $\mathscr{G}$, the entropy $H : \text{MARG}(\mathscr{G}) \to \mathbb{R}$ lacks a closed form. A popular approximation is the so-called *Bethe entropy* (Bethe, 1935),[19] which we

[19] Named after the physicist Hans Bethe (1906–2005).

denote by $H_{\text{Bethe}} : \text{LOCAL}(\mathcal{G}) \to \mathbb{R}$:

$$H_{\text{Bethe}}(\boldsymbol{\mu}) := \sum_{i \in \mathcal{V}} (1 - \deg(i)) H_i(\boldsymbol{\mu}_i) + \sum_{\alpha \in \mathcal{F}} H_\alpha(\boldsymbol{\mu}_\alpha), \tag{4.45}$$

where $H_i(\boldsymbol{\mu}_i) := -\sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) \log \mu_i(y_i)$ and $H_\alpha(\boldsymbol{\mu}_\alpha) := -\sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \mu_\alpha(\boldsymbol{y}_\alpha) \log \mu_\alpha(\boldsymbol{y}_\alpha)$ are *local entropies* (at the variable and the factor nodes, respectively).[20] The next proposition states that this approximation is exact for tree-structured graphs.

Proposition 4.5 *If $\mathcal{G}$ has no cycles, then $H = H_{\text{Bethe}}$.*

Proof. From Proposition 4.3, we have that $\text{MARG}(\mathcal{G}) = \text{LOCAL}(\mathcal{G})$, hence H and H_{Bethe} have the same domain. That $H(\boldsymbol{\mu})$ equals the right hand side of Eq. 4.45 is an immediate consequence of the reparametrization of the distribution in Eq. 4.12. ∎

For a general graph $\mathcal{G}$, Bethe's entropy approximation is inexact and neither lower or upper bounds the true entropy. Furthermore, H_{Bethe} is not concave in general (unlike H). The quality of the approximation strongly depends on the topology of the graph. While in many practical problems the two functions are "close" enough, we will see some disappointing examples in Chapter 5 where H_{Bethe} is a very poor approximation.

To summarize, we have discussed two approximations so far; both ignore global effects caused by cycles and both are tight when $\mathcal{G}$ is acyclic:

- The marginal polytope $\text{MARG}(\mathcal{G})$ is outer approximated by $\text{LOCAL}(\mathcal{G})$;
- The entropy H is approximated by the Bethe "entropy" H_{Bethe}.

Putting these two pieces together, we obtain the following variational approximation for the log-partition function (cf. Eq. 4.36):

$$\log Z(\boldsymbol{\theta}) \approx \max_{\boldsymbol{\mu} \in \text{LOCAL}(\mathcal{G})} \boldsymbol{\theta} \cdot \boldsymbol{\mu} + H_{\text{Bethe}}(\boldsymbol{\mu}). \tag{4.47}$$

The maximizer of Eq. 4.47 (call it $\tilde{\boldsymbol{\mu}}(\boldsymbol{\theta})$) can be regarded as a surrogate for the true marginals, and hopefully $\tilde{\boldsymbol{\mu}}(\boldsymbol{\theta}) \approx \boldsymbol{\mu}(\boldsymbol{\theta})$. The following important result, established by Yedidia et al. (2001), relates the loopy BP algorithm with the Bethe variational problem.

Theorem 4.6 (Yedidia et al. (2001)) *Any stationary point resulting from the updates in the sum-product loopy BP algorithm corresponds to a local optimum of the problem in Eq. 4.47.*

We point to Yedidia et al. (2001) for a formal proof. Essentially, the messages in loopy BP correspond to Lagrange multipliers in a dual formulation of the problem in Eq. 4.47, and the vector of beliefs locally optimizes Eq. 4.47. However, the sum-product loopy BP algorithm may not converge, and the local optimum may not be a global one—this is because the Bethe variational problem is in general nonconvex, due to the nonconcavity of H_{Bethe}.

[20] To better understand this approximation, observe that for a pairwise MRF, H_{Bethe} can be rewritten as:

$$H_{\text{Bethe}}(\boldsymbol{\mu}) = \sum_{i \in \mathcal{V}} H_i(\boldsymbol{\mu}_i) - \sum_{ij \in \mathcal{F}} I_{ij}(\boldsymbol{\mu}_{ij}), \tag{4.46}$$

where $I_{ij}(\boldsymbol{\mu}_{ij}) := \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \mu_{ij}(y_i, y_j) \log \frac{\mu_{ij}}{\mu_i(y_i)\mu_j(y_j)}$ is the *mutual information* between each pair of variables. In words, the Bethe entropy approximation equals the sum of the node entropies (as if the variables were all independent), discounted by the pairwise mutual informations.

Generalized message-passing algorithms. Finding good entropy approximations and mapping them to message-passing algorithms is a very active area of research. Yedidia et al. (2001, 2005) proposed a generalized BP algorithm that works with the more refined Kikuchi entropy approximation, at the expense of additional computational burden. There is also much work building approximations of the form

$$H(\boldsymbol{\mu}) \approx \sum_{i \in \mathcal{V}} c_i H_i(\boldsymbol{\mu}_i) + \sum_{\alpha \in \mathcal{F}} c_\alpha H_\alpha(\boldsymbol{\mu}_\alpha), \tag{4.48}$$

for several choices of scalars c_i an c_α, called *entropy counting numbers*. Particularly appealing is the case where those scalars are chosen so that the approximate entropy is concave, making the variational approximation convex. An example (for pairwise Markov networks) is the sum-product *tree-reweighted BP* algorithm (Wainwright et al., 2005b), guaranteed to produce an upper bound of H, which in turn yields an upper bound of the log-partition function. Other approximations in the form (4.48) have been proposed by Wiegerinck and Heskes (2003); Weiss et al. (2007); Hazan and Shashua (2010).

4.6 Linear Programming Relaxation for MAP Inference

Approximate MAP inference is in some sense easier than marginal inference, since we do not need to worry about approximating the entropy; *i.e.*, only the marginal polytope matters. We describe approximate algorithms related with the local polytope approximation described in Section 4.4; tighter polyhedral and semi-definite approximations exist, which result in more expensive algorithms (Sontag, 2010; Wainwright and Jordan, 2008).

From Proposition 4.4, we have that the MAP inference problem (Eq. 4.34) is equivalent to the following integer linear program (ILP):

$$\begin{array}{ll} \text{maximize} & \boldsymbol{\theta} \cdot \boldsymbol{\mu} \\ \text{w.r.t.} & \boldsymbol{\mu} \in \text{LOCAL}(\mathscr{G}), \quad \boldsymbol{\mu} \text{ integer.} \end{array} \tag{4.49}$$

Ignoring the integer constraint of Eq. 4.49 yields the so-called Schlesinger's linear relaxation (Schlesinger, 1976), which we call the *LP-MAP inference problem*:

$$\boxed{\begin{array}{lll} \textbf{LP-MAP:} & \text{maximize} & \boldsymbol{\theta} \cdot \boldsymbol{\mu} \\ & \text{w.r.t.} & \boldsymbol{\mu} \in \text{LOCAL}(\mathscr{G}). \end{array}} \tag{4.50}$$

Since it is a relaxation, the solution of LP-MAP provides an upper bound of the optimal objective value in Eq. 4.49. While any off-the-shelf LP solver can be employed for this task, specialized algorithms have been designed that exploit the structure of graph, achieving superior performance on several benchmarks (Yanover et al., 2006). However, Theorem 4.6 cannot be extended to the max-product case (described in Section 4.2); *i.e.*, even if max-product loopy BP attains a stationary point, that point is not necessarily a solution of the LP-MAP problem (Wainwright and Jordan, 2008). A number of message-passing algorithms have been proposed that slightly change the message updates, fixing this issue (Wainwright et al., 2005a; Kolmogorov, 2006; Werner, 2007; Globerson and Jaakkola, 2008; Ravikumar et al., 2010). Some of these algorithms give certificates that they have found the true MAP

Algorithm 4 MPLP Algorithm for LP-MAP (Globerson and Jaakkola, 2008)

input: factor graph $\mathcal{G}$, parameters $\boldsymbol{\theta}$
initialize $\gamma = \mathbf{0}$
repeat
  for each factor $\alpha \in \mathcal{F}$ **do**
    pass messages from variables in $\mathcal{N}(\alpha)$ to factor α:

$$\delta_{i\to\alpha}(y_i) := \theta_i(y_i) + \sum_{\beta\in\mathcal{N}(i)\setminus\{\alpha\}} \gamma_{\beta\to i}(y_i).$$

    pass messages from factor α to the variables in $\mathcal{N}(\alpha)$:

$$\gamma_{\alpha\to i}(y_i) := -\delta_{i\to\alpha}(y_i) + \frac{1}{\deg(\alpha)} \max_{\boldsymbol{y}_\alpha \sim y_i} \left(\theta_\alpha(\boldsymbol{y}_\alpha) + \sum_{j\in\mathcal{N}(\alpha)} \delta_{j\to\alpha}(y_j) \right).$$

  end for
  compute current value of the dual objective:

$$d := \sum_{i\in\mathcal{V}} \max_{y_i\in\mathcal{Y}_i} \left(\theta_i(y_i) + \sum_{\alpha\in\mathcal{N}(i)} \gamma_{\alpha\to i}(y_i) \right) + \sum_{\alpha\in\mathcal{F}} \max_{\boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha} \left(\theta_\alpha(\boldsymbol{y}_\alpha) - \sum_{i\in\mathcal{N}(\alpha)} \gamma_{\alpha\to i}(y_i) \right).$$

until convergence or small enough progress in the dual.
output: approximate MAP solution $\widehat{\boldsymbol{y}}$:

$$\widehat{y}_i = \arg\max_{y_i\in\mathcal{Y}_i} \left(\theta_i(y_i) + \sum_{\alpha\in\mathcal{N}(i)} \gamma_{\alpha\to i}(y_i) \right).$$

assignment when the relaxation is tight (*e.g.*, the "weak tree agreement" condition in tree-reweighted methods). We terminate this chapter by briefly describing two classes of LP-MAP inference algorithms proposed in the literature: *message-passing algorithms* based on dual coordinate descent, and *dual decomposition algorithms* based on the subgradient method. We refer to Wainwright and Jordan (2008); Sontag et al. (2011) for thorough descriptions.

4.6.1 Block Coordinate Descent Algorithms

Many coordinate descent methods have been proposed in the literature to solve the LP-MAP problem. These methods are characterized by different reformulations of the linear program, leading to different duals, and by the choice of blocks of coordinates they optimize over. Werner (2007) proposed a max-sum diffusion algorithm reminiscent of a technique due to Kovalevsky and Koval (1975); Globerson and Jaakkola (2008) proposed the MPLP algorithm as a way of "fixing" max-product belief propagation, along with several variants; Sontag and Jaakkola (2009) proposed a more efficient choice of blocks. Some refinements of the methods above, along with an empirical comparison, are presented in the recent monograph by Sontag et al. (2011). MPLP is illustrated as Algorithm 4; in Appendix A.2, we show that MPLP is a block coordinate descent algorithm. We will use MPLP in Chapter 6 as a baseline.

Note that, unlike the loopy BP algorithm, the messages in the MPLP algorithm have a

sequential order, *i.e.*, one must loop through each factor, compute the incoming and outgoing messages involving all the variables linked to that factor, and move to the next factor. Each step improves the dual objective by moving along a block of coordinates. Under certain conditions, if the relaxation is tight, one might obtain a certificate of optimality. When the relaxation is not tight, it is sometimes possible to reduce the size of the problem or to use a cutting-plane method (Sontag et al., 2008) to make progress toward the true MAP. One advantage of block coordinate descent algorithms is that they are provably convergent (unlike the loopy BP algorithm). However, they may get stuck at a corner, which is a general disadvantage of coordinate descent algorithms in non-smooth optimization (see Bertsekas et al. 1999, Sect. 6.3.4 for details). We refer to Globerson and Jaakkola (2008) and Sontag et al. (2011) for additional properties of this and other coordinate descent algorithms.

4.6.2 Dual Decomposition with the Projected Subgradient Algorithm

A different class of algorithms for LP-MAP is based on *dual decomposition*, a classical optimization technique (Dantzig and Wolfe, 1960; Everett III, 1963; Shor, 1985). This method has been proposed in the context of graphical models by Komodakis et al. (2007); Johnson et al. (2007), and has been shown quite effective in many NLP problems (Koo et al., 2010; Rush et al., 2010; Auli and Lopez, 2011; Rush and Collins, 2011; Chang and Collins, 2011). Like the MPLP algorithm (and other message passing algorithms), the dual decomposition method is derived via a reformulation and dualization of Eq. 4.50. The reformulation consists of:

1. Adding new variables $\mu_i^\alpha(y_i)$, for each factor α, which are "replicas" of the pseudo-marginals $\mu_i(y_i)$. To avoid confusion, we rename the original variables $\mu_i(y_i)$ to $\zeta_i(y_i)$.

2. Enforcing agreement among those variables—this is done by adding equality constraints of the form $\mu_i^\alpha(y_i) = \zeta_i(y_i)$, for every $\alpha \in \mathcal{N}(i)$.

For convenience, we also introduce "split" log-potentials $\theta_i^\alpha(y_i)$; any choice that satisfies

$$\sum_{\alpha \in \mathcal{N}(i)} \theta_i^\alpha(y_i) = \theta_i(y_i) \tag{4.51}$$

can be used (a simple choice would be to set $\theta_i^\alpha(y_i) := \deg(i)^{-1}\theta_i(y_i)$). Given the characterization of the local polytope as the intersection of "smaller" marginal polytopes (cf. Eq. 4.41), we have that the LP-MAP problem in Eq. 4.50 can be expressed as:

$$\begin{aligned}
\text{maximize} \quad & \sum_{\alpha \in \mathcal{F}} \left(\sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \theta_i^\alpha(y_i)\mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha)\mu_\alpha(\boldsymbol{y}_\alpha) \right) \\
\text{w.r.t.} \quad & \boldsymbol{\mu}|_\alpha \in \text{MARG}(\mathcal{G}|_\alpha), \quad \forall \alpha \in \mathcal{F}, \\
& \zeta_i(y_i) \in \mathbb{R}, \quad \forall i \in \mathcal{V}, y_i \in \mathcal{Y}_i, \\
\text{s.t.} \quad & \mu_i^\alpha(y_i) = \zeta_i(y_i), \quad \forall i \in \mathcal{V}, \alpha \in \mathcal{N}(i), y_i \in \mathcal{Y}_i,
\end{aligned} \tag{4.52}$$

where $\text{MARG}(\mathcal{G}|_\alpha)$ is the marginal polytope of the α-subgraph of $\mathcal{G}$, and we denote by

$$\boldsymbol{\mu}|_\alpha := \left((\mu_i^\alpha(.))_{i \in \mathcal{N}(\alpha)}, \mu_\alpha(.) \right) \tag{4.53}$$

a marginal vector in that α-subgraph. By introducing a Lagrange multiplier $\lambda_i^\alpha(y_i)$ for each agreement constraint, we can write the Lagrangian function as:

$$\begin{aligned} L(\boldsymbol{\mu}, \boldsymbol{\zeta}, \boldsymbol{\lambda}) \;=\; & \sum_{\alpha \in \mathcal{F}} \left(\sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\theta_i^\alpha(y_i) + \lambda_i^\alpha(y_i)) \mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) \mu_\alpha(\boldsymbol{y}_\alpha) \right) \\ & - \sum_{\alpha \in \mathcal{F}} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \lambda_i^\alpha(y_i) \zeta_i(y_i). \end{aligned} \tag{4.54}$$

This function is to be maximized with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\zeta}$, and minimized with respect to $\boldsymbol{\lambda}$. Since the $\boldsymbol{\zeta}$-variables are unconstrained, we have

$$\max_{\boldsymbol{\mu}, \boldsymbol{\zeta}} L(\boldsymbol{\mu}, \boldsymbol{\zeta}, \boldsymbol{\lambda}) = \begin{cases} \sum_{\alpha \in \mathcal{F}} g_\alpha(\boldsymbol{\lambda}|_\alpha) & \text{if } \boldsymbol{\lambda} \in \Lambda, \\ +\infty & \text{otherwise,} \end{cases} \tag{4.55}$$

where

$$\Lambda := \left\{ \boldsymbol{\lambda} \;\middle|\; \sum_{\alpha \in \mathcal{N}(i)} \lambda_i^\alpha(y_i) = 0, \;\; \forall i \in \mathcal{V}, \;\; y_i \in \mathcal{Y}_i \right\}, \tag{4.56}$$

and $g_\alpha(\boldsymbol{\lambda}|_\alpha)$ is the solution of the following local problem (called the *α-subproblem*):

$$g_\alpha(\boldsymbol{\lambda}|_\alpha) \;:=\; \max_{\boldsymbol{\mu}|_\alpha \in \mathrm{MARG}(\mathscr{G}|_\alpha)} \left(\sum_{i, y_i} (\theta_i^\alpha(y_i) + \lambda_i^\alpha(y_i)) \mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) \mu_\alpha(\boldsymbol{y}_\alpha) \right). \tag{4.57}$$

The dual problem becomes

$$\begin{array}{ll} \text{minimize} & \sum_{\alpha \in \mathcal{F}} g_\alpha(\boldsymbol{\lambda}|_\alpha) \\ \text{w.r.t.} & \boldsymbol{\lambda} \in \Lambda. \end{array} \tag{4.58}$$

In the literature, the optimization problem in Eq. 4.58 is commonly referred to as the *master*, and each α-subproblem in Eq. 4.57 as a *slave*. Note that each of these α-subproblems is itself a MAP assignment problem in the α-subgraph $\mathscr{G}|_\alpha$ (cf. 4.34). As a consequence, the solution $\widehat{\boldsymbol{\mu}}|_\alpha$ will be integer and will correspond to a particular configuration $\widehat{\boldsymbol{y}}_\alpha$.

The dual problem (4.58) can be solved with a *projected subgradient algorithm*.[21] By Danskin's rule, a subgradient of $g_\alpha(\boldsymbol{\lambda}|_\alpha)$ is readily given by

$$\frac{\partial g_\alpha(\boldsymbol{\lambda}|_\alpha)}{\partial \lambda_i^\alpha(y_i)} = \widehat{\mu}_i^\alpha(y_i), \quad \forall i, \alpha \in \mathcal{N}(i); \tag{4.59}$$

and the projection onto Λ amounts to a centering operation. The α-subproblems (Eq. 4.57) can be handled in parallel and then have their solutions gathered for computing this projection and update the Lagrange variables. Putting these pieces together yields Algorithm 5. For compactness, we use vector notation: we write $\boldsymbol{\theta}_\alpha := (\theta_\alpha(\boldsymbol{y}_\alpha))_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha}$, $\boldsymbol{\theta}_i := (\theta_i(y_i))_{y_i \in \mathcal{Y}_i}$, and similarly for $\widehat{\boldsymbol{\mu}}_i^\alpha$, $\widehat{\boldsymbol{\zeta}}_i$, $\boldsymbol{\omega}_i^\alpha$ and $\boldsymbol{\lambda}_i^\alpha$. We assume a black-box procedure ComputeMap, which receives log-potentials as input and outputs the local MAP (as a degenerate marginal vector).

[21] The same algorithm can be derived by applying Lagrangian relaxation to the original MAP. A slightly different formulation is presented by Sontag et al. (2011) which yields a subgradient algorithm with no projection.

Algorithm 5 Dual Decomposition with Projected Subgradient (Komodakis et al., 2007)

input: graph $\mathcal{G}$, parameters $\boldsymbol{\theta}$, maximum number of iterations T, stepsizes $(\eta_t)_{t=1}^T$
initialize $\boldsymbol{\lambda} = \mathbf{0}$
for $t = 1$ **to** T **do**
  for each factor $\alpha \in \mathcal{F}$ **do**
    set unary log-potentials $\boldsymbol{\omega}_i^\alpha := \deg(i)^{-1}\boldsymbol{\theta}_i + \boldsymbol{\lambda}_i^\alpha$, for $i \in \mathcal{N}(\alpha)$
    set log-potential vector $\boldsymbol{\omega}|_\alpha := ((\boldsymbol{\omega}_i^\alpha)_{i\in\mathcal{N}(\alpha)}, \boldsymbol{\theta}_\alpha)$
    set $\widehat{\boldsymbol{\mu}}|_\alpha := \textsc{ComputeMap}(\boldsymbol{\omega}|_\alpha)$
  end for
  compute average $\boldsymbol{\zeta}_i := \deg(i)^{-1} \sum_{\alpha\in\mathcal{N}(i)} \widehat{\boldsymbol{\mu}}_i^\alpha$
  update $\boldsymbol{\lambda}_i^\alpha := \boldsymbol{\lambda}_i^\alpha - \eta_t\,(\widehat{\boldsymbol{\mu}}_i^\alpha - \boldsymbol{\zeta}_i)$
end for
output: dual variable $\boldsymbol{\lambda}$

Algorithm 5 inherits the properties of subgradient algorithms, hence it converges to the optimum of (4.58) if the stepsize sequence $(\eta_t)_{t\in T}$ is diminishing and nonsummable: $\eta_t \geq 0, \forall t$; $\lim \eta_t = 0$; and $\sum_{t=1}^\infty \eta_t = \infty$ (Bertsekas et al., 1999). In practice, convergence can be quite slow if the number of slaves is large, as we will see in Chapter 7. In Chapter 6, we will propose a new LP-MAP inference algorithm that is more suitable for that kind of problem.

Part II

Inference

Chapter 5

Constrained Structured Prediction

In Chapters 3 and 4, we have described models and algorithms for structured prediction. We have seen how graphical models are a useful representational framework, but we restricted attention to strictly positive potential functions. In this chapter, we present novel contributions for *constrained structured prediction models*, which incorporate hard constraints as a way of ruling out "forbidden" configurations. The contributions presented in this chapter are:

- We provide a unified representation for constrained structured prediction problems by formalizing the concept of a *constrained factor graph*. This representation puts together several models previously proposed in the literature, such as constrained conditional models and grounded Markov logic networks, providing a unified treatment.

- We characterize the geometry of constrained factor graphs. This is done by extending the machinery of Chapter 4 to our constrained case. In particular, we extend the notions the marginal and local polytopes of constrained graphs, establish Fenchel-Legendre duality, and derive variational representations.

- We introduce an inventory of hard constraint factors that are able to express arbitrary first-order logic constraints. We also provide a concise representation of their marginal polytopes, which will be useful in later chapters.

- We provide the necessary algorithmic tools to deal with these factors. Specifically, we describe how to compute sum-product (and max-product) messages, how to compute marginals (and max-marginals), as well as the partition function and the factor entropies. We also provide more general results concerning how all these quantities depend solely on a black-box algorithm for computing the marginals (or max-marginals) and evaluating the partition function associated with each factor, independently.

Some of the results presented in this chapter were originally introduced in Martins et al. (2010f), namely the inventory of hard constraint logic factors, as well as the message update equations, the marginals, *etc*. That paper, however, was focused on a particular application, dependency parsing, so many details and proofs were omitted from the manuscript. This chapter presents those results more leisurely, and adopts a general framework perspective rather than being application-oriented. It also extends the paper by presenting the geometric characterizations listed above.

5.1 Motivation and Related Work

In Chapter 4, we have described graphical models with *strictly positive potentials*, which results in a distribution $P(\boldsymbol{Y})$ with full support on the product set $\prod_{i \in \mathcal{V}} \mathcal{Y}_i$. Constrained models, on the other hand, assume that the set of admissible outputs may be *properly contained* in that product set, *i.e.*, $\mathcal{Y}(x) \subset \prod_{i \in \mathcal{V}} \mathcal{Y}_i$. There are several motivations for considering constraints:

- **Sometimes the set of admissible outputs $\mathcal{Y}(x)$ is inherently constrained.** This is so in problems arising in error-correcting coding theory (Richardson and Urbanke, 2008), bipartite matching (Duchi et al., 2007), computer vision (Nowozin and Lampert, 2009), and also natural language processing (Sutton, 2004; Smith and Eisner, 2008). For those problems, only certain arrangements of parts yield valid outputs (words in a codebook, matchings, connected regions, and trees, respectively). While constraints can in practice be enforced by employing log-potentials with large absolute values, that strategy precludes exploiting the structure and sparsity of the constraints and often leads to inefficient algorithms.

- **We may want to inject prior knowledge in the form of declarative constraints.** These could be, *e.g.*, first order logic expressions written by experts. The inclusion of such constraints yields more accurate models, and is especially useful when annotated data is scarce (Roth and Yih, 2004; Punyakanok et al., 2005; Riedel and Clarke, 2006; Richardson and Domingos, 2006; Chang et al., 2008; Poon and Domingos, 2009; Meza-Ruiz and Riedel, 2009).

- **We may want to enrich our model by adding new parts that are logical functions of existing parts.** This can be accomplished by adding those parts as variable nodes in the graphical model, and expressing the logical functions as hard constraint factors, in order to enforce consistency with the already existing variables. This turns out to play a very important role in the multi-commodity flow models for dependency parsing that we will introduce in Chapter 7.

5.1.1 Related Work

The importance of modeling constraints in structured problems was realized a long time back. Early attempts to fuse the semantics of probability theory and first-order logic appear in Gaifman (1964); Halpern (1990); more recently, this line of research has gained prominence through probabilistic programming systems, such as BLOG (Milch et al., 2007) and Markov logic networks (Richardson and Domingos, 2006). Related frameworks are probabilistic Horn abduction (Poole, 1993), probabilistic constraint logic programming (Riezler, 1998), probabilistic relational models (Friedman et al., 1999), and relational Markov networks (Taskar et al., 2002). Many of these are formalisms for probabilistic reasoning in knowledge databases with soft and hard first-order logic constraints. Once grounded, those networks become constrained graphical models.

A related line of research is that of *constrained conditional models* (Roth and Yih, 2004; Punyakanok et al., 2005; Chang et al., 2008). In that framework, one starts from a vanilla structured conditional model (such as a CRF), in which inference and learning are tractable; then, a list of domain-dependent side constraints is specified, allowing us to inject rich prior

knowledge. These are usually declarative constraints written in first-order logic. In general, exact inference becomes intractable; most of the previous work resorts to small and medium scale problems, employing off-the-shelf ILP solvers for MAP decoding. A common strategy is to ignore the constraints at training time—this strategy is called "L+I" ("*Learning Plus Inference*") in Punyakanok et al. (2005). We will interpret this strategy later as a relaxation of the marginal polytope, which we show to be looser than LP-MAP. Constrained conditional models have been applied in semantic role labeling (Roth and Yih, 2005), coreference resolution (Denis and Baldridge, 2007), sentence compression (Clarke and Lapata, 2008), and dependency parsing (Riedel and Clarke, 2006). They have also been used in the context of semi-supervised constraint-driven learning (Chang et al., 2007).

Apart from the work mentioned above, there have been ad-hoc adaptations of message passing algorithms to solve particular constrained problems in graphical models, exploiting the structure of the constraints. Examples are the bipartite matching model of Duchi et al. (2007), the loopy BP dependency parser of Smith and Eisner (2008), and the sparse message passing algorithm for weighted SAT of Culotta et al. (2007).

This chapter is organized as follows: we study the geometry of constrained models in Section 5.2. In Section 5.3 we provide an inventory of hard constraint factors that can express logic constraints. We derive expressions for computing messages, marginals, and max-marginals in Section 5.4. Section 5.5 characterizes the LP-MAP and Bethe entropy approximations. We suggest paths for future work in section 5.6, and we conclude in Section 5.7.

5.2 Constrained Graphical Models and Their Geometry

We have seen in Chapter 4 that graphical models with strictly positive potentials can be seen as exponential families (cf. Eq. 4.16):

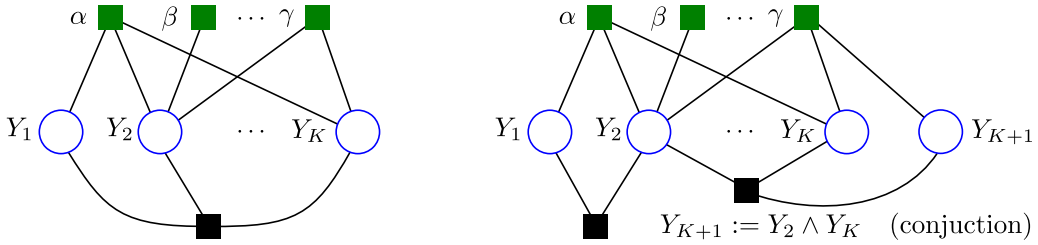
$$P_{\boldsymbol{\theta}}(\boldsymbol{y}) \;=\; \frac{1}{Z(\boldsymbol{\theta})} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(\boldsymbol{y}))Q(\boldsymbol{y}), \tag{5.1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^R$ is a vector of factor and variable log-potentials, and $\boldsymbol{\phi}(\boldsymbol{y}) := \boldsymbol{\chi}(\boldsymbol{y}) \in \mathbb{R}^R$ is an indicator vector of local configurations. In that section—and in all Chapter 4—we restricted attention to the case where Q was constant and hence non-informative. There might be good reasons, however, for a less trivial choice of Q. For example, we may let Q vanish on some configurations (those which are not in the set of admissible outputs $\mathcal{Y}(x) \subseteq \prod_{i \in \mathcal{V}} \mathcal{Y}_i$):

$$Q(\boldsymbol{y}) = \begin{cases} 1, & \text{if } \boldsymbol{y} \in \mathcal{Y}(x) \\ 0, & \text{otherwise.} \end{cases} \tag{5.2}$$

This choice ensures that any distribution in the exponential family assigns null probability to any $\boldsymbol{y} \notin \mathcal{Y}(x)$. To allow specifying a list of constraints with different scopes, we assume a factorization $Q(\boldsymbol{y}) := \prod_\beta Q_\beta(\boldsymbol{y}_\beta)$, where $Q_\beta(\boldsymbol{y}_\beta) := [\![\boldsymbol{y}_\beta \in \mathcal{S}_\beta]\!]$, with each $\beta \subseteq \mathcal{V}$ and $\mathcal{S}_\beta$ being an *acceptance set* of partial configurations on β.

Definition 5.1 *A* constrained factor graph *is a bipartite undirected graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{F} \cup \mathcal{H}, \mathcal{E})$, where each edge in $\mathcal{E}$ has one endpoint in $\mathcal{V}$ and another in $\mathcal{F} \cup \mathcal{H}$, and the sets $\mathcal{V}$, $\mathcal{F}$, and $\mathcal{H}$ are all disjoint. Elements of $\mathcal{V}$ are called* variable nodes*, elements of $\mathcal{F}$ are* soft factor nodes*, and those of*



$Y_1, Y_2, \ldots, Y_K$ must form a valid codeword

either Y_1 or Y_2 must be 1

Figure 5.1: Constrained factor graphs. We represent soft factors in *green* above the variable nodes, and hard constraints in *black* below the variable nodes. Left: a global constraint that confines the set of admissible outputs to a particular codebook. Right: Typical hard constraints in declarative constraint programming. One of them is a factor connecting existing variables to a new extra variable. This allows introducing scores that depend on the evaluation of a logical function of the former.

$\mathcal{H}$ *are* hard constraint factor nodes. *In addition:*

- *Each variable* $i \in \mathcal{V}$ *has a strictly positive potential function* $\psi_i : \mathcal{Y}_i \to \mathbb{R}_{++}$.
- *Each soft factor* $\alpha \in \mathcal{F}$ *has a strictly positive potential function* $\psi_\alpha : \mathcal{Y}_\alpha \to \mathbb{R}_{++}$.
- *Each hard factor* $\beta \in \mathcal{H}$ *has a binary potential function* $\psi_\beta : \mathcal{Y}_\beta \to \{0, 1\}$ *defined as* $\psi_\beta(\boldsymbol{y}_\beta) = [\![\boldsymbol{y}_\beta \in \mathcal{S}_\beta]\!]$, *where* $\mathcal{S}_\beta \subseteq \mathcal{Y}_\beta$ *is an* acceptance set.

Figure 5.1 shows examples of constrained factor graphs, where the hard constraint factors have different usages. In the sequel, we will use constrained factor graphs to represent conditional probability distributions $P_{\boldsymbol{w}}(\boldsymbol{Y}|X = x)$ defined as structured linear models, in which potentials on variables and soft factors are parameterized via $\psi_i(y_i) := \exp(\boldsymbol{w} \cdot \boldsymbol{f}_i(x, y_i))$ and $\psi_\alpha(\boldsymbol{y}_\alpha) := \exp(\boldsymbol{w} \cdot \boldsymbol{f}_\alpha(x, \boldsymbol{y}_\alpha))$. Hard factors, in turn, induce a function

$$Q(x, \boldsymbol{y}) \quad := \quad \prod_{\beta \in \mathcal{H}} [\![\boldsymbol{y}_\beta \in \mathcal{S}_\beta]\!]. \tag{5.3}$$

Hence, $P_{\boldsymbol{w}}(\boldsymbol{Y}|X = x)$ can be written as

$$P_{\boldsymbol{w}}(\boldsymbol{y}|x) \quad := \quad \begin{cases} \dfrac{1}{Z(\boldsymbol{w}, x)} \exp\left(\displaystyle\sum_{p \in \mathcal{P}} \boldsymbol{w} \cdot \boldsymbol{f}_p(x, \boldsymbol{y}_p)\right), & \text{if } \boldsymbol{y}_\beta \in \mathcal{S}_\beta,\ \forall \beta \in \mathcal{H}, \\ 0, & \text{otherwise,} \end{cases} \tag{5.4}$$

where $\mathcal{P} := \mathcal{V} \cup \mathcal{F}$ is the set of *places* (cf. Eq. 4.26). The set of admissible outputs is

$$\mathcal{Y}(x) = \left\{ \boldsymbol{y} \in \prod_{i \in \mathcal{V}} \mathcal{Y}_i \;\middle|\; \forall \beta \in \mathcal{H} : \boldsymbol{y}_\beta \in \mathcal{S}_\beta(x) \right\}, \tag{5.5}$$

and the partition function is

$$Z(\boldsymbol{w}, x) = \sum_{\boldsymbol{y} \in \mathcal{Y}(x)} \exp\left(\sum_{p \in \mathcal{P}} \boldsymbol{w} \cdot \boldsymbol{f}_p(x, \boldsymbol{y}_p)\right). \tag{5.6}$$

Binary Constrained Factor Graphs. As mentioned in Section 5.1.1, many problems involving first-order logic probabilistic models have been considered (Friedman et al., 1999; Richardson and Domingos, 2006). Given some evidence, those models can be "compiled" into a factor graph whose variable nodes represent truth values, *i.e.*, we have $\mathcal{Y}_i = \{0, 1\}$ for every $i \in \mathcal{V}$. We call such a graph a *binary constrained factor graph*. When all variables are binary, it is convenient to use a more compact representation instead of the overcomplete parametrization. For constrained pairwise MRFs, an appealing choice is the representation described in Example 4.1 for Ising models:

$$P_{\boldsymbol{s}}(\boldsymbol{y}|x) \;=\; \begin{cases} \dfrac{1}{Z(\boldsymbol{s}, x)} \exp\left(\displaystyle\sum_i s_i y_i + \sum_{ij} s_{ij} y_i y_j\right), & \text{if } \boldsymbol{y}_\beta \in \mathcal{S}_\beta,\ \forall \beta \in \mathcal{H}, \\ 0, & \text{otherwise.} \end{cases} \tag{5.7}$$

Any canonical overcomplete parameter vector $\boldsymbol{\theta}$ can be converted to a parameter vector $\boldsymbol{s}$ through Eq. 4.25. Without loss of generality, we may assume that $\boldsymbol{s}$ has the form $s_i := \boldsymbol{w} \cdot \boldsymbol{f}_i(x)$ and $s_{ij} := \boldsymbol{w} \cdot \boldsymbol{f}_{ij}(x)$. We will come back to this parametrization in several places.

5.2.1 Marginal and Local Polytopes

In Section 4.3.2, we have defined the marginal polytope of an unconstrained factor graph, as the set of realizable marginal vectors. We now extend that definition to constrained factor graphs $\mathcal{G} = (\mathcal{V} \cup \mathcal{F} \cup \mathcal{H}, \mathcal{E})$. We let $\mathcal{P} = \mathcal{V} \cup \mathcal{F}$ be the set of places of $\mathcal{G}$ (note that this set contains the variable nodes and the soft factor nodes, but *does not contain* the hard factor nodes). Recall that a *part* is a pair $(p, \boldsymbol{y}_p)$, where $p \in \mathcal{P}$ is a place and $\boldsymbol{y}_p \in \mathcal{Y}_p$ is a local configuration. We denote by $\mathcal{R} := \{(p, \boldsymbol{y}_p) \mid p \in \mathcal{P}, \boldsymbol{y}_p \in \mathcal{Y}_p\}$ the set of parts of $\mathcal{G}$. We consider marginal vectors $\boldsymbol{\mu} := (\mu_r)_{r \in \mathcal{R}}$ indexed by the set of parts, such that $\mu_{(p, \boldsymbol{y}_p)} \equiv \mu_p(\boldsymbol{y}_p)$ denotes the posterior probability $\Pr\{\boldsymbol{Y}_p = \boldsymbol{y}_p | X = x\}$. Given $\widehat{\boldsymbol{y}} \in \mathcal{Y}(x)$, we define its indicator vector $\boldsymbol{\chi}(\widehat{\boldsymbol{y}}) \in \mathbb{R}^{|\mathcal{R}|}$ as the vector whose entries $[\chi(\widehat{\boldsymbol{y}})]_{(p, \boldsymbol{y}_p)}$ are 1 if $\widehat{\boldsymbol{y}}_p = \boldsymbol{y}_p$, and 0 otherwise.

Definition 5.2 (Marginal Polytope) *The* marginal polytope *of a constrained graph $\mathcal{G}$ is the set of marginals which are realizable for distributions that vanish outside $\mathcal{Y}(x)$:*

$$\begin{aligned} \mathrm{MARG}(\mathcal{G}) : \;&=\; \left\{ \boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{R}|} \;\middle|\; \begin{array}{l} \exists P(\boldsymbol{Y}) \text{ s.t. } P(\boldsymbol{y}) = 0,\ \forall \boldsymbol{y} \notin \mathcal{Y}(x) \ \wedge \\ \boldsymbol{\mu} = \mathbb{E}_{\boldsymbol{Y} \sim P(\boldsymbol{Y})}[\boldsymbol{\chi}(\boldsymbol{Y})] \end{array} \right\} \\ &=\; \mathrm{conv}\{\boldsymbol{\chi}(\boldsymbol{y}) \mid \boldsymbol{y} \in \mathcal{Y}(x)\}. \end{aligned} \tag{5.8}$$

As before, the vertices of the marginal polytope are in one-to-one correspondence with the elements of $\mathcal{Y}(x)$. Note that each component of a marginal vector $\boldsymbol{\mu} \in \mathrm{MARG}(\mathcal{G})$ stands for a marginal probability of some *variable* or *soft factor* configuration—there are no components for *hard factor* configurations. Hard constraint factors play an important role, however: they "eliminate" the vertices corresponding to forbidden configurations, chopping off pieces of the polytope.

In Section 4.4, we have described an outer approximation of the marginal polytope, called the *local polytope*, which can be seen as the intersection of "smaller" marginal polytopes, one per factor, lifted to the ambient space $\mathbb{R}^{|\mathcal{R}|}$ (cf. Eq. 4.41). To generalize that notion to constrained graphs, we need first to define what is the "smaller" marginal polytope of a

hard constraint factor. Let $\beta \in \mathcal{H}$ be a hard factor connected to a set of variables $\mathcal{N}(\beta)$, with an acceptance set $\mathcal{S}_\beta$. For each $\widehat{\boldsymbol{y}}_\beta \in \mathcal{S}_\beta$, define $\boldsymbol{\chi}_\beta(\widehat{\boldsymbol{y}}_\beta)$ as the vector in $\mathbb{R}^{\sum_{i\in\mathcal{N}(\beta)}|\mathcal{Y}_i|}$ whose (i, y_i)-th component (with $i \in \mathcal{N}(\beta)$ and $y_i \in \mathcal{Y}_i$) is 1 if $\widehat{y}_i = y_i$, and 0 otherwise. We define

$$\mathrm{MARG}(\mathcal{G}|_\beta) = \mathrm{conv}\{\boldsymbol{\chi}_\beta(\boldsymbol{y}_\beta) \mid \boldsymbol{y}_\beta \in \mathcal{S}_\beta\}. \tag{5.9}$$

We then define the local polytope as:

$$\mathrm{LOCAL}(\mathcal{G}) := \left\{\boldsymbol{\mu} \in \mathbb{R}^{|\mathcal{R}|} \mid \boldsymbol{\mu}|_\alpha \in \mathrm{MARG}(\mathcal{G}|_\alpha) \quad \text{for every } \alpha \in \mathcal{F}\cup\mathcal{H}\right\}, \tag{5.10}$$

where, for soft factors $\alpha \in \mathcal{F}$, we denote by $\boldsymbol{\mu}|_\alpha$ the restriction of $\boldsymbol{\mu}$ to the entries $\mu_\alpha(.)$ and $\mu_i(.)$ for every $i \in \mathcal{N}(\alpha)$; and for hard factors $\beta \in \mathcal{H}$, we denote by $\boldsymbol{\mu}|_\beta$ the restriction of $\boldsymbol{\mu}$ to the entries $\mu_i(.)$ for every $i \in \mathcal{N}(\beta)$.

Proposition 5.1 *Let $\mathcal{G}$ be a constrained factor graph, and denote by $\mathcal{G}_{\mathrm{u}}$ the unconstrained graph that is obtained from $\mathcal{G}$ by removing the hard factors. The following chain of inclusions hold:*

$$\mathrm{MARG}(\mathcal{G}) \subseteq \mathrm{LOCAL}(\mathcal{G}) \subseteq \mathrm{LOCAL}(\mathcal{G}_{\mathrm{u}}). \tag{5.11}$$

Moreover, if $\mathcal{G}_{\mathrm{u}}$ is free of cycles, then $\mathrm{MARG}(\mathcal{G}) \subseteq \mathrm{LOCAL}(\mathcal{G}) \subseteq \mathrm{MARG}(\mathcal{G}_{\mathrm{u}})$.

Proof. Any marginal vector $\boldsymbol{\mu} \in \mathrm{MARG}(\mathcal{G})$ much satisfy $\boldsymbol{\mu}|_\alpha \in \mathrm{MARG}(\mathcal{G}|_\alpha)$ for every $\alpha \in \mathcal{F}\cup\mathcal{H}$, hence we have $\mathrm{MARG}(\mathcal{G}) \subseteq \mathrm{LOCAL}(\mathcal{G})$. We can express $\mathrm{LOCAL}(\mathcal{G})$ as:

$$\mathrm{LOCAL}(\mathcal{G}) = \mathrm{LOCAL}(\mathcal{G}_{\mathrm{u}}) \cap \bigcap_{\beta\in\mathcal{H}} \mathrm{conv}\{\boldsymbol{\chi}(\boldsymbol{y}) \mid \boldsymbol{y}_\beta \in \mathcal{S}_\beta\}, \tag{5.12}$$

which implies $\mathrm{LOCAL}(\mathcal{G}) \subseteq \mathrm{LOCAL}(\mathcal{G}_{\mathrm{u}})$, proving the first chain of inclusions. For the second one, note that, if $\mathcal{G}_{\mathrm{u}}$ is free of cycles, Proposition 4.3 states that $\mathrm{LOCAL}(\mathcal{G}_{\mathrm{u}}) = \mathrm{MARG}(\mathcal{G}_{\mathrm{u}})$, from which the result follows. ∎

Proposition 5.2 *The integral points of* $\mathrm{LOCAL}(\mathcal{G})$ *are the vertices of* $\mathrm{MARG}(\mathcal{G})$. *Hence:*

$$\mathrm{MARG}(\mathcal{G}) = \mathrm{conv}(\mathrm{LOCAL}(\mathcal{G}) \cap \mathbb{Z}^{|\mathcal{R}|}). \tag{5.13}$$

Proof. Analogous to Proposition 4.4. ∎

Propositions 5.1–5.2 allow us to understand geometrically some important approximations that are made in practice. For a constrained graph $\mathcal{G}$ which is constructed from a cycle-free unconstrained graph $\mathcal{G}_{\mathrm{u}}$ by adding hard constraints, we have the two following outer bounds for the (generally intractable) marginal polytope $\mathrm{MARG}(\mathcal{G})$:

- the *local polytope* $\mathrm{LOCAL}(\mathcal{G})$, whose integer vertices are precisely the vertices of $\mathrm{MARG}(\mathcal{G})$, but which may contain additional fractional vertices (cf. Proposition 5.2);
- the *marginal polytope of the unconstrained graph*, $\mathrm{MARG}(\mathcal{G}_{\mathrm{u}})$, a looser bound which may contain additional integer vertices (namely, the configurations which are ruled out by the constraints).

This will be next illustrated with a concrete example.[1]

Example 5.1 *Consider the factor graph $\mathcal{G}$ in Figure 5.2, with two binary variables ($\mathcal{Y}_1 = \mathcal{Y}_2 = \{0,1\}$), one pairwise factor connecting the two variables, and one hard constraint factor β imposing that at least one of the two variables takes the value 1 (*i.e.*, with an acceptance set $\mathcal{S}_\beta = \{01, 10, 11\}$).*

With an overcomplete parametrization there are 8 parameters: 2 for each variable, and 4 for each possible configuration in the pairwise factor. The marginal polytope $\mathrm{MARG}(\mathcal{G})$ is thus an object embedded in $\mathbb{R}^8$. However, there are only three degrees of freedom, and in fact $\mathrm{MARG}(\mathcal{G})$ can be seen as a lifted version of a 3-dimensional polytope. Let us consider a minimal parametrization through the marginal probabilities $z_1 := \mu_1(1)$, $z_2 := \mu_2(1)$ and $z_{12} := \mu_{12}(1,1)$. As can be easily verified by writing a probability table, the remaining marginals can be obtained from these as:

$$\begin{aligned} \mu_1(0) &= 1 - z_1, & \mu_2(0) &= 1 - z_2, \\ \mu_{12}(1,0) &= z_1 - z_{12}, & \mu_{12}(0,1) &= z_2 - z_{12}, \\ \mu_{12}(0,0) &= 1 - z_1 - z_2 + z_{12}. \end{aligned} \tag{5.14}$$

We can thus parameterize a marginal vector as $\boldsymbol{z} = (z_1, z_2, z_{12})$. Noting that the set of admissible outputs is $\mathcal{Y} = \{01, 10, 11\}$ and that $y_{12} = y_1 \wedge y_2$, we can regard $\mathrm{MARG}(\mathcal{G})$ as a lifted version of the polytope[2]

$$\mathcal{Z} = \mathrm{conv}\{(0,1,0), (1,0,0), (1,1,1)\}. \tag{5.15}$$

Consider now the graph $\mathcal{G}_\mathrm{u}$ that is obtained from $\mathcal{G}$ by removing the hard constraint. The polytope $\mathrm{MARG}(\mathcal{G}_\mathrm{u})$ is an outer bound of $\mathrm{MARG}(\mathcal{G})$ and is a lifted version of the polytope

$$\begin{aligned} \mathcal{Z}_\mathrm{u} &= \mathrm{conv}\{(0,0,0), (0,1,0), (1,0,0), (1,1,1)\} \\ &= \left\{ (z_1, z_2, z_{12}) \in [0,1]^3 \;\middle|\; \begin{array}{l} z_{12} \le z_1,\ z_{12} \le z_2, \\ z_{12} \ge z_1 + z_2 - 1 \end{array} \right\}. \end{aligned} \tag{5.16}$$

Finally, let us see how the local polytope $\mathrm{LOCAL}(\mathcal{G})$ looks. Since the unconstrained part of the graph is cycle-free, we have that $\mathrm{LOCAL}(\mathcal{G}_\mathrm{u}) = \mathrm{MARG}(\mathcal{G}_\mathrm{u})$. It will be shown in Section 5.3.2 that the marginal polytope of the β-subgraph $\mathcal{G}|_\beta$ is a lifted version of

$$\mathcal{Z}_\beta = \{(z_1, z_2) \in [0,1]^2 \mid z_1 + z_2 \ge 1\}. \tag{5.17}$$

From Eq. 5.12, we then have that $\mathrm{LOCAL}(\mathcal{G})$ is a lifted version of

$$\begin{aligned} \bar{\mathcal{Z}} &= \left\{ (z_1, z_2, z_{12}) \in [0,1]^3 \;\middle|\; \begin{array}{l} z_{12} \le z_1,\ z_{12} \le z_2, \\ z_{12} \ge z_1 + z_2 - 1, \\ z_1 + z_2 \ge 1 \end{array} \right\} \\ &= \mathrm{conv}\left\{ (0,1,0), (1,0,0), (1,1,1), \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \right\}. \end{aligned} \tag{5.18}$$

Hence we have a chain $\mathcal{Z} \subset \bar{\mathcal{Z}} \subset \mathcal{Z}_\mathrm{u}$; the two outer bounds of the marginal polytope are the local

[1] These two outer bounds are commonly used in practice, as we shall see in the sequel. The first one is used in *turbo-training* (Chapters 6–7), the second one is precisely what is used in the L+I training of constrained conditional models (Roth and Yih, 2004; Punyakanok et al., 2005; Chang et al., 2008).

[2] See Appendix B for a definition of *lifting*.

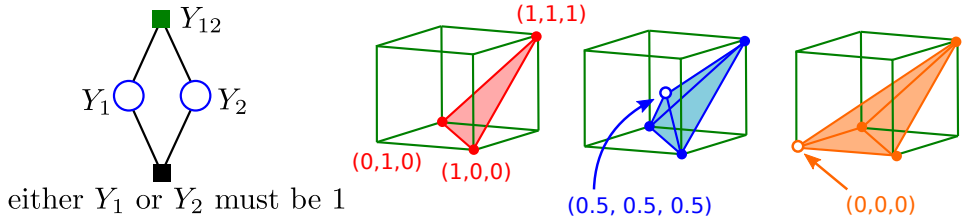


Figure 5.2: Left: A constrained factor graph $\mathcal{G}$ with two binary variables ($\mathcal{Y}_1 = \mathcal{Y}_2 = \{0, 1\}$), one pairwise factor, and one hard constraint factor imposing that at least one of the variables has the value 1. Right: The marginal polytope $\text{MARG}(\mathcal{G})$ (in red), and its two outer bounds: the local polytope $\text{LOCAL}(\mathcal{G})$ (in blue), and the polytope $\text{MARG}(\mathcal{G}_u)$ (in orange). Each of these outer bounds introduce one extra vertex, and we have a chain $\text{MARG}(\mathcal{G}) \subseteq \text{LOCAL}(\mathcal{G}) \subseteq \text{MARG}(\mathcal{G}_u)$. See text for details.

polytope $\bar{\mathcal{Z}}$, which has an additional fractional vertex $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, and a looser polytope $\mathcal{Z}_u$ (the marginal polytope of the unconstrained graph), with an additional integer vertex $(0, 0, 0)$. Figure 5.2 shows the three nested polytopes.

5.2.2 Duality and Variational Representations

Most theoretical results seen in Chapter 4 still hold for constrained factor graphs, including Propositions 4.1–4.2, and seamlessly Theorem 4.6 (see Yedidia et al. 2004, Conjectures 1–2). We state here the adaptation of Propositions 4.1–4.2 for constrained linear conditional models. In doing so, we also depart from a "canonical overcomplete parameterization." Rather, we extend those propositions to the more general scenario where arbitrary features are allowed and some weights are *tied* (*i.e.*, shared accross factors). This is the most common scenario in NLP problems.

Denote by $\mathcal{E}_x$ the exponential family of distributions of the form in Eq. 5.4, for a fixed $x \in \mathcal{X}$. As before (Section 4.3), we let $H(P(\boldsymbol{Y}|X = x)) := \mathbb{E}_{\boldsymbol{Y}}[-\log P(\boldsymbol{Y}|x)]$ denote the entropy of $P(\boldsymbol{Y}|X = x)$. Recall that the feature vectors $\boldsymbol{f}(x, y) \in \mathbb{R}^D$ decompose as in Eq. 3.12:

$$\boldsymbol{f}(x, \boldsymbol{y}) := \sum_{p \in \mathcal{P}} \boldsymbol{f}_p(x, \boldsymbol{y}_p), \tag{5.19}$$

where $\mathcal{P} = \mathcal{V} \cup \mathcal{F}$ is the set of places. For convenience, we introduce the D-by-$|\mathcal{R}|$ matrix $\mathbf{F}(x)$, which "stacks" the local feature vectors $\boldsymbol{f}_p(x, \boldsymbol{y}_p)$ for each part $(p, \boldsymbol{y}_p) \in \mathcal{R}$.

Proposition 5.3 *There is a map coupling each distribution $P_{\boldsymbol{w}}(.|x) \in \mathcal{E}_x$ to a unique $\boldsymbol{\mu} \in \text{MARG}(\mathcal{G})$ such that $\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{\chi}(\boldsymbol{Y})] = \boldsymbol{\mu}$. Define $H(\boldsymbol{\mu}) := H(P_{\boldsymbol{w}}(\boldsymbol{Y}|X = x))$ if some $P_{\boldsymbol{w}}(.|x)$ is coupled to $\boldsymbol{\mu}$, and $H(\boldsymbol{\mu}) := -\infty$ if no such $P_{\boldsymbol{w}}(.|x)$ exists. Then:*

1. *The log-partition function has the following variational representation:*

$$\log Z(\boldsymbol{w}, x) = \max_{\boldsymbol{\mu} \in \text{MARG}(\mathcal{G})} \boldsymbol{w}^\top \mathbf{F}(x)\boldsymbol{\mu} + H(\boldsymbol{\mu}). \tag{5.20}$$

2. *The problem in Eq. 5.20 is convex and its solution is attained at $\widehat{\boldsymbol{\mu}} := \boldsymbol{\mu}(\boldsymbol{w})$, i.e., the marginal vector corresponding to the distribution $P_{\boldsymbol{w}}(.|x)$.*

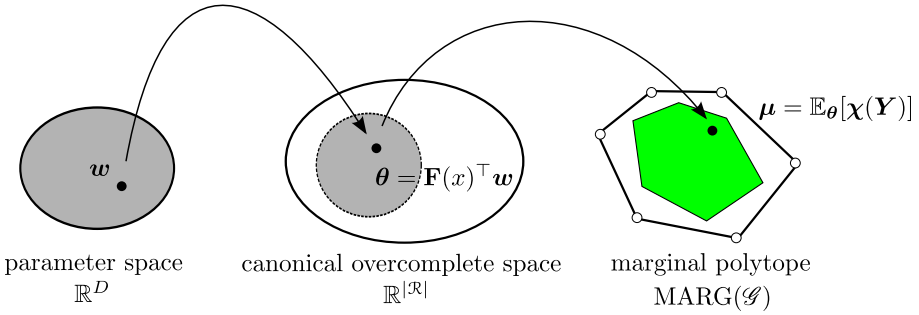


Figure 5.3: Dual parametrization of the distributions in $\mathscr{E}_x$. Our parameter space (left) is linearly mapped to the space of canonical overcomplete parameters (middle). The latter is mapped to the marginal polytope $\mathrm{MARG}(\mathcal{G})$ (right). In general only a subset of $\mathrm{MARG}(\mathcal{G})$ is reachable from our parameter space. Any distribution can be parametrized by a vector $w \in \mathbb{R}^D$ or by a point $\mu \in \mathrm{MARG}(\mathcal{G})$. Note that different parameter vectors w may parametrize the same distribution $P_w(.|x)$; each equivalence class $[w]$ is an affine subspace of $\mathbb{R}^D$.

3. *The gradient of the log-partition function is* $\nabla_w \log Z(w, x) = \mathbf{F}(x)\widehat{\mu}$.

4. *The entropy of* $P_w(.|x)$ *can be expressed as:*

$$H(P_w(Y|X = x) = -w^\top \mathbf{F}(x)\widehat{\mu} + \log Z(w, x). \tag{5.21}$$

5. *The MAP configuration* $\widehat{y} := \arg\max_{y \in \mathcal{Y}(x)} P_w(y|x)$ *can be obtained by solving the LP:*

$$\chi(\widehat{y}) = \arg \max_{\mu \in \mathrm{MARG}(\mathcal{G})} w^\top \mathbf{F}(x)\mu. \tag{5.22}$$

Proof. Wainwright and Jordan (2008, Theorem 3.4) provide a proof for a minimal representation, and extend it in their Appendix B.1 to the canonical overcomplete case (as stated in our Proposition 4.2). Those results are derived for general exponential families with arbitrary base measures, hence hold for any Q in Eq. 5.1, which includes constrained factor graphs. The canonical overcomplete representation corresponds to the case where $\mathbf{F}(x)$ is the identity matrix. In that case, the map from the parameter space to the relative interior of the marginal polytope is surjective. In the general case, we have a linear map $w \mapsto \theta = \mathbf{F}(x)^\top w$ that "places" our parameters $w \in \mathbb{R}^D$ onto a *linear subspace* of the canonical overcomplete parameter space; therefore, our map $w \mapsto \mu$ is not necessarily onto $\mathrm{relint}\,\mathrm{MARG}(\mathcal{G})$, unlike in Wainwright and Jordan (2008), and our $H(\mu)$ is defined slightly differently: it can take the value $-\infty$ if no w maps to μ. This does not affect the expression in (4.36), since the solution of this optimization problem with our $H(\mu)$ replaced by theirs is also the feature expectation under $P_w(.|x)$ and the associated μ, by definition, always yields a finite $H(\mu)$. ∎

Figure 5.3 provides an illustration of the dual parametrization implied by Proposition 5.3. When the graph has cycles, it is often intractable to solve any of the optimization problems in Eqs. 5.20 and 5.22. In Chapter 4, we have discussed two approximations:

- Replacing $\mathrm{MARG}(\mathcal{G})$ by the *local polytope* $\mathrm{LOCAL}(\mathcal{G})$;
- Replacing $H(P_w(Y|X = x))$ by *Bethe's entropy approximation* $H_{\text{Bethe}}(P_w(Y|X = x))$.

Most of the story told in Chapter 4 connecting these surrogates to approximate inference algorithms still applies to constrained factor graphs—this includes the loopy BP algorithm, as well as the other message passing and dual decomposition algorithms. There are some questions that need to be answered though, to which we devote the remaining sections:

- **How to compute messages or solve subproblems involving hard constraint factors?** In particular, how to deal with hard factors with a large degree, sidestepping the exponential runtime with respect to that number? Intuitively, it looks like we should be able to exploit the sparsity of the potential functions for those graphs. We will see in Section 5.4 that this intuition is correct, and we will provide a general recipe for computing messages and solving subproblems for a wide range of factors.

- **When hard constraints come into play, how good are the approximate inference algorithms?** In particular, how accurate is the Bethe entropy approximation, and how does that affect the sum-product loopy BP algorithm? As expected, this strongly depends on the topology of the graph and on the actual hard constraints. We will see that in general, the story is more intricate in marginal inference (since there is need of approximating the entropy), than in MAP inference (where only the marginal polytope is approximated).

- **How tractable are the marginal and local polytopes?** As suggested in Example 5.1, the marginal polytope of a constrained factor graph $\mathscr{G}$ is in general intractable, even when the factor graph without the constraints $\mathscr{G}_u$ is tractable. However, as the same example shows, the local polytope of $\mathscr{G}$ is a tighter approximation than the one that results from ignoring the constraints. For dealing with this local polytope, all we need is to be able to express the "smaller" marginal polytopes for the hard factors (cf. Eq. 5.9). We will see in Section 5.3 that those polytopes have often a compact representation.

- **How can we express constraints in first-order logic in a modular way, and run approximate inference algorithms that handle those constraints?** This will also be addressed in Section 5.3, where we introduce a small set of hard constraint factors that has great expressive power. In particular, we will be able to run approximate inference for any problem involving constraints in first-order logic in linear time, with respect to the size of the longest clause.

5.3 An Inventory of Hard Constraint Factors

In this section, we present hard constraint factors that work as building blocks for writing constraints in first-order logic. Each of the factors discussed here performs a logical function, and hence we represent them graphically through *logic gates*. We start by enumerating these factors, and later in Section 5.4 we will show how to compute messages and perform other calculations that are specific to each of them, but which will serve the same purpose—learning and inference in constrained models.

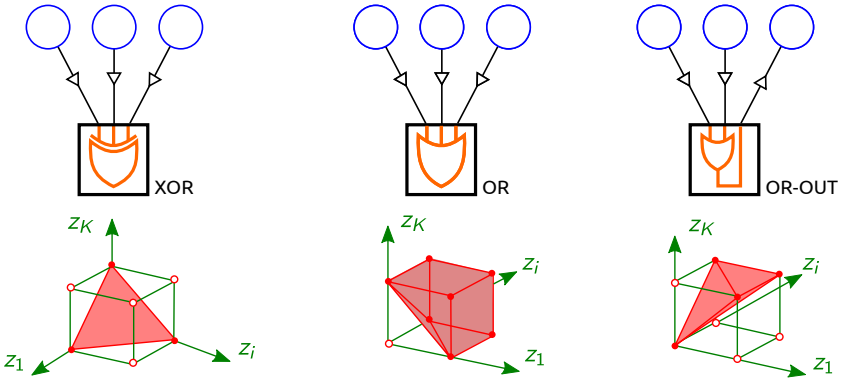


Figure 5.4: Logic factors introduced in this thesis and their marginal polytopes. Left: the marginal polytope of the XOR factor is the probability simplex. Middle: the OR factor has as marginal polytope a faulty hypercube, with one vertex removed. Right: the OR-with-output factor has a more intricate marginal polytope.

5.3.1 One-hot XOR and Uniqueness Quantification

The one-hot XOR factor is linked to $K \geq 1$ binary variables and is defined through the following potential function:

$$\psi_{\text{XOR}}(y_1, \ldots, y_K) := \begin{cases} 1 & \text{if } \exists! k \in \{1, \ldots, K\} \text{ s.t. } y_k = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.23}$$

where the symbol $\exists!$ means "there is one and only one." In words, the potential is zero-valued unless *exactly one* of the inputs (among $y_1, \ldots, y_K$) takes the value 1. The name XOR stems from the fact that, with $K = 2$, the potential function behaves like a logic Exclusive-OR. Hence ψ_{XOR} can be seen as a generalization of Exclusive-OR for $K \geq 2$. The prefix "one-hot" serves to emphasize that this generalization evaluates to 1 if there is precisely one "active" input (where "active" means having a value of 1), avoiding confusion with another commonly used generalization related with parity checks.[3]

The XOR factor appeared in Smith and Eisner (2008) under the name "Exactly1," where it was employed in the context of dependency parsing to constrain each word to have a single head. Since the need of imposing uniqueness constraints is frequent in NLP, it is a very useful factor in practice, as it can express a statement in first-order logic of the form[4]

$$\exists! y : R(y). \tag{5.24}$$

The marginal polytope associated with the XOR factor is, by definition, the convex hull

[3] To be more precise, that alternative definition of Exclusive-OR for $K \geq 2$ evaluates to 1 if there is an odd number of active inputs. It is also a widely used factor in models for error-correcting decoding, such as low-density parity-check codes (Gallager, 1962; Feldman et al., 2005; Richardson and Urbanke, 2008).

[4] It can also be employed for binarizing a categorical variable—in that case, one needs to constrain the variable to take a unique value, and that is done by connecting all possible values to an XOR factor.

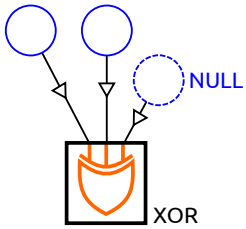


Figure 5.5: A logical constraint "AtMost1" constructed with a XOR factor with a slack variable. Either one of the first two variables is active, or none is, turning on the slack variable.

of the acceptance set, which is a lifted version of the set:

$$\mathcal{Z}_{\text{XOR}} = \text{conv}\left\{\boldsymbol{y} \in \{0,1\}^K \mid \exists! k \in \{1,\ldots,K\} \text{ s.t. } y_k = 1\right\} \tag{5.25}$$

$$= \left\{\boldsymbol{z} \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} z_k = 1\right\}; \tag{5.26}$$

in other words, the marginal polytope of the XOR factor is the *probability simplex*. This is depicted graphically in Figure 5.4.

An XOR factor can also be used to define the function "AtMost1" introduced by Smith and Eisner (2008) (which evaluates to 1 if there is *at most* one active input), by adding one extra input y_{K+1} (which represents a NULL value). This is illustrated in Figure 5.5.

5.3.2 OR and Existential Quantification

The OR factor represents a disjunction of $K \geq 1$ binary variables. It is defined through the following potential function:

$$\psi_{\text{OR}}(y_1,\ldots,y_K) := \begin{cases} 1 & \text{if } \exists k \in \{1,\ldots,K\} \text{ s.t. } y_k = 1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.27}$$

where the symbol $\exists$ has the usual meaning "there is at least one." In other words, the potential is one-valued in all cases except when all the inputs are inactive, in which case it is zero-valued— hence the potential function behaves like a logic OR.

The difference with respect to the XOR factor is that OR imposes existence but not uniqueness. It can be used to represent a statement in first-order logic of the form

$$\exists y : R(y). \tag{5.28}$$

The marginal polytope associated with the OR factor is a lifted version of the set:

$$\mathcal{Z}_{\text{OR}} = \text{conv}\left\{\boldsymbol{y} \in \{0,1\}^K \mid \exists k \in \{1,\ldots,K\} \text{ s.t. } y_k = 1\right\} \tag{5.29}$$

$$= \left\{\boldsymbol{z} \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} z_k \geq 1\right\}; \tag{5.30}$$

geometrically, it is a "faulty" hypercube, *i.e.*, a hypercube which was carved by removing

one vertex (in this case, the origin). This is depicted in Figure 5.4.

Since it behaves like a disjunction function, it is appealing to use an OR factor as a component of a larger network that encodes a more complex logical statement, *e.g.*, involving disjunctions of predicates. On the other hand, it is also useful in isolation: there are many problems in natural language processing and relational learning that make use of categorical variables that can take *one or more* values. An OR factor can be employed to impose this constraint that they take *at least one* value.

5.3.3 Negations and De Morgan's law

We extend the two factors above—as well as the ones we will present in the sequel—to accommodate *negated* inputs. Later, we will see that all computations that involve the original factors can be easily extended to allow negated inputs while reusing the same black box that solves the original problems. The ability to handle negated variables adds a great degree of flexibility. For example, it allows us to handle negated conjunctions (NAND; discussed in the case $K = 2$ by Smith and Eisner 2008):

$$\begin{aligned} \psi_{\text{NAND}}(y_1, \ldots, y_K) &:= \begin{cases} 0 & \text{if } y_k = 1, \forall k \in \{1, \ldots, K\} \\ 1 & \text{otherwise,} \end{cases} \\ &= \psi_{\text{OR}}(\neg y_1, \ldots, \neg y_K) \end{aligned} \tag{5.31}$$

as well as implications (IMPLY):

$$\begin{aligned} \psi_{\text{IMPLY}}(y_1, \ldots, y_K, y_{K+1}) &:= \begin{cases} 1 & \text{if } \left(\bigwedge_{k=1}^{K} y_k\right) \Rightarrow y_{K+1} \\ 0 & \text{otherwise,} \end{cases} \\ &= \psi_{\text{OR}}(\neg y_1, \ldots, \neg y_K, y_{K+1}). \end{aligned} \tag{5.32}$$

This is so because, from De Morgan's laws, we have that $\neg \left(\bigwedge_{k=1}^{K} Q_k(x)\right)$ is equivalent to $\bigvee_{k=1}^{K} \neg Q_k(x)$, and that $\bigwedge_{i=1}^{K} Q_k(x) \Rightarrow R(x)$ is equivalent to $\bigvee_{k=1}^{n} \neg Q_k(x) \vee R(x)$.

Let α be a binary constrained factor with marginal polytope $\mathcal{Z}_\alpha$, and β be a factor obtained from α by negating the kth variable. Then, the marginal polytope associated with the factor β, which we denote by $\mathcal{Z}_\beta$, is a simple symmetric transformation of $\mathcal{Z}_\alpha$:

$$\mathcal{Z}_\beta = \left\{ \boldsymbol{z} \in [0,1]^K \;\middle|\; (z_1, \ldots, z_{k-1}, 1 - z_k, z_{k+1}, \ldots, z_K) \in \mathcal{Z}_\alpha \right\}. \tag{5.33}$$

By induction, this can be generalized to an arbitrary number of negated variables.

5.3.4 Logical Variable Assignments: XOR-with-output and OR-with-output

All cases seen above involve taking a group of existing variables and defining a constraint. Alternatively, we may want to define a new variable (say, y_{K+1}) which is the result of an operation involving other variables (say, $y_1, \ldots, y_K$), as in Figure 5.1 (right). Among other things, this will allow us to deal with "soft constraints," *i.e.*, constraints that can be violated but whose violation will decrease the score by some penalty.

We start with the XOR-with-output factor, which we define as follows:

$$\psi_{\text{XOR}-\text{out}}(y_1,\ldots,y_K,y_{K+1}) := \begin{cases} 1 & \text{if } y_{K+1}=1 \wedge \exists! k \in \{1,\ldots,K\} : y_k = 1 \\ 1 & \text{if } y_{K+1}=0 \wedge \forall k \in \{1,\ldots,K\} : y_k = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{5.34}$$

In words, this factor constrains at most one of the variables $y_1,\ldots,y_K$ to be active; if one is active, it constrains $y_{K+1}=1$; if all are inactive, then it constrains $y_{K+1}=0$. Interestingly, this factor can be expressed using a regular XOR factor *where the last variable is negated*:

$$\psi_{\text{XOR}-\text{out}}(y_1,\ldots,y_K,y_{K+1}) = \psi_{\text{XOR}}(y_1,\ldots,y_K,\neg y_{K+1}). \tag{5.35}$$

Using Eq. 5.33, we then have that the marginal polytope associated with the XOR-with-output factor is a lifted version of the set:

$$\mathcal{Z}_{\text{XOR}-\text{out}} = \left\{ \boldsymbol{z} \in [0,1]^{K+1} \;\middle|\; \sum_{k=1}^{K} z_k = z_{K+1} \right\}. \tag{5.36}$$

Another important logical assignment factor is OR-with-output:

$$\psi_{\text{OR}-\text{out}}(y_1,\ldots,y_K,y_{K+1}) := \begin{cases} 1 & \text{if } y_{K+1} = \bigvee_{k\in\{1,\ldots,K\}} y_k \\ 0 & \text{otherwise.} \end{cases} \tag{5.37}$$

This factor constrains the variable y_{K+1} to indicate the existence (not necessarily uniqueness) of an active variable among $y_1,\ldots,y_K$. It can be used to impose the following statement in first-order logic:

$$T(x) := \exists z : R(x,z). \tag{5.38}$$

Unlike the XOR-with-output case, the OR-with-output factor cannot be built by reusing an OR or XOR factor with some inputs negated.[5]

The marginal polytope associated with the OR factor is a lifted version of the set:

$$\mathcal{Z}_{\text{OR}-\text{out}} = \text{conv}\left\{ \boldsymbol{y} \in \{0,1\}^{K+1} \;\middle|\; y_{K+1} = \bigvee_{k\in\{1,\ldots,K\}} y_k \right\} \tag{5.39}$$

$$= \left\{ \boldsymbol{z} \in [0,1]^{K+1} \;\middle|\; \sum_{k=1}^{K} z_k \geq z_{K+1},\ z_k \leq z_{K+1}, \forall k \in \{1,\ldots,K\} \right\}. \tag{5.40}$$

This is also depicted graphically in Figure 5.4.

[5] It can however be equivalently expressed as the product of $K+1$ OR factors, since we have

$$\begin{aligned} \psi_{\text{OR}-\text{out}}(y_1,\ldots,y_K,y_{K+1}) &= \bigwedge_{k=1}^{K} (y_k \Rightarrow y_{K+1}) \wedge \left(y_{K+1} \Rightarrow \bigvee_{k=1}^{K} y_k \right) \\ &= \left(\prod_{k=1}^{K} \psi_{\text{OR}}(\neg y_k, y_{K+1}) \right) \psi_{\text{OR}}(y_1,\ldots,y_K,\neg y_{K+1}), \end{aligned}$$

The reason we consider $\psi_{\text{OR}-\text{out}}$ is that it may be beneficial to have a larger factor instead of many small ones, as far as we can carry out all the necessary computations using the larger factors.

Soft constraints in first order logic. As mentioned above, factors that represent logical functions with output variables allow us to deal with soft constraints. For example, suppose we want a soft OR constraint involving the variables $Y_1, \ldots, Y_K$, whose violation will affect the score by a penalty q. One needs to do the following:

1. Introduce a new "slack" variable (call it Y_{K+1}) for representing the event that the constraint is violated; assign a log-potential of $-q$ to this variable, which will be the amount of penalty for the violation.

2. Introduce an OR-with-output factor connecting inputs $Y_1, \ldots, Y_K$ and output Y_{K+1}.

5.3.5 AND-with-output and Expressions with Universal Quantifiers

All the factors described so far will be placed in a binary factor graph. Interpreting the values of the variables as memberships in a subset, the corresponding MAP inference problem can be seen an instance of set-valued (or *pseudo-Boolean*) optimization (Boros and Hammer, 2002) with side constraints. We may want to boost the expressive power of the model by introducing score terms that depend on the *conjunction* of several variables (say $y_1, \ldots, y_K$), *i.e.*, of the form $s_{1\ldots K} \times \prod_{k=1}^{K} y_k$. For $K > 1$, such scores are non-linear. An important technique in pseudo-Boolean optimization is *linearization*: one first creates an additional variable (say y_{K+1}) which is constrained to evaluate to the conjunction $\bigwedge_{k=1}^{K} y_k$, and then replace the non-linear term by $s_{1\ldots K} \times y_{K+1}$. The definition of the new variable can be made through another important logical assignment factor, AND-with-output:

$$\psi_{\text{AND}-\text{out}}(y_1, \ldots, y_K, y_{K+1}) := \begin{cases} 1 & \text{if } y_{K+1} = \bigwedge_{k=1}^{K} y_k \\ 0 & \text{otherwise.} \end{cases} \tag{5.41}$$

The AND-with-output factor can be used to impose the following statement in first-order logic:

$$T(x) := \forall z : R(x, z). \tag{5.42}$$

By De Morgan's laws, an AND-with-output factor can be constructed from an OR-with-output by negating all the variables that are linked to it:

$$\psi_{\text{AND}-\text{out}}(y_1, \ldots, y_K, y_{K+1}) = \psi_{\text{OR}-\text{out}}(\neg y_1, \ldots, \neg y_K, \neg y_{K+1}). \tag{5.43}$$

5.4 Computing Messages and Solving Local Subproblems

Now that we have defined constrained factor graphs, characterized their geometry, and introduced logic factors, we turn our attention to *inference algorithms*. We are going to see how message passing and dual decomposition algorithms can also operate in constrained factor graphs. Before discussing the global behaviour of those algorithms (which we will do in Section 5.5), we will describe how they behave *locally*—namely, how they can compute messages or solve local subproblems efficiently.

Recall from Section 4.2 that the cost of computing the factor-to-variable messages for a general factor α is $O(\exp(\deg(\alpha)))$, since it requires a summation/maximization over exponentially many configurations. At first sight, this fact seems to preclude the usage of the hard

constraint factors presented in Section 5.3, for any but a very small value of $K := \deg(\alpha)$. It turns out, however, that all those factors have potential functions which are extremely *sparse*, and that this sparsity can be exploited algorithmically. We are going to show that, for all these cases, the factor-to-variable messages can be computed in time $O(\deg(\alpha))$—*i.e.*, the dependency in the factor degree is *linear* (rather than exponential). This extends previous findings of Duchi et al. (2007) and Smith and Eisner (2008), among others, who have obtained similar results for other combinatorial factors.

We go farther by providing a general recipe for the computation of messages in *arbitrary factors* (Algorithms 6–7), which depend only on oracles for the following tasks:

- An oracle that, given the potential function of the factor, computes the local partition function and marginals (for the sum-product case);
- An oracle that, given the potential function of the factor, computes the mode and the max-marginals (for the max-product case).

5.4.1 Sum-Product Messages

Recall that, in the sum-product variant of the BP algorithm, the factor beliefs and the factor-to-variable messages take the form (cf. Eqs. 4.8 and 4.10):

$$M_{\alpha\to i}(y_i) \propto \sum_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{\substack{j\in\mathcal{N}(\alpha)\\ j\neq i}} M_{j\to\alpha}(y_j) \tag{5.44}$$

$$b_\alpha(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i\in\mathcal{N}(\alpha)} M_{i\to\alpha}(y_i). \tag{5.45}$$

A close look into Eq. 5.45 reveals that the variable-to-factor messages $M_{i\to\alpha}(y_i)$ participate in that expression as potential functions participate in a factored distribution; the case of Eq. 5.44 is similar, but the probability function is integrated over the value assignments consistent with y_i, resembling the definition of a marginal probability. We make this observation formal through Proposition 5.4, a key result which provides a way of computing messages, beliefs and entropies given the ability of computing marginals and the partition function.

Proposition 5.4 *Let $\mathcal{G}_\alpha$ be the α-subgraph of $\mathcal{G}$, and consider a probability distribution factoring according to $\mathcal{G}_\alpha$:*

$$P_\omega(\boldsymbol{y}_\alpha) \propto \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i\in\mathcal{N}(\alpha)} \exp(\omega_i(y_i)). \tag{5.46}$$

Let $Z_\alpha(\omega)$ be the corresponding partition function, and let $\mu_i(y_i;\omega) := P_\omega(y_i)$, for every $i \in \mathcal{N}(\alpha)$, denote the variable marginals.

Now consider an execution of the sum-product (loopy) BP algorithm run on the original graph $\mathcal{G}$, and define ω as given by the variable-to-factor messages:

$$\omega_i(y_i) := \log M_{i\to\alpha}(y_i). \tag{5.47}$$

Then we have the following:

1. *At any point of the algorithm, we have the following expression for the factor beliefs (Eq. 4.10): $b_\alpha(\boldsymbol{y}_\alpha) = P_\omega(\boldsymbol{y}_\alpha)$;*

2. *Upon convergence, the variable beliefs (Eq. 4.9) satisfy* $b_i(y_i) = \mu_i(y_i; \omega)$;

3. *At any point of the algorithm, the factor-to-variable messages can be computed as:*

$$M_{\alpha \to i}(y_i) \propto \frac{\mu_i(y_i;\omega)}{M_{i\to\alpha}(y_i)} = \mu_i(y_i;\omega)\exp(-\omega_i(y_i)). \tag{5.48}$$

4. *Upon convergence, the entropy of the factor,* $H_\alpha(b_\alpha(.))$*, can be computed as:*

$$\begin{aligned} H_\alpha(b_\alpha(.)) &= \log Z_\alpha(\omega) - \sum_{i\in\mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i;\omega) \log M_{i\to\alpha}(y_i) \\ &= \log Z_\alpha(\omega) - \sum_{i\in\mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i;\omega)\omega_i(y_i). \end{aligned} \tag{5.49}$$

Proof. The first statement is immediate from the definition of factor beliefs (see Eq. 4.10) along with Eq. 5.47. The second statement is a consequence of the calibration equation of the factor and marginal beliefs (Eq. 4.11): given that $b_\alpha(\boldsymbol{y}_\alpha) = P_\omega(\boldsymbol{y}_\alpha)$, we have that $b_i(y_i)$ corresponds to the marginals of that distribution. The third statement can be obtained by rewriting Eq. 5.44 as

$$M_{\alpha\to i}(y_i) \propto M_{i\to\alpha}(y_i)^{-1} \sum_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j\in\mathcal{N}(\alpha)} M_{j\to\alpha}(y_j), \tag{5.50}$$

and observing that the right hand side equals $M_{i\to\alpha}(y_i)^{-1}\mu_i(y_i)$. Finally, the fourth statement is a consequence of the first one and the conjugate duality relation between the log-partition function and the negative entropy, expressed in Proposition 5.3. ∎

Proposition 5.4 tells us that all the necessary computations in the sum-product (loopy) BP algorithm which involve factor α (namely, computing the outgoing messages, the beliefs, and the contribution of this factor to the Bethe entropy approximation) can be performed by invoking a black box routine that just computes the partition function and the marginals for the α-subgraph. Hence, we need only to be concerned with this black-box routine, which is, naturally, a property of the factor α.

We next revisit the hard constraint factors introduced in Section 5.3 and, in each case, obtain closed-form expressions for the factor-to-variable message ratios in terms of their variable-to-factor counterparts. We also derive closed-form expressions for the marginals, partition function, and entropies. Since in all these factors the variables are binary-valued, and the formulae for computing the messages are up to a normalization constant, only the following *message ratios* matter:

$$m_{\alpha\to i} := M_{\alpha\to i}(1)/M_{\alpha\to i}(0), \qquad m_{i\to\alpha} := M_{i\to\alpha}(1)/M_{i\to\alpha}(0). \tag{5.51}$$

The formulae for computing messages for a particular factor α can thus be seen as a *transfer function* that maps from messages $m_{i\to\alpha}$, to messages $m_{\alpha\to i}$, $\forall i \in \mathcal{N}(\alpha)$.

Since some of the derivations are a bit tedious, we leave all the details to Appendix C.1 and just summarize the results below.

The One-hot XOR Factor. To simplify notation and make things clearer, we assume the factor is linked to variables 1 to K, and replace the subscript α by XOR. We refer to the full assignment $\boldsymbol{y}_\alpha$ simply as $\boldsymbol{y}$. Recall that we have $\omega_i(y_i) := \log M_{i\to\text{XOR}}(y_i)$; since only ratios matter, we will deal with $m_{i\to\text{XOR}} := \exp(\omega_i(1) - \omega_i(0))$. Note that we have $M_{i\to\text{XOR}}(0) = (1 + m_{i\to\text{XOR}})^{-1}$ and $M_{i\to\text{XOR}}(1) = m_{i\to\text{XOR}}(1 + m_{i\to\text{XOR}})^{-1}$.

In Appendix C.1, we derive the following expressions for the partition function and the marginals:

$$Z_{\text{XOR}}(\omega) = \sum_{i=1}^{K} m_{i\to\text{XOR}} \times \prod_{i=1}^{K}(1 + m_{i\to\text{XOR}})^{-1} \tag{5.52}$$

$$\mu_i(1;\omega) = \frac{m_{i\to\text{XOR}}}{\sum_{k=1}^{K} m_{k\to\text{XOR}}} \tag{5.53}$$

$$\mu_i(0;\omega) = \frac{\sum_{k=1,k\neq i}^{K} m_{k\to\text{XOR}}}{\sum_{k=1}^{K} m_{k\to\text{XOR}}}. \tag{5.54}$$

From Proposition 5.4, this leads to the following message updates:

$$\boxed{m_{\text{XOR}\to i} = \left(\sum_{k=1,k\neq i}^{K} m_{k\to\text{XOR}}\right)^{-1}.} \tag{5.55}$$

The message updates in Eq. 5.55 have an intuitive interpretation in terms of competition between variables: if all variables (apart from the ith) have sent messages that express a large belief in an inactive state (*i.e.*, low value of $m_{k\to\text{XOR}}$), then $m_{\text{XOR}\to i}$ will be set to a large value, pushing the ith value to "take the lead" in being the active variable of the one-hot XOR. On the other hand, if some other variable k "thinks" she should be active (through a large value of $m_{k\to\text{XOR}}$), then it will "shut off" the competing variables by making each ith variable ($i \neq k$) receive a low value of $m_{\text{XOR}\to i}$.

The entropy can expressed in terms of the beliefs as follows (see Appendix C.1):

$$\boxed{H_{\text{XOR}}(b_{\text{XOR}}(.)) = -\sum_{i=1}^{K} b_i(1)\log b_i(1).} \tag{5.56}$$

This, again, has an intuitive interpretation. From the marginal polytope equations (Eq. 5.25), we know that for the XOR factor we must have $\sum_{i=1}^{K} b_i(1) = 1$, hence the vector $(b_i(1))_{i=1}^{K}$ represents a probability distribution of a random variable with K possible outcomes—the same as the number of allowed configurations of the XOR factor. Then, Eq. 5.56 tells us that the entropy H_{XOR} equals the entropy of that distribution.

In sum, we can compute all the relevant quantities (factor-to-variable messages, log-partition function, marginals and factor entropy) in time *linear* in K (rather than exponential).

The OR Factor. We next turn to the OR factor. In Appendix C.1, we derive expressions for the log-partition function and ratio of marginals:

$$Z_{\text{OR}}(\omega) = 1 - \prod_{i=1}^{K}(1 + m_{i\to\text{OR}})^{-1}; \quad \frac{\mu_i(1;\omega)}{\mu_i(0;\omega)} = m_{i\to\text{OR}}\left(1 - \prod_{k=1,k\neq i}^{K}(1 + m_{k\to\text{OR}})^{-1}\right)^{-1}. \tag{5.57}$$

From Proposition 5.4, we obtain the message updates:

$$\boxed{m_{\text{OR}\to i} = \left(1 - \prod_{k=1,k\neq i}^{K} (1 + m_{k\to\text{OR}})^{-1}\right)^{-1}}. \tag{5.58}$$

As in the XOR case, the message updates in Eq. 5.58 have an interpretation in terms of competition between variables. However, competition is less sharp. As before, if all variables (apart from the ith) have sent messages that give large likelihood to an inactive state (*i.e.*, low value of $m_{k\to\text{OR}}$), then $m_{\text{OR}\to i}$ will be set to a large value, pushing the ith value to be active. However, if some other variable $k \neq i$ believes she should be active (through a large value of $m_{k\to\text{OR}}$), then the product in Eq. 5.58 will pushed to zero, which will yield $m_{\text{OR}\to i} \approx 1$, a non-informative message. In other words, if there is evidence that one of the variables (say the kth) connected to a OR-factor is active, this will have a neutral effect on $m_{\text{OR}\to i}$ for every $i \neq k$, since the factor does not "care" anymore about the state of the other variables.

The entropy is given by:

$$\begin{aligned} H_{\text{OR}}(b_{\text{OR}}(.)) \;=\; & \log\left(1 - \prod_{i=1}^{K}(1 + m_{i\to\text{OR}})^{-1}\right) + \sum_{i=1}^{K}\log(1 + m_{i\to\text{OR}}) \\ & - \sum_{i=1}^{K}\left(1 + m_{i\to\text{OR}} - \prod_{k=1,k\neq i}^{K}(1 + m_{k\to\text{OR}})^{-1}\right)^{-1} \times m_{i\to\text{OR}}\log m_{i\to\text{OR}}. \end{aligned} \tag{5.59}$$

Unlike the XOR case, it does not seem straightforward to write this expression as a function of the marginals only; yet, the entropy can still be computed from the incoming messages in time *linear* in K. In sum, all the relevant quantities (factor-to-variable messages, log-partition function, marginals and factor entropy) can again be computed in linear time.

The OR-with-output Factor. In Appendix C.1, we derive expressions for the log-partition function associated with the OR-with-output factor:

$$Z_{\text{OR-out}}(\omega) \;=\; M_{K+1\to\text{OR-out}}(1)\left(1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right) + \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0). \tag{5.60}$$

where $M_{i\to\text{OR}-\text{out}}(1) = m_{i\to\text{OR}-\text{out}}(1 + m_{i\to\text{OR}-\text{out}})^{-1}$, by definition of $m_{i\to\text{OR}-\text{out}}$. The ratio of marginals is:

$$\frac{\mu_i(1;\omega)}{\mu_i(0;\omega)} \;=\; \begin{cases} m_{i\to\text{OR-out}}\left(1 - (1 - m^{-1}_{K+1\to\text{OR-out}})\prod_{k=1,k\neq i}^{K}(1 + m_{k\to\text{OR-out}})^{-1}\right)^{-1} & \text{if } 1 \leq i \leq K \\ m_{K+1\to\text{OR-out}} \times \left(\prod_{k=1}^{K}(1 + m_{k\to\text{OR-out}}) - 1\right) & \text{if } i = K+1. \end{cases} \tag{5.61}$$

This leads to the following message updates:

$$\boxed{m_{\text{OR-out}\to i} = \left(1 - (1 - m^{-1}_{K+1\to\text{OR-out}}) \prod_{k=1,k\neq i}^{K} (1 + m_{k\to\text{OR-out}})^{-1}\right)^{-1},} \tag{5.62}$$

for $i \leq K$, and

$$\boxed{m_{\text{OR-out}\to K+1} = \prod_{k=1}^{K}(1 + m_{k\to\text{OR-out}}) - 1.} \tag{5.63}$$

The message updates in Eqs. 5.62–5.63 have again an intuitive interpretation. In Eq. 5.62, if there is evidence that the output variable is likely to be active (large $m_{K+1\to\text{OR-out}}$), then the expression becomes close to the message passing in the OR factor. On the contrary, if it is likely that y_{K+1} is zero ($m_{K+1\to\text{OR-out}}$ close to zero), then the message sent from the factor to each variable $i \leq K$ is going to be close to zero as well. Eq. 5.63 is also very intuitive: if at least one of the messages $m_{k\to\text{OR-out}}$ is large, then $m_{\text{OR-out}\to K+1}$ will also become large, pushing the output variable to be active; only if all the input variables are likely to be inactive (every $m_{k\to\text{OR-out}}$ close to zero) will the message $m_{\text{OR-out}\to K+1}$ be a small value.

Like in the previous factors, the entropy can be computed via:

$$\begin{aligned} H_{\text{OR-out}}(b_{\text{OR-out}}(.)) &= \log Z_{\text{OR-out}}(\omega) - \sum_{i=1}^{K+1} \mu_i(1;\omega) \log m_{i\to\text{OR-out}} \\ &\quad + \sum_{i=1}^{K+1} \log(1 + m_{i\to\text{OR-out}}). \end{aligned} \tag{5.64}$$

Again, all the relevant quantities (factor-to-variable messages, log-partition function, marginals and factor entropy) can be computed in linear time.

5.4.2 Max-Product Messages

We now turn to the *max-product variant* of the BP algorithm, for which we derive results similar to those obtained in the sum-product variant. In max-product BP, the factor-to-variable messages take the form (cf. Eq. 4.13):

$$M_{\alpha\to i}(y_i) \propto \max_{\boldsymbol{y}_\alpha \sim y_i} \left(\psi_\alpha(\boldsymbol{y}_\alpha) \prod_{j\neq i} M_{j\to\alpha}(y_j)\right); \tag{5.65}$$

that is, they are analogous to the sum-product case, but where the summation is replaced with a maximization. The next proposition is the analogous of Proposition 5.4 for the max-product variant of the BP algorithm, and provides a way of computing messages given the ability of computing *max-marginals*.[6]

Proposition 5.5 *Let $\mathcal{G}_\alpha$ be the α-subgraph of $\mathcal{G}$, and consider a probability distribution factoring*

[6]Most of the results in this section may be obtained from the ones in the previous section by taking the "zero temperature limit" of the sum-product case.

according to $\mathscr{G}_\alpha$*:*

$$P_\omega(\boldsymbol{y}_\alpha) \propto \widetilde{P}_\omega(\boldsymbol{y}_\alpha) := \psi_\alpha(\boldsymbol{y}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \exp(\omega_i(y_i)). \tag{5.66}$$

Above, we have denoted by $\widetilde{P}_\omega(\boldsymbol{y}_\alpha)$ *the unnormalized distribution. Let*

$$P^*_\alpha(\omega) := \max_{\boldsymbol{y}_\alpha} \widetilde{P}_\omega(\boldsymbol{y}_\alpha) \tag{5.67}$$

be the "mode" of this unnormalized distribution, and let

$$\nu_i(y_i; \omega) := \max_{\boldsymbol{y}_\alpha \sim y_i} \widetilde{P}_\omega(\boldsymbol{y}_\alpha), \tag{5.68}$$

for every $i \in \mathcal{N}(\alpha)$ *and* $y_i \in \mathcal{Y}_i$*, denote the variable max-marginals.*

Now consider an execution of the max-product (loopy) BP algorithm run on the original graph $\mathscr{G}$*, and define* ω *as given by the variable-to-factor messages:*

$$\omega_i(y_i) := \log M_{i \to \alpha}(y_i). \tag{5.69}$$

Then we have the following:

1. *At any point of the algorithm, we have the following expression for the factor beliefs (Eq. 4.10):* $b_\alpha(\boldsymbol{y}_\alpha) \propto \widetilde{P}_\omega(\boldsymbol{y}_\alpha)$*;*

2. *Upon convergence, the variable beliefs (Eq. 4.9) satisfy* $b_i(y_i) \propto \nu_i(y_i; \omega)$*;*

3. *At any point of the algorithm, the factor-to-variable messages can be computed as:*

$$M_{\alpha \to i}(y_i) \propto \frac{\nu_i(y_i; \omega)}{M_{i \to \alpha}(y_i)} = \nu_i(y_i; \omega) \exp(-\omega_i(y_i)). \tag{5.70}$$

Proof. Analogous to the proof of Proposition 5.4. ∎

Similarly to the sum-product case, Proposition 5.5 is of great significance as it states that all the necessary computations in the max-product (loopy) BP algorithm which involve factor α can be performed by invoking a black box routine that computes the MAP and the max-marginals for the α-subgraph.

We next revisit the hard constraint factors defined in Section 5.3 and, in each case, obtain closed-form expressions for the factor-to-variable max-product message ratios in terms of their variable-to-factor counterparts (see Eq. 5.51). We use throughout the compact notation $[K]$ to denote the set of integers between 1 and K, *i.e.*, $[K] := \{1, \ldots, K\}$. Again, we just summarize the results below, leaving the details of the derivations to Appendix C.2.

The One-hot XOR Factor. In Appendix C.2, we obtained expressions for the mode and max-marginals associated with the one-hot XOR factor:

$$P^*_{\text{XOR}}(\omega) = \max_{i\in[K]} \left(m_{i\to\text{XOR}} \times \prod_{i=1}^{K} (1 + m_{i\to\text{XOR}})^{-1} \right), \tag{5.71}$$

$$\nu_i(1;\omega) = m_{i\to\text{XOR}} \prod_{k=1}^{K} (1 + m_{k\to\text{XOR}})^{-1}, \tag{5.72}$$

$$\nu_i(0;\omega) = \max_{j\in[K]\setminus\{i\}} \left(m_{j\to\text{XOR}} \times \prod_{k=1}^{K} (1 + m_{i\to\text{XOR}})^{-1} \right). \tag{5.73}$$

From Proposition 5.5, we have that the message updates are given by:

$$\boxed{m_{\text{XOR}\to i} = \left(\max_{k\neq i} m_{k\to\text{XOR}} \right)^{-1}}. \tag{5.74}$$

The message updates in Eq. 5.74 have an intuitive interpretation in terms of competition between variables, just like in the sum-product case (cf. Eq. 5.55). The only difference is that the summation in Eq. 5.55 became a maximization in Eq. 5.74.

The OR Factor. In Appendix C.2, we derived expressions for the mode and ratio of max-marginals associated with the OR factor:

$$P^*_{\text{OR}}(\omega) = \left(\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR}}(y_k) \right) \times \min\left\{ 1, \max_{k\in[K]} \frac{M_{k\to\text{OR}}(1)}{M_{k\to\text{OR}}(0)} \right\}, \tag{5.75}$$

$$\frac{\nu_i(1;\omega)}{\nu_i(0;\omega)} = m_{i\to\text{OR}} \times \max\left\{ 1, \min_{k\in[K]\setminus\{i\}} m^{-1}_{k\to\text{OR}} \right\}. \tag{5.76}$$

As a consequence, we obtain the following simple expression for the max-product message updates, invoking Proposition 5.5:

$$\boxed{m_{\text{OR}\to i} = \max\left\{ 1, \min_{k\neq i} m^{-1}_{k\to\text{OR}} \right\}}. \tag{5.77}$$

The OR-with-output Factor. The mode in the OR-with-output factor is

$$P^*_{\text{OR-out}}(\omega) = \max\left\{ \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0), \quad M_{K+1\to\text{OR-out}}(1) \times \left(\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k) \right) \times \min\left\{ 1, \max_{k\in[K]} \frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)} \right\} \right\}. \tag{5.78}$$

We now turn to the ratio of max-marginals. For $i \leq K$, we have:

$$\frac{\nu_i(1;\omega)}{\nu_i(0;\omega)} = m_{i\to\text{OR-out}} \times \min\left\{ \begin{array}{l} m_{K+1\to\text{OR-out}} \times \prod_{\substack{k=1\\k\neq i}}^{K} \max\{1, m_{k\to\text{OR-out}}\}, \\ \max\left\{1, \min_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}^{-1}\right\} \end{array} \right\}. \quad (5.79)$$

For $i = K+1$, we have:

$$\frac{\nu_{K+1}(1;\omega)}{\nu_{K+1}(0;\omega)} = m_{K+1\to\text{OR-out}} \times \left(\prod_{k=1}^{K} \max\{1, m_{k\to\text{OR-out}}\}\right) \times \min\left\{1, \max_{k\in[K]} m_{k\to\text{OR-out}}\right\} \quad (5.80)$$

This leads to the following simple expression for the max-product message updates:

$$\boxed{m_{\text{OR-out}\to i} = \min\left\{ m_{K+1\to\text{OR-out}} \prod_{\substack{k=1\\k\neq i}}^{K} \max\{1, m_{k\to\text{OR-out}}\}, \max\left\{1, \min_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}^{-1}\right\} \right\}} \quad (5.81)$$

for $i \leq K$, and

$$\boxed{m_{\text{OR-out}\to K+1} = \left(\prod_{k=1}^{K} \max\{1, m_{k\to\text{OR-out}}\}\right) \times \min\left\{1, \max_{k\in[K]} m_{k\to\text{OR-out}}\right\}.} \quad (5.82)$$

5.4.3 Negations

As mentioned in Section 5.3.3, the ability of endowing the factors seen above with *negated* variables adds a lot of expressive power, enabling us to construct formulas in first order logic by using the factors XOR, OR and OR-with-output as components.

To use factors with negations, we need to be able to compute with them. We are going to see that all quantities of interest in the sum-product and max-product algorithms—messages, beliefs, entropies, and modes—can easily be computed when some of the variables are negated, by invoking the same black box procedures that compute the log-partition function and the marginals (in the sum-product case) or the mode and the max-marginals (in the max-product case), albeit with some pre- and post-processing. This is a consequence as the following proposition, which holds for any factor (soft or hard):

Proposition 5.6 *Let α be a factor linked to K variables, and let β be a factor obtained from α by modifying the potential function as follows:*

$$\psi_\beta(y_1, \ldots, y_K) := \psi_\alpha(\sigma_1(y_1), \ldots, \sigma_K(y_K)), \quad (5.83)$$

where each $\sigma_i : \mathcal{Y}_i \to \mathcal{Y}_i$ is a permutation function that relabels the variables. Let $\boldsymbol{\theta} = (\theta_i(.))_{i=1}^K$ be a vector of variable log-potentials, and consider the scrambled vector

$$\boldsymbol{\theta}'_i(y_i) = \boldsymbol{\theta}_i(\sigma_i^{-1}(y_i)), \quad (5.84)$$

where $\sigma_i^{-1} : \mathcal{Y}_i \to \mathcal{Y}_i$ *is the inverse permutation of* σ_i.

1. *Denote by* $Z_\alpha(\boldsymbol{\theta})$ *(resp.* $Z_\beta(\boldsymbol{\theta})$*) the value of the partition function using factor potential* ψ_α *(resp.* ψ_β*) and variable log-potentials* $\boldsymbol{\theta}$*. Then we have the following:*

$$Z_\beta(\boldsymbol{\theta}) = Z_\alpha(\boldsymbol{\theta}'). \tag{5.85}$$

2. *Furthermore, denote by* $\mu_{i;\alpha}(y_i;\boldsymbol{\theta})$ *(resp.* $\mu_{i;\beta}(y_i;\boldsymbol{\theta})$*) the marginal probability of the event* $Y_i = y_i$ *for factor* α *(resp.* β*) with variable log-potentials* $\boldsymbol{\theta}$*. Then we have:*

$$\mu_{i;\beta}(y_i;\boldsymbol{\theta}) = \mu_{i;\alpha}(\sigma_i(y_i);\boldsymbol{\theta}'). \tag{5.86}$$

3. *Denote by* $P^*_\alpha(\boldsymbol{\theta})$ *(resp.* $P^*_\beta(\boldsymbol{\theta})$*) the "mode" of the unnormalized distribution using factor potential* ψ_α *(resp.* ψ_β*) and variable log-potentials* $\boldsymbol{\theta}$*. Then we have the following:*

$$P^*_\beta(\boldsymbol{\theta}) = P^*_\alpha(\boldsymbol{\theta}'). \tag{5.87}$$

4. *Furthermore, denote by* $\nu_{i;\alpha}(y_i;\boldsymbol{\theta})$ *(resp.* $\nu_{i;\beta}(y_i;\boldsymbol{\theta})$*) the max-marginal associated with the event* $Y_i = y_i$ *for factor* α *(resp.* β*) with variable log-potentials* $\boldsymbol{\theta}$*. Then we have:*

$$\nu_{i;\beta}(y_i;\boldsymbol{\theta}) = \nu_{i;\alpha}(\sigma_i(y_i);\boldsymbol{\theta}'). \tag{5.88}$$

Proof. Items 1 and 2 are immediate from the definition of partition function and marginals:

$$\begin{aligned} Z_\beta(\boldsymbol{\theta}) &= \sum_{y_1,\ldots,y_K} \psi_\beta(y_1,\ldots,y_K) \prod_{i=1}^{K} \exp(\boldsymbol{\theta}_i(y_i)) \\ &= \sum_{y_1,\ldots,y_K} \psi_\alpha(\sigma_1(y_1),\ldots,\sigma_K(y_K)) \prod_{i=1}^{K} \exp(\boldsymbol{\theta}'_i(\sigma_i(y_i))) \\ &= Z_\alpha(\boldsymbol{\theta}'), \end{aligned} \tag{5.89}$$

and

$$\begin{aligned} \mu_{i;\beta}(y_i;\boldsymbol{\theta}) &= \sum_{\substack{y_1,\ldots,y_{i-1} \\ y_{i+1},\ldots,y_K}} \psi_\beta(y_1,\ldots,y_i,\ldots,y_K) \prod_{k=1}^{K} \exp(\boldsymbol{\theta}_k(y_k)) \\ &= \sum_{\substack{y_1,\ldots,y_{i-1} \\ y_{i+1},\ldots,y_K}} \psi_\alpha(\sigma_1(y_1),\ldots,\sigma_i(y_i),\ldots,\sigma_K(y_K)) \prod_{k=1}^{K} \exp(\boldsymbol{\theta}'_k(\sigma_k(y_k))) \\ &= \mu_{i;\alpha}(\sigma_i(y_i);\boldsymbol{\theta}'). \end{aligned} \tag{5.90}$$

Items 3 and 4 and analagous, replacing summations with maximizations. ∎

The next corollary follows from applying Proposition 5.6 to a hard constraint factor α with binary variables, for which the only permutations are the identity and the negation. For convenience, we state it for the case of a reduced parametrization (see Section 5.2, Eq. 5.7),

in the form

$$P(\boldsymbol{y}) \propto \exp\left(\sum_{i\in\mathcal{V}} s_i y_i\right) Q_\alpha(\boldsymbol{y}), \tag{5.91}$$

where each $y_i \in \{0,1\}$.

Corollary 5.7 *Let α be a hard constraint factor linked to K variables, with acceptance set $\mathcal{S}_\alpha$, and let β be a factor obtained from α by negating the ith input variable:*

$$\mathcal{S}_\beta := \left\{\boldsymbol{y} \in \{0,1\}^K \mid (y_1,\dots,\neg y_i,\dots,y_K) \in \mathcal{S}_\alpha\right\}. \tag{5.92}$$

Let $s := (s_k)_{k=1}^K$ be a vector of variable log-potentials in the reduced parametrization.

1. *Denote by $Z_\alpha(s) := \sum_{\boldsymbol{y}\in\mathcal{S}_\alpha} \exp\left(\sum_{k=1}^K s_k y_k\right)$ and $Z_\beta(s) := \sum_{\boldsymbol{y}\in\mathcal{S}_\beta} \exp\left(\sum_{k=1}^K s_k y_k\right)$ the value of the partition function in each of the factors, when parameterized by s. Then we have the following:*
$$Z_\beta(s_1,\dots,s_i,\dots,s_K) = \exp(s_i) \times Z_\alpha(s_1,\dots,-s_i,\dots,s_K). \tag{5.93}$$

2. *Furthermore, denote by $z_{k;\alpha}(s)$ (resp. $z_{k;\beta}(s)$) the marginal probability of the event $Y_k = 1$ for factor α (resp. β) with variable log-potentials s. Then we have:*
$$z_{k;\beta}(s_1,\dots,s_i,\dots,s_K) = \begin{cases} 1 - z_{i;\alpha}(s_1,\dots,-s_i,\dots,s_K) & \text{if } k = i \\ z_{k;\alpha}(s_1,\dots,-s_i,\dots,s_K) & \text{if } k \neq i. \end{cases} \tag{5.94}$$

3. *Denote by $P^*_\alpha(s) := \max_{\boldsymbol{y}\in\mathcal{S}_\alpha} \exp\left(\sum_{k=1}^K s_k y_k\right)$ and $P^*_\beta(s) := \max_{\boldsymbol{y}\in\mathcal{S}_\beta} \exp\left(\sum_{k=1}^K s_k y_k\right)$ the value of the unnormalized mode in each of the factors, when parameterized by s. Then we have the following:*
$$P^*_\beta(s_1,\dots,s_i,\dots,s_K) = \exp(s_i) \times P^*_\alpha(s_1,\dots,-s_i,\dots,s_K). \tag{5.95}$$

4. *Furthermore, denote by $\nu_{k;\alpha}(l;s)$ (resp. $\nu_{k;\beta}(l;s)$) the marginal probability of the event $Y_k = l$, for factor α (resp. β) with variable log-potentials s. Then we have:*
$$\nu_{l;\beta}(s_1,\dots,s_i,\dots,s_K) = \begin{cases} \nu_{i;\alpha}(1-l;s_1,\dots,-s_i,\dots,s_K) & \text{if } k = i \\ \nu_{k;\alpha}(l;s_1,\dots,-s_i,\dots,s_K) & \text{if } k \neq i. \end{cases} \tag{5.96}$$

A direct consequence of Corollary 5.7 is that, for both the sum-product and max-product cases, the factor-to-variable messages at factor β can be computed as follows:

$$m_{\beta\to k}(\omega_1,\dots,\omega_K) = \begin{cases} 1/m_{\alpha\to i}(\omega'_1,\dots,\omega'_K) & \text{if } k = i \\ m_{\alpha\to k}(\omega'_1,\dots,\omega'_K) & \text{if } k \neq i. \end{cases} \tag{5.97}$$

where $\omega_k := m_{k\to\beta}$ for $k \in \{1,\dots,K\}$, and ω' is defined as $\omega'_k = \omega_k$ for $k \neq i$, and $\omega'_i = 1/\omega_i$. By invoking this corollary several times, we end up with a procedure for any number of negated variables. There are three steps:

1. A preprocessing step in which one inverts the incoming messages $m_{k\to\beta}$ that correspond to the negated variables;

2. The main step in which one computes the outgoing messages $m_{\alpha\to k}$ using a black box for the factor α without negations;

3. A postprocessing step in which one inverts back the outgoing messages that correspond to the negated variables, yielding $m_{\beta\to k}$.

Example 5.2 (AND-with-output.) *The AND-with-output factor emulates the logical expression $y_{K+1} := \bigwedge_{k=1}^{K} y_k$. This can be represented as a OR-with-output factor where all variables are negated (since we have $\neg y_{K+1} := \bigvee_{k=1}^{K} \neg y_k$ by de Morgan's rules). Hence the sum-product messages can be obtained from the ones in Eqs. 5.62–5.63, by invoking Corollary 5.7:*

$$m_{\text{AND}-\text{out}\to i} = 1 - (1 - m_{K+1\to\text{AND}-\text{out}}) \prod_{k=1,k\neq i}^{K} (1 + m_{k\to\text{AND}-\text{out}}^{-1})^{-1}, \tag{5.98}$$

for $i \leq K$, and

$$m_{\text{AND}-\text{out}\to K+1} = \left(\prod_{k=1}^{K} (1 + m_{k\to\text{AND}-\text{out}}^{-1}) - 1 \right)^{-1}., \tag{5.99}$$

and similarly for the max-product messages.

Algorithms 6 and 7 summarize the procedure of computing sum-product and max-product messages, respectively, in a hard constraint factor linked to binary variables, some possibly negated. In both cases, two procedures are called as subroutines: for the sum-product case, the procedures are ComputePartitionFunction, a black-box that computes Z from a vector of log-potentials, and ComputeMarginals, another black-box that computes the marginal vector $\boldsymbol{z}$ from the same vector of log-potentials. For the max-product case, the oracles are ComputeMode, a black-box that computes the mode P^* from a vector of log-potentials, and ComputeMaxMarginals, another black-box that computes the vector of max-marginals $\boldsymbol{\nu}$ from the same vector of log-potentials.[7]

If one of the goals is to approximate the value of global partition function $Z(\boldsymbol{\theta})$, then we can invoke the Eq. 5.21 in Proposition 5.3 and compute it from the Bethe approximation of the entropy. The latter can be computed upon convergence of the loopy BP algorithm, from the variable entropies and the factor entropies (Eq. 4.45). Note that Algorithm 6 provides the factor entropy as output. We will exploit this fact in Chapter 8, where we introduce a new learning algorithm that needs to evaluate the partition function, in addition to computing its gradient.

5.4.4 Generalized Message-Passing Algorithms

In the last section, we provided a recipe for running sum-product and max-product loopy BP in constrained graphs (Algorithms 6–7). That recipe can be extended in a straighforward manner to other variants of message-passing algorithms, such as "fractional" BP algorithms

[7] Note that, in line 19 of both algorithms, and since we are dealing with constrained factors, there is a chance that the denominators $1 - z_i$ (or $\nu_i(0)$) vanish. In that case, the message ratio $m_{\beta\to i}$ should be interpreted as being $+\infty$—this means $M_{\beta\to i} = 1$ and $M_{\beta\to i} = 0$.

Algorithm 6 Computing Sum-Product Messages in a Binary Hard Factor

input:

- factor β, which is made of α by negating some variables,
- set of negated variables $\mathcal{A}$,
- incoming messages $m_{i\to\beta}$.

Form a vector of log-potentials $\boldsymbol{s} = (s_i)_{i=1}^K$, with $s_i := \log m_{i\to\beta}$
Initialize bias $b = 0$
for $i = 1$ **to** K **do**
  if $i \in \mathcal{A}$ **then**
    Set $s'_i \leftarrow -s_i$ (flip log-potential)
    Increment bias $b \leftarrow b + s_i$
  else
    Set $s'_i \leftarrow s_i$
  end if
end for

Compute $Z_\alpha \leftarrow \textsc{ComputePartitionFunction}(\boldsymbol{s}'; \alpha)$
Compute $\boldsymbol{z} \leftarrow \textsc{ComputeMarginals}(\boldsymbol{s}'; \alpha)$
for $i = 1$ **to** K **do**
  if $i \in \mathcal{A}$ **then**
    $z_i \leftarrow 1 - z_i$
  end if
  Compute outgoing message $m_{\beta\to i} \leftarrow \frac{z_i}{1-z_i} \exp(-s_i)$
end for

Compute partition function $Z_\beta \leftarrow Z_\alpha \times \exp(b)$
Compute entropy $H_\beta \leftarrow \log Z_\beta - \boldsymbol{s} \cdot \boldsymbol{z}$
output:

- outgoing messages $m_{\beta\to i}$,
- current beliefs $b_i(1) := z_i$,
- factor entropy H_β.

(Wiegerinck and Heskes, 2003; Weiss et al., 2007; Hazan and Shashua, 2010), tree-reweighted algorithms (Wainwright et al., 2005b,a; Kolmogorov, 2006), and block coordinate descent algorithms (Werner, 2007; Globerson and Jaakkola, 2008), some of which discussed in Sections 4.5–4.6. Those generalized message-passing algorithms pass messages of the form

$$M_{\alpha\to i}(y_i) \quad \propto \quad \left(M_{i\to\alpha}^{\rho_{i,\alpha}-\rho_i}(y_i) \bigoplus_{\boldsymbol{y}_\alpha \sim y_i} \psi_\alpha^{\rho_\alpha}(\boldsymbol{y}_\alpha) \prod_{j\in\mathcal{N}(\alpha), j\neq i} M_{j\to\alpha}^{\rho_{j,\alpha}}(y_j) \right)^{1/\rho_\alpha} \tag{5.100}$$

where $\bigoplus$ denotes either max or $\sum$, and the exponents ρ_i, ρ_α, and $\rho_{i,\alpha}$ are specific to each algorithm (in standard BP, they are all set to 1). By renormalizing the messages, it is simple to express those messages as a function of the marginals (in the sum-product case) or the max-marginals (in the max-product case).

Algorithm 7 Computing Max-Product Messages in a Binary Hard Factor

input:

- factor β, which is made of α by negating some variables,
- set of negated variables $\mathcal{A}$,
- incoming messages $m_{i \to \beta}$.

Form a vector of log-potentials $\boldsymbol{s} = (s_i)_{i=1}^K$, with $s_i := \log m_{i \to \beta}$
Initialize bias $b = 0$
for $i = 1$ **to** K **do**
  if $i \in \mathcal{A}$ **then**
    Set $s'_i \leftarrow -s_i$ (flip log-potential)
    Increment bias $b \leftarrow b + s_i$
  else
    Set $s'_i \leftarrow s_i$
  end if
end for

Compute $P^*_\alpha \leftarrow \textsc{ComputeMode}(\boldsymbol{s}'; \alpha)$
Compute $\boldsymbol{\nu} \leftarrow \textsc{ComputeMaxMarginals}(\boldsymbol{s}'; \alpha)$
for $i = 1$ **to** K **do**
  if $i \in \mathcal{A}$ **then**
    Swap $\nu_i(0)$ and $\nu_i(1)$
  end if
  Compute outgoing message $m_{\beta \to i} \leftarrow \frac{\nu_i(1)}{\nu_i(0)} \exp(-s_i)$
end for

Compute mode $P^*_\beta \leftarrow P^*_\alpha \times \exp(b)$
output:

- outgoing messages $m_{\beta \to i}$,
- current beliefs $b_i := \nu_i$,
- factor mode P^*_β.

5.4.5 Local Subproblems in Dual Decomposition

In Section 4.6.2, we have described another technique for LP-MAP inference, called dual decomposition, with uses the projected subgradient algorithm (Algorithm 5). That algorithm has a similar flavor as message passing algorithms. One can regard the local MAP estimates (the $\boldsymbol{\mu}$-variables) and the Lagrange multipliers (the $\boldsymbol{\lambda}$-variables) as *messages* that are exchanged between variables and factors. The analogue of the problem of computing factor-to-variable messages is that of solving the local MAP subproblems, *i.e.*, the $\boldsymbol{\mu}$-updates in Algorithm 5. As seen in Section 4.6.2, all that is necessary is a subroutine ComputeMAP that computes the MAP assignment of the α-subgraph induced by each factor α. In Section 5.4.2, we have already provided expressions for computing the modes of factors XOR, OR, and OR-with-output; and Corollary 5.7 allows extending those expressions for other factors encoding first-order logic expressions.

5.5 Approximate Inference in Constrained Factor Graphs

We now provide a *global* perspective on how approximate inference works with the machinery described above for computing messages and solving local subproblems. To have a sense about the quality of the approximations in graphs with cycles, we consider problems that can be formulated as inference in constrained graphs, but which are simple enough so that exact solutions can be derived analytically and compared with the approximations. This will give insight about the strengths and limitations of the approximate methods.

We start by showing how any arbitrary factor graph with multi-valued variables can be *binarized*, a process that has the side effect of adding hard constraints and cycles. We then show that this transformation preserves the LP-MAP approximation (used in MAP inference), but *not* the Bethe approximation (used in marginal inference). The two main points that will be illustrated are the following:

- For a wide range of problems, one can find a representation as a binary constrained factor graph that does not affect MAP inference (a positive result);
- Even for very simple problems, marginal inference and entropy computation can be strongly affected by using that representation (a negative result).

5.5.1 Binarization of a Factor Graph

So far, we have focused on binary constrained factor graphs. However, there are many problems for which it is natural to use *multi-valued* random variables, for which $|\mathcal{Y}_i| > 2$. How to extend our techniques to perform inference in such multi-valued factor graphs? One possible way is to *binarize* the graph, as we describe next.

Let $\mathcal{G} = (\mathcal{V} \cup \mathcal{F}, \mathcal{E}')$ be a unconstrained factor graph, where the associated random variables $Y_1, \ldots, Y_{|\mathcal{V}|}$ are multi-valued (*i.e.*, $|\mathcal{Y}_i| \geq 2$, for each $i \in \mathcal{V}$). We will derive a *binary* constrained factor graph $\mathcal{G}' = (\mathcal{V}' \cup \mathcal{F}' \cup \mathcal{H}', \mathcal{E}')$ which is equivalent to $\mathcal{G}$, in the sense that any probability distribution represented by $\mathcal{G}$ can be represented by $\mathcal{G}'$ and vice-versa.

Recall the definition of the local polytope of $\mathcal{G}$ in Eq. 4.40:

$$\text{LOCAL}(\mathcal{G}) = \left\{ \boldsymbol{\mu} \in \mathbb{R}^R \;\middle|\; \begin{array}{ll} \sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) = 1, & \forall i \in \mathcal{V} \\ \mu_i(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha), & \forall i \in \mathcal{V},\ y_i \in \mathcal{Y}_i,\ \alpha \in \mathcal{N}(i) \\ \mu_\alpha(\boldsymbol{y}_\alpha) \geq 0, & \forall \alpha \in \mathcal{F},\ \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha \end{array} \right\}. \quad (5.101)$$

A close look reveals that the constraints in Eq. 5.101 are essentially the same as those that define the marginal polytopes of the XOR factor (Eq. 5.25) and the XOR-with-output factor (Eq. 5.36); therefore, an equivalent set of constraints can be obtained by introducing binary variables for each configuration of Y_i and $\mathbf{Y}_\alpha$ and linking them to XOR and XOR-with-output factors. This leads to the following procedure for constructing $\mathcal{G}'$:

1. For any $i \in \mathcal{V}$ and $y_i \in \mathcal{Y}_i$, introduce a binary variable U_{i,y_i} and add the corresponding variable node to $\mathcal{G}'$. Set the log-potential function on this variable to 0 on $U_{i,y_i} = 0$ and to $\theta_i(y_i)$ on $U_{i,y_i} = 1$.
2. For any $\alpha \in \mathcal{F}$ and $\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha$, introduce a binary variable $U_{\alpha,\boldsymbol{y}_\alpha}$ and add the corresponding variable node to $\mathcal{G}'$. Set the log-potential function on this variable to 0 on $U_{\alpha,\boldsymbol{y}_\alpha} = 0$ and to $\theta_\alpha(\boldsymbol{y}_\alpha)$ on $U_{\alpha,\boldsymbol{y}_\alpha} = 1$.

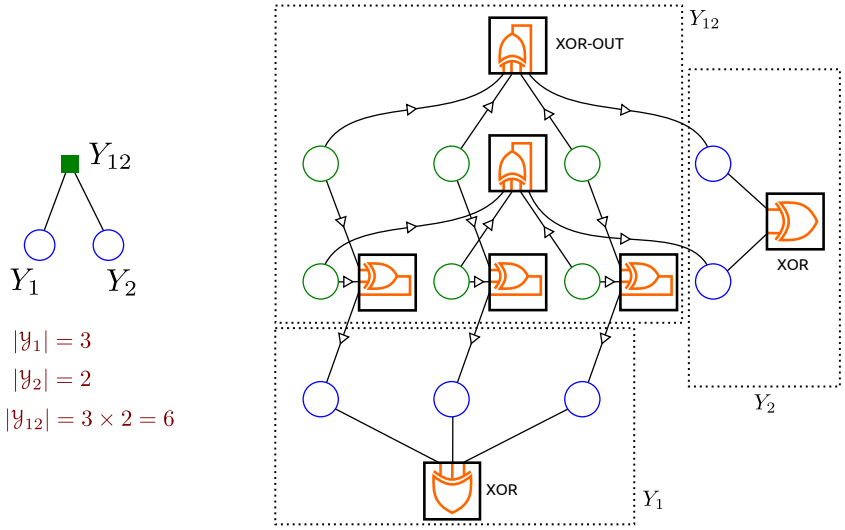


Figure 5.6: Left: A simple factor graph with multi-valued variables. Right: A transformation of that graph into a binary constrained factor graph, yielding 11 variables and 7 factors.

3. For any $i \in \mathcal{V}$, add a XOR factor to $\mathcal{G}'$ connecting the variables $\{U_{i,y_i} \mid y_i \in \mathcal{Y}_i\}$. From Eq. 5.25, we have that the corresponding marginal polytope imposes the constraints:

$$\sum_{y_i \in \mathcal{Y}_i} \mu_i(y_i) = 1, \quad \mu_i(y_i) \geq 0, \ \forall y_i \in \mathcal{Y}_i, \tag{5.102}$$

the first of which corresponding to the original *normalization* constraints in $\text{LOCAL}(\mathcal{G})$.

4. For any $\alpha \in \mathcal{F}$, $i \in \mathcal{N}(\alpha)$, and $y_i \in \mathcal{Y}_i$, add a XOR-with-output factor to $\mathcal{G}'$ connecting the input variables $\{U_{\alpha,\boldsymbol{y}_\alpha} \mid \boldsymbol{y}_\alpha \sim y_i\}$ to the output variable U_{i,y_i}. From Eq. 5.36, we have that the marginal polytope associated with this factor imposes the constraints:

$$\mu_i(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha), \quad \mu_\alpha(\boldsymbol{y}_\alpha) \geq 0, \ \forall \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha, \tag{5.103}$$

the first of which corresponding to the original *marginalization* constraints in $\text{LOCAL}(\mathcal{G})$, and the second implying the *non-negativity* constraints.

Figure 5.6 illustrates the above procedure when the original graph is a simple pairwise MRF with only two variables.

Proposition 5.8 *For any choice of log-potentials $\boldsymbol{\theta}$, the probability distributions induced by $\mathcal{G}$ and $\mathcal{G}'$ are the same. Moreover, each iteration of the loopy BP algorithm (as well as other message-passing and dual decomposition algorithms) has the same asymptotic cost in the two graphs.*

Proof. The XOR factors guarantee that, for every $i \in \mathcal{V}$, exactly one of the variables U_{i,y_i} is active, and the XOR-with-output factors guarantee that the variables $U_{\alpha,\boldsymbol{y}_\alpha}$ and U_{i,y_i} are consistent; it is straightforward to show that the set of admissible outputs in $\mathcal{G}'$ is in one-to-one correspondence with $\mathcal{Y}_1 \times \ldots \times \mathcal{Y}_{|\mathcal{V}|}$, which is the output set of $\mathcal{G}$. In addition, the way we have set the log-potentials makes it clear that any configuration in $\mathcal{G}'$ will have the

same score as the corresponding configuration in $\mathcal{G}$. This proves the first claim. Let us now compare the runtimes of one round of message passing in both graphs. The original factor graph $\mathcal{G}$ has $|\mathcal{V}|$ variables and $|\mathcal{F}|$ factors. For each factor α, a total of $\deg(\alpha)$ messages $M_{\alpha\to i}(.)$ need to be passed, each with a computation cost of $O(|\mathcal{Y}_\alpha|)$. As for the variable-to-factor messages, the total computation cost is $\sum_{i\in\mathcal{V}} \deg(i) \times |\mathcal{Y}_i|$. Hence the total runtime of a round of message passing is

$$O\left(\sum_{i\in\mathcal{V}} \deg(i) \times |\mathcal{Y}_i| + \sum_{\alpha\in\mathcal{F}} \deg(\alpha) \times |\mathcal{Y}_\alpha|\right). \tag{5.104}$$

The binary constrained factor graph $\mathcal{G}'$ has $\sum_i |\mathcal{Y}_i| + \sum_\alpha |\mathcal{Y}_\alpha|$ variables, $|\mathcal{V}|$ XOR factors, and $\sum_\alpha \sum_{i\in\mathcal{N}(\alpha)} |\mathcal{Y}_i|$ XOR-with-output factors. The total number of messages is $\sum_i |\mathcal{Y}_i|$ (for the XOR factors) and $\sum_\alpha \deg(\alpha) \times |\mathcal{Y}_\alpha| + \sum_\alpha \sum_{i\in\mathcal{N}(\alpha)} |\mathcal{Y}_i|$ (for the XOR-with-output ones). Each of these messages has a constant computation cost (since the cost per XOR or XOR-with-output is linear in the factor degree). As for the variable-to-factor messages, the total computation cost is $\sum_{i\in\mathcal{V}}(1+\deg(i)) \times |\mathcal{Y}_i| + \sum_{\alpha\in\mathcal{F}} \deg(\alpha) \times |\mathcal{Y}_\alpha|$. We then have a total runtime per iteration of message passing of $O\left(2\sum_{i\in\mathcal{V}}(1+\deg(i)) \times |\mathcal{Y}_i| + 2\sum_{\alpha\in\mathcal{F}} \deg(\alpha) \times |\mathcal{Y}_\alpha|\right)$. Hence, although there is some overhead, the two graphs yield the same asymptotic cost per iteration. ■

5.5.2 Local Polytope Approximation and LP-MAP Inference

Let us turn to the LP-MAP problem in the binarized graph. A consequence of Proposition 5.8 is that the marginal polytopes of $\mathcal{G}$ and $\mathcal{G}'$ are the same (up to a lifting operation). The next proposition goes farther by establishing that this also happens for the *local* polytopes:

Proposition 5.9 *The local polytopes* $\text{LOCAL}(\mathcal{G})$ *and* $\text{LOCAL}(\mathcal{G}')$ *are the same (up to lifting). This has the following chain of implications:*

1. *The LP-MAP problems in* $\text{LOCAL}(\mathcal{G})$ *and* $\text{LOCAL}(\mathcal{G}')$ *have the same solution.*
2. *If $\mathcal{G}$ does not have cycles, then the local polytope approximation* $\text{LOCAL}(\mathcal{G}')$ *is tight (even if $\mathcal{G}'$ has cycles!) and the LP-MAP problem on $\mathcal{G}'$ is guaranteed to have integer solutions* (i.e., *it yields the true MAP).*
3. *The projected subgradient method for dual decomposition (Section 4.6.2) leads to the same solution, when run on* $\text{LOCAL}(\mathcal{G})$ *or* $\text{LOCAL}(\mathcal{G}')$.
4. *The block coordinate message passing algorithms discussed in Section 4.6.1 (MPLP, max-sum diffusion, and TRW-S), when converging and satisfying weak agreement conditions both on* $\text{LOCAL}(\mathcal{G})$ *or* $\text{LOCAL}(\mathcal{G}')$, *lead to the same solution.*

Proof. The main claim follows directly from the construction of $\mathcal{G}'$. The following claims are immediate. We used the fact that the projected subgradient method for dual decomposition converges to the solution of LP-MAP, and that MPLP, max-sum diffusion and TRW-S, when converging and satisfying weak agreement conditions, also yield that solution. ■

Note that the equivalence established in Proposition 5.9 would not be trivial, had we not examined the local polytopes of both graphs. In general, the constrained graph $\mathcal{G}'$ has cycles, even when $\mathcal{G}$ is a very simple graph without cycles, such as an edge connecting two variables

(as in Figure 5.6). Those cycles, however, turn out not to affect the polytope approximation and the MAP inference algorithms mentioned above.

Example 5.3 (Context-free parsing.) *We are going to see how context-free grammars can be binarized. Recall the definition of a context-free grammar* $(\Lambda, \Sigma, \mathcal{P}, S)$*, in Definition 2.1. Suppose we want to parse a sentence with* L *words using this grammar. In Eqs. 2.10–2.11, we considered sets of parts for anchored constituents and anchored production rules:*

$$\mathcal{R}_{\text{s}} := \left\{ \langle X, i, j \rangle \;\middle|\; \begin{array}{c} X \in \Lambda, \\ 1 \leq i \leq j \leq L \end{array} \right\}, \tag{5.105}$$

$$\mathcal{R}_{\text{p}} := \left\{ \langle X, Y, Z, i, j, k \rangle \;\middle|\; \begin{array}{c} \langle X \to YZ \rangle \in \mathcal{P}, \\ 1 \leq i \leq k < j \leq L \end{array} \right\} \bigcup \left\{ \langle X, i \rangle \;\middle|\; \begin{array}{c} \langle X \to x_i \rangle \in \mathcal{P}, \\ 1 \leq i \leq L \end{array} \right\}. \tag{5.106}$$

Taskar et al. (2004b) has shown that the marginal polytope associated with this context-free grammar and this sentence is defined by the following set of equations:[8]

$$\textbf{\textit{Non-negative marginals:}} \quad z_r(y_r) \geq 0, \quad \forall r \in \mathcal{R}_{\text{s}} \cup \mathcal{R}_{\text{p}}, \tag{5.107}$$

$$\textbf{\textit{Normalization:}} \quad z_{\langle S,1,L \rangle} = 1, \tag{5.108}$$

$$\textbf{\textit{Inside constraints:}} \quad z_{\langle X,i,j \rangle} = \sum_{Y,Z,k} z_{\langle X,Y,Z,i,k,j \rangle}, \quad \forall \langle X, i, j \rangle \in \mathcal{R}_{\text{s}}, \tag{5.109}$$

$$\textbf{\textit{Outside constraints:}} \quad z_{\langle X,i,j \rangle} = \sum_{Y,Z,k} z_{\langle Y,X,Z,i,j,k \rangle} + \sum_{Y,Z,k} z_{\langle Y,Z,X,k,i-1,j \rangle}, \quad \forall \langle X, i, j \rangle \in \mathcal{R}_{\text{s}}, \tag{5.110}$$

where the sums above are over the sets of non-terminals and indexes that yield valid productions according to the grammar. Essentially, these constraints express the inside-outside equations in the dynamic program for context-free parsing. A close look into Eqs. 5.107–5.110 reveals that all these constraints can be expressed by XOR and XOR-with-output factors, in a binary constrained factor graph $\mathcal{G}$ *whose variables are the possible constituents and production rules. There are two kinds of factors:* inside factors*, which express the constraints in Eq. 5.109, and* outside factors*, which express the constraints in Eq. 5.110. These are drawn in Figure 5.7. By construction, the local polytope of* $\mathcal{G}$ *coincides with the marginal polytope associated with the grammar—therefore, the solutions of the LP-MAP inference problem are integer, and correspond to valid parse trees. In other words, LP-MAP inference on this graph is* exact*, and equivalent to context free parsing.*[9] *Since there are no fractional vertices, we have from Proposition 5.2 we conclude that* $\text{LOCAL}(\mathcal{G}) = \text{MARG}(\mathcal{G})$*. This raises the question of whether* $\mathcal{G}$ *has cycles.*

It turns out that $\mathcal{G}$ *has cycles even for very simple instances: the simplest counter-example is for a three-word sentence and a complete grammar with a single non-terminal (*i.e.*, a grammar for bracketing)—see Figure 5.7 for an illustration of this example. The existence of cycles motivated some authors to consider a broader representational formalism,* case-factor diagrams *(McAllester et al.,*

[8] Actually, in Taskar et al. (2004b), the marginal polytope was presented without proof. A formal proof has been derived later by Michael Collins (personal communication).

[9] Note that even though LP-MAP inference algorithms run on this graph have guarantees of getting the most likely parse, their cost per iteration is not better than the CKY algorithm. However, this representation of phrase-based parsing may be useful if used as a starting point to add more global features and constraints into the model. In Chapter 7 we use a similar rationale for *dependency* parsing.

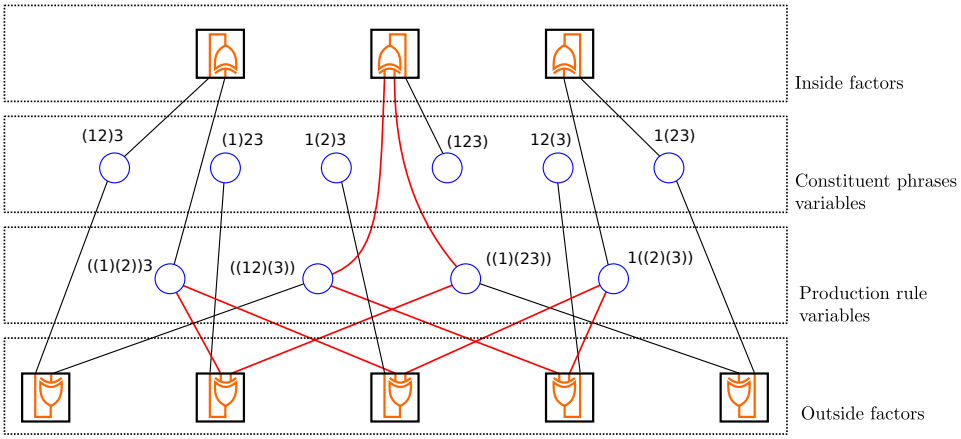


Figure 5.7: Representation of context-free parsing as a binary constrained factor graph with XOR and XOR-with-output factors, for a sentence with three words, and a bracketing grammar. In red, a cycle in this graph, of length 8.

2008). Our example shows that cycles are not a major reason for concern if we are only interested in MAP inference.

5.5.3 Bethe Entropy Approximation and Marginal Inference

The previous section established a "positive" result: the binarization procedure of Section 5.5.1 *preserves* the local polytope and hence the LP-MAP inference problem. We now turn our attention to marginal inference. Recall the connection described in Section 4.5 between sum-product loopy BP and the Bethe entropy approximation H_{Bethe}. One could be led to conjecture that the Bethe approximation of the entropy is also preserved by the binarization procedure. Unfortunately, this does *not* hold even in very simple graphs, as indicated by the following counter-example.

Example 5.4 *Let $\mathcal{G}$ be a very simple graph comprised of one factor linked to two variables Y_1 and Y_2, each taking values in $\mathcal{Y}_1 = \mathcal{Y}_2 = \{1, \ldots, L\}$. The true entropy is*

$$H(\boldsymbol{\mu}) = -\sum_{y_1, y_2} \mu_{12}(y_1, y_2) \log(\mu_{12}(y_1, y_2)). \tag{5.111}$$

Consider now the binarization of $\mathcal{G}$, which we denote by $\mathcal{G}'$, and let us compute its Bethe entropy approximation using Eq. 4.45. Each of the $2L$ variable nodes associated with the variable-value pairs in the original graph has a local entropy given by:

$$H_{i,y_i}(\mu_i(y_i)) = -\mu_i(y_i) \log(\mu_i(y_i)) - (1 - \mu_i(y_i)) \log(1 - \mu_i(y_i)), \tag{5.112}$$

for $i \in \{1, 2\}$; each of these variables participates in two factors, hence these entropies have counting number $1 - 2 = -1$. Each of the L^2 variable nodes associated with the edge-values in the original

graph has a local entropy given by:

$$H_{12,y_{12}}(\mu_{12}(y_1,y_2)) = -\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2)) - (1-\mu_{12}(y_1,y_2))\log(1-\mu_{12}(y_1,y_2)); \tag{5.113}$$

since each of these variables participates in two factors, their counting number is also $1-2=-1$. *The contribution of the variable nodes to the Bethe approximation is thus*

$$\begin{aligned}&\sum_{i\in\{1,2\}}\sum_{y_i}(\mu_i(y_i)\log(\mu_i(y_i)) + (1-\mu_i(y_i))\log(1-\mu_i(y_i)))\\ &+\sum_{y_1,y_2}(\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2)) + (1-\mu_{12}(y_1,y_2))\log(1-\mu_{12}(y_1,y_2)))\,.\end{aligned} \tag{5.114}$$

As for the factors, we have that each of the two XOR factors connected with the variable-value pairs in the original graph has an entropy (cf. 5.56):

$$H_i(\mu_i(.)) = -\sum_{y_i}\mu_i(y_i)\log(\mu_i(y_i)) \qquad \text{for } i\in\{1,2\}; \tag{5.115}$$

as for the 2L *XOR-with-output factors connecting the edge-values with a variable-value pair, they have entropies (cf. 5.56, and Corollary 5.7):*

$$\begin{aligned}H_{1,y_1}(\mu_1(y_1),\mu_{12}(y_1,.)) &= -\sum_{y_2}\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2)) - (1-\mu_1(y_1))\log(1-\mu_1(y_1)),\\ H_{2,y_2}(\mu_2(y_2),\mu_{12}(.,y_2)) &= -\sum_{y_1}\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2)) - (1-\mu_2(y_2))\log(1-\mu_2(y_2)).\end{aligned}$$

Hence the total factor contribution is:

$$\begin{aligned}-\sum_{i\in\{1,2\}}\sum_{y_i}\mu_i(y_i)\log(\mu_i(y_i)) - 2\sum_{y_1,y_2}\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2))\\ -\sum_{i\in\{1,2\}}\sum_{y_i}(1-\mu_i(y_i))\log(1-\mu_i(y_i)).\end{aligned} \tag{5.116}$$

Summing Eqs. 5.114 and 5.116 yields:

$$H_{\text{Bethe}}(\boldsymbol{\mu}) = -\sum_{y_1,y_2}\mu_{12}(y_1,y_2)\log(\mu_{12}(y_1,y_2)) - \sum_{y_1,y_2}(1-\mu_{12}(y_1,y_2))\log(1-\mu_{12}(y_1,y_2)), \tag{5.117}$$

which differs from the true entropy by the amount:

$$H_{\text{Bethe}}(\boldsymbol{\mu}) - H(\boldsymbol{\mu}) = -\sum_{y_1,y_2}(1-\mu_{12}(y_1,y_2))\log(1-\mu_{12}(y_1,y_2)). \tag{5.118}$$

In Martins et al. (2010f, extended version, Appendix B), we show another example in which the original graph $\mathscr{G}$ is free of cycles—which guarantees that marginal inference is exact with the sum-product algorithm—whereas the same algorithm on the binarized graph $\mathscr{G}'$ fails miserably.

5.6 Future Work

There are many possible avenues for future research in constrained structured prediction. One of them is to extend this formalism for broader or related models, such as *case-factor diagrams* (McAllester et al., 2008), *AND/OR search spaces* (Dechter and Mateescu, 2007), and *sum-product networks* (Poon and Domingos, 2011). Case-factor diagrams, for example, are a framework that subsumes pairwise MRFs and probabilistic context-free grammars. Not much is known about their geometry, and it is not known if the duality and variational perspectives of factor graphs can be extended to this broader class of models. Even though the results that we have established in this chapter suggest that binary constrained factor graphs are already very expressive and can be used for MAP inference with probabilistic context-free grammars, no analogue result exists for marginal inference. It is conceivable that the geometric properties of case-factor diagrams can shed some light on the problem, and contribute toward better entropy approximations. We plan to address this in future work.

In fact, obtaining good entropy approximations for constrained graphical models is an important open problem that deserves further attention. There has been a lot of research for the unconstrained case (Wiegerinck and Heskes, 2003; Weiss et al., 2007; Hazan and Shashua, 2010). Of particular interest are convex approximations and upper bounds of the partition function, such as the ones underlying tree-reweighted BP (Wainwright et al., 2005b).

Interestingly, we can draw a connection between entropy approximations and the posterior regularization framework recently proposed by Ganchev et al. (2010), which involves posterior expectation constraints. Although the rationale is totally different from ours—the idea there is to modify the Expectation-Maximization (EM) algorithm by introducing prior knowledge in the form of constraints on the posterior marginals—their constrained E-step can be rewritten in the following form:

$$\begin{aligned} &\text{maximize } \boldsymbol{w}^\top \mathbf{F}(x)\boldsymbol{\mu} + H_{\text{PR}}(\boldsymbol{\mu}) \\ &\quad \text{w.r.t. } \boldsymbol{\mu} \in \text{MARG}(\mathcal{G}_{\text{u}}) \\ &\quad\quad \text{s.t. } \boldsymbol{\mu}_\beta \in \text{MARG}(\mathcal{G}|_\beta), \text{for each } \beta \in \mathcal{H}. \end{aligned} \tag{5.119}$$

Above, $\mathcal{H}$ is a set of hard constraint factors, where for each factor $\beta \in \mathcal{H}$, the posterior constraint associated with that factor is represented by the polytope $\text{MARG}(\mathcal{G}|_\beta)$; $\mathcal{G}_{\text{u}}$ is the original unconstrained factor graph; and the entropic term in the objective, $H_{\text{PR}}(\boldsymbol{\mu})$, regards only the unconstrained part of the graph, ignoring the constraints. The global optimization problem can be seen as an approximate marginal inference problem, with an entropy approximation of the form

$$H_{\text{PR}}(\boldsymbol{\mu}) \approx \sum_{i \in \mathcal{V}} c_i H_i(\boldsymbol{\mu}_i) + \sum_{\alpha \in \mathcal{F} \cup \mathcal{H}} c_\alpha H_\alpha(\boldsymbol{\mu}_\alpha), \tag{5.120}$$

where the counting numbers are $c_i = 1 - |\mathcal{N}(i) \setminus \mathcal{H}|$, $c_\alpha = 1$ for $\alpha \in \mathcal{F}$, and $c_\beta = 0$ for $\beta \in \mathcal{H}$. Note that this entropy approximation in distinct from Bethe's approximation (cf. Eq. 4.45). It would be interesting to see how the message-passing algorithms used in factor graphs can help solve the E-step in the posterior regularization framework, or vice-versa.

Another problem is that of developing efficient algorithms for computing messages. In

the scope of sum-product BP, we have seen that this problem is intimately related with the problem of computing marginals and the partition function, which are instances of *counting* problems. In some cases, efficient procedures exist for computing these quantities. We have seen an example in Section 2.3, which involves inverting a matrix and computing a determinant. Other instances exist in the literature for which fast Fourier transforms enable to carry out this computation efficiently (Liao et al., 2005).

The connection established between constrained graphical models and LP-MAP inference, as well as the possible binarization of tractable graphical models, opens up the possibility of determining whether certain linear programs have integral solutions based on the topology of the graph. This kind of criterion could be an alternative to other properties that establish sufficient conditions for the integrality of LPs, such as the total unimodularity of the constraint matrix (Schrijver, 2003), which is usually difficult to check, or the property of total dual integrality (Martin et al., 1990), which applies to directed hypergraphs and is related to dynamic programming algorithms. This connection can be fruitful in providing better conditions to characterize integral LPs: it is easy to constuct factor graphs without cycles where the marginal polytope equations do not yield totally unimodular constraint matrices, and yet admit integral solutions (since the graph is free of cycles). Some examples are: a single OR-with-output factor with three input variables (and one output), and a single unconstrained factor with three binary variables. Conversely, it is also simple to find examples of totally unimodular matrices that correspond to graphs with loops. Therefore, neither of these characterizations subsumes the other.

On a more practical side, future work will also contemplate how the achievements described in this chapter can be integrated with current systems for constrained prediction, such as Markov logic networks. Current software implementations, *e.g.*, Alchemy,[10] include alternative inference algorithms, such as lifted BP and sampling methods. Lifted methods are orthogonal to the procedures described here for computing messages, and it is likely that the two can be used together to tackle hard constraint factors.

5.7 Conclusions

In this chapter, we provided a theoretical study of *constrained factor graphs*, a unified representational framework for constrained structured prediction. This framework is useful in constrained conditional models and grounded Markov logic networks, with many potential applications to NLP. Our results lead to a better theoretical understanding of these models.

We show how to handle arbitrary first-order logic constraints by introducing hard constraint factors that work as building blocks for expressing such constraints. We provided a concise representation of the marginal polytopes associated with these factors, and we derived closed-form expressions for computing messages, marginals, max-marginals, the partition function, the factor entropies, and the mode. We have shown how all these quantities can be computed in linear time, contrasting with dense factors, for which the analogous operations have *exponential* runtime. We also saw that all that is necessary for computing sum-product and max-product messages in arbitrary factors is an oracle that computes marginals, max-marginals, and the partition function.

[10] Available at http://alchemy.cs.washington.edu/.

In addition, we have shown how any factor graph can be *binarized* through a procedure that preserves the marginal and local polytope of the original graph. If the original graph does not have cycles, then LP-MAP inference in the binarized graph is exact, even if cycles *are* present in the latter. This reinforces the idea that the tractability of MAP inference holds in much broader scenarios than just tree-structured graphs, corroborating previous results (Weiss and Freeman, 2001b,a; Bayati et al., 2005; Huang and Jebara, 2007). On the other hand, the entropy approximations are not preserved by this binarization procedure, and they can be poor in the binarized graph, even for very simple examples. This suggests that approximate marginal inference in constrained models may be a much more difficult task.

The results obtained in this chapter will be invoked for presenting contributions in later chapters. For example, Chapter 6 will propose the AD^3 algorithm, dealing with the hard constraint factors introduced here. In Chapter 8, we will reuse the variational representation of the log-partition function, and discuss the impact of approximate inference in the learning problem.

Chapter 6

Alternating Directions Dual Decomposition

In this chapter, we introduce a new LP-MAP inference algorithm called AD^3 (for *Alternating Directions Dual Decomposition*). At its heart lies a method well-known in the optimization community called the "alternating direction method of multipliers" (ADMM),[1] an augmented Lagrangian method which was first proposed in the 1970s (Glowinski and Marroco, 1975; Gabay and Mercier, 1976) and has seen a recent surge of interest in machine learning and signal processing (Afonso et al., 2010; Mota et al., 2010; Goldfarb et al., 2010; Boyd et al., 2011). We show how AD^3 is able to solve the LP-MAP problem more efficiently than other methods, by allying the effectiveness of ADMM with the modularity of dual decomposition methods. Our main contributions in this chapter are:

- We derive the AD^3 algorithm and establish its convergence properties, blending classical and newer results (Glowinski and Le Tallec, 1989; Eckstein and Bertsekas, 1992; He and Yuan, 2011; Wang and Banerjee, 2012).
- We derive efficient procedures for projecting onto the marginal polytopes of the hard constraint factors introduced in Section 5.3. This paves the way for using AD^3 in problems with declarative constraints. Up to a logarithmic term, the cost of projecting is asymptotically the same as that of passing messages.
- We introduce a new active set method for solving the local quadratic problem for arbitrary factors. This method requires only an oracle that computes the local MAP (the same requirement as the projected subgradient method discussed in Section 4.6.2), which allows plugging in combinatorial machinery.
- We show how AD^3 can be wrapped into a branch-and-bound procedure to retrieve the true MAP, rendering the method *exact*.

The AD^3 algorithm was originally introduced in Martins et al. (2010d, 2011a),[2] where some of the convergence results were established, along with procedures for projecting on the marginal polytopes of hard constraint factors. We include in this chapter a substantial

[1] Also known by other names, such as "method of alternating directions" and "Douglas-Rachford splitting."
[2] The algorithm was called DD-ADMM in those papers.

amount of new material: the $O(1/\epsilon)$ rate of convergence of AD^3, the active set method for general factors, and the branch-and-bound procedure for obtaining the exact MAP.

6.1 Motivation and Related Work

In the current chapter, we address the problem of LP-MAP inference, defined in Section 4.6. There is a string of previous work concerning algorithms for solving that problem (Wainwright et al., 2005a; Kolmogorov, 2006; Werner, 2007; Globerson and Jaakkola, 2008; Ravikumar et al., 2010). Our focus is on *dual decomposition* methods, of which the projected subgradient algorithm of Komodakis et al. (2007); Komodakis and Paragios (2009), described in Section 4.6.2, is a paradigmatic example. We first discuss the intrinsic limitations of that method and suggest ways of overcoming them, pointing to the literature. This will pave the way for the introduction of our AD^3 algorithm, which is the subject of the following sections.

A nice property of the projected subgradient algorithm is that it is guaranteed to converge to the solution of the LP-MAP problem; this is a clear advantage over other message-passing algorithms, including the coordinate descent algorithms discussed in Section 4.6.1, which can get stuck at corners. However, this convergence can be painfully slow when the number of overlapping components is large (cf. Figure 7.8 in the following chapter). This should not be surprising: being a "consensus algorithm," it attempts to reach a consensus among all overlapping components; the larger this number, the harder it gets to achieve the desired consensus. This chapter addresses the following question:

Can we get faster consensus with a different dual decomposition algorithm?

To gain intuition about the problem, it is instructive to look at two things:

1. **The objective function in the dual problem** (Eq. 4.58). This function is *non-smooth*—this is why we need "subgradients" instead of "gradients." It is well-known that non-smooth optimization lacks some of the nice properties of its smooth counterpart. For example, there is no guarantee of monotonic improvement in the objective after each subgradient update (see Bertsekas et al. 1999, p. 611). Convergence is ensured by using a schedule that progressively decreases the stepsize, leading to slower convergence rates (in general, $O(1/\epsilon^2)$ iterations are required for ϵ-accuracy).

2. **How the projected subgradient method promotes consensus.** A close look at Algorithm 5 reveals that this is done solely by the Lagrange multipliers—in a economic interpretation, these represent "price adjustments" that will lead to a reallocation of resources. However, no "memory" exists about past allocations or adjustments, so the workers never know how far they are from consensus before reallocating—it is conceivable that a smart use of these quantities could accelerate convergence.

Johnson et al. 2007, to obviate the first of these problems, proposed smoothing the objective function through an "entropic" perturbation—which essentially boils down to replacing the max in Eq. 4.58 by a "soft-max." As a result, all the local subproblems become marginal computations (rather than MAP), and can be solved with marginal inference algorithms such as the sum-product algorithm. The amount of perturbation is controlled by a "temperature parameter"—at low temperatures, the solution becomes sufficiently close from the one that

would be obtained without perturbation.[3] A related and asymptotically faster method was proposed later by Jojic et al. (2010), who, in addition to smoothing the objective function in Eq. 4.58, addresses the resulting smooth optimization problem with an *accelerated gradient method* invented by Nesterov (1983), which interleaves the gradient updates to account for momentum.[4] This approach guarantees an ϵ-accurate solution after $O(1/\epsilon)$ iterations, an improvement over the $O(1/\epsilon^2)$ bound of the subgradient algorithm.

However, the methods above come with some drawbacks:

- To be accurate, they need to operate at near-zero temperatures—for example, the $O(1/\epsilon)$ iteration bound of Jojic et al. (2010) requires setting the temperature to $O(\epsilon)$. In many problems, this may lead to numerical instabilities—this is so in problems in which marginal computation requires inverting a matrix which becomes ill-conditioned at low temperatures (as in non-projective dependency parsing, cf. the matrix-tree formula in Eq. 2.19); and in the OR and OR-with-output factors defined in Section 5.3, whose marginal computation requires evaluating the product of K large numbers (cf. 5.57). Typical strategies to deal with numerical overflow and underflow which involve approximating and thresholding may affect the global convergence rate.

- The solution of the local subproblems is always *dense*. Some marginals can be low, but they will never be zero; since this computation is embedded in an iterative algorithm, thresholding them can also compromise convergence. This is in deep constrast with the projected subgradient algorithm, for which the solution of each local subproblem is a MAP estimate and hence always a *single* configuration. This property leads to important runtime savings, achieved by caching the local subproblems; those savings are not readily available in the methods of Johnson et al. 2007 and Jojic et al. (2010).

In this chapter, we take a different approach that also yields a $O(1/\epsilon)$ iteration bound and does not suffer from either of these two drawbacks. The key idea is to ally the simplicity of dual decomposition with the effectiveness of *augmented Lagrangian methods*, which have a long-standing history in optimization (Hestenes, 1969; Powell, 1969; Glowinski and Marroco, 1975; Gabay and Mercier, 1976), and have recently been shown to be very competitive in some problems (Afonso et al., 2010; Goldfarb et al., 2010); see Boyd et al. 2011 for a recent monograph on the subject. The result is AD^3, a new algorithm for LP-MAP inference.

Like the projected subgradient method, AD^3 is an iterative "consensus algorithm," alternating between a *broadcast* operation, where subproblems are distributed across local workers, and a *gather* operation, where the local solutions are assembled by a controller. The main difference is that AD^3 also broadcasts to the workers *the current global solution*, allowing them to regularize their subproblems toward that solution. This has the consequence of speeding up consensus, without sacrificing the modularity of dual decomposition. Additional properties of AD^3 are:

- it is suitable for heavy parallelization (many overlapping components);

- it is provably convergent, even when local subproblems are only solved approximately;

[3] Johnson (2008) noted later a connection with other entropic proximal-point methods.

[4] The idea of smoothing a non-smooth function and then applying accelerated gradient techniques is due to Nesterov (2005), who established the convergence rate of such algorithms.

- it produces optimality certificates for the MAP (when the relaxation is tight);
- it keeps track of primal and dual residuals, allowing monitoring the LP-MAP optimization process and stopping whenever a desired accuracy level is attained.

After providing background on augmented Lagrangian methods (Section 6.2), we introduce and analyze AD^3 (Section 6.3), and then devote special attention to how to solve the local subproblems (Section 6.4). We will see that for all logic factors described in Section 5.3, those subproblems can be solved efficiently with sort operations. A general procedure is then introduced in Section 6.5 that solves the subproblems for arbitrary factors, requiring only a black-box for computing the local MAP. For instances where the LP-MAP relaxation is not tight, we devise in Section 6.6 a branch-and-bound procedure. Experiments with synthetic Ising and Potts models, and two real-world tasks, protein design and frame-semantic parsing, demonstrate the success of our approach (Section 6.7).

We postpone to Chapter 7 the evaluation of AD^3 algorithm in a set of dependency parsing experiments. There, we demonstrate that AD^3 exhibits better performance than the projected subgradient method when the factor graph contains many overlapping factors.

6.2 Augmented Lagrangian Methods

Before describing AD^3, we start with a brief overview of augmented Lagrangian methods. Consider the following general convex optimization problem with equality constraints:

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{u}) + g(\boldsymbol{v}) \\ \text{w.r.t.} & \boldsymbol{u} \in \mathcal{U}, \boldsymbol{v} \in \mathcal{V} \\ \text{s.t.} & \mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} = \boldsymbol{c}, \end{array} \tag{6.1}$$

where $\mathcal{U} \subseteq \mathbb{R}^P$ and $\mathcal{V} \subseteq \mathbb{R}^Q$ are convex sets and $f : \mathcal{U} \to \mathbb{R}$ and $g : \mathcal{V} \to \mathbb{R}$ are convex functions. If it were not for the equality constraints in the last line, the problem in Eq. 6.1 would be *separable* into two independent problems, $\min\{f(\boldsymbol{u}) \mid \boldsymbol{u} \in \mathcal{U}\}$ and $\min\{g(\boldsymbol{v}) \mid \boldsymbol{v} \in \mathcal{V}\}$, *i.e.*, the optimal value of (6.1) would be the sum of those two minima.

6.2.1 Method of Multipliers

For any $\eta \geq 0$, an equivalent problem to the one in Eq. 6.1 is

$$\begin{array}{ll} \text{minimize} & f(\boldsymbol{u}) + g(\boldsymbol{v}) + \frac{\eta}{2}\|\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}\|^2 \\ \text{w.r.t.} & \boldsymbol{u} \in \mathcal{U}, \boldsymbol{v} \in \mathcal{V} \\ \text{s.t.} & \mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} = \boldsymbol{c}. \end{array} \tag{6.2}$$

The only difference between Eqs. 6.1 and 6.2 is that the latter's objective function has an extra Euclidean penalty term which penalizes violations of the equality constraints; the scalar $\eta \geq 0$ controls the intensity of this penalty. Since this term vanishes at feasibility, the two problems have the same solution. The objective function of Eq. 6.2, however, is always η-strongly convex, even if $f(\boldsymbol{u}) + g(\boldsymbol{v})$ is not. The Lagrangian of this problem (call it L_η) is:

$$L_\eta(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{\lambda}) = f(\boldsymbol{u}) + g(\boldsymbol{v}) + \boldsymbol{\lambda} \cdot (\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}) + \frac{\eta}{2}\|\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}\|^2. \tag{6.3}$$

Note that L_η equals the Lagrangian of the original problem, plus the Euclidean penalty term. It is called the *η-augmented Lagrangian* of Eq. 6.1.

Augmented Lagrangian methods (Bertsekas et al., 1999, Sect. 4.2) are a class of optimization methods that address the problem in Eq. 6.1 by seeking a saddle point of L_{η_t}, for some sequence $(\eta_t)_{t\in\mathbb{N}}$. The earliest instance is the *method of multipliers* (Hestenes, 1969; Powell, 1969), which alternates between a *joint* update of $\boldsymbol{u}$ and $\boldsymbol{v}$ through

$$(\boldsymbol{u}^{t+1}, \boldsymbol{v}^{t+1}) \quad := \quad \arg\min\{L_{\eta_t}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{\lambda}^t) \mid \boldsymbol{u} \in \mathcal{U}, \boldsymbol{v} \in \mathcal{V}\} \tag{6.4}$$

and a gradient update of the Lagrange multiplier,

$$\boldsymbol{\lambda}^{t+1} \quad := \quad \boldsymbol{\lambda}^t + \eta_t(\mathbf{A}\boldsymbol{u}^{t+1} + \mathbf{B}\boldsymbol{v}^{t+1} - \boldsymbol{c}). \tag{6.5}$$

Under some assumptions, this method is convergent, and convergence can even be superlinear when the sequence $(\eta_t)_{t\in\mathbb{N}}$ is increasing (see Bertsekas et al. 1999, Sect. 4.2).

A shortcoming of the method of multipliers is that the optimization problem in Eq. 6.4 is often difficult, since the penalty term of the augmented Lagrangian couples the variables $\boldsymbol{u}$ and $\boldsymbol{v}$ (observe that the problem is separable when $\eta_t = 0$).

6.2.2 Alternating Direction Method of Multipliers

The *alternating direction method of multipliers* (ADMM) avoids the shortcoming above, by replacing the joint optimization in Eq. 6.4 by a single block Gauss-Seidel step, in which $\boldsymbol{u}$ and $\boldsymbol{v}$ are *alternately* updated. That is, the following updates are made instead of Eq. 6.4:

$$\begin{aligned} \boldsymbol{u}^{t+1} &:= \arg\min_{\boldsymbol{u}\in\mathcal{U}} L_{\eta_t}(\boldsymbol{u}, \boldsymbol{v}^t, \boldsymbol{\lambda}^t) \\ &= \arg\min_{\boldsymbol{u}\in\mathcal{U}} f(\boldsymbol{u}) + \mathbf{A}^\top\boldsymbol{\lambda}^t \cdot \boldsymbol{u} + \frac{\eta_t}{2}\|\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v}^t - \boldsymbol{c}\|^2, \end{aligned} \tag{6.6}$$

and

$$\begin{aligned} \boldsymbol{v}^{t+1} &:= \arg\min_{\boldsymbol{v}\in\mathcal{V}} L_{\eta_t}(\boldsymbol{u}^{t+1}, \boldsymbol{v}, \boldsymbol{\lambda}^t) \\ &= \arg\min_{\boldsymbol{v}\in\mathcal{V}} g(\boldsymbol{v}) + \mathbf{B}^\top\boldsymbol{\lambda}^t \cdot \boldsymbol{v} + \frac{\eta_t}{2}\|\mathbf{A}\boldsymbol{u}^{t+1} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}\|^2. \end{aligned} \tag{6.7}$$

This is followed by an update of the Lagrange multipliers as stated in Eq. 6.5. In many problems, the two minimizations in Eqs. 6.6–6.7 are much simpler than the joint maximization in 6.4; one example arises in our LP-MAP inference problem, as we shall see.

ADMM was invented in the 1970s by Glowinski and Marroco (1975) and Gabay and Mercier (1976) and is related or analogous to other optimization techniques, such as Douglas-Rachford splitting, Spingarn's method of partial inverses, and some proximal point methods (see Boyd et al. 2011 for an historical overview). It can also be seen as an instance of the more general problem of finding a root of a maximal monotone operator (Eckstein and Bertsekas, 1992), and this insight has been used to derive generalized variants of ADMM. One such variant was proposed by Fortin and Glowinski (1983), who consider a more general

$\boldsymbol{\lambda}$-update, involving an extra parameter $\tau > 0$:

$$\boldsymbol{\lambda}^{t+1} \quad := \quad \boldsymbol{\lambda}^t + \tau\eta_t(\mathbf{A}\boldsymbol{u}^{t+1} + \mathbf{B}\boldsymbol{v}^{t+1} - \boldsymbol{c}). \tag{6.8}$$

In the sequel, we will present the convergence results of ADMM and its variants when applied to the LP-MAP inference problem.

6.3 LP-MAP Inference with the AD^3 Algorithm

We next apply ADMM to the LP-MAP inference problem described in Section 4.6, which will result in the AD^3 algorithm.

6.3.1 Derivation of AD^3

Let us first recall the variable splitting formulation of the LP-MAP inference problem, adapting Eq. 4.52 to incorporate the hard constraint factors:

$$\begin{aligned}
\text{maximize} \quad & \sum_{\alpha\in\mathcal{F}}\left(\sum_{i\in\mathcal{N}(\alpha)}\sum_{y_i\in\mathcal{Y}_i}\theta_i^\alpha(y_i)\mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha}\theta_\alpha(\boldsymbol{y}_\alpha)\mu_\alpha(\boldsymbol{y}_\alpha)\right) + \sum_{\beta\in\mathcal{H}}\sum_{i\in\mathcal{N}(\beta)}\sum_{y_i\in\mathcal{Y}_i}\theta_i^\beta(y_i)\mu_i^\beta(y_i) \\
\text{w.r.t.} \quad & \boldsymbol{\mu}|_\alpha \in \text{MARG}(\mathcal{G}|_\alpha), \quad \forall\alpha\in\mathcal{F}\cup\mathcal{H}, \\
& \zeta_i(y_i) \in \mathbb{R}, \quad \forall i\in\mathcal{V}, y_i\in\mathcal{Y}_i, \\
\text{s.t.} \quad & \mu_i^\alpha(y_i) = \zeta_i(y_i), \quad \forall i\in\mathcal{V}, \alpha\in\mathcal{N}(i), y_i\in\mathcal{Y}_i,
\end{aligned} \tag{6.9}$$

where for each soft factor $\alpha\in\mathcal{F}$ we define (as in Eq. 4.52) $\boldsymbol{\mu}|_\alpha := ((\mu_i(.))_{i\in\mathcal{N}(\alpha)}, \mu_\alpha(.))$, and for each hard constraint factor $\beta\in\mathcal{H}$ we define $\boldsymbol{\mu}|_\beta := (\mu_i(.))_{i\in\mathcal{N}(\beta)}$.[5] As in Section 4.6.2, the "split" log-potentials $\theta_i^\alpha(y_i)$ are chosen so that

$$\sum_{\alpha\in\mathcal{N}(i)}\theta_i^\alpha(y_i) = \theta_i(y_i); \tag{6.10}$$

the simplest choice being to set $\theta_i^\alpha(y_i) := \deg(i)^{-1}\theta_i(y_i)$.

We refer to the problem in Eq. 6.9 as the *relaxed primal*. We call the original MAP problem (Eq. 4.34) the *primal*, and the problem after dual decomposition (Eq. 4.58) the *dual*. Observe that the relaxed primal problem takes the form in Eq. 6.1 by doing the following:

- replacing "max" with "min" and considering the negative of the objective function;
- identifying $\boldsymbol{u}$ and $\boldsymbol{v}$ in Eq. 6.1 respectively with $\boldsymbol{\mu}$ and $\boldsymbol{\zeta}$;
- letting $\mathcal{U}$ in Eq. 6.1 be the Euclidean product of marginal polytopes $\prod_{\alpha\in\mathcal{F}\cup\mathcal{H}}\text{MARG}(\mathcal{G}|_\alpha)$, and setting $\mathcal{V} := \mathbb{R}^{\sum_i|\mathcal{Y}_i|}$;
- identifying $f(\boldsymbol{u})$ in Eq. 6.1 with the objective of Eq. 6.9, and letting $g\equiv 0$;

[5] Recall that, by convention (Definition 5.1 and thereafter), the hard constraint factors do not have a factor log-potential and are not associated with factor marginals; they are defined in terms of their acceptance set $\mathcal{S}_\beta$, and in the binary case, their marginal polytope $\text{MARG}(\mathcal{G}|_\beta)$ is geometrically equivalent to $\text{conv}\,\mathcal{S}_\beta$.

- letting in Eq. 6.1 the matrix $\mathbf{A}$ be the identity, $-\mathbf{B}$ be a matrix with entries in $\{0,1\}$ encoding the matching between the components of $\boldsymbol{\mu}$ and $\boldsymbol{\zeta}$, and $\boldsymbol{c} = 0$.

This falls into the class of a "general consensus optimization" problems, as described by Boyd et al. (2011, Sect. 7.2).

The η-augmented Lagrangian function associated with Eq. 6.9 is

$$\begin{aligned} L_\eta(\boldsymbol{\mu}, \boldsymbol{\zeta}, \boldsymbol{\lambda}) \;=\; & \sum_{\alpha \in \mathcal{F}} \left(\sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\theta_i^\alpha(y_i) + \lambda_i^\alpha(y_i)) \mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) \mu_\alpha(\boldsymbol{y}_\alpha) \right) \\ & + \sum_{\beta \in \mathcal{H}} \sum_{i \in \mathcal{N}(\beta)} \sum_{y_i \in \mathcal{Y}_i} (\theta_i^\beta(y_i) + \lambda_i^\beta(y_i)) \mu_i^\beta(y_i) \\ & - \sum_{\alpha \in \mathcal{F} \cup \mathcal{H}} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \lambda_i^\alpha(y_i) \zeta_i(y_i) \\ & - \frac{\eta}{2} \sum_{\alpha \in \mathcal{F} \cup \mathcal{H}} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\mu_i^\alpha(y_i) - \zeta_i(y_i))^2 . \end{aligned} \tag{6.11}$$

This is the standard Lagrangian that we had derived in Eq. A.7 with additional terms for the hard constraint factors, and the additional Euclidean penalty term in the last line. The ADMM updates are the following:

$$\textbf{Broadcast:} \quad \boldsymbol{\mu}^{(t)} := \arg\max_{\boldsymbol{\mu} \in \mathcal{M}} L_{\eta_t}(\boldsymbol{\mu}, \boldsymbol{\zeta}^{(t-1)}, \boldsymbol{\lambda}^{(t-1)}), \tag{6.12}$$

$$\textbf{Gather:} \quad \boldsymbol{\zeta}^{(t)} := \arg\max_{\boldsymbol{\zeta}} L_{\eta_t}(\boldsymbol{\mu}^{(t)}, \boldsymbol{\zeta}, \boldsymbol{\lambda}^{(t-1)}), \tag{6.13}$$

$$\textbf{Multiplier updates:} \quad \boldsymbol{\lambda}_i^{\alpha(t)} := \boldsymbol{\lambda}_i^{\alpha(t-1)} - \tau \eta_t \left(\boldsymbol{\mu}_i^{\alpha(t)} - \boldsymbol{\zeta}_i^{(t)} \right), \forall \alpha \in \mathcal{F}, i \in \mathcal{N}(a), \tag{6.14}$$

where, in Eq. 6.12, $\mathcal{M} := \prod_{\alpha \in \mathcal{F} \cup \mathcal{H}} \text{MARG}(\mathcal{G}|_\alpha)$. We next analyze each of these updates.

Broadcast step. Crucially, the maximization with respect to $\boldsymbol{\mu}$ (6.12) can be carried out in parallel at each factor, as in the projected subgradient algorithm (Algorithm 5). The only difference is that, instead of a local MAP computation, each soft-factor worker now needs to solve a *quadratic program* of the form:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\theta_i^\alpha(y_i) + \lambda_i^\alpha(y_i)) \mu_i^\alpha(y_i) + \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) \mu_\alpha(\boldsymbol{y}_\alpha) \\ & - \frac{\eta}{2} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\mu_i^\alpha(y_i) - \zeta_i(y_i))^2 \\ \text{w.r.t.} \quad & \boldsymbol{\mu}|_\alpha \in \text{MARG}(\mathcal{G}|_\alpha), \end{aligned} \tag{6.15}$$

and each *hard-factor* worker needs to solve a similar quadratic program, but without the term for the factor marginals:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in \mathcal{N}(\beta)} \sum_{y_i \in \mathcal{Y}_i} (\theta_i^\beta(y_i) + \lambda_i^\beta(y_i)) \mu_i^\beta(y_i) - \frac{\eta}{2} \sum_{i \in \mathcal{N}(\beta)} \sum_{y_i \in \mathcal{Y}_i} \left(\mu_i^\beta(y_i) - \zeta_i(y_i) \right)^2 \\ \text{w.r.t.} \quad & \boldsymbol{\mu}|_\beta \in \text{MARG}(\mathcal{G}|_\beta), \end{aligned} \tag{6.16}$$

The only difference between the subproblems in Eqs. 6.15–6.16 and the corresponding subproblems in the projected subgradient algorithm is the inclusion of a Euclidean penalty term, which regularizes toward the current $\boldsymbol{\zeta}$. This term penalizes deviations from the global consensus. We defer to Sections 6.4 and Section 6.5 a detailed derivation of procedures to solve the local subproblems in Eqs. 6.15–6.16.

Gather step. The maximization of L_{η_t} with respect to $\boldsymbol{\zeta}$ (Eq. 6.13) has a closed form. Indeed, this problem is separable into independent optimizations, one for each $i \in \mathcal{V}$ and $y_i \in \mathcal{Y}_i$:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \sum_{\alpha \in \mathcal{N}(i)} \left(\zeta_i(y_i) - \left(\mu_i^\alpha(y_i) - \eta_t^{-1} \lambda_i^\alpha(y_i) \right) \right)^2 \\ \text{w.r.t.} \quad & \zeta_i(y_i) \in \mathbb{R}, \end{aligned} \tag{6.17}$$

whose solution is

$$\zeta_i(y_i)^{(t)} := \frac{1}{\deg(i)} \sum_{\alpha \in \mathcal{N}(i)} \left(\mu_i^\alpha(y_i) - \eta_t^{-1} \lambda_i^\alpha(y_i) \right). \tag{6.18}$$

This update can be simplified further when the penalty η_t is kept constant:

Proposition 6.1 *Let $\boldsymbol{\lambda}^0 = \mathbf{0}$ and consider a sequence of ADMM iterations indexed by $t \in \mathbb{N}$, each performing the updates in Eqs. 6.12–6.14, with $\eta_t := \eta$ kept constant. Then, we have for each $t \in \mathbb{N}$:*

$$\sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\lambda}_i^{\alpha(t)} = 0, \quad \forall i \in \mathcal{V}. \tag{6.19}$$

As a consequence, under the assumptions above, the $\boldsymbol{\zeta}$-update is simply an averaging operation:

$$\zeta_i(y_i)^{(t)} := \frac{1}{\deg(i)} \sum_{\alpha \in \mathcal{N}(i)} \mu_i^\alpha(y_i). \tag{6.20}$$

Proof. Use induction: Eq. 6.19 is trivially satisfied for $\boldsymbol{\lambda}^{(0)} = \mathbf{0}$; for $t > 0$, if $\boldsymbol{\lambda}^{(t-1)}$ satisfies Eq. 6.19, then, after the Lagrange multiplier updates in Eq. 6.14, we have

$$\begin{aligned} \sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\lambda}_i^{\alpha(t)} &= \sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\lambda}_i^{\alpha(t-1)} - \tau \eta_t \left(\sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\mu}_i^{\alpha(t)} - \deg(i) \boldsymbol{\zeta}_i^{(t)} \right) \\ &= \sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\lambda}_i^{\alpha(t-1)} - \tau \eta_t \left(\sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\mu}_i^{\alpha(t)} - \sum_{\alpha \in \mathcal{N}(i)} \left(\boldsymbol{\mu}_i^{\alpha(t)} - \eta_t^{-1} \boldsymbol{\lambda}_i^{\alpha(t-1)} \right) \right) \\ &= (1 - \tau) \sum_{\alpha \in \mathcal{N}(i)} \boldsymbol{\lambda}_i^{\alpha(t-1)} = 0. \end{aligned} \tag{6.21}$$

■

Putting all the pieces together, we obtain AD^3, depicted as Algorithm 8. AD^3 retains the modularity of the projected subgradient algorithm described in Section 4.6.2 (Algorithm 5). Both algorithms are iterative "consensus algorithms," alternating between a *broadcast* operation, where simple subproblems are distributed across local workers (lines 5–9 in Algorithm 8), and a *gather* operation, where the local solutions are assembled by a controller,

Algorithm 8 Alternating Directions Dual Decomposition (AD^3)

1: **input:** factor graph $\mathcal{G}$, parameters $\boldsymbol{\theta}$, penalty constant η, constant τ
2: initialize $\boldsymbol{\zeta}$ uniformly (*i.e.*, $\zeta_i(y_i) = 1/|\mathcal{Y}_i|$, $\forall i \in \mathcal{V}, y_i \in \mathcal{Y}_i$)
3: initialize $\boldsymbol{\lambda} = \mathbf{0}$
4: **repeat**
5: **for each** factor $\alpha \in \mathcal{F} \cup \mathcal{H}$ **do**
6: set unary log-potentials $\boldsymbol{\omega}_i^\alpha := \deg(i)^{-1}\boldsymbol{\theta}_i + \boldsymbol{\lambda}_i^\alpha$, for $i \in \mathcal{N}(\alpha)$
7: set $\boldsymbol{\omega}|_\alpha := ((\boldsymbol{\omega}_i^\alpha)_{i\in\mathcal{N}(\alpha)}, \boldsymbol{\theta}_\alpha)$ if $\alpha \in \mathcal{F}$, or $\boldsymbol{\omega}|_\alpha := (\boldsymbol{\omega}_i^\alpha)_{i\in\mathcal{N}(\alpha)}$ if $\alpha \in \mathcal{H}$
8: set $\widehat{\boldsymbol{\mu}}|_\alpha := \textsc{SolveQP}(\boldsymbol{\omega}|_\alpha; \boldsymbol{\zeta}|_\alpha)$ (Eq. 6.15 or Eq. 6.16)
9: **end for**
10: compute average $\boldsymbol{\zeta}_i := \deg(i)^{-1} \sum_{\alpha\in\mathcal{N}(i)} \widehat{\boldsymbol{\mu}}_i^\alpha$
11: update $\boldsymbol{\lambda}_i^\alpha := \boldsymbol{\lambda}_i^\alpha - \tau\eta\,(\widehat{\boldsymbol{\mu}}_i^\alpha - \boldsymbol{\zeta}_i)$
12: **until** convergence.
13: **output:** primal variables $\boldsymbol{\zeta}$ and $\boldsymbol{\mu}$, dual variable $\boldsymbol{\lambda}$

which updates the current global solution (line 10) and adjusts multipliers to promote a consensus (line 11). The main practical difference is that AD^3 also broadcasts the current global solution ζ to the workers, allowing them to regularize their subproblems toward that solution. This is expressed in line 8 through the procedure SolveQP, which replaces the procedure ComputeMAP of Algorithm 5. This has the consequence of speeding up consensus, as we shall see.

6.3.2 Convergence Analysis

We now establish the convergence of AD^3, which follows directly from the general convergence properties of ADMM. Remarkably, convergence is ensured with a fixed stepsize/penalty, without the need for annealing. This is because, as we approach convergence, the penalty term must go to zero to ensure (relaxed) primal feasibility.

Proposition 6.2 (Convergence.) *Let $(\boldsymbol{\mu}^{(t)}, \boldsymbol{\zeta}^{(t)}, \boldsymbol{\lambda}^{(t)})_t$ be the sequence of iterates produced by Algorithm 8 with a fixed $\eta_t = \eta$ and $0 < \tau \leq (\sqrt{5}+1)/2 \simeq 1.61$. Then the following holds:*

1. *Primal feasibility of the relaxed primal (Eq. 6.9) is achieved in the limit,* i.e.,

$$\|\boldsymbol{\mu}_i^{\alpha(t)} - \boldsymbol{\zeta}_i^{(t)}\| \to 0, \quad \forall \alpha \in \mathcal{F} \cup \mathcal{H},\ i \in \mathcal{N}(a); \tag{6.22}$$

2. *The sequence $(\boldsymbol{\zeta}^{(t)}, \boldsymbol{\mu}^{(t)})_{t\in\mathbb{N}}$ converges to an optimal solution of the relaxed primal (Eq. 6.9);*
3. *The sequence $(\boldsymbol{\lambda}^{(t)})_{t\in\mathbb{N}}$ converges to an optimal solution of the dual (Eq. 4.58);*
4. *Every element $\boldsymbol{\lambda}^{(t)}$ in the latter sequence is dual feasible. As a consequence, the objective of Eq. 4.58 evaluated at $\boldsymbol{\lambda}^{(t)}$ approaches the optimal value from above.*

Proof. 1, 2, and 3 are general properties of ADMM algorithms (Glowinski and Le Tallec, 1989, Thm. 4.2). All necessary conditions are met: the problem in Eq. 6.9 is convex and the coupling constraints are such that the corresponding matrix $\mathbf{B}$ in Eq. 6.1 is full column rank. Point 4 is a consequence of Proposition 6.1 and the fact that the dual constraint set Λ is precisely defined by Eq. 6.19 (see Eq. 4.56). ∎

The first and last points in Proposition 6.2 reveal an important feature of AD^3: after a sufficient decrease of the penalty term, we have at hand a nearly-feasible primal-dual pair. We next make this more precise by describing how AD^3 allows us to keep track of a primal and dual residuals, and how it can be stopped once they fall below a threshold.

Primal and dual residuals. Recall that the projected subgradient method is able to provide optimality certificates when the relaxation is tight (*i.e.*, when the solution of the LP-MAP problem is the true MAP). While this is good enough when tight relaxations are frequent, as in the settings explored by Rush et al. (2010), Koo et al. (2010), and Rush and Collins (2011), it is hard to know when to stop when a relaxation gap exists. We would like to have similar guarantees concerning the relaxed primal.[6] A general weakness of subgradient algorithms is that they do not have this capacity, and so are usually stopped rather arbitrarily by specifying a maximum number of iterations. In contrast, ADMM allows to keep track of primal and dual residuals (Boyd et al., 2011). This allows providing certificates not only for the exact primal solution (when the relaxation is tight), but also to terminate when a near optimal solution of the relaxed primal problem has been found. The *primal residual* $r_P^{(t)}$ measures the amount by which the agreement constraints are violated:

$$r_P^{(t)} = \frac{\sum_{\alpha \in \mathcal{F} \cup \mathcal{H}} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} (\mu_i^{\alpha(t)}(y_i) - \zeta_i^{(t)}(y_i))^2}{\sum_{i \in \mathcal{V}} \sum_{y_i \in \mathcal{Y}_i} \deg(i)}; \tag{6.23}$$

where the constant in the denominator ensures that $r_P^{(t)} \in [0, 1]$. The *dual residual* r_D^t is the amount by which a dual optimality condition is violated (see Boyd et al. 2011, p. 18, for details). It is computed via:

$$r_D^{(t)} = \frac{\sum_{i \in \mathcal{V}} \sum_{y_i \in \mathcal{Y}_i} \deg(i) (\zeta_i(y_i)^{(t)} - \zeta_i(y_i)^{(t-1)})^2}{\sum_{i \in \mathcal{V}} \sum_{y_i \in \mathcal{Y}_i} \deg(i)}, \tag{6.24}$$

and also satisfies $r_D^{(t)} \in [0, 1]$. Our stopping criterion is thus that these two residuals are below a threshold, *e.g.*, 10^{-3}.

Picking the parameters η and τ. An important issue is that of picking the penalty and stepsize coefficients, which boils down to picking the hyperparameters η and τ. In practice, we found that τ does not have a great impact on the convergence speed, so we often set $\tau = 1$ in the experiments, which renders the penalty coefficient equal to the stepsize and is the most common approach in ADMM implementations. However, it is often important to choose a "good" η. Figure 6.1 shows how different choices of η affect the progress in the dual objective, for a synthetic Potts grid. Typically, a small value of η leads to a stable albeit slower progress, while a large value may be a plus in the long run, but leads to strong oscillations in earlier iterations. A practical alternative, which we follow in the parsing experiments (Section 7.4), is to dynamically adjust η in earlier iterations, as described in Boyd et al. (2011,

[6] This problem is more important than it may look. Problems with many slaves tend to be less exact, hence relaxation gaps are frequent. Also, when decoding is embedded in training, it is useful to obtain the *fractional* solution of the relaxed primal (rather than an approximate integer solution). See Kulesza and Pereira (2007) and Martins et al. (2009c) for details.

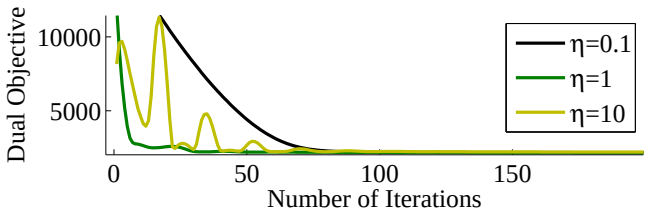


Figure 6.1: Progress in the dual objective for different values of η (the factor graph is a binarized 20×20 Potts grid with 8 states). For $\eta = 0.1$, progress is too slow; for $\eta = 10$ there are strong oscillations, which makes the algorithm slow to take off. In this example, convergence is faster with an intermediate value, $\eta = 1$.

p.20), and freeze it afterwards, so that convergence is not compromised.

Approximately solving the local subproblems. The next proposition states that convergence may still hold if the local subproblems are only solved approximately, provided the sequence of residual errors is summable. This again follows from a general result established for the ADMM algorithm. We refer to Eckstein and Bertsekas (1992) for a proof.

Proposition 6.3 (Eckstein and Bertsekas (1992)) *Let $\eta_t = \eta$ be fixed and $\tau = 1$. Consider the sequence of residual errors $\boldsymbol{\varepsilon}^{(t)} := (\boldsymbol{\varepsilon}_\alpha^{(t)})_{\alpha \in \mathcal{F}}$, where*

$$\boldsymbol{\varepsilon}_\alpha^{(t)} := \widehat{\boldsymbol{\mu}}|_\alpha^{(t)} - \tilde{\boldsymbol{\mu}}|_\alpha^{(t)}, \tag{6.25}$$

where $\widehat{\boldsymbol{\mu}}|_\alpha$ is the true solution of Eq. 6.15 or 6.16, and $\tilde{\boldsymbol{\mu}}|_\alpha$ is an estimate of $\widehat{\boldsymbol{\mu}}|_\alpha$ retrieved by an approximate algorithm. Then, all points 1–4 in Proposition 6.2 still hold, provided $\sum_{t=1}^{\infty} \|\boldsymbol{\varepsilon}^{(t)}\| < \infty$.

The importance of Proposition 6.3 will be clear in Section 6.5, when we describe a general iterative algorithm for solving the local quadratic subproblems. Essentially, Proposition 6.3 allows these subproblems to be solved numerically up to some accuracy without compromising global convergence, as long as the accuracy of the solutions improves sufficiently fast over ADMM iterations.

Convergence rate. Even though ADMM was invented in the 1970s, no convergence rate was known until very recently. The next proposition states the iteration bound of AD^3, which turns out to be asymptotically equivalent to the one of Jojic et al. (2010); both are $O(1/\epsilon)$, and therefore better than the $O(1/\epsilon^2)$ bound of the projected subgradient algorithm.

Proposition 6.4 (Convergence rate.) *Under the conditions stated in Proposition 6.2 and setting $\tau = 1$, the AD^3 algorithm yields an ϵ-accurate objective value after $O(1/\epsilon)$ iterations.*

Proof. See Appendix D. The proof invokes results recently established by He and Yuan (2011) and Wang and Banerjee (2012) concerning convergence in a variational inequality setting, which we adapt to establish a $O(1/t)$ convergence rate in the dual objective. By setting $\epsilon = O(1/t)$, the result follows. ∎

6.4 Solving the Local Subproblems

We next show how to efficiently obtain an *exact* solution of Eqs. 6.15–6.16 for a variety of cases, including binary graphs with logic constraints. The results of this section will be complemented, in Section 6.5, with a new procedure, which will allow us to handle *arbitrary* factors, further extending the scope of problems where AD^3 is applicable.

By subtracting a constant to the objective, rescaling, and turning the maximization into a minimization, the problem in Eq. 6.15 can be written more compactly as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \sum_{i \in \mathcal{N}(\alpha)} \sum_{y_i \in \mathcal{Y}_i} \Big(\mu_i^\alpha(y_i) - \eta^{-1} \omega_i(y_i) \Big)^2 - \eta^{-1} \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha) \mu_\alpha(\boldsymbol{y}_\alpha) \\ \text{w.r.t.} \quad & \boldsymbol{\mu}|_\alpha \in \text{MARG}(\mathcal{G}|_\alpha), \end{aligned} \tag{6.26}$$

where $\omega_i(y_i) := \eta \zeta_i(y_i) + \theta_i^\alpha(y_i) + \lambda_i^\alpha(y_i)$. For hard factors, the corresponding problem is even more compact:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \sum_{i \in \mathcal{N}(\beta)} \sum_{y_i \in \mathcal{Y}_i} \Big(\mu_i^\beta(y_i) - \eta^{-1} \omega_i(y_i) \Big)^2 \\ \text{w.r.t.} \quad & \boldsymbol{\mu}|_\beta \in \text{MARG}(\mathcal{G}|_\beta), \end{aligned} \tag{6.27}$$

i.e., it amounts to computing an Euclidean projection onto the marginal polytope.

We will see that, in many cases, Eqs. 6.26–6.27 admit a closed-form solution or can be solved exactly by an efficient procedure: this is so for binary pairwise factors, the logic hard constraint factors of Section 5.3, and factors linked to multi-valued variables, once binarized. In the first two cases, and also in the last case if the factors are pairwise, the asymptotic computational cost is the same as that of MAP computations, up to a logarithmic factor.

6.4.1 Binary pairwise factors

In this section, we obtain the closed form solution of problem Eq. 6.26 for binary pairwise factors. Define $z_1 := \mu_1(1)$, $z_2 := \mu_2(1)$, and $z_{12} := \mu_{12}(1,1)$. In Appendix E.1, we rewrite the problem in Eq. 6.26 as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(z_1 - c_1)^2 + \frac{1}{2}(z_2 - c_2)^2 - c_{12} z_{12} \\ \text{w.r.t.} \quad & z_1, z_2, z_{12} \in [0,1]^3 \\ \text{s.t.} \quad & z_{12} \leq z_1, \quad z_{12} \leq z_2, \quad z_{12} \geq z_1 + z_2 - 1, \end{aligned} \tag{6.28}$$

where we have substituted

$$c_1 = (\eta^{-1}\omega_1(1) + 1 - \eta^{-1}\omega_1(0) - \eta^{-1}\theta_{12}(00) + \eta^{-1}\theta_{12}(10))/2 \tag{6.29}$$

$$c_2 = (\eta^{-1}\omega_2(1) + 1 - \eta^{-1}\omega_2(0) - \eta^{-1}\theta_{12}(00) + \eta^{-1}\theta_{12}(01))/2 \tag{6.30}$$

$$c_{12} = (\eta^{-1}\theta_{12}(00) - \eta^{-1}\theta_{12}(10) - \eta^{-1}\theta_{12}(01) + \eta^{-1}\theta_{12}(11))/2. \tag{6.31}$$

Observe that we can assume $c_{12} \geq 0$ without loss of generality—indeed, if $c_{12} < 0$, we recover this case by redefining $c_1' = c_1 + c_{12}$, $c_2' = 1 - c_2$, $c_{12}' = -c_{12}$, $z_2' = 1 - z_2$, $z_{12}' = z_1 - z_{12}$. Thus,

assuming that $c_{12} \geq 0$, the lower bound constraints $z_{12} \geq z_1 + z_2 - 1$ and $z_{12} \geq 0$ are always inactive and can be ignored. Hence, (6.28) can be simplified to:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(z_1 - c_1)^2 + \frac{1}{2}(z_2 - c_2)^2 - c_{12} z_{12} \\ \text{w.r.t.} \quad & z_1, z_2, z_{12} \\ \text{s.t.} \quad & z_{12} \leq z_1, \quad z_{12} \leq z_2, \quad z_1 \in [0,1], \quad z_2 \in [0,1]. \end{aligned} \tag{6.32}$$

The next proposition establishes a closed form solution for this problem, which immediately translates into a procedure for SolveQP for binary pairwise factors.

Proposition 6.5 *The problem in Eq. 6.32, with $c_{12} \geq 0$, has the following closed form solution:*

$$(z_1^*, z_2^*) = \begin{cases} ([c_1]_{\mathbb{U}}, & [c_2 + c_{12}]_{\mathbb{U}}) & \text{if } c_1 > c_2 + c_{12} \\ ([c_1 + c_{12}]_{\mathbb{U}}, & [c_2]_{\mathbb{U}}) & \text{if } c_2 > c_1 + c_{12} \\ ([(c_1 + c_2 + c_{12})/2]_{\mathbb{U}}, & [(c_1 + c_2 + c_{12})/2]_{\mathbb{U}}) & \text{otherwise,} \end{cases} \tag{6.33}$$

and $z_{12}^* = \min\{z_1^*, z_2^*\}$, where $[x]_{\mathbb{U}} := \min\{\max\{x, 0\}, 1\}$ denotes projection (clipping) onto the unit interval $\mathbb{U} := [0,1]$.

Proof. See Appendix E.1. ∎

6.4.2 Hard constraint factors

We next turn to *hard constraint factors*, whose local subproblems in AD^3 take the form in Eq. 6.27. We have seen in Chapter 5 that such factors are important in many applications, ranging from error-correcting coding to NLP, and we have presented in Section 5.3 an inventory of hard constraint factors that allow expressing constraints in first-order logic, along with an algebraic characterization of their marginal polytopes (see Figure 5.4). For a binary hard constraint factor with acceptance set $\mathcal{S}_\beta$, we have seen that its marginal polytope is equivalent to the convex hull $\mathcal{Z}_\beta = \operatorname{conv} \mathcal{S}_\beta$.

The problem in Eq. 6.27 is precisely a *projection onto the marginal polytope* of the hard constraint factor. Noting that $\|\boldsymbol{\mu}_i^\beta - \eta^{-1}\boldsymbol{\omega}_i^\beta\|^2 = (\mu_i^\beta(1) - \eta^{-1}\omega_i^\beta(1))^2 + (1 - \mu_i^\beta(1) - \eta^{-1}\omega_i^\beta(0))^2$, which equals a constant plus $2(\mu_i^\beta(1) - \eta^{-1}(\omega_i^\beta(1) + 1 - \omega_i^\beta(0))/2)^2$, we have that Eq. 6.27 can be written as:

$$\operatorname{proj}_\beta(\boldsymbol{z}_0) := \arg\min_{\boldsymbol{z} \in \mathcal{Z}_\beta} \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{z}_0\|^2, \tag{6.34}$$

where $\boldsymbol{z}_{0i} := (\omega_i^\beta(1) + 1 - \omega_i^\beta(0))/2$, for each $i \in \mathcal{N}(\beta)$. We next show how to compute this projection for the factors introduced in Section 5.3. Several auxiliary results and proofs can be shown in Appendix E.2.

XOR Factor. Consider the one-hot XOR factor introduced in Section 5.3. Recall that for this case, the marginal polytope $\mathcal{Z}_{\text{XOR}}$ is the probability simplex, as stated in Eq. 5.25:

$$\mathcal{Z}_{\text{XOR}} = \left\{ \boldsymbol{z} \in [0,1]^K \;\middle|\; \sum_{k=1}^K z_k = 1 \right\}. \tag{6.35}$$

Algorithm 9 Projection onto simplex (Duchi et al., 2008)

Input: z_0
Sort z_0 into y_0: $y_1 \geq \ldots \geq y_K$
Find $\rho = \max \left\{ j \in [K] \mid y_{0j} - \frac{1}{j} \left((\sum_{r=1}^{j} y_{0r}) - 1 \right) > 0 \right\}$
Define $\tau = \frac{1}{\rho} \left(\sum_{r=1}^{\rho} y_{0r} - 1 \right)$
Output: z s.t. $z_i = \max\{z_{0i} - \tau, 0\}$.

Hence the quadratic problem in Eq. 6.26 reduces to that of *projecting onto the simplex*. That problem is well-known in the optimization community (see, *e.g.*, Michelot 1986); by writing the KKT conditions, it is simple to show that the solution z^* is a soft-thresholding of z_0, and therefore the problem can be reduced to that of finding the right threshold. Algorithm 9 provides an efficient procedure; it requires a sort operation, which renders its cost $O(K \log K)$. A proof of correctness appears in Duchi et al. (2008).[7]

OR Factor. Consider the OR factor introduced in Section 5.3. For this case, Eq. 5.29 gives us the following expression for the marginal polytope, which we reproduce here:

$$\mathcal{Z}_{\text{OR}} = \left\{ z \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} z_k \geq 1 \right\}. \tag{6.36}$$

In Appendix E.2, we introduce the *Sifting Lemma* (Lemma E.1), which guarantees the correctness of the following procedure for computing a projection onto $\mathcal{Z}_{\text{OR}}$:

1. Set $\tilde{z}$ as the projection of z_0 onto the unit cube. This can be done by clipping each coordinate to the unit interval $\mathbb{U} = [0,1]$, i.e., by setting $\tilde{z}_i = [z_{0i}]_{\mathbb{U}} = \min\{1, \max\{0, z_{0i}\}\}$. If $\sum_{i=1}^{K} \tilde{z}_i \geq 1$, then return $\tilde{z}$. Else go to step 2.
2. Return the projection of z_0 onto the simplex (use Algorithm 9).

The validity of the second step stems from the following fact: if the relaxed problem in the first step does not return a feasible point then, from the Sifting Lemma, the constraint $\sum_{i=1}^{K} z_i \geq 1$ has to be active, *i.e.*, we must have $\sum_{i=1}^{K} z_i = 1$. This, in turn, implies that $z \leq 1$, hence the problem becomes equivalent to the XOR case. In sum, the worst-case runtime is $O(K \log K)$, although it is $O(K)$ if the first step succeeds.

OR-with-output Factor. Consider now the OR-with-output factor introduced in Section 5.3, whose marginal polytope is given by Eq. 5.39:

$$\mathcal{Z}_{\text{OR}-\text{out}} = \left\{ z \in [0,1]^{K+1} \;\middle|\; \sum_{k=1}^{K} z_k \geq z_{K+1},\ z_k \leq z_{K+1}, \forall k \in \{1, \ldots, K\} \right\}. \tag{6.37}$$

Solving the quadratic problem for this factor is slightly more complicated than in the previous two cases; however, we next see that it can also be addressed in $O(K \log K)$ with a sort operation.

[7] In the high-dimensional case, a red-black tree can be used to reduce this cost to $O(K)$ (Duchi et al., 2008). In later iterations of AD^3, great speed-ups can be achieved in practice since this procedure is repeatedly invoked with only small changes to the coefficients.

The polytope $\mathcal{Z}_{\text{OR}-\text{out}}$ can be expressed as the intersection of the following three sets:[8]

$$\mathbb{U}^{K+1} := [0,1]^{K+1} \tag{6.38}$$

$$\mathcal{A}_1 := \{z \in \mathbb{R}^{K+1} \mid z_k \leq z_{K+1}, k = 1, \ldots, K\} \tag{6.39}$$

$$\mathcal{A}_2 := \left\{ z \in [0,1]^{K+1} \;\middle|\; \sum_{k=1}^{K} z_k \geq z_{K+1} \right\}. \tag{6.40}$$

We further define $\mathcal{A}_0 := [0,1]^{K+1} \cap \mathcal{A}_1$. From the Sifting Lemma (Lemma E.1), we have that the following procedure is correct:

1. Set $\tilde{z}$ as the projection of z_0 onto the unit cube. If $\tilde{z} \in \mathcal{A}_1 \cap \mathcal{A}_2$, then we are done: just return $\tilde{z}$. Else, if $\tilde{z} \in \mathcal{A}_1$ but $\tilde{z} \notin \mathcal{A}_2$, go to step 3. Otherwise, go to step 2.

2. Set $\tilde{z}$ as the projection of z_0 onto $\mathcal{A}_0$ (we will describe how to do this later). If $\tilde{z} \in \mathcal{A}_2$, return $\tilde{z}$. Otherwise, go to step 3.

3. Return the projection of z_0 onto the set $\{z \in [0,1]^{K+1} \mid \sum_{k=1}^{K} z_k = z_{K+1}\}$. This set is precisely the marginal polytope of the XOR-with-output factor (cf. Eq. 5.36), hence the projection corresponds to the local subproblem for that factor. As described in Section 5.3, XOR-with-output is equivalent to a XOR with the last input negated. For this, we can employ the procedure already described for the XOR factor, which consists of a projection onto the simplex (using Algorithm 9).[9]

Note that the first step above can be omitted; however, it avoids performing step 2 (which requires a sort) unless it is really necessary. To completely specify the algorithm, we only need to explain how to compute the projection onto $\mathcal{A}_0$ (step 2):

Procedure 6.1 *To project onto $\mathcal{A}_0 = [0,1]^{K+1} \cap \mathcal{A}_1$:*

2a. *Set $\tilde{\tilde{z}}$ as the projection of z_0 onto $\mathcal{A}_1$. Algorithm 10 shows how to do this.*

2b. *Set $\tilde{z}$ as the projection of $\tilde{\tilde{z}}$ onto the unit cube (with the usual clipping procedure).*

The proof that the composition of these two projections yields the desired projection onto $\mathcal{A}_0$ is a bit involved, and we included it in the appendix (Proposition E.2).[10] We only need to describe how to project onto $\mathcal{A}_1$ (step 2a), which is written as the following problem:

$$\min_z \frac{1}{2}\|z - z_0\|^2 \quad \text{s.t.} \quad z_k \leq z_{K+1}, \;\; \forall k = 1, \ldots, K. \tag{6.41}$$

[8] Actually, the set $\mathbb{U}^{K+1}$ is redundant, since we have $\mathcal{A}_2 \subseteq \mathbb{U}^{K+1}$ and therefore $\mathcal{Z}_{\text{OR}-\text{out}} = \mathcal{A}_1 \cap \mathcal{A}_2$. However it is computationally advantageous to consider this redundancy, as we shall see.

[9] This will be clarified in the sequel, when we show how to handle negations.

[10] Note that in general, the composition of individual projections is not equivalent to projecting onto the intersection. In particular, commuting steps 2a and 2b would make our procedure incorrect.

Algorithm 10 Projection onto $\mathcal{A}_1$

Input: z_0
Sort $z_{01}, \ldots, z_{0K}$ into $y_1 \geq \ldots \geq y_K$
Find $\rho = \min \left\{ j \in [K+1] \mid \frac{1}{j} \left(z_{0,K+1} + \sum_{r=1}^{j-1} y_r \right) > y_j \right\}$
Define $\tau = \frac{1}{\rho} \left(z_{0,K+1} + \sum_{r=1}^{\rho-1} y_r \right)$
Output: z s.t. $z_{K+1} = \tau$ and $z_i = \min\{z_{0i}, \tau\}$, $i = 1, \ldots, K$.

It can be successively rewritten as:

$$\begin{aligned} & \min_{z_{K+1}} \frac{1}{2}(z_{K+1} - z_{0,K+1})^2 + \sum_{i=1}^{K} \min_{z_k \leq z_{K+1}} \frac{1}{2}(z_k - z_{0k})^2 \\ = & \min_{z_{K+1}} \frac{1}{2}(z_{K+1} - z_{0,K+1})^2 + \sum_{k=1}^{K} \frac{1}{2}(\min\{z_{K+1}, z_{0k}\} - z_{0k})^2 \\ = & \min_{z_{K+1}} \frac{1}{2}(z_{K+1} - z_{0,K+1})^2 + \frac{1}{2} \sum_{k \in \mathcal{I}(z_{K+1})} (z_{K+1} - z_{0k})^2. \end{aligned} \tag{6.42}$$

where $\mathcal{I}(z_{K+1}) \triangleq \{k \in [K] : z_{0k} \geq z_{K+1}\}$. Assuming that the set $\mathcal{I}(z_{K+1})$ is given, the previous is a sum-of-squares problem whose solution is

$$z^*_{K+1} = \frac{z_{0,K+1} + \sum_{k \in \mathcal{I}(z_{K+1})} z_{0k}}{1 + |\mathcal{I}(z_{K+1})|}. \tag{6.43}$$

The set $\mathcal{I}(z_{K+1})$ can be determined by inspection after sorting $z_{01}, \ldots, z_{0K}$. The procedure is shown in Algorithm 10.

Negations. Finally, we show how a factor that contains *negated inputs* (see Section 5.3.3) can have its problem reduced to that of a factor without negations.

Let factor β' be constructed from β by negating one of the inputs (without loss of generality, the first one, y_1)—*i.e.*, $\mathcal{S}_{\beta'} = \{\boldsymbol{y}_{\beta'} \mid (1 - y_1, y_2, \ldots, y_K) \in \mathcal{S}_\beta\}$. Then, if we have a procedure for evaluating the operator proj_β, we can use it for evaluating $\text{proj}_{\beta'}$ through the change of variable $z'_1 := 1 - z_1$, which turns the objective function into $(1 - z'_1 - z_{01})^2 = (z'_1 - (1 - z_{01}))^2$. Naturally, the same idea holds when there is more than one negated input. The overall procedure computes $\boldsymbol{z} = \text{proj}_{\beta'}(\boldsymbol{z}_0)$:

1. For each input i, set $z'_{0i} = z_{0i}$ if it is not negated and $z'_{0i} = 1 - z_{0i}$ otherwise.
2. Obtain $\boldsymbol{z}'$ as the solution of $\text{proj}_\beta(\boldsymbol{z}'_0)$.
3. For each input i, set $z_i = z'_i$ if it is not negated and $z_i = 1 - z'_i$ otherwise.

6.4.3 Larger factors and multi-valued variables

For general factors, a closed-form solution of the problem in Eq. 6.26 is not readily available. This is a disadvantage relative to the projected subgradient algorithm, which can easily handle factors with multi-valued variables, and even certain structured factors such as chains

or trees, by using dynamic programming (or other combinatorial algorithm) for the corresponding local MAP computations. A similar strategy for computing an *exact* solution for these special structures does not seem possible in AD^3.

However, AD^3 can still tackle general factors with multi-valued variables by employing the *binarization* procedure described in Section 5.5.1. After binarization, the resulting graph is a binary constrained graph for which we already developed the machinery necessary for solving the AD^3 subproblems. As described in Section 5.5.1, this transformation preserves the LP-MAP problem and, up to log factors, it results in the same asymptotic complexity (per iteration) of message-passing algorithms in the original graph.

An alternative strategy that sidesteps binarization is to use an *inexact* algorithm that becomes sufficiently accurate as Algorithm 8 proceeds (exploiting the convergence result in Proposition 6.3); this can be achieved by warm-starting with the solution obtained in the previous iteration. We reserve Section 6.5 to describe in full detail an active-set method that implements this strategy, and which can gracefully handle coarser decompositions, in which each factor is a subgraph such as a chain or a tree.

6.5 An Active Set Method for Handling Arbitrary Factors

We next describe an iterative procedure for tackling the local AD^3 subproblems of *arbitrary* factors. The only interface between our procedure and the factor is a black-box implementation of the function ComputeMAP, just like the projected subgradient algorithm (Algorithm 5). This enables plugging combinatorial algorithms that are tailored for computing the MAP on certain factors.

There are several ways to convert a quadratic problem into a sequence of linear problems—the Frank-Wolfe algorithm is perhaps the simplest example (Frank and Wolfe, 1956). However, that algorithm is not suitable for our task, since it is too slow to converge and does not exploit the *sparsity* of the solution (which we establish in Proposition 6.6 below). Instead, we propose an *active set method* (Nocedal and Wright, 1999, Sect. 16.4). For simplicity, we limit our description to the *soft factor* case (Eq. 6.26), but the method extends in a straightforward manner to hard factors, as will be discussed later.

Let us start by writing the local subproblem in Eq. 6.26 in the following compact form:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{u}-\boldsymbol{a}\|^2 - \boldsymbol{b}^\top \boldsymbol{v} \\ \text{with respect to} \quad & \boldsymbol{u} \in \mathbb{R}^{\sum_{i\in\mathcal{N}(\alpha)} |\mathcal{Y}_i|}, \quad \boldsymbol{v} \in \mathbb{R}^{|\mathcal{Y}_\alpha|} \\ \text{subject to} \quad & \begin{cases} \boldsymbol{u} = \mathbf{M}\boldsymbol{v} \\ \mathbf{1}^\top \boldsymbol{v} = 1 \\ \boldsymbol{v} \geq 0, \end{cases} \end{aligned} \tag{6.44}$$

where we have introduced variables $\boldsymbol{u}$ and $\boldsymbol{v}$ to denote the variable marginals $\boldsymbol{u} := (\mu_i^\alpha(.))_{i\in\mathcal{N}(\alpha)}$ and the factor marginals $\boldsymbol{v} := \mu_\alpha(.)$, respectively; and similarly for the parameters $\boldsymbol{a} := (\eta^{-1}\omega_i(.))_{i\in\mathcal{N}(\alpha)}$ and $\boldsymbol{b} := \eta^{-1}\theta_\alpha(.)$. The three sets of constraints in (6.44) come from the

following representation of the marginal polytope $\mathrm{MARG}(\mathcal{G}|_\alpha)$:[11]

$$\mathrm{MARG}(\mathcal{G}|_\alpha) = \left\{ \boldsymbol{\mu}|_\alpha \;\middle|\; \begin{array}{ll} \mu_i^\alpha(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha), & \forall i \in \mathcal{N}(\alpha),\ y_i \in \mathcal{Y}_i, \\ \sum_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \mu_\alpha(\boldsymbol{y}_\alpha) = 1, & \\ \mu_\alpha(\boldsymbol{y}_\alpha) \geq 0, & \forall \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha \end{array} \right\}, \quad (6.45)$$

where we have introduced the matrix $\mathbf{M}$ in Eq. 6.44 with $\sum_i |\mathcal{Y}_i|$ rows and $|\mathcal{Y}_\alpha|$ columns, to encode the consistency constraints, *i.e.*, given by $[\mathbf{M}]_{(i,y_i),\boldsymbol{y}_\alpha} := [\![\boldsymbol{y}_\alpha \sim y_i]\!]$.

A crucial result is that the problem in Eq. 6.44 always admits a *sparse solution*, as we show in the next proposition:

Proposition 6.6 *The problem in Eq 6.44 admits a solution $\boldsymbol{v}^* \in \mathbb{R}^{|\mathcal{Y}_\alpha|}$ with at most $\sum_{i \in \mathcal{N}(\alpha)} |\mathcal{Y}_i| - \mathcal{N}(\alpha) + 1$ nonzeros.*

Proof. See Appendix E.3. ∎

The fact that the solution lies in a low dimensional subspace, as established in Proposition 6.6, makes active set methods appealing. Such methods keep track of an *active set* given by the nonzero components of $\boldsymbol{v}$; we now know that we only need to maintain at most $O(\sum_i |\mathcal{Y}_i|)$ elements in the active set—note the *additive*, rather than multiplicative, dependency on the number of values of the variables. This alleviates eventual concerns about memory and storage. Before introducing the method in full detail, we need to analyze the dual and write the KKT conditions of the problem in Eq. 6.44.

Lagrangian and Dual Problem. The problem (6.44) has $O(\mathcal{Y}_\alpha)$ variables and constraints, a number which grows exponentially fast with $\deg(\alpha)$. We next derive a dual formulation with only $O(\sum_i |\mathcal{Y}_i|)$ variables. The Lagrangian of (6.44) is

$$L(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \tau, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{a}\|^2 - \boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{w}^\top(\mathbf{M}\boldsymbol{v} - \boldsymbol{u}) - \tau(1 - \mathbf{1}^\top \boldsymbol{v}) - \boldsymbol{\lambda}^\top \boldsymbol{v}. \quad (6.46)$$

Equating $\nabla_{\boldsymbol{u}} L(\boldsymbol{u})$ and $\nabla_{\boldsymbol{v}} L(\boldsymbol{v})$ to zero leads to the equations:

$$\boldsymbol{u} = \boldsymbol{a} - \boldsymbol{w} \quad (6.47)$$

$$\mathbf{M}^\top \boldsymbol{w} + \boldsymbol{b} = \tau \mathbf{1} - \boldsymbol{\lambda}. \quad (6.48)$$

Since the Lagrange variables $\boldsymbol{\lambda}$ are constrained to be non-negative, the dual problem takes the form (after replacing the maximization with a minimization and subtracting a constant to the objective):

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{w} - \boldsymbol{a}\|^2 + \tau \\ \text{with respect to} \quad & \boldsymbol{w} \in \mathbb{R}^{\sum_i |\mathcal{Y}_i|}, \quad \tau \in \mathbb{R} \\ \text{subject to} \quad & \mathbf{M}^\top \boldsymbol{w} + \boldsymbol{b} \leq \tau \mathbf{1}. \end{aligned} \quad (6.49)$$

[11] Eq. 6.45 follows directly from the constraints that characterize the local polytope (Eq. 4.40), by noting that summing over the first constraints $\mu_i^\alpha(y_i) = \sum_{\boldsymbol{y}_\alpha \sim y_i} \mu_\alpha(\boldsymbol{y}_\alpha)$, we obtain that, for each $i \in \mathcal{N}(\alpha)$, it holds $\sum_{y_i \in \mathcal{Y}_i} \mu_i^\alpha(y_i) = \sum_{\boldsymbol{y}_\alpha} \mu_\alpha(\boldsymbol{y}_\alpha)$.

This problem can be seen as a "projection with slack" onto the set $\{\boldsymbol{w} \mid \mathbf{M}^\top \boldsymbol{w} + \boldsymbol{b} \leq \mathbf{0}\}$.[12]

We will apply the active set method of Nocedal and Wright (1999, Sect. 16.4) to the dual problem (6.49). Let $\boldsymbol{m}_r$ denote the rth column of matrix $\mathbf{M}$. This method keeps track of a working set of constraints which are assumed to be *active*:

$$\mathcal{W} := \left\{ r \in \{1, \ldots, |\mathcal{Y}_\alpha|\} \mid \boldsymbol{m}_r^\top \boldsymbol{w} + b_r - \tau = 0 \right\}; \tag{6.50}$$

by complementary slackness, at the optimum this set contains the support of $\boldsymbol{v}^*$: $\mathcal{W} \supseteq \{r \in \{1, \ldots, |\mathcal{Y}_\alpha|\} \mid v_r > 0\}$.

KKT conditions. The KKT equations of (6.49) are:

$$\begin{aligned}
\boldsymbol{u} - \boldsymbol{a} + \boldsymbol{w} &= 0 \quad (\nabla_{\boldsymbol{u}} L = 0) && (6.51)\\
\mathbf{M}^\top \boldsymbol{w} + \boldsymbol{b} &= \tau \mathbf{1} - \boldsymbol{\lambda} \quad (\nabla_{\boldsymbol{v}} L = 0) && (6.52)\\
\mathbf{M}\boldsymbol{v} &= \boldsymbol{u} \quad \text{(Primal feasibility)} && (6.53)\\
\mathbf{1}^\top \boldsymbol{v} &= 1 \quad \text{(Primal feasibility)} && (6.54)\\
\boldsymbol{y} &\geq \mathbf{0} \quad \text{(Primal feasibility)} && (6.55)\\
\boldsymbol{\lambda} &\geq \mathbf{0} \quad \text{(Dual feasibility)} && (6.56)\\
\boldsymbol{\lambda}^\top \boldsymbol{v} &= \mathbf{0} \quad \text{(Complementary slackness).} && (6.57)
\end{aligned}$$

We can eliminate variables $\boldsymbol{u}$ and $\boldsymbol{w}$ and reduce the above set of equations to

$$\begin{aligned}
\mathbf{M}^\top \mathbf{M}\boldsymbol{v} + \tau \mathbf{1} &= \mathbf{M}^\top \boldsymbol{a} + \boldsymbol{b} + \boldsymbol{\lambda} && (6.58)\\
\mathbf{1}^\top \boldsymbol{v} &= 1 && (6.59)\\
\boldsymbol{v} &\geq \mathbf{0} && (6.60)\\
\boldsymbol{\lambda} &\geq \mathbf{0} && (6.61)\\
\boldsymbol{\lambda}^\top \boldsymbol{v} &= \mathbf{0}. && (6.62)
\end{aligned}$$

Let $\mathcal{J} := \{r \in \{1, \ldots, |\mathcal{Y}_\alpha|\} \mid v_r > 0\}$ denote the support of $\boldsymbol{v}$, *i.e.*, the indices corresponding to nonzero entries. Obviously, we do not know the set $\mathcal{J}$ beforehand; along the algorithm, we maintain our best guess through the active set $\mathcal{W}$ (Eq. 6.50). Denote by $\boldsymbol{v}_\mathcal{J}$ and $\boldsymbol{\lambda}_\mathcal{J}$ the subvectors formed by only those indices. Denote by $\mathbf{M}_\mathcal{J}$ the submatrix of $\mathbf{M}$ with columns in $\mathcal{J}$. The above set of equations imply $\boldsymbol{\lambda}_\mathcal{J} = \mathbf{0}$ and the following system in matricial form:

$$\begin{bmatrix} \mathbf{M}_\mathcal{J}^\top \mathbf{M}_\mathcal{J} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_\mathcal{J} \\ \tau \end{bmatrix} = \begin{bmatrix} \mathbf{M}_\mathcal{J}^\top \boldsymbol{a} + \boldsymbol{b}_\mathcal{J} \\ 1 \end{bmatrix}. \tag{6.63}$$

If the matrix on the left-hand side is non-singular, the system has a unique solution $(\widehat{\boldsymbol{v}}_\mathcal{J}, \widehat{\tau})$.[13] Padding $\widehat{\boldsymbol{v}}_\mathcal{J}$ with zeros, we form $\widehat{\boldsymbol{v}}$ (this never needs to be done explicitly, of course). If $\widehat{\boldsymbol{v}}$ and

[12] Interestingly, this problem has affinities with well-known formulations in machine learning using L_2-regularization, such as support vector machines and the MIRA algorithm.

[13] If at some point of the active-set algorithm the matrix becomes singular, we can still compute a solution by looking at its null space and proceed with the execution.

$\widehat{\tau}$ satisfy the KKT conditions (6.58–6.62), *i.e.*, if we have $\widehat{\boldsymbol{v}} \geq \mathbf{0}$ and

$$\mathbf{M}^\top \mathbf{M} \widehat{\boldsymbol{v}} + \widehat{\tau} \mathbf{1} \quad \geq \quad \mathbf{M}^\top \boldsymbol{a} + \boldsymbol{b}, \tag{6.64}$$

then the set $\mathcal{I}$ is correct and $\widehat{\boldsymbol{v}}$ is a solution of the QP (Eq. 6.44).

To use this rationale algorithmically, the following two operations need to be performed:

- Solving the KKT system (6.63). This involves decomposing or inverting a matrix of size $|\mathcal{I}|$-by-$|\mathcal{I}|$. Since we need to perform this operation repeatedly after inserting or removing one element from the active set, this procedure can be done efficiently (namely, in time $O(|\mathcal{I}|^2)$) by keeping track of the inverted matrix. From Proposition 6.6, this runtime is manageable, since we can limit the size of the active set to $|\mathcal{I}| = O(\sum_i |Y_i|)$. Note also that after inserting a new configuration $\boldsymbol{y}_\alpha$ to the active set, this will originate a new column in the matrix $\mathbf{M}_\mathcal{I}$, and for the matrix inversion stated above, we need to update the product $\mathbf{M}_\mathcal{I}^\top \mathbf{M}_\mathcal{I}$. From the definition of $\mathbf{M}$ and simple algebra, we have that the $(\boldsymbol{y}_\alpha, \boldsymbol{y}'_\alpha)$ entry in $\mathbf{M}^\top \mathbf{M}$ is just the *number of common values* between the configurations $\boldsymbol{y}_\alpha$ and $\boldsymbol{y}'_\alpha$. Hence, when a new configuration $\boldsymbol{y}_\alpha$ is added to the active set $\mathcal{W}$, that configuration needs to be compared with all the others already in $\mathcal{W}$.

- Checking if any of the constraints (6.64) is violated. By setting $\widehat{\boldsymbol{u}} = \mathbf{M}\widehat{\boldsymbol{v}}$ and $\widehat{\boldsymbol{w}} = \boldsymbol{a} - \widehat{\boldsymbol{u}}$, the constraints can be written as $\mathbf{M}^\top \widehat{\boldsymbol{w}} + \boldsymbol{b} \leq \widehat{\tau}\mathbf{1}$, and they hold if and only if

$$\max_{r \in \mathcal{Y}_\alpha} \boldsymbol{m}_r^\top \widehat{\boldsymbol{w}} + \boldsymbol{b}_r \leq \widehat{\tau}. \tag{6.65}$$

  Hence, to verify the constraints we need to compute the maximum on the left hand side; interestingly that maximum is nothing but the MAP response of the factor α to variable log-potentials $\widehat{\boldsymbol{w}}$ and the factor log-potentials $\boldsymbol{b}$, and this value can be computed through the function ComputeMAP.

Active set algorithm. Algorithm 11 depicts the whole procedure, which is an instantiation of Nocedal and Wright (1999, Alg. 16.1) adapted to our purposes. At each iteration s, an active set $\mathcal{W}_s$ is maintained which stores our current guess about the support of $\boldsymbol{v}$.[14] The active set is initialized arbitrarily: *e.g.*, in the first outer iteration of AD^3 it can be initialized with the single output $\boldsymbol{y}_\alpha$ which is the MAP given log-potentials $\boldsymbol{a}$ and $\boldsymbol{b}$; and in subsequent AD^3 iterations it can be warm-started with the solution obtained in the previous outer iteration. At each inner iteration, the KKT system (Eq. 6.63) is solved given the current active set.

If the solution is the same as in the previous round, a black-box ComputeMAP is then invoked to check for violations of the KKT constraints; if there are no violations, the algorithm returns; otherwise it adds the most violated constraint to the active set.

If the solution of the KKT system $\widehat{\boldsymbol{v}}$ is different from the one in the previous round ($\widehat{\boldsymbol{v}}_s$), a line search is made to set $\widehat{\boldsymbol{v}}_{s+1}$ in-between $\widehat{\boldsymbol{v}}_s$ and $\widehat{\boldsymbol{v}}$. The stepsize α_s is chosen to yield the most possible progress while keeping the constraints satisfied. This has a closed form

[14] The meaning of our active set is the opposite of the active set as defined by Nocedal and Wright 1999, since we are tackling the dual problem.

Algorithm 11 Active Set Algorithm for Solving a General AD^3 Subproblem

input: Parameters $\boldsymbol{a}, \boldsymbol{b}, \mathbf{M}$, starting point $\boldsymbol{v}_0$
initialize $\mathcal{W}_0$ as the support of $\boldsymbol{v}_0$
for $s = 0, 1, 2, \ldots$ **do**
  solve the KKT system and obtain $\widehat{\boldsymbol{v}}$ and $\widehat{\tau}$ (Eq. 6.63)
  if $\widehat{\boldsymbol{v}} = \boldsymbol{v}_s$ **then**
    compute $\widehat{\boldsymbol{u}} := \mathbf{M}\widehat{\boldsymbol{v}}$ and $\widehat{\boldsymbol{w}} := \boldsymbol{a} - \widehat{\boldsymbol{u}}$
    obtain the tighter constraint $r = \textsc{ComputeMAP}(\widehat{\boldsymbol{w}}, \boldsymbol{b})$
    if $\boldsymbol{m}_r^\top \widehat{\boldsymbol{w}} + \boldsymbol{b}_r \leq \widehat{\tau}$ **then**
      return solution $\widehat{\boldsymbol{u}}$ and $\widehat{\boldsymbol{v}}$
    else
      add the most violated constraint to the active set: $\mathcal{W}_{s+1} := \mathcal{W}_s \cup \{r\}$
    end if
  else
    compute the interpolation constant α_s (Eq. 6.66)
    set $\boldsymbol{v}_{s+1} := (1 - \alpha_s)\boldsymbol{v}_s + \alpha_s \widehat{\boldsymbol{v}}$
    if there are blocking constraints **then**
      pick a blocking constraint r
      remove that blocking constraint from the active set: $\mathcal{W}_{s+1} := \mathcal{W}_s \setminus \{r\}$
    end if
  end if
end for
output: $\widehat{\boldsymbol{u}}$ and $\widehat{\boldsymbol{v}}$

solution, as derived by Nocedal and Wright (1999, p. 457):

$$\alpha_s := \min \left\{ 1, \min_{r \in \mathcal{W}_s \text{ s.t. } v_{s,r} > \widehat{v}_r} \frac{v_{s,r}}{v_{s,r} - \widehat{v}_r} \right\}. \tag{6.66}$$

A constraint r is called *blocking* if $v_{s,r} > \widehat{v}_r$. If there are blocking constraints, the minimizer r above is picked and removed from the active set.

Each iteration of Algorithm 11 always improves the dual objective, and with a suitable strategy to prevent cycles and stalling, the algorithm is guaranteed to stop after a finite number of steps (Nocedal and Wright, 1999, Theo. 16.5). In practice, since it is run as a subroutine of AD^3, we don't need to run Algorithm 11 to optimality, which is particularly convenient in early iterations of AD^3—note that this is enabled by Proposition 6.3, which shows that global convergence is preserved as long as one makes the algorithm sufficiently precise in later AD^3 iterations. The ability of warm-starting each outer AD^3 iteration with the solution computed in the previous round is very useful in practice: we have observed that, thanks to this warm-starting strategy, very few inner iterations are typically necessary until the correct active set is identified.

6.6 Exact Inference with Branch-and-Bound

Finally, it is worthwhile to recall that AD^3, as just described, solves the LP-MAP *relaxation* of the actual problem. In some problems, this relaxation is tight (in which case a certificate of optimality will be obtained), but this is not always the case. When a fractional solution is ob-

tained, it is desirable to have a strategy to recover the exact MAP solution. Two observations are noteworthy. First, as we saw in Section 4.6, the optimal value of the relaxed problem LP-MAP provides an upper bound to the original problem MAP. In particular, any feasible dual point provides an upper bound to the original problem's optimal value. Second, during execution of the AD^3 algorithm, we always keep track of a sequence of feasible dual points (as guaranteed by Proposition 6.2, item 4). Therefore, each iteration constructs tighter and tighter upper bounds. We thus have all that is necessary for implementing a *branch-and-bound search* that finds the exact solution of the ILP. The procedure works recursively as follows:

1. Initialize $L = -\infty$ (our best value so far).
2. Run Algorithm 8. If the solution $\boldsymbol{\zeta}^*$ is integer, return $\boldsymbol{\zeta}^*$ and set L to the objective value. If along the execution we obtain an upper bound less than L, then Algorithm 8 can be safely stopped and return "infeasible"—this is the *bound* part. Otherwise (if $\boldsymbol{\zeta}^*$ is fractional) go to step 3.
3. Find the "most fractional" component of $\boldsymbol{\zeta}^*$ (call it $\zeta_j^*(.)$) and *branch*: for every $y_j \in \mathcal{Y}_j$, constrain $\zeta_j(y_j) = 1$ and go to step 2, eventually obtaining an integer solution $\boldsymbol{\zeta}^*|_{y_j}$ or infeasibility. Return the $\boldsymbol{\zeta}^* \in \{\boldsymbol{\zeta}^*|_{y_j}\}_{y_j \in \mathcal{Y}_j}$ that yields the largest objective value.

Although this procedure may have worst-case exponential runtime (which is not surprising since the problem is NP-hard), in many problems for which the relaxations are near-exact it is found empirically very effective. We describe an example for frame-semantic parsing in Section 6.7.4.

6.7 Experiments

We next provide an empirical comparison between AD^3 (Algorithm 8) and four other approximate MAP inference algorithms, all of which address the LP-MAP problem:

- Star-MSD (Sontag et al., 2011), an acceleration of the max-sum diffusion algorithm (Kovalevsky and Koval, 1975; Werner, 2007) based on star updates. This is an instance of a block coordinate descent algorithm, as briefly mentioned in Section 4.6.1;
- Generalized MPLP (Globerson and Jaakkola, 2008), the algorithm that we have described in detail in Section 4.6.1;
- The projected subgradient algorithm for dual decomposition (Komodakis et al., 2007), depicted as Algorithm 5 in Section 4.6.2;
- The accelerated version of dual decomposition introduced by Jojic et al. 2010, which we described briefly in the beginning of this chapter (Section 6.1).

All these algorithms address the LP-MAP problem; the first two use message-passing, performing block coordinate descent in the dual; and the last two are dual decomposition algorithms. The accelerated dual decomposition algorithm is guaranteed to converge to a ϵ-accurate solution after $O(1/\epsilon)$ iterations, where ϵ is a prescribed hyperparameter. The primal and dual objectives are the same for all algorithms. All the baselines have the same

algorithmic complexity per iteration—this complexity is asymptotically the same as AD^3 run on a binarized graph, but is different from AD^3 with the active set method. In all experiments with AD^3, we set $\tau = 1$.

6.7.1 Binary Pairwise MRFs

Figure 6.2 shows typical plots for an Ising model (binary pairwise MRF) on a random grid, with single node log-potentials chosen as $\theta_i(1) - \theta_i(0) \sim \mathcal{U}[-1, 1]$ and mixed edge couplings in $\mathcal{U}[-\rho, \rho]$, where $\rho \in \{0.5, 1, 1.5, 2\}$. Decompositions are edge-based for all methods. For MPLP and Star-MSD, primal feasible solutions $(\widehat{y}_i)_{i \in \mathcal{V}}$ were obtained by decoding the single node messages, as in Globerson and Jaakkola (2008); for the dual decomposition methods we use $\widehat{y}_i = \operatorname{argmax}_{y_i} \mu_i(y_i)$.

We observe that the projected subgradient is the slowest, taking a long time to find a "good" primal feasible solution, arguably due to the large number of components. The accelerated dual decomposition method of Jojic et al. (2010) is also not competitive in this setting, as it consumes many iterations before it reaches a near-optimal region.[15] MPLP performs slightly better than Star-MSD and both are comparable to AD^3 in terms of convergence of the dual objective. However, AD^3 outperforms all competitors at obtaining a "good" feasible primal solution in early iterations (it retrieved the true MAP in all cases, in ≤ 200 iterations). We conjecture that this rapid progress in the primal is due to the penalty term in the augmented Lagrangian (Eq. 6.11), which is very effective at pushing for a feasible primal solution of the relaxed LP.

6.7.2 Multi-valued Pairwise MRFs

To assess the effectiveness of AD^3 in the non-binary case, we evaluated it against the accelerated dual decomposition algorithm and MPLP in a Potts model (multi-valued pairwise MRF) with single node log-potentials chosen as $\theta_i(y_i) \sim \mathcal{U}[-1, 1]$ and edge log-potentials as $\theta_{ij}(y_i, y_j) \sim \mathcal{U}[-10, 10]$ if $y_i = y_j$ and 0 otherwise. For the two baselines, we used the same edge decomposition as before, since they can handle multi-valued variables; for AD^3, we binarized the graph as described in Section 6.4.3. Figure 6.3 (left) shows the best dual solutions obtained at each iteration for the three algorithms. We observe that MPLP decreases the objective very rapidly in the beginning and then slows down. The accelerated dual decomposition algorithm has much better performance than in the binary case and manages to converge faster (which matches the encouraging results obtained by Jojic et al. 2010), but it is relatively slow to take off. AD^3 has the best features of both methods.

We also evaluate AD^3 in the original graph, employing the active set method described in Section 6.5 to tackle the quadratic subproblems. We set the maximum number of inner iterations in the active set method to 10, and we compared the performance with AD^3 run in the binarized graph. The result is shown in Figure 6.3 (right). Since now the iterations are not directly comparable (since AD^3 with the active set method needs to do more work per iteration), we plot the objectives over runtime. The variant of AD^3 with the active set method is clearly superior.

[15] It is conceivable that the early iterations could make faster progress by annealing ϵ. Here we have just used the variant described by Jojic et al. (2010).

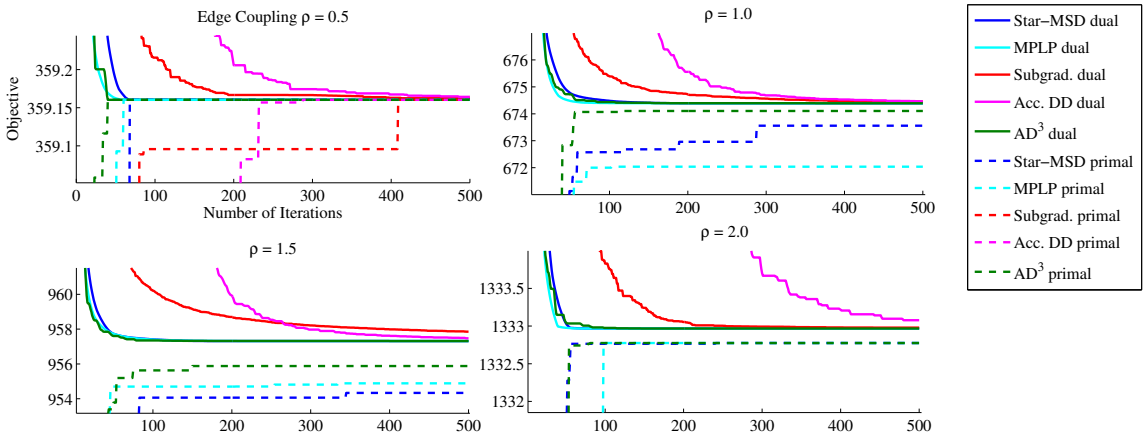


Figure 6.2: Results for 30×30 random Ising models with several edge couplings. We plot the dual objectives and the best primal feasible solution at each iteration. For the subgradient method, we set $\eta_t = \eta_0 / t$, with η_0 yielding the maximum dual improvement in 10 iterations, with halving steps (those iterations are not plotted). For accelerated dual decomposition, we plot the most favorable $\epsilon \in \{0.1, 1, 10, 100\}$. For AD^3, we set $\eta = 5.0$ and $\tau = 1.0$. All decompositions are edge-based.

6.7.3 Protein Design

We next evaluate AD^3 in benchmark protein design problems, using the dataset of Yanover et al. (2006). In these problems, the input is a 3D shape, and the goal is to find the most stable sequence of amino-acids in that shape. The problem can be represented as a pairwise Markov network whose variables denote the identity of amino-acids and rotamer configurations, yielding hundreds of possible states for each node.[16]

We compare the performance of AD^3 with MPLP, using David Sontag's implementation.[17] Figure 6.4 shows the progress in the dual objective over runtime, for two of the most largest problem instances. In both cases, MPLP steeply decreases the objective at first, but then reaches a plateau and eventually halts, with no significant improvement over consecutive iterations. AD^3 rapidly surpasses MPLP in getting a better dual objective.[18] For AD^3, we use the active set method for solving the subproblems, which led to a better performance than the graph binarization technique.

6.7.4 Frame Semantic Parsing

Finally, we report experiments on a natural language processing task which involves logic constraints: frame-semantic parsing, using the FrameNet lexicon (Fillmore, 1976). For each

[16] The protein design data set is available from http://www.jmlr.org/papers/volume7/yanover06a/Rosetta_Design_Dataset.tgz.

[17] Available at http://cs.nyu.edu/~dsontag/code/mplp_ver1.tgz. Note that Sontag's code implements a tightening approach that is able to retrieve the true MAP (Sontag et al., 2008); here, we only run their initial set of iterations that solves the LP-MAP relaxation, the same problem that AD^3 addresses.

[18] Note however that the ultimate goal of this task is to get a good primal solution. Experimentally, we observed that MPLP often gets better primal solutions than AD^3, regardless being consistently worse in optimizing the dual. However, the ability of quickly decreasing the dual is very desirable in tightening approaches, where larger factors are iteratively added to the model to tighten the relaxation. It is likely that a tightened AD^3 would obtain a competitive performance in this task.

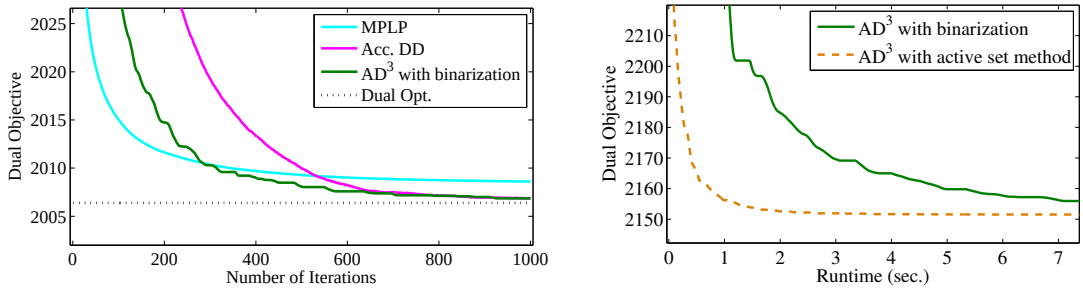


Figure 6.3: Left: Results for a random 20×20 random Potts model with 8-valued nodes and coupling factor $\rho = 10$. We plot the dual objectives, and the value of the true dual optimum. For the accelerated dual decomposition algorithm, we set $\epsilon = 1.0$; for AD^3, we set $\eta = 0.5$. Right: Comparison between AD^3 with the active set method, and AD^3 run on the binarized graph, for a random model of the same kind. In both cases, we set $\eta = 1$, the most favourable choice in $\{0.01, 0.1, 1, 10\}$. For the active set method, we limited the maximum number of inner iterations to 10.

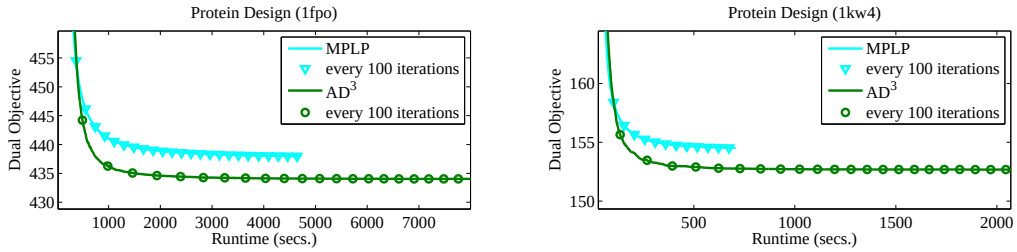


Figure 6.4: Experiments on two of the largest protein files in the protein design dataset of Yanover et al. (2006), with 3167 (1fbo) and 1163 (1kw4) factors. The results are representative of the performance obtained in other protein files. In AD^3, we self-adjust η as proposed by Boyd et al. (2011), with an initial value of $\eta = 1.0$. We also show the iterations of both methods. Observe that the greater cost per iteration in AD^3 is amortized in later iterations, since we can take advantage of warm-starting in the active set method.

instance, the goal is to predict the set of arguments and roles for a predicate word in a sentence, while respecting several constraints about the frames that can be evoked. The resulting graphical models are binary constrained factor graphs with XOR and AtMostOne factors.[19]

Figure 6.5 shows a comparison between AD^3, MPLP and the projected subgradient algorithm for the five most difficult examples, the ones in which the LP relaxation is not tight. Again, AD^3 outperforms all the competitors. Since these examples have a fractional LP-MAP solution, we applied the branch-and-bound procedure described in Section 6.6 to obtain the exact MAP for these examples. The whole dataset contains 4,462 instances, which were parsed by this exact variant of the AD^3 algorithm in only 4.78 seconds, against 43.12 seconds of CPLEX, a state-of-the-art commercial ILP solver.

In the next chapter (Sections 7.4–7.5), we will illustrate the application of AD^3 for *dependency parsing*, which we tackle as a constrained inference problem. We defer to Section 7.5 the detailed empirical analysis of AD^3 in that task. We will also analyze runtime and describe

[19] We refer to the recent thesis by Das (2012, Sect. 5.5) for more details on this task.

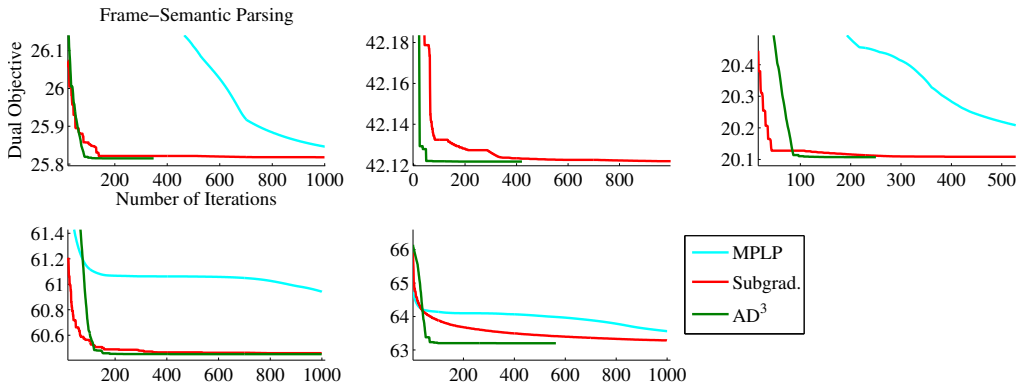


Figure 6.5: Experiments in frame-semantic parsing, using models described in Das (2012). We selected the five most difficult examples, which have between 321 and 884 variables, and between 32 and 59 factors. Unlike MPLP and the projected subgradient algorithms, which did not converge after 1000 iterations, AD^3 achieves convergence in a few hundreds of iterations for all but one example. For the projected subgradient algorithm, we set $\eta_t = \eta_0/t$, with $\eta_0 = 1.0$ being the most favorable choice for all examples. For AD^3, we used the procedure described above to self-adjust η.

how one may greatly benefit from caching and warm-starting the subproblems.

6.8 Future Work

The contributions presented in this chapter open up several directions for future research.

The active set method requires only a black box for computing the local MAP for each factor. By taking larger components of the graph, such as sequences or trees, for which MAP inference is tractable, we can reduce substantially the number of subproblems, which may lead to speed-ups. In addition, there are many problems which are characterized by combinatorial subproblems, such as the composition of head automata with maximum spanning tree algorithms for dependency parsing (Koo and Collins, 2010), as well as several problems in parsing and machine translation (Auli and Lopez, 2011; Rush and Collins, 2011; Chang and Collins, 2011). The active set method can be applied to such problems, using combinatorial algorithms for local MAP computations. It would be interesting to compare AD^3 with the projected subgradient algorithm in the number of oracle calls, for a broad range of problems. We leave this for future work.

It could also be interesting to try other algorithms for solving the quadratic problems, instead of the active set algorithm. Interior point methods could be a good fit for some problems; however they usually fail to capture the sparsity. Other options are cyclic projection algorithms and the smoothening method of Nesterov (2005); the latter would require a black box for computing *marginals* (rather than MAP).

Another idea is to replace the quadratic penalty of ADMM by a general Bregman divergence, in an attempt to obtain easier subproblems. We point out that an entropic penalty would *not* lead to the same subproblems as in Jojic et al. (2010): it would lead to the problem of minimizing non-strictly convex free energies with different counting numbers. Although extensions of ADMM to Bregman penalties have been considered in the literature, to the best of our knowledge, convergence has been shown only for quadratic penalties (see also Boyd

et al. 2011, p. 24). Quadratic penalties also have the important advantage that they lead to sparse solutions, which can be exploited algorithmically (as we do in our active set method). The convergence proofs, however, can be extended to Mahalanobis distances, since they correspond to an affine transformation of the subspace defined by the equality constraints of Eq. 6.2. Simple operations, such as scaling these constraints, do not affect the algorithms that are used to solve the subproblems. Therefore, we can generalize AD^3 by adding a a number of free parameters to specify these scalings; tweaking those parameters might eventually lead to even faster convergence.

Note also that AD^3 is dual decomposable, hence the subproblems can all be solved in parallel. There is a good chance that significant speed-ups may be achieved in large problems through parallelization, for example in multi-core architectures, or through GPU programming. This sort of techniques has been applied with great success to other message-passing algorithms, giving rise to parallel frameworks such as GraphLab[20] (Low et al., 2010). A significant amount of computation can also be saved by *caching* the subproblems, which tend to become more and more similar across later iterations. We will see instances of this in Chapter 7.

The branch-and-bound algorithm for obtaining the exact MAP deserves further experimental study. One advantage of AD^3 over other algorithms for solving the LP-MAP problem is that AD^3 is able to quickly produce sharp upper bounds, which is a useful property when embedded in a branch-and-bound procedure. For many problems, there are effective rounding procedures that can also produce lower bounds, which can be exploited for guiding the search. There are also alternatives to branch-and-bound, for example tightening procedures of the same kind as Sontag et al. (2008), which keep adding larger factors to decrease the duality gap. The variant of AD^3 with the active set method can be used for handling these larger factors.

Interestingly, the AD^3 algorithm, unlike other message-passing and dual decomposition algorithms, does not have any connection at all with dynamic programming, nor it is likely that dynamic programming algorithms can be useful for solving its quadratic subproblems. These quadratic subproblems are interesting *per se*, as they can also arise in other proximal point methods in which Euclidean penalties are used. Therefore, investigating efficient procedures for projecting onto the marginal polytopes of other factors can be a fruitful line of research. In particular, AD^3 can be adapted to solve other problems than LP-MAP, in which the objective function may not be linear. For example, if we change the objective function of LP-MAP (or MAP) by incorporating an Euclidean penalty of the form $\sum_{i \in \mathcal{V}} \sum_{y_i \in \mathcal{Y}_i} \boldsymbol{\mu}_i(y_i)$, this will promote a sparse solution, in which some "marginals" will be integer, indicating a factor configuration, while others will be fractional, implicitly representing a list of outputs with non-negligible probability. This can be useful for developing *pruning models*, as an alternative to typical approaches based on hard thresholding or K-best lists. Note that, for this kind of problem, the AD^3 algorithm can be employed with minimal changes and the same subproblem solvers, since all subproblems are still quadratic.

[20] Available at http://graphlab.org/.

6.9 Discussion

In this chapter, we introduced AD^3, a new LP-MAP inference algorithm based on the alternating direction method of multipliers of (Glowinski and Marroco, 1975; Gabay and Mercier, 1976). AD^3 maintains the modularity of dual decomposition methods, but achieves faster consensus, by penalizing, for each subproblem, deviations from the current global solution.

We established the convergence properties of AD^3, blending general results for ADMM algorithms obtained by Glowinski and Le Tallec (1989); Eckstein and Bertsekas (1992); He and Yuan (2011); Wang and Banerjee (2012). Importantly, AD^3 converges to an ϵ-accurate solution with an iteration bound of $O(1/\epsilon)$, like the accelerated dual decomposition method of Jojic et al. (2010).

We have shown how the AD^3 algorithm can be applied to the constrained factor graphs introduced in the previous chapter, by deriving efficient procedures for projecting onto the marginal polytopes of the hard constraint factors introduced in Section 5.3. This paves the way for using AD^3 in problems with declarative constraints. Up to a logarithmic term, the cost of projecting on those factors is asymptotically the same as that of passing messages. We also provided a closed-form solution of the AD^3 subproblem for pairwise binary factors.

We suggested two approaches for dealing with factors involving multi-valued variables: one is *binarizing* the graph, as described in Section 5.5.1, and reverting to the already known projection procedures. The other is a new *active set method*, which requires only an oracle that computes the local MAP (the same requirement as the projected subgradient method discussed in Section 4.6.2). The active set method seems more adequate than other algorithms for quadratic programming, since it deals well with sparse solutions—which are guaranteed in this problem, as we show in Proposition 6.6—and it can take advantage of warm starting.

Experiments in synthetic and real-world datasets have shown that AD^3 is able to solve the LP-MAP problem more efficiently than other methods for a variety of tasks, including synthetic problems involving Ising and Potts grid models, as well as protein design and frame-semantic parsing problems. We also show how AD^3 can be wrapped in a branch-and-bound procedure to retrieve the true MAP, rendering the method *exact*. This approach looks particularly useful for problems in which the LP-MAP relaxation is not very loose, which happens with the frame-semantic parsing application that we use as a testbed. In the next chapter, we will include additional experiments in dependency parsing, where we show that AD^3 can handle a large number of factors better than other algorithms. We will also see how the algorithm can benefit from caching the subproblems.

During the preparation of this thesis, some related methods have been proposed in the literature. Meshi and Globerson (2011) also propose applying ADMM to inference in graphical models, although they insert the quadratic penalty in the dual problem (the one underlying the MPLP algorithm) rather than in the primal. Quite recently, Yedidia et al. (2011) proposed the divide-and-concur algorithm for decoding of low-density parity check codes (LDPC), which shares resemblances with AD^3.[21] Even more recently, Barman et al. (2011) applies an algorithm analogous to AD^3 for the same LDPC decoding problem, by deriving an efficient algorithm for projecting onto the parity polytope. It would be interesting to compare the performance of this specialized projection algorithm with our active set method.[22]

[21] Their approach can be seen as non-convex ADMM.

[22] Recall that our active set method requires only a black box for solving the local MAP problem. In the case

of the parity polytope this can be done in linear time, while projecting onto the polytope takes $O(L \log L)$ time, using the algorithm of Barman et al. (2011).

Chapter 7

Turbo Parsers: Linear Programming Relaxations for Dependency Parsing

In this chapter, we introduce and characterize *turbo parsers*[1]—dependency parsers that run approximate inference in a *loopy* factor graph, ignoring the global effects caused by the loops. Several parsers fall into this class, most noticeably the ones proposed by Smith and Eisner (2008), Martins et al. (2009b, 2010f, 2011c) and Koo et al. (2010). All these can be regarded as approximate inference algorithms over a loopy factor graph. In this chapter, we give special emphasis to parsers built from *linear relaxations* (i.e., LP-MAP inference, as described in Section 4.6). The contributions presented on this chapter are:

- We consider new models for dependency parsing that can accommodate a wide range of global features (Figure 7.1).
- To perform inference under such models, we derive a new concise ILP formulation using multi-commodity flows. By "concise" we mean that the size of the program (number of variables and constraints) grows polynomially with the length of the sentence, an improvement over previous work, for which this growth is exponential (Riedel and Clarke, 2006).
- We show that the resulting parser, once a linear relaxation is made, is an instance of a turbo parser. In doing so, we establish some equivalence results with other turbo parsers (Smith and Eisner, 2008; Koo et al., 2010), by characterizing the underlying factor graphs and the optimization problems that are addressed in the corresponding inference algorithms.
- We show how to apply the AD^3 algorithm (introduced in Chapter 6) to dependency parsing, allowing us to exploit the structure of the linear program described by the factor graph. We also describe how certain computational tricks (such as caching) enable important speed-ups.

We show the robustness of our parsers by providing an empirical evaluation for 14 languages, with state-of-the-art results. We provide some error analysis, and compare the per-

[1]The name stems from "turbo codes," a class of high-performance error-correcting codes introduced by Berrou et al. (1993) for which decoding algorithms are equivalent to running belief propagation in a graph with loops (McEliece et al., 1998).

formance of the AD^3 algorithm with the projected subgradient algorithm and a commercial LP solver (CPLEX).

The concise ILP formulation and the use of linear relaxations in statistical parsing was introduced in Martins et al. (2009b); this chapter extends that paper with some new material: new parts for head bigrams and for directed paths, and a tighter relaxation built from the linear program of head automata. The unified view of these and other parsers as "turbo parsers" was established in Martins et al. (2010f). The AD^3 algorithm was originally applied to this problem in Martins et al. (2011c).

7.1 Motivation and Related Work

The motivation for introducing turbo parsers comes from two important observations with respect to previously existing linear models in statistical parsing, such as the ones we have described in Chapter 2:

- Models that permit exact inference are usually forced to make unrealistic independence assumptions, or to have a stringent decomposition into parts, for the sake of tractability. As a result, these models are strongly limited in the amount of output context that can be encoded in the features, becoming *under-expressive*. This is the case, *e.g.*, of arc-factored models for dependency parsing (Eisner, 1996; McDonald et al., 2005b), and of first-order probabilistic context-free grammars (Charniak, 1997; Johnson, 1998).

- On the other hand, empirical performance typically improves whenever one breaks these strong locality assumptions, even though this has the common downside of making exact inference very costly or intractable. In practice, approximate decoders using greedy search, beam-search, or sampling methods seem to lead to good accuracy-runtime tradeoffs, and search error is not usually seen as a problem (McDonald et al., 2006; Huang, 2008; Finkel et al., 2008). However, many of these methods are either heuristic in nature or the underlying approximation is not well understood.

The recent advances in the graphical models literature, summarized in Chapter 4, are a natural fit to statistical parsing. In fact, the approximate inference algorithms described therein have important advantages over other traditional methods: firstly, the nature of the approximation is reasonably well understood, and secondly, they often come with guarantees or optimality certificates. Given our achievements already described in Chapters 5 and 6, we are well equipped to devise feature-rich statistical parsing, without a great deal of concern about the locality of the features:

- We need not worry about hard constraints (inherent in any parsing problem), since we have already described in Chapter 5 how this machinery extends to constrained factor graphs.

- We have proposed in Chapter 6 the AD^3 algorithm, which solves the LP-MAP problem and is suitable for constrained models.

Our formulations are very flexible and can accommodate a wide range of non-arc-factored parts. In our experiments, we encode features involving arcs, siblings (consecutive or arbitrary), grandparents, head bigrams, directed paths, and non-projective arcs; a graphic representation of these parts is shown in Figure 7.1. Many extensions are possible and relatively

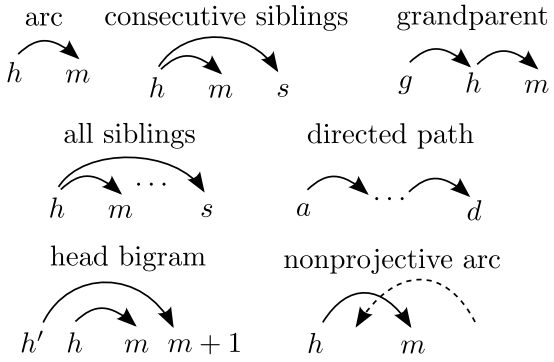


Figure 7.1: Parts used by our parser. *Arcs* are the basic parts: any dependency tree can be "read out" from the arcs it contains. *Consecutive siblings* and *grandparent* parts introduce horizontal and vertical Markovization (McDonald et al., 2006; Carreras, 2007). We break the horizontal Markov assumption via *all siblings* parts (which look at arbitrary pairs of siblings, not necessarily consecutive) and the vertical one through parts which indicate a *directed path* between two words. Inspired by transition-based parsers, we also adopt *head bigram* parts, which look at the heads attached to consecutive words. Finally, we have parts which indicate if an arc is *non-projective* (*i.e.*, if it spans words that do not descend from its head).

simple to include. For example, if expert knowledge is available, one can include additional hard or soft first-order logic constraints, as typically done in Markov logic networks and constrained conditional models (Richardson and Domingos, 2006; Chang et al., 2008).

This chapter is organized as follows: in Section 7.2, we derive a concise ILP formulation for dependency parsing. In Section 7.3, we proceed to show how the resulting parser, once a linear relaxation is made, is an instance of a turbo parser, by deriving its underlying graphical model; we also characterize other turbo parsers, by writing down their optimization problems. This paves the way for Section 7.4, where we show how to apply the AD^3 algorithm (introduced in Chapter 6) to dependency parsing. Section 7.5 provides an empirical evaluation on data in 14 languages, along with a qualitative error analysis and a comparison with competing algorithms. Section 7.6 concludes and points directions for future work.

7.2 Dependency Parsing as an Integer Linear Program

Much attention has recently been devoted to integer linear programming formulations of NLP problems, with interesting results in applications like semantic role labeling (Roth and Yih, 2005; Punyakanok et al., 2004), machine translation (Germann et al., 2001), word alignment (Lacoste-Julien et al., 2006), summarization (Clarke and Lapata, 2008), and coreference resolution (Denis and Baldridge, 2007), among others. The rationale for the development of ILP formulations is to incorporate non-local features or global constraints, often difficult to handle with traditional dynamic programming algorithms. ILP formulations focus on the declarative representation of problems, rather than procedural design. While solving an ILP is NP-hard in general, fast solvers exist today that make it a practical solution for many NLP problems.

In this section, we present new *concise* ILP formulations for projective and non-projective dependency parsing. By focusing on the modeling part and abstracting away from algo-

rithm design, our formulations pave the way for efficient exploitation of global features and constraints in parsing applications, leading to models which are more powerful than others considered in the literature.

We point out that casting dependency parsing as an ILP is not new: Riedel and Clarke (2006) were the first to come up with an ILP formulation, which had exponentially many constraints; they tackle the problem using a cutting-plane method. However, this approach appears to be too slow for practical purposes, hence *efficient* formulations have remained an open problem, which we now address.

Our formulations offer the following comparative advantages:

- The numbers of variables and constraints are polynomial in the sentence length, as opposed to requiring exponentially many constraints, eliminating the need for incremental procedures like the cutting-plane algorithm;
- LP relaxations permit fast online discriminative training of the constrained model;
- Soft constraints may be automatically learned from data. In particular, our formulations handle higher-order arc interactions (like siblings and grandparents), look at the heads of consecutive words in the sentence, model word valency, and can learn to favor nearly-projective parses whenever this is observed in the data.
- We handle interactions that cannot be captured by horizontal and vertical Markovization without an explosion of the number of states. For example, we include scores for arbitrary sibling arcs (not necessarily consecutive) and for pairs of words which have a directed path in the dependency tree.

Many other extensions are possible and relatively simple to include. This will become clear when we represent our formulations as constrained factor graphs, in Section 7.3. We postpone to Section 7.5 the experimental analysis of our turbo parsers on standard parsing tasks.

7.2.1 The Arborescence Polytope

At the heart of our formulation lies a concise characterization of the *arborescence polytope*, which we introduce in this section. For basic definitions concerning dependency parsing, projective and non-projective arcs and trees, we refer to the background material presented in Section 2.3.

Recall the definition of a dependency tree (Definition 2.2). Let $x := (x_0, x_1, \ldots, x_L)$ be a sentence with L words, where $x_0 := *$ is a special symbol for the designated root. We will reason about the entire set $\mathcal{Y}(x)$ of potential dependency parse trees for x.

Let us first regard each token in $\{0, \ldots, L\}$ as a node in a fully connected directed graph (or *digraph*) $\mathcal{D} = (\mathcal{N}, \mathcal{A})$; *i.e.*, $\mathcal{N} = \{0, \ldots, L\}$ is the set of nodes, and $\mathcal{A} = \mathcal{N}^2$ is the set of all possible arcs connecting the nodes.[2] Using terminology from graph theory, we say that $\mathcal{B} \subseteq \mathcal{A}$ is an *r-arborescence* of the directed graph $\mathcal{D}$ (also called a "directed spanning tree of $\mathcal{D}$ with designated root r") if $(\mathcal{V}, \mathcal{B})$ is a (directed) tree rooted at r. This is illustrated

[2] The general case where $\mathcal{A} \subseteq \mathcal{N}^2$ is also of interest; it arises whenever a constraint or a lexicon forbids some arcs from appearing in dependency tree. It may also arise as a consequence of a first-stage pruning step where some candidate arcs are eliminated; a practical instance of this will be described in the experimental section (Section 7.5).

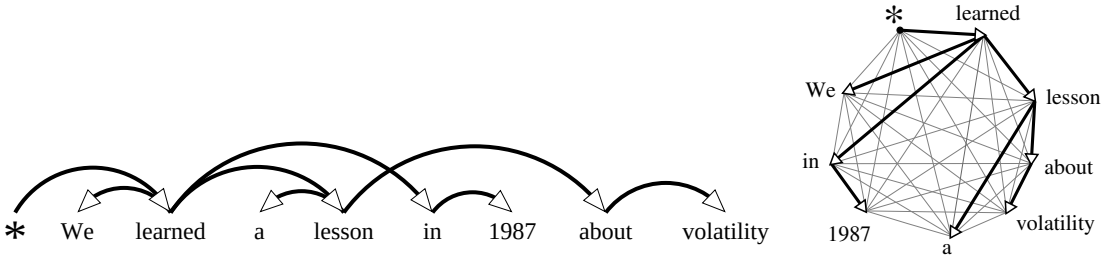


Figure 7.2: The dependency tree plotted in Figure 2.10, and its depiction as an arborescence of the fully connected graph connecting all words.

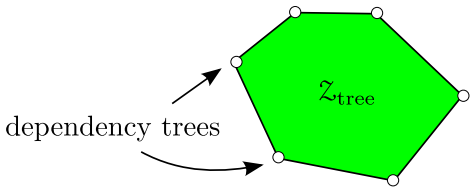


Figure 7.3: The arborescence polytope $\mathcal{Z}_{\text{tree}}$. Each vertex corresponds to a dependency parse tree in the output set $\mathcal{Y}(x)$.

for a particular sentence in Figure 7.2. From Definition 2.2, we have that the set $\mathcal{Y}(x)$ of legal dependency parse trees of x is in bijection with the set of 0-arborescences of $\mathscr{D}$—*i.e.*, each arborescence can be seen as a potential dependency tree for x. We can thus regard the dependency parsing problem as a search problem in the set of arborescences of $\mathscr{D}$.

We next provide a polyhedral characterization of this set of arborescences. Given a dependency tree $y \in \mathcal{Y}(x)$, we define its *incidence vector* $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{A}|}$ as:

$$\boldsymbol{z} := (z_a)_{a \in \mathcal{A}}, \quad \text{with } z_a = [\![a \in y]\!]; \tag{7.1}$$

that is, $\boldsymbol{z}$ is a binary vector indexed by all potential arcs, where each component z_a is 1 if arc a is in the tree y, and 0 otherwise. The set of all incidence vectors of the trees in $\mathcal{Y}(x)$ is a set of points in $\{0,1\}^{|\mathcal{A}|}$. Its convex hull is a polyhedron called the *arborescence polytope*, which we denote by $\mathcal{Z}_{\text{tree}}$. Since all incidence vectors lie in $\{0,1\}^{|\mathcal{A}|}$, none can be expressed as a convex combination of the others; therefore they are precisely the vertices of $\mathcal{Z}_{\text{tree}}$. In other words, each vertex of the arborescence polytope $\mathcal{Z}_{\text{tree}}$ can be identified with a dependency tree in $\mathcal{Y}(x)$. Figure 7.3 provides a schematic representation.

So far, the fact that we have been able to characterize our output set geometrically as the set of vertices of a polytope did not buy us much—we still have an exponential number of vertices to represent. It is desirable to represent the arborescence polytope in terms of a set of linear inequalities. The Minkowski-Weyl theorem (Rockafellar, 1970) guarantees that such a representation must exist—*i.e.*, there must be a $P \in \mathbb{N}$, some P-by-$|\mathcal{A}|$ matrix $\mathbf{A}$, and some vector $\boldsymbol{b}$ in $\mathbb{R}^P$ such that $\mathcal{Z}_{\text{tree}}$ can be represented as

$$\mathcal{Z}_{\text{tree}} = \{\boldsymbol{z} \in \mathbb{R}^{|\mathcal{A}|} \mid \mathbf{A}\boldsymbol{z} \leq \boldsymbol{b}\}. \tag{7.2}$$

However, it is not obvious how to obtain a *concise* representation—one in which the number of linear inequalities P grows only polynomially with the number of words L. This is the

subject of the following sections. We will describe three possible representations of $\mathcal{Z}_{\text{tree}}$:

- a *cycle-based representation*, where P has exponential growth with L; this was the representation implicitly used by Riedel and Clarke (2006);
- a *single-commodity flow representation* requiring $O(L^2)$ variables and constraints—this is only an approximate representation, as it defines an outer bound of $\mathcal{Z}_{\text{tree}}$ and not $\mathcal{Z}_{\text{tree}}$ itself;
- a *multi-commodity flow representation* requiring $O(L^3)$ variables and constraints, and which we will show to be exact.

The last two representations are inspired by the commodity flow models described by Magnanti and Wolsey (1994) in the combinatorial optimization literature; while Magnanti and Wolsey (1994) addressed the *undirected* minimum spanning tree problem, their analysis carries over to our case, where arcs are directed.[3]

7.2.2 A Cycle-Based Representation

Recall the digraph $\mathcal{D} = (\mathcal{N}, \mathcal{A})$ defined in the previous section. Clearly, for an arbitrary subset of arcs $\mathcal{B} \subseteq \mathcal{A}$ to be a 0-arborescence of $\mathcal{D}$ it is necessary and sufficient that it touches all the nodes, and that the overall structure is a directed tree rooted at 0, *i.e.*, a graph free of cycles where 0 is the only node without a parent. Formally:

Proposition 7.1 *A subgraph $y = (\mathcal{N}, \mathcal{B})$ is a legal dependency tree* (i.e., $y \in \mathcal{Y}(x)$) *if and only if the following conditions are met:*

1. *Each node in $\mathcal{N} \setminus \{0\}$ has exactly one incoming arc in $\mathcal{B}$,*
2. *0 has no incoming arcs in $\mathcal{B}$,*
3. *$\mathcal{B}$ does not contain cycles.*

Proof. Straightforward from the definition of directed trees and arborescences. ∎

For each node $n \in \mathcal{N}$, let $\delta^-(n) := \{(i,j) \in \mathcal{A} \mid j = n\}$ denote its set of incoming arcs, and $\delta^+(n) := \{(i,j) \in \mathcal{A} \mid i = n\}$ denote its set of outgoing arcs. The two first conditions can be easily expressed by linear constraints on the incidence vector z:

$$\sum_{a \in \delta^-(j)} z_a = 1, \quad \text{for every } j \in \mathcal{N} \setminus \{0\}, \tag{7.3}$$

$$\sum_{a \in \delta^-(0)} z_a = 0. \tag{7.4}$$

The third condition is somewhat harder to express. We need to enumerate each possible cycle and forbid all configurations that contain that cycle. Denote by $\mathscr{C} \subseteq 2^{\mathcal{A}}$ the set of all

[3] A similar representation has been proposed by Wong (1984). See Schrijver (2003, p. 903) for other pointers to the literature.

possible cycles, where each cycle $\mathcal{C} \in \mathscr{C}$ is represented by the set of arcs it contains (hence $\mathcal{C}$ is a subset of $\mathcal{A}$). The following constraints do the job:

$$\sum_{a \in \mathcal{C}} z_a \leq |\mathcal{C}| - 1, \quad \text{for every } \mathcal{C} \in \mathscr{C}. \tag{7.5}$$

Unfortunately, the set $\mathscr{C}$ grows exponentially fast with L, which yields an exponential number of constraints of the form in Eq. 7.5. To obviate this problem, Riedel and Clarke (2006), when optimizing over the constraint set defined by Eqs. 7.3–7.5, employed a cutting-plane method, where constraints are added incrementally as violations are detected. The resulting algorithm is still slow, and an arc-factored model is used as a surrogate during training (i.e., the hard constraints are only used at test time), which implies a discrepancy between the model that is optimized and the one that is actually going to be used. The next sections propose ILP formulations that eliminate the need for cycle constraints; in fact, they require only a *polynomial* number of constraints.

7.2.3 A Single Commodity Flow Representation

The reason why the cycle-based representation is not concise is that it requires to forbid every cycle, in accordance to condition 3 of Proposition 7.1. However, there is an alternative condition, as we state next.

Proposition 7.2 *A subgraph $y = (\mathcal{N}, \mathcal{B})$ is a legal dependency tree (i.e., $y \in \mathcal{Y}(x)$) if and only if it satisfies conditions 1 and 2 of Proposition 7.1 plus the following replacement of condition 3:*

3′. *$\mathcal{B}$ is connected. In other words, there is a directed path from the root 0 to every other node in $\mathcal{N} \setminus \{0\}$ consisting only of arcs in $\mathcal{B}$.*

Proof. Clearly, any directed tree which spans $\{0, \ldots, L\}$ must satisfy conditions 1 and 2 and be connected, hence it also satisfies 3′. It suffices to show that conditions 1, 2 and 3′ imply condition 3 of Proposition 7.1, *i.e.*, that these conditions prevent cycles. We prove this by contradiction. Let y satisfy 1, 2 and 3′. Conditions 1 and 2 imply that there are L arcs in $\mathcal{B}$. Suppose that y has a cycle $\mathcal{C}$ of length L'; then there are L' arcs in $\mathcal{C}$, covering L' nodes, and $L - L'$ arcs in $\mathcal{B} \setminus \mathcal{C}$. In order for $\mathcal{B}$ be connected, $\mathcal{B} \setminus \mathcal{C}$ must cover the remaning $L - L' + 1$ nodes plus at least one node in $\mathcal{C}$, a total of $L - L' + 2$ nodes. This cannot be done with only $L - L'$ arcs while maintaining connectivity within $\mathcal{B} \setminus \mathcal{C}$. ∎

Proposition 7.2 states that conditions 1-2-3 are equivalent to 1-2-3′, in the sense that both define the same set $\mathcal{Y}(x)$. However, as we will see, the latter set of conditions is more convenient. Connectedness of graphs can be imposed via *flow constraints*, by requiring that, for any $n \in \mathcal{N} \setminus \{0\}$, there is a directed path in $\mathcal{B}$ connecting 0 to n.

We first describe an inexact *single commodity flow* formulation that requires only $O(L^2)$ variables and constraints. Our formulation adapts the one described by Magnanti and Wolsey (1994, p. 39), originally proposed for the *undirected* minimum spanning tree problem. Under this model, the root node must send one unit of flow to every other node. By making use of extra variables, $\boldsymbol{\phi} := (\phi_a)_{a \in \mathcal{A}}$, to denote the flow of commodities through each arc, we are led to the following constraints in addition to Eqs. 7.3–7.4:

- Root sends flow L:

$$\sum_{a \in \delta^+(0)} \phi_a = L. \tag{7.6}$$

- Each node consumes one unit of flow:

$$\sum_{a \in \delta^-(j)} \phi_a - \sum_{a \in \delta^+(j)} \phi_a = 1, \quad \text{for every } j \in \mathcal{N} \setminus \{0\}. \tag{7.7}$$

- Flow is zero on disabled arcs:

$$\phi_a \leq L z_a, \quad \text{for every } a \in \mathcal{A}. \tag{7.8}$$

- Each arc indicator lies in the unit interval:

$$z_a \in [0, 1], \quad \text{for every } a \in \mathcal{A}. \tag{7.9}$$

These linear constraints project an outer bound of the arborescence polytope, *i.e.*,

$$\begin{aligned} \mathcal{Z}_{\text{sc}} &:= \left\{ \boldsymbol{z} \in \mathbb{R}^{|A|} \mid \exists \boldsymbol{\phi} : \ (\boldsymbol{z}, \boldsymbol{\phi}) \text{ satisfy Eqs. 7.3–7.4 and Eqs. 7.6–7.9} \right\} \\ &\supseteq \mathcal{Z}_{\text{tree}}. \end{aligned} \tag{7.10}$$

Furthermore, the *integer* points of $\mathcal{Z}_{\text{sc}}$ are precisely the incidence vectors of dependency trees in $\mathcal{Y}(x)$; these are obtained by inserting integer constraints, or equivalently by replacing Eq. 7.9 by

$$z_a \in \{0, 1\}, \quad \text{for every } a \in \mathcal{A}. \tag{7.11}$$

A representation as in Eq. 7.10 is called a *lifted* representation, in the sense that it involves extra variables (in this case, the flows $\boldsymbol{\phi}$) to build a higher-dimensional polytope which is then projected onto the original $\boldsymbol{z}$-space (see the formal definition of *lifting* in Appendix B).

7.2.4 A Multi-Commodity Flow Representation

We next present a *multi-commodity flow representation* of the arborescence polytope, which is still polynomial (even though it has cubic rather than quadratic dependency on the sentence length L), but has the advantage of being exact. Like the single-commodity representation, it relies on the characterization of arborescences in terms of a connectedness constraint, as put forth by Proposition 7.2. It is adapted from the *multicommodity directed flow* model of Magnanti and Wolsey (1994, p. 44), albeit not exactly copied, since the latter is for undirected spanning trees. We prove in Proposition 7.3 that this representation is exact.

In the multi-commodity flow model, every node $k \neq 0$ defines a commodity: one unit of commodity k originates at the root node and must be delivered to node k; the variable ϕ_{ij}^k denotes the flow of commodity k in arc (i, j). This model differs from the single commodity model by replacing the constraints in Eqs. 7.6–7.9 by the following ones in Eqs. 7.12–7.15:

- The root sends one unit of commodity to each node:

$$\sum_{a \in \delta^-(0)} \phi_a^k - \sum_{a \in \delta^+(0)} \phi_a^k = -1, \qquad \text{for every } k \in \mathcal{N} \setminus \{0\}. \tag{7.12}$$

- Any node consumes its own commodity and no other:

$$\sum_{a \in \delta^-(j)} \phi_a^k - \sum_{a \in \delta^+(j)} \phi_a^k = [\![j = k]\!], \qquad \text{for every } j, k \in \mathcal{N} \setminus \{0\}. \tag{7.13}$$

- Disabled arcs do not carry any flow:

$$\phi_a^k \leq z_a, \qquad \text{for every } a \in \mathcal{A} \text{ and } k \in \mathcal{N}. \tag{7.14}$$

- All variables lie in the unit interval:

$$z_a \in [0,1], \quad \phi_a^k \in [0,1], \qquad \text{for every } a \in \mathcal{A} \text{ and } k \in \mathcal{N}. \tag{7.15}$$

Consider the polytope obtained by imposing the constraints in Eqs. 7.3–7.4 and Eqs. 7.12–7.15 and projecting out the flow variables $\boldsymbol{\phi}$:

$$\mathcal{Z}_{\text{mc}} \;:=\; \left\{ \boldsymbol{z} \in \mathbb{R}^{|A|} \mid \exists \boldsymbol{\phi} : \; (\boldsymbol{z}, \boldsymbol{\phi}) \text{ satisfy Eqs. 7.3–7.4 and Eqs. 7.12–7.15} \right\}. \tag{7.16}$$

We then have the following:

Proposition 7.3 *The multi-commodity flow representation is exact:*

$$\mathcal{Z}_{\text{mc}} = \mathcal{Z}_{\text{tree}}. \tag{7.17}$$

Proof. Let $\mathcal{S} \in \mathcal{N}$ be an arbitrary subset of the nodes. We denote by $\delta^+(\mathcal{S}) := \{(i,j) \in \mathcal{A} \mid i \in \mathcal{S}, j \notin \mathcal{S}\}$ the set of arcs which depart from $\mathcal{S}$. The set of these arcs form a *cut* of the graph, associated with the partition $\mathcal{S} \cup (\mathcal{N} \setminus \mathcal{S})$. One way of imposing connectedness is by requiring every possible cut to have at least one arc installed. This can be expressed by the following set of constraints:

$$\sum_{a \in \delta^+(\mathcal{S})} z_a \geq 1, \quad \text{for every proper subset } \mathcal{S} \subset \mathcal{N} \text{ s.t. } 0 \in \mathcal{S}. \tag{7.18}$$

Magnanti and Wolsey (1994, Proposition 5.2) show that the constraints in Eq. 7.18, along with Eqs. 7.3–7.4 and the non-negativity constraint $\boldsymbol{z} \geq \boldsymbol{0}$, are an exact representation of the arborescence polytope $\mathcal{Z}_{\text{tree}}$.[4] We show that our multi-commodity flow representation is equivalent to that set of constraints, by applying the min-cut max-flow theorem (Schrijver, 2003). For every node $k \in \mathcal{N} \setminus \{0\}$, consider the set $\mathcal{S}_k$ of all possible proper subsets $\mathcal{S} \subset \mathcal{N}$ s.t. $0 \in \mathcal{S}$ and $k \notin \mathcal{S}$. Then, express all constraints in Eq. 7.18 as L non-linear constraints, one per $k \in \mathcal{N} \setminus \{0\}$:

$$\min_{\mathcal{S} \in \mathcal{S}_k} \sum_{a \in \delta^+(\mathcal{S})} z_a \geq 1. \tag{7.19}$$

[4] Note that there are exponentially many constraints in Eq. 7.18, one per each proper subset of nodes, hence that formulation, albeit exact, is not practical.

Note that this inequality is actually an equality, since for the set $\mathcal{S} = \mathcal{N} \setminus \{k\}$, the set $\delta^+(\mathcal{S})$ equals $\delta^-(k)$ and from the single parent constraint (Eq. 7.3) we have $\sum_{a \in \delta^-(k)} z_a = 1$. Hence, we can write instead

$$\min_{\mathcal{S} \in \mathcal{S}_k} \sum_{a \in \delta^+(\mathcal{S})} z_a = 1. \tag{7.20}$$

Now, for a fixed $\boldsymbol{z}$, we can interpret the minimization in Eq. 7.20 as a min-cut problem with source 0 and sink k, where the capacity of each arc a is given by z_a. The constraint states that the min-cut is 1; from the min-cut max-flow theorem, this value must equal the solution of the equivalent max-flow problem; the capacity and the conservation-of-flow constraints of that max-flow problem are precisely the constraints that we have written in Eqs. 7.12–7.14. Putting all the pieces together, we have that the multi-commodity flow representation is equivalent to the exact cut-based representation with the constraint in Eq. 7.18; therefore it is also exact. ∎

To sum up, we have introduced multi-commodity flow constraints in Eqs. 7.12–7.14 which provide a lifted representation of the arborescence polytope $\mathcal{Z}_{\text{tree}}$. By Proposition 7.3, this representation is exact, without the need of integer constraints. We will exploit this fact in the sequel.

7.2.5 Arc-Factored Model

Now that we have a polyhedral representation of the arborescence polytope, it is straightforward to formulate arc-factored dependency parsing as an LP.

At first sight, this may look like a futile exercise: as we have seen in Section 2.3, the arc-factored model, which is rather simplistic, already permits efficient parsing with graph-based algorithms—asymptotic runtimes are $O(L^3)$ with the Chu-Liu-Edmonds' algorithm (Chu and Liu, 1965; Edmonds, 1967), or $O(L^2)$ with Tarjan's or Gabow's algorithm (Tarjan, 1977; Gabow et al., 1986). In fact, the $O(L^2)$ bound is optimal, since any algorithm needs to touch every potential arc, and there is a quadratic number of them.[5] Hence, there is no advantage at all in casting arc-factored parsing as a linear program and employing an off-the-shelf solver, as those asymptotic runtimes are unbeatable, and gains in the non-asymptotic regime are very unlikely. It is legitimate to ask, *why are we interested in these LP formulations in the first place?* We will defer the answer to this question until we reach Sections 7.2.6–7.2.10, when we enrich the model with non-arc-factored features. For now, consider the arc-factored case as an instructive exercise.

By storing the arc-local feature vectors into the columns of a matrix $\mathbf{F}(x) := [\boldsymbol{f}_a(x)]_{a \in \mathcal{A}}$, and defining the score vector $\boldsymbol{s} := \mathbf{F}(x)^\top \boldsymbol{w}$ (each entry is an arc score) the inference problem, which originally is

$$\begin{array}{ll} \text{maximize} & \boldsymbol{w} \cdot \boldsymbol{f}(x, y) \\ \text{w.r.t.} & y \in \mathcal{Y}(x) \end{array} \tag{7.21}$$

[5] In practice, most parsers include a preprocessing stage that prunes the set of potential arcs beforehand, hence in full rigor the bound may not be optimal if we consider this pruning. Gabow's algorithm takes $O(|\mathcal{A}| + L \log L)$ time for a general set of potential arcs $\mathcal{A}$; with sufficiently aggressive pruning, we could have a constant number of potential heads for each word, making $\mathcal{A}$ have size $O(L)$. The asymptotic runtime would then be $O(L \log L)$—which is still optimal up to logarithmic terms.

can be rewritten, using the multi-commodity flow representation, as the LP

$$\begin{array}{ll} \text{maximize} & \boldsymbol{s} \cdot \boldsymbol{z} \\ \text{w.r.t.} & \boldsymbol{z} \in \mathcal{Z}_{\text{mc}}(x); \end{array} \tag{7.22}$$

which in terms of linear inequalities (and adding the multi-commodity flow variables) takes the form

$$\begin{array}{ll} \text{maximize} & \boldsymbol{s} \cdot \boldsymbol{z} \\ \text{w.r.t.} & \boldsymbol{z}, \boldsymbol{\phi} \\ \text{s.t.} & \mathbf{A} \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{\phi} \end{bmatrix} \leq \boldsymbol{b}, \end{array} \tag{7.23}$$

where $\mathbf{A}$ is a sparse constraint matrix (with $O(L^3)$ non-zero elements), and $\boldsymbol{b}$ is the constraint vector; $\mathbf{A}$ and $\boldsymbol{b}$ encode the constraints in Eqs. 7.3–7.4 and Eqs. 7.12–7.15. Note that we do not need any integer constraints, since the solution of this LP is guaranteed to be integer, by virtue of Proposition 7.3.

An alternative (pursued in Martins et al. 2009b) is to cast arc-factored dependency parsing as an ILP using the single-commodity flow formulation (*i.e.*, letting $\mathbf{A}$ and $\boldsymbol{b}$ encode the constraints in Eqs. 7.3–7.4 and Eqs. 7.6–7.9 instead, and constraining $\boldsymbol{z}$ to be integer). That approach yields only a quadratic number of variables and constraints, but requires the integer constraint for exactness; by dropping it the problem becomes the LP relaxation. Although this procedure gives competitive results (as attested in the experimental section of Martins et al. 2009b), we will not pursue that alternative here for the sake of simplicity.

7.2.6 Pairwise Interactions: Siblings, Grandparents and Head Bigrams

In practice, as we stated in Section 2.3, the performance of the arc-factored model can be rather poor, due to the strong factorization assumptions implied by this model. As shown by McDonald and Pereira (2006) and Carreras (2007), the inclusion of features that correlate sibling and grandparent arcs may be highly beneficial.[6] Unfortunately, any attempt to extend the arc-factored model to include pairwise arc information renders the problem NP-hard (McDonald and Satta, 2007), so we must rely on approximate methods. This is where our (integer) concise LP formulations can be useful: if we manage to extend them to accommodate this additional pairwise information then we will have a concise ILP for dependency parsing with rich features; relaxing this ILP will yield an approximate problem which is solvable in polynomial time.

To accomplish the aforementioned goal, we employ a *linearization strategy* which was first proposed in the context of pseudo-Boolean optimization, for handling higher-order interactions among Boolean variables (Boros and Hammer, 2002). Suppose that we have a model with features of the form $\boldsymbol{f}_{a_1,\ldots,a_K}(x)$ (*i.e.*, features whose values depend on the simultaneous inclusion of arcs $a_1, \ldots, a_K$ on a candidate dependency tree). Define extra variables $z_{a_1 \ldots a_K} := z_{a_1} \wedge \ldots \wedge z_{a_K}$. This logical relation can be expressed by the following

[6] By *sibling features* we mean features that depend on pairs of sibling arcs (i.e., of the form (h, m) and (h, s)); by *grandparent features* we mean features that depend on pairs of grandparent arcs (of the form (h, m) and (g, h)). See Figure 7.1.

$O(K)$ agreement constraints:

$$\begin{aligned} z_{a_1 \dots a_K} &\leq z_{a_i}, \quad \text{for each } i = 1, \dots, K, \\ z_{a_1 \dots a_K} &\geq \sum_{i=1}^{K} z_{a_i} - K + 1. \end{aligned} \tag{7.24}$$

We apply this strategy to pairwise interactions, in which we assume to be given a set of arc pairs $\mathcal{P}$. In the sequel, we consider the following pairs, all illustrated in Figure 7.1:

- Sibling arcs, of the form (h, m) and (h, s);[7]
- Grandparent arcs, of the form (h, m) and (g, h);
- Head bigram arcs, of the form (h, m) and $(h', m + 1)$.

Each of these three groups contains $O(L^3)$ arcs, so $|\mathcal{P}| = O(L^3)$.[8] For each pair of arcs $(a, b) \in \mathcal{P}$, we assume that we have a score s_{ab} given by the inner product of a feature vector $\boldsymbol{f}_{ab}(x)$ with the weight vector $\boldsymbol{w}$; this score contributes to the total objective if arcs a and b are both included. Accordingly, we add new variables $\boldsymbol{z}_{\mathcal{P}} := (z_{ab})_{(a,b)\in\mathcal{P}}$. The linearization strategy above yields the following agreement constraints between the $\boldsymbol{z}_{\mathcal{P}}$ and the already existing $\boldsymbol{z}$-variables, for every $(a, b) \in \mathcal{P}$:

$$\begin{aligned} z_{ab} &\leq z_a \\ z_{ab} &\leq z_b \\ z_{ab} &\geq z_a + z_b - 1. \end{aligned} \tag{7.25}$$

Composing this with the optimization problem in Eq. 7.22, we finally get

$$\begin{aligned} &\text{maximize} && \textstyle\sum_{a\in\mathcal{A}} s_a z_a + \sum_{(a,b)\in\mathcal{P}} s_{ab} z_{ab} \\ &\text{w.r.t.} && \boldsymbol{z}, \boldsymbol{z}_{\mathcal{P}} \\ &\text{s.t.} && \boldsymbol{z} \in \mathcal{Z}_{\text{mc}}, \\ & && z_{ab} \leq z_a, \\ & && z_{ab} \leq z_b, \\ & && z_{ab} \geq z_a + z_b - 1, \quad \text{for every } (a, b) \in \mathcal{P}. \end{aligned} \tag{7.26}$$

This is also a linear program with $O(L^3)$ variables and constraints. However, even though $\mathcal{Z}_{\text{mc}}$ coincides with the arborescence polytope $\mathcal{Z}_{\text{tree}}$ (and therefore all its vertices are integer), the inclusion of the extra pairwise variables and the extra constraints make the linear program no longer guaranteed to have integer solutions. For the optimization problem to be exact, we need to add the constraint

$$\boldsymbol{z} \text{ integer.} \tag{7.27}$$

This fact is not surprising, since we know from McDonald and Satta (2007) that the original problem is NP-hard. Yet, relaxing the integer constraint in Eq. 7.27 yields a very powerful approximate parser, as we will see in the experimental section.

[7] Note that these are *arbitrary* siblings, not necessarily consecutive as the ones considered, *e.g.*, by Eisner (1996) and McDonald and Pereira (2006). Those will be discussed in Section 7.2.7.

[8] Other pairs of arcs can be useful, for example Smith and Eisner (2008) consider pairs of crossing arcs which are $O(L^4)$ in total.

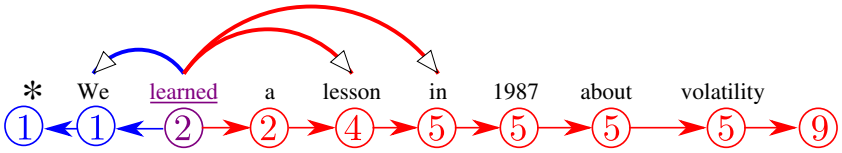


Figure 7.4: Head automata for the word *learned* in the sentence of Figure 7.2, seen as sequencial models. We depict in blue the left automaton and in red the right automaton. The states of each model (shows as numerals inside the circles) represent the position of the last modifier.

7.2.7 Head Automata and Consecutive Siblings

We have just described a strategy that allows us to incorporate features that look at *sibling* arcs; however, those features ignore the relative position of the siblings. In particular, they cannot tell if two sibling arcs (h, m) and (h, s) are *consecutive*, in the sense that there is no arc (h, m') in the tree with m' between m and s. In constrast, previous models for projective and non-projective parsing (Eisner, 1996; Alshawi, 1996; Koo et al., 2010) have made use of *head automata* for modeling *consecutive siblings*. Such models allow incorporating features that fire when m is the closest child of h (either on the left or on the right side), or when (h, m) and (h, s) are two consecutive sibling arcs on the same side.

Note that consecutive sibling features are neither more nor less expressive than the ones that refer to *arbitrary* siblings: the latter allow correlating siblings that are far apart, and on either side of the head, regardless of what happens in-between; they appear to be particularly useful in free-order languages, but not only (we will see in Section 7.5 that these features also improve the accuracy in English datasets). The former only look at consecutive ones on the same side, but model the fact that no other sibling exists in-between—this is commonly refered to as *horizontal Markovization* (the term *vertical Markovization* refers to grandparent arcs). We point out that the strategy that we describe here to handle consecutive siblings is different from the one that we have employed in our previously published work (Martins et al., 2009b); the one we present now is better, since it leads to tighter relaxations while requiring the same number of variables and constraints, namely $O(L^3)$.

We start by describing head automata models and their underlying polytopes, which we call *head automata polytopes*. Then, we will see how a head automaton can be seen as a solver of a linear optimization problem over the head automaton polytope.

Without loss of generality, let us fix a word position h (which will be the head word) and consider its potential dependents on the *right* side—it is straightforward to generalize what follows for the sequence of left-side dependents. Define the following "consecutive sibling" indicator variables, for each h, m, s with $0 \leq h \leq m < s \leq L+1$:[9]

$$z^{\text{cs}}_{hms} := \begin{cases} 1 & \text{if } (h, m) \text{ and } (h, s) \text{ are consecutive siblings,} \\ 0 & \text{otherwise.} \end{cases} \tag{7.28}$$

Given h, we can stack all these variables in a vector of right-dependent indicators $z^{\text{cs}}_{h,\rightarrow} :=$

[9] For convenience, we define also these variables for the case where $m = h$ (in which case they indicate that s is the first child of h) and for $s = L+1$ (which indicate that m is the last child of h). When $h = m$ and $s = L+1$, they indicate that h does not have children on the right side.

$(z^{\text{cs}}_{hms})_{h\le m<s\le L+1}$, and analogously we can form a vector for the left-dependents, $z^{\text{cs}}_{h,\leftarrow} := (z^{\text{cs}}_{hms})_{h\ge m>s\ge 0}$. We can further stack all these vectors in a $O(L^3)$-dimensional vector, $z^{\text{cs}} := ((z^{\text{cs}}_{h,\rightarrow})_h, (z^{\text{cs}}_{h,\leftarrow})_h)$. We assume there is a score vector s^{cs} containing the corresponding scores, in addition to the already described arc-factored scores. That is:

- s_{hm} is the score associated with word m being a modifier of h,
- s^{cs}_{hhs} is the score associated with word s being the *first* modifier of h,
- $s^{\text{cs}}_{hm(L+1)}$ is the score associated with word m being the *last* modifier of h,
- s^{cs}_{hms} is the score associated with words (h, m) and (h, s) being *consecutive* siblings.

In a head automaton, the goal is, for a fixed h, to find the most likely sequence of dependents on one of the sides. We will assume that the automaton is for $h = 0$ and is looking for right-dependents; the general case follows easily. The problem is equivalent to that of finding a Viterbi path $(m_1, \ldots, m_L)$ for a chain model whose possible states for M_j are $\{0, \ldots, j\}$; the event $M_j = a$ means that "the last modifier, up to j, is a." This is illustrated in Figure 7.4. Between words j and $j+1$, only two transitions may occur: either $M_{j+1} = m_j$ (which happens if $j+1$ is not a modifier), or $M_{j+1} = j+1$ (which happens otherwise). Since this is a chain model, the marginal polytope is exactly characterized by *local consistency constraints* and *hard constraints* (see Chapter 5). Let $\mu_i(a)$ be the posterior marginal for the event $M_i = a$, and $\mu_{i,i+1}(a, b)$ the posterior marginal for the event $M_i = a \wedge M_{i+1} = b$. Local consistency constraints assert that

$$\sum_{a=0}^{i} \mu_i(a) = 1, \quad i \in \{1, \ldots, L\} \tag{7.29}$$

$$\sum_{b=0}^{i+1} \mu_{i,i+1}(a, b) = \mu_i(a), \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, L\} \tag{7.30}$$

$$\sum_{a=0}^{i} \mu_{i,i+1}(a, b) = \mu_{i+1}(b), \quad i \in \{1, \ldots, L\}, b \in \{1, \ldots, i+1\} \tag{7.31}$$

$$\mu_i(a) \ge 0, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\} \tag{7.32}$$

$$\mu_{i,i+1}(a, b) \ge 0, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\}, b \in \{1, \ldots, i+1\}. \tag{7.33}$$

Hard constraints assert that impossible configurations must receive zero marginals:

$$\mu_{i,i+1}(a, b) = 0, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\}, b \notin \{a, i+1\}. \tag{7.34}$$

Plugging (7.34) in (7.30)–(7.31) yields:

$$\mu_{i,i+1}(a, a) + \mu_{i,i+1}(a, i+1) = \mu_i(a), \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\} \tag{7.35}$$

$$\mu_{i,i+1}(b, b) = \mu_{i+1}(b), \quad i \in \{1, \ldots, L\}, b \in \{1, \ldots, i+1\} \tag{7.36}$$

$$\sum_{a=0}^{i} \mu_{i,i+1}(a, i+1) = \mu_{i+1}(i+1), \quad i \in \{1, \ldots, L\}, \tag{7.37}$$

and plugging further (7.36) in (7.35) yields:

$$\mu_{i+1}(a) + \mu_{i,i+1}(a, i+1) = \mu_i(a), \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, L\}. \tag{7.38}$$

The marginal polytope is thus characterized by (7.29), (7.38), (7.37), and (7.32)–(7.33). We next make the variable replacements

- $z_{hi} = \mu_i(i)$, the posterior marginal for the event that i is a modifier of h;
- $z^{\text{sc}}_{ha(i+1)} = \mu_{i,i+1}(a, i+1)$, the posterior marginal for the event that (h, a) and $(h, i+1)$ are consecutive siblings;
- $\omega_{hai} := \mu_i(a)$, an extra variable for the event that, up to i, the last modifier of h is a.

The overall optimization problem associated with the $(h, \rightarrow)$-automaton becomes that of maximizing

$$\sum_{j=h+1}^{L} s_{hj} z_{hj} + \sum_{j=h}^{L} \sum_{k=j+1}^{L} s_{hjk} z_{hjk} \tag{7.39}$$

subject to:

$$\omega_{hii} = z_{hi}, \quad i \in \{1, \ldots, L\} \tag{7.40}$$

$$\sum_{a=0}^{i} \omega_{hai} = 1, \quad i \in \{1, \ldots, L\} \tag{7.41}$$

$$\omega_{ha(i+1)} + z^{\text{cs}}_{ha(i+1)} = \omega_{hai}, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\} \tag{7.42}$$

$$\sum_{a=0}^{i} z^{\text{cs}}_{ha(i+1)} = z_{h(i+1)}, \quad i \in \{1, \ldots, L\} \tag{7.43}$$

$$\omega_{hai} \geq 0, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\} \tag{7.44}$$

$$z^{\text{cs}}_{ha(i+1)} \geq 0, \quad i \in \{1, \ldots, L\}, a \in \{1, \ldots, i\}. \tag{7.45}$$

Let $\boldsymbol{z}_h := (z_{hm})_{m=h+1}^{L}$ and $\boldsymbol{\omega}_h := (\omega_{ham})_{h \leq a \leq m \leq L}$. We define the $(h, \rightarrow)$-head automaton polytope as the set

$$\mathcal{Z}_{\text{head},h,\rightarrow} := \left\{ (\boldsymbol{z}_h, \boldsymbol{z}^{\text{cs}}_{h,\rightarrow}) \;\middle|\; \exists \boldsymbol{\omega} \;:\; (\boldsymbol{z}_h, \boldsymbol{z}^{\text{cs}}_{h,\rightarrow}, \boldsymbol{\omega}_h\} \text{ satisfy Eqs. 7.40–7.45} \right\}. \tag{7.46}$$

This polytope is isomorphic to the marginal polytope underlying the chain model associated with the head automaton. As such, its vertices are integer and correspond to the valid assignments of the $(\boldsymbol{z}_h, \boldsymbol{z}^{\text{cs}}_{h,\rightarrow})$-variables. Note that the head automata are all disjoint, in the sense that each of them involves their own set of variables, with no overlaps. We define the *head automaton polytope* $\mathcal{Z}_{\text{head}}$ as the Euclidean product of all $(h, \rightarrow)$- and $(h, \leftarrow)$-head automaton polytopes:

$$\mathcal{Z}_{\text{head}} := \prod_{h=0}^{L} \mathcal{Z}_{\text{head},h,\leftarrow} \times \prod_{h=0}^{L} \mathcal{Z}_{\text{head},h,\rightarrow}. \tag{7.47}$$

To sum up, our LP formulation can be extended for accommodating consecutive siblings. All we need is to add the $O(L^3)$ extra variables and constraints in Eqs. 7.40–7.45. Even though both polytopes $\mathcal{Z}_{\text{mc}}$ and $\mathcal{Z}_{\text{head}}$ have only integer vertices, that does not happen with their intersection. Therefore, the resulting LP formulation that composes the arc-factored model with the head automata is not exact; if one adds the integer constraints in Eq. 7.27, then we obtain an ILP which is exact. Without the integer constraints, one obtains an LP relaxation which is equivalent to one used in the sibling model in Koo et al. (2010).

7.2.8 Valency Indicators

A crucial fact about dependency grammars is that words have preferences about the number and arrangement of arguments and modifiers they accept. Therefore, it may be desirable to include features that indicate, for a candidate dependency tree, how many dependency links depart from each word; denote these quantities by $v_i := \sum_{a \in \delta^+(i)} z_a$, for each $i \in \mathcal{N}$. We call v_i the *valency* of the ith word.

In Martins et al. (2009b), we added valency indicators $z_{ik}^{\text{val}} := [\![v_i = k]\!]$ for $i \in \mathcal{N}$ and $k = 0, \ldots, L$. This way, we were able to penalize candidate dependency trees that assign unusual valencies to some of their words, by specifying a individual cost for each possible value of valency. The following $O(|L|^2)$ constraints encode the agreement between valency indicators and the other variables:

$$\begin{aligned} \sum_{k=0}^{L} k z_{ik}^{\text{val}} &= \sum_{a \in \delta^+(i)} z_a, \quad i \in \mathcal{N} \\ \sum_{k=0}^{L} z_{ik}^{\text{val}} &= 1, \quad i \in \mathcal{N} \\ z_{ik}^{\text{val}} &\geq 0, \quad i \in \mathcal{N}, k \in \{0, \ldots, L\}. \end{aligned} \tag{7.48}$$

Experimentally, valency features did not help much, so we do not consider them further.

7.2.9 Directed Path Indicators

One of the key advantages of the multicommodity flow model that we have presented in Section 7.2.4 is that it directly models ancestor-descendant relationships—namely, if an arc (i, j) carries flow directed to a node k (*i.e.*, if $\phi_{ij}^k = 1$), then there is a directed path in the tree between j and k. Hence, we may define variables z_{jk}^{path} that indicate if there is a path from j to k. Since each node except the root has only one incoming arc, the following linear equalities are enough to describe these new variables:

$$\begin{aligned} z_{jk}^{\text{path}} &= \sum_{a \in \delta^-(j)} \phi_a^k, \quad j, k \in \mathcal{N} \setminus \{0\} \\ z_{0k}^{\text{path}} &= 1, \quad k \in \mathcal{N} \setminus \{0\}. \end{aligned} \tag{7.49}$$

We define features that fire when two words bear an ancestor-descendant relationship, giving rise to a score term in the objective that rewards/penalizes trees for which that relationship holds. For example, we might learn to prefer trees for which certain nouns descend from verbs that are semantically related.

In the very same way that arbitrary siblings can be seen as a way of relaxing the horizontal Markov property that underlies consecutive siblings, *directed path features* relax the vertical Markov property underlying *grandparent features*. This was illustrated in Figure 7.1.

Note that this is only one possible usage of ancestor-descendant relationships. There are other ways in which one can exploit this important information given as a by-product of the multicommodity flow model. For example, it is possible to include features that look at spans headed by a given word—the span boundaries are simply the leftmost and rightmost descendant of that word.

7.2.10 Nonprojective Arc Indicators

For most languages, dependency parse trees tend to be nearly projective (cf. Buchholz and Marsi 2006). We may wish to make our model capable of learning to prefer "nearly" projective parses whenever that behavior is observed in the data. We refer to the background sections on dependency parsing (Section 2.3) for the definitions of "projective arc" and "projective tree" (Definitions 2.3–2.4).

In our model, we consider features that fire for arcs which are non-projective. By definition, these are arcs whose span contains words which do not descend from the head, and as such, this can be captured by making use of the directed path variables $\boldsymbol{z}^{\text{path}}$ (Section 7.2.9). The procedure is as follows. Define indicators $\boldsymbol{z}^{\text{np}} := (z_a^{\text{np}})_{a \in \mathcal{A}}$, where

$$z_a^{\text{np}} := [\![a \in y \text{ and } a \text{ is nonprojective}]\!]. \tag{7.50}$$

From the definition of projective arcs (Definition 2.3), we have that $z_a^{\text{np}} = 1$ if and only if the arc is active ($z_a = 1$) and there is some node k in the span of $a = (i, j)$ such that $z_{ik}^{\text{path}} = 0$. We are led to the following $O(L^3)$ constraints for $(i, j) \in \mathcal{A}$:

$$\begin{aligned} z_{ij}^{\text{np}} &\leq z_{ij} \\ z_{ij}^{\text{np}} &\geq z_{ij} - z_{ik}^{\text{path}}, \quad \min(i,j) \leq k \leq \max(i,j) \\ z_{ij}^{\text{np}} &\leq -\sum_{k=\min(i,j)+1}^{\max(i,j)-1} z_{ik}^{\text{path}} + |j - i| - 1. \end{aligned} \tag{7.51}$$

7.3 Turbo Parsers and Their Factor Graphs

In this section, we show a formal connection between three different approximate dependency parsers recently proposed by Smith and Eisner (2008), Martins et al. (2009b) and Koo et al. (2010). While these parsers are differently motivated, we show that all correspond to inference in a factor graph, and all optimize objective functions over local approximations of the marginal polytope. The connection is made clear by showing the factor graphs that underlie each of these parsers, and by writing the explicit declarative optimization problem that they attempt to solve. The machinery described in Chapter 5 for constrained graphical models will allow us to recognize that each of these parsers can be regarded as a *turbo parser*: they all can be seen as approximate inference algorithms that ignore loops in a factor graph. Their success parallels similar approximations in other fields, such as statistical image processing and error-correcting coding.

We summarize the situation as follows:

- Smith and Eisner (2008) proposed a factor graph (which we reproduce in Figure 7.5) in which they run sum-product loopy BP. For the configuration that includes arc-factored and consecutive sibling features, this factor graph is precisely the one that is used by Koo et al. (2010) in their dual decomposition framework. Since the underlying factor graph explicitly contains a factor that emulates the "tree" constraint, we call it a *tree-based factor graph*.[10]

[10] Note that we are refering here to two graphs of a different nature: one is the *factor graph*, whose nodes

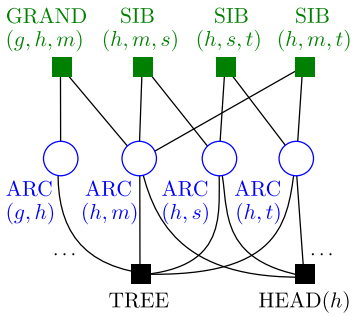


Figure 7.5: Factor graph corresponding to the dependency parsing model of (Smith and Eisner, 2008) with sibling and grandparent features. Circles denote variable nodes, and squares denote factor nodes. Note the loops created by the inclusion of pairwise factors (GRAND and SIB).

- Martins et al. (2009b) approximate parsing as the solution of a linear program, where a wide set of linear constraints encapsulate the requirement that the output is a valid dependency tree, while allowing to accommodate a large set of rich features. This is the parser to which we have devoted the entire Section 7.2.

At first sight, it is not clear what is being optimized in Smith and Eisner (2008) and what, if any, factor graph is being used by Martins et al. (2009b). Here, we fill the blanks in the two approaches: we derive explicitly the variational problem addressed in the former reference, and we provide the underlying factor graph in the latter, which will be called a *flow-based factor graph* since it is related with the multi-commodity flow formulation presented in Section 7.2. Furthermore, we will show that for some of the configurations, the local polytope approximations of all the three aforementioned parsers are exactly the same. So, where do these parsers differ? Essentially, in three aspects:

- the kind of inference they run: marginal inference in Smith and Eisner (2008), versus MAP inference in Martins et al. (2009b) and Koo et al. (2010);
- the optimization algorithms they implement: loopy belief propagation in Smith and Eisner (2008), the dual simplex method implemented by an off-the-shelf LP solver in Martins et al. (2009b), and the projected subgradient algorithm for dual decomposition in Koo et al. (2010).
- The features they use. Even when the parsers use the same parts, the part-local features might not be the same. This explains some differences in performance, *e.g.*, between Martins et al. (2009b) and Koo et al. (2010).

The next subsections characterize the tree-based and flow-based factor graphs, as well as the inference procedures run on them.

7.3.1 A Tree-Based Factor Graph

We depict in Figure 7.5 the factor graph used in the dependency parser of Smith and Eisner (2008); we denote this factor graph by $\mathcal{G}_{\text{tree}}$. There are $O(L^2)$ variable nodes in $\mathcal{G}_{\text{tree}}$, each of

represent dependency links; the other is the *dependency tree*, whose nodes are words.

them a binary variable representing a potential arc $a = (h, m)$. There are $O(L^3)$ pairwise soft factors connecting pairs of arcs that form sibling and grandparent relationships, of the kind described in Section 7.2.6. There is also a hard constraint "tree" factor connected to all the variables, which imposes that the set of active arcs form a well-defined dependency tree (*i.e.*, an arborescence). Finally, there are $O(L)$ factors that emulate head automata, of the kind described in Section 7.2.7.[11]

Let us recall how all the computations involving the tree and head automata factors are performed efficiently in Smith and Eisner's sum-product loopy BP and in Koo et al.'s projected subgradient algorithms:

- From Proposition 5.4, we have that all that is necessary for computing the sum-product messages sent by a factor is to be able to compute the marginals of the subgraph comprised only of that factor (see also Algorithm 6). As described in the background section (Section 2.3), for the tree factor this can be done in $O(L^3)$ time by invoking the matrix-tree theorem. As a consequence, $O(L^3)$ is also the time necessary to compute all outgoing messages in that factor. What about the head automata factors? As shown in Section 7.2.7, a head automaton is equivalent to a sequence model, representable as a chain graph. Therefore, the marginals can be computed by running the forward-backward algorithm, which costs $O(L^2)$ time for each automaton. Since there are $O(L)$ such automata, and we still have $O(L^3)$ pairwise factors whose messages can be computed in constant time, each iteration of loopy BP has a total runtime of $O(L^3)$.

- Let us turn to the projected subgradient algorithm of Koo et al. (2010). In this algorithm, the only operations that are necessary at factor-level are local MAP computations. For the tree factor, a MAP computation can be done in time $O(L^3)$ with Chu-Liu-Edmonds' algorithm (or even faster using Tarjan's or Gabow's algorithms, as we have seen). For the head automata, all we need is to run the Viterbi algorithm which takes $O(L^3)$ time in total ($O(L^2)$ per automaton). Hence, one iteration of the projected subgradient algorithm also has a runtime of $O(L^3)$.

What are these algorithms optimizing? The graphical model machinery described in Chapters 4–5 allows us to interpret both approximate parsers as procedures for optimizing a function over the local polytope $\text{LOCAL}(\mathcal{G}_{\text{tree}})$. Let us start by characterizing this polytope.

As before, we denote by $\mathcal{A}$ the set of candidate arcs, and by $\mathcal{P} \subseteq \mathcal{A}^2$ the set of pairs of arcs that have factors. Let $\boldsymbol{\mu} := (\boldsymbol{\mu}_{\mathcal{A}}, \boldsymbol{\mu}_{\mathcal{P}})$ with $\boldsymbol{\mu}_{\mathcal{A}} = (\mu_a(.))_{a\in\mathcal{A}}$ be the variable marginals, and $\boldsymbol{\mu}_{\mathcal{P}} = (\mu_{ab}(.,.))_{(a,b)\in\mathcal{P}}$ be the factor marginals, for the pairwise soft factors. Since all variables are binary, we will use a "Boltzmann" parametrization and will write, for each $a \in \mathcal{A}$, $\mu_a(1) = z_a$ and $\mu_a(0) = 1 - z_a$, where z_a is a variable constrained to $[0, 1]$. Let $\boldsymbol{z}_{\mathcal{A}} := (z_a)_{a\in\mathcal{A}}$; since the tree factor is a hard constraint factor, its marginal polytope is defined as the convex hull of its acceptance set, which is precisely the arborescence polytope $\mathcal{Z}_{\text{tree}}$ defined in Section 7.2.1. Hence we have the constraint

$$\boldsymbol{z}_{\mathcal{A}} \in \mathcal{Z}_{\text{tree}}. \tag{7.52}$$

For the head automata, we denote by $\boldsymbol{z}^{\text{cs}} := (z^{\text{cs}}_{hms})_{hms}$ the vector of marginal probabilities for consecutive siblings, where z^{cs}_{hms} denotes a marginal probability for the event that (h, m)

[11] Smith and Eisner (2008) also proposed other variants with more factors, which we omit for brevity.

and (h, s) are consecutive siblings on the same side of h. The marginal polytope for each head automaton $(h, \rightarrow)$ is the set $\mathcal{Z}_{\text{head},h,\rightarrow}$ defined in Eq. 7.46 (and analogously for $(h, \leftarrow)$). The marginal polytope for the entire set of head automata is the polytope $\mathcal{Z}_{\text{head}}$ defined in Eq. 7.47. We thus have

$$(\boldsymbol{z}_{\mathcal{A}}, \boldsymbol{z}^{\text{cs}}) \in \mathcal{Z}_{\text{head}}. \tag{7.53}$$

Let us now turn to the pairwise factors. It is straightforward to write a contingency table and obtain the following local agreement constraints at each pairwise factor involving arcs a and b:

$$\begin{aligned} \mu_{ab}(1,1) &= z_{ab}, & \mu_{ab}(0,0) &= 1 - z_a - z_b + z_{ab} \\ \mu_{ab}(1,0) &= z_a - z_{ab}, & \mu_{ab}(0,1) &= z_b - z_{ab}. \end{aligned}$$

Noting that all these marginals are constrained to the unit interval, one can get rid of all variables μ_{ab} and write everything as

$$\begin{aligned} &z_a \in [0,1], \quad z_b \in [0,1], \quad z_{ab} \in [0,1], \\ &z_{ab} \leq z_a, \quad z_{ab} \leq z_b, \quad z_{ab} \geq z_a + z_b - 1, \end{aligned} \tag{7.54}$$

The local polytope $\text{LOCAL}(\mathcal{G}_{\text{tree}})$ is thus formed by the sets of constraints in Eqs. 7.52, 7.53 and 7.54.

Now let us write the variational problem that loopy BP optimizes. For simplicity, we will follow Martins et al. (2010f) and refer only to the configuration without the head automata (*i.e.*, just the tree factor and the pairwise soft factors). For deriving the Bethe entropy approximation, we need first to compute the factor entropies. Start by noting that the tree-factor entropy H_{tree} can be obtained in closed form by computing the marginals $\bar{\boldsymbol{z}}_{\mathcal{A}}$ and the partition function Z from the incoming messages, as shown in Algorithm 6. Both quantities can be computed using the matrix-tree theorem. Let

$$I_{a;b}(z_a, z_b, z_{ab}) = \sum_{y_a, y_b} \mu_{ab}(y_a, y_b) \log \frac{\mu_{ab}(y_a, y_b)}{\mu_a(y_a)\mu_b(y_b)} \tag{7.55}$$

be the mutual information associated with each pairwise factor. From Eq. 4.45, we can write the overall Bethe entropy approximation as:

$$\begin{aligned} H_{\text{Bethe}}(\boldsymbol{\mu}) &= \sum_{a \in \mathcal{A}} (1 - \deg(a)) H_a(\boldsymbol{z}_a) + \sum_{(a,b) \in \mathcal{P}} H_{ab}(\boldsymbol{z}_{ab}) + H_{\text{tree}}(\boldsymbol{z}_{\mathcal{A}}) \\ &= H_{\text{tree}}(\boldsymbol{z}_{\mathcal{A}}) - \sum_{(a,b) \in \mathcal{P}} I_{a;b}(z_a, z_b, z_{ab}). \end{aligned} \tag{7.56}$$

The approximate variational expression becomes

$$\begin{aligned} \log Z(\boldsymbol{\theta}, x) \approx \quad \max_{\boldsymbol{z}} \quad & \boldsymbol{\theta}^\top \mathbf{F}(x) \boldsymbol{z} + H_{\text{tree}}(\boldsymbol{z}_{\mathcal{A}}) - \sum_{(a,b) \in \mathcal{P}} I_{a;b}(z_a, z_b, z_{ab}) \\ \text{s.t.} \quad & z_{ab} \leq z_a, \quad z_{ab} \leq z_b, \\ & z_{ab} \geq z_a + z_b - 1, \quad \forall (a,b) \in \mathcal{P}, \\ & \boldsymbol{z}_{\mathcal{A}} \in \mathcal{Z}_{\text{tree}}, \end{aligned} \tag{7.57}$$

whose maximizer corresponds to the beliefs returned by Smith and Eisner's loopy BP algorithm (if it converges). The reader familiar with Bethe approximations for pairwise graphs

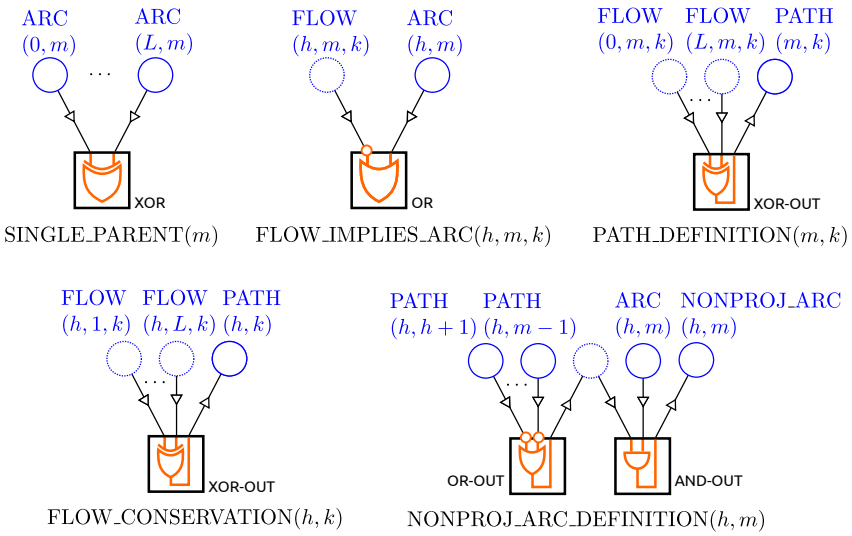


Figure 7.6: Details of the factor graph underlying the parser of (Martins et al., 2009b). Dashed circles represent auxiliary variables.

will note that Eq. 7.57 only differs from the pure pairwise case without hard constraints by replacing the sum of variable entropies, $\sum_{a \in \mathcal{A}} H(z_a)$ by the tree entropy $H_{\text{tree}}(z_{\mathcal{A}})$.

7.3.2 A Flow-Based Factor Graph

We now turn to our concise ILP formulation (Martins et al., 2009b), described in Section 7.2. As mentioned above, this formulation is exact but the corresponding problem is NP-hard, hence we suggest dropping the integer constraints to obtain an LP relaxation of the original problem.

This section sheds some light on this approximation. Namely, we will construct a factor graph $\mathcal{G}_{\text{flow}}$, and we will show that the LP relaxation corresponds to LP-MAP inference in that graph. In other words, we will show that it corresponds to an optimization of the form in Eq. 4.50, where the marginal polytope $\text{MARG}(\mathcal{G}_{\text{flow}})$ is replaced by the local surrogate $\text{LOCAL}(\mathcal{G}_{\text{flow}})$. The flow-based factor graph is represented in Figure 7.6, and we next describe it in detail.

For completeness, we consider all the parts illustrated in Figure 7.1, which were described in Sections 7.2.5–7.2.10 (we leave out the valency features, whose interpretation would be a little bit more involved). Let us start by describing the variables nodes of $\mathcal{G}_{\text{flow}}$. There are two kinds of variable nodes:

- those that correspond to binary variables that contribute score terms to the objective of the ILP;
- some additional *auxiliary variables* that are necessary to build some of the constraints.

The ones of the first kind are:

- Arc variables $z := (z_a)_{a \in \mathcal{A}}$, denoting potential arcs in the tree;

- Consecutive sibling variables $\boldsymbol{z}^{\text{cs}} := (z^{\text{cs}}_{hms})_{h,m,s}$, each denoting a pair of consecutive siblings (h, m) and (h, s) in the tree;
- Directed path variables $\boldsymbol{z}^{\text{path}} := (z^{\text{path}}_{ij})^{L}_{i,j=0}$, where each z^{path}_{ij} represents the event that word i is an ancestor of word j in the dependency tree;
- Non-projective arc variables $\boldsymbol{z}^{\text{np}} := (z^{\text{np}}_{a})_{a\in\mathcal{A}}$, each z^{np}_{a} denoting that a is a non-projective arc in the dependency tree.

The auxiliary variable nodes are:

- The flow variables $\boldsymbol{\phi} := (\phi^{k}_{a})_{a\in\mathcal{A},k=1,\dots,L}$ that denote multi-commodity flows;
- The auxiliary variables $\boldsymbol{\omega} := (\omega_{ham})_{h,a,m}$ that denote states in the head automata.

For clarity, we represent the auxiliary variables as dashed circles in Figure 7.6. (In general, any lifted formulation involves dashed circles.)

Let us now turn to the factors of $\mathcal{G}_{\text{flow}}$. Again, there are two kinds:

- *soft factors*, all of them pairwise, allowing to deal with pairwise arc features (for siblings, grandparents and head-bigrams). These factors connect pairs of arc variables, precisely as was done in the tree-based factor graph (Section 7.3.1).
- *hard constraint factors*, which represent the linear constraints of the ILP.

It remains to describe the hard constraint factors. A few of them are unary and serve only to seed some of the path and flow variables, imposing the constraints

$$z^{\text{path}}_{0k} = z^{\text{path}}_{kk} = 1, \quad \forall k, \tag{7.58}$$

$$\phi^{h}_{(h,m)} = 0, \quad \forall h, m; \tag{7.59}$$

i.e., that *any word descends from the root and from itself*, and that *arcs leaving a word carry no flow to that word*. Then, there is a set of factors which serve to impose the multi-commodity flow constraints, and which replace the tree factor in Figure 7.5:

- $O(L)$ XOR factors, each connecting all arc variables of the form $\{(h, m)\}_{h=0,\dots,L}$ for a fixed m. These ensure that *each word has exactly one parent*. Each factor yields a local agreement constraint (see the marginal polytope expression in Eq. 5.25):

$$\sum_{h=0}^{L} z_{(h,m)} = 1, \quad m \in \{1, \dots, L\}; \tag{7.60}$$

  these are equivalent to the single-parent constraints in Eq. 7.3. The constraint that the root has no parent (Eq. 7.4) can be trivially imposed by removing all arc candidates of the form $(h, 0)$, which are structurally impossible.

- $O(L^3)$ IMPLY factors, each expressing that *if an arc carries flow, then that arc must be active*. Such factors are pairwise OR factors with the first input negated, hence, the local agreement constraints are:

$$\phi^{k}_{a} \leq z_{a}, \quad a \in \mathcal{A}, k \in \{1, \dots, L\}. \tag{7.61}$$

These correspond to the multi-commodity flow constraints in Eq. 7.14.

- $O(L^2)$ XOR-with-output factors, which impose the constraint that *each path variable* z^{path}_{mk} *is active if and only if exactly one incoming arc in* $\{(h,m)\}_{h=0,\ldots,L}$ *carries flow to* k. Such factors are XOR factors with the last input negated, and hence their local constraints are:

$$z^{\text{path}}_{mk} = \sum_{h=0}^{n} \phi^{k}_{(h,m)}, \quad m,k \in \{1,\ldots,L\}. \tag{7.62}$$

These correspond to the constraints in Eq. 7.49 that are the very definition of the path variables.

- $O(L^2)$ XOR-with-output factors to impose the constraint that *words do not consume other words' commodities*; *i.e.*, if $h \neq k$ and $k \neq 0$, then there is a path from h to k if and only if exactly one outgoing arc in $\{(h,m)\}_{m=1,\ldots,L}$ carries flow to k. The resulting marginal polytopes are defined by the constraints:

$$z^{\text{path}}_{hk} = \sum_{m=1}^{n} \phi^{k}_{(h,m)}, \; h,k \in \{0,\ldots,L\}, \; k \notin \{0,h\}. \tag{7.63}$$

It is easy to see that these correspond to the multi-commodity flow constraints in Eq. 7.13 for the case $j \neq k$, once we make the substitution in Eq. 7.62.

These factors are equivalent to the tree factor, in the sense that composing all their marginal polytopes yields the marginal polytope of the tree factor, *i.e.*, the arborescence polytope. In addition, we have another set of factors which replace the head automata factors in Figure 7.5:

- A XOR factor requiring that *there is exactly one word which is the last modifier of* h *up to* i. This imposes the constraint in Eq. 7.41:

$$\sum_{a=0}^{i} \omega_{hai} = 1, \quad i \in \{1,\ldots,L\}, \tag{7.64}$$

- A XOR-with-output factor imposing that, for each word i and each state a (the last modifier up to i), *either that word preserves the previous state or it modifies* h (cf. Eq. 7.42):

$$\omega_{ha(i+1)} + z^{\text{cs}}_{ha(i+1)} = \omega_{hai}, \quad i \in \{1,\ldots,L\}, a \in \{1,\ldots,i\} \tag{7.65}$$

- A XOR-with-output factor imposing that *each potential arc* $(h, i+1)$ *is active if and only if it is the next sibling of some other arc, which must be unique* (or a first child of h, in which case $z^{\text{cs}}_{hh(i+1)} = 1$ by definition). The corresponding marginal polytope emulates Eq. 7.43:

$$\sum_{a=0}^{i} z^{\text{cs}}_{ha(i+1)} = z_{h(i+1)}, \quad i \in \{1,\ldots,L\} \tag{7.66}$$

We must also collapse the variables ω_{hii} and z_{hi}, in accordance to Eq. 7.40. Finally, there is a last set of factors that serve to accommodate the non-projective arc features:

- For each arc $a \in \mathcal{A}$ of the form $a = (h, m)$, we have one OR-with-output factor connected to all the path variables z_{hk}^{path} for k between h and m, each negated, whose output is an extra variable (call it γ_a) that indicates that *there is some word k in the span of a which does not descend from h*. Suppose without loss of generality that $h < m$. From Eq. 5.39, we have that the corresponding marginal polytope is defined by the following set of constraints:

$$\begin{aligned} \gamma_a &\geq 1 - z_{hk}^{\text{path}}, \quad \text{for every } k \in \{h+1, \ldots, m-1\}, \\ \gamma_a &\leq - \sum_{k \in \{h+1,\ldots,m-1\}} z_{hk}^{\text{path}} + |m-h| - 1. \end{aligned} \tag{7.67}$$

- For each arc $a \in \mathcal{A}$ of the form $a = (h, m)$, one AND-with-output factor connected to z_a and γ_a, whose output is z_a^{np}. This signals that *the tree contains the arc a and that this arc is non-projective*. The corresponding constraints are:

$$\begin{aligned} z_a^{\text{np}} &\leq z_a, \\ z_a^{\text{np}} &\leq \gamma_a, \\ z_a^{\text{np}} &\geq z_a + \gamma_a - 1. \end{aligned} \tag{7.68}$$

Taken together, the constraints in Eqs. 7.67–7.68 are equivalent to the non-projectivity constraints in Eq. 7.51.

Let us summarize what we have built. We defined a factor graph $\mathcal{G}_{\text{flow}}$, represented in Figure 7.6; MAP inference in such graph is equivalent to the ILP formulation that we have introduced in Section 7.2. Moreover, LP-MAP inference, which replaces the marginal polytope of this graph by the local polytope $\text{LOCAL}(\mathcal{G}_{\text{flow}})$, defined by the constraints in Eqs. 7.54–7.68, is precisely the LP relaxation of that ILP formulation. Moreover, if we resort only to the parts considered in Section 7.3.1—namely, the ones for pairwise arcs (grandparents, arbitrary siblings, and head bigrams) as well as the consecutive siblings—then we also have

$$\text{LOCAL}(\mathcal{G}_{\text{flow}}) = \text{LOCAL}(\mathcal{G}_{\text{tree}}), \tag{7.69}$$

since, as we have seen, the multi-commodity flow model is an exact representation of the arborescence polytope, and the same holds for the head automaton polytope. Therefore, although the approaches of Smith and Eisner (2008) and Martins et al. (2009b) look very different, in reality both are variational approximations emanating from Proposition 5.3, respectively for marginal and MAP inference. However, they operate on distinct factor graphs, respectively the ones in Figure 7.5 and Figure 7.6.[12]

[12] Given what was just exposed, it seems appealing to try max-product loopy BP on the factor graph of Figure 7.5, or sum-product loopy BP on the one in Figure 7.6. Both attempts present serious challenges: the former requires computing messages sent by the tree factor, which requires $O(L^2)$ calls to the Chu-Liu-Edmonds algorithm and hence $O(L^5)$ time. No obvious strategy seems to exist for simultaneous computation of all messages, unlike in the sum-product case. The latter is even more challenging, as standard sum-product loopy BP has serious issues in the factor graph of Figure 7.6; we construct in Martins et al. (2010f, extended version, Appendix B) a simple example with a very poor Bethe approximation. This might be fixed by using other variants of sum-product BP, e.g., ones in which the entropy approximation is concave.

7.4 Dependency Parsing with AD^3

In this section, we describe how the AD^3 algorithm presented in Chapter 6 can be applied to our rich feature models for dependency parsing. Before doing so, we briefly review the related work on dual decomposition.

7.4.1 Related Work: Dual Decomposition in NLP

A typical source of intractability in NLP problems comes from the combinatorial explosion inherent in the composition of two or more tractable models (Bar-Hillel et al., 1964; Tromble and Eisner, 2006). In a recent paper, Rush et al. (2010) proposed a dual decomposition framework as a way of combining models which alone permit efficient decoding, but whose combination is intractable. These are problems in which the global score decomposes as $f(y) = f_1(z_1) + f_2(z_2)$, where z_1 and z_2 are two overlapping "views" of the output, so that the problem of finding the output $y \in \mathcal{Y}$ with the largest score $f(y)$ can be written as

$$\begin{array}{ll} \text{maximize} & f_1(z_1) + f_2(z_2) \\ \text{w.r.t.} & z_1 \in \mathcal{Y}_1, z_2 \in \mathcal{Y}_2 \\ \text{s.t.} & z_1 \sim z_2. \end{array} \tag{7.70}$$

Above, the notation $z_1 \sim z_2$ means that z_1 and z_2 "agree on their overlaps," and an isomorphism $\mathcal{Y} \simeq \{\langle z_1, z_2 \rangle \in \mathcal{Y}_1 \times \mathcal{Y}_2 \mid z_1 \sim z_2\}$ is assumed. One way of addressing this problem is to introduce Lagrange multipliers for the agreement constraints and to tackle the dual with the projected subgradient algorithm, a technique known as *Lagrangian relaxation*. This results in a relaxation of the original problem which is equivalent to the one described in our background chapters (Section 4.6.2). This technique has proven quite effective in parsing (Koo et al., 2010; Auli and Lopez, 2011) as well as machine translation (Rush and Collins, 2011; Chang and Collins, 2011). The application to dependency parsing by Koo et al. (2010) has already been mentioned in Section 7.3.2, when we described inference in the tree-based factor graph.

We will next formalize these notions and proceed to compositions of an *arbitrary* number of models. Of special interest is the unexplored setting where this number is very large and each component very simple. We show here that the success of dual decomposition with the projected subgradient algorithm is strongly tied to the ability of finding a "good" decomposition, *i.e.*, one involving few overlapping components (or *slaves*). With many components, the algorithm exhibits extremely slow convergence (cf. Figure 7.8). Unfortunately, a lightweight decomposition is not always at hand, either because the problem does not factor in a natural way, or because one would like to incorporate features that cannot be easily absorbed in few tractable components. Examples include some of the features that we have discussed in Section 7.2, as well as features generated by statements in first-order logic, features that violate Markov assumptions, or history features such as the ones employed in transition-based parsers.

7.4.2 Dual Decomposition With Many Overlapping Components

We will see that the AD^3 algorithm, introduced in Chapter 6, is a good fit for the kind of problems mentioned above. AD^3 retains the modularity of the subgradient-based method, but it speeds up consensus by regularizing each slave subproblem towards the averaged votes obtained in the previous round (cf. Algorithm 8). While this yields more involved subproblems (with a quadratic term), we have shown in Chapter 6 how exact solutions can still be efficiently computed for all kinds of first-order logic, which is all we need for the flow-based factor graph described in the previous section. As a result, we obtain parsers that can handle very rich features, do not require specifying a decomposition, and can potentially be parallelized.

Let us first establish the basic setup, where we characterize the kind of NLP problems for which AD^3 is potentially useful. These are problems characterized by many overlapping components or parts; such problems abound in the scope of text analysis at sentence level (parsing, compositional semantics, *etc.*) and at document or corpus level (summarization, entity resolution, *etc*). The ingredients are the following:

Basic parts. We let $\mathcal{R}$ be a set of *basic parts*, such that each element $y \in \mathcal{Y}$ can be identified with a subset of $\mathcal{R}$. The exact meaning of a "basic part" is problem dependent. For example, in dependency parsing, $\mathcal{R}$ can be the set of all possible dependency arcs (see Figure 7.1); in phrase-based parsing, it can be the set of possible spans; in sequence labeling, it can be the set of possible labels at each position. Our only assumption is that we can "read out" y from the basic parts it contains. For convenience, we represent y as a binary vector, $\boldsymbol{y} = (y(r))_{r\in\mathcal{R}}$, where $y(r) = 1$ if part r belongs to y, and 0 otherwise.

Decomposition. We generalize the decomposition in Eq. 7.70 by considering sets $\mathcal{Y}_1, \ldots, \mathcal{Y}_S$ for $S \geq 2$. Each $\mathcal{Y}_s$ is associated with its own set of parts $\mathcal{R}_s$, in the same sense as above; we represent the elements of $\mathcal{Y}_s$ as binary vectors $\boldsymbol{z}_s = (z_s(r))_{r\in\mathcal{R}_s}$. Examples are vectors indicating a tree structure, a sequence, or an assignment of variables to a factor, in which case it may happen that only some binary vectors are legal. Some parts in $\mathcal{R}_s$ are basic, while others are not. We denote by $\bar{\mathcal{R}}_s = \mathcal{R}_s \cap \mathcal{R}$ the subset of the ones that are. In addition, we assume that:

- $\mathcal{R}_1, \ldots, \mathcal{R}_S$ jointly cover $\mathcal{R}$, *i.e.*, $\mathcal{R} \subseteq \bigcup_{s=1}^{S} \mathcal{R}_s$;
- Only basic parts may overlap, *i.e.*, $\mathcal{R}_s \cap \mathcal{R}_t \subseteq \mathcal{R}, \forall s, t \in \{1, \ldots, S\}$;
- Each $\boldsymbol{z}_s \in \mathcal{Y}_s$ is completely defined by its entries indexed by elements of $\bar{\mathcal{R}}_s$, from which we can guess the ones in $\mathcal{R}_s \setminus \bar{\mathcal{R}}_s$. This implies that each $y \in \mathcal{Y}$ has a unique decomposition $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_S)$.

Fig. 7.1 shows several parts used in dependency parsing models; in phrase-based parsing, these could be spans and production rules anchored in the surface string; in sequence labeling, they can be unigram, bigram, and trigram labels.[13]

[13] There is a lot of flexibility about how to decompose the model into S components: each set $\mathcal{R}_s$ can correspond to a single factor in a factor graph (Smith and Eisner, 2008), or to a entire subgraph enclosing several factors (Koo et al., 2010), or even to a formula in Markov logic (Richardson and Domingos, 2006). In these examples, the basic parts may correspond to individual variable-value pairs.

Global consistency. We want to be able to read out $y \in \mathcal{Y}$ by "gluing" together the components $(z_1, \ldots, z_S)$. This is only meaningful if they are "globally consistent," a notion which we make precise. Two components $z_s \in \mathcal{Y}_s$ and $z_t \in \mathcal{Y}_t$ are said to be *consistent* (denoted $z_s \sim z_t$) if they agree on their overlaps, *i.e.*, if $z_s(r) = z_t(r), \forall r \in \mathcal{R}_s \cap \mathcal{R}_t$. A complete assignment $(z_1, \ldots, z_S)$ is *globally consistent* if all pairs of components are consistent. This is equivalent to the existence of a witness vector $(u(r))_{r \in \mathcal{R}}$ such that $z_s(r) = u(r), \forall s, r \in \bar{\mathcal{R}}_s$.

With this setup, assuming that the score function decomposes as $f(z) = \sum_{s=1}^{S} f_s(z_s)$, the decoding problem (which extends Eq. 7.70 for $S \geq 2$) becomes:

$$\begin{array}{lll} P: & \text{maximize} & \sum_{s=1}^{S} f_s(z_s) \\ & \text{w.r.t.} & z_s \in \mathcal{Y}_s, \quad \forall s \\ & & (u(r))_{r \in \mathcal{R}} \in \mathbb{R}^{|\mathcal{R}|}, \\ & \text{s.t.} & z_s(r) = u(r), \quad \forall s, r \in \bar{\mathcal{R}}_s. \end{array} \tag{7.71}$$

We call the equality constraints expressed in the last line the "agreement constraints." It is these constraints that complicate the problem, which would otherwise be exactly separable into S subproblems. Note that each set $\mathcal{Y}_s$ in Eq. 7.71 is combinatorial, hence non-convex. Let $\mathcal{Z}_s$ denote the convex hull of $\mathcal{Y}_s$. The relaxed problem is

$$\begin{array}{lll} P': & \text{maximize} & \sum_{s=1}^{S} f_s(z_s) \\ & \text{w.r.t.} & z_s \in \mathcal{Z}_s, \quad \forall s \\ & & \langle u(r) \rangle_{r \in \mathcal{R}} \in \mathbb{R}^{|\mathcal{R}|}, \\ & \text{s.t.} & z_s(r) = u(r), \quad \forall s, r \in \bar{\mathcal{R}}_s. \end{array} \tag{7.72}$$

AD^3 updates. Regarding each component as a factor in a factor graph, we can apply the AD^3 algorithm in a straighforward manner. The updates become the following (cf. Algorithm 8):[14]

- z-updates, for each $s = 1, \ldots, S$:

$$z_s^{t+1} = \arg \max_{z_s \in \mathcal{Z}_s(x)} \left(f_s(z_s) + \sum_{r \in \bar{\mathcal{R}}_s} \lambda_s(r) z_s(r) - \frac{\eta}{2} \sum_{r \in \bar{\mathcal{R}}_s} (z_s(r) - u^t(r))^2 \right), \tag{7.73}$$

- u-updates:

$$u^{t+1}(r) = \frac{1}{|\{s : r \in \bar{\mathcal{R}}_s\}|} \sum_{s: r \in \bar{\mathcal{R}}_s} z_s^{t+1}(r), \tag{7.74}$$

- λ-updates:

$$\lambda_s^{t+1}(r) = \lambda_s^t(r) - \tau\eta(z_s^{t+1}(r) - u^{t+1}(r)). \tag{7.75}$$

We end this section by listing in Table 7.4 all the factors in the flow-based factor graph, including their interpretation as first order logic statements. Each of these factors is a components in the AD^3 algorithm. We also show in Table 7.4 the asymptotic runtimes for solving the corresponding subproblems.

[14] We set $\tau = 1.5$ and follow the procedure described in Section 6.3.2 for adjusting the penalty constant. We initialize $\eta = 0.03$ and then increase/decrease η by a factor of 2 whenever the primal residual becomes > 10 times larger/smaller than the dual residual.

Arc-factored	bilexical, unilexical, POS	$w_h \wedge w_m$ and POS back-offs
	head and modifier contexts	$p_{h-1} \wedge p_h \wedge p_{h+1} \wedge p_{m-1} \wedge p_m \wedge p_{m+1}$
	in-between	$p_h \wedge p_i \wedge p_m$, for some i between h and m
	arc direction and length	$[\![h < m]\!]$, $\lvert h - m \rvert$ (binned)
	morphological	all combinations of morphological features of heads and modifiers (if available)
Grandparents	bilexical, unilexical, POS	$w_g \wedge p_h \wedge w_m$, $w_g \wedge w_h \wedge p_m$, $p_g \wedge w_h \wedge w_m$, *etc.*
	arc directions and lengths	$[\![h < m]\!] \wedge [\![g < h]\!]$, $\lvert g - h \rvert + \lvert h - m \rvert$, *etc.*
Siblings, all-siblings	bilexical, unilexical, POS	$w_h \wedge p_m \wedge w_s$, $w_h \wedge w_m \wedge p_s$, $w_h \wedge w_m \wedge w_s$, *etc.*
	arc directions and lengths	$[\![h < m]\!] \wedge [\![m < s]\!]$, $\lvert h - m \rvert + \lvert h - s \rvert$, *etc.*
Head bigram	bilexical, unilexical, POS	$w_h \wedge w_{h'} \wedge p_m \wedge p_{m-1}$, etc.
	arc directions and lengths	$[\![h < m]\!] \wedge [\![h' < m + 1]\!]$, $\lvert h' - m + 1 \rvert + \lvert h - m \rvert$, *etc.*
Directed path	bilexical, unilexical, POS	$w_a \wedge w_d$ and POS back-offs, for noun/verb pairs only
	ancestor	w_a, p_a (this counts the number of descendants)
	descendant	w_d, p_d (this computes the depth of a word/POS in the tree)
Nonprojective arc	bilexical, unilexical, POS	$w_h \wedge w_m$ and POS backoffs
	bias	1 (this counts the number of non-projective arcs)
	in-between	$p_h \wedge p_i \wedge p_m$, for some i between h and m
	arc direction and length	$[\![h < m]\!]$, $\lvert h - m \rvert$ (binned)

Table 7.1: Features used in our parser. Each w designates a word appended with a POS tag, and each p denotes a POS tag only. We also incorporate backed off versions of these features (reducing the context and replacing lexical items by POS tags). We also incorporate versions of some of these features with lemmas instead of words, and coarse POS tags (when that information exists in the dataset). The arc lengths are binned (lengths $1, \ldots, 5, \geq 5, \geq 10$) and converted to binary features. Some variants of these features were first proposed by McDonald et al. (2005a, 2006); Carreras (2007).

7.5 Experiments

In this section, we evaluate empirically several configurations of turbo parsers.[15]

Dataset Description. We used 14 datasets with non-projective dependencies from the CoNLL-2006 and CoNLL-2008 shared tasks (Buchholz and Marsi, 2006; Surdeanu et al., 2008). These datasets contain tokenized sentences with pre-computed information regarding the words, lemmas, fine and coarse part-of-speech (POS) tags predicted by an external tagger, and morphological information (about gender, number, case, *etc.*). We point to the references above for additional language-specific details.

In addition, we also used a projective English dataset derived from the Penn Treebank by applying the standard head rules of Yamada and Matsumoto (2003). As usual, we train on sections §02–21, use §22 as validation data, and test on §23. We ran SVMTool (Giménez and Marquez, 2004) to obtain automatic POS tags for §22–23.

Training and Test Procedures. We trained by running 10 iterations of the cost-augmented MIRA algorithm (Crammer et al., 2006) with LP-relaxed decoding (Martins et al., 2009c); we describe this strategy in further detail in Chapter 8. In our experiments, we did not force the parser to output projective trees or unique roots for any of the datasets; everything is learned from the data. Following common practice (Charniak and Johnson, 2005; Carreras et al., 2008), we employed a coarse-to-fine procedure to prune away unlikely candidate arcs,

[15] We focus on parsers based on LP-MAP inference; later in Chapter 8 we will compare this sort of parsers with the ones based on marginal inference, namely the loopy BP parser of Smith and Eisner (2008). Additional experiments comparing exact decoding with relaxed decoding appear in Martins et al. (2009b).

as described by Koo and Collins (2010): the pruner is a probabilistic arc-factored model which we use to eliminate arcs with posterior probability inferior to 10^{-4}. In addition, we limit the number of candidate heads to each word to 10.

To ensure valid parse trees at test time, we rounded fractional solutions as described in Martins et al. (2009b): first, solve the LP relaxation; if the solution is integer, we are done; otherwise, we consider the restriction of the z-variables which are indexed by arcs, $\tilde{z} := (z_a)_{a \in A}$, and project it onto the feasible set $\mathcal{Y}(x)$. That projection can be computed in a straightforward way by finding a maximal arborescence in the directed graph whose weights are defined by $\tilde{z}$, which can done with the Chu-Liu-Edmonds algorithm. In practice, we have found that solutions were integral most of the time.

The parts used in our full model are the ones depicted in Figure 7.1. For all the datasets, we used a common set of features, which we describe in Table 7.1. Note that a subgradient-based method could handle some of those parts efficiently (*arcs*, *consecutive siblings*, *grandparents*, and *head bigrams*) by composing arc-factored models, head automata, and a sequence labeler. However, no lightweight decomposition seems possible for incorporating parts for *all siblings*, *directed paths*, and *non-projective arcs*. Table 7.4 shows the first-order logical formulae that encode the constraints in our model. Each formula gives rise to a subproblem which is efficiently solvable (see Chapter 6).

By ablating some of rows of Table 7.4 we recover known methods:

- Resorting to the *tree* and *consecutive sibling* formulae gives one of the models in Koo et al. (2010), with the *same* linear relaxation (see Section 7.3.2 for a proof of this fact);
- Resorting to *tree*, *all siblings*, *grandparent*, and *non-projective arcs*, recovers the multi-commodity flow configuration originally proposed by Martins et al. (2009b); the relaxation is also the same.[16]

The experimental results are shown in Table 7.2. For comparison, we include the best published results for each dataset (to the best of our knowledge), among transition-based parsers (Nivre et al., 2006; Huang and Sagae, 2010), graph-based parsers (McDonald et al., 2006; Koo and Collins, 2010), hybrid methods (Nivre and McDonald, 2008; Martins et al., 2008a), and turbo parsers (Martins et al., 2010f; Koo et al., 2010). Our full model achieved the best reported scores for 7 datasets. The last two columns show a consistent improvement (with the exceptions of Chinese and Arabic) when using the full set of features over a second order model with grandparent and consecutive siblings, which is our reproduction of the model of Koo et al. (2010).[17]

Feature ablation and error analysis. We conducted a simple ablation study by training several models on the English PTB with different sets of features. Table 7.3 shows the results. As expected, performance keeps increasing as we use models with greater expressive power. Figure 7.7 shows examples of parses that were correctly predicted by the full model, but not by the G+CS model.

[16] As pointed out in Section 7.2, although Martins et al. (2009b) also incorporated consecutive siblings in one of their configurations, our constraints are tighter than theirs.

[17] Note however that the actual results of Koo et al. (2010) are higher than our reproduction, as can be seen in the second column. The differences are due to the features that were used and on the way the models were trained. The cause is not search error: exact decoding with an ILP solver (CPLEX) revealed no significant difference with respect to our G+CS column.

	Best known UAS		G+CS	Full
Arabic	80.18	Martins et al. (2008a)	**81.12**	81.10 (-0.02)
Bulgar.	92.88	Martins et al. (2010f)	93.04	**93.50** (+0.46)
Chinese	**91.89**	Martins et al. (2010f)	91.05	90.62 (-0.43)
Czech	88.78	Martins et al. (2010f)	88.80	**89.46** (+0.66)
English	92.57	Koo et al. (2010)	92.45	**92.68** (+0.23)
Danish	91.78	Koo et al. (2010)	91.70	**91.86** (+0.16)
Dutch	**85.81**	Koo et al. (2010)	84.77	85.53 (+0.76)
German	91.49	Martins et al. (2010f)	91.29	**91.89** (+0.60)
Japane.	93.42	Martins et al. (2010f)	93.62	**93.72** (+0.10)
Portug.	**93.03**	Koo et al. (2010)	92.05	92.29 (+0.24)
Slovene	86.21	Koo et al. (2010)	86.09	**86.95** (+0.86)
Spanish	**87.04**	Martins et al. (2010f)	85.99	86.74 (+0.75)
Swedish	**91.36**	Koo et al. (2010)	89.94	90.16 (+0.22)
Turkish	**77.55**	Koo et al. (2010)	76.24	76.64 (+0.40)
PTB §23	**93.04**	Koo and Collins (2010)	92.19	92.53 (+0.34)

Table 7.2: Unlabeled attachment scores, excluding punctuation. In columns 3–4, "Full" is our full model, and "G+CS" is our reproduction of the model of Koo et al. (2010), *i.e.*, the same as "Full" but with all features ablated excepted for grandparents and consecutive siblings.

	AF	+G+CS	+AS	+NP	Full
PTB §22	91.02	92.13	92.32	92.36	92.41
PTB §23	91.36	92.19	92.41	92.50	92.53

Table 7.3: Feature ablation experiments. AF is an arc-factored model; +G+CS adds grandparent and consecutive siblings; +AS adds all-siblings; +NP adds non-projective arcs; Full adds the bigram and directed paths.

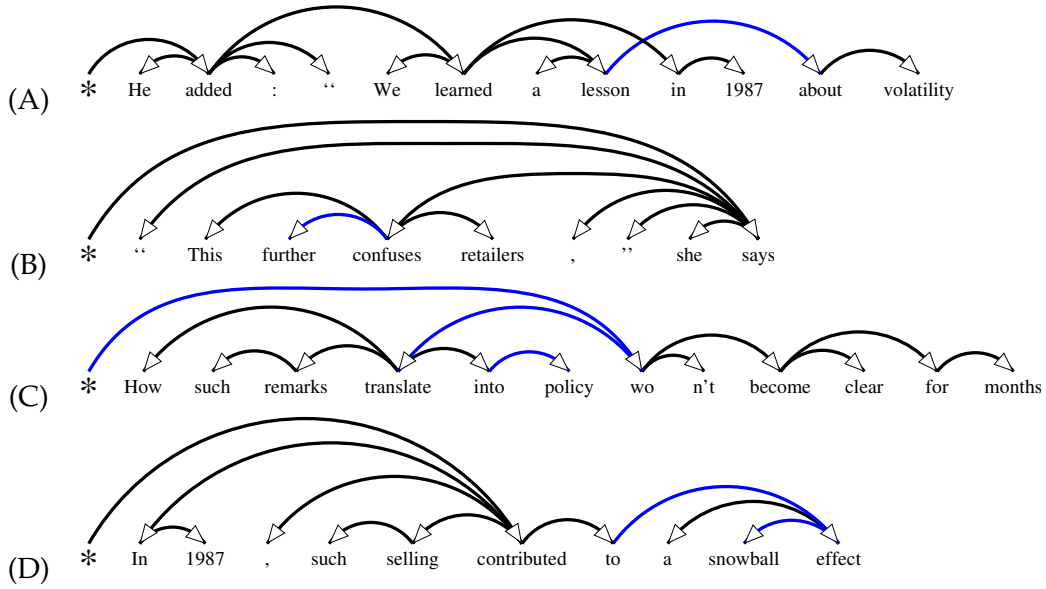


Figure 7.7: Sentences of the English non-projective dataset (CoNLL 2008) that were correctly parsed by the full model, but not by the G+CS model. Arcs shown in blue are those that were missed by the G+CS model. See text for an explanation.

In (A), the G+CS model has predicted *1987* as the parent of *about*. The full model got it right, arguably due to the nonprojectivity features that find the nonprojective arc *lesson→about* to be likely.

In (B), the G+CS model attached *further* to *retailers*. The word *further* forms an adverbial phrase which enjoys considerable freedom about where it can be placed in a sentence. Hence, features that look at *all siblings* attached to a head word (rather than just consecutive ones) may help parsing sentences without a rigid word ordering.

(C) is an example where *path* features seem to help. The G+CS model predicted 3 arcs incorrectly, because it assumed that *policy won't become clear for months* was a phrase (hence it predicted *→*translate→wo(n't)→policy*). The full model may have found unlikely the long path that would descend from *translate*, and prefered a more horizontal parse.

Example (D) seems simple to parse; the word *snowball*, however, was incorrectly tagged as a verb by the part-of-speech tagger and confused the G+CS model, which predicted *to→snowball→effect*. The full model got it right, arguably because of the head bigram features, which give a low score to configurations in which two consecutive words (in this case *a* and *snowball*) have crossing dependency attachments in opposite sides. This shows that a parser with many features may gain robustness to errors in the pipeline.

Convergence speed and optimality. Figure 7.8 compares the performance of AD^3 and the projected subgradient algorithms in the validation section of the PTB.[18] For the second order model, the subgradient method has more slaves than in Koo et al. (2010): it has a slave imposing the tree constraint (whose subproblems consists on finding a minimum spanning tree) and several for the all-sibling parts, yielding an average number of 310.5 and a maxi-

[18]The learning rate in the subgradient method was set as $\eta_t = \eta_0/(1 + N_{\text{incr}}(t))$, as in Koo et al. (2010), where $N_{\text{incr}}(t)$ is the number of dual increases up to the tth iteration, and η_0 is chosen to maximize dual decrease after 20 iterations (on a per sentence basis). Those preliminary iterations are not plotted in Figure 7.8.

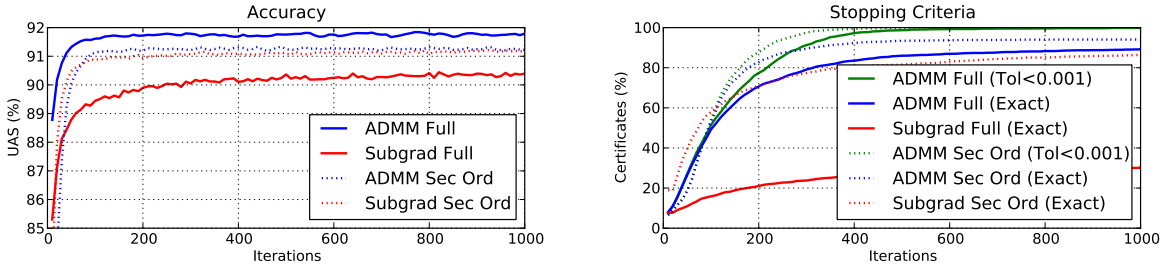


Figure 7.8: UAS including punctuation (left) and fraction of optimality certificates (right) accross iterations of the subgradient and AD^3 algorithms, in PTB §22. "Full" is our full model; "Sec Ord" is a second-order model with grandparents and all siblings, for which the subgradient method uses a coarser decomposition with the tree-based factor graph. Since subgradient and AD^3 are solving the same problems, the solid lines (as the dashed ones) would meet in the limit, however subgradient converges very slowly for the full model. The right plot shows optimality certificates for both methods, indicating that an exact solution of P has been found; for AD^3 we also plot the fraction of instances that converged to an accurate solution of P' (primal and dual residuals $< 10^{-3}$) and hence can be stopped.

mum of 4310 slaves. These numbers are still manageable, and we observe that a "good" UAS is achieved relatively quickly. The AD^3 method has many more slaves due to the multicommodity flow constraints (average 1870.8, maximum 65446), yet it attains optimality sooner, as can be observed in the right plot. For the full model, the subgradient-based method becomes extremely slow, and the UAS score severely degrades (after 1000 iterations it is 2% less than the one obtained with AD^3, with very few instances having been solved to optimality). The reason is the number of slaves: in this configuration and dataset the average number of slaves per instance is 3327.4, and the largest number is 113207. On the contrary, AD^3 keeps a robust performance, with a large fraction of optimality certificates in early iterations.

Runtime and caching strategies. Despite its suitability to problems with many overlapping components, our parser is still 1.6 times slower than Koo et al. (2010) (0.34 against 0.21 sec./sent. in PTB §23), and is far below the speed of transition-based parsers (*e.g.*, Huang and Sagae (2010) take 0.04 sec./sent. on the same data, although accuracy is lower, 92.1%). Our implementation, however, is not fully optimized. We next describe how considerable speed-ups are achieved by caching the subproblems, following a strategy similar to Koo et al. (2010).

Figure 7.9 illustrates the point. After a few iterations, many variables $u(r)$ see a consensus being achieved (*i.e.*, $u^t(r) = z_s^{t+1}(r), \forall s$) and enter an idle state: they are left unchanged by the u-update in Eq. 7.74, and so do the Lagrange variables $\lambda_s^{t+1}(r)$ (Eq. 7.75). If by iteration t all variables in a subproblem s are idle, then $z_s^{t+1}(r) = z_s^t(r)$, hence the subproblem does not need to be resolved.[19] Fig. 7.9 shows that many variables and subproblems are left untouched after the first few rounds.

Finally, Fig. 7.10 compares the runtimes of our implementation of AD^3 with those achieved

[19]Even if not all variables are idle in s, caching may still be useful: note that the z-updates in Eq. 7.73 tend to be sparse for our subproblems (these are Euclidean projections onto polytopes with 0/1 vertices, which tend to hit corners). Another trick that may accelerate the algorithm is warm-starting: since many subproblems involve a sort operation, storing the sorted indexes may speedup the next round.

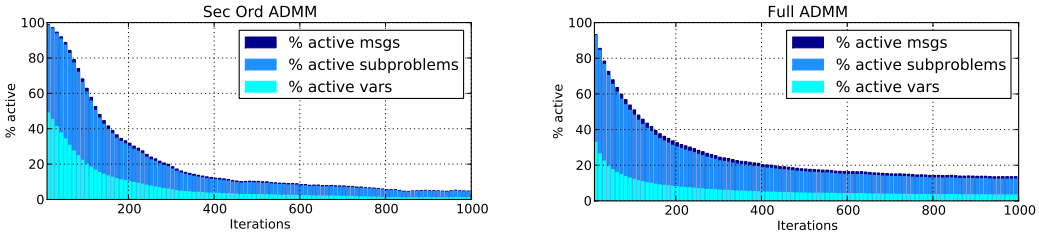


Figure 7.9: Fraction of active variables, subproblems and messages along AD^3 iterations (second order model and full model). The number of active messages denotes the total number of variables (active or not) that participate in an active factor.

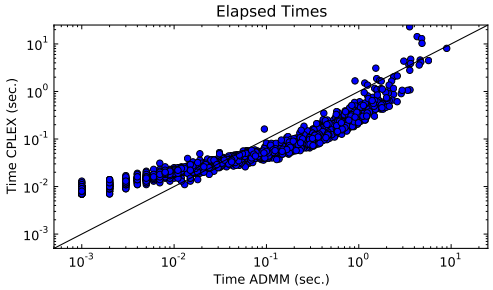


Figure 7.10: Runtimes of AD^3 and CPLEX on PTB §22 (each point is a sentence). Average runtimes are 0.362 (AD^3) and 0.565 sec./sent. (CPLEX).

by a state-of-the-art LP solver, CPLEX, in its best performing configuration: the simplex algorithm applied to the dual LP. We observe that AD^3 is faster in some regimes but slower in others. For short sentences ($<$ 15 words), AD^3 tends to be faster. For longer sentences, CPLEX is quite effective as it uses good heuristics for the pivot steps in the simplex algorithm; however, we observed that it sometimes gets trapped on large problems. Note also that AD^3 is not fully optimized, and that it is much more amenable to parallelization than the simplex algorithm, since it is composed of many independent slaves. This suggests potentially significant speed-ups in multi-core environments.

7.6 Conclusions and Future Work

In this chapter, we have introduced turbo parsers, feature-rich dependency parsers that run approximate inference in a loopy factor graph, ignoring the global effects caused by the loops. We have publicly released our turbo parsers as an open-source project.[20]

We started by deriving a new polynomial-size ILP formulation for dependency parsing based on multi-commodity flows, contrasting with a previous formulation that requires exponentially many constraints (Riedel and Clarke, 2006). Then, we constructed a factor graph whose LP-MAP inference problem is equivalent to the linear relaxation of the previous ILP; hence, the resulting parser is a turbo parser. For other turbo parsers (Smith and Eisner, 2008; Koo et al., 2010), we characterize the underlying factor graphs and the optimization problems that are addressed by the corresponding inference algorithms.

We then applied the AD^3 algorithm, introduced in Chapter 6, to solve our linear relax-

[20]See http://www.ark.cs.cmu.edu/TurboParser/.

ation. We devised a caching technique that provides significant speed-ups. We evaluated our parsers in 14 languages, with state-of-the-art results. Our experiments confirmed that the AD^3 algorithm is particularly suitable for handling many overlapping components, comparing favourably against the projected subgradient method in several aspects: it is faster to reach a consensus, it has better stopping conditions, and it works better in non-lightweight decompositions.[21]

There are several ways in which we could extend the work presented in this chapter. Many other global features or constraints could be thrown into the parsing model, for example using prior linguistic knowledge. An example of global features that might be useful are features that depend on *word spans*—this sort of features is useful in phrase-structure parsers, and can be incorporated here by defining logical operations on the path variables. Basically, the span of a word h is (i, j) if i is the leftmost word for which there is a path from h to i, and j is the rightmost word for which there is a path from h to j. It is straighforward to incorporate those features by employing the logic factors described in Chapter 5.

Other syntactic formalisms, such as phrase-structure grammars, tree-adjoining grammars, or lexical-functional grammars may also be addressed with similar approaches. AD^3 may be useful in other frameworks involving logical constraints, such as the models for compositional semantics presented by Liang et al. (2011). Non-logical constraints may also yield efficient subproblems, *e.g.*, the budget constraints in summarization and compression (Clarke and Lapata, 2008; Martins and Smith, 2009; Berg-Kirkpatrick et al., 2011). The marginal polytope of such factors (for a budget B) is

$$\mathcal{Z}_B := \left\{ (z_1, \ldots, z_K) \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} z_k \leq B \right\}; \tag{7.76}$$

computing the MAP amounts to obtaining the B variables with the highest positive scores. Projecting onto the set $\mathcal{Z}_B$ can be done with cyclic projection algorithms, or employing the active set method described in Chapter 6.

[21] In Martins et al. (2011a), we have also shown that AD^3 outperforms MPLP and Star-MSD, although our experiments there are for dependency parsers with fewer features.

		# Slaves	Runtime	Description
Tree	$\exists! h : \texttt{arc}(h, m), \quad m \neq 0$	$O(L)$	$O(L \log L)$	Each non-root word has a head
	$\texttt{flow}(h, m, k) \Rightarrow \texttt{arc}(h, m)$	$O(L^3)$	$O(1)$	Only active arcs may carry flow
	$\texttt{path}(m, d) := \exists! h : \texttt{flow}(h, m, d), \quad m \neq 0$	$O(L^2)$	$O(L \log L)$	
	$\texttt{path}(h, d) := \exists! m : \texttt{flow}(h, m, d)$	$O(n^2)$	$O(n \log n)$	Paths and flows are consistent
	$\texttt{path}(0, m) := \textsc{true}, \quad \texttt{flow}(h, m, m) := \textsc{true}$			
All siblings	$\texttt{sibl}(h, m, s) := \texttt{arc}(h, m) \wedge \texttt{arc}(h, s)$	$O(L^3)$	$O(1)$	By definition
Grandp.	$\texttt{grand}(g, h, m) := \texttt{arc}(g, h) \wedge \texttt{arc}(h, m)$	$O(n^3)$	$O(1)$	By definition
Head Bigram	$\texttt{bigram}(b, h, m) := \texttt{arc}(b, m-1) \wedge \texttt{arc}(h, m), \quad m \neq 0$	$O(n^3)$	$O(1)$	By definition
Consec. Sibl.	$\texttt{lastsibl}(h, m, m) := \texttt{arc}(h, m)$			
	$\exists! m \in [h, k] : \texttt{lastsibl}(h, m, k)$	$O(L^2)$	$O(L \log L)$	Head automaton model
	$\texttt{lastsibl}(h, m, k) := \texttt{lastsibl}(h, m, k+1)$			
	$\quad \oplus \texttt{nextsibl}(h, m, k+1)$	$O(L^3)$	$O(1)$	
	$\texttt{arc}(h, m) := \exists! s \in [h, m] : \texttt{nextsibl}(h, s, m)$	$O(L^2)$	$O(L \log L)$	
Nonproj. Arc	$\texttt{nonproj}(h, m) := \texttt{arc}(h, m) \wedge \exists k \in [h, m] : \neg \texttt{path}(h, k)$	$O(L^2)$	$O(L \log L)$	By definition

Table 7.4: First-order logic formulae underlying our dependency parser. The basic parts are the predicate variables $\texttt{arc}(h, m)$ (indicating an arc linking head h to modifier m), $\texttt{path}(a, d)$ (indicating a directed path from ancestor a to descendant d), $\texttt{nextsibl}(h, m, s)$ (indicating that (h, m) and (h, s) are consecutive siblings), $\texttt{nonproj}(h, m)$ (indicating that (h, m) is a non-projective arc), as well as the auxiliary variables $\texttt{flow}(h, m, d)$ (indicating that arc (h, m) carries flow to d), and $\texttt{lastsibl}(h, m, k)$ (indicating that, up to position k, the last seen modifier of h occurred at position m). The non-basic parts are the pairwise factors $\texttt{sibl}(h, m, s)$, $\texttt{grand}(g, h, m)$, and $\texttt{bigram}(b, h, m)$; as well as each logical formula. Columns 3–4 indicate the number of parts of each kind, and the time complexity for solving each subproblem with AD^3. For a sentence of length L, there are $O(L^3)$ parts and the total complexity is $O(L^3 \log L)$.

Part III

Learning

Chapter 8

Learning Structured Classifiers with Dual Coordinate Ascent

In the previous three chapters, we have presented novel contributions for structured inference, with applications to natural language processing. Along the way, we have always assumed possession of a pre-trained structured model, ready to make predictions. We now switch gears and address the *learning* problem, which concerns the training of such models. The following original contributions are presented in this chapter:

- We present a unified framework for learning structured classifiers that handles a wide family of convex loss functions, properly including CRFs, structured SVMs, and the structured perceptron (see Chapter 3 for background on these losses).

- We introduce a new class of online algorithms that optimize any loss in this family. For the structured hinge loss, the algorithm reduces to 1-best MIRA (Crammer and Singer, 2003; McDonald et al., 2005a).[1] For the structured logistic loss, we obtain a variant of MIRA for CRFs.

- We show that these algorithms implicitly perform coordinate ascent in a dual formulation, generalizing the framework established by Shalev-Shwartz and Singer (2006) to a larger set of loss functions and to structured output prediction.

- We address the impact of approximate inference on the learning problem.

Our experiments on two NLP problems (named entity recognition and dependency parsing) show that our algorithm converges to accurate models at least as fast as stochastic gradient descent, without the need to specify any learning rate parameter.

Some of the results presented in this chapter—namely the family of loss functions and the new online algorithm—were originally introduced in Martins et al. (2010f), and are described more broadly in a technical report (Martins et al., 2010c). We will refer occasionally to results presented in Martins et al. (2009c), which concerns the impact of approximate LP-MAP inference in the learning procedure.

[1] Also called "online passive-aggressive" by Crammer et al. (2006).

8.1 Motivation and Previous Work

We have seen in Chapter 2 that many important problems in NLP, such as tagging, named entity recognition, and parsing, require classifiers with structured outputs. Learning those classifiers discriminatively typically involves the minimization of a regularized loss function, of the kind described in Section 3.4. The well-known cases of CRFs (Lafferty et al., 2001) and structured SVMs (Taskar et al., 2003; Tsochantaridis et al., 2004; Altun et al., 2003) correspond to different choices of loss functions. For large-scale settings, the underlying optimization problem is often difficult to tackle in its batch form, making online algorithms desirable. Some examples, discussed in Section 3.5, are the structured perceptron (Collins, 2002a), stochastic gradient descent (SGD) (LeCun et al., 1998), and the margin infused relaxed algorithm (MIRA) (Crammer and Singer, 2003; Crammer et al., 2006).

In this chapter, we start by establishing a *unified representation* for several convex loss functions traditionally used in structured classification (Section 8.2). We will use results derived in Section 5.2.2 (namely, Proposition 5.3) to describe how all these losses can be expressed in *variational form* as optimization problems over the marginal polytope. After doing so, we make use of convex duality to derive new *online learning algorithms* (Section 8.4) that share the "passive-aggressive" property of MIRA but can be applied to a wider variety of loss functions. This includes as a particular case the logistic loss that underlies CRFs, yielding a procedure analogous to MIRA for training probabilistic models. The primal-dual interpretation of such algorithms builds on previous work by Shalev-Shwartz and Singer (2006) and Kakade and Shalev-Shwartz (2008), who regard them as dual coordinate ascent procedures. We generalize their framework to a larger set of loss functions and to structured prediction.

The updates that we derive in Section 8.4 share the remarkable simplicity of SGD, with an important advantage:

Our algorithms do *not* require the specification of a learning rate parameter.

That is, they sidestep what is arguably the most annoying aspect of SGD: tuning a learning rate parameter and specifying an annealing schedule. Instead, the step sizes are *automatically* computed as a function of the loss and of its gradient. This is a difference with respect to standard gradient methods, which only require gradient information for computing the updates, dispensing the *evaluation* of the loss function itself. Nevertheless, the additional computation required for loss evaluations is negligible in the case of the family of loss functions herein addressed, since the methods used to compute the gradient also provide the value of the loss for free.

Two important problems in NLP provide an experimental testbed (Section 8.5): *named entity recognition* and *dependency parsing*. We employ feature-rich models where exact inference is sometimes intractable. To be as general as possible, we devise a framework that fits any structured classification problem representable as a factor graph with soft and hard constraints (Section 8.2); this includes problems with loopy graphs, such as some variants of the dependency parsers of Smith and Eisner (2008).

8.2 A Family of Loss Functions for Structured Classification

We assume the general framework discussed in Chapter 3 for learning structured linear models, in which the learning problem takes the form:

$$\begin{array}{ll} \text{minimize} & \Omega(\boldsymbol{w}) + \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{w}; x^n, y^n) \\ \text{w.r.t.} & \boldsymbol{w} \in \mathbb{R}^D, \end{array} \tag{8.1}$$

where $\Omega : \mathbb{R}^D \to \mathbb{R}$ is a convex regularizer and $L : \mathbb{R}^D \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a convex loss function. In this chapter, we assume L_2-regularization, $\Omega(\boldsymbol{w}) := \Omega_\lambda^{L_2}(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2$, and consider the following family of loss functions parametrized by non-negative scalars β and γ:

$$\boxed{L_{\beta,\gamma}(\boldsymbol{w}; x, y) := \frac{1}{\beta}\log \sum_{y' \in \mathcal{Y}(x)} \exp\Big[\beta\Big(\boldsymbol{w}\cdot\big(\boldsymbol{f}(x,y') - \boldsymbol{f}(x,y)\big) + \gamma\rho(y',y)\Big)\Big].} \tag{8.2}$$

This family subsumes some well-known cases:

- The *logistic loss* (in CRFs), $L_{\text{CRF}}(\boldsymbol{w}; x, y) := -\log P_{\boldsymbol{w}}(y|x)$, corresponds to $\beta = 1$ and $\gamma = 0$ (cf. Eq. 3.16).
- The *hinge loss* of structured SVMs, $L_{\text{SSVM}}(\boldsymbol{w}; x, y) := \max_{y' \in \mathcal{Y}(x)} \boldsymbol{w}^\top(\boldsymbol{f}(x,y') - \boldsymbol{f}(x,y)) + \rho(y',y)$, corresponds to the limit case $\beta \to \infty$ and any $\gamma > 0$ (cf. Eq. 3.20).
- The loss L_{SP} underlying the structured perceptron is obtained for $\beta \to \infty$ and $\gamma = 0$ (cf. Eq. 3.31).
- The *softmax-margin loss* recently proposed in Gimpel and Smith (2010) is obtained with $\beta = \gamma = 1$.

For any choice of $\beta > 0$ and $\gamma \geq 0$, the resulting loss function is convex in $\boldsymbol{w}$, since, up to a scale factor, it is the composition of the (convex) log-sum-exp function with an affine map.[2] In Section 8.4 we present a dual coordinate ascent online algorithm to handle the learning problem in Eq. 8.1, for this family of losses.

8.3 Loss Evaluation and Differentiation

Recall that in Chapter 5 we have considered constrained factor graphs $\mathcal{G}$, which assume a decomposition of the feature vector over *places*,

$$\boldsymbol{f}(x,\boldsymbol{y}) = \sum_{p \in \mathcal{P}} \boldsymbol{f}_p(x, \boldsymbol{y}_p), \tag{8.3}$$

where the places are the *variable nodes* and the *soft factor nodes* of the factor graph $\mathcal{G}$. We considered the corresponding set of *parts* $\mathcal{R} = \{(p, \boldsymbol{y}_p) \mid p \in \mathcal{P}, \boldsymbol{y}_p \in \mathcal{Y}_p\}$, and formed the D-by-$|\mathcal{R}|$ feature matrix $\mathbf{F}(x)$, whose columns are the local feature vectors $\boldsymbol{f}_p(x, \boldsymbol{y}_p)$, for

[2] Some important non-convex losses can also be written as differences of losses in this family. By defining $\delta L_{\beta,\gamma} = L_{\beta,\gamma} - L_{\beta,0}$, the case $\beta = 1$ yields $\delta L_{\beta,\gamma}(\boldsymbol{w}; x, y) = \log \mathbb{E}_{\boldsymbol{w}} \exp \rho(Y, y)$, which is an upper bound on $\mathbb{E}_{\boldsymbol{w}}\rho(Y, y)$, used in minimum risk training (Smith and Eisner, 2006). For $\beta = \infty$, $\delta L_{\beta,\gamma}$ becomes a structured *ramp loss* (Collobert et al., 2006).

each $(p, \boldsymbol{y}_p) \in \mathcal{R}$. Then, we discussed the geometry of constrained factor graphs, and in Proposition 5.3 we established the following variational representation of the log-partition function (cf. Eq. 5.20):

$$\log Z(\boldsymbol{w}, x) = \max_{\boldsymbol{\mu} \in \text{MARG}(\mathcal{G})} \boldsymbol{w}^\top \mathbf{F}(x) \boldsymbol{\mu} + H(\boldsymbol{\mu}), \tag{8.4}$$

where H is the entropy function expressed in terms of the marginal parametrization $\boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{w})$, *i.e.*:

$$H(\boldsymbol{\mu}) := \begin{cases} \mathbb{E}_{\boldsymbol{w}}[-\log P_{\boldsymbol{w}}(\boldsymbol{Y}|X = x)] & \text{if there is } \boldsymbol{w} \text{ such that } \boldsymbol{\mu} = \boldsymbol{\mu}(\boldsymbol{w}) \\ -\infty & \text{otherwise.} \end{cases} \tag{8.5}$$

We now invoke Proposition 5.3 to derive a variational expression for evaluating any loss $L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ in (8.2), and compute its gradient as a by-product.[3] This is crucial for the learning algorithms to be introduced in Section 8.4. Our only assumption is that the cost function $\rho(y', y)$ is decomposable. Recall from Definition 3.3 that a cost function is called *decomposable* if it can be written as $\rho(\widehat{y}, y) = \sum_{p \in \mathcal{P}} \rho_p(\widehat{y}_p, y_p)$, where each ρ_p is a local cost function.

Lemma 8.1 (Decomposable cost as a linear function.) *Given $y \in \mathcal{Y}(x)$, any decomposable cost function $\rho(., y)$, seen as a function of the first argument, can be written as a linear function of the output indicator vector:*

$$\rho(\widehat{y}, y) = \boldsymbol{c}(y) \cdot \boldsymbol{\chi}(\widehat{y}), \tag{8.6}$$

where $\boldsymbol{c}(y) \in \mathbb{R}^{|\mathcal{R}|}$.

Proof. Trivial: set $[c(y)]_{p, \boldsymbol{y}'_p} = \rho_p(y'_p, y_p)$. ∎

For example, for the Hamming loss, Lemma 8.1 holds with $\boldsymbol{c}(y) = \mathbf{1} - \boldsymbol{\chi}(y)$; Other examples are shown in Taskar et al. (2006b). Crucially, if the cost ρ in the family (8.2) is decomposable, then we have a variational representation for the loss functions in this family, as the next proposition asserts.

Proposition 8.2 (Loss and gradient.) *Let ρ be a decomposable cost function written in the form (8.6). The following variational representation for the loss $L_{\beta,\gamma}$ holds:*

$$L_{\beta,\gamma}(\boldsymbol{w}; x, y) = \max_{\boldsymbol{\mu} \in \text{MARG}(\mathcal{G})} \boldsymbol{w}^\top \mathbf{F}(x)(\boldsymbol{\mu} - \boldsymbol{\chi}(y)) + \frac{1}{\beta} H(\boldsymbol{\mu}) + \gamma \boldsymbol{c}(y)^\top \boldsymbol{\mu}. \tag{8.7}$$

Furthermore, the gradient of $L_{\beta,\gamma}$ at $\boldsymbol{w}$ is given by:

$$\nabla L_{\beta,\gamma}(\boldsymbol{w}; x, y) = \mathbf{F}(x)(\widehat{\boldsymbol{\mu}} - \boldsymbol{\chi}(y)). \tag{8.8}$$

where $\widehat{\boldsymbol{\mu}}$ is a maximizer in Eq. 8.7.

Proof. Let $\boldsymbol{\theta} = \mathbf{F}(x)^\top \boldsymbol{w}$ be the vector of factor log-potentials. Under the decomposable cost assumption, $L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ becomes expressible in terms of the log-partition function of

[3] Our description also applies to the (non-differentiable) hinge loss case, when $\beta \to \infty$, if we replace all instances of "the gradient" in the text by "a subgradient."

a distribution whose log-potentials are set to $\beta(\boldsymbol{\theta} + \gamma \boldsymbol{c}(y))$. From the first statement of Proposition 5.3, we immediately obtain Eq. 8.7. Now, let $\widehat{\boldsymbol{\mu}}$ be a maximizer in (8.7); from the second statement of Proposition 5.3 we obtain Eq. 8.8. ■

Two examples follow.

Sequence Labeling. Without hard constraints, the graphical model does not contain loops, and therefore $L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ and $\nabla L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ may be easily computed by setting the log-potentials as described above and running the forward-backward algorithm.

Dependency Parsing. For the arc-factored model, $L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ and $\nabla L_{\beta,\gamma}(\boldsymbol{w}; x, y)$ may be computed exactly by modifying the log-potentials, and invoking the matrix-tree theorem to compute the log-partition function and the marginals (Smith and Smith, 2007; Koo et al., 2007; McDonald and Satta, 2007), as seen in Chapter 2. For richer models where arc interactions are considered, exact inference is intractable. Both the marginal polytope and the entropy lack concise closed form expressions. We have discussed two factor graph representations in Section 7.3, which underlie two approximate approaches: the loopy BP algorithm for computing pseudo-marginals (Smith and Eisner, 2008); and the LP-relaxation method for approximating the most likely parse tree (Martins et al., 2009b). Both methods optimize over outer bounds of the marginal polytope, as discussed in Section 7.3.

8.4 Online Learning with Dual Coordinate Ascent

We now proceed to the main contribution of this chapter, a *dual coordinate ascent* approach to learn the model parameters $\boldsymbol{w}$. This approach extends the primal-dual view of online algorithms put forth by Shalev-Shwartz and Singer (2006) to structured classification; it handles any loss in the family (8.2). In the case of the hinge loss, we recover the online passive-aggressive algorithm of Crammer et al. 2006 (also known as MIRA), as well as its K-best variants. With the logistic loss, we obtain a new passive-aggressive algorithm for CRFs.

Start by noting that the learning problem in Eq. 8.1 is not affected by scaling the objective by N. It will also be convenient to pull the regularization constant λ out of the regularizer; in the L_2 case this is done by writing $\Omega_\lambda^{L_2} = \lambda \Omega_1^{L_2}$. With this in mind, consider a sequence of primal objectives $P_1(\boldsymbol{w}), \ldots, P_{N+1}(\boldsymbol{w})$ to be minimized, each of the form

$$P_t(\boldsymbol{w}) = \lambda N \Omega(\boldsymbol{w}) + \sum_{i=1}^{t-1} L(\boldsymbol{w}; x^i, y^i).$$

Our goal is to minimize $P_{N+1}(\boldsymbol{w})$; for simplicity we consider online algorithms with only one pass over the data, but the analysis can be extended to the case where multiple epochs are allowed.

Below, we let $\bar{\mathbb{R}} := \mathbb{R} \cup \{+\infty\}$ be the extended reals and, given a function $f : \mathbb{R}^D \to \bar{\mathbb{R}}$, we denote by $f^\star : \mathbb{R}^D \to \bar{\mathbb{R}}$ its *convex conjugate*, $f^\star(\boldsymbol{v}) = \sup_{\boldsymbol{u}} \boldsymbol{u} \cdot \boldsymbol{v} - f(\boldsymbol{u})$ (see Appendix B for a background of convex analysis). The next proposition, proved in (Kakade and Shalev-Shwartz, 2008), states a generalized form of Fenchel duality, which involves a dual vector $\boldsymbol{\xi}_i \in \mathbb{R}^d$ per each instance.

Proposition 8.3 (Kakade and Shalev-Shwartz 2008) *The Lagrange dual of* $\min_w P_t(w)$ *is*

$$\max_{\xi_1,\ldots,\xi_{t-1}} D_t(\xi_1,\ldots,\xi_{t-1}), \tag{8.9}$$

where

$$D_t(\xi_1,\ldots,\xi_{t-1}) = -\lambda N \Omega^\star\left(-\frac{1}{\lambda N}\sum_{i=1}^{t-1}\xi_i\right) - \sum_{i=1}^{t-1} L^\star(\xi_i; x^i, y^i). \tag{8.10}$$

If $\Omega(w) = \Omega_1^{L_2}(w) = \frac{1}{2}\|w\|^2$*, then* $\Omega = \Omega^\star$*, and strong duality holds for any convex* L*, i.e.,* $P_t(w^*) = D_t(\xi_1^*,\ldots,\xi_{t-1}^*)$ *where* w^* *and* $\xi_1^*,\ldots,\xi_{t-1}^*$ *are respectively the primal and dual optima. Moreover, the following primal-dual relation holds:*

$$w^* = -\frac{1}{\lambda N}\sum_{i=1}^{t-1}\xi_i^*. \tag{8.11}$$

We can therefore transform our problem into that of maximizing $D_{N+1}(\xi_1,\ldots,\xi_N)$. *Dual coordinate ascent* (DCA) is an umbrella name for algorithms that manipulate a single dual coordinate at a time. In our setting, the largest such improvement at round t is achieved by $\xi_t := \arg\max_\xi D_{t+1}(\xi_1,\ldots,\xi_{t-1},\xi)$. The next proposition characterizes the mapping of this subproblem back into the primal space, shedding light on the connections with known online algorithms.

Proposition 8.4 *Let* $w^t := -\frac{1}{\lambda N}\sum_{i=1}^{t-1}\xi_i$*. The Lagrange dual of* $\max_\xi D_{t+1}(\xi_1,\ldots,\xi_{t-1},\xi)$ *is*

$$\min_w \frac{\lambda N}{2}\|w - w^t\|^2 + L(w; x^t, y^t). \tag{8.12}$$

Proof. From (8.10),

$$\begin{aligned}
&\max_\xi D_{t+1}(\xi_1,\ldots,\xi_{t-1},\xi)\\
&= \max_\xi -\frac{1}{2\lambda N}\left\|\sum_{i=1}^{t-1}\xi_i + \xi\right\|^2 - L^\star(\xi; x^t, y^t) - \sum_{i=1}^{t-1} L^\star(\xi_i; x^i, y^i)\\
&= \max_\xi -\frac{1}{2\lambda N}\left\|-\lambda N w^t + \xi\right\|^2 - L^\star(\xi; x^t, y^t) + \text{constant}\\
&=^{\text{(i)}} \max_\xi -\frac{1}{2\lambda N}\left\|-\lambda N w^t + \xi\right\|^2 - \max_w(\xi\cdot w - L(w; x^t, y^t)) + \text{constant}\\
&=^{\text{(ii)}} \min_w \max_\xi -\frac{1}{2\lambda N}\left\|-\lambda N w^t + \xi\right\|^2 - \xi\cdot w + L(w; x^t, y^t) + \text{constant}\\
&= \min_w\left(\max_\xi \xi\cdot(-w) - \frac{1}{2\lambda N}\left\|\xi - \lambda N w^t\right\|^2\right) + L(w; x^t, y^t) + \text{constant}\\
&=^{\text{(iii)}} \min_w \frac{\lambda N}{2}\left\|w - w^t\right\|^2 + L(w; x^t, y^t) + \text{constant},
\end{aligned} \tag{8.13}$$

where in (i) we invoked the definition of convex conjugate; in (ii) we interchange min and max since strong duality holds (as stated by Kakade and Shalev-Shwartz (2008), a sufficient condition is that Ω is strongly convex, L is convex and dom L is polyhedral); and in (iii) we used the facts that $\Omega(w) = \|w\|^2/2$ is conjugate of itself, and that $g(u) = tf(u - u_0)$ implies

Algorithm 12 Dual coordinate ascent (DCA)

input: data $\mathcal{D}$, regularization constant λ, number of epochs K
initialize $\boldsymbol{w}^1 = \mathbf{0}$; set $N = |\mathcal{D}|$ and $T = NK$
for $t = 1$ **to** T **do**
  choose $n = n(t) \in \{1, \dots, N\}$ and take training pair (x^n, y^n)
  update $\boldsymbol{w}^{t+1}$ by solving (8.12) exactly or approximately (see Algorithm 13)
end for
output: the last weight vector $\widehat{\boldsymbol{w}} \leftarrow \boldsymbol{w}^{T+1}$, or the average $\widehat{\boldsymbol{w}} \leftarrow \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}^{t+1}$.

Algorithm 13 Parameter updates in DCA

input: current model $\boldsymbol{w}^t$, instance (x, y), regularization constant λ
obtain $\boldsymbol{\chi}(y)$ from y
solve the variational problem in Eq. 8.7 to obtain $\widehat{\boldsymbol{\mu}}$ and $L(\boldsymbol{w}^t, x, y)$
compute the gradient: $\nabla L_{\beta,\gamma}(\boldsymbol{w}^t; x, y) := \mathbf{F}(x)(\widehat{\boldsymbol{\mu}} - \boldsymbol{\chi}(y))$ (Eq. 8.8)
compute stepsize:

$$\eta_t := \min\left\{\frac{1}{\lambda N}, \frac{L(\boldsymbol{w}^t; x, y)}{\|\nabla L(\boldsymbol{w}^t; x, y)\|^2}\right\}$$

update the model: $\boldsymbol{w}^{t+1} := \boldsymbol{w}^t - \eta_t \nabla L(\boldsymbol{w}^t; x, y)$
output: $\boldsymbol{w}^{t+1}$

$g^\star(\boldsymbol{v}) = \boldsymbol{u}_0 \cdot \boldsymbol{v} + t f^\star(\boldsymbol{v}/t)$. ■

Assembling these pieces together yields Alg. 12, where the solution of (8.12) is carried out by Alg. 13, as explained next. While the problem in Eq. 8.12 is easier than the batch problem in Eq. 8.1, an exact solution may still be prohibitively expensive in large-scale settings, particularly because it has to be solved repeatedly. We thus adopt a simpler strategy that still guarantees some improvement in the dual. Noting that L is non-negative, we may rewrite the problem in Eq. 8.12 as

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda N}{2}\|\boldsymbol{w} - \boldsymbol{w}^t\|^2 + \xi \\ \text{w.r.t.} \quad & \boldsymbol{w} \in \mathbb{R}^D,\ \xi \geq 0 \\ \text{s.t.} \quad & L(\boldsymbol{w}; x^t, y^t) \leq \xi. \end{aligned} \tag{8.14}$$

From the convexity of L, we may take its first-order Taylor approximation around $\boldsymbol{w}^t$ to obtain the lower bound $L(\boldsymbol{w}; x^t, y^t) \geq L(\boldsymbol{w}^t; x^t, y^t) + (\boldsymbol{w} - \boldsymbol{w}^t) \cdot \nabla L(\boldsymbol{w}_t; x_t, y_t)$. Therefore the true minimum in (8.12) is lower bounded by the solution value of the following problem:

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda N}{2}\|\boldsymbol{w} - \boldsymbol{w}^t\|^2 + \xi \\ \text{w.r.t.} \quad & \boldsymbol{w} \in \mathbb{R}^D,\ \xi \geq 0 \\ \text{s.t.} \quad & L(\boldsymbol{w}^t; x^t, y^t) + (\boldsymbol{w} - \boldsymbol{w}^t) \cdot \nabla L(\boldsymbol{w}^t; x^t, y^t) \leq \xi. \end{aligned} \tag{8.15}$$

This is a simple Euclidean projection problem with slack, which admits the closed form

solution $\boldsymbol{w}^* = \boldsymbol{w}^t - \eta_t \nabla L(\boldsymbol{w}^t; x^t, y^t)$, with

$$\eta_t = \min \left\{ \frac{1}{\lambda N}, \frac{L(\boldsymbol{w}^t; x^t, y^t)}{\|\nabla L(\boldsymbol{w}^t; x^t, y^t)\|^2} \right\}. \tag{8.16}$$

Example: 1-best MIRA. If L is the structured hinge loss, we obtain, from Eq. 8.8:

$$\nabla L_{\text{SSVM}}(\boldsymbol{w}; x, y) = \mathbf{F}(x)(\boldsymbol{\chi}(\widehat{y}) - \boldsymbol{\chi}(y)) = \boldsymbol{f}(x, \widehat{y}) - \boldsymbol{f}(x, y), \tag{8.17}$$

where $\widehat{y} = \arg\max_{y' \in \mathcal{Y}(x)} \boldsymbol{w} \cdot (\boldsymbol{f}(x, y') - \boldsymbol{f}(x, y)) + \rho(y', y)$. The update becomes

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_t (\boldsymbol{f}(x^t, \widehat{y}^t) - \boldsymbol{f}(x^t, y^t)), \tag{8.18}$$

with

$$\eta_t = \min \left\{ \frac{1}{\lambda N}, \frac{\boldsymbol{w}^t \cdot (\boldsymbol{f}(x^t, \widehat{y}^t) - \boldsymbol{f}(x^t, y^t)) + \rho(\widehat{y}^t, y^t)}{\|\boldsymbol{f}(x^t, \widehat{y}^t) - \boldsymbol{f}(x^t, y^t)\|^2} \right\}. \tag{8.19}$$

This is precisely the max-loss variant of the 1-best MIRA algorithm (Crammer et al., 2006, Sect. 8), which we have described in our Section 3.5.4. Hence, while MIRA was originally motivated by a conservativeness-correctness tradeoff, it turns out that it also performs coordinate ascent in the dual.

Example: CRFs. This framework immediately allows us to extend 1-best MIRA for CRFs, which optimizes the structured logistic loss. In that case, the exact problem in (8.14) can be expressed as

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda N}{2} \|\boldsymbol{w} - \boldsymbol{w}^t\|^2 + \xi \\ \text{w.r.t.} \quad & \boldsymbol{w} \in \mathbb{R}^D, \ \xi \geq 0 \\ \text{s.t.} \quad & -\log P_{\boldsymbol{w}}(y^t | x^t) \leq \xi. \end{aligned} \tag{8.20}$$

In words: stay as close as possible to the previous parameter vector, but correct the model so that the conditional probability $P_{\boldsymbol{w}}(y_t | x_t)$ becomes large enough. From Eq. 8.8, we have

$$\begin{aligned} \nabla L_{\text{CRF}}(\boldsymbol{w}; x, y) &= \mathbf{F}(x)(\widehat{\boldsymbol{\mu}} - \boldsymbol{\chi}(y)) \\ &= \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{f}(x, Y)] - \boldsymbol{f}(x, y), \end{aligned} \tag{8.21}$$

where now $\widehat{\boldsymbol{\mu}}$ is an expectation instead of a mode. The update becomes

$$\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta_t (\mathbb{E}_{\boldsymbol{w}^t}[\boldsymbol{f}(x^t, Y^t)] - \boldsymbol{f}(x^t, y^t)), \tag{8.22}$$

with

$$\begin{aligned} \eta_t &= \min \left\{ \frac{1}{\lambda N}, \frac{\boldsymbol{w}^t \cdot (\mathbb{E}_{\boldsymbol{w}^t}[\boldsymbol{f}(x^t, Y^t)] - \boldsymbol{f}(x^t, y^t)) + H(P_{\boldsymbol{w}^t}(.|x_t))}{\|\mathbb{E}_{\boldsymbol{w}^t}[\boldsymbol{f}(x^t, Y^t)] - \boldsymbol{f}(x^t, y^t)\|^2} \right\} \\ &= \min \left\{ \frac{1}{\lambda N}, \frac{-\log P_{\boldsymbol{w}}(y^t | x^t)}{\|\mathbb{E}_{\boldsymbol{w}^t}[\boldsymbol{f}(x^t, Y^t)] - \boldsymbol{f}(x^t, y^t)\|^2} \right\}. \end{aligned} \tag{8.23}$$

Thus, the difference with respect to standard 1-best MIRA (8.19) consists of replacing the feature vector of the *loss-augmented mode* $\boldsymbol{f}(x^t, \widehat{y}^t)$ by the *expected* feature vector $\mathbb{E}_{\boldsymbol{w}^t}[\boldsymbol{f}(x^t, Y^t)]$, and the value of the *cost function* $\rho(\widehat{y}^t, y^t)$ by that of the *entropy function* $H(P_{\boldsymbol{w}^t}(.|x^t))$. The difference with respect to SGD is that the stepsize η_t is automatically adjusted through Eq. 8.23.

Example: k-best MIRA. Tighter approximations to the problem in Eq. 8.12 can be built by using the variational representation machinery; see Eq. 8.7 for losses in the family $L_{\beta,\gamma}$. Plugging this variational representation into the constraint in (8.14) we obtain the following *semi-infinite* quadratic program:

$$\begin{aligned} \text{minimize} \quad & \frac{\lambda N}{2}\|\boldsymbol{w} - \boldsymbol{w}^t\|^2 + \xi \\ \text{w.r.t.} \quad & \boldsymbol{w} \in \mathbb{R}^D,\ \xi \geq 0 \\ \text{s.t.} \quad & \boldsymbol{w} \in \mathcal{H}(\boldsymbol{\mu}; y^t, \beta, \gamma), \quad \forall \boldsymbol{\mu} \in \text{MARG}(\mathcal{G}), \end{aligned} \tag{8.24}$$

where each $\mathcal{H}(\boldsymbol{\mu}; y, \beta, \gamma) := \{\boldsymbol{w} \in \mathbb{R}^D \mid \boldsymbol{a}^\top \boldsymbol{w} \leq b\}$ is a *half-space*, with $\boldsymbol{a} = \mathbf{F}(x)(\boldsymbol{\mu} - \boldsymbol{\chi}(y))$ and $b = \xi - \gamma(\boldsymbol{c}^\top \boldsymbol{\mu}) - \beta^{-1} H(\boldsymbol{\mu})$. The constraint set in (8.24) is a convex set defined by the intersection of uncountably many half-spaces (indexed by the points in the marginal polytope).[4] Our first-order Taylor approximation consisted of relaxing the problem in Eq. 8.24 by discarding all half-spaces except the one indexed by the $\widehat{\boldsymbol{\mu}}$ which solved the variational problem (8.7). However, tigher relaxations can be obtained by keeping some of the other half-spaces. For the hinge loss, rather than just using the mode $\widehat{\boldsymbol{\mu}} = \boldsymbol{\chi}(\widehat{y})$, we may well rank the k-best outputs and add a half-space constraint for each. This procedure approximates the constraint set by a polyhedron and the resulting problem can be addressed using row-action methods, such as Hildreth's algorithm (Censor and Zenios, 1997). This corresponds precisely to k-best MIRA.[5]

8.5 Experiments

We report experiments on two tasks: *named entity recognition* and *dependency parsing*.[6] For each, we compare DCA (Algorithm 12) with SGD. We report results for several values of the regularization parameter $C = 1/(\lambda N)$. The learning rate schedule for SGD is set according to the formula $\eta_t = \eta/(1 + t/N)$, as suggested by LeCun et al. (1998). We choose η using dev-set validation after a single epoch, as in Collins et al. (2008).

Named Entity Recognition. We use the English data from the CoNLL 2003 shared task (Tjong Kim Sang and De Meulder, 2003), which consist of English news articles annotated with four entity types: person, location, organization, and miscellaneous. We used a standard set of feature templates, as in Kazama and Torisawa (2007), with token shape features (Collins,

[4] Interestingly, when the hinge loss is used, only a finite (albeit exponentially many) of these half-spaces are necessary, those indexed by *vertices* of the marginal polytope. In that case, the constraint set is polyhedral.

[5] The *prediction-based* variant of 1-best MIRA (Crammer et al., 2006) is also a particular case, where one also picks a single half-space, the one indexed by the *prediction* under the current model $\boldsymbol{w}^t$, rather than the mode of $L_{\text{SSVM}}(\boldsymbol{w}^t, x^t, y^t)$.

[6] The experiments in named entity recognition were made by Kevin Gimpel, co-author of Martins et al. (2010c), who also generated the corresponding plots.

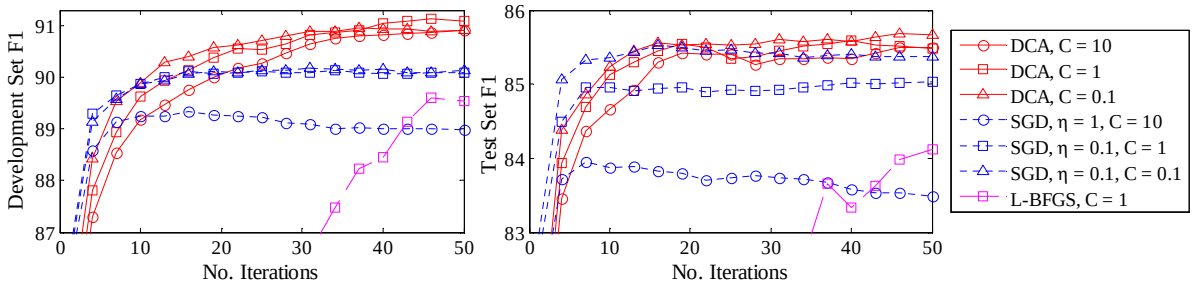


Figure 8.1: Named entity recognition. Learning curves for DCA (Alg. 12), SGD, and L-BFGS. The SGD curve for $C = 10$ is lower than the others because dev-set validation chose a suboptimal value of η. DCA, by contrast, does not require choosing any hyperparameters other than C. L-BFGS ultimately converges after 121 iterations to an F_1 of 90.53 on the development data and 85.31 on the test data.

2002b) and simple gazetteer features; a feature was included iff it occurs at least once in the training set (total 1,312,255 features). The task is evaluated using the F_1 measure computed at the granularity of entire entities. We set $\beta = 1$ and $\gamma = 0$ (the CRF case). In addition to SGD, we also compare with L-BFGS (Liu and Nocedal, 1989), a common choice for optimizing conditional log-likelihood. We used $\{10^a, a = -3, \ldots, 2\}$ for the set of values considered for η in SGD. Figure 8.1 shows that DCA (which only requires tuning one hyperparameter) reaches better-performing models than the baselines.

Dependency Parsing. We trained non-projective dependency parsers for three languages (Arabic, Danish, and English), using datasets from the CoNLL-X and CoNLL-2008 shared tasks (Buchholz and Marsi, 2006; Surdeanu et al., 2008). Performance is assessed by the unlabeled attachment score (UAS), the fraction of non-punctuation words which were assigned the correct parent. We adapted the *turbo parsers* described in Chapter 7 to handle any loss function $L_{\beta,\gamma}$, via Algorithm 12; for decoding, we used the sum-product loopy BP algorithm of Smith and Eisner (2008). We used the pruning strategy described in (Martins et al., 2009b) and tried two feature configurations: one arc-factored model, for which decoding is exact, and another model with second-order features (siblings and grandparents) for which it is approximate.

The comparison with SGD for the CRF case is shown in Figure 8.2. For the arc-factored models, the learning curve of DCA seems to lead faster to an accurate model. Notice that the plots do not account for the fact that SGD requires four extra iterations to choose the learning rate. For the second-order models of Danish and English, however, DCA did not perform as well.[7]

Finally, Table 8.1 shows results obtained for different settings of β and γ.[8] Interestingly, we observe that the higher scores are obtained for loss functions that are "between" SVMs

[7]Further analysis showed that for $\sim$15% of the training instances, loopy BP led to very poor variational approximations of $\log Z(\boldsymbol{w}, x)$, yielding estimates $P_{\boldsymbol{w}^t}(y^t|x^t) > 1$, thus a negative learning rate (see (8.23)), that we truncate to zero. Thus, no update occurs for those instances, explaining the slower convergence. A possible way to fix this problem is to use techniques that guarantee upper bounds on the log-partition function (Wainwright and Jordan, 2008).

[8]Observe that there are only two degrees of freedom: indeed, (λ, β, γ) and $(\lambda', \beta', \gamma')$ lead to equivalent learning problems if $\lambda' = \lambda/a$, $\beta' = \beta/a$ and $\gamma' = a\gamma$ for any $a > 0$, with the solutions related via $\boldsymbol{w}' = a\boldsymbol{w}$.

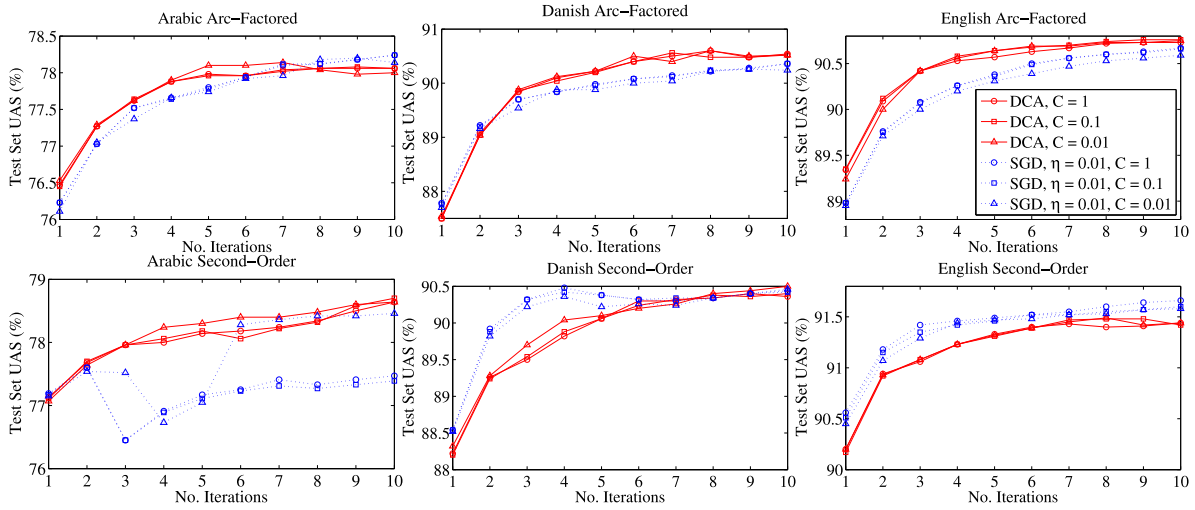


Figure 8.2: Learning curves for DCA (Alg. 12) and SGD, the latter with the learning rate $\eta = 0.01$ chosen from $\{0.001, 0.01, 0.1, 1\}$ using the same procedure as before. The instability when training the second-order models might be due to the fact that inference there is approximate.

	β	1	1	1	1	3	5	∞
	γ	0 (CRF)	1	3	5	1	1	1 (SVM)
NER	BEST C	1.0	10.0	1.0	1.0	1.0	1.0	1.0
	F_1 (%)	85.48	85.54	85.65	85.72	85.55	85.48	85.41
DEPENDENCY PARSING	BEST C	0.1	0.01	0.01	0.01	0.01	0.01	0.1
	UAS (%)	90.76	90.95	91.04	91.01	90.94	90.91	90.75

Table 8.1: Varying β and γ: neither the CRF nor the SVM are optimal. We report only the results for the best C, chosen from $\{0.001, 0.01, 0.1, 1\}$ with dev-set validation. For named entity recognition, we show test set F_1 after $K = 50$ iterations (empty cells will be filled in in the final version). Dependency parsing experiments used the arc-factored model on English and $K = 10$.

and CRFs.

8.6 Conclusions and Future Work

We presented a general framework for aggressive online learning of structured classifiers by optimizing any loss function in a wide family. The technique does not require a learning rate to be specified. We derived an efficient technique for evaluating the loss function and its gradient, which invokes our results in Chapter 5 regarding the variational representation of log-partition functions. The result is an algorithmic framework that puts in the same ground the training of SVMs, CRFs, and the use of loss functions in-between, shedding some light on connections between these loss functions and learning algorithms such as MIRA and the structured perceptron, which are widely used in NLP. Experiments in named entity recognition and dependency parsing showed that the algorithm converges to accurate models at least as fast as stochastic gradient descent.

There are some open problems and paths for future research in the context of the work presented here.

First, note that the primal-dual characterization of our algorithms pretends that they only make one pass over the data; basically, we let the number of dual variables grow with the number of epochs. While this makes sense in the context of online learning, it is not what is usually regarded as dual coordinate ascent (with a fixed number of coordinates). It is straightforward, however, to change our algorithms slightly so that the number of dual variables is N (the dataset size), independent of the number of passes. This, however, requires that one stores, for each instance n, the last solution $\widehat{\boldsymbol{\mu}}_n$ obtained for that instance; then, we need to correct $\boldsymbol{w}^t$, before each parameter update, based on that solution. An evaluation of the modified algorithm will be subject of future work.

Second, the family of loss functions that we have presented leaves open the choice of the hyperparameters β and γ. An automatic choice of these coefficients based on the training data would be highly desirable, and we defer this to future work.

Third, the impact of approximate inference in the learning problem deserves further study. A step forward was given by Kulesza and Pereira (2007), who have shown cases where learning with approximate inference fails. Before that, Wainwright (2006) suggested that it may be beneficial to consider the same approximation (*e.g.*, a tree-reweighted entropy approximation) at training and test times. However, our preliminary experiments in that direction were not very successful. Later, Finley and Joachims (2008) provided an empirical study of learning with LP-relaxations. In Martins et al. (2009c), we provided conditions that guarantee algorithmic separability under LP-relaxed inference. The conditions are that separability in the exact setting must hold with a large enough margin, which depends on a coefficient expressing the tightness of the local polytope approximation. Those conditions, however, are too stringent, and it is possible to find counter-examples in fully connected pairwise MRFs for which they will never hold.[9] We point out, however, that for our parsing problems, the local polytope approximation is much tighter than in fully connected pairwise MRFs. Apart from these geometric characterizations, it appears that the theory of approximation algorithms (Vazirani, 2001) could be useful in this context. We intend to pursue this line of research in the future.

[9] David Sontag, personal communication.

Chapter 9

Online Learning of Structured Predictors with Multiple Kernels

Training structured predictors often requires considerable time and effort from the practicioner to get the model right. Many successful approaches rely on clever feature engineering, or (in the scope of kernel-based machine learning) on the ability of designing a kernel function that suits the task at hand. In this chapter, we propose a new approach for learning structured prediction models that are able to *learn* the kernel function automatically from the data. The following novel contributions are presented:

- We adapt the framework of *multiple kernel learning* (MKL, Lanckriet et al. 2004) to structured prediction.
- We propose *a new family of online algorithms* that can tackle many variants of MKL and group-Lasso. These algorithms can also deal with overlapping groups of features, provided a *co-shrinkage property* that we introduce is satisfied.
- We show regret, convergence, and generalization bounds for those algorithms, mixing results from *online convex programming* (Shalev-Shwartz, 2011) and the *theory of proximity operators* (Bauschke and Combettes, 2011).
- We propose ways of circumventing one of the drawbacks of kernel-based training algorithms in large-scale tasks, its *quadratic dependency* on the training data. Through a small set of preliminary experiments on a handwriting recognition task, we show how accurate models can be obtained rapidly by combining a sparse kernel with a feature-based model.

Most of the results presented in this chapter—with the exception of the co-shrinkage property and a set of toy experiments on online binary classification—were originally introduced in Martins et al. (2010b,e) and further developed in Martins et al. (2011b).

Some of the key results presented here will be later reused in Chapter 10, where we discuss structured sparsity and present a wide set of experiments in structured prediction tasks.

9.1 Motivation and Related Work

Throughtout this thesis, we have dealt with models and algorithms for structured prediction (Lafferty et al., 2001; Taskar et al., 2003; Tsochantaridis et al., 2004). These models have been shown to excel in many tasks, ranging from natural language processing to computer vision, robotics, and computational biology. The untold story is that a substantial effort is often necessary to achieve success in those tasks. Obtaining (structured) predictors with good generalization capability typically involves two steps: (i) designing features or similarity measures (*i.e.*, kernels), (ii) minimizing a regularized empirical loss function. In practice, one typically spends a considerable amount of time iterating over these two steps, as they are strongly interdependent. This loop is particularly expensive when training structured predictors in large scale settings, where achieving state-of-the-art performance usually involves extensive feature/kernel engineering.

Kernel engineering can be partially automated by using the kernel learning approach pioneered by Lanckriet et al. (2004) and Bach et al. (2004), where a combination of multiple kernels is learned from the data. While multi-class and scalable multiple kernel learning (MKL) algorithms have been proposed (Sonnenburg et al., 2006; Zien and Ong, 2007; Rakotomamonjy et al., 2008; Chapelle and Rakotomamonjy, 2008; Xu et al., 2009; Kloft et al., 2010), none are well suited for large-scale structured prediction, for the following reason: they all involve an inner loop in which a standard learning problem (*e.g.*, an SVM) needs to be repeatedly solved; in large-scale structured prediction, as we discussed in Section 3.5, this standard learning problem is too expensive to tackle in batch form, and online methods are usually preferred (Bottou, 1991b; Collins, 2002a; Ratliff et al., 2006; Vishwanathan et al., 2006; Crammer et al., 2006; Collins et al., 2008). These online methods are fast in achieving low generalization error, but converge slowly to the training objective, resulting unattractive for repeated use in the inner loop of a MKL algorithm. Note that early stopping is inadequate, since it can misguide the overall MKL optimization.

In this chapter, we propose a stand-alone online MKL algorithm that iterates between subgradient and proximal steps, which overcomes the above difficulty. Our theoretical analysis and experiments show that the proposed algorithm:

- is simple, flexible, and compatible with sparse and non-sparse MKL;
- can handle composite regularizers, even if they overlap;
- is suitable for large-scale structured prediction;
- can learn time-varying concepts;
- is equipped with regret, convergence, and generalization guarantees.

When applied to structured prediction, the proposed algorithm is termed SPOM (*Structured Prediction by Online MKL*). Our approach extends and kernelizes the forward-backward splitting scheme Fobos (Duchi and Singer, 2009), whose regret bound we improve. In passing, we show how to efficiently compute the proximal projections associated with the squared L_1-norm, despite the fact that the underlying optimization problem is not separable. Finally, we also consider composite regularizers, possibly overlapping, which have recently been the subject of intense research (Bach, 2008b; Zhao et al., 2009; Jenatton et al.,

2009; Liu and Ye, 2010a,b; Mairal et al., 2010). We show that a simple approach where the proximal steps are applied sequentially still converges if the regularizers satisfy the so-called "co-shrinkage" condition.

This chapter is organized as follows: after reviewing structured prediction and MKL (Section 9.2), we present a new class of online proximal algorithms which handle composite regularizers with multiple proximal steps (Section 9.3). We derive convergence rates and show how these algorithms are applicable to MKL, group-Lasso, and other structured sparsity formalisms.[1] In Section 9.4, we apply this algorithm to structured prediction (yielding SPOM), in a handwritten text recognition problem. We obtain encouraging results in terms of runtime and accuracy.

9.2 Multiple Kernel Learning and Group Sparsity

9.2.1 Inference and Learning with Kernels

We assume the setting described in the background chapters, where we have an input set $\mathcal{X}$ and an output set $\mathcal{Y}$. Given an input $x \in \mathcal{X}$, let $\mathcal{Y}(x) \subseteq \mathcal{Y}$ be its set of admissible outputs; in structured prediction, this set is assumed to be structured and exponentially large in the "size" of x. We let $\mathcal{U} := \{(x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}(x)\}$ denote the set of admissible input-output pairs. Given labeled data $\mathcal{D} := ((x_1, y_1), \ldots, (x_N, y_N)) \in \mathcal{U}^N$, the goal is to learn a *compatibility function* $g_{\boldsymbol{w}} : \mathcal{U} \to \mathbb{R}$ that allows to make predictions via

$$x \mapsto \hat{y}(x) := \arg \max_{y \in \mathcal{Y}(x)} g_{\boldsymbol{w}}(x, y). \tag{9.1}$$

Section 3.2 described linear functions, $g_{\boldsymbol{w}}(x, y) := \boldsymbol{w} \cdot \boldsymbol{f}(x, y)$, where $\boldsymbol{w}$ is a parameter vector and $\boldsymbol{f}(x, y)$ a feature vector; here, we still resort to the linear case, but assume more broadly that these vectors live in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, with corresponding kernel $K : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$. We denote by $\langle ., . \rangle_{\mathcal{H}}$ the inner product in $\mathcal{H}$, and by $\|.\|_{\mathcal{H}}$ the corresponding norm, $\|\boldsymbol{w}\|_{\mathcal{H}} := \sqrt{\langle \boldsymbol{w}, \boldsymbol{w} \rangle}$. The kernel evaluation corresponds to the inner product of the feature vectors in $\mathcal{H}$:[2]

$$K((x, y), (x', y')) = \langle \boldsymbol{f}(x, y), \boldsymbol{f}(x', y') \rangle_{\mathcal{H}}. \tag{9.2}$$

For convenience, we often drop the subscript $\mathcal{H}$ when the underlying Hilbert space is clear from the context. In the sequel, features will be used implicitly or explicitly, as convenience determines.

As before, we formulate the learning problem as the minimization of a regularized empirical risk functional,

$$\widehat{g}_{\boldsymbol{w}} := \arg \min_{g_{\boldsymbol{w}} \in \mathcal{H}} \Omega(g_{\boldsymbol{w}}) + \frac{1}{N} \sum_{i=1}^{N} L(g_{\boldsymbol{w}}; x_i, y_i), \tag{9.3}$$

[1] We will elaborate more on their application to structured sparsity in Chapter 10, where we also present additional experiments in sequence labeling and dependency parsing tasks.

[2] We omit details on kernels and reproducing kernel Hilbert spaces. There is an extense literature about kernel-based methods for machine learning. We recommend Schölkopf and Smola (2002) for a thorough description of this subject.

where $\Omega : \mathcal{H} \to \mathbb{R}_+$ is a regularizer, and $L : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a convex loss function. We have described several loss functions in Chapter 3, such as the structured logistic loss of CRFs (Section 3.4.2) and the structured hinge loss of structured SVMs (Section 3.4.3). In kernel-based learning, these losses are written as:

$$L_{\text{CRF}}(g_w; x, y) := \log \sum_{y' \in \mathcal{Y}(x)} \exp\left(g_w(x, y') - g_w(x, y)\right), \tag{9.4}$$

$$L_{\text{SSVM}}(g_w; x, y) := \max_{y' \in \mathcal{Y}(x)} \left(g_w(x, y') - g_w(x, y) + \rho(y', y)\right). \tag{9.5}$$

With L_2-regularization, $\Omega(g_w) \equiv \Omega_\lambda^{L_2}(g_w) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2$, the solution of Eq. 9.3 can be expressed as a kernel expansion, according to the structured version of the representer theorem (Hofmann et al., 2008, Corollary 13). We next discuss alternative forms of regularization that take into consideration another level of structure—now in the feature space.

9.2.2 Block-Norm Regularization and Kernel Learning

Selecting relevant features or picking a good kernel are ubiquitous problems in statistical learning, both of which have been addressed with sparsity-promoting regularizers. We first illustrate this point for the explicit feature case. Often, features exhibit a natural "block" structure: for example, many models in NLP consider *feature templates*—these are binary features indexed by each word w in the vocabulary, by each part-of-speech tag t, by each pair (w, t), etc. Each of these templates correspond to a *block* (also called a *group*) in the feature space $\mathcal{H}$. Thus, $\mathcal{H}$ is endowed with a block structure, where each block (indexed by $m = 1, \ldots, M$) is itself a "smaller" feature space $\mathcal{H}_m$; formally, $\mathcal{H}$ is a direct sum of Hilbert spaces: $\mathcal{H} = \bigoplus_{m=1}^M \mathcal{H}_m$.

Group-Lasso. Consider the goal of learning a model in the presence of many irrelevant feature templates. A well-known criterion is the group-Lasso (Bakin, 1999; Yuan and Lin, 2006), which uses the following block-norm regularizer: $\Omega_\lambda^{\text{GL}}(f_w) = \lambda \sum_{m=1}^M \|\boldsymbol{w}_m\|$. This can be seen as the L_1-norm of the vector containing the L_2-norms of the groups. When Ω_{GL} is used in (9.3), it promotes *group sparsity*, i.e., solutions in which only a few groups are nonzero; within the mth group, either the optimal $\boldsymbol{w}_m^*$ is identically zero—in which case the entire group is discarded—or it is non-sparse. We will come back to group-Lasso regularizers in Chapter 10.

Sparse MKL. In MKL (Lanckriet et al. 2004), the kernel function is learned as a convex combination of M prespecified base kernel functions $\{K_1, \ldots, K_M\}$, *i.e.*, $K = \sum_{m=1}^M \beta_m K_m$, where the coefficients $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)$ are constrained to belonging to the simplex $\Delta^M := \{\boldsymbol{\beta} \geq \mathbf{0} \mid \|\boldsymbol{\beta}\|_1 = 1\}$. This corresponds precisely to the direct sum of RKHS described above, where the RKHS $\mathcal{H}$ induced by K is written as $\mathcal{H} = \bigoplus_{m=1}^M \mathcal{H}_m$, each $\mathcal{H}_m$ being the RKHS induced by K_m. Denoting by $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}_m}$ the inner products associated with K and K_m, respectively, the compatibility function associated with K is $g_w(x, y) = \langle \boldsymbol{w}, \boldsymbol{f}(x, y) \rangle_{\mathcal{H}} = \sum_{m=1}^M \beta_m \langle \boldsymbol{w}_m, \boldsymbol{f}_m(x, y) \rangle_{\mathcal{H}_m} = \sum_{m=1}^M \beta_m g_{\boldsymbol{w}_m}(x, y)$. The kernel learning problem is then formu-

lated as an outer minimization of (9.3) w.r.t. $\boldsymbol{\beta} \in \Delta^M$:

$$\min_{\boldsymbol{\beta}\in\Delta^M} \min_{g_{w_1}\in\mathcal{H}_1,\ldots,g_{w_M}\in\mathcal{H}_M} \frac{\lambda}{2}\sum_{m=1}^{M}\beta_m\|\boldsymbol{w}_m\|^2 + \frac{1}{N}\sum_{i=1}^{N} L\left(\sum_{m=1}^{M}\beta_m g_{w_m}; x_i, y_i\right). \tag{9.6}$$

Remarkably, as shown by Bach et al. (2004) and Rakotomamonjy et al. (2008), a simple change of variable allows transforming this joint optimization over $\boldsymbol{\beta}$ and $\boldsymbol{w}$ into a single optimization of the form (9.3) with a block-structured regularizer $\Omega_{\text{MKL}}(g_w) = \frac{1}{2}(\sum_{m=1}^{M}\|\boldsymbol{w}_m\|)^2$. Note that this coincides with the square of the group-Lasso regularizer; in fact, the two problems are equivalent up to a change of λ (Bach, 2008a). Hence, this MKL formulation promotes sparsity in the number of selected kernels (*i.e.*, only a few nonzero entries in $\boldsymbol{\beta}$).

Non-Sparse MKL. A more general MKL formulation (not necessarily sparse) was recently proposed by Kloft et al. (2010). Define, for $p \geq 1$, the set $\Delta_p^M := \{\boldsymbol{\beta} \geq \mathbf{0} \mid \|\boldsymbol{\beta}\|_p = 1\}$. Then, by modifying the constraint in (9.6) to $\boldsymbol{\beta} \in \Delta_p^M$, the resulting problem can also be written in form (9.3), with the following block-structured regularizer:

$$\Omega_{\text{MKL},q}(g_w) = \frac{1}{2}\left(\sum_{m=1}^{M}\|\boldsymbol{w}_m\|^q\right)^{2/q} := \frac{1}{2}\|\boldsymbol{w}\|_{2,q}^2, \tag{9.7}$$

where $q = 2p/(p+1)$. The function $\|.\|_{2,q}$ satisfies the axioms of a norm, and is called the $L_{2,q}$ mixed norm. Given a solution $\boldsymbol{w}^*$, the optimal kernel coefficients can be computed as $\beta_m^* \propto \|\boldsymbol{w}_m^*\|^{2-q}$. Note that the case $p = q = 1$ corresponds to sparse MKL and that $p = \infty, q = 2$ corresponds to standard L_2-regularization with a sum-of-kernels.

9.2.3 Learning the Kernel in Structured Prediction

Until Martins et al. (2010e), all proposed formulations of MKL were applied to classification problems with small numbers of classes. The algorithmic obstacles that prevented the straightforward application of those formulations to structured prediction will be discussed in Section 9.2.4; here, we formally present our MKL formulation for structured prediction.

As we saw in Chapter 3 of this thesis, in structured prediction, model factorization assumptions are needed to make the inference problem (3.5) tractable. This can be accomplished by defining a set of parts over which the model decomposes (Taskar et al., 2003). Suppose, for example, that outputs correspond to vertex labelings in a pairwise Markov network $(\mathcal{V}, \mathcal{E})$. Then, we may let each part be either a vertex or an edge, and write the feature vector as $\boldsymbol{f}(x,y) = (\boldsymbol{f}_{\mathcal{V}}(x,y), \boldsymbol{f}_{\mathcal{E}}(x,y))$, with

$$\boldsymbol{f}_{\mathcal{V}}(x,y) = \sum_{i\in\mathcal{V}} \boldsymbol{\psi}_{\mathcal{V}}(x,i) \otimes \boldsymbol{\zeta}_{\mathcal{V}}(y_i), \tag{9.8}$$

$$\boldsymbol{f}_{\mathcal{E}}(x,y) = \sum_{ij\in\mathcal{E}} \boldsymbol{\psi}_{\mathcal{E}}(x,ij) \otimes \boldsymbol{\zeta}_{\mathcal{E}}(y_i,y_j), \tag{9.9}$$

where $\otimes$ denotes the Kronecker product, $\boldsymbol{\psi}_{\mathcal{V}}$ and $\boldsymbol{\psi}_{\mathcal{E}}$ are input feature vectors, and $\boldsymbol{\zeta}_{\mathcal{V}}$ and $\boldsymbol{\zeta}_{\mathcal{E}}$ are *local* output feature vectors which are functions, respectively, of a single vertex and a single edge. Notice that this decomposition into parts does not concern the input: both $\boldsymbol{\psi}_{\mathcal{V}}$ and $\boldsymbol{\psi}_{\mathcal{E}}$ may depend on the entire x. Using the decomposition in (9.8)–(9.9), the compatibility

function defined in (3.5) takes the form

$$\begin{aligned} f_{\boldsymbol{w}}(x,y) &= \langle \boldsymbol{w}, \boldsymbol{f}(x,y)\rangle \\ &= \sum_{i\in\mathcal{V}} \langle \boldsymbol{w}_{\mathcal{V}}, \boldsymbol{\psi}_{\mathcal{V}}(x,i)\otimes\boldsymbol{\zeta}_{\mathcal{V}}(y_i)\rangle + \sum_{ij\in\mathcal{E}} \langle \boldsymbol{w}_{\mathcal{E}}, \boldsymbol{\psi}_{\mathcal{E}}(x,ij)\otimes\boldsymbol{\zeta}_{\mathcal{E}}(y_i,y_j)\rangle \\ &= \sum_{i\in\mathcal{V}} \theta_i(y_i;x,\boldsymbol{w}_{\mathcal{V}}) + \sum_{ij\in\mathcal{E}} \theta_{ij}(y_i,y_j;x,\boldsymbol{w}_{\mathcal{E}}), \end{aligned}$$

where $\theta_i(y_i;x,\boldsymbol{w}_V)$ and $\theta_{ij}(y_i,y_j;x,\boldsymbol{w}_{\mathcal{E}})$ are local scores. The decoding problem (3.5) becomes equivalent to that of finding the MAP configuration in a pairwise Markov random field, which can be done using the methods discussed in Chapter 4.

In terms of kernels, the decomposition in (9.8–9.9) can be written as

$$\begin{aligned} K(u,u') = K((x,y),(x',y')) &= \sum_{i,i'\in\mathcal{V}} K_{\mathcal{V}}((x,i),(x',i'))L_{\mathcal{V}}(y_i,y'_{i'}) \\ &\quad + \sum_{ij,i'j'\in\mathcal{E}} K_{\mathcal{E}}((x,ij),(x',i'j'))L_{\mathcal{E}}((y_i,y_j),(y'_{i'},y'_{j'})), \end{aligned}$$

where

$$\begin{aligned} K_{\mathcal{V}}((x,i),(x',i')) &:= \langle \boldsymbol{\psi}_{\mathcal{V}}(x,i), \boldsymbol{\psi}_{\mathcal{V}}(x',i')\rangle, \\ L_{\mathcal{V}}(y_i,y'_{i'}) &:= \langle \boldsymbol{\zeta}_{\mathcal{V}}(y_i), \boldsymbol{\zeta}_{\mathcal{V}}(y'_{i'})\rangle, \\ K_{\mathcal{E}}((x,ij),(x',i'j')) &:= \langle \boldsymbol{\psi}_{\mathcal{E}}(x,ij), \boldsymbol{\psi}_{\mathcal{E}}(x',i'j')\rangle, \\ L_{\mathcal{E}}((y_i,y_j),(y'_{i'},y'_{j'})) &:= \langle \boldsymbol{\zeta}_{\mathcal{E}}(y_i,y_j), \boldsymbol{\zeta}_{\mathcal{E}}(y'_{i'},y'_{j'})\rangle. \end{aligned} \tag{9.10}$$

A common simplifying option consists in letting $\boldsymbol{\zeta}_{\mathcal{V}}$ and $\boldsymbol{\zeta}_{\mathcal{E}}$ be identity feature mappings (with the former scaled by $\beta_0 > 0$) and $\boldsymbol{\psi}_{\mathcal{E}} \equiv 1$. We can then learn the kernel $K_{\mathcal{V}}$ as a combination of base kernels $\{K_{\mathcal{V},m}\}_{m=1}^{M}$, yielding a kernel decomposition of the form

$$\begin{aligned} K((x,y),(x',y')) &= \beta_0 \cdot |\{ij,i'j' \in \mathcal{E} : y_i = y'_{i'}, y_j = y'_{j'}\}| \\ &\quad + \sum_{i,i'\in\mathcal{V}:\, y_i = y'_{i'}} \sum_{m=1}^{M} \beta_m K_{\mathcal{V},m}((x,i),(x',i')). \end{aligned}$$

In our sequence labeling experiments (Section 9.4), vertices and edges correspond to unigrams and bigrams of labels. We explore two strategies: learning $\beta_1,\ldots,\beta_M$, with $\beta_0 = 1$ fixed, or also learning β_0.

9.2.4 Existing MKL Algorithms

Early approaches to MKL (Lanckriet et al., 2004; Bach et al., 2004) considered the dual of (9.6) in the form of a quadratically constrained quadratic program or of a second order cone program, and thus were limited to small/medium scale problems. Subsequent work focused on scalability: Sonnenburg et al. (2006) proposed a semi-infinite linear programming formulation and a cutting plane algorithm; Rakotomamonjy et al. (2008) proposed a gradient-based method (SimpleMKL) for optimizing the kernel coefficients $\boldsymbol{\beta}$; Chapelle and Rakotomamonjy (2008) proposed to use Newton steps; Kloft et al. (2010) adopted a Gauss-Seidel scheme, al-

ternating between optimization w.r.t. $\boldsymbol{\beta}$ and the SVM instances; Xu et al. (2009) proposed an extended level method.

The methods mentioned in the previous paragraph are all *wrapper-based* algorithms: they repeatedly solve problems of the form (9.3) (or smaller chunks of it, as in Kloft et al. 2010) in an inner loop, while adjusting the kernel coefficients $\boldsymbol{\beta}$ in an outer loop. Although warm-starting may offer considerable speed-ups, convergence relies on the exactness (or prescribed accuracy in the dual) of these solutions, which constitutes a serious obstacle when using such algorithms for structured prediction. As seen in Section 3.5, large-scale solvers for structured SVMs lack strong convergence guarantees; the best methods require $O(\frac{1}{\epsilon})$ rounds to converge to ϵ-accuracy. Sophisticated second-order methods are intractable, since the kernel matrix is exponentially large and hard to invert; furthermore, there are typically many support vectors, since they are indexed by elements of $\mathcal{Y}(x_i)$.

In contrast, we tackle (9.6) in *primal* form. Rather than repeatedly calling off-the-shelf solvers for (9.3), we propose a stand-alone online algorithm with runtime comparable to that of solving a *single* instance of (9.3) by fast online methods. This paradigm shift paves the way for extending MKL to structured prediction, a vast unexplored territory.

9.3 Online Proximal Algorithms

We frame our online MKL algorithm in a wider class of *online proximal algorithms*. The theory of proximity operators (Moreau, 1962), which is widely known in optimization and has recently gained prominence in the signal processing community (Combettes and Wajs, 2006; Wright et al., 2009), provides tools for analyzing these algorithms and generalizes many known results, sometimes with remarkable simplicity. We thus start by summarizing its important concepts in Section 9.3.1.

9.3.1 Proximity Operators and Moreau Projections

We have presented a few basic concepts of convex analysis in Appendix B; we now complement them with some important results concerning direct sums of Hilbert spaces.

Let $\mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_M$ be the direct sum of M Hilbert spaces. Then, if $\varphi : \mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_M \to \bar{\mathbb{R}}$ is block-separable, *i.e.*, $\varphi(\boldsymbol{x}) = \sum_{k=1}^{M} \varphi_k(\boldsymbol{x}_k)$, where $\boldsymbol{x}_k \in \mathcal{H}_k$, it is a known fact that its proximity operator inherits the same block-separability: $[\operatorname{prox}_{\varphi}(\boldsymbol{x})]_k = \operatorname{prox}_{\varphi_k}(\boldsymbol{x}_k)$ (Wright et al., 2009). For example, the proximity operator of the mixed $L_{2,1}$-norm, which is block-separable, has this form. The following proposition extends this result by showing how to compute proximity operators of functions (maybe not separable) that only depend on the L_2-norms of their blocks; *e.g.*, the proximity operator of the *squared* $L_{2,1}$-norm reduces to that of *squared* L_1.

Proposition 9.1 *Let* $\varphi : \mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_M \to \bar{\mathbb{R}}$ *be of the form*

$$\varphi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) = \psi(\|\boldsymbol{x}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{x}_M\|_{\mathcal{H}_M}) \tag{9.11}$$

for some $\psi : \mathbb{R}^M \to \bar{\mathbb{R}}$. *Then,*

$$M_\varphi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) = M_\psi(\|\boldsymbol{x}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{x}_M\|_{\mathcal{H}_M}), \tag{9.12}$$

$$\left[\operatorname{prox}_\varphi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M)\right]_k = [\operatorname{prox}_\psi(\|\boldsymbol{x}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{x}_M\|_{\mathcal{H}_M})]_k(\boldsymbol{x}_k / \|\boldsymbol{x}_k\|_{\mathcal{H}_k}). \tag{9.13}$$

Proof. See Appendix F.1. ∎

Finally, we recall the *Moreau decomposition*, relating the proximity operators of Fenchel conjugate functions (Combettes and Wajs, 2006) and present a corollary, proved in Appendix F.2, that is the key to our regret bound in Section 9.3.4.

Proposition 9.2 (Moreau (1962)) *Let* $\varphi : \mathcal{H} \to \bar{\mathbb{R}}$ *be a convex, lsc, and proper function. Then,*

$$\boldsymbol{x} = \operatorname{prox}_\varphi(\boldsymbol{x}) + \operatorname{prox}_{\varphi^\star}(\boldsymbol{x}), \tag{9.14}$$

$$\|\boldsymbol{x}\|^2 = 2\big(M_\varphi(\boldsymbol{x}) + M_{\varphi^\star}(\boldsymbol{x})\big). \tag{9.15}$$

Corollary 9.3 *Let* $\varphi : \mathcal{H} \to \bar{\mathbb{R}}$ *be as in Proposition 9.2, and* $\bar{\boldsymbol{x}} := \operatorname{prox}_\varphi(\boldsymbol{x})$. *Then, any* $\boldsymbol{y} \in \mathcal{H}$ *satisfies*

$$\frac{\|\boldsymbol{y} - \bar{\boldsymbol{x}}\|^2}{2} - \frac{\|\boldsymbol{y} - \boldsymbol{x}\|^2}{2} + \frac{\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|^2}{2} \leq \varphi(\boldsymbol{y}) - \varphi(\bar{\boldsymbol{x}}). \tag{9.16}$$

Although the Fenchel dual $\varphi^\star$ does not appear in (9.16), it plays a crucial role in proving Corollary 9.3. Later (in Proposition 9.5), we provide a generalization of Corollary 9.3 in which $\bar{\boldsymbol{x}}$ is the outcome of *sequential* proximal steps (rather than a single proximal step). This will be used constructively in Algorithm 14, to be presented next.

9.3.2 An Online Proximal Gradient Scheme

The general algorithmic scheme that we propose and analyze in this paper is presented as Algorithm 14. It deals (in an online fashion[3]) with problems of the form

$$\min_{\boldsymbol{w} \in \mathcal{W}} \Omega(\boldsymbol{w}) + \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{w}; x^i, y^i), \tag{9.17}$$

where $\mathcal{W} \subseteq \mathcal{H}$ is a convex set and the regularizer Ω has a composite form $\Omega(\boldsymbol{w}) = \sum_{j=1}^{J} \Omega_j(\boldsymbol{w})$. This encompasses all formulations described in Section 9.2.1–9.2.2: standard L_2-regularized SVMs and CRFs, group-Lasso, and sparse and non-sparse variants of MKL. For all of these, we have $\mathcal{W} = \mathcal{H}$ and $J = 1$. Moreover, using $J > 1$ allows new variants of block-norm regularization, as discussed in Section 9.3.3. Our only assumption is that each $\operatorname{prox}_{\Omega_j}$ is "simple" (*i.e.*, has closed form or can be cheaply computed), whereas $\operatorname{prox}_\Omega$ may not be.

Algorithm 14 shares with *stochastic gradient descent* (SGD, Bottou 1991b) the fact that it performs "noisy" (sub)gradient steps based on a single instance (x_n, y_n) (lines 5–6), which makes it specially suitable for problems with large N. However, unlike SGD, each round of

[3] For simplicity, we focus on the pure online setting, *i.e.*, each parameter update uses a single observation; analogous algorithms may be derived for the batch and mini-batch cases.

Algorithm 14 Online Proximal Algorithm

1: **input:** data $\mathcal{D}$, number of rounds T, learning rate sequence $(\eta_t)_{t=1,\dots,T}$
2: initialize $\boldsymbol{w}^1 = \mathbf{0}$; set $N = |\mathcal{D}|$
3: **for** $t = 1$ **to** T **do**
4: choose $n = n(t) \in \{1, \dots, N\}$ and take training pair (x^n, y^n)
5: obtain a subgradient $\boldsymbol{g} \in \partial L(\boldsymbol{w}^t; x^n, y^n)$
6: $\tilde{\boldsymbol{w}}^t = \boldsymbol{w}^t - \eta_t \boldsymbol{g}$ (subgradient step)
7: **for** $j = 1$ **to** J **do**
8: $\tilde{\boldsymbol{w}}^{t+\frac{j}{J}} = \mathrm{prox}_{\eta_t \Omega_j}(\tilde{\boldsymbol{w}}^{t+\frac{j-1}{J}})$ (proximal step)
9: **end for**
10: $\boldsymbol{w}^{t+1} = \mathrm{proj}_{\mathcal{W}}(\tilde{\boldsymbol{w}}^{t+1})$ (projection step)
11: **end for**
12: **output:** the last model $\boldsymbol{w}^{T+1}$ or the averaged model $\bar{\boldsymbol{w}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{w}^t$

Algorithm 14 computes subgradients only w.r.t. the loss function L, followed by J proximal steps, one per each term Ω_j (line 7). For sparsity-promoting regularizers, such as $\Omega(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_1$ or $\Omega(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_{2,1}$, SGD and Algorithm 14 behave very differently. Due to the fact that those regularizers are non-differentiable at the origin, SGD is very unlikely to obtain a sparse solution, since the components that should be zero only oscillate asymptotically towards zero. In contrast, since prox_Ω has a thresholding effect, Algorithm 14 is likely to return sparse solutions after a finite number of steps.

As in the Pegasos algorithm (Shalev-Shwartz et al., 2007), the projection step (line 10) is used to accelerate convergence. Even when the original learning problem is unconstrained (*i.e.*, (9.17), with $\mathcal{W} = \mathcal{H}$), if it is known beforehand that the optimum lies in a convex set $\Xi \subseteq \mathcal{H}$, then solving (9.17) with $\mathcal{W} = \Xi$ ensures that each iterate $\boldsymbol{w}_t$ is confined to a bounded region containing the optimum.

Before analyzing Algorithm 14, we remark that it includes, as particular cases, many well-known online learners:

- if $\Omega = 0$ and $\eta_t \propto \frac{1}{\sqrt{t}}$, Algorithm 14 is the online projected subgradient algorithm of Zinkevich (2003);
- if $\Omega = 0$, $L = L_{\mathrm{SSVM}} + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$, and $\eta_t = \frac{1}{\lambda t}$, Algorithm 14 becomes Pegasos, a popular algorithm for learning SVMs that has been extended to structured prediction (Shalev-Shwartz et al., 2007, 2010);
- If $\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$, Algorithm 14 is equivalent to the truncated gradient descent method of Langford et al. (2009);
- If $J = 1$, Algorithm 14 coincides with Fobos (Duchi and Singer, 2009), which was used for learning SVMs and also for group-Lasso (but not for structured prediction).

In Section 9.3.5, we show how to kernelize Algorithm 14 and apply it to sparse MKL. The case $J > 1$ has applications in variants of MKL or group-Lasso with composite regularizers (Tomioka and Suzuki, 2010; Friedman et al., 2010; Bach, 2008b; Zhao et al., 2009). For those cases, Algorithm 14 is seamlessly more suitable than Fobos since it sidesteps the (often difficult) evaluation of prox_Ω, requiring only sequential evaluations of prox_{Ω_j} for $j = 1, \dots, J$.

Algorithm 15 Proximity Operator of L_1^2

1: **input:** vector $\boldsymbol{x} \in \mathbb{R}^M$ and parameter $\lambda > 0$
2: sort the entries of $|\boldsymbol{x}|$ into $\boldsymbol{y}$ (yielding $y_1 \geq \ldots \geq y_M$)
3: set $\rho = \max \left\{ j \in \{1, ..., M\} \,|\, y_j - \frac{\lambda}{1+j\lambda} \sum_{r=1}^{j} y_r > 0 \right\}$
4: **output:** $\boldsymbol{z} = \text{soft}(\mathbf{x}, \tau)$, where $\tau = \frac{\lambda}{1+\rho\lambda} \sum_{r=1}^{\rho} y_r$

9.3.3 Proximity Operators of Block-Norm Regularizers

For Algorithm 14 to handle the MKL and group-Lasso problems (described in Section 9.2.2), it needs to compute the proximal steps for block-norm regularizers. Proposition 9.1 above is crucial for this purpose; as a particular instance of (9.13), computing any $L_{2,q}^r$-proximity operator can be reduced to computing an L_q^r one, the result of which is used to scale each block independently. The following are some examples of block-norm regularizers.

Group-Lasso. This corresponds to $q = r = 1$, so we are left with the problem of computing the L_1-proximity operator, which has a well-known closed form solution: the soft-threshold function, as presented in Section 9.3.1.

Sparse MKL. This corresponds to $q = 1$, $r = 2$, and there are two options: one is to transform the problem back into group-Lasso, by removing the square from Ω_{MKL}; as shown by Bach (2008a) and pointed out in Section 9.2, these two problems are equivalent in the sense that they have the same regularization path. The other option (that we adopt) is to tackle Ω_{MKL} directly, which makes possible the comparison with other MKL algorithms, for the same values of λ, as reported in Section 9.4. Proposition 9.1 enables reducing the evaluation of a $L_{2,1}^2$-proximity operator to that of L_1^2. However, L_1^2 is not separable (unlike L_1), hence the proximity operator cannot be evaluated coordinate-wise. This apparent difficulty has led some authors (*e.g.*, Suzuki and Tomioka 2009) to adopt the first option. However, despite the non-separability of L_1^2, this proximal step can still be efficiently computed (with $O(M \log M)$ cost), using Algorithm 15, which only requires sorting the group weights.[4] Correctness of this algorithm is shown in Appendix F.8.

Non-Sparse MKL. For the case $q \geq 1$ and $r = 2$, a direct evaluation of the proximity operator of $L_{2,q}^2$ is more involved. It seems advantageous to transform this problem into an equivalent one, which uses a separable $L_{2,q}^q$ regularizer instead (the two problems are also equivalent up to a change in the regularization constant). The resulting proximal step amounts to solving M scalar equations of the form $b - b_0 + \tau q b^{q-1} = 0$, w.r.t. b, which is still valid for $q \geq 2$ (unlike the method described by Kloft et al. 2010). This can be done very efficiently using Newton's method.

Other variants. Many other variants of MKL and group-Lasso can be handled by Algorithm 14, with $J > 1$. For example, the elastic net MKL (Tomioka and Suzuki, 2010) uses a sum of two regularizers, $\frac{\sigma}{2}\|.\|^2 + \frac{1-\sigma}{2}\|.\|_{2,1}^2$. In hierarchical Lasso and group-Lasso with

[4] A similar algorithm was proposed independently by Kowalski and Torrésani (2009) in a different context.

overlaps (Bach, 2008b; Zhao et al., 2009; Jenatton et al., 2009), each feature may appear in more than one group. Algorithm 14 handles these problems seamlessly by enabling a proximal step for each group.[5] Sparse group-Lasso (Friedman et al., 2010) simultaneously promotes group-sparsity and sparsity *within* each group, using a regularizer of the form $\sigma\|.\|_{2,1} + (1-\sigma)\|.\|_1$; Algorithm 14 can handle this regularizer by using two proximal steps, both involving simple soft-thresholding: one at the group level, and another within each group.

9.3.4 Regret, Convergence, and Generalization Bounds

We next show that, for a convex loss L and under standard assumptions, Algorithm 14 converges up to ϵ precision, with high confidence, in $O(1/\epsilon^2)$ iterations. If L or Ω are strongly convex, this bound is improved to $\widetilde{O}(1/\epsilon)$, where $\widetilde{O}$ hides logarithmic terms. Our proofs combine tools of online convex programming (Zinkevich, 2003; Hazan et al., 2007) and classical results about proximity operators (Moreau, 1962). We start by introducing the following important concept.

Definition 9.1 (co-shrinkage property) *We say that a sequence of convex, lsc, and proper regularizers $\Omega_1, \dots, \Omega_J$ is* co-shrinking *if*

$$\forall\, j' < j, \quad \Omega_{j'}(\boldsymbol{w}) \geq \Omega_{j'}(\mathrm{prox}_{\Omega_j}(\boldsymbol{w})),$$

that is, the proximity operator of any Ω_j does not increase the value of any previous $\Omega_{j'}$. Moreover, $\Omega_1, \dots, \Omega_J$ is said to be strongly co-shrinking *if, $\forall \sigma_1, \dots, \sigma_J > 0$, the sequence $\sigma_1\Omega_1, \dots, \sigma_J\Omega_J$ is co-shrinking. Finally, we say that a family of regularizers $\mathcal{R} = \{\Omega_j \mid j \in \mathcal{J}\}$ has the* co-shrinkage property *(resp.* strong co-shrinkage property*) if any pair of elements in $\mathcal{R}$ is co-shrinking (resp. strongly co-shrinking). This implies that any finite sequence in $\mathcal{R}$ is (strongly) co-shrinking.*

We note that most regularizers arising in sparse and non-sparse modelling are strongly co-shrinking, as the next proposition attests. The proof is presented in Appendix F.3.

Proposition 9.4 *Let $\mathcal{H} = \bigoplus_{m=1}^{M} \mathcal{H}_m$ be a block-structured Hilbert space. Let $\mathcal{B} \subseteq \{1, \dots, M\}$ be a family of blocks, and, given $\boldsymbol{w} \in \mathcal{H}$, denote by $\boldsymbol{w}_{\mathcal{B}} := (\boldsymbol{w}_m)_{m \in \mathcal{B}}$. The following set of regularizers has the strong co-shrinkage property:*

$$\mathcal{R}_{\text{norms}} = \left\{ \boldsymbol{w} \mapsto \|\boldsymbol{w}_{\mathcal{B}}\|_{2,p}^{q} \;\middle|\; p, q \geq 1, \mathcal{B} \subseteq \{1, \dots, M\} \right\}. \tag{9.18}$$

A consequence of Proposition 9.4 is that any sequence of regularizers $\Omega_1, \dots, \Omega_J$, each of the form $\Omega_j(\boldsymbol{w}) = \|\boldsymbol{w}_{\mathcal{B}_j}\|_{2,p_j}^{q_j}$, with $p_j, q_j \geq 1$ and $\mathcal{B}_j \subseteq \{1, \dots, M\}$, is strongly co-shrinking. Note that the subsets of blocks $\mathcal{B}_1, \dots, \mathcal{B}_J$ can overlap, *i.e.*, we may have $\mathcal{B}_i \cap \mathcal{B}_j \neq \varnothing$ for some $i \neq j$. An example where this arises is the overlapping group-Lasso regularizer, where such overlaps occur and we have $\mathcal{H}_m = \mathbb{R}$, and $p_j = q_j = 1$ for all j.

[5] Recently, a lot of effort has been placed on ways for computing the proximal step for regularizers with overlapping groups (Liu and Ye, 2010a,b; Mairal et al., 2010). Algorithm 14 suggests an alternative approach: split the regularizer into several non-overlapping parts and apply sequential proximal steps. Although in general $\mathrm{prox}_{\Omega_J} \circ \dots \circ \mathrm{prox}_{\Omega_1} \neq \mathrm{prox}_{\Omega_J \circ \dots \circ \Omega_1}$, Algorithm 14 is still applicable, as we will see in Section 9.3.4.

The next result (proved in Appendix F.4) extends Corollary 9.3 for *compositions* of proximity operators of co-shrinking functions, $\operatorname{prox}_{\Omega_J} \circ \ldots \circ \operatorname{prox}_{\Omega_1}$. It is crucial for proving the correctness of Algorithm 14 when composite regularizers are used.

Proposition 9.5 *Let $\Omega_1, \ldots, \Omega_J : \mathcal{H} \to \bar{\mathbb{R}}$ be convex, lsc, and proper functions which are co-shrinking. Denote $\Omega = \sum_{j=1}^{J} \Omega_j$, and let $\bar{\boldsymbol{w}} := \operatorname{prox}_{\Omega_J} \circ \ldots \circ \operatorname{prox}_{\Omega_1}(\boldsymbol{w})$. Then, any $\boldsymbol{\xi} \in \mathcal{H}$ satisfies*

$$\frac{\|\boldsymbol{\xi} - \bar{\boldsymbol{w}}\|^2}{2} - \frac{\|\boldsymbol{\xi} - \boldsymbol{w}\|^2}{2} + \frac{\|\boldsymbol{w} - \bar{\boldsymbol{w}}\|^2}{2J} \leq \Omega(\boldsymbol{\xi}) - \Omega(\bar{\boldsymbol{w}}). \tag{9.19}$$

Note that the only difference between the bounds in Corollary 9.3 and Proposition 9.5 is that the latter includes a J in the denominator of the last term of the left hand side. This reflects the "approximation" that is being made by taking sequencial proximal steps $\operatorname{prox}_{\Omega_J} \circ \ldots \circ \operatorname{prox}_{\Omega_1}$ rather than a single proximal step $\operatorname{prox}_{\Omega_1 + \ldots + \Omega_J}$.

The key to our regret, convergence, and generalization bounds for Algorithm 14 is the following lemma (proved in Appendix F.5).

Lemma 9.6 *Let L be convex and G-Lipschitz on $\mathcal{W}$, and let $\Omega = \sum_{j=1}^{J} \Omega_j$ satisfy the following conditions:* **(i)** *each Ω_j is convex, lsc, and proper;* **(ii)** *$\Omega_1, \ldots, \Omega_J$ are strongly co-shrinking;* **(iii)** *$\Omega(\boldsymbol{w}) \geq \Omega(\Pi_{\mathcal{W}}(\boldsymbol{w}))$,* i.e., *projecting the argument onto $\mathcal{W}$ does not increase Ω. Then, for any $\widehat{\boldsymbol{w}} \in \mathcal{W}$, at each round t of Algorithm 14,*

$$L(\boldsymbol{w}^t) + \Omega(\boldsymbol{w}^{t+1}) \leq L(\widehat{\boldsymbol{w}}) + \Omega(\widehat{\boldsymbol{w}}) + \epsilon_t, \tag{9.20}$$

where

$$\epsilon_t = \frac{\eta_t}{2} G^2 + \frac{\|\widehat{\boldsymbol{w}} - \boldsymbol{w}^t\|^2 - \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t+1}\|^2}{2\eta_t}. \tag{9.21}$$

If L is σ-strongly convex, this bound can be strengthened by replacing ϵ_t with $\epsilon'_t = \epsilon_t - \frac{\sigma}{2}\|\widehat{\boldsymbol{w}} - \boldsymbol{w}^t\|^2$.

The bound in Lemma 9.6 is tighter[6] than the one derived by Duchi and Singer (2009) for the special case of $J = 1$ (*i.e.*, for Fobos). For the even more special case of $J = 1$ and $\Omega = \|\cdot\|_1$, the bound derived by Langford et al. (2009) matches the one in Lemma 9.6, showing that our lemma extends their bound to a much wider class of problems. Finally, note that the conditions **(i)–(iii)** are not restrictive: they hold whenever the regularizers belong to the wide family in Proposition 9.4.

Let $\boldsymbol{w}^*$ be the best fixed hypothesis, *i.e.*, the one obtained in batch mode from the same observations,

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in \mathcal{W}} \sum_{t=1}^{T} \left(\Omega(\boldsymbol{w}) + L(\boldsymbol{w}; x^{n(t)}, y^{n(t)}) \right), \tag{9.22}$$

and the cumulative regret of an online algorithm be defined as

$$\operatorname{Reg}_T := \sum_{t=1}^{T} \left(\Omega(\boldsymbol{w}^t) + L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) \right) - \sum_{t=1}^{T} \left(\Omega(\boldsymbol{w}^*) + L(\boldsymbol{w}^*; x^{n(t)}, y^{n(t)}) \right). \tag{9.23}$$

We next characterize Algorithm 14 in terms of its cumulative regret.

[6] Instead of the term $\frac{\eta_t}{2} G^2$ in (9.21), their bound has $7\frac{\eta_t}{2} G^2$, as can be seen from their Eq. 9 with $A = 0$ and $\eta_t = \eta_{t+\frac{1}{2}}$.

Proposition 9.7 (regret bounds) *Let the conditions of Lemma 9.6 hold, $\Omega \geq 0$, and $\Omega(\mathbf{0}) = 0$. Then:*

1. *Running Algorithm 14 with fixed learning rate η yields*

$$\mathrm{Reg}_T \leq \frac{\eta T}{2} G^2 + \frac{\|\boldsymbol{w}^*\|^2}{2\eta}. \tag{9.24}$$

   Setting[7] *$\eta = \|\boldsymbol{w}^*\|/(G\sqrt{T})$, yields a sublinear regret of $\|\boldsymbol{w}^*\| G\sqrt{T}$.*

2. *If $\mathcal{W}$ is bounded with diameter F (i.e., $\forall \boldsymbol{w}, \boldsymbol{w}' \in \mathcal{W}$, $\|\boldsymbol{w} - \boldsymbol{w}'\| \leq F$) and the learning rate is $\eta_t = \eta_0/\sqrt{t}$, with arbitrary $\eta_0 > 0$, then,*

$$\mathrm{Reg}_T \quad \leq \quad \left(\frac{F^2}{2\eta_0} + G^2\eta_0\right)\sqrt{T}. \tag{9.25}$$

   Setting $\eta_0 = F/(\sqrt{2}G)$, yields $\mathrm{Reg}_T \leq FG\sqrt{2T}$.

3. *If L is σ-strongly convex, and $\eta_t = 1/(\sigma t)$, then*

$$\mathrm{Reg}_T \leq G^2(1 + \log T)/(2\sigma). \tag{9.26}$$

Proof. See Appendix F.6. ∎

Once an online-to-batch conversion is specified, regret bounds allow obtaining PAC bounds on optimization and generalization errors. The following proposition can be proved using the techniques of Cesa-Bianchi et al. (2004) and Shalev-Shwartz et al. (2007).

Proposition 9.8 (optimization and estimation error) *If the assumptions of Proposition 9.7 hold and $\eta_t = \eta_0/\sqrt{t}$ as in item 2 in Proposition 9.7, then the version of Algorithm 14 that returns the averaged model solves (9.17) with ϵ-accuracy*[8] *in $T = O((F^2G^2 + \log(1/\delta))/\epsilon^2)$ iterations with probability at least $1 - \delta$. If L is also σ-strongly convex and $\eta_t = 1/(\sigma t)$ as in item 3 of Proposition 9.7, then, for the version of Algorithm 14 that returns $\boldsymbol{w}^{T+1}$, we get $T = \widetilde{O}(G^2/(\sigma\delta\epsilon))$. The generalization bounds are of the same orders.*

We now pause to examine some concrete cases. The requirement that the loss is G-Lipschitz holds for the hinge and logistic losses, where $G = 2\max_{u\in\mathcal{U}} \|\boldsymbol{f}(u)\|$ (see Appendix F.7), as well as the family of loss functions presented in Chapter 8. These losses are not strongly convex, and therefore Algorithm 14 has only $O(1/\epsilon^2)$ convergence. If the *regularizer* Ω is σ-strongly convex, a possible workaround to obtain $\widetilde{O}(1/\epsilon)$ convergence is to let L "absorb" that strong convexity by redefining $\tilde{L}(\boldsymbol{w}; x^t, y^t) = L(\boldsymbol{w}; x^t, y^t) + \sigma\|\boldsymbol{w}\|^2/2$. Since neither the $L_{2,1}$-norm nor its square are strongly convex, we cannot use this trick for the MKL case, but it *does* apply for non-sparse MKL ($L^2_{2,q}$-norms are strongly convex for $q > 1$) and for elastic MKL. Still, the $O(1/\epsilon^2)$ rate for MKL is competitive with the best batch algorithms that tackle the dual; *e.g.*, the method of Xu et al. (2009) achieves ϵ primal-dual gap in $O(1/\epsilon^2)$ iterations.[9] Some losses of interest (*e.g.*, the squared loss, or the modified

[7] Note that this requires knowing both $\|\boldsymbol{w}^*\|$ and the number of rounds T in advance.

[8] *I.e.*, it returns a feasible solution whose objective value is less than ϵ apart from the optimum.

[9] On the other hand, batch proximal gradient methods for smooth losses can be accelerated to achieve $O(1/\sqrt{\epsilon})$ convergence in the primal objective (Beck and Teboulle, 2009).

Algorithm 16 SPOM

1: **input:** data $\mathcal{D}$, regularizarion constant λ, number of rounds T, radius γ, learning rate sequence $(\eta_t)_{t=1}^T$
2: initialize $\boldsymbol{w}^1 \leftarrow \mathbf{0}$
3: **for** $t = 1$ **to** T **do**
4: choose $n = n(t) \in \{1, \dots, N\}$ and take training pair (x^n, y^n)
5: compute scores for $m = 1, \dots, M$: $f_m(x^n, y) = \langle \boldsymbol{w}_m^t, \boldsymbol{f}_m(x^n, y) \rangle$
6: decode: $\widehat{y}_t \in \arg\max_{y \in \mathcal{Y}(x)} \sum_{m=1}^M f_m(x^n, y) + \rho(y, y^n)$
7: Gradient step for $m = 1, \dots, M$: $\tilde{\boldsymbol{w}}_m^t = \boldsymbol{w}_m^t - \eta_t(\boldsymbol{f}_m(x^n, \widehat{y}^n) - \boldsymbol{f}_m(x^n, y^n))$
8: compute weights for $m = 1, \dots, M$: $\tilde{b}_m^t = \|\tilde{\boldsymbol{w}}_m^t\|$
9: shrink weights $\boldsymbol{b}^t = \mathrm{prox}_{\eta_t \lambda \|\cdot\|_{2,1}^2}(\tilde{\boldsymbol{b}}^t)$ with Algorithm 15
10: Proximal step for $m = 1, \dots, M$: $\tilde{\boldsymbol{w}}_m^{t+1} = b_m^t / \tilde{b}_m^t \cdot \tilde{\boldsymbol{w}}_m^t$
11: Projection step: $\boldsymbol{w}^{t+1} = \tilde{\boldsymbol{w}}^{t+1} \cdot \min\{1, \gamma / \|\tilde{\boldsymbol{w}}^{t+1}\|\}$
12: **end for**
13: compute $\beta_m \propto \|\boldsymbol{w}_m^{T+1}\|$ for $m = 1, \dots, M$
14: return $\boldsymbol{\beta}$, and the last model $\boldsymbol{w}^{T+1}$

loss $\tilde{L}$ above) are G-Lipschitz in any compact subset of $\mathcal{H}$ but not in $\mathcal{H}$. However, if the optimal solution is known to lie in some compact set $\mathcal{W}$, we can run Algorithm 14 with the projection step, making the analysis still applicable.

9.3.5 SPOM: Structured Prediction with Online MKL

The instantiation of Algorithm 14 for structured prediction and $\Omega_{\mathrm{MKL}}(\boldsymbol{w}) = \frac{\lambda}{2}\|\boldsymbol{w}\|_{2,1}^2$ yields the SPOM algorithm (Algorithm 16). We consider $L = L_{\mathrm{SSVM}}$; adapting to any generalized linear model (*e.g.*, $L = L_{\mathrm{CRF}}$) is straightforward. As discussed in the last paragraph of Section 9.3.4, the inclusion of an apparently vacuous projection may accelerate convergence. Hence, an optional upper bound γ on $\|\boldsymbol{w}\|$ is accepted as input. Suitable values of γ for the SVM and CRF cases are given in Appendix F.7.

In line 5, the scores of candidate outputs are computed blockwise; as described in Section 9.2.3, a factorization over parts is assumed and the scores are for partial output assignments. Line 6 gathers all these scores and performs decoding (loss-augmented inference for the SVM case, or marginal inference for the CRF case). Line 10 is where the block structure is taken into account, by applying a proximity operator which corresponds to a blockwise shrinkage/thresolding, with some blocks possibly being set to zero.

Although Algorithm 16 is described with explicit features, it can be kernelized, as shown next. Observe that the parameters of the mth block after round t can be written as $\boldsymbol{w}_m^{t+1} = \sum_{s=1}^t \alpha_{ms}^{t+1}(\boldsymbol{f}_m(x^{n(s)}, y^{n(s)}) - \boldsymbol{f}_m(x^{n(s)}, \hat{y}^{n(s)}))$, where

$$\begin{aligned}
\alpha_{ms}^{t+1} &= \eta_s \prod_{r=s}^{t} \left((b_m^r / \tilde{b}_m^r) \min\{1, \gamma / \|\tilde{\boldsymbol{w}}^{r+1}\|\} \right) \\
&= \begin{cases} \eta_t (b_m^t / \tilde{b}_m^t) \min\{1, \gamma / \|\tilde{\boldsymbol{w}}^{t+1}\|\} & \text{if } s = t \\ \alpha_{ms}^t (b_m^t / \tilde{b}_m^t) \min\{1, \gamma / \|\tilde{\boldsymbol{w}}^{t+1}\|\} & \text{if } s < t. \end{cases}
\end{aligned}$$

Therefore, the inner products in line 5 can be kernelized. The cost of this step is $O(\min\{N, t\})$, instead of the $O(d_m)$ (where d_m is the dimension of the mth block) for the explicit feature

case. After the decoding step (line 6), the supporting pair $(x^t, \widehat{y}^t)$ is stored. Lines 9, 11 and 13 require the *norm* of each group, which can be manipulated using kernels: indeed, after each gradient step (line 7), we have (denoting $u^t = (x^{n(t)}, y^{n(t)})$ and $\widehat{u}^t = (x^{n(t)}, \widehat{y}^{n(t)})$)

$$\begin{aligned} \|\tilde{\boldsymbol{w}}_m^t\|^2 &= \|\boldsymbol{w}_m^t\|^2 - 2\eta_t \langle \boldsymbol{w}_m^t, \boldsymbol{f}_m(u^t)\rangle + \eta_t^2 \|\boldsymbol{f}_m(\widehat{u}^t) - \boldsymbol{f}_m(u^t)\|^2 \\ &= \|\boldsymbol{w}_m^t\|^2 - 2\eta_t f_m(\widehat{u}^t) + \eta_t^2 (K_m(u^t, u^t) + K_m(\widehat{u}^t, \widehat{u}^t) - 2K_m(u^t, \widehat{u}^t)); \end{aligned} \quad (9.27)$$

and the proximal and projection steps merely scale these norms. When the algorithm terminates, it returns the kernel coefficients $\boldsymbol{\beta}$ and the sequence (α_{mt}^{T+1}).

One can also use explicit features in some blocks and implicit features in others. Blocks with explicit features can have their scores computed directly by evaluating the inner product between the feature and the weight vector; this is advantageous when $d_m \ll N$, which happens often when some kernels in the combination are linear. In case of sparse explicit features, an implementation trick analogous to the one used by Shalev-Shwartz et al. (2007) (where each $\boldsymbol{w}_m$ is represented by its norm and an unnormalized vector) can substantially reduce the amount of computation. In the case of implicit features with a sparse kernel matrix, sparse storage of this matrix (which we adopt in our implementation) can also significantly speed up the algorithm, eliminating its dependency on N in line 5. Note that all steps involving block-specific computation can be carried out in parallel using multi-core machines, making Algorithm 16 capable of handling many kernels (large M).

9.4 Experiments

We evaluate SPOM (Algorithm 16) on a structured prediction task, handwriting recognition, formulated as a sequence labeling problem (Section 9.4.1). We then illustrate the robustness of online MKL for handling a particular kind of concept drift in a binary classification problem (Section 9.4.2).

9.4.1 Handwriting Recognition

We formulate the task of recognizing sequences of handwritten characters as a sequence labeling problem, which clearly falls in the structured prediction category. We use the OCR dataset of Taskar et al. (2003), which has a total of 6,877 words and 52,152 characters.[10] Each character (the input) is a 16×8 binary image, with one of 26 labels (the characters `a` to `z`, the output to predict). As Taskar et al. (2003), we address this sequence labeling problem with a structural SVM; however, we use the SPOM algorithm to *learn* the kernel from the data. We use an indicator basis function to represent the correlation between consecutive outputs, *i.e.*, we made the common simplification described in Section 9.2.3 where $L_E((y_i, y_{i+1}), (y'_{i'}, y'_{i'+1})) = \mathbb{I}(y_i = y'_{i'} \wedge y_{i+1} = y'_{i'+1})$ in (9.10).

MKL versus average kernel. Our first experiment (upper part of Table 9.1; solid lines in Figure 9.1) compares linear, quadratic, and Gaussian kernels, either used individually, combined via a simple average, or with MKL (via SPOM). The results show that MKL outperforms the others by $\geq 2\%$, and that learning the bigram weight β_0 (Section 9.2.3) did not

[10] Available at `www.cis.upenn.edu/~taskar/ocr`.

Table 9.1: Results for handwriting recognition. Averages over 10 runs (same folds as Taskar et al. (2003), training on one and testing on the others). The linear and quadratic kernels are normalized to unit diagonal. In all cases, 20 epochs were used, with η_0 in (9.25) picked from $\{0.01, 0.1, 1, 10\}$ by selecting the one that most decreases the objective after 5 epochs. In all cases, the regularization coefficient $C = 1/(\lambda N)$ was chosen with 5-fold cross-validation from $\{0.1, 1, 10, 10^2, 10^3, 10^4\}$.

Kernel	Training Runtimes	Test Acc. (per char.)
Linear (L)	6 sec.	$71.8 \pm 3.9\%$
Quadratic (Q)	116 sec.	$85.5 \pm 0.3\%$
Gaussian (G) ($\sigma^2 = 5$)	123 sec.	$84.1 \pm 0.4\%$
Average $(L + Q + G)/3$	118 sec.	$84.3 \pm 0.3\%$
MKL $\beta_1 L + \beta_2 Q + \beta_3 G$	**279 sec.**	$\mathbf{87.5 \pm 0.3\%}$
MKL $\beta_0, \beta_1 L + \beta_2 Q + \beta_3 G$	**282 sec.**	$\mathbf{87.5 \pm 0.4\%}$
B_1-Spline (B_1)	8 sec.	$75.4 \pm 0.9\%$
Average $(L + B_1)/2$	15 sec.	$83.0 \pm 0.3\%$
MKL $\beta_1 L + \beta_2 B_1$	**15 sec.**	$\mathbf{85.2 \pm 0.3\%}$
MKL $\beta_0, \beta_1 L + \beta_2 B_1$	**16 sec.**	$\mathbf{85.2 \pm 0.3\%}$

make any difference. We note, however, that each epoch of SPOM (without parallelization) is more costly than the the single kernel algorithms. To provide a fair analysis, we show in Figure 9.1 the test set accuracies as a function of the training runtime. We can see that, even without parallelization, the MKL approach achieves an accurate model sooner (with a slight advantage over the quadratic and Gaussian kernels).

Feature and kernel sparsity. The second experiment aims at showing SPOM's ability to exploit both *feature* and *kernel* sparsity. A disadvantage of the Gaussian and quadratic kernels used above is that they yield dense kernel matrices, rendering a quadratic runtime with respect to the sample size. However, we have seen in Section 9.3.5 that SPOM can have a much faster runtime when some kernels are linear or when some kernel matrices are sparse. To illustrate this, we learn a combination of the linear kernel above (explicit features) with a generalized B_1-spline kernel, given by $K(\boldsymbol{x}, \boldsymbol{x}') = \max\{0, 1 - \|\boldsymbol{x} - \boldsymbol{x}'\|/h\}$, with h chosen so that the kernel matrix has approximately $\sim 95\%$ of zeros. The rationale is to combine the strength of a simple feature-based kernel with that of one depending only on a few nearest neighbors. The results (bottom part of Tab. 9.1) show that MKL outperforms by $\sim 10\%$ the individual kernels, and by more than 2% the averaged kernel. Perhaps more importantly, the accuracy is not much worse than the best one obtained in the previous experiment, while the runtime is much faster (15 versus 279 seconds). Figure 9.1 (dashed lines) is striking in showing the ability of producing a reasonable model very fast.

SPOM versus wrapper-based methods. To assess the effectiveness of SPOM as a kernel learning algorithm, we compare it with two wrapper-based MKL algorithms: a Gauss-Seidel method alternating between optimizing the SVM and the kernel coefficients (see, e.g., Kloft et al. 2010), and a gradient method (*SimpleMKL*, Rakotomamonjy et al. 2008)[11]. In both

[11] Code available in http://asi.insa-rouen.fr/enseignants/~arakotom/code/mklindex.html.

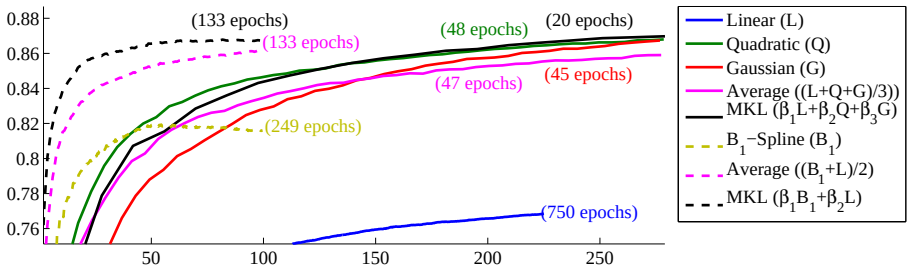


Figure 9.1: Test set accuracies of single kernel and multiple kernel methods as a function of the training stopping times (in seconds).

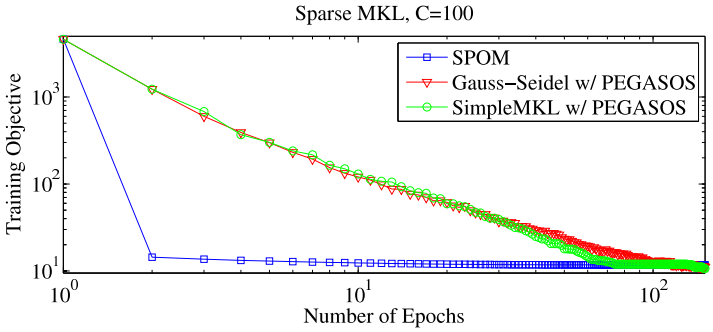


Figure 9.2: Comparison between SPOM (Algorithm 16) and two wrapper based methods in the OCR dataset, with $C = 100$. The wrapper-based methods run 20 epochs of PEGASOS in their first SVM call; subsequent calls run 3 epochs with warm-starting. With only 20–30 passes over the data, SPOM approaches a region very close to the optimum; the wrapper-based methods need about 100 epochs.

cases, the SVMs were optimized with structured PEGASOS. Despite the fact that each SVM is strongly convex and has $O(\frac{1}{\epsilon})$ convergence, its combination with an outer loop becomes time-consuming, even if we warm-start each SVM optimization. This is worse when regularization is weak (small λ). In contrast, SPOM, with its overall $O(\frac{1}{\epsilon^2})$ convergence, is stable and very fast to converge to a near-optimal region, as attested in Figure 9.2, suggesting its usefulness in settings where each epoch is costly.

9.4.2 Online Binary Classification

In our last experiment, we show how the online nature of Algorithm 14 provides an *adaptive* behaviour that is useful in handling a certain kind of drift, by combining the *online learning* paradigm with the ability of *learning the kernel* function. For the scenario we will next describe, online MKL succeeds where standard online learners (*i.e.*, with a fixed kernel) are doomed to fail.

We revisit the "Christmas stars" synthetic problem posed in Sonnenburg et al. (2006): a binary classification task where two concentric star-like shapes are to be distinguished from each other. If the diameters of the two stars are different enough, a Gaussian kernel with an appropriate width performs well on this task. The kernel learning problem consists of

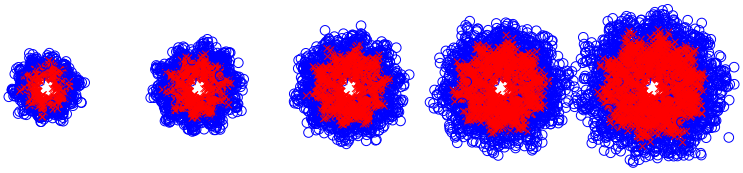


Figure 9.3: The "expanding stars" synthetic dataset, with $N = 5000$ data points, evenly split into two classes; shown are the first 1000, 2000, 3000, 4000, and 5000 points. The two classes are star-shaped and concentric. The diameter of the outermost class increases linearly from 4 to 14; the innermost class has a diameter 1.5 times smaller. The noise is Gaussian for both classes, with standard deviation of 10% of the corresponding star diameters.

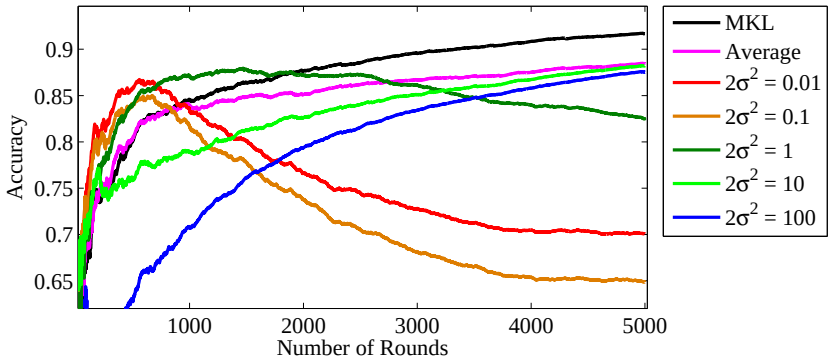


Figure 9.4: Performance of single kernel online learners and a multiple kernel online learner on the "expanding stars" dataset, for Gaussian kernels with different widths. Shown is the fraction of correct predictions up to each round. The regularization parameter $C = 1/(\lambda N)$ was set to 10.

selecting the appropriate width, which is addressed by Sonnenburg et al. (2006) using a batch MKL formulation where several Gaussian kernels of different widths are combined. Here, we present a variant of this problem, in which data is not *i.i.d*. Instead, both stars *expand* as time evolves: their diameters increase linearly with the round number t, while the ratio is kept constant. Figure 9.3 shows the generated data points.

The reason why this problem is challenging is that there is no single Gaussian kernel (with a fixed width) that is suitable for distinguishing between the two stars: the "best" width should increase with t. This is illustrated in Figure 9.4, which shows that there is no single kernel which when plugged into the online algorithm yields good performance over the entire dataset. In contrast, the online MKL algorithm commits fewer mistakes and is able to keep improving its performance as the rounds proceed. This is because it is able to *adapt* its kernel in addition to its decision boundary.

9.5 Related work

Discriminative learning of structured predictors has been an active area of research since the seminal works of Lafferty et al. (2001); Collins (2002a); Altun et al. (2003); Taskar et al. (2003); Tsochantaridis et al. (2004).

Following the introduction of MKL by Lanckriet et al. (2004), a string of increasingly efficient algorithms were proposed (Sonnenburg et al., 2006; Zien and Ong, 2007; Rakotomamonjy et al., 2008; Chapelle and Rakotomamonjy, 2008; Xu et al., 2009; Suzuki and Tomioka, 2009; Kloft et al., 2010), although none was applied to structured prediction. Group-Lasso is due to Bakin (1999); Yuan and Lin (2006), after which many variants and algorithms appeared, all working in batch form: Bach (2008b); Zhao et al. (2009); Jenatton et al. (2009); Friedman et al. (2010).

Independently from us, Jie et al. (2010) recently proposed an online algorithm for multiclass MKL (called OM-2), which differs from ours in that, rather than subgradient and proximal steps, online updates perform coordinate descent in the dual. Our algorithm is more flexible: while OM-2 is limited to $L^2_{2,q}$-regularization, with $q > 1$, and becomes slow when $q \to 1$, we efficiently handle the $L^2_{2,1}$ case as well as arbitrary composite regularizers. Jie et al. (2010) also have not addressed structured prediction.

Proximity operators are well known in convex analysis and optimization (Moreau, 1962; Lions and Mercier, 1979) and have recently seen wide use in signal processing; see Combettes and Wajs (2006), Wright et al. (2009), and references therein. Specifically, the theory of proximity operators (see Appendix F.1) underlies the proofs of our regret bounds (Proposition 9.7).

9.6 Conclusions and Future Work

We proposed a new method for multiple kernel learning of structured predictors. To accomplish this, we introduced a class of online proximal algorithms applicable to many variants of MKL and group-Lasso. We provided a theoretical analysis of its convergence rate, in the form of regret, optimization, and generalization bounds.

Experiments on a structured prediction task (character sequence recognition) have shown

that the algorithm achieves state-of-the-art performance. Moreover, we have illustrated experimentally how the ability to learn the kernel online offers a new paradigm for problems in which the underlying geometry (induced by the similarities between objects) evolves over time.

Our work may impact other problems. In structured prediction, the ability to promote structured sparsity might be useful for learning simultaneously the structure and parameters of graphical models. The online proximal gradient algorithm that we proposed will be used in the next chapter for structured sparsity applications.

In a different direction, the use of sparse kernels redeems one of the disadvantages of kernel methods in large scale problems: the quadratic dependency on the dataset size. Previous approaches to obviate this, such as budgeted learning methods, look inadequate in NLP problems with lexical features, since the feature space grows with the data, and throwing away support vectors implies throwing away features that can be useful. It would be interesting to try combining sparse kernels with linear features in other problems, such as named entity recognition or parsing. The sparse kernel could be used to recognize long matches of input text. This line of research could bring together ideas from memory-based learning and structured linear models.

Chapter 10

Structured Sparsity for Structured Prediction

While a lot of progress has been made in efficient training of linear models with various loss functions, the problem of endowing learners with a mechanism for feature selection is still unsolved. Common approaches employ ad hoc filtering or L_1-regularization; both ignore the structure of the feature space, preventing practicioners from encoding structural prior knowledge. In this chapter, we fill this gap by adopting regularizers that promote *structured sparsity*, along with efficient algorithms to handle them. The following novel contributions are presented:

- We apply group-Lasso regularization to structured prediction problems. We consider several possible choices of groups of features, with or without overlaps.
- We propose using those regularizers for the problem of learning feature templates. We show that this framework is able to model the structure of the feature space through coarse-to-fine regularization.
- For learning using those regularizers, we use the class of online proximal gradient algorithms developed in Chapter 9, which under the perceptron loss become a new algorithm we call *sparseptron*. We enhance the algorithms with delayed proximal steps and a novel budget-driven shrinkage technique, in order to achieve memory and computational efficiency.

Experiments on three tasks (chunking, entity recognition, and dependency parsing) show gains in performance, compactness, and model interpretability.

The results presented in this chapter were originally introduced in Martins et al. (2011d). The online algorithm was introduced in Martins et al. (2011b), along with a thorough convergence analysis, which we have presented in Chapter 9 of this thesis.

10.1 Motivation and Previous Work

As seen in previous chapters, models for structured outputs are in demand across natural language processing, with applications in information extraction, parsing, and machine translation. State-of-the-art models usually involve linear combinations of features and are

trained discriminatively; examples are conditional random fields (Lafferty et al., 2001), structured support vector machines (Altun et al., 2003; Taskar et al., 2003; Tsochantaridis et al., 2004), and the structured perceptron (Collins, 2002a); all of them were reviewed in Sections 3.4–3.5. In all these cases, the underlying optimization problems differ only in the choice of loss function; choosing among them has usually a small impact on predictive performance.

In this chapter, we are concerned with *model selection*: which features should be used to define the prediction score? The fact that models with few features ("sparse" models) are desirable for several reasons (compactness, interpretability, good generalization) has stimulated much research work which has produced a wide variety of methods (Della Pietra et al., 1997; Guyon and Elisseeff, 2003; McCallum, 2003). Our focus is on methods which embed this selection into the learning problem via the regularization term. We depart from previous approaches in that we seek to make decisions jointly about all candidate features, and we want to promote sparsity patterns that go beyond the mere cardinality of the set of features. For example, we want to be able to select entire *feature templates* (rather than features individually), or to make the inclusion of some features depend on the inclusion of other features.

We achieve the goal stated above by employing regularizers which promote *structured sparsity*. Such regularizers are able to encode prior knowledge and guide the selection of features by modeling the structure of the feature space. Lately, this type of regularizers has received a lot of attention in computer vision, signal processing, and computational biology (Zhao et al., 2009; Kim and Xing, 2010; Jenatton et al., 2009; Obozinski et al., 2010; Jenatton et al., 2010; Bach et al., 2011). Eisenstein et al. (2011) employed structured sparsity in computational sociolinguistics. However, none of these works have addressed structured prediction. Here, we combine these two levels of structure: structure in the output space, and structure in the feature space. The result is a framework that allows building structured predictors with high predictive power, while reducing manual feature engineering. We obtain models that are interpretable, accurate, and often much more compact than L_2-regularized ones. Compared with L_1-regularized models, ours are often more accurate and yield faster runtime.

10.2 Sparse Modeling in Structured Prediction

Consider again the learning problem associated with structured linear classifiers, that we have described in Chapter 3:

$$\widehat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}\in\mathbb{R}^D} \Omega(\boldsymbol{w}) + \frac{1}{N}\sum_{n=1}^{N} L(\boldsymbol{w}, \boldsymbol{x}^n, \boldsymbol{y}^n), \tag{10.1}$$

where Ω is a *regularizer* and L is a *loss* function.

In practice, it has been observed that the choice of loss (which we have addressed in Chapter 8) has far less impact than the model design and choice of features. Hence, in this chapter, we focus our attention on the regularization term in Eq. 10.1. We specifically address ways in which this term can be used to help design the model by promoting *structured sparsity*. While this has been a topic of intense research in signal processing and compu-

tational biology (Jenatton et al., 2009; Liu and Ye, 2010b; Bach et al., 2011), it has not yet received much attention in the NLP community, where the choice of regularization for supervised learning has essentially been limited to the following (cf. Section 3.2.1 for details and pointers to the literature):

- L_2-regularization: $\Omega_\lambda^{L_2}(\boldsymbol{w}) := \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$;
- L_1-regularization: $\Omega_\tau^{L_1}(\boldsymbol{w}) := \tau\|\boldsymbol{w}\|_1$,

The latter is known as "Lasso," as popularized by Tibshirani (1996) in the context of sparse regression. In the two cases above, λ and τ are nonnegative coefficients controlling the intensity of the regularization. Each of the regularizers above is appealing for different reasons: $\Omega_\lambda^{L_2}$ usually leads to easier optimization and robust performance; $\Omega_\tau^{L_1}$ encourages sparser models, where only a few features receive nonzero weights; see Gao et al. (2007) for an empirical comparison. More recently, Petrov and Klein (2008b) applied L_1 regularization for structure learning in phrase-based parsing; a comparison with L_2 appears in Petrov and Klein (2008a). Elastic nets interpolate between L_1 and L_2, having been proposed by Zou and Hastie (2005) and used by Lavergne et al. (2010) to regularize CRFs.

Despite their differences, the regularizers mentioned above share a common property: all of them "ignore" the *structure* of the feature space, since they all treat each dimension independently—we call them *unstructured* regularizers, as opposed to the structured ones that we next describe.

10.3 Structured Sparsity: Group-Lasso Regularization

We are interested in regularizers that share with $\Omega_\tau^{L_1}$ the ability to promote sparsity, so that they can be used for selecting features. In addition, we want to endow the feature space $\mathbb{R}^D$ with additional structure, so that features are not penalized individually (as in the L_1-case) but collectively, encouraging entire *groups* of features to be discarded. The choice of groups will allow encoding prior knowledge regarding the kind of sparsity patterns that are intended in the model. This can be achieved with *group-Lasso regularization*, which we next describe.

10.3.1 The Group Lasso

To capture the structure of the feature space, we group our D features into M groups $G_1, \ldots, G_M$, where each $G_m \subseteq \{1, \ldots, D\}$. Ahead, we discuss meaningful ways of choosing group decompositions; for now, let us assume a sensible choice is obvious to the model designer. Denote by $\boldsymbol{w}_m = (w_d)_{d \in G_m}$ the subvector of those weights that correspond to the features in the m-th group, and let $d_1, \ldots, d_M$ be nonnegative scalars (one per group). We consider the following *group-Lasso* regularizers:

$$\Omega_{\boldsymbol{d}}^{\mathrm{GL}} = \sum_{m=1}^{M} d_m \|\boldsymbol{w}_m\|_2. \tag{10.2}$$

These regularizers were first proposed by Bakin (1999) and Yuan and Lin (2006) in the context of regression. If $d_1 = \ldots = d_M$, $\Omega_{\boldsymbol{d}}^{\mathrm{GL}}$ becomes the "L_1 norm of the L_2 norms." Interestingly,

this is also a norm, called the mixed $L_{2,1}$-norm.[1] These regularizers subsume the L_1 and L_2 cases, which correspond to trivial choices of groups:

- If each group is a singleton, *i.e.*, $M = D$ and $G_d = \{w_d\}$, and $d_1 = \ldots = d_M = \tau$, we recover L_1-regularization.

- If there is a single group spanning all the features, *i.e.*, $M = 1$ and $G_1 = \{1, \ldots, D\}$, then the right hand side of Eq. 10.2 becomes $d_1 \|\boldsymbol{w}\|_2$. This is equivalent to L_2 regularization.[2]

We next present some non-trivial examples concerning different topologies of $\mathscr{B} = \{G_1, \ldots, G_M\}$.

Non-overlapping groups. Let us first consider the case where $\mathscr{B}$ is a *partition* of the feature space: the groups cover all the features ($\bigcup_m G_m = \{1, \ldots, D\}$), and they do not overlap ($G_a \cap G_b = \varnothing$, $\forall a \neq b$). Then, $\Omega_{\boldsymbol{d}}^{\text{GL}}$ is termed a *non-overlapping group-Lasso* regularizer. It encourages sparsity patterns in which entire groups are discarded. A judicious choice of groups can lead to very compact models and pinpoint relevant groups of features. The following examples lie in this category:

- The two cases above (L_1 and L_2 regularization).

- *Label-based groups*. In multi-label classification, where $\mathcal{Y} = \{1, \ldots, K\}$, features are typically designed as conjunctions of input features with label indicators, i.e., they take the form $\boldsymbol{f}(x, y) = \boldsymbol{\psi}(x) \otimes \boldsymbol{e}_y$, where $\boldsymbol{\psi}(x) \in \mathbb{R}^{D_X}$, $\boldsymbol{e}_y \in \mathbb{R}^K$ has all entries zero except the y-th entry, which is 1, and $\otimes$ denotes the Kronecker product. Hence $\boldsymbol{f}(x, y)$ can be reshaped as a D_X-by-K matrix, and we can let each group correspond to a row. In this case, all groups have the same size and we typically set $d_1 = \ldots = d_M$. A similar design can be made for sequence labeling problems, by considering a similar grouping for the unigram features.[3]

- *Template-based groups*. In NLP, features are commonly designed via *templates*. For example, a template such as $w_0 \wedge p_0 \wedge p_{-1}$ denotes the word in the current position (w_0) conjoined with its part-of-speech (p_0) and that of the previous word (p_{-1}). This template encloses many features corresponding to different instantiantions of w_0, p_0, and p_{-1}. In Section 10.5, we learn *feature templates* from the data, by associating each group to a feature template, and letting that group contain all features that are instantiations of this template. Since groups have different sizes, it is a good idea to let d_m increase with the group size, so that larger groups pay a larger penalty for being included.

[1] In the statistics literature, such mixed-norm regularizers, which group features and then apply a separate norm for each group, are called *composite absolute penalties* (Zhao et al., 2009); other norms besides $L_{2,1}$ can be used, such as $L_{\infty,1}$ (Quattoni et al., 2009; Wright et al., 2009; Eisenstein et al., 2011).

[2] To be precise, this is not *exactly* the same as the L_2 regularization scenario we have described so far, where the L_2 norm is squared. However it can be shown that both regularizers lead to identical learning problems (Eq. 10.1) up to a transformation of the regularization constant.

[3] The same idea is also used in multitask learning, where labels correspond to tasks (Caruana, 1997); and in multiple kernel learning, where they correspond to kernels—see Chapter 9.

Tree-structured groups. More generally, we may let the groups in $\mathscr{B}$ overlap but be nested, *i.e.*, we may want them to form a *hierarchy* (two distinct groups either have empty intersection or one is contained in the other). This induces a partial order on $\mathscr{B}$ (the set inclusion relation $\supseteq$), endowing it with the structure of a partially ordered set (*poset*).

A convenient graphical representation of the poset $(\mathscr{B}, \supseteq)$ is its *Hasse diagram*. Each group is a node in the diagram, and an arc is drawn from group G_a to group G_b if $G_b \subset G_a$ and there is no b' s.t. $G_b \subset G_{b'} \subset G_a$. When the groups are nested, this diagram is a *forest* (a union of directed trees). The corresponding regularizer enforces sparsity patterns where a group of features is only selected if *all its ancestors are also selected*.[4] Hence, entire subtrees in the diagram can be pruned away. Examples are:

- The *elastic net*. The diagram of $\mathscr{B}$ has a root node for $G_1 = \{1, \dots, D\}$ and D leaf nodes, one per each singleton group (see Figure 10.1).

- The *sparse group-Lasso*. This regularizer was proposed by Friedman et al. (2010):

$$\Omega^{\text{SGL}}_{\boldsymbol{d},\boldsymbol{\tau}}(\boldsymbol{w}) = \sum_{m=1}^{M'} \left(d_m \|\boldsymbol{w}_m\|_2 + \tau_m \|\boldsymbol{w}_m\|_1\right), \tag{10.3}$$

  where the total number of groups is $M = M' + D$, and the components $\boldsymbol{w}_1, \dots, \boldsymbol{w}_{M'}$ are non-overlapping. This regularizer promotes sparsity at both group and feature levels (*i.e.*, it eliminates entire groups and sparsifies within each group).

Graph-structured groups. In general, the groups in $\mathscr{B}$ may overlap without being nested. In this case, the Hasse diagram of $\mathscr{B}$ is a directed acyclic graph (DAG). As in the tree-structured case, a group of features is only selected if all its ancestors are also selected. Based on this property, Jenatton et al. (2009) suggested a way of reverse engineering the groups from the desired sparsity pattern. We next describe a strategy for *coarse-to-fine feature template selection* that directly builds on that idea.

Suppose that we are given M feature templates $\mathscr{T} = \{T_1, \dots, T_M\}$ which are partially ordered according to some criterion, such that if $T_a \preceq T_b$ we would like to include T_b in our model only if T_a is also included. This criterion could be a measure of coarseness: we may want to let coarser part-of-speech features precede finer lexical features, *e.g.*, $p_0 \wedge p_1 \preceq w_0 \wedge w_1$, or conjoined features come after their elementary parts, *e.g.*, $p_0 \preceq p_0 \wedge p_1$. The order does not need to be total, so some templates may not be comparable (*e.g.*, we may want $p_0 \wedge p_{-1}$ and $p_0 \wedge p_1$ not to be comparable). To achieve the sparsity pattern encoded in $(\mathscr{T}, \preceq)$, we choose $\mathscr{B} = (G_1, \dots, G_M)$ as follows: let $I(T_a)$ be the set of features that are instantiations of template T_a; then define $G_a = \bigcup_{b:a \preceq b} I(T_b)$, for $a = 1, \dots, M$. It is easy to see that $(\mathscr{B}, \supseteq)$ and $(\mathscr{T}, \preceq)$ are isomorph posets (their Hasse diagrams have the same shape; see Figure 10.1). The result is a "coarse-to-fine" regularizer, which prefers to select feature templates that are coarser before zooming into finer features.

[4] We say that a group of features G_m is selected if *some* feature in G_m (but not necessarily all) has a nonzero weight.

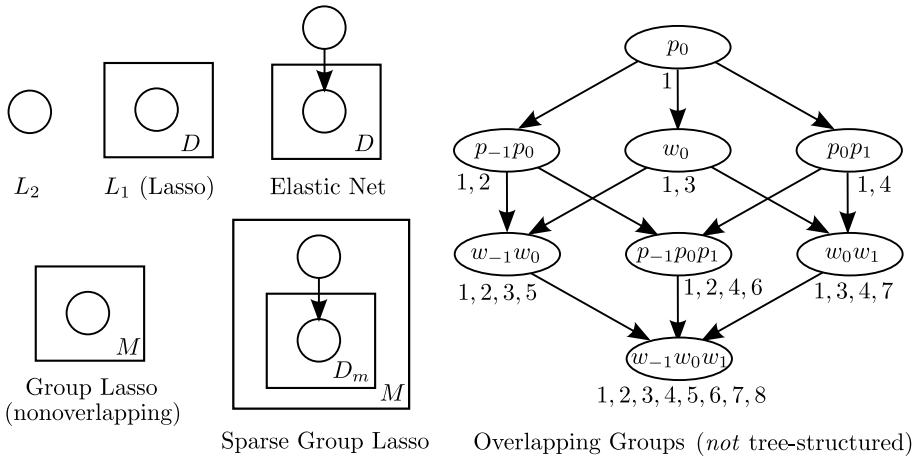


Figure 10.1: Hasse diagrams of several group-based regularizers. For all tree-structured cases, we use the same plate notation that is traditionally used in probabilistic graphical models. The rightmost diagram represents a coarse-to-fine regularizer: each node is a template involving contiguous sequences of words (w) and POS tags (p); the symbol order $\varnothing \preceq p \preceq w$ induces a template order ($T_a \preceq T_b$ iff at each position i $[T_a]_i \preceq [T_b]_i$). Digits below each node are the group indices where each template belongs.

10.3.2 Bayesian Interpretation

The prior knowledge encoded in the group-Lasso regularizer (Eq. 10.2) comes with a Bayesian interpretation, as we next describe. In a probabilistic model (*e.g.*, in the CRF case, where $L = L_{\text{CRF}}$), the optimization problem in Eq. 3.6 can be seen as maximum *a posteriori* estimation of $\boldsymbol{w}$, where the regularization term $\Omega(\boldsymbol{w})$ corresponds to the negative log of a prior distribution (call it $p(\boldsymbol{w})$). It is well-known that L_2-regularization corresponds to choosing independent zero-mean Gaussian priors, $w_d \sim \mathcal{N}(0, \lambda^{-1})$, and that L_1-regularization results from adopting zero-mean Laplacian priors, $p(w_d) \propto \exp(\tau|w_d|)$.

Figueiredo (2002) provided an alternative interpretation of L_1-regularization in terms of a two-level hierarchical Bayes model, which happens to generalize to the non-overlapping group-Lasso case, where $\Omega = \Omega_{\boldsymbol{d}}^{\text{GL}}$. As in the L_2-case, we also assume that each parameter receives a zero-mean Gaussian prior, but now with a *group-specific variance* τ_m, *i.e.*, $\boldsymbol{w}_m \sim \mathcal{N}(\boldsymbol{0}, \tau_m \boldsymbol{I})$ for $m = 1, \ldots, M$. This reflects the fact that some groups should have their feature weights shrunk more towards zero than others. The variances $\tau_m \geq 0$ are not pre-specified but rather generated by a one-sided exponential hyperprior $p(\tau_m|d_m) \propto \exp(-d_m^2 \tau_m/2)$. It can be shown that after marginalizing out τ_m, we obtain

$$\begin{aligned} p(\boldsymbol{w}_m|d_m) &= \int_0^\infty p(\boldsymbol{w}_m|\tau_m) p(\tau_m|d_m) d\tau_m \\ &\propto \exp\left(-d_m \|\boldsymbol{w}_m\|\right). \end{aligned} \tag{10.4}$$

Hence, the non-overlapping group-Lasso corresponds to the following two-level hierachical Bayes model: independently for each $m = 1, \ldots, M$,

$$\tau_m \sim \text{Exp}(d_m^2/2), \qquad \boldsymbol{w}_m \sim \mathcal{N}(0, \tau_m \boldsymbol{I}). \tag{10.5}$$

10.3.3 Prox-operators

Before introducing our learning algorithm for handling group-Lasso regularization, we recall the concept of a Ω-*proximity operator*, defined in Appendix B. This is the function $\text{prox}_\Omega : \mathbb{R}^D \to \mathbb{R}^D$ defined as follows:

$$\text{prox}_\Omega(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}'} \frac{1}{2}\|\boldsymbol{w}' - \boldsymbol{w}\|^2 + \Omega(\boldsymbol{w}'). \tag{10.6}$$

Proximity operators generalize Euclidean projections and have many interesting properties; see Bach et al. (2011) for an overview. By requiring zero to be a subgradient of the objective function in Eq. 10.6, we obtain the following closed expression (called *soft-thresholding*) for the $\Omega_\tau^{L_1}$-proximity operator:

$$[\text{prox}_{\Omega_\tau^{L_1}}(\boldsymbol{w})]_d = \begin{cases} w_d - \tau & \text{if } w_d > \tau \\ 0 & \text{if } |w_d| \leq \tau \\ w_d + \tau & \text{if } w_d < -\tau. \end{cases} \tag{10.7}$$

For the *non-overlapping* group Lasso case, the proximity operator is given by

$$[\text{prox}_{\Omega_d^{\text{GL}}}(\boldsymbol{w})]_m = \begin{cases} \mathbf{0} & \text{if } \|\boldsymbol{w}_m\|_2 \leq d_m \\ \frac{\|\boldsymbol{w}_m\|_2 - d_m}{\|\boldsymbol{w}_m\|_2}\boldsymbol{w}_m & \text{otherwise.} \end{cases} \tag{10.8}$$

which can be seen as a generalization of Eq. 10.7: if the L_2-norm of the m-th group is less than d_m, the entire group is discarded; otherwise it is scaled so that its L_2-norm decreases by an amount of d_m.

When groups overlap, the proximity operator lacks a closed form. When $\mathscr{B}$ is tree-structured, it can still be efficiently computed by a recursive procedure (Jenatton et al., 2010). When $\mathscr{B}$ is *not* tree-structured, no specialized procedure is known, and a convex optimizer is necessary to solve Eq. 10.6.

10.4 Online Proximal Gradient Algorithms

We now turn our attention to efficient ways of handling group-Lasso regularizers. Several fast and scalable algorithms having been proposed for training L_1-regularized CRFs, based on quasi-Newton optimization (Andrew and Gao, 2007), coordinate descent (Sokolovska et al., 2010; Lavergne et al., 2010), and stochastic gradients (Carpenter, 2008; Langford et al., 2009; Tsuruoka et al., 2009). The algorithm that we use in this chapter (Algorithm 17) is a variation of the one introduced in Chapter 9 for multiple kernel learning. It extends the stochastic gradient methods for group-Lasso regularization.

Algorithm 17 addresses the learning problem in Eq. 10.1 by alternating between online (sub-)gradient steps with respect to the loss term, and proximal steps with respect to the regularizer. Proximal-gradient methods are very popular in sparse modeling, both in batch (Liu and Ye, 2010b; Bach et al., 2011) and online (Duchi and Singer, 2009; Xiao, 2009) settings. The reason we have chosen the algorithm proposed in Chapter 9 is that it effectively handles overlapping groups, without the need of evaluating prox_Ω (which, as seen in Section 10.3.3,

Algorithm 17 Online Sparse Prox-Grad Algorithm

input: data $\mathcal{D}$, $(\Omega_j)_{j=1}^J$, number of rounds T, gravity sequence $((\sigma_{jt})_{j=1}^J)_{t=1}^T$, stepsize sequence $(\eta_t)_{t=1}^T$
initialize $\boldsymbol{w} = \mathbf{0}$
for $t = 1$ **to** T **do**
  choose $n = n(t) \in \{1, \ldots, N\}$ and take training pair (x^n, y^n)
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta_t \nabla L(\boldsymbol{\theta}; x^n, y^n)$ (gradient step)
  for $j = 1$ **to** J **do**
    $\boldsymbol{w} = \text{prox}_{\eta_t \sigma_{jt} \Omega_j}(\boldsymbol{w})$ (proximal step)
  end for
end for
output: $\boldsymbol{w}$

can be costly if $\mathscr{B}$ is not tree-structured). To do so, we decompose Ω as

$$\Omega(\boldsymbol{w}) = \sum_{j=1}^{J} \sigma_j \Omega_j(\boldsymbol{w}) \tag{10.9}$$

for some $J \geq 1$, and nonnegative $\sigma_1, \ldots, \sigma_J$; each Ω_j-proximal operator is assumed easy to compute. Such a decomposition always exists: if $\mathscr{B}$ does not have overlapping groups, take $J = 1$. Otherwise, find $J \leq M$ disjoint sets $\mathscr{B}_1, \ldots, \mathscr{B}_J$ such that $\bigcup_{j=1}^J \mathscr{B}_j = \mathscr{B}$ and the groups on each $\mathscr{B}_j$ are non-overlapping. The proximal steps are then applied sequentially, one per each Ω_j. Overall, Algorithm 17 satisfies the following important requirements:

- *Computational efficiency.* Each gradient step at round t is *linear* in the number of features that fire for that instance and *independent* of the total number of features D. Each proximal step is *linear* in the number of groups M, and does not need be to performed every round (as we will see later).

- *Memory efficiency.* Only a small active set of features (those that have nonzero weights) need to be maintained. Entire groups of features can be deleted after each proximal step. Furthermore, only the features which correspond to nonzero entries in the gradient vector need to be inserted in the active set; for some losses (the structured hinge loss L_{SSVM} and the structured perceptron loss L_{SP}) many irrelevant features are never instantianted.

- *Convergence.* With high probability, Algorithm 17 produces an ϵ-accurate solution after $T \leq O(1/\epsilon^2)$ rounds, for a suitable choice of stepsizes and holding σ_{jt} constant, $\sigma_{jt} = \sigma_j$ (Proposition 9.8 in Chapter 9). This result can be generalized to any sequence $(\sigma_{jt})_{t=1}^T$ such that $\sigma_j = \frac{1}{T} \sum_{t=1}^T \sigma_{jt}$.

We next describe several algorithmic ingredients that make Algorithm 17 effective in sparse modeling.

Budget-Driven Shrinkage. Algorithm 17 requires the choice of a "gravity sequence." We follow Langford et al. (2009) and set $(\sigma_{jt})_{j=1}^J$ to zero for all t which is not a multiple of some prespecified integer K; this way, proximal steps need only be performed each K rounds,

yielding a significant speed-up when the number of groups M is large. A direct adoption of the method of Langford et al. (2009) would set $\sigma_{jt} = K\sigma_j$ for those rounds; however, we have observed that such a strategy makes the number of groups vary substantially in early epochs. We use a different strategy: for each $\mathscr{B}_j$, we specify a *budget* of $B_j \geq 0$ groups (this may take into consideration practical limitations, such as the available memory). If t is a multiple of K, we set σ_{jt} as follows:

1. If $\mathscr{B}_j$ does not have more than B_j nonzero groups, set $\sigma_{jt} = 0$ and do nothing.
2. Otherwise, sort the groups in $\mathscr{B}_j$ by decreasing order of their L_2-norms. Check the L_2-norms of the B_j-th and B_{j+1}-th entries in the list and set σ_{jt} as the mean of these two divided by η_t.
3. Apply a $\eta_t\sigma_{jt}\Omega_j$-proximal step using Eq. 10.8. At the end of this step, no more than B_j groups will remain nonzero.[5]

If the average of the gravity steps converge, $\lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} \sigma_{jt} \to \sigma_j$, then the limit points σ_j implicitly define the regularizer, via $\Omega = \sum_{j=1}^{J} \sigma_j\Omega_j$.[6] Hence, we have shifted the control of the amount of regularization to the budget constants B_j, which unlike the σ_j have a clear meaning and can be chosen under practical considerations.

Space and Time Efficiency. The proximal steps in Algorithm 17 have a *scaling* effect on each group, which affects all features belonging to that group (see Eq. 10.8). We want to avoid explicitly updating each feature in the active set, which could be time consuming. We mention two strategies that can be used for the *non-overlapping* group Lasso case.

- The first strategy is suitable when M is large and only a few groups ($\ll M$) have features that fire in each round; this is the case, *e.g.*, of *label-based groups* (see Section 10.3.1). It consists of making *lazy updates* (Carpenter, 2008), *i.e.*, to delay the update of all features in a group until at least one of them fires; then apply a cumulative penalty. The amount of the penalty can be computed if one assigns a timestamp to each group.
- The second strategy is suitable when M is small and some groups are very populated; this is the typical case of *template-based groups* (Section 10.3.1). Two operations need to be performed: updating each feature weight (in the gradient steps), and scaling entire groups (in the proximal steps). We adapt a trick due to Shalev-Shwartz et al. (2007): represent the weight vector of the m-th group, $\boldsymbol{w}_m$, by a triple $(\boldsymbol{\xi}_m, c_m, \rho_m) \in \mathbb{R}^{|G_m|} \times \mathbb{R}_+ \times \mathbb{R}_+$, such that $\boldsymbol{w}_m = c_m\boldsymbol{\xi}_m$ and $\|\boldsymbol{w}_m\|^2 = \rho_m$. This representation allows performing the two operations above in constant time, and it keeps track of the group L_2-norms, necessary in the proximal updates.

For sufficient amounts of regularization, our algorithm has a low memory footprint. Only features that, at some point, intervene in the gradient computed in line 5 need to be instantiated; and all features that receive zero weights after some proximal step can be deleted from the model (cf. Figure 10.2).

[5] When overlaps exist (e.g. the coarse-to-fine case), we specify a total pseudo-budget B ignoring the overlaps, which induces budgets $B_1, \ldots, B_J$ which sum to B. The number of actually selected groups may be less than B, however, since in this case some groups can be shrunk more than once. Other heuristics are possible.

[6] The convergence assumption can be sidestepped by freezing the σ_j after a fixed number of iterations.

	MIRA	Group Lasso (template-based)				
F_1 (%)	93.10	92.99	93.28	93.59	93.42	93.40
model size (# features)	5,300,396	71,075	158,844	389,065	662,018	891,378

Table 10.1: Results for text chunking.

	MIRA	Lasso			Group Lasso (template-based)		
		$C = 0.1$	$C = 0.5$	$C = 1$	$B = 100$	$B = 200$	$B = 300$
Spa. dev/test	70.38/74.09	69.19/71.9	70.75/72.38	71.7/74.03	71.79/73.62	72.08/75.05	71.48/73.3
	8,598,246	68,565	1,017,769	1,555,683	83,036	354,872	600,646
Dut. dev/test	69.15/71.54	64.07/66.35	66.82/69.42	70.43/71.89	69.48/72.83	71.03/73.33	71.2/72.59
	5,727,004	164,960	565,704	953,668	128,320	447,193	889,660
Eng. dev/test	83.95/79.81	80.92/76.95	82.58/78.84	83.38/79.35	85.62/80.26	85.86/81.47	85.03/80.91
	8,376,901	232,865	870,587	1,114,016	255,165	953,178	1,719,229

Table 10.2: Results for named entity recognition. Each cell shows F_1 (%) and the number of features.

Sparseptron and Debiasing. Although Algorithm 17 allows to simultaneously select features and learn the model parameters, it has been observed in the sparse modeling literature that Lasso-like regularizers usually have a strong bias which may harm predictive performance. A post-processing stage is usually taken (called *debiasing*), in which the model is refitted without any regularization and using only the selected features (Wright et al., 2009). If a final debiasing stage is to be performed, Algorithm 17 only needs to worry about feature selection, hence it is appealing to choose a loss function that makes this procedure as simple as possible. Examining the input of Algorithm 17, we see that both a gravity and a stepsize sequence need to be specified. The former can be taken care of by using budget-driven shrinkage, as described above. The stepsize sequence can be set as $\eta_t = \eta_0 / \sqrt{\lceil t/N \rceil}$, which ensures convergence, however η_0 requires tuning. Fortunately, for the structured perceptron loss L_{SP} (Eq. 3.31), Algorithm 17 is independent of η_0, up to a scaling of $\boldsymbol{w}$, which does not affect predictions.[7] We call the instantiation of Algorithm 17 with a group-Lasso regularizer and the loss L_{SP} the *sparseptron*. Overall, we propose the following two-stage approach:

1. Run the sparsepton for a few epochs and discard the features with zero weights.
2. Refit the model without any regularization and using the loss L which one wants to optimize.

10.5 Experiments

We present experiments in three structured prediction tasks for several group choices: text chunking, named entity recognition, and arc-factored dependency parsing (see Chapter 2 for a detailed description of these tasks).

Text Chunking. We use the English dataset provided in the CoNLL 2000 shared task (Sang and Buchholz, 2000), which consists of 8,936 training and 2,012 testing sentences (sections 15–18 and 20 of the WSJ.) The input observations are the token words and their POS tags;

[7] To see why this is the case, note that both gradient and proximal updates come scaled by η_0; and that the gradient of the loss is $\nabla L_{\mathrm{SP}}(\boldsymbol{w}, x, y) = \boldsymbol{f}(x, \hat{y}) - \boldsymbol{f}(x, y)$, where $\hat{y}$ is the prediction under the current model, which is insensitive to the scaling of $\boldsymbol{w}$. This independence on η_0 does not hold when the loss is L_{SSVM} or L_{CRF}.

we want to predict the sequences of `IOB` tags representing phrase chunks. We built 96 contextual feature templates as follows:

- Up to 5-grams of POS tags, in windows of 5 tokens on the left and 5 tokens on the right;
- Up to 3-grams of words, in windows of 3 tokens on the left and 3 tokens on the right;
- Up to 2-grams of word shapes, in windows of 2 tokens on the left and 2 tokens on the right. Each shape replaces characters by their types (case sensitive letters, digits, and punctuation), and deletes repeated types—*e.g.*, `Confidence` and `2,664,098` are respectively mapped to `Aa` and `0,0+,0+` (Collins, 2002b).

We defined unigram features by conjoining these templates with each of the 22 output labels. An additional template was defined to account for label bigrams—features in this template do not look at the input string, but only at consecutive pairs of labels.[8]

We evaluate the ability of group-Lasso regularization to perform *feature template selection*. To do that, we ran 5 epochs of the sparseptron algorithm with template-based groups and budget-driven shrinkage (budgets of 10, 20, 30, 40, and 50 templates were tried). For each group $\mathscr{B}_m$, we set $d_m = \log_2 |G_m|$, which is the average number of bits necessary to encode a feature in that group, if all features were equiprobable. We set $K = 1000$ (the number of instances between consecutive proximal steps). Then, we refit the model with 10 iterations of the max-loss 1-best MIRA algorithm (Crammer et al., 2006).[9] Table 10.1 compares the F_1 scores and the model sizes obtained with the several budgets against those obtained by running 15 iterations of MIRA with the original set of features. Note that the total number of iterations is the same; yet, the group-Lasso approach has a much smaller memory footprint (see Figure 10.2) and yields much more compact models. The small memory footprint comes from the fact that Algorithm 17 may entertain a large number of features without ever instantiating all of them. The predictive power is comparable (although some choices of budget yield slightly better scores for the group-Lasso approach).[10]

Named Entity Recognition. We experiment with the Spanish, Dutch, and English datasets provided in the CoNLL 2002/2003 shared tasks (Sang, 2002; Sang and De Meulder, 2003). For Spanish, we use the POS tags provided by Carreras (`http://www.lsi.upc.es/~nlp/tools/nerc/nerc.html`); for English, we ignore the syntactic chunk tags provided with the dataset. Hence, all datasets have the same sort of input observations (words and POS) and all have 9 output labels. We use the feature templates described above plus some additional ones (yielding a total of 452 templates):

- Up to 3-grams of shapes, in windows of size 3;
- For prefix/suffix sizes of 1, 2, 3, up to 3-grams of word prefixes/suffixes, in windows of size 3;

[8] State-of-the-art models use larger output contexts, such as label trigrams and 4-grams. We resort to bigram labels as we are mostly interested in identifying relevant unigram templates.

[9] This variant optimizes the L_{SSVM} loss, as described in Chapter 8. For the refitting, we used unregularized MIRA. For the baseline (described next), we used L_2-regularized MIRA and tuned the regularization constant with cross-validation.

[10] We also tried label-based group-Lasso and sparse group-Lasso (Section 10.3.1), with less impressive results.

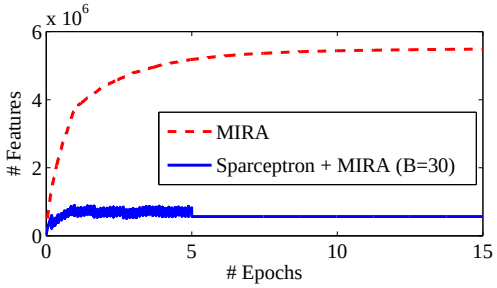


Figure 10.2: Memory footprints of the MIRA and sparseptron algorithms in text chunking. The oscillation in the first 5 epochs (bottom line) comes from the proximal steps each $K = 1000$ rounds. The features are then frozen and 10 epochs of unregularized MIRA follow. Overall, the sparseptron requires $< 7.5\%$ of the memory as the MIRA baseline.

- Up to 5-grams of case, punctuation, and digit indicators, in windows of size 5.

As before, an additional feature template was defined to account for label bigrams. We do feature template selection (same setting as before) for budget sizes of 100, 200, and 300. We compare with both MIRA (using all the features) and the sparseptron with a standard Lasso regularizer $\Omega_\tau^{L_1}$, for several values of $C = 1/(\tau N)$. Table 10.2 shows the results. We observe that template-based group-Lasso wins both in terms of accuracy and compactness. Note also that the ability to discard feature *templates* (rather than individual features) yields faster test runtime than models regularized with the standard Lasso: fewer templates will need to be instantiated, with a speed-up in score computation.

Multilingual Dependency Parsing. We trained non-projective dependency parsers for 6 languages using the CoNLL-X shared task datasets (Buchholz and Marsi, 2006): Arabic, Danish, Dutch, Japanese, Slovene, and Spanish. We chose the languages with the smallest datasets, because regularization is more important when data is scarce. The output to be predicted from each input sentence is the set of dependency links, which jointly define a spanning tree. We use arc-factored models, for which exact inference is tractable (McDonald et al., 2005b). We defined $M = 684$ feature templates for each candidate arc by conjoining the words, shapes, lemmas, and POS of the head and the modifier, as well as the contextual POS, and the distance and direction of attachment. We followed the same two-stage approach as before, and compared with a baseline which selects feature templates by ranking them according to the information gain criterion. This baseline assigns a score to each template T_m which reflects an empirical estimate of the mutual information between T_m and the binary variable A that indicates the presence/absence of a dependency link:

$$IG_m \triangleq \sum_{f \in T_m} \sum_{a \in \{0,1\}} P(f,a) \log_2 \frac{P(f,a)}{P(f)P(a)}, \tag{10.10}$$

where $P(f,a)$ is the joint probability of feature f firing and an arc being active ($a = 1$) or innactive ($a = 0$), and $P(f)$ and $P(a)$ are the corresponding marginals. All probabilities are estimated from the empirical counts of events observed in the data.

The results are plotted in Figure 10.3, for budget sizes of 200, 300, and 400. We observe that for all but one language (Spanish is the exception), non-overlapping group-Lasso regularization is more effective at selecting feature templates than the information gain criterion,

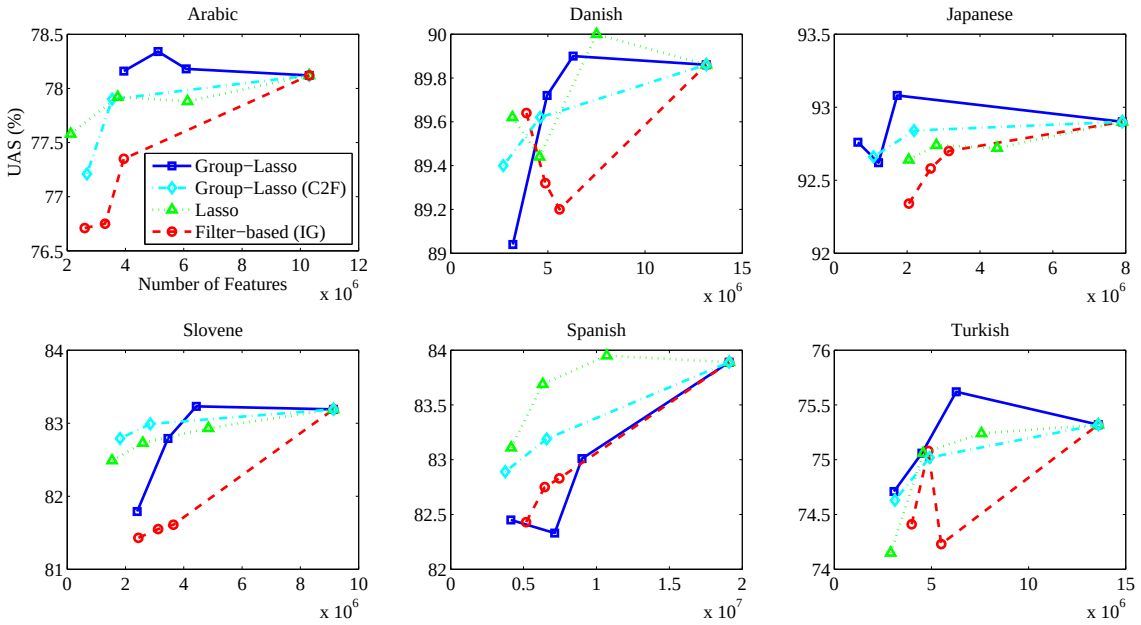


Figure 10.3: Comparison between non-overlapping group-Lasso, coarse-to-fine group-Lasso (C2F), and a filter-based method based on information gain for selecting feature templates in multilingual dependency parsing. The x-axis is the total number of features at different regularization levels, and the y-axis is the unlabeled attachment score. The plots illustrate how accurate the parsers are as a function of the model sparsity achieved, for each method. The standard Lasso (which does not select templates, but individual features) is also shown for comparison.

	Ara.	Dan.	Jap.	Slo.	Spa.	Tur.
Bilexical	++	+			+	
Lex. → POS	+		+			
POS → Lex.	++	+	+		+	+
POS → POS			++	+		
Middle POS	++	++	++	++	++	++
Shape	++	++	++	++		
Direction		+	+	+	+	+
Distance	++	+	+	+	+	+

Table 10.3: Variation of feature templates that were selected accross languages. Each line groups together similar templates, involving lexical, contextual POS, word shape information, as well as attachment direction and length. Empty cells denote that very few or none of the templates in that category was selected; + denotes that some were selected; ++ denotes that most or all were selected.

and slightly better than coarse-to-fine group-Lasso. For completeness, we also display the results obtained with a standard Lasso regularizer. Table 10.3 shows what kind of feature templates were most selected for each language. Some interesting patterns can be observed: morphologically-rich languages with small datasets (such as Turkish and Slovene) seem to avoid lexical features, arguably due to potential for overfitting; in Japanese, contextual POS appear to be specially relevant. It should be noted, however, that some of these patterns may be properties of the datasets rather than of the languages themselves.

10.6 Related Work

The online proximal gradient algorithm used in this chapter was proposed for multiple kernel learning in Martins et al. (2011b), along with a theoretical analysis, which we include in Chapter 9. Budget-driven shrinkage and the sparseptron are novel techniques, to the best of our knowledge. Apart from Martins et al. (2011b), the only work we are aware of which combines structured sparsity with structured prediction is Schmidt and Murphy (2010); however, their goal is to predict the structure of graphical models, while we are mostly interested in the structure of the feature space. Schmidt and Murphy (2010) used generative models, while our approach emphasizes discriminative learning.

Mixed norm regularization has been used for a while in statistics as a means to promote structured sparsity. Group Lasso is due to Bakin (1999) and Yuan and Lin (2006), after which a string of variants and algorithms appeared (Bach, 2008b; Zhao et al., 2009; Jenatton et al., 2009; Friedman et al., 2010; Obozinski et al., 2010). The flat (non-overlapping) case has tight links with learning formalisms such as multiple kernel learning (Lanckriet et al., 2004) and multi-task learning (Caruana, 1997). The tree-structured case has been addressed by Kim and Xing (2010), Liu and Ye (2010b) and Mairal et al. (2010), along with $L_{\infty,1}$ and $L_{2,1}$ regularization. Graph-structured groups are discussed in Jenatton et al. (2010), along with a DAG representation. In NLP, mixed norms have been used recently by Graça et al. (2009) in posterior regularization, and by Eisenstein et al. (2011) in a multi-task regression problem.

10.7 Conclusions and Future Work

In this chapter, we have explored two levels of structure in NLP problems: structure on the outputs, and structure on the feature space. We have shown how the latter can be useful in model design, through the use of regularizers which promote structured sparsity. We propose an online algorithm with minimal memory requirements for exploring large feature spaces. Our algorithm, which specializes into the *sparseptron*, yields a mechanism for selecting entire groups of features. We apply the sparseptron for selecting feature templates in three structured prediction tasks, with advantages over filter-based methods, L_1, and L_2 regularization in terms of performance, compactness, and model interpretability.

The work described in this chapter is just a first incursion in promoting structured sparsity in NLP; there are plenty of avenues for future research. The results that we have obtained for coarse-to-fine regularization are a little bit disappointing, since this strategy is outperformed by non-overlapping groups. However, we cannot draw strong conclusions from this. Our online algorithms, despite their scalabilility and memory efficiency advantages, have slow convergence, and it may happen that the final solution we obtain is not the optimal one—this is worse in the coarse-to-fine regularization case, since the groups overlap, and the proximal step is not exact.

Another important issue are the prior weights $\{d_m\}_{m=1}^M$. These weights play a very important role, and it is not obvious how to set them when groups are very unbalanced. A principled strategy for choosing the weights would be highly desirable. A related topic is that of learning the groups automatically from the data, a problem which has been addressed in some previous work (Huang et al., 2009; Lorbert et al., 2010; Grave et al., 2011), but which is still far from solved. This would be an alternative to the approach taken here, in which the choice of groups is done manually as a way of expressing prior knowledge about the intended sparsity patterns.

Finally, it is worth mentioning that if the problem at hand is not very large, other online or batch algorithms may be a better choice for achieving structured sparsity. For non-overlapping groups, we have observed in preliminary experiments that the regularized dual averaging algorithm proposed by Xiao (2010) is very effective at getting sparse iterates, and it could be an alternative option. Other batch algorithms may also be a good alternative, as they have have excelled in signal processing applications (Bioucas-Dias and Figueiredo, 2007; Wright et al., 2009; Beck and Teboulle, 2009). For overlapping groups, the ADMM algorithm (see Chapter 6 for the use of such algorithm in inference, rather than learning) has also been used with great success (Afonso et al., 2010).

Part IV

Conclusions

Chapter 11

Conclusions

In this chapter, we summarize our contributions and highlight some open problems, suggesting possible directions for future research. Fine-grained discussions about each topic can be found at the end of the corresponding chapters.

11.1 Summary of Contributions

In this thesis, we have advanced the state of the art in structured prediction. Generally speaking, our contributions fall into two realms:

- We have proposed a new methodology for *inference* in structured prediction, in which the design of models is decoupled from algorithmic considerations. This allows constructing rich models that incorporate complex and global features and constraints. We have shown how to handle these models with principled approximate algorithms based on relaxations. We have applied the proposed methodology to an important problem in NLP, dependency parsing, with substantial gains in predictive power.
- We have contributed new methods for *learning* structured predictors in an online manner. We have made progress on two fronts: by proposing *regularizers* that are capable of selecting features by promoting structured sparsity; and by presenting algorithms that deal with various of *loss functions* under a unified framework.

Regarding the inference problem, we have contributed a better theoretical understanding of *constrained structured prediction* (Chapter 5). We provided a formal characterization of constrained graphical models and their geometry, extending known results for unconstrained models. We made these models capable of handling declarative constraints by introducing a simple set of logic factors, and we derived analytical expressions for their messages, marginals, entropies, and marginal polytopes. Our contributions are relevant for several frameworks that have been considered in NLP, such as constrained conditional models (Roth and Yih, 2004) and Markov logic networks (Richardson and Domingos, 2006).

Given a unified treatment of constrained models, we introduced *a new dual decomposition inference algorithm*, called AD^3 (Chapter 6), whose convergence properties we analyzed. The algorithm has the same modular architecture of previous dual decomposition algorithms, such as the ones proposed by Komodakis et al. (2007); Rush et al. (2010): a centralized controller iteratively broadcasts subproblems to workers, and gathers their local solutions to

make a global update. AD^3 is particularly suitable for handling declarative constraints, and we have derived procedures for solving the subproblems associated with all logic factors above. In addition, we have provided an active set method for handling arbitrary factors, requiring only an oracle for computing the local MAP at each factor. Our experiments with benchmark datasets (Ising and Potts models, protein design, and frame semantic parsing) give encouraging results.

We applied the machinery above to an important problem in NLP, *dependency parsing* (Chapter 7). We started by providing a concise ILP formulation based on multi-commodity flows, improving on previous formulations which require an exponential number of constraints (Riedel and Clarke, 2006). Having done this, we opened the door for incorporating rich global features and constraints into the model, which led to a substantial impact in performance. We then relaxed this ILP and showed that the resulting parser is an instance of a *turbo parser*: it performs approximate inference in a loopy graphical model, ignoring global effects caused by the loops. We explicitly derived the underlying graphical model and showed that other parsers Smith and Eisner (2008); Koo et al. (2010) are also turbo parsers.

Regarding the learning problem, we started by considering a wide family of loss functions, which generalizes CRFs, SVMs, and the loss underlying the structured perceptron algorithm (Chapter 8). We presented a new family of online algorithms that can be seen as *dual coordinate ascent* algorithms, and that generalize the MIRA algorithm (Crammer and Singer, 2003; Crammer et al., 2006) to other loss functions, including CRFs. The resulting algorithm is similar to online and stochastic gradient descent, but it does not require specifying a learning rate hyperparameter.

We then turned to the *regularizer* in the learning problem (Chapters 9–10). We proposed new *online proximal-gradient algorithms* that can gracefully handle block structured regularizers, providing their regret and convergence properties. We use those regularizers in the context of structured prediction, to learn combinations of multiple kernels (Chapter 9), and to identify relevant feature templates (Chapter 10), in which we take into account the structure of the feature space and the kind of sparsity patterns that are desired. Our algorithms are able to explore large feature spaces with fast runtime and minimal memory requirements.

11.2 Future Work

There are many possible avenues for future research in the scope of this thesis. We outline just a few.

11.2.1 Broader Constrained Formalisms and Better Entropy Approximations

We linked constrained structured prediction to constrained factor graphs, but in fact it could be useful to consider broader formalisms, such as case-factor diagrams (McAllester et al., 2008), AND/OR search spaces (Dechter and Mateescu, 2007), and sum-product networks (Poon and Domingos, 2011). Not much is known about the geometry underlying these formalisms, and it is an open problem to obtain duality and variational characterizations of the partition functions and entropies in these broader classes. In the same line, our "negative" result in Chapter 5 concerning the poor quality of Bethe entropy approximations in some

constrained graphical models puts a demand for better entropy approximations, another open problem whose study has mostly been focused on the unconstrained case (Wiegerinck and Heskes, 2003; Wainwright et al., 2005b; Weiss et al., 2007; Hazan and Shashua, 2010). The connection mentioned in Section 5.6 with the posterior regularization framework of Ganchev et al. (2010) is also potentially useful.

11.2.2 Applying AD^3 to Other Combinatorial Problems

So far, the AD^3 algorithm has been applied to two combinatorial NLP problems, dependency parsing (Chapter 7) and frame-semantic parsing (Section 6.7; see more details in Das 2012). However, there is a wide universe of NLP problems involving logical constraints for which AD^3 looks useful, such as compositional semantics (Carpenter, 1997; Liang et al., 2011), coreference resolution (Denis and Baldridge, 2007), syntax-based and phrase-based machine translation (Rush and Collins, 2011; Chang and Collins, 2011) and summarization (Clarke and Lapata, 2008; Martins and Smith, 2009; Berg-Kirkpatrick et al., 2011), to name just a few. The last also involves budget constraints, which AD^3 never addressed before.

A careful empirical analysis of AD^3 on a wide set of problems is desirable. Many add-ons have been incorporated in AD^3 since the algorithm was first proposed in Martins et al. (2011a), and it is likely that new ones might be incorporated in the future. Initially, dense, large and combinatorial factors were seen as a bottleneck, when compared with other dual decomposition and message-passing algorithms. The active set method described in Section 6.5 addresses this issue, enabling the application of AD^3 to any decomposable problem as long as the components have local MAP oracles. It would be interesting to compare AD^3 with the projected subgradient algorithm of Komodakis et al. (2007) and Rush et al. (2010) regarding the number of oracle calls necessary to achieve a desired level of precision.

Other engineering tricks can be used to speed up AD^3, for example through parallelization, as discussed in Section 6.8. Another interesting direction concerns the branch-and-bound search procedure described in Section 6.6. While the successful results in frame-semantic parsing are promising, additional experiments need to be made to ensure the robustness of the method for general problems. It is likely that further engineering can speed up the search, through careful caching and warm-starting of the branch-and-bound iterations. An alternative approach to branch-and-bound is a cutting-plane tightening procedure similar to that in Sontag et al. (2008). It would be interesting to compare these two approaches.

11.2.3 Sparse Structured Prediction

In the context of the AD^3 algorithm, we have devised a procedure for maximizing a linear score regularized by an Euclidean penalty, which pushes for a sparse solution, expressed as a combination of a few outputs—see, *e.g.*, Proposition 6.6, which puts a bound on the support of the solution. This quadratic problem is interesting on its own, since the solution implicitly defines a sparse distribution over the output space. This can be useful for developing *pruning models*, as an alternative to typical approaches based on hard thresholding or K-best lists (Charniak et al., 2006; Weiss and Taskar, 2010). The AD^3 algorithm can be adapted for this kind of problem with minimal changes, since all subproblems are still quadratic.

11.2.4 Turbo Parsers and Other Syntactic Formalisms

Given the ability of AD^3 to deal with global features and constraints, a lot of feature engineering could be done for incorporating better global features and constraints into our parsing models, using expert linguistic knowledge. We suggested some in Section 7.6. It would also be useful to approach turbo parsing using the variant of AD^3 with the active set method, comparing its speed with the dual decomposition parsers of Koo et al. (2010).

It is also possible to consider tighter relaxations for parsing, either polyhedral or semidefinite. Examples are the Sherali-Adams, Lovász-Schrijver and Lasserre hierarchies studied by Sontag (2010). Our branch-and-bound procedure can also be applied. However, our experiments suggest that simple linear relaxations are already quite effective and often tight, so search error does not seem to be an issue, at least for dependency parsing.

Other syntactic formalisms, such as phrase-structure grammars, tree-adjoining grammars, or lexical-functional grammars may also be addressed with similar approaches to the ones described in Chapter 6. The AD^3 algorithm looks particularly suitable for dealing with the kind of logical constraints induced by unification-based grammars.

11.2.5 Learning With Approximate Inference

The impact of approximate inference on the learning problem deserves further study. While some previous work has addressed this issue (Wainwright, 2006; Kulesza and Pereira, 2007; Finley and Joachims, 2008; Martins et al., 2009c), there is still a huge gap between theory and practice. In Martins et al. (2009c), we provided conditions that guarantee algorithmic separability under LP-relaxed inference, however the bounds seem too loose to be useful in practice. On the other hand, most theoretical counter-examples seem to rarely happen in practice. It would be useful to have a theory of learning with approximate inference, establishing necessary conditions for learnability and providing a better theoretical understanding of this problem. It is likely that polyhedral characterizations (Grünbaum, 2003; Schrijver, 2003) and the theory of approximation algorithms (Vazirani, 2001) can be useful in this context.

11.2.6 Structured Sparsity in NLP

Our work on structured sparsity is a first step in modeling the structure of the feature space; however, much is left to be done. Broadly speaking, discovering structure in language has been a long-term goal since the early days of computational linguistics, and there is hope that the current advances in sparse modeling can be the key to accomplish that goal.

In our dissertation, we have focused on online algorithms, and we were particularly concerned about memory efficiency. For small and medium-scale problems, there are batch algorithms with better optimization guarantees that might be more suitable for the task. Examples are SpaRSA (Wright et al., 2009) and FISTA (Beck and Teboulle, 2009), which can be easily extended to tree-structured group-Lasso regularization. These algorithms have a larger memory footprint, but asymptotically have much faster convergence rates—for example, for differentiable loss functions with Lipschitz continuous gradients (such as CRFs), FISTA has an iteration bound of $O(1/\sqrt{\epsilon})$, which is much better than the $O(1/\epsilon^2)$ bound of online algorithms. For non-hierarchical overlapping groups, the ADMM algorithm has

also been used with great success (Afonso et al., 2010). There is also the online regularized dual averaging algorithm by Xiao (2010), which is very effective in practice at getting sparse iterates. It would be interesting to evaluate the performance of such algorithms on NLP tasks.

Another important issue are the prior weights in the penalty term. Currently, setting these weights is a black art, and it would be highly desirable to have a principled strategy that works when groups are unbalanced.

Part V

Appendices

Appendix A

Background Proofs

A.1 Proof of Proposition 2.1

To prove Proposition 2.1, we need a couple of lemmata.

Lemma A.1 *Let y be a dependency tree. If an arc $(h, m) \in y$ is non-projective, then it must be "crossed inward" by some other arc $(h', m') \in y$,* i.e., *$m' \neq h$ is in the span of (h, m) and h' is outside that span (see Figure A.1).*

Proof. If (h, m) is non-projective, then by definition there must be some $k \in \text{span}(h, m)$ such that $h \not\preceq k$. Let $\{(0, k_1), (k_1, k_2), \ldots (k_{J-1}, k_J)\}$ be the path of arcs from the root to $k_J := k$; then h is not touched by any arc in this path, and since 0 lies outside $\text{span}(h, m)$ there must be some k_j with $1 \leq j \leq J$ such that k_j is inside the span and k_{j-1} is not. Hence the arc (k_{j-1}, k_j) crosses (h, m) inward. ■

Note that the converse of Lemma A.1 does not hold: in the tree $\{(0, 1), (4, 2), (1, 3), (1, 4)\}$ the arc $(1, 3)$ is projective, despite the fact that it is crossed inward by $(4, 2)$.

Lemma A.2 *Let y be a dependency tree. If two arcs cross each other, then at least one is non-projective.*

Proof. Let (h, m) and (h', m') be the crossing arcs. There are three scenarios that must be considered (see Figure A.1): (i) both arcs cross outward; (ii) one crosses inward and the other outward; (iii) both cross inward. Let us start with (i) and assume, w.l.o.g., that $m' < h < h' < m$. Suppose that (h, m) is projective; then we must have $h \preceq h'$. Similarly, if we assume that (h', m') is projective; we must have $h' \preceq h$. Thus, if the two arcs are projective we must have $h = h'$, which leads to a contradiction. For case (ii) assume w.l.o.g. that $h < h' < m < m'$. If (h, m) is projective, we must have $h \preceq h'$; if (h', m') is projective, we must have $h' \preceq m$. But since h is the immediate predecessor of m and $h' \neq m$, we must have $h' \preceq h$ as well, which again leads to $h' = h$. Finally, assume for case (iii), w.l.o.g., that that $h < m' < m < h'$. If (h, m) is projective, we must have $h \preceq m'$ and similarly to the previous case this implies $h \preceq h'$. By symmetry, if (h', m') is projective we must have $h' \preceq h$ and we are lead to yet another contradiction. ■

We now prove Proposition 2.1, *i.e.*, that a dependency tree is projective if and only if no arcs cross. From Lemma A.1 we have that any tree which is not projective must have crossing arcs. This proves the direction $\Leftarrow$. To prove the direction $\Rightarrow$, invoke Lemma A.2: any pair of

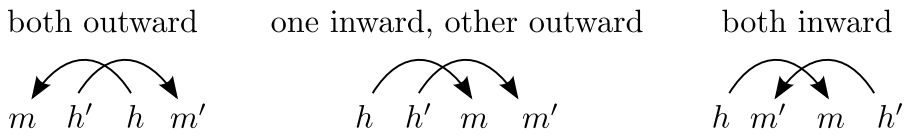


Figure A.1: The three ways arcs can cross each other.

crossing arcs implies the existence of at least one non-projective arc, which renders the tree not projective.

A.2 Derivation of the MPLP Algorithm

Consider the LP-MAP problem of Eq. 4.50. By writing the local polytope inequalities explicitly, and introducing new variables $\zeta_\alpha^i(\boldsymbol{y}_\alpha) := \mu_\alpha(\boldsymbol{y}_\alpha)$, which are copies of the already existing factor marginals $\mu_\alpha(.)$ (one copy per variable $i \in \mathcal{N}(\alpha)$), we can rewrite the LP-MAP problem as:

$$\begin{aligned}
\text{maximize} \quad & \sum_{i\in\mathcal{V}} \sum_{y_i\in\mathcal{Y}_i} \theta_i(y_i)\mu_i(y_i) + \sum_{\alpha\in\mathcal{F}} \sum_{\boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha} \theta_\alpha(\boldsymbol{y}_\alpha)\mu_\alpha(\boldsymbol{y}_\alpha) && \text{(A.1)}\\
\text{w.r.t.} \quad & \boldsymbol{\mu}, \boldsymbol{\zeta} \\
\text{s.t.} \quad & \mu_i(y_i) = \textstyle\sum_{\boldsymbol{y}_\alpha\sim y_i} \zeta_\alpha^i(\boldsymbol{y}_\alpha), \qquad \forall \alpha \in \mathcal{F}, i \in \mathcal{N}(\alpha), y_i \in \mathcal{Y}_i, && \text{(A.2)}\\
& \zeta_\alpha^i(\boldsymbol{y}_\alpha) = \mu_\alpha(\boldsymbol{y}_\alpha), \qquad \forall \alpha \in \mathcal{F}, i \in \mathcal{N}(\alpha), \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha, && \text{(A.3)}\\
& \textstyle\sum_{y_i\in\mathcal{Y}_i} \mu_i(y_i) = 1, \qquad \forall i \in \mathcal{V}, && \text{(A.4)}\\
& \mu_i(y_i) \geq 0, \qquad \forall i \in \mathcal{V}, \forall y_i \in \mathcal{Y}_i, && \text{(A.5)}\\
& \zeta_\alpha^i(\boldsymbol{y}_\alpha) \geq 0, \qquad \forall \alpha \in \mathcal{F}, i \in \mathcal{N}(\alpha), \boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha. && \text{(A.6)}
\end{aligned}$$

We then introduce Lagrange multipliers $\gamma_{\alpha\to i}(y_i)$ and $\delta_\alpha^i(\boldsymbol{y}_\alpha)$ for each of the constraints (A.2) and (A.3). The Lagrangian function becomes:

$$\begin{aligned}
L(\boldsymbol{\mu}, \boldsymbol{\zeta}, \boldsymbol{\gamma}, \boldsymbol{\delta}) \;=\; & \sum_{i\in\mathcal{V}} \sum_{y_i\in\mathcal{Y}_i} \left(\theta_i(y_i) + \sum_{\alpha\in\mathcal{N}(i)} \gamma_{\alpha\to i}(y_i) \right) \mu_i(y_i) \\
& + \sum_{\alpha\in\mathcal{F}} \sum_{\boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha} \left(\theta_\alpha(\boldsymbol{y}_\alpha) - \sum_{i\in\mathcal{N}(\alpha)} \delta_\alpha^i(\boldsymbol{y}_\alpha) \right) \mu_\alpha(\boldsymbol{y}_\alpha) \\
& - \sum_{\alpha\in\mathcal{F}} \sum_{\boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha} \sum_{i\in\mathcal{N}(\alpha)} \left(\gamma_{\alpha\to i}(y_i) - \delta_\alpha^i(\boldsymbol{y}_\alpha) \right) \zeta_\alpha^i(\boldsymbol{y}_\alpha). \qquad \text{(A.7)}
\end{aligned}$$

We want to maximize this function with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\zeta}$, subject to the simplex constraints (A.4–A.5) and the non-negativity constraint A.6, and to minimize it with respect to $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$. Let us take into consideration the following facts: (i) maximizing a linear function over the simplex amounts to picking the largest coordinate; (ii) a maximization of L with respect to $\boldsymbol{\zeta}$ would explode to $+\infty$, unless $\gamma_{\alpha\to i}(y_i) - \delta_\alpha^i(\boldsymbol{y}_\alpha) \geq 0$ holds for every factor α, variable i, and assignment $\boldsymbol{y}_\alpha$; (iii) at optimality, those constraints must be active, so the inequalities can be

replaced by equalities. This originates the constraint

$$\gamma_{\alpha\to i}(y_i) = \max_{\boldsymbol{y}_\alpha \sim y_i} \delta^i_\alpha(\boldsymbol{y}_\alpha). \tag{A.8}$$

Putting these ingredients together, we obtain the following expression for the dual problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i\in\mathcal{V}} \max_{y_i\in\mathcal{Y}_i} \left(\theta_i(y_i)\mu_i(y_i) + \sum_{\alpha\in\mathcal{N}(i)} \max_{\boldsymbol{y}_\alpha\sim y_i} \delta^i_\alpha(\boldsymbol{y}_\alpha) \right) \\ \text{w.r.t.} \quad & \boldsymbol{\delta} \\ \text{s.t.} \quad & \theta_\alpha(\boldsymbol{y}_\alpha) = \textstyle\sum_{i\in\mathcal{N}(\alpha)} \delta^i_\alpha(\boldsymbol{y}_\alpha), \qquad \forall \alpha\in\mathcal{F}, \boldsymbol{y}_\alpha\in\mathcal{Y}_\alpha. \end{aligned} \tag{A.9}$$

This dual is then optimized by a coordinate descent method: at each time, one considers a single factor α and optimizes with respect to $\boldsymbol{\delta}_\alpha := (\delta^i_\alpha(.))_{i\in\mathcal{N}(\alpha)}$, helding the remaining components of $\boldsymbol{\delta}$ fixed. The contribution of those components to the dual objective is the sum $\sum_{\beta\in\mathcal{N}(i)\setminus\{\alpha\}} \max_{\boldsymbol{y}_\beta\sim y_i} \delta^i_\beta(\boldsymbol{y}_\beta)$, which by Eq. A.8 equals $\sum_{\beta\in\mathcal{N}(i)\setminus\{\alpha\}} \gamma_{\beta\to i}(y_i)$. Introducing the variable

$$\delta_{i\to\alpha}(y_i) \quad := \quad \theta_i(y_i) + \sum_{\beta\in\mathcal{N}(i)\setminus\{\alpha\}} \gamma_{\beta\to i}(y_i), \tag{A.10}$$

it can be shown (Globerson and Jaakkola, 2008) that the solution $\widehat{\boldsymbol{\delta}}_\alpha$ of that problem is:

$$\widehat{\delta}^i_\alpha(\boldsymbol{y}_\alpha) = -\delta_{i\to\alpha}(y_i) + \frac{1}{\deg(\alpha)} \left(\theta_\alpha(\boldsymbol{y}_\alpha) + \sum_{j\in\mathcal{N}(\alpha)} \delta_{j\to\alpha}(y_j) \right). \tag{A.11}$$

Hence, a coordinate descent optimization that goes sequentially through each factor $\alpha\in\mathcal{F}$ seeking to optimize $\boldsymbol{\delta}_\alpha$ yields a message-passing scheme in which, for such factor, one first updates the incoming messages through Eq. A.10, and then updates the outgoing messages as follows:

$$\gamma_{\alpha\to i}(y_i) \quad := \quad -\delta_{i\to\alpha}(y_i) + \frac{1}{\deg(\alpha)} \max_{\boldsymbol{y}_\alpha\sim y_i} \left(\theta_\alpha(\boldsymbol{y}_\alpha) + \sum_{j\in\mathcal{N}(\alpha)} \delta_{j\to\alpha}(y_j) \right). \tag{A.12}$$

Appendix B

Convex Analysis and Optimization

In this section, we briefly review some notions of convex analysis that are used throughout the thesis. For more details on this subject, see *e.g.* Rockafellar (1970); Bertsekas et al. (1999); Boyd and Vandenberghe (2004); Bauschke and Combettes (2011).

B.1 Convex Sets, Hulls, and Polytopes

Let $\mathcal{V}$ be a vector space over the field of the real numbers $\mathbb{R}$. We will be mostly concerned with the case where $\mathcal{V}$ is the Euclidean space $\mathbb{R}^D$ or, more generally, a Hilbert space $\mathcal{H}$.

A set $\mathcal{C} \subseteq \mathcal{V}$ is *convex* if, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ and $\lambda \in [0,1]$, we have $\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y} \in \mathcal{C}$. This implies that for any finite set of points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subseteq \mathcal{C}$ and any choice of non-negative coefficients $\lambda_1, \dots, \lambda_N$ such that $\sum_{i=1}^N \lambda_i = 1$, we have $\sum_{i=1}^N \lambda_i \boldsymbol{x}_i \in \mathcal{C}$. The vector $\sum_{i=1}^N \lambda_i \boldsymbol{x}_i$ is called a *convex combination* of $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$.

The *convex hull* of a set $\mathcal{X} \subseteq \mathcal{V}$ is the set of all convex combinations of the elements of $\mathcal{X}$:

$$\operatorname{conv} \mathcal{X} := \left\{ \sum_{i=1}^N \lambda_i \boldsymbol{x}_i \;\middle|\; N \in \mathbb{N},\ \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subseteq \mathcal{X},\ \sum_{i=1}^N \lambda_i = 1,\ \lambda_i \geq 0, \forall i \right\}; \tag{B.1}$$

it is also the smallest convex set that contains $\mathcal{X}$.

The *affine hull* of a set $\mathcal{X} \subseteq \mathcal{V}$ is the set of all *affine combinations* of the elements of $\mathcal{X}$,

$$\operatorname{aff} \mathcal{X} := \left\{ \sum_{i=1}^N \lambda_i \boldsymbol{x}_i \;\middle|\; N \in \mathbb{N},\ \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \subseteq \mathcal{X},\ \sum_{i=1}^N \lambda_i = 1 \right\}; \tag{B.2}$$

it is also the smallest affine set that contains $\mathcal{X}$.

Let $\mathcal{H}$ be a Hilbert space with inner product $\langle ., . \rangle$ and corresponding norm $\|.\|$ given by $\|\boldsymbol{x}\| := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$. Given $\boldsymbol{x} \in \mathcal{H}$, we denote by $\mathcal{B}_\gamma(\boldsymbol{x}) := \{\boldsymbol{y} \in \mathcal{H} \mid \|\boldsymbol{x} - \boldsymbol{y}\| \leq \gamma\}$ the ball with radius γ centered in $\boldsymbol{x}$. The *relative interior* of a set $\mathcal{X} \subseteq \mathcal{H}$ is its interior relative to the affine hull $\mathcal{X}$,

$$\operatorname{relint} \mathcal{X} := \{\boldsymbol{x} \in \mathcal{X} \mid \exists \gamma > 0 \ :\ \mathcal{B}_\gamma(\boldsymbol{x}) \cap \operatorname{aff} \mathcal{X} \subseteq \mathcal{X}\}. \tag{B.3}$$

A *convex polytope* $\mathcal{Z} \subseteq \mathbb{R}^D$ is a set which can be expressed as a convex hull of a finite set of points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\} \in \mathbb{R}^D$. We say that this finite set is *minimal* is no point x_i can be written as a convex combination of the others. If $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ is a minimal set, then its elements

are the *vertices* of $\mathcal{Z}$. A representation of this form, $\mathcal{Z} = \text{conv}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, is called a *vertex representation* of the convex polytope $\mathcal{Z}$.

A *convex polyhedron* is a set $\mathcal{P} \subseteq \mathbb{R}^D$ which can be written as $\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^D \mid \mathbf{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, for some N-by-D matrix $\mathbf{A}$ and vector $\boldsymbol{b} \in \mathbb{R}^N$. Minkowsky-Weyl theorem (see, *e.g.*, Rockafellar 1970) states that a set $\mathcal{P}$ is a polytope if and only if it is a bounded polyhedron.

Given a convex polytope $\mathcal{Z} \subseteq \mathbb{R}^D$, we say that a convex polyhedron $\mathcal{P} \subseteq \mathbb{R}^P$, with $P \geq D$, is a *lifting* or a *lifted version* of $\mathcal{Z}$ if we have $\mathcal{Z} = \{\boldsymbol{z} \in \mathbb{R}^D \mid (\boldsymbol{z}, \boldsymbol{y}) \in \mathcal{P}\}$.

B.2 Convex Functions, Subdifferentials, Proximity Operators, and Moreau Projections

Let $\mathcal{H}$ be a Hilbert space with inner product $\langle ., . \rangle$ and induced norm $\|.\|$. Let $\bar{\mathbb{R}} := \mathbb{R} \cup \{+\infty\}$ be the set of extended reals. Throughout, we let $\varphi : \mathcal{H} \to \bar{\mathbb{R}}$ be a *convex*, *lower semicontinuous* (lsc) and *proper* function—this means, respectively, that:

- The effective domain of φ, $\text{dom}\varphi := \{\boldsymbol{x} \in \mathcal{H} \mid \varphi(\boldsymbol{x}) < +\infty\}$ and the epigraph of φ, $\text{epi}\varphi := \{(\boldsymbol{x}, t) \in \mathcal{H} \times \mathbb{R} \mid \varphi(\boldsymbol{x}) \leq t\}$, are both convex sets;
- $\text{epi}\varphi$ is closed in $\mathcal{H} \times \mathbb{R}$;
- $\text{dom}\varphi \neq \varnothing$.

The *subdifferential* of φ at $\boldsymbol{x}_0$ is the set

$$\partial\varphi(\boldsymbol{x}_0) := \{\boldsymbol{g} \in \mathcal{H} \mid \forall \boldsymbol{x} \in \mathcal{H},\ \varphi(\boldsymbol{x}) - \varphi(\boldsymbol{x}_0) \geq \langle \boldsymbol{g}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle\}, \tag{B.4}$$

the elements of which are the *subgradients*. We say that φ is *G-Lipschitz* in $\mathcal{S} \subseteq \mathcal{H}$ if $\forall \boldsymbol{x} \in \mathcal{S}, \forall \boldsymbol{g} \in \partial\varphi(\boldsymbol{x}), \|\boldsymbol{g}\| \leq G$. We say that φ is *σ-strongly convex* in $\mathcal{S}$ if

$$\forall \boldsymbol{x}_0 \in \mathcal{S}, \quad \forall \boldsymbol{g} \in \partial\varphi(\boldsymbol{x}_0), \quad \forall \boldsymbol{x} \in \mathcal{H}, \quad \varphi(\boldsymbol{x}) \geq \varphi(\boldsymbol{x}_0) + \langle \boldsymbol{g}, \boldsymbol{x} - \boldsymbol{x}_0 \rangle + (\sigma/2)\|\boldsymbol{x} - \boldsymbol{x}_0\|^2. \tag{B.5}$$

The *Fenchel conjugate* of φ, denoted $\varphi^\star : \mathcal{H} \to \bar{\mathbb{R}}$, is defined by

$$\varphi^\star(\boldsymbol{y}) := \sup_{\boldsymbol{x}} \langle \boldsymbol{y}, \boldsymbol{x} \rangle - \varphi(\boldsymbol{x}). \tag{B.6}$$

$\varphi^\star$ is always convex, since it is the supremum of a family of affine functions. Some examples follow:

- If φ is an affine function, $\varphi(\boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{x} \rangle + b$, then $\varphi^\star(\boldsymbol{y}) = -b$ if $\boldsymbol{y} = \boldsymbol{a}$ and $-\infty$ otherwise.
- If $\mathcal{H} = \mathbb{R}^D$ and φ is the ℓ_p-norm, $\varphi(\boldsymbol{x}) = \|\boldsymbol{x}\|_p$, then $\varphi^\star$ is the indicator of the unit ball induced by the dual norm, $\varphi^\star(\boldsymbol{y}) = 0$ if $\|\boldsymbol{y}\|_q \leq 1$ and $+\infty$ otherwise, with $p^{-1} + q^{-1} = 1$.
- If $\mathcal{H} = \mathbb{R}^D$ and φ is half of the squared ℓ_p-norm, $\varphi(\boldsymbol{x}) = \|\boldsymbol{x}\|_p^2/2$, then $\varphi^\star$ is half of the squared dual norm, $\varphi^\star(\boldsymbol{y}) = \|\boldsymbol{y}\|_q^2/2$, with $p^{-1} + q^{-1} = 1$.
- If φ is convex, lsc, and proper, then $\varphi^{\star\star} = \varphi$.

- If $\psi(\boldsymbol{x}) = t\varphi(\boldsymbol{x} - \boldsymbol{x}_0)$, with $t \in \mathbb{R}_+$ and $\boldsymbol{x}_0 \in \mathcal{H}$, then $\psi^\star(\boldsymbol{y}) = \langle \boldsymbol{x}_0, \boldsymbol{y} \rangle + t\varphi^\star(\boldsymbol{y}/t)$.

The *Moreau envelope* of φ, denoted $M_\varphi : \mathcal{H} \to \bar{\mathbb{R}}$, and the *proximity operator* of φ, denoted $\text{prox}_\varphi : \mathcal{H} \to \mathcal{H}$, are defined, respectively, as

$$M_\varphi(\boldsymbol{y}) := \inf_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \varphi(\boldsymbol{x}), \tag{B.7}$$

$$\text{prox}_\varphi(\boldsymbol{y}) := \arg\inf_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \varphi(\boldsymbol{x}). \tag{B.8}$$

Proximity operators generalize Euclidean projectors in the following sense: consider the indicator function $\iota_\mathcal{C}$ of a convex set $\mathcal{C} \subseteq \mathcal{H}$, *i.e.*, $\iota_\mathcal{C}(\boldsymbol{x}) = 0$ if $\boldsymbol{x} \in \mathcal{C}$, and $+\infty$ otherwise. Then, $\text{prox}_{\iota_\mathcal{C}}$ is the Euclidean projector onto $\mathcal{C}$ and $M_{\iota_\mathcal{C}}$ is the residual of the projection. Two other important examples of proximity operators are:

- if $\varphi(\boldsymbol{x}) = (\lambda/2)\|\boldsymbol{x}\|^2$ (with $\lambda \geq 0$) then $\text{prox}_\varphi(\boldsymbol{y}) = \boldsymbol{y}/(1+\lambda)$;
- if $\mathcal{H} = \mathbb{R}^d$ and $\varphi(\boldsymbol{x}) = \tau\|\boldsymbol{x}\|_1$, then $\text{prox}_\varphi(\boldsymbol{y}) = \text{soft}(\boldsymbol{y}, \tau)$ is the *soft-threshold* function (Donoho and Johnstone, 1994), defined as:

$$[\text{soft}(\boldsymbol{y}, \tau)]_k = \text{sgn}(y_k) \cdot \max\{0, |y_k| - \tau\}. \tag{B.9}$$

We terminate by stating *Danskin's theorem*, which allows us to compute subgradients of functions that have a variational representation. A proof can be found in Bertsekas et al. (1999, p.717).

Theorem B.1 (Danskin (1966)) *Let $\mathcal{C} \subseteq \mathbb{R}^D$ be a compact set and $\varphi : \mathbb{R}^K \times \mathcal{C} \to \mathbb{R}$ be continuous and such that $\varphi(., \boldsymbol{x}) : \mathbb{R}^K \to \mathbb{R}$ is convex for every $\boldsymbol{x} \in \mathcal{C}$. Then:*

1. *The function $\psi : \mathbb{R}^K \to \mathbb{R}$ given by*

$$\psi(\boldsymbol{y}) := \max_{\boldsymbol{x} \in \mathcal{C}} \varphi(\boldsymbol{y}, \boldsymbol{x}) \tag{B.10}$$

   is convex.

2. *Let $\mathcal{C}(y)$ be the set of maximizing points in Eq. B.10. If $\varphi(\boldsymbol{y}, \boldsymbol{x})$ is continuously differentiable with respect to $\boldsymbol{y}$ for each $\boldsymbol{x} \in \mathcal{C}$, then the subdifferential of ψ is given by:*

$$\partial\psi(\boldsymbol{y}) = \text{conv}\left\{ \frac{\partial\varphi(\boldsymbol{y}, \boldsymbol{x})}{\partial \boldsymbol{y}} \;\middle|\; \boldsymbol{x} \in \mathcal{C}(y) \right\}. \tag{B.11}$$

Appendix C

Derivation of Messages for Logic Factors

C.1 Sum-Product Messages

The One-hot XOR Factor. For the partition function, we need to sum over all the allowed configurations (*i.e.*, those in the acceptance set $\mathcal{S}_{\text{XOR}}$). This can be done explicitly for the XOR factor, since the cardinality of this set is linear in K. We obtain:

$$\begin{aligned} Z_{\text{XOR}}(\omega) &= \sum_{\boldsymbol{y}\in\mathcal{S}_{\text{XOR}}} \prod_{k=1}^{K} M_{k\to\text{XOR}}(y_k) \\ &= \sum_{i=1}^{K} M_{i\to\text{XOR}}(1) \prod_{k=1,k\neq i}^{K} M_{k\to\text{XOR}}(0) \\ &= \left(\sum_{i=1}^{K} \frac{M_{i\to\text{XOR}}(1)}{M_{i\to\text{XOR}}(0)}\right) \times \prod_{k=1}^{K} M_{k\to\text{XOR}}(0) \\ &= \sum_{i=1}^{K} m_{i\to\text{XOR}} \times \prod_{i=1}^{K} (1+m_{i\to\text{XOR}})^{-1}. \end{aligned} \tag{C.1}$$

Expressions for the marginals can be derived similarly. We obtain:

$$\begin{aligned} \mu_i(1;\omega) &= Z_{\text{XOR}}(\omega)^{-1} \times \sum_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{XOR}} \\ y_i=1}} \prod_{k=1}^{K} M_{k\to\text{XOR}}(y_k) \\ &= Z_{\text{XOR}}(\omega)^{-1} \times M_{i\to\text{XOR}}(1) \prod_{k=1,k\neq i}^{K} M_{k\to\text{XOR}}(0) \\ &= Z_{\text{XOR}}(\omega)^{-1} \times m_{i\to\text{XOR}} \prod_{k=1}^{K} (1+m_{k\to\text{XOR}})^{-1} \\ &= \frac{m_{i\to\text{XOR}}}{\sum_{k=1}^{K} m_{k\to\text{XOR}}} \end{aligned} \tag{C.2}$$

and

$$\mu_i(0;\omega) \;=\; 1-\mu_i(1;\omega) \;=\; \frac{\sum_{k=1,k\neq i}^{K} m_{k\to \mathrm{XOR}}}{\sum_{k=1}^{K} m_{k\to \mathrm{XOR}}}, \tag{C.3}$$

so the marginal ratio is:

$$\frac{\mu_i(1;\omega)}{\mu_i(0;\omega)} = \frac{m_{i\to \mathrm{XOR}}}{\sum_{k=1,k\neq i}^{K} m_{k\to \mathrm{XOR}}}. \tag{C.4}$$

From Proposition 5.4, we have

$$m_{\mathrm{XOR}\to i} \;=\; \frac{M_{\mathrm{XOR}\to i}(1)}{M_{\mathrm{XOR}\to i}(0)} \;=\; m_{k\to \mathrm{XOR}}^{-1} \times \frac{\mu_i(1;\omega)}{\mu_i(0;\omega)}, \tag{C.5}$$

hence

$$\boxed{m_{\mathrm{XOR}\to i} = \left(\sum_{k=1,k\neq i}^{K} m_{k\to \mathrm{XOR}}\right)^{-1}}. \tag{C.6}$$

Finally, the entropy is given by:

$$\begin{aligned}
H_{\mathrm{XOR}}(b_{\mathrm{XOR}}(.)) &= \log Z_{\mathrm{XOR}}(\omega) - \sum_{i=1}^{K}\sum_{y_i\in\{0,1\}} \mu_i(y_i;\omega)\log M_{i\to \mathrm{XOR}}(y_i) \\
&= \log Z_{\mathrm{XOR}}(\omega) - \sum_{i=1}^{K} \mu_i(1;\omega)\log m_{i\to \mathrm{XOR}} + \sum_{i=1}^{K}\log(1+m_{i\to \mathrm{XOR}}) \\
&= \log\left(\sum_{i=1}^{K} m_{i\to \mathrm{XOR}}\right) - \sum_{i=1}^{K} \mu_i(1;\omega)\log m_{i\to \mathrm{XOR}} \\
&= \log\left(\sum_{i=1}^{K} m_{i\to \mathrm{XOR}}\right) - \sum_{i=1}^{K}\left(\frac{m_{i\to \mathrm{XOR}}}{\sum_{k=1}^{K} m_{k\to \mathrm{XOR}}}\right)\log m_{i\to \mathrm{XOR}} \\
&= -\sum_{i=1}^{K}\left(\frac{m_{i\to \mathrm{XOR}}}{\sum_{k=1}^{K} m_{k\to \mathrm{XOR}}}\right)\log\left(\frac{m_{i\to \mathrm{XOR}}}{\sum_{k=1}^{K} m_{k\to \mathrm{XOR}}}\right),
\end{aligned} \tag{C.7}$$

that is,

$$\boxed{H_{\mathrm{XOR}}(b_{\mathrm{XOR}}(.)) = -\sum_{i=1}^{K} b_i(1)\log b_i(1).} \tag{C.8}$$

The OR Factor. We next turn to the OR factor. For the partition function, note that (unlike the XOR case) we cannot efficiently enumerate all the allowed configurations in $\mathcal{S}_{\mathrm{OR}}$, since there are 2^K-1 of them. However, we can write the sum as if it was over all 2^K configurations—which will yield a simple expression— and then discount the all-zeros configuration, which is the one which does not belong to $\mathcal{S}_{\mathrm{OR}}$. More concretely, note that we have:

$$\sum_{\boldsymbol{y}\in\{0,1\}^K}\prod_{k=1}^{K} M_{k\to \mathrm{OR}}(y_k) = \prod_{k=1}^{K}\sum_{y_k\in\{0,1\}} M_{k\to \mathrm{OR}}(y_k), \tag{C.9}$$

which equals 1 if we assume that the incoming messages are normalized—which we can without loss of generality.[1] We thus obtain:

$$\begin{aligned} Z_{\mathrm{OR}}(\omega) &= \sum_{\boldsymbol{y}\in\mathcal{S}_{\mathrm{OR}}} \prod_{k=1}^{K} M_{k\to\mathrm{OR}}(y_k) \\ &= \sum_{y\in\{0,1\}^K} \prod_{k=1}^{K} M_{k\to\mathrm{OR}}(y_k) - \prod_{k=1}^{K} M_{k\to\mathrm{OR}}(0) \\ &= 1 - \prod_{i=1}^{K} (1+m_{i\to\mathrm{OR}})^{-1}. \end{aligned} \tag{C.10}$$

For the marginals, we can follow the same procedure:

$$\begin{aligned} \mu_i(1;\omega) &= Z_{\mathrm{OR}}(\omega)^{-1} \times \sum_{\substack{\boldsymbol{y}\in\mathcal{S}_{\mathrm{OR}} \\ y_i=1}} \prod_{k=1}^{K} M_{k\to\mathrm{OR}}(y_k) \\ &= Z_{\mathrm{OR}}(\omega)^{-1} \times M_{i\to\mathrm{OR}}(1) \times \sum_{\substack{y_1,\dots,y_{i-1} \\ y_{i+1},\dots,y_K}} \prod_{k=1}^{K} M_{k\to\mathrm{OR}}(y_k) \\ &= Z_{\mathrm{OR}}(\omega)^{-1} \times M_{i\to\mathrm{OR}}(1), \end{aligned} \tag{C.11}$$

and $\mu_i(0;\omega) = 1 - \mu_i(1;\omega)$, which yields the following ratio of marginals:

$$\begin{aligned} \frac{\mu_i(1;\omega)}{\mu_i(0;\omega)} &= \frac{Z_{\mathrm{OR}}(\omega)^{-1} \times M_{i\to\mathrm{OR}}(1)}{1 - Z_{\mathrm{OR}}(\omega)^{-1} \times M_{i\to\mathrm{OR}}(1)} \\ &= M_{i\to\mathrm{OR}}(1) \times \left(Z_{\mathrm{OR}}(\omega) - M_{i\to\mathrm{OR}}(1)\right)^{-1} \\ &= m_{i\to\mathrm{OR}}(1+m_{i\to\mathrm{OR}})^{-1} \left(1 - \prod_{k=1}^{K}(1+m_{k\to\mathrm{OR}})^{-1} - m_{i\to\mathrm{OR}}(1+m_{i\to\mathrm{OR}})^{-1}\right)^{-1} \\ &= m_{i\to\mathrm{OR}} \left(1 - \prod_{\substack{k=1 \\ k\neq i}}^{K}(1+m_{k\to\mathrm{OR}})^{-1}\right)^{-1}. \end{aligned} \tag{C.12}$$

From Proposition 5.4, we now have that the message updates are given by:

$$\boxed{m_{\mathrm{OR}\to i} = \left(1 - \prod_{k=1,k\neq i}^{K} (1+m_{k\to\mathrm{OR}})^{-1}\right)^{-1}}. \tag{C.13}$$

[1] Recall that the message update equations (Eqs. 4.7 and 4.8) and the belief equations (Eqs. 4.9 and 4.10) are unaffected by multiplying each $M_{i\to\alpha}$ or $M_{\alpha\to i}$ by a positive constant.

Finally, the entropy is given by:

$$\begin{aligned} H_{\text{OR}}(b_{\text{OR}}(.)) &= \log Z_{\text{OR}}(\omega) - \sum_{i=1}^{K} \sum_{y_i \in \{0,1\}} \mu_i(y_i;\omega) \log M_{i\to\text{OR}}(y_i) \\ &= \log Z_{\text{OR}}(\omega) - \sum_{i=1}^{K} \mu_i(1;\omega) \log m_{i\to\text{OR}} + \sum_{i=1}^{K} \log(1+m_{i\to\text{OR}}) \\ &= \log\left(1 - \prod_{i=1}^{K}(1+m_{i\to\text{OR}})^{-1}\right) \\ &\quad - \sum_{i=1}^{K}\left(1 + m_{i\to\text{OR}} - \prod_{k=1,k\neq i}^{K}(1+m_{k\to\text{OR}})^{-1}\right)^{-1} \times m_{i\to\text{OR}} \log m_{i\to\text{OR}} \\ &\quad + \sum_{i=1}^{K} \log(1+m_{i\to\text{OR}}). \end{aligned} \tag{C.14}$$

The OR-with-output Factor. We next derive expressions for the log-partition function and marginals associated with the OR-with-output factor. As in the case of the OR-factor, we cannot efficiently enumerate all the allowed configurations in $\mathcal{S}_{\text{OR-out}}$, since there are 2^{K-1} of them. However, we can decouple the sum in two parts: one which includes all assignments for which $y_{K+1} = 1$, which can be reduced to the computation of the partition function for the OR factor; and other which considers the case where $y_{K+1} = 0$, whose only allowed configuration is the all-zeros one. Thus, we obtain:

$$\begin{aligned} Z_{\text{OR-out}}(\omega) &= \sum_{\boldsymbol{y}\in\mathcal{S}_{\text{Or-out}}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= M_{K+1\to\text{OR-out}}(1)\left(\sum_{y_1,\cdots,y_K}\prod_{k=1}^{K} M_{k\to\text{OR-out}}(y_k) - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right) + \\ &\quad \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0) \\ &= M_{K+1\to\text{OR-out}}(1)\left(1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right) + \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0). \end{aligned} \tag{C.15}$$

The marginals can be computed similarly. We consider first the case where $i \leq K$:

$$\begin{aligned} \mu_i(1;\omega) &= Z_{\text{OR-out}}(\omega)^{-1} \times \sum_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{or-out}} \\ y_i=1}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= Z_{\text{OR-out}}(\omega)^{-1} \times M_{i\to\text{OR-out}}(1) \times M_{K+1\to\text{OR-out}}(1), \end{aligned} \tag{C.16}$$

where we have used the fact that

$$\sum_{\substack{y_1,\cdots,y_{i-1} \\ y_{i+1},\cdots,y_K}} \prod_{k=1,k\neq i}^{K} M_{k\to\text{OR-out}}(y_k) = 1. \tag{C.17}$$

and $\mu_i(0;\omega) = 1 - \mu_i(1;\omega)$. After some algebra, the ratio of marginals becomes:

$$\begin{aligned}
\frac{\mu_i(1;\omega)}{\mu_i(0;\omega)} &= \frac{M_{i\to\text{OR-out}}(1) \times M_{K+1\to\text{OR-out}}(1)}{Z_{\text{OR-out}}(\omega) - M_{i\to\text{OR-out}}(1) \times M_{K+1\to\text{OR-out}}(1)} \\
&= \left(\frac{Z_{\text{OR-out}}(\omega)}{M_{i\to\text{OR-out}}(1) \times M_{K+1\to\text{OR-out}}(1)} - 1 \right)^{-1} \\
&= \left(\frac{M_{K+1\to\text{OR-out}}(1) \left(1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right) + \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)}{M_{i\to\text{OR-out}}(1) \times M_{K+1\to\text{OR-out}}(1)} - 1 \right)^{-1} \\
&= \left(\frac{1 - \left(1 - m_{K+1\to\text{OR-out}}^{-1}\right) \times \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)}{M_{i\to\text{OR-out}}(1)} - 1 \right)^{-1} \\
&= m_{i\to\text{OR-out}} \left(1 - (1 - m_{K+1\to\text{OR-out}}^{-1}) \prod_{k=1, k\neq i}^{K} (1 + m_{k\to\text{OR-out}})^{-1} \right)^{-1}. \qquad \text{(C.18)}
\end{aligned}$$

We next consider the case where $i = K+1$; it is easier to derive $\mu_{K+1}(0;\omega)$ instead of $\mu_{K+1}(1;\omega)$:

$$\begin{aligned}
\mu_{K+1}(0;\omega) &= Z_{\text{OR-out}}(\omega)^{-1} \times \sum_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR-out}} \\ y_{K+1}=0}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\
&= Z_{\text{OR-out}}(\omega)^{-1} \times \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0), \qquad \text{(C.19)}
\end{aligned}$$

and $\mu_{K+1}(1;\omega) = 1 - \mu_{K+1}(0;\omega)$. After some algebra, the ratio of marginals becomes:

$$\begin{aligned}
\frac{\mu_{K+1}(1;\omega)}{\mu_{K+1}(0;\omega)} &= \frac{Z_{\text{OR-out}}(\omega) - \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)}{\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)} \\
&= \frac{Z_{\text{OR-out}}(\omega)}{\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)} - 1 \\
&= \frac{M_{K+1\to\text{OR-out}}(1) \left(1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right) + \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)}{\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)} - 1 \\
&= \frac{M_{K+1\to\text{OR-out}}(1) \left(1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)\right)}{\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0)} \\
&= m_{K+1\to\text{OR-out}} \times \frac{1 - \prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)}{\prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)} \\
&= m_{K+1\to\text{OR-out}} \times \left(\prod_{k=1}^{K} (1 + m_{k\to\text{OR-out}}) - 1 \right). \qquad \text{(C.20)}
\end{aligned}$$

From Proposition 5.4, we then obtain the following message updates:

$$\boxed{m_{\text{OR-out}\to i} = \left(1 - (1 - m_{K+1\to\text{OR-out}}^{-1}) \prod_{k=1,k\neq i}^{K} (1 + m_{k\to\text{OR-out}})^{-1}\right)^{-1},} \tag{C.21}$$

for $i \leq K$, and

$$\boxed{m_{\text{OR-out}\to K+1} = \prod_{k=1}^{K}(1 + m_{k\to\text{OR-out}}) - 1.} \tag{C.22}$$

Like in the previous factors, the entropy can be computed via:

$$\begin{aligned} H_{\text{OR-out}}(b_{\text{OR-out}}(.)) &= \log Z_{\text{OR-out}}(\omega) - \sum_{i=1}^{K+1} \sum_{y_i\in\{0,1\}} \mu_i(y_i;\omega) \log M_{i\to\text{OR-out}}(y_i) \\ &= \log Z_{\text{OR-out}}(\omega) - \sum_{i=1}^{K+1} \mu_i(1;\omega) \log m_{i\to\text{OR-out}} \\ &\quad + \sum_{i=1}^{K+1} \log(1 + m_{i\to\text{OR-out}}). \end{aligned} \tag{C.23}$$

C.2 Max-Product Messages

The One-hot XOR Factor. We are now going to compute expressions for the mode and max-marginals associated with the one-hot XOR factor. Then, invoking Proposition 5.5, we can provide closed-form expressions for the messages and beliefs.

Let us start with the mode, which requires us to maximize over all the allowed configurations (*i.e.*, those in the acceptance set $\mathcal{S}_{\text{XOR}}$). Like in the sum-product case, this can be done explicitly, since the cardinality of $\mathcal{S}_{\text{XOR}}$ is linear in K. We obtain:

$$\begin{aligned} P^*_{\text{XOR}}(\omega) &= \max_{\boldsymbol{y}\in\mathcal{S}_{\text{XOR}}} \prod_{k=1}^{K} M_{k\to\text{XOR}}(y_k) \\ &= \max_{i\in[K]} \left(M_{i\to\text{XOR}}(1) \prod_{\substack{k=1\\k\neq i}}^{K} M_{k\to\text{XOR}}(0) \right) \\ &= \left(\max_{i\in[K]} \frac{M_{i\to\text{XOR}}(1)}{M_{i\to\text{XOR}}(0)}\right) \times \prod_{k=1}^{K} M_{k\to\text{XOR}}(0) \\ &= \max_{i\in[K]} m_{i\to\text{XOR}} \times \prod_{i=1}^{K}(1 + m_{i\to\text{XOR}})^{-1}. \end{aligned} \tag{C.24}$$

For the max-marginals, we obtain:

$$\begin{aligned} \nu_i(1;\omega) &= \max_{\substack{\boldsymbol{y}\in S_{\mathrm{XOR}} \\ y_i=1}} \prod_{k=1}^{K} M_{k\to\mathrm{XOR}}(y_k) \\ &= M_{i\to\mathrm{XOR}}(1) \prod_{\substack{k=1 \\ k\neq i}}^{K} M_{k\to\mathrm{XOR}}(0) \\ &= m_{i\to\mathrm{XOR}} \prod_{k=1}^{K} (1+m_{k\to\mathrm{XOR}})^{-1} \end{aligned} \tag{C.25}$$

and

$$\begin{aligned} \nu_i(0;\omega) &= \max_{\substack{\boldsymbol{y}\in S_{\mathrm{XOR}} \\ y_i=0}} \prod_{k=1}^{K} M_{k\to\mathrm{XOR}}(y_k) \\ &= M_{i\to\mathrm{XOR}}(0) \times \max_{j\in[K]\setminus\{i\}} \left(M_{j\to\mathrm{XOR}}(1) \prod_{\substack{k=1 \\ k\notin\{i,j\}}}^{K} M_{k\to\mathrm{XOR}}(0) \right) \\ &= M_{i\to\mathrm{XOR}}(0) \times \max_{j\in[K]\setminus\{i\}} m_{j\to\mathrm{XOR}} \times \prod_{\substack{k=1 \\ k\neq i}}^{K} (1+m_{i\to\mathrm{XOR}})^{-1} \\ &= \max_{j\in[K]\setminus\{i\}} m_{j\to\mathrm{XOR}} \times \prod_{k=1}^{K} (1+m_{i\to\mathrm{XOR}})^{-1}, \end{aligned} \tag{C.26}$$

so the marginal ratio is:

$$\frac{\nu_i(1;\omega)}{\nu_i(0;\omega)} = \frac{m_{i\to\mathrm{XOR}}}{\max_{k\in[K]\setminus\{i\}} m_{k\to\mathrm{XOR}}}. \tag{C.27}$$

From Proposition 5.5, we now have that the message updates are given by:

$$M_{\mathrm{XOR}\to i}(1) \propto M_{i\to\mathrm{XOR}}(1)^{-1} \times \nu_i(1;\omega) \tag{C.28}$$

and

$$M_{\mathrm{XOR}\to i}(0) \propto M_{i\to\mathrm{XOR}}(0)^{-1} \times \nu_i(0;\omega), \tag{C.29}$$

whence, since we have

$$m_{\mathrm{XOR}\to i} = \frac{M_{i\to\mathrm{XOR}}(1)}{M_{i\to\mathrm{XOR}}(0)} = m_{k\to\mathrm{XOR}}^{-1} \times \frac{\nu_i(1;\omega)}{\nu_i(0;\omega)}, \tag{C.30}$$

we obtain

$$\boxed{m_{\mathrm{XOR}\to i} = \left(\max_{k\neq i} m_{k\to\mathrm{XOR}} \right)^{-1}.} \tag{C.31}$$

The OR Factor. We next derive expressions for the mode and max-marginals associated with the OR factor. Let us start with the mode. The acceptance set $\mathcal{S}_{\text{OR}}$ accepts all configurations except the all-zeros one, where $y_1 = \ldots = y_K = 0$. Assuming for the moment that the all-zeros configuration would not receive the largest score, we could easily compute the mode as the componentwise product of the maxima,

$$\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR}}(y_k). \tag{C.32}$$

The only scenario that would get us in trouble would be if for every $k \in [K]$ we have $M_{k\to\text{OR}}(1) < M_{k\to\text{OR}}(0)$, since in that case the unique MAP would be the all-zeros configuration, which is forbidden. This condition can be written equivalently as

$$\max_{k\in[K]} \frac{M_{k\to\text{OR}}(1)}{M_{k\to\text{OR}}(0)} < 1. \tag{C.33}$$

If (C.33) holds, then the true mode of the OR factor is instead the *second best configuration*; in other words, we would need to look at the largest ratio $\frac{M_{k\to\text{OR}}(1)}{M_{k\to\text{OR}}(0)}$ and flip the corresponding y_k to 1; the mode would then be the value in Eq. C.32 times this largest ratio. This can all be written compactly via the following expression:

$$P^*_{\text{OR}}(\omega) = \left(\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR}}(y_k)\right) \times \min\left\{1, \max_{k\in[K]} \frac{M_{k\to\text{OR}}(1)}{M_{k\to\text{OR}}(0)}\right\}. \tag{C.34}$$

The minimum with one in Eq. C.34 is taken to ensure that the multiplication just mentioned takes place if and only if condition (C.33) holds.

We next turn to the max-marginals. By definition, we have

$$\begin{aligned} \nu_i(1;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR}} \\ y_i=1}} \prod_{k=1}^{K} M_{k\to\text{OR}}(y_k) \\ &= M_{i\to\text{OR}}(1) \prod_{\substack{k=1 \\ k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR}}(y_k), \end{aligned} \tag{C.35}$$

since any assignment in which $y_i = 1$ will necessarily be in the acceptance set. The computation of $b_i(0;\omega)$ is less trivial but may be carried out with a similar reasoning as the computation of the mode:

$$\begin{aligned} \nu_i(0;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR}} \\ y_i=0}} \prod_{k=1}^{K} M_{k\to\text{OR}}(y_k) \\ &= M_{i\to\text{OR}}(0) \left(\prod_{\substack{k=1 \\ k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR}}(y_k)\right) \times \min\left\{1, \max_{k\in[K]\setminus\{i\}} \frac{M_{k\to\text{OR}}(1)}{M_{k\to\text{OR}}(0)}\right\}. \end{aligned} \tag{C.36}$$

The ratio of the max-marginals is:

$$\begin{aligned}
\frac{\nu_i(1;\omega)}{\nu_i(0;\omega)} &= \frac{M_{i\to\mathrm{OR}}(1)\prod_{\substack{k=1\\k\neq i}}^{K}\max_{y_k} M_{k\to\mathrm{OR}}(y_k)}{M_{i\to\mathrm{OR}}(0)\left(\prod_{\substack{k=1\\k\neq i}}^{K}\max_{y_k} M_{k\to\mathrm{OR}}(y_k)\right)\times\min\left\{1,\max_{k\in[K]\setminus\{i\}}\frac{M_{k\to\mathrm{OR}}(1)}{M_{k\to\mathrm{OR}}(0)}\right\}} \\
&= m_{i\to\mathrm{OR}}\times\frac{1}{\min\left\{1,\max_{k\in[K]\setminus\{i\}} m_{k\to\mathrm{OR}}\right\}} \\
&= m_{i\to\mathrm{OR}}\times\max\left\{1,\min_{k\in[K]\setminus\{i\}} m_{k\to\mathrm{OR}}^{-1}\right\}. \qquad \text{(C.37)}
\end{aligned}$$

As a consequence, we obtain the following simple expression for the max-product message updates, invoking Proposition 5.5:

$$\boxed{m_{\mathrm{OR}\to i}=\max\left\{1,\min_{k\neq i} m_{k\to\mathrm{OR}}^{-1}\right\}.} \qquad \text{(C.38)}$$

The OR-with-output Factor. We now turn to the problem of computing the mode in the OR-with-output factor. Two things can happen:

- Either the MAP assignment has $y_{K+1}=0$, in which case everything must be zero;
- Or the MAP assignment has $y_{K+1}=1$, in which case the problem can be reduced to the OR case, as at least one y_k (with $k\in[K]$) must be one.

Hence, the mode will be the maximum of these two possibilities. The first possibility corresponds to the score

$$\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0); \qquad \text{(C.39)}$$

the second corresponds to $M_{K+1\to\text{OR-out}}(1)$ times the score in Eq. C.34. Putting the pieces together, we obtain

$$\begin{aligned}
P^*_{\text{OR-out}}(\omega) &= \max\left\{\prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0),\quad M_{K+1\to\text{OR-out}}(1)\times\right. \\
&\qquad \left.\left(\prod_{k=1}^{K}\max_{y_k} M_{k\to\text{OR-out}}(y_k)\right)\times\min\left\{1,\max_{k\in[K]}\frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)}\right\}\right\}. \qquad \text{(C.40)}
\end{aligned}$$

We now turn to the max-marginals. Let us first handle the case where $i \leq K$. By definition, we have

$$\begin{aligned} \nu_i(1;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR-out}} \\ y_i=1}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= M_{i\to\text{OR-out}}(1) M_{K+1\to\text{OR-out}}(1) \prod_{\substack{k=1 \\ k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k), \end{aligned} \tag{C.41}$$

since any assignment in which $y_i = 1$ will necessarily be in the acceptance set, provided $y_{K+1} = 1$. As in the OR case, the computation of $\nu_i(0;\omega)$ is less trivial but may be carried out with a similar reasoning as the computation of the mode:

$$\begin{aligned} \nu_i(0;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR-out}} \\ y_i=0}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= M_{i\to\text{OR-out}}(0) \times \max \left\{ \prod_{\substack{k=1 \\ k\neq i}}^{K+1} M_{k\to\text{OR-out}}(0), \quad M_{K+1\to\text{OR-out}}(1) \times \right. \\ &\qquad \left. \left(\prod_{\substack{k=1 \\ k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k) \right) \times \min \left\{ 1, \max_{k\in[K]\setminus\{i\}} \frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)} \right\} \right\}. \end{aligned} \tag{C.42}$$

Let us now address the case where $i = K+1$. We have that the computation of $\nu_{K+1}(1;\omega)$ can be reduced to computing a mode of the OR-factor:

$$\begin{aligned} \nu_{K+1}(1;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR-out}} \\ y_{K+1}=1}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= M_{K+1\to\text{OR-out}}(1) \times \max_{y_1,\ldots,y_K\in\mathcal{S}_{\text{OR}}} \prod_{k=1}^{K} M_{k\to\text{OR-out}}(y_k) \\ &= M_{K+1\to\text{OR-out}}(1) \times \\ &\qquad \left(\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k) \right) \times \min \left\{ 1, \max_{k\in[K]} \frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)} \right\}. \end{aligned} \tag{C.43}$$

As for $\nu_{K+1}(0;\omega)$, it is immediate from the definition:

$$\begin{aligned} \nu_{K+1}(0;\omega) &= \max_{\substack{\boldsymbol{y}\in\mathcal{S}_{\text{OR-out}} \\ y_{K+1}=0}} \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(y_k) \\ &= \prod_{k=1}^{K+1} M_{k\to\text{OR-out}}(0). \end{aligned} \tag{C.44}$$

Given the max-marginals, we can now compute their ratios. As before, we start with the case $i \leq K$. After some algebra, one obtains:

$$\begin{aligned}
\frac{\nu_i(1;\omega)}{\nu_i(0;\omega)} &= \frac{m_{i\to\text{OR-out}} \times \prod_{\substack{k=1\\k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k)}{\max\left\{\begin{array}{l} m_{K+1\to\text{OR-out}}^{-1} \times \prod_{\substack{k=1\\k\neq i}}^{K} M_{k\to\text{OR-out}}(0), \\ \left(\prod_{\substack{k=1\\k\neq i}}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k)\right) \times \min\left\{1, \max_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}\right\} \end{array}\right\}} \\
&= \frac{m_{i\to\text{OR-out}}}{\max\left\{\begin{array}{l} m_{K+1\to\text{OR-out}}^{-1} \times \prod_{\substack{k=1\\k\neq i}}^{K} \min_{y_k} \frac{M_{k\to\text{OR-out}}(0)}{M_{k\to\text{OR-out}}(y_k)}, \\ \min\left\{1, \max_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}\right\} \end{array}\right\}} \\
&= m_{i\to\text{OR-out}} \times \min\left\{\begin{array}{l} m_{K+1\to\text{OR-out}} \times \prod_{\substack{k=1\\k\neq i}}^{K} \max\{1, m_{k\to\text{OR-out}}\}, \\ \max\left\{1, \min_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}^{-1}\right\} \end{array}\right\},
\end{aligned} \tag{C.45}$$

where in the last equality we have used the fact that

$$\left(\min_{y_k} \frac{M_{k\to\text{OR-out}}(0)}{M_{k\to\text{OR-out}}(y_k)}\right)^{-1} = \max_{y_k} \frac{M_{k\to\text{OR-out}}(y_k)}{M_{k\to\text{OR-out}}(0)} = \max\{1, m_{k\to\text{OR-out}}\}. \tag{C.46}$$

For $i = K+1$, we have:

$$\begin{aligned}
\frac{\nu_{K+1}(1;\omega)}{\nu_{K+1}(0;\omega)} &= m_{K+1\to\text{OR-out}} \times \\
&\qquad \frac{\left(\prod_{k=1}^{K} \max_{y_k} M_{k\to\text{OR-out}}(y_k)\right) \times \min\left\{1, \max_{k\in[K]} \frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)}\right\}}{\prod_{k=1}^{K} M_{k\to\text{OR-out}}(0)} \\
&= m_{K+1\to\text{OR-out}} \times \\
&\qquad \left(\prod_{k=1}^{K} \max_{y_k} \frac{M_{k\to\text{OR-out}}(y_k)}{M_{k\to\text{OR-out}}(0)}\right) \times \min\left\{1, \max_{k\in[K]} \frac{M_{k\to\text{OR-out}}(1)}{M_{k\to\text{OR-out}}(0)}\right\} \\
&= m_{K+1\to\text{OR-out}} \times \\
&\qquad \left(\prod_{k=1}^{K} \max\{1, m_{k\to\text{OR-out}}\}\right) \times \min\left\{1, \max_{k\in[K]} m_{k\to\text{OR-out}}\right\}
\end{aligned} \tag{C.47}$$

As a consequence of the facts above, we obtain the following simple expression for the max-product message updates:

$$\boxed{m_{\text{OR-out}\to i} = \min\left\{ m_{K+1\to\text{OR-out}} \prod_{\substack{k=1\\k\neq i}}^{K} \max\{1, m_{k\to\text{OR-out}}\}, \max\left\{1, \min_{k\in[K]\setminus\{i\}} m_{k\to\text{OR-out}}^{-1}\right\}\right\}} \tag{C.48}$$

for $i \leq K$, and

$$\boxed{m_{\text{OR-out}\to K+1} = \left(\prod_{k=1}^{K} \max\{1, m_{k\to\text{OR-out}}\}\right) \times \min\left\{1, \max_{k\in[K]} m_{k\to\text{OR-out}}\right\}.} \tag{C.49}$$

Appendix D

Proof of Convergence Rate of ADMM and AD^3

In this appendix, we prove the $O(1/\epsilon)$ convergence bound of the ADMM algorithm, first in the sense of a variational inequality involving the primal and dual variables, using results recently established by He and Yuan (2011) and Wang and Banerjee (2012), and then in terms of the dual objective value, which follows easily from the aforementioned results. We then consider the special case of AD^3, where we interpret the constants in the bound in terms of properties of the graphical model.

We start with the following proposition, which states the variational inequality associated with the Lagrangian function of the problem in Eq. 6.1. Recall that (6.1) is equivalent to

$$\max_{\boldsymbol{\lambda}\in\Lambda} \min_{\boldsymbol{u}\in\mathcal{U},\boldsymbol{v}\in\mathcal{V}} L(\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}), \tag{D.1}$$

where $L = L_0$ is the standard Lagrangian (the expression in Eq. 6.3 with $\eta = 0$), and $\Lambda := \{\boldsymbol{\lambda} \mid \min_{\boldsymbol{u}\in\mathcal{U},\boldsymbol{v}\in\mathcal{V}} L(\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}) > -\infty\}$.

Proposition D.1 (Variational inequality.) *Let $\mathcal{W} := \mathcal{U}\times\mathcal{V}\times\Lambda$. Given $\boldsymbol{w} = (\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}) \in \mathcal{W}$, define $h(\boldsymbol{w}) := f(\boldsymbol{u}) + g(\boldsymbol{v})$ and $F(\boldsymbol{w}) := (\mathbf{A}^\top\boldsymbol{\lambda}, \mathbf{B}^\top\boldsymbol{\lambda}, -(\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}-\boldsymbol{c}))$. Then, $\boldsymbol{w}^* := (\boldsymbol{u}^*,\boldsymbol{v}^*,\boldsymbol{\lambda}^*) \in \mathcal{W}$ is a primal-dual solution of Eq. D.1 if and only if:*

$$\forall \boldsymbol{w}\in\mathcal{W}, \quad h(\boldsymbol{w}) - h(\boldsymbol{w}^*) + (\boldsymbol{w}-\boldsymbol{w}^*)\cdot F(\boldsymbol{w}^*) \geq 0. \tag{D.2}$$

Proof. Assume $\boldsymbol{w}^*$ is a primal-dual solution of Eq. D.1. Then, from the saddle point conditions, we have for every $\boldsymbol{w} := (\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}) \in \mathcal{W}$:

$$L(\boldsymbol{u}^*,\boldsymbol{v}^*,\boldsymbol{\lambda}) \leq L(\boldsymbol{u}^*,\boldsymbol{v}^*,\boldsymbol{\lambda}^*) \leq L(\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}^*). \tag{D.3}$$

Hence:

$$\begin{aligned}
0 &\leq L(\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}^*) - L(\boldsymbol{u}^*,\boldsymbol{v}^*,\boldsymbol{\lambda}) \\
&= f(\boldsymbol{u}) + g(\boldsymbol{v}) + \boldsymbol{\lambda}^*\cdot(\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}-\boldsymbol{c}) - f(\boldsymbol{u}^*) - g(\boldsymbol{v}^*) - \boldsymbol{\lambda}\cdot(\mathbf{A}\boldsymbol{u}^*+\mathbf{B}\boldsymbol{v}^*-\boldsymbol{c}) \\
&= h(\boldsymbol{w}) - h(\boldsymbol{w}^*) + \boldsymbol{u}\cdot\mathbf{A}^\top\boldsymbol{\lambda}^* + \boldsymbol{v}\cdot\mathbf{B}^\top\boldsymbol{\lambda}^* - (\boldsymbol{\lambda}-\boldsymbol{\lambda}^*)\cdot(\mathbf{A}\boldsymbol{u}^*+\mathbf{B}\boldsymbol{v}^*-\boldsymbol{c}) - \boldsymbol{\lambda}^*\cdot(\mathbf{A}\boldsymbol{u}^*+\mathbf{B}\boldsymbol{v}^*) \\
&= h(\boldsymbol{w}) - h(\boldsymbol{w}^*) + (\boldsymbol{w}-\boldsymbol{w}^*)\cdot F(\boldsymbol{w}^*).
\end{aligned} \tag{D.4}$$

Conversely, let w^* satisfy Eq. D.2. Taking $w = (u^*, v^*, \lambda)$, we obtain $L(u^*, v^*, \lambda) \leq L(u^*, v^*, \lambda^*)$. Taking $w = (u, v, \lambda^*)$, we obtain $L(u^*, v^*, \lambda^*) \leq L(u, v, \lambda^*)$. Hence (u^*, v^*, λ^*) is a saddle point, and therefore it is a primal-dual solution. ■

We next reproduce the result established by Wang and Banerjee (2012) (which is related to previous work by He and Yuan (2011)) concerning the convergence rate of the general ADMM algorithm in terms of the variational inequality stated in Proposition D.1. The proof relies itself on variational inequalities associated with the updates in the ADMM algorithm (see Facchinei and Pang 2003 for a comprehensive overview of variational inequalities and complementarity problems).

Proposition D.2 (Variational convergence rate.) *Assume the conditions stated in Proposition 6.2, $\tau = 1$, and $\lambda^0 = \mathbf{0}$. Let $\bar{w}_T = \frac{1}{T}\sum_{t=1}^{T} w^t$, where the $w^t := (u^t, v^t, \lambda^t)$ are the ADMM iterates. Then we have after T iterations:*

$$\forall w \in \mathcal{W}, \quad h(\bar{w}_T) - h(w) + (\bar{w}_T - w) \cdot F(\bar{w}_T) \leq \frac{C}{T}, \tag{D.5}$$

where

$$C = \frac{\eta}{2}\|\mathbf{A}u + \mathbf{B}v^0 - c\|^2 + \frac{1}{2\eta}\|\lambda\|^2. \tag{D.6}$$

Proof. From the variational inequality associated with the u-update in Eq. 6.6 we have, for every $u \in \mathcal{U}$,

$$\begin{aligned} 0 \quad &\leq \quad \nabla_u L_\eta(u^{t+1}, v^t, \lambda^t) \cdot (u - u^{t+1}) \\ &= \quad \nabla f(u^{t+1}) \cdot (u - u^{t+1}) + (u - u^{t+1}) \cdot \mathbf{A}^\top(\lambda^t + \eta(\mathbf{A}u^{t+1} + \mathbf{B}v^t - c)) \\ &\leq^{(i)} \quad f(u) - f(u^{t+1}) + (u - u^{t+1}) \cdot \mathbf{A}^\top(\lambda^t + \eta(\mathbf{A}u^{t+1} + \mathbf{B}v^t - c)) \\ &\leq^{(ii)} \quad f(u) - f(u^{t+1}) + (u - u^{t+1}) \cdot \mathbf{A}^\top\lambda^{t+1} + \eta\mathbf{A}(u - u^{t+1}) \cdot \mathbf{B}(v^t - v^{t+1}), \end{aligned} \tag{D.7}$$

where in (i) we have used the convexity of f, and in (ii) we used Eq. 6.8 for the λ-updates. Similarly, we have the following from the variational inequality associated with the v-update in Eq. 6.7, for every $v \in \mathcal{V}$:

$$\begin{aligned} 0 \quad &\leq \quad \nabla_v L_\eta(u^{t+1}, v^{t+1}, \lambda^t) \cdot (v - v^{t+1}) \\ &= \quad \nabla g(v^{t+1}) \cdot (v - v^{t+1}) + (v - v^{t+1}) \cdot \mathbf{B}^\top(\lambda^t + \eta(\mathbf{A}u^{t+1} + \mathbf{B}v^{t+1} - c)) \\ &\leq^{(i)} \quad g(v) - g(v^{t+1}) + (v - v^{t+1}) \cdot \mathbf{B}^\top\lambda^{t+1}, \end{aligned} \tag{D.8}$$

where in (i) we have used the convexity of g. Summing (D.7) and (D.8) we obtain, for every $w \in \mathcal{W}$,

$$\begin{aligned} &h(w^{t+1}) - h(w) + (w^{t+1} - w) \cdot F(w^{t+1}) \\ &\quad \leq \quad \eta\mathbf{A}(u - u^{t+1}) \cdot \mathbf{B}(v^t - v^{t+1}) + \eta^{-1}(\lambda - \lambda^{t+1}) \cdot (\lambda^{t+1} - \lambda^t). \end{aligned} \tag{D.9}$$

We next rewrite the two terms in the right hand side. We have

$$\begin{aligned}\eta \mathbf{A}(\boldsymbol{u}-\boldsymbol{u}^{t+1})\cdot\mathbf{B}(\boldsymbol{v}^t-\boldsymbol{v}^{t+1}) &= \frac{\eta}{2}\Big(\|\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}^t-\boldsymbol{c}\|^2-\|\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}^{t+1}-\boldsymbol{c}\|^2\\ &+ \|\mathbf{A}\boldsymbol{u}^{t+1}+\mathbf{B}\boldsymbol{v}^{t+1}-\boldsymbol{c}\|^2-\|\mathbf{A}\boldsymbol{u}^{t+1}+\mathbf{B}\boldsymbol{v}^t-\boldsymbol{c}\|^2\Big) \quad \text{(D.10)}\end{aligned}$$

and

$$\eta^{-1}(\boldsymbol{\lambda}-\boldsymbol{\lambda}^{t+1})\cdot(\boldsymbol{\lambda}^{t+1}-\boldsymbol{\lambda}^t) = \frac{1}{2\eta}\Big(\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^t\|^2-\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^{t+1}\|^2-\|\boldsymbol{\lambda}^t-\boldsymbol{\lambda}^{t+1}\|^2\Big). \quad \text{(D.11)}$$

Summing (D.9) over t and noting that $\eta^{-1}\|\boldsymbol{\lambda}^t-\boldsymbol{\lambda}^{t+1}\|^2=\eta\|\mathbf{A}\boldsymbol{u}^{t+1}+\mathbf{B}\boldsymbol{v}^{t+1}-\boldsymbol{c}\|^2$, we obtain by the telescoping sum property:

$$\begin{aligned}&\sum_{t=0}^{T-1}\Big(h(\boldsymbol{w}^{t+1})-h(\boldsymbol{w})+(\boldsymbol{w}^{t+1}-\boldsymbol{w})\cdot F(\boldsymbol{w}^{t+1})\Big)\\ &\leq \frac{\eta}{2}\left(\|\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}^0-\boldsymbol{c}\|^2-\|\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}^T-\boldsymbol{c}\|^2-\sum_{t=0}^{T-1}\|\mathbf{A}\boldsymbol{u}^{t+1}+\mathbf{B}\boldsymbol{v}^t-\boldsymbol{c}\|^2\right)\\ &\quad+\frac{1}{2\eta}\Big(\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^0\|^2-\|\boldsymbol{\lambda}-\boldsymbol{\lambda}^T\|^2\Big)\\ &\leq \frac{\eta}{2}\|\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}^0-\boldsymbol{c}\|^2+\frac{1}{2\eta}\|\boldsymbol{\lambda}\|^2. \quad \text{(D.12)}\end{aligned}$$

Now, using the convexity of h, we have that $h(\bar{\boldsymbol{w}}_T)=h\left(\frac{1}{T}\sum_{t=0}^{T-1}\boldsymbol{w}^{t+1}\right)\leq\frac{1}{T}\sum_{t=0}^{T-1}h(\boldsymbol{w}^{t+1})$. Note also that, for every $\tilde{\boldsymbol{w}}$, the function $\boldsymbol{w}\mapsto(\boldsymbol{w}-\tilde{\boldsymbol{w}})\cdot F(\boldsymbol{w})$ is affine:

$$\begin{aligned}(\boldsymbol{w}-\tilde{\boldsymbol{w}})\cdot F(\boldsymbol{w}) &= (\boldsymbol{u}-\tilde{\boldsymbol{u}})\cdot\mathbf{A}^\top\boldsymbol{\lambda}+(\boldsymbol{v}-\tilde{\boldsymbol{v}})\cdot\mathbf{B}^\top\boldsymbol{\lambda}-(\boldsymbol{\lambda}-\tilde{\boldsymbol{\lambda}})\cdot(\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}-\boldsymbol{c})\\ &= -\tilde{\boldsymbol{u}}\cdot\mathbf{A}^\top\boldsymbol{\lambda}-\tilde{\boldsymbol{v}}\cdot\mathbf{B}^\top\boldsymbol{\lambda}+\tilde{\boldsymbol{\lambda}}\cdot(\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}-\boldsymbol{c})+\boldsymbol{\lambda}\cdot\boldsymbol{c}\\ &= -(\mathbf{A}\tilde{\boldsymbol{u}}+\mathbf{B}\tilde{\boldsymbol{v}}-\boldsymbol{c})\cdot\boldsymbol{\lambda}+\tilde{\boldsymbol{\lambda}}\cdot(\mathbf{A}\boldsymbol{u}+\mathbf{B}\boldsymbol{v}-\boldsymbol{c})\\ &= F(\tilde{\boldsymbol{w}})\cdot\boldsymbol{w}-\boldsymbol{c}\cdot\tilde{\boldsymbol{\lambda}}. \quad \text{(D.13)}\end{aligned}$$

As a consequence,

$$\frac{1}{T}\sum_{t=0}^{T-1}\Big(h(\boldsymbol{w}^{t+1})+(\boldsymbol{w}^{t+1}-\boldsymbol{w})\cdot F(\boldsymbol{w}^{t+1})\Big)\geq h(\bar{\boldsymbol{w}}_T)+(\bar{\boldsymbol{w}}_T-\boldsymbol{w})\cdot F(\bar{\boldsymbol{w}}_T), \quad \text{(D.14)}$$

and from (D.12):

$$h(\bar{\boldsymbol{w}}_T)-h(\boldsymbol{w})+(\bar{\boldsymbol{w}}_T-\boldsymbol{w})\cdot F(\bar{\boldsymbol{w}}_T)\leq C/T, \quad \text{(D.15)}$$

with C as in Eq. D.6. Note also that, since Λ is convex and for every t we have $\boldsymbol{\lambda}^t\in\Lambda$, we also have $\bar{\boldsymbol{\lambda}}_T\in\Lambda$. ■

Our next step is to reuse the bound stated in Proposition D.2 to derive a convergence rate for the dual problem.

Proposition D.3 (Dual convergence rate.) *Assume the conditions stated in Proposition D.2, with*

$\bar{\boldsymbol{w}}_T$ *defined analogously. Let* $\varphi : \Lambda \to \mathbb{R}$ *denote the dual objective function:*

$$\varphi(\boldsymbol{\lambda}) := \min_{\boldsymbol{u}\in\mathcal{U},\boldsymbol{v}\in\mathcal{V}} L(\boldsymbol{u},\boldsymbol{v},\boldsymbol{\lambda}), \tag{D.16}$$

and let $\boldsymbol{\lambda}^* = \arg\max_{\lambda\in\Lambda} \varphi(\boldsymbol{\lambda})$ *be a dual solution. Then, after* T *iterations of ADMM, we achieve an* $O(\frac{1}{T})$*-accurate solution* $\bar{\boldsymbol{\lambda}}_T$*:*

$$\varphi(\boldsymbol{\lambda}^*) - \frac{C}{T} \le \varphi(\bar{\boldsymbol{\lambda}}_T) \le \varphi(\boldsymbol{\lambda}^*), \tag{D.17}$$

where the constant C *is given by*

$$\begin{aligned} C &= \frac{\eta}{2}\left(\|\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2 + \|\mathbf{A}\widehat{\boldsymbol{u}}_T + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2\right) + \frac{1}{2\eta}\left(\|\boldsymbol{\lambda}^*\|^2 + \|\bar{\boldsymbol{\lambda}}_T\|^2\right) \\ &\le \frac{5\eta}{2}\left(\max_{\boldsymbol{u}\in\mathcal{U}} \|\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2\right) + \frac{5}{2\eta}\|\boldsymbol{\lambda}^*\|^2. \end{aligned} \tag{D.18}$$

Proof. By applying Proposition D.2 to $\boldsymbol{w} = (\bar{\boldsymbol{u}}_T, \bar{\boldsymbol{v}}_T, \boldsymbol{\lambda})$ we obtain for arbitrary $\boldsymbol{\lambda} \in \Lambda$:

$$-(\bar{\boldsymbol{\lambda}}_T - \boldsymbol{\lambda}) \cdot (\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\bar{\boldsymbol{v}}_T - \boldsymbol{c}) \le O(1/T). \tag{D.19}$$

By applying Proposition D.2 to $\boldsymbol{w} = (\boldsymbol{u}, \boldsymbol{v}, \bar{\boldsymbol{\lambda}}_T)$ we obtain for arbitrary $\boldsymbol{u} \in \mathcal{U}$ and $\boldsymbol{v} \in \mathcal{V}$:

$$\begin{aligned} &f(\bar{\boldsymbol{u}}_T) + g(\bar{\boldsymbol{v}}_T) + (\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\bar{\boldsymbol{v}}_T - \boldsymbol{c}) \cdot \bar{\boldsymbol{\lambda}}_T \\ &\quad\le f(\boldsymbol{u}) + g(\boldsymbol{v}) + (\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}) \cdot \bar{\boldsymbol{\lambda}}_T + O(1/T). \end{aligned} \tag{D.20}$$

In particular, let $\varphi(\bar{\boldsymbol{\lambda}}_T) := \min_{\boldsymbol{u}\in\mathcal{U},\boldsymbol{v}\in\mathcal{V}} L(\boldsymbol{u},\boldsymbol{v},\bar{\boldsymbol{\lambda}}_T) = L(\widehat{\boldsymbol{u}}_T, \widehat{\boldsymbol{v}}_T, \bar{\boldsymbol{\lambda}}_T)$ be the value of the dual objective at $\bar{\boldsymbol{\lambda}}_T$, where $(\widehat{\boldsymbol{u}}_T, \widehat{\boldsymbol{v}}_T)$ are the corresponding minimizers. We then have:

$$f(\bar{\boldsymbol{u}}_T) + g(\bar{\boldsymbol{v}}_T) + (\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\bar{\boldsymbol{v}}_T - \boldsymbol{c}) \cdot \bar{\boldsymbol{\lambda}}_T \le \varphi(\bar{\boldsymbol{\lambda}}_T) + O(1/T). \tag{D.21}$$

Finally we have (letting $\boldsymbol{w}^* = (\boldsymbol{u}^*, \boldsymbol{v}^*, \boldsymbol{\lambda}^*)$ be the optimal primal-dual solution):

$$\begin{aligned} f(\boldsymbol{u}^*) + g(\boldsymbol{v}^*) &= \varphi(\boldsymbol{\lambda}^*) \\ &= \min_{\boldsymbol{u}\in\mathcal{U},\boldsymbol{v}\in\mathcal{V}} f(\boldsymbol{u}) + g(\boldsymbol{v}) + \boldsymbol{\lambda}^* \cdot (\mathbf{A}\boldsymbol{u} + \mathbf{B}\boldsymbol{v} - \boldsymbol{c}) \\ &\le f(\bar{\boldsymbol{u}}_T) + g(\bar{\boldsymbol{v}}_T) + \boldsymbol{\lambda}^* \cdot (\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\bar{\boldsymbol{v}}_T - \boldsymbol{c}) \\ &\le^{(i)} f(\bar{\boldsymbol{u}}_T) + g(\bar{\boldsymbol{v}}_T) + \bar{\boldsymbol{\lambda}}_T \cdot (\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\bar{\boldsymbol{v}}_T - \boldsymbol{c}) + O(1/T) \\ &\le^{(ii)} \varphi(\bar{\boldsymbol{\lambda}}_T) + O(1/T), \end{aligned} \tag{D.22}$$

where in (i) we have used Eq. D.19 and in (ii) we have used Eq. D.21. Note that, since any dual objective value is a lower bound, we also have $\varphi(\bar{\boldsymbol{\lambda}}_T) \le f(\boldsymbol{u}^*) + g(\boldsymbol{v}^*)$. Since we applied Proposition D.2 twice, the constant hidden in the O-notation becomes

$$C = \frac{\eta}{2}\left(\|\mathbf{A}\bar{\boldsymbol{u}}_T + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2 + \|\mathbf{A}\widehat{\boldsymbol{u}}_T + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2\right) + \frac{1}{2\eta}\left(\|\boldsymbol{\lambda}^*\|^2 + \|\bar{\boldsymbol{\lambda}}_T\|^2\right). \tag{D.23}$$

Even though C depends on $\bar{\boldsymbol{u}}_T$, $\widehat{\boldsymbol{u}}_T$, and $\bar{\boldsymbol{\lambda}}_T$, it is easy to obtain an upper bound on C when $\mathcal{U}$ is a bounded set, using the fact that the sequence $(\boldsymbol{\lambda}^t)_{t\in\mathbb{N}}$ is bounded by a constant, which

implies that the average $\bar{\boldsymbol{\lambda}}_T$ is also bounded. Indeed, from Boyd et al. (2011, p.107), we have that

$$V^t := \eta^{-1}\|\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^t\|^2 + \eta\|\mathbf{B}(\boldsymbol{v}^* - \boldsymbol{v}^t)\|^2 \tag{D.24}$$

is a Lyapunov function, *i.e.*, $0 \leq V^{t+1} \leq V^t$ for every $t \in \mathbb{N}$. This implies that $V^t \leq V^0 = \eta^{-1}\|\boldsymbol{\lambda}^*\|^2 + \eta\|\mathbf{B}(\boldsymbol{v}^* - \boldsymbol{v}^0)\|^2$; since $V^t \geq \eta^{-1}\|\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^t\|^2$, we can replace above and write:

$$\begin{aligned} 0 &\geq \|\boldsymbol{\lambda}^* - \boldsymbol{\lambda}^t\|^2 - \|\boldsymbol{\lambda}^*\|^2 - \eta^2\|\mathbf{B}(\boldsymbol{v}^* - \boldsymbol{v}^0)\|^2 \\ &= \|\boldsymbol{\lambda}^t\|^2 - 2\boldsymbol{\lambda}^* \cdot \boldsymbol{\lambda}^t - \eta^2\|\mathbf{B}(\boldsymbol{v}^* - \boldsymbol{v}^0)\|^2 \\ &\geq \|\boldsymbol{\lambda}^t\|^2 - 2\|\boldsymbol{\lambda}^*\|\|\boldsymbol{\lambda}^t\| - \eta^2\|\mathbf{B}(\boldsymbol{v}^* - \boldsymbol{v}^0)\|^2, \end{aligned} \tag{D.25}$$

where in the last line we invoked the Cauchy-Schwarz inequality. By solving a quadratic equation, it is easy to see that we must have

$$\|\boldsymbol{\lambda}^t\| \leq \|\boldsymbol{\lambda}^*\| + \sqrt{\|\boldsymbol{\lambda}^*\|^2 + \eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2}, \tag{D.26}$$

which in turn implies

$$\begin{aligned} \|\boldsymbol{\lambda}^t\|^2 &\leq 2\|\boldsymbol{\lambda}^*\|^2 + \eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2 + 2\|\boldsymbol{\lambda}^*\|\sqrt{\|\boldsymbol{\lambda}^*\|^2 + \eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2} \\ &\leq 2\|\boldsymbol{\lambda}^*\|^2 + \eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2 + 2(\|\boldsymbol{\lambda}^*\|^2 + \eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2) \\ &= 4\|\boldsymbol{\lambda}^*\|^2 + 3\eta^2\|\mathbf{B}(\boldsymbol{v}^0 - \boldsymbol{v}^*)\|^2 \\ &= 4\|\boldsymbol{\lambda}^*\|^2 + 3\eta^2\|\mathbf{A}\boldsymbol{u}^* + \mathbf{B}\boldsymbol{v}^0 - \boldsymbol{c}\|^2, \end{aligned} \tag{D.27}$$

where the last line follows from the fact that $\mathbf{A}\boldsymbol{u}^* + \mathbf{B}\boldsymbol{v}^* = \boldsymbol{c}$. Replacing (D.27) in (D.23) yields Eq. D.18. ∎

We are finally ready to see how the bounds above apply to the AD^3 algorithm, and in particular, how the constants in the bound relate to the structure of the graphical model.

Proposition D.4 (Dual convergence rate of AD^3.) *After T iterations of AD^3, we achieve an $O(\frac{1}{T})$-accurate solution $\bar{\boldsymbol{\lambda}}_T := \sum_{t=0}^{T-1} \boldsymbol{\lambda}^{(t)}$:*

$$\varphi(\boldsymbol{\lambda}^*) - \frac{C}{T} \leq \varphi(\bar{\boldsymbol{\lambda}}_T) \leq \varphi(\boldsymbol{\lambda}^*), \tag{D.28}$$

where the constant C is given by

$$C = \frac{5\eta}{2}\sum_i \deg(i)(1 - |\mathcal{Y}_i|^{-1}) + \frac{5}{2\eta}\|\boldsymbol{\lambda}^*\|^2. \tag{D.29}$$

Hence, as expected, the constant in the bound increases with the number of overlapping variables, the size of the sets $\mathcal{Y}_i$, and the magnitude of the optimal dual vector $\boldsymbol{\lambda}^*$.

Proof. With the uniform initialization of the $\boldsymbol{\zeta}$-variables in AD^3, the first term in the second line of Eq. D.18 is maximized by a choice of $\boldsymbol{\mu}$-variables that puts all mass in a single

configuration, for each factor $\alpha \in \mathcal{F} \cup \mathcal{H}$. That is, we have for each $i \in \mathcal{N}(\alpha)$:

$$\begin{aligned} \max_{\boldsymbol{\mu}_i^\alpha} \|\boldsymbol{\mu}_i^\alpha - |\mathcal{Y}_i|^{-1}\mathbf{1}\|^2 &= \left((1-|\mathcal{Y}_i|^{-1})^2 + (|\mathcal{Y}_i|-1)|\mathcal{Y}_i|^{-2}\right) \\ &= 1 - |\mathcal{Y}_i|^{-1}. \end{aligned} \tag{D.30}$$

Hence, the constant C becomes:

$$\begin{aligned} C &= \frac{5\eta}{2}\sum_\alpha \sum_i \max_{\boldsymbol{\mu}_i^\alpha} \|\boldsymbol{\mu}_i^\alpha - |\mathcal{Y}_i|^{-1}\mathbf{1}\|^2 + \frac{5}{2\eta}\|\boldsymbol{\lambda}^*\|^2 \\ &= \frac{5\eta}{2}\sum_i \deg(i)(1-|\mathcal{Y}_i|^{-1}) + \frac{5}{2\eta}\|\boldsymbol{\lambda}^*\|^2. \end{aligned} \tag{D.31}$$

■

A consequence of Proposition D.4 is that, after $T = O(1/\epsilon)$ iterations of AD^3, we have a dual solution which yields an objective value ϵ-close to the optimal value. Asymptotically, this is a better bound than the $O(1/\epsilon^2)$ iteration bound of the subgradient-based dual decomposition algorithm of Komodakis et al. (2007), and is equivalent to the $O(1/\epsilon)$ bound achieved by the accelerated method of Jojic et al. (2010), despite the fact that the latter requires specifying ϵ beforehand to set a "temperature" parameter. The bounds derived so far for all these algorithms are with respect to the *dual* problem—an open problem is to obtain bounds related to primal convergence.

Appendix E

Derivation of Solutions for Quadratic Problems in AD^3

E.1 Binary pairwise factors

In this section, we derive in detail the closed form solution of problem Eq. 6.26 for binary pairwise factors. Recall that, if α is a pairwise factor (*i.e.*, if $N(\alpha) = 2$), then the marginal polytope $\mathrm{MARG}(\mathscr{G}|_\alpha)$ is given by:

$$\mathrm{MARG}(\mathscr{G}|_\alpha) = \left\{ (\mu_1(.), \mu_2(.), \mu_{12}(.)) \;\middle|\; \begin{array}{ll} \sum_{y_1 \in \mathcal{Y}_1} \mu_1(y_1) = 1 & \\ \sum_{y_2 \in \mathcal{Y}_2} \mu_2(y_2) = 1 & \\ \mu_1(y_1) = \sum_{y_2 \in \mathcal{Y}_2} \mu_{12}(y_1, y_2), & \forall y_1 \in \mathcal{Y}_1 \\ \mu_2(y_2) = \sum_{y_1 \in \mathcal{Y}_1} \mu_{12}(y_1, y_2), & \forall y_2 \in \mathcal{Y}_2 \\ \mu_{12}(y_1, y_2) \geq 0, & \forall y_1 \in \mathcal{Y}_1, y_2 \in \mathcal{Y}_1 \end{array} \right\}. \quad \text{(E.1)}$$

If besides being pairwise, factor α is also binary (*i.e.*, if $\mathcal{Y}_1 = \mathcal{Y}_2 = \{0,1\}$), then we may reparameterize our problem by introducing new variables[1] $z_1 := \mu_1(1)$, $z_2 := \mu_2(1)$, and $z_{12} := \mu_{12}(1,1)$. Noting that $\boldsymbol{\mu}_1 = (1 - z_1, z_1)$, $\boldsymbol{\mu}_2 = (1 - z_2, z_2)$, and $\boldsymbol{\mu}_{12} = (1 - z_1 - z_2 + z_{12}, z_1 - z_{12}, z_2 - z_{12}, z_{12})$, the problem in Eq. 6.26 becomes:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\left[\left(1 - z_1 - \frac{\omega_1(0)}{\eta}\right)^2 + \left(z_1 - \frac{\omega_1(1)}{\eta}\right)^2 + \left(1 - z_2 - \frac{\omega_2(0)}{\eta}\right)^2 + \left(z_2 - \frac{\omega_2(1)}{\eta}\right)^2\right] \\ & -\eta^{-1}\left[\theta_{12}(00)(1 - z_1 - z_2 + z_{12}) + \theta_{12}(10)(z_1 - z_{12}) + \theta_{12}(01)(z_2 - z_{12}) + \theta_{12}(11) z_{12}\right] \\ \text{w.r.t.} \quad & z_1, z_2, z_{12} \in [0,1]^3 \\ \text{s.t.} \quad & z_{12} \leq z_1, \quad z_{12} \leq z_2, \quad z_{12} \geq z_1 + z_2 - 1, \quad (z_1, z_2, z_{12}). \end{aligned} \quad \text{(E.2)}$$

or, multiplying the objective by the constant $\frac{1}{2}$:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}(z_1 - c_1)^2 + \frac{1}{2}(z_2 - c_2)^2 - c_{12} z_{12} \\ \text{w.r.t.} \quad & z_1, z_2, z_{12} \in [0,1]^3 \\ \text{s.t.} \quad & z_{12} \leq z_1, \quad z_{12} \leq z_2, \quad z_{12} \geq z_1 + z_2 - 1, \end{aligned} \quad \text{(E.3)}$$

[1] Cf. Example 5.1 and Eq. 5.14, where the same reparametrization was made.

where we have substituted

$$
\begin{aligned}
c_1 &= (\eta^{-1}\omega_1(1) + 1 - \eta^{-1}\omega_1(0) - \eta^{-1}\theta_{12}(00) + \eta^{-1}\theta_{12}(10))/2 && \text{(E.4)}\\
c_2 &= (\eta^{-1}\omega_2(1) + 1 - \eta^{-1}\omega_2(0) - \eta^{-1}\theta_{12}(00) + \eta^{-1}\theta_{12}(01))/2 && \text{(E.5)}\\
c_{12} &= (\eta^{-1}\theta_{12}(00) - \eta^{-1}\theta_{12}(10) - \eta^{-1}\theta_{12}(01) + \eta^{-1}\theta_{12}(11))/2. && \text{(E.6)}
\end{aligned}
$$

Now, notice that in (E.3) we can assume $c_{12} \geq 0$ without loss of generality—indeed, if $c_{12} < 0$, we recover this case by redefining $c_1' = c_1 + c_{12}$, $c_2' = 1 - c_2$, $c_{12}' = -c_{12}$, $z_2' = 1 - z_2$, $z_{12}' = z_1 - z_{12}$. Thus, assuming that $c_{12} \geq 0$, the lower bound constraints $z_{12} \geq z_1 + z_2 - 1$ and $z_{12} \geq 0$ are always inactive and can be ignored. Hence, (E.3) can be simplified to:

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{2}(z_1 - c_1)^2 + \frac{1}{2}(z_2 - c_2)^2 - c_{12}z_{12} \\
\text{w.r.t.} \quad & z_1, z_2, z_{12} \\
\text{s.t.} \quad & z_{12} \leq z_1, \quad z_{12} \leq z_2, \quad z_1 \in [0,1], \quad z_2 \in [0,1]. && \text{(E.7)}
\end{aligned}
$$

We next establish a closed form solution for this problem, proving Proposition 6.5. Let $[x]_{\mathbb{U}} := \min\{\max\{x, 0\}, 1\}$ denote projection (clipping) onto the unit interval $\mathbb{U} := [0,1]$. Note that if $c_{12} = 0$, the problem becomes separable, and a solution is

$$
z_1^* = [c_1]_{\mathbb{U}}, \quad z_2^* = [c_2]_{\mathbb{U}}, \quad z_{12}^* = \min\{z_1^*, z_2^*\}, \tag{E.8}
$$

which complies with Eq. 6.33. We next analyze the case where $c_{12} > 0$. The Lagrangian function of (E.7) is:

$$
\begin{aligned}
L(\boldsymbol{z}, \boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \;=\; & \frac{1}{2}(z_1 - c_1)^2 + \frac{1}{2}(z_2 - c_2)^2 - c_{12}z_{12} + \mu_1(z_{12} - z_1) + \mu_2(z_{12} - z_2) \\
& - \lambda_1 z_1 - \lambda_2 z_2 + \nu_1(z_1 - 1) + \nu_2(z_2 - 1). && \text{(E.9)}
\end{aligned}
$$

At optimality, the following KKT conditions need to be satisfied:

$$
\begin{aligned}
\nabla_{z_1} L(\boldsymbol{z}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = 0 \quad &\Rightarrow \quad z_1^* = c_1 + \mu_1^* + \lambda_1^* - \nu_1^* && \text{(E.10)}\\
\nabla_{z_2} L(\boldsymbol{z}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = 0 \quad &\Rightarrow \quad z_2^* = c_2 + \mu_2^* + \lambda_2^* - \nu_2^* && \text{(E.11)}\\
\nabla_{z_{12}} L(\boldsymbol{z}^*, \boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = 0 \quad &\Rightarrow \quad c_{12} = \mu_1^* + \mu_2^* && \text{(E.12)}\\
\lambda_1^* z_1^* = 0 & && \text{(E.13)}\\
\lambda_2^* z_2^* = 0 & && \text{(E.14)}\\
\mu_1^*(z_{12}^* - z_1^*) = 0 & && \text{(E.15)}\\
\mu_2^*(z_{12}^* - z_2^*) = 0 & && \text{(E.16)}\\
\nu_1^*(z_1^* - 1) = 0 & && \text{(E.17)}\\
\nu_2^*(z_2^* - 1) = 0 & && \text{(E.18)}\\
\boldsymbol{\mu}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^* \geq 0 & && \text{(E.19)}\\
z_{12}^* \leq z_1^*, \quad z_{12}^* \leq z_2^*, \quad z_1^* \in [0,1], \quad z_2^* \in [0,1] & && \text{(E.20)}
\end{aligned}
$$

We are going to consider three cases separately:

1. $\boxed{z_1^* > z_2^*}$

From the primal feasibility conditions (E.20), this implies $z_1^* > 0$, $z_2^* < 1$, and $z_{12}^* < z_1^*$. Complementary slackness (E.13,E.18,E.15) implies in turn $\lambda_1^* = 0$, $\nu_2^* = 0$, and $\mu_1^* = 0$. From (E.12) we have $\mu_2^* = c_{12}$. Since we are assuming $c_{12} > 0$, we then have $\mu_2^* > 0$, and complementary slackness (E.16) implies $z_{12}^* = z_2^*$.

Plugging the above into (E.10)–(E.11) we obtain

$$z_1^* = c_1 - \nu_1^* \leq c_1, \qquad z_2^* = c_2 + \lambda_2^* + c_{12} \geq c_2 + c_{12}. \tag{E.21}$$

Now we have the following:

- Either $z_1^* = 1$ or $z_1^* < 1$. In the latter case, $\nu_1^* = 0$ by complementary slackness (E.17), hence $z_1^* = c_1$. Since in any case we must have $z_1^* \leq c_1$, we conclude that $z_1^* = \min\{c_1, 1\}$.
- Either $z_2^* = 0$ or $z_2^* > 0$. In the latter case, $\lambda_2^* = 0$ by complementary slackness (E.14), hence $z_2^* = c_2 + c_{12}$. Since in any case we must have $z_2^* \geq \lambda_2$, we conclude that $z_2^* = \max\{0, c_2 + c_{12}\}$.

In sum:

$$z_1^* = \min\{c_1, 1\}, \quad z_{12}^* = z_2^* = \max\{0, c_2 + c_{12}\}, \tag{E.22}$$

and our assumption $z_1^* > z_2^*$ can only be valid if $c_1 > c_2 + c_{12}$.

2. $\boxed{z_1^* < z_2^*}$

By symmetry, we have

$$z_2^* = \min\{c_2, 1\}, \quad z_{12}^* = z_1^* = \max\{0, c_1 + c_{12}\}, \tag{E.23}$$

and our assumption $z_1^* < z_2^*$ can only be valid if $c_2 > c_1 + c_{12}$.

3. $\boxed{z_1^* = z_2^*}$

In this case, it is easy to verify that we must have $z_{12}^* = z_1^* = z_2^*$, and we can rewrite our optimization problem in terms of one variable only (call it z). The problem becomes that of minimizing $\frac{1}{2}(z - c_1)^2 + \frac{1}{2}(z - c_2)^2 - c_{12}z$, which equals a constant plus $(z - \frac{c_1+c_2+c_{12}}{2})^2$, subject to $z \in \mathbb{U} \triangleq [0, 1]$. Hence:

$$z_{12}^* = z_1^* = z_2^* = \left[\frac{c_1 + c_2 + c_{12}}{2}\right]_{\mathbb{U}}. \tag{E.24}$$

Putting all the pieces together, we obtain the solution displayed in Eq. 6.33.

E.2 Hard constraint factors

E.2.1 Sifting Lemma

Some of the procedures that we will derive in the sequel make use of a simple *sifting* technique, in which a projection is computed by evaluating one or more projections onto larger sets. The following lemma is the key to those procedures.

Algorithm 18 Dykstra's algorithm for projecting onto $\bigcap_{j=1}^{J} \mathcal{C}_j$

Input: Point $\boldsymbol{x}_0 \in \mathbb{R}^D$, convex sets $\mathcal{C}_1, \ldots, \mathcal{C}_J$
Initialize $\boldsymbol{x}^{(0)} = \boldsymbol{x}_0$, $\boldsymbol{u}_j^{(0)} = \mathbf{0}$ for all $j = 1, \ldots, J$
$t \leftarrow 1$
repeat
  for $j = 1$ **to** J **do**
    Set $s = j + (t-1)J$
    Set $\tilde{\boldsymbol{x}}_0 = \boldsymbol{x}^{(s-1)} - \boldsymbol{u}_j^{(t-1)}$
    Set $\boldsymbol{x}^{(s)} = \text{proj}_{\mathcal{C}_j}(\tilde{\boldsymbol{x}}_0)$, and $\boldsymbol{u}_j^{(t)} = \boldsymbol{x}^{(s)} - \tilde{\boldsymbol{x}}_0$
  end for
  $t \leftarrow t + 1$
until convergence.
Output: $\boldsymbol{x}$

Lemma E.1 (Sifting Lemma.) *Consider a problem of the form*

$$P: \qquad \min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x}) \quad s.t. \quad g(\boldsymbol{x}) \leq 0, \tag{E.25}$$

where $\mathcal{X}$ is nonempty convex subset of $\mathbb{R}^D$ and $f : \mathcal{X} \to \mathbb{R}$ and $g : \mathcal{X} \to \mathbb{R}$ are convex functions. Suppose that the problem (E.25) is feasible and bounded below, and let $\mathcal{A}$ be the set of solutions of the relaxed problem $\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$, *i.e.* $\mathcal{A} = \{\boldsymbol{x} \in \mathcal{X} \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}'),\ \forall \boldsymbol{x}' \in \mathcal{X}\}$. *Then:*

1. *if for some $\tilde{\boldsymbol{x}} \in \mathcal{A}$ we have $g(\tilde{\boldsymbol{x}}) \leq 0$, then $\tilde{\boldsymbol{x}}$ is also a solution of the original problem P;*
2. *otherwise (if for all $\tilde{\boldsymbol{x}} \in \mathcal{A}$ we have $g(\tilde{\boldsymbol{x}}) > 0$), then the inequality constraint is necessarily active in P, i.e., problem P is equivalent to* $\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$ *s.t.* $g(\boldsymbol{x}) = 0$.

Proof. Let f^* be the optimal value of P. The first statement is obvious: since $\tilde{\boldsymbol{x}}$ is a solution of a relaxed problem we have $f(\tilde{\boldsymbol{x}}) \leq f^*$; hence if $\tilde{\boldsymbol{x}}$ is feasible this becomes an equality. For the second statement, assume that $\exists \boldsymbol{x} \in \mathcal{X}$ s.t. $g(\boldsymbol{x}) < 0$ (otherwise, the statement holds trivially). The nonlinear Farkas' lemma (Bertsekas et al., 2003, Prop. 3.5.4, p. 204) implies that there exists some $\lambda^* \geq 0$ s.t. $f(\boldsymbol{x}) - f^* + \lambda^* g(\boldsymbol{x}) \geq 0$ holds for all $\boldsymbol{x} \in \mathcal{X}$. In particular, this also holds for an optimal $\boldsymbol{x}^*$ (i.e., such that $f^* = f(\boldsymbol{x}^*)$), which implies that $\lambda^* g(\boldsymbol{x}^*) \geq 0$. However, since $\lambda^* \geq 0$ and $g(\boldsymbol{x}^*) \leq 0$ (since $\boldsymbol{x}^*$ has to be feasible), we also have $\lambda^* g(\boldsymbol{x}^*) \leq 0$, *i.e.*, $\lambda^* g(\boldsymbol{x}^*) = 0$. Now suppose that $\lambda^* = 0$. Then we have $f(\boldsymbol{x}) - f^* \geq 0$, $\forall \boldsymbol{x} \in \mathcal{X}$, which implies that $\boldsymbol{x}^* \in \mathcal{A}$ and contradicts the assumption that $g(\tilde{\boldsymbol{x}}) > 0, \forall \tilde{\boldsymbol{x}} \in \mathcal{A}$. Hence we must have $g(\boldsymbol{x}^*) = 0$. ∎

E.2.2 OR-with-output Factor

We have presented an algorithm for solving the local subproblem for the OR-with-output factor which requires, as an intermediate step, projecting onto the set $\mathcal{A}_0$. For the last projection, we suggested Procedure 6.1. We now prove its correctness.

Proposition E.2 *Procedure 6.1 is correct.*

Proof. The proof is divided into the following parts:

1. We show that Procedure 6.1 corresponds to the first iteration of Dykstra's projection algorithm (Boyle and Dykstra, 1986) applied to sets $\mathcal{A}_1$ and $[0,1]^{K+1}$;

2. We show that Dykstra's converges in one iteration if a specific condition is met;

3. We show that with the two sets above that condition is met.

The first part is rather trivial. Dykstra's algorithm is shown as Algorithm 18; when $J = 2$, $\mathcal{C}_1 = \mathcal{A}_1$ and $\mathcal{C}_2 = [0,1]^{K+1}$, and noting that $\boldsymbol{u}_1^{(1)} = \boldsymbol{u}_2^{(1)} = 0$, we have that the first iteration becomes Procedure 6.1.

We turn to the second part, to show that, when $J = 2$, the fact that $\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)}$ implies that $\boldsymbol{x}^{(s)} = \boldsymbol{x}^{(2)}$, $\forall s > 3$. In words, if at the second iteration t of Dijkstra's, the value of $\boldsymbol{x}$ does not change after computing the first projection, then it will never change, so the algorithm has converged and $\boldsymbol{x}$ is the desired projection. To see that, consider the moment in Algorithm 18 when $t = 2$ and $j = 1$. After the projection, we update $\boldsymbol{u}_1^{(2)} = \boldsymbol{x}^{(3)} - (\boldsymbol{x}^{(2)} - \boldsymbol{u}_1^{(1)})$, which when $\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)}$ equals $\boldsymbol{u}_1^{(1)}$, *i.e.*, $\boldsymbol{u}_1$ keeps unchanged. Then, when $t = 2$ and $j = 2$, one first computes $\tilde{\boldsymbol{x}}_0 = \boldsymbol{x}^{(3)} - \boldsymbol{u}_2^{(1)} = \boldsymbol{x}^{(3)} - (\boldsymbol{x}^{(2)} - \boldsymbol{x}_0) = \boldsymbol{x}_0$, *i.e.*, the projection is the same as the one already computed at $t = 1$, $j = 2$. Hence the result is the same, *i.e.*, $\boldsymbol{x}^{(4)} = \boldsymbol{x}^{(2)}$, and similarly $\boldsymbol{u}_2^{(2)} = \boldsymbol{u}_2^{(1)}$. Since neither $\boldsymbol{x}$, $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ changed in the second iteration, and subsequent iterations only depend on these values, we have that $\boldsymbol{x}$ will never change afterwards.

Finally, we are going to see that, regardless of the choice of $\boldsymbol{z}_0$ in Procedure 6.1 ($\boldsymbol{x}_0$ in Algorithm 18) we always have $\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)}$. Looking at Algorithm 10, we see that after $t = 1$:

$$x_k^{(1)} = \begin{cases} \tau, & \text{if } k = K+1 \text{ or } x_{0k} \geq \tau \\ x_{0k}, & \text{otherwise,} \end{cases} \qquad u_{1k}^{(1)} = \begin{cases} \tau - x_{0k}, & \text{if } k = K+1 \text{ or } x_{0k} \geq \tau \\ 0, & \text{otherwise,} \end{cases}$$

$$x_k^{(2)} = [x_k^{(1)}]_{\mathbb{U}} = \begin{cases} [\tau]_{\mathbb{U}}, & \text{if } k = K+1 \text{ or } x_{0k} \geq \tau \\ [x_{0k}]_{\mathbb{U}}, & \text{otherwise.} \end{cases} \tag{E.26}$$

Hence in the beginning of the second iteration ($t = 2$, $j = 1$), we have

$$\tilde{x}_{0k} = x_k^{(2)} - u_{1k}^{(1)} = \begin{cases} [\tau]_{\mathbb{U}} - \tau + x_{0k}, & \text{if } k = K+1 \text{ or } x_{0k} \geq \tau \\ [x_{0k}]_{\mathbb{U}}, & \text{otherwise.} \end{cases} \tag{E.27}$$

Now two things should be noted about Algorithm 10:

- If a constant is added to all entries in $\boldsymbol{z}_0$, the set $\mathcal{I}(z_{K+1})$ remains the same, and τ and $\boldsymbol{z}$ are affected by the same constant;

- Let $\boldsymbol{z}_0'$ be such that $z_{0k}' = z_{0k}$ if $k = K+1$ or $z_{0k} \geq \tau$, and $z_{0k}' \leq \tau$ otherwise. Let $\boldsymbol{z}'$ be the projected point when such $\boldsymbol{z}_0'$ is given as input. Then $\mathcal{I}(z_{K+1}') = \mathcal{I}(z_{K+1})$, $\tau' = \tau$, $z_k' = z_k$ if $k = K+1$ or $z_{0k} \geq \tau$, and $z_k' = z_{0k}'$ otherwise.

The two facts above allow to relate the projection of $\tilde{\boldsymbol{x}}_0$ (in the second iteration) with that of $\boldsymbol{x}_0$ (in the first iteration). Using $[\tau]_{\mathbb{U}} - \tau$ as the constant, and noting that, for $k \neq K+1$ and $x_{0k} < \tau$, we have $[x_{0k}]_{\mathbb{U}} - [\tau]_{\mathbb{U}} + \tau \geq \tau$ if $x_{0k} < \tau$, the two facts imply that:

$$x_k^{(3)} = \begin{cases} x_k^{(1)} + [\tau]_{\mathbb{U}} - \tau = [\tau]_{\mathbb{U}}, & \text{if } k = K+1 \text{ or } x_{0k} \geq \tau \\ [x_{0k}]_{\mathbb{U}}, & \text{otherwise;} \end{cases} \tag{E.28}$$

hence $\boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)}$, which concludes the proof. ■

E.3 Proof of Proposition 6.6

We first show that the rank of the matrix $\mathbf{M}$ is at most $\sum_{i \in \mathcal{N}(\alpha)} |\mathcal{Y}_i| - \mathcal{N}(\alpha) + 1$. For each $i \in \mathcal{N}(\alpha)$, let us consider the $|\mathcal{Y}_i|$ rows of $\mathbf{M}$. By definition of $\mathbf{M}$, the set of entries on these rows which have the value 1 form a partition of $\mathcal{Y}_\alpha$, hence, summing these rows yields the all-ones row vector, and this happens for each $i \in \mathcal{N}(\alpha)$. Hence we have at least $\mathcal{N}(\alpha) - 1$ rows that are linearly dependent. This shows that the rank of $\mathbf{M}$ is at most $\sum_{i \in \mathcal{N}(\alpha)} |\mathcal{Y}_i| - \mathcal{N}(\alpha) + 1$. Let us now rewrite (6.44) as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\boldsymbol{u} - \boldsymbol{a}\|^2 + g(\boldsymbol{u}) \\ \text{with respect to} \quad & \boldsymbol{u} \in \mathbb{R}^{\sum_i |\mathcal{Y}_i|}, \end{aligned} \tag{E.29}$$

where $g(\boldsymbol{u})$ is the solution value of the following linear problem:

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{b}^\top \boldsymbol{v} \\ \text{with respect to} \quad & \boldsymbol{v} \in \mathbb{R}^{|\mathcal{Y}_\alpha|} \\ \text{subject to} \quad & \begin{cases} \mathbf{M}\boldsymbol{v} = \boldsymbol{u} \\ \mathbf{1}^\top \boldsymbol{v} = 1 \\ \boldsymbol{v} \geq 0. \end{cases} \end{aligned} \tag{E.30}$$

From the simplex constraints (last two lines), we have that problem (E.30) is bounded below (i.e., $g(\boldsymbol{u}) > -\infty$). Furthermore, problem (E.30) is feasible (i.e., $g(\boldsymbol{u}) < +\infty$) if and only if $\boldsymbol{u}$ satisfies the non-negativity and normalization constraints for every $i \in \mathcal{N}(\alpha)$:

$$\sum_{y_i} u_i(y_i) = 1, \quad u_i(y_i) \geq 0, \ \forall y_i. \tag{E.31}$$

Those constraints imply $\mathbf{1}^\top \boldsymbol{v} = 1$. Hence we can add the constraints (E.31) to the problem in (E.29), discard the constraint $\mathbf{1}^\top \boldsymbol{v} = 1$ in (E.30), and assume that the resulting problem (which we reproduce below) is feasible and bounded below:

$$\begin{aligned} \text{minimize} \quad & -\boldsymbol{b}^\top \boldsymbol{v} \\ \text{with respect to} \quad & \boldsymbol{v} \in \mathbb{R}^{|\mathcal{Y}_\alpha|} \\ \text{subject to} \quad & \begin{cases} \mathbf{M}\boldsymbol{v} = \boldsymbol{u} \\ \boldsymbol{v} \geq 0. \end{cases} \end{aligned} \tag{E.32}$$

Problem (E.32) is a linear program in standard form. Since it is feasible and bounded, it admits a solution at a vertex of the constraint set. We have that a point $\widehat{\boldsymbol{v}}$ is a vertex if and only if the columns of $\mathbf{M}$ indexed by $\{\boldsymbol{y}_\alpha \mid v_\alpha(\boldsymbol{y}_\alpha) \neq 0\}$ are linearly independent. We cannot have more than $\sum_{i \in \mathcal{N}(\alpha)} |\mathcal{Y}_i| - \mathcal{N}(\alpha) + 1$ of these columns, since this is the rank of $\mathbf{M}$. It follows that (E.32) (and hence (6.44)) has a solution $\boldsymbol{v}^*$ with at most $\sum_{i \in \mathcal{N}(\alpha)} |\mathcal{Y}_i| - \mathcal{N}(\alpha) + 1$ nonzeros.

Appendix F

Proofs for Online Proximal Gradient Algorithms

F.1 Proof of Proposition 9.1

Let $\mathcal{H} = \mathcal{H}_1 \oplus \cdots \oplus \mathcal{H}_M$. We have respectively:

$$\begin{aligned}
M_\varphi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M) &= \min_{\boldsymbol{y}} \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{x}\|_{\mathcal{H}}^2 + \varphi(\boldsymbol{y}) \\
&= \min_{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M} \frac{1}{2}\sum_{k=1}^{M} \|\boldsymbol{y}_k - \boldsymbol{x}_k\|_{\mathcal{H}_k}^2 + \psi(\|\boldsymbol{y}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{y}_M\|_{\mathcal{H}_M}) \\
&= \min_{\boldsymbol{u} \in \mathbb{R}_+^M} \left(\psi(u_1, \ldots, u_M) + \frac{1}{2}\sum_{k=1}^{M} \min_{\boldsymbol{y}_k : \|\boldsymbol{y}_k\|_{\mathcal{H}_k} = u_k} \|\boldsymbol{y}_k - \boldsymbol{x}_k\|_{\mathcal{H}_k}^2 \right) \\
&= \min_{\boldsymbol{u} \in \mathbb{R}_+^M} \left(\psi(u_1, \ldots, u_M) + \frac{1}{2}\sum_{k=1}^{M} \min_{\boldsymbol{y}_k} \left\| \frac{u_k \boldsymbol{y}_k}{\|\boldsymbol{y}_k\|_{\mathcal{H}_k}} - \boldsymbol{x}_k \right\|_{\mathcal{H}_k}^2 \right) \quad (*) \\
&= \min_{\boldsymbol{u} \in \mathbb{R}_+^M} \left(\psi(u_1, \ldots, u_M) + \frac{1}{2}\sum_{k=1}^{M} (u_k - \|\boldsymbol{x}_k\|_{\mathcal{H}_k})^2 \right) \\
&= M_\psi(\|\boldsymbol{x}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{x}_M\|_{\mathcal{H}_M}),
\end{aligned} \tag{F.1}$$

where the solution of the innermost minimization problem in $(*)$ is $\boldsymbol{y}_k = \frac{u_k}{\|\boldsymbol{x}_k\|}\boldsymbol{x}_k$. This can be verified by noting that

$$\begin{aligned}
\left\| \frac{u_k \boldsymbol{y}_k}{\|\boldsymbol{y}_k\|_{\mathcal{H}_k}} - \boldsymbol{x}_k \right\|_{\mathcal{H}_k}^2 &= u_k^2 + \|\boldsymbol{x}_k\|_{\mathcal{H}_k}^2 - \frac{2u_k}{\|\boldsymbol{y}_k\|_{\mathcal{H}_k}} \langle \boldsymbol{x}_k, \boldsymbol{y}_k \rangle_{\mathcal{H}_k} \\
&\leq u_k^2 + \|\boldsymbol{x}_k\|_{\mathcal{H}_k}^2 - 2u_k \|\boldsymbol{x}_k\|_{\mathcal{H}_k} \\
&= (u_k - \|\boldsymbol{x}_k\|_{\mathcal{H}_k})^2,
\end{aligned} \tag{F.2}$$

due to Cauchy-Schwarz inequality, with equality iff $\boldsymbol{y}_k = u_k \boldsymbol{x}_k / \|\boldsymbol{x}_k\|$. Therefore:

$$[\mathrm{prox}_\varphi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M)]_k = [\mathrm{prox}_\psi(\|\boldsymbol{x}_1\|_{\mathcal{H}_1}, \ldots, \|\boldsymbol{x}_M\|_{\mathcal{H}_M})]_k \frac{\boldsymbol{x}_k}{\|\boldsymbol{x}_k\|_{\mathcal{H}_k}}. \tag{F.3}$$

F.2 Proof of Corollary 9.3

Let $\check{\boldsymbol{x}} = \mathrm{prox}_{\varphi*}(\boldsymbol{x})$. From (9.14)–(9.15), we have that

$$\begin{aligned}
\frac{1}{2}\|\boldsymbol{x}\|^2 &= \frac{1}{2}\|\bar{\boldsymbol{x}}-\boldsymbol{x}\|^2 + \varphi(\bar{\boldsymbol{x}}) + \frac{1}{2}\|\check{\boldsymbol{x}}-\boldsymbol{x}\|^2 + \varphi^*(\check{\boldsymbol{x}}) \\
&= \frac{1}{2}\|\bar{\boldsymbol{x}}-\boldsymbol{x}\|^2 + \varphi(\bar{\boldsymbol{x}}) + \frac{1}{2}\|\bar{\boldsymbol{x}}\|^2 + \varphi^*(\boldsymbol{x}-\bar{\boldsymbol{x}}) \\
&= \frac{1}{2}\|\bar{\boldsymbol{x}}-\boldsymbol{x}\|^2 + \varphi(\bar{\boldsymbol{x}}) + \frac{1}{2}\|\bar{\boldsymbol{x}}\|^2 + \sup_{\boldsymbol{u}\in\mathcal{H}} \left(\langle \boldsymbol{u}, \boldsymbol{x}-\bar{\boldsymbol{x}}\rangle - \varphi(\boldsymbol{u})\right) \\
&\geq \frac{1}{2}\|\bar{\boldsymbol{x}}-\boldsymbol{x}\|^2 + \varphi(\bar{\boldsymbol{x}}) + \frac{1}{2}\|\bar{\boldsymbol{x}}\|^2 + \langle \boldsymbol{y}, \boldsymbol{x}-\bar{\boldsymbol{x}}\rangle - \varphi(\boldsymbol{y}) \\
&= \frac{1}{2}\|\boldsymbol{x}\|^2 + \langle \bar{\boldsymbol{x}}-\boldsymbol{y}, \bar{\boldsymbol{x}}-\boldsymbol{x}\rangle - \varphi(\boldsymbol{y}) + \varphi(\bar{\boldsymbol{x}}),
\end{aligned} \tag{F.4}$$

whence

$$\langle \bar{\boldsymbol{x}}-\boldsymbol{y}, \bar{\boldsymbol{x}}-\boldsymbol{x}\rangle \leq \varphi(\boldsymbol{y}) - \varphi(\bar{\boldsymbol{x}}). \tag{F.5}$$

Now note that the left hand side of (F.5) can be written as:

$$\langle \bar{\boldsymbol{x}}-\boldsymbol{y}, \bar{\boldsymbol{x}}-\boldsymbol{x}\rangle = \frac{1}{2}\|\boldsymbol{y}-\bar{\boldsymbol{x}}\|^2 - \frac{1}{2}\|\boldsymbol{y}-\boldsymbol{x}\|^2 + \frac{1}{2}\|\bar{\boldsymbol{x}}-\boldsymbol{x}\|^2.$$

This concludes the proof.

F.3 Proof of Proposition 9.4

We first show that, when $\mathcal{H} = \mathbb{R}^M$, the family $\mathcal{R}'_{\text{norms}} = \{\|.\|_p^q \mid p, q \geq 1\}$ is strongly co-shrinking. We know that, for any $\sigma > 0$ and $p, q \geq 1$, the $\sigma\|.\|_p^q$-proximity operator shrinks each component, *i.e.*, given any $\boldsymbol{b} = (b_1, \ldots, b_m) \in \mathbb{R}^M$, we have $|b_m| \geq |[\mathrm{prox}_{\sigma\|.\|_p^q}(\boldsymbol{b})]_m|$ for $m = 1, \ldots, M$. Let $p', q' \geq 1$. We use the fact that the function $x \mapsto x^\alpha$ is increasing in $\mathbb{R}_+$ for any $\alpha > 0$. Summing the M inequalities above, after raising each of them to the power of p', we obtain $\sum_{m=1}^M |b_m|^{p'} \geq \sum_{m=1}^M |[\mathrm{prox}_{\sigma\|.\|_p^q}(\boldsymbol{b})]_m|^{p'}$. Raising both sides to the power of q'/p' (which is > 0) and scaling by $\sigma' > 0$, we obtain $\sigma'\|\boldsymbol{b}\|_{p'}^{q'} \geq \sigma'\|\mathrm{prox}_{\sigma\|.\|_p^q}(\boldsymbol{b})\|_{p'}^{q'}$, which proves the desired fact.

Now, invoking Proposition 9.1, we can generalize the above result for the structured block case, *i.e.*, for $\mathcal{R}''_{\text{norms}} = \{\|.\|_{2,p}^q \mid p, q \geq 1\}$. We also have that the $\sigma\|.\|_{2,p}^q$-proximity operator shrinks each block norm, *i.e.*, given any $\boldsymbol{w} \in \mathcal{H}$, we have $\|\boldsymbol{w}_m\| \geq \|[\mathrm{prox}_{\sigma\|.\|_{2,p}^q}(\boldsymbol{w})]_m\|$ for $m = 1, \ldots, M$ (here the norms are those induced by each RKHS $\mathcal{H}_m$). Hence, we can anagously derive the fact that $\mathcal{R}''_{\text{norms}}$ is co-shrinking.

Finally, we consider the broader case of the family $\mathcal{R}_{\text{norms}} \supseteq \mathcal{R}''_{\text{norms}}$, which involves the subsets $\mathcal{B} \subseteq \{1, \ldots, M\}$. Clearly, the proximity operator associated with the function $\boldsymbol{w} \mapsto \sigma\|\boldsymbol{w}_\mathcal{B}\|_{2,p}^q$ is still shrinking for any $\mathcal{B}$: it can be easily verified that $[\mathrm{prox}_{\sigma\|.\|_{2,p}^q}(\boldsymbol{w})]_m = [\mathrm{prox}_{\sigma\|.\|_{2,p}^q}(\boldsymbol{w})]_m$ if $m \in \mathcal{B}$, and $[\mathrm{prox}_{\sigma\|.\|_{2,p}^q}(\boldsymbol{w})]_m = \boldsymbol{w}_m$ otherwise. Letting $\mathcal{B}' \subseteq \{1, \ldots, M\}$ and summing $|\mathcal{B}'|$ inequalities, we obtain $\sum_{m\in\mathcal{B}'} \|\boldsymbol{w}_m\|^{p'} \geq \sum_{m\in\mathcal{B}'} \|[\mathrm{prox}_{\sigma\|.\|_{2,p}^q}(\boldsymbol{w})]_m\|^{p'}$, which implies analagously that $\mathcal{R}_{\text{norms}}$ also has the strong co-shrinkage property.

F.4 Proof of Proposition 9.5

Let $\bar{\boldsymbol{w}}_1, \ldots, \bar{\boldsymbol{w}}_J$ the sequence of points obtained after applying each proximity operator, *i.e.*, $\bar{\boldsymbol{w}}_j = \operatorname{prox}_{\Omega_j}(\bar{\boldsymbol{w}}_{j-1})$ for $j = 1, \ldots, J$, where we define $\bar{\boldsymbol{w}}_0 \triangleq \boldsymbol{w}$ and $\bar{\boldsymbol{w}}_J \triangleq \bar{\boldsymbol{w}}$. From Corollary 9.3, we have

$$\frac{\|\boldsymbol{\xi} - \bar{\boldsymbol{w}}_j\|^2}{2} - \frac{\|\boldsymbol{\xi} - \bar{\boldsymbol{w}}_{j-1}\|^2}{2} + \frac{\|\bar{\boldsymbol{w}}_{j-1} - \bar{\boldsymbol{w}}_j\|^2}{2} \leq \Omega_j(\boldsymbol{\xi}) - \Omega_j(\bar{\boldsymbol{w}}_j). \tag{F.6}$$

Summing these inequalities for $j = 1, \ldots, J$ and applying the telescopic property, we obtain

$$\begin{aligned} \frac{\|\boldsymbol{\xi} - \bar{\boldsymbol{w}}\|^2}{2} - \frac{\|\boldsymbol{\xi} - \boldsymbol{w}\|^2}{2} + \sum_{j=1}^{J} \frac{\|\bar{\boldsymbol{w}}_{j-1} - \bar{\boldsymbol{w}}_j\|^2}{2} &\leq \Omega(\boldsymbol{\xi}) - \sum_{j=1}^{J} \Omega_j(\bar{\boldsymbol{w}}_j) \\ &\leq \Omega(\boldsymbol{\xi}) - \Omega(\bar{\boldsymbol{w}}), \end{aligned} \tag{F.7}$$

where the last inequality follows from the co-shrinkage property of $\Omega_1, \ldots, \Omega_J$. Finally, from the convexity of the norm, we have that

$$\frac{J}{2} \sum_{j=1}^{J} \frac{1}{J} \|\bar{\boldsymbol{w}}_{j-1} - \bar{\boldsymbol{w}}_j\|^2 \geq \frac{J}{2} \left\| \sum_{j=1}^{J} \frac{1}{J} (\bar{\boldsymbol{w}}_{j-1} - \bar{\boldsymbol{w}}_j) \right\|^2 = \frac{1}{2J} \|\boldsymbol{w} - \bar{\boldsymbol{w}}\|^2, \tag{F.8}$$

which concludes the proof.

F.5 Proof of Lemma 9.6

Let $u(\widehat{\boldsymbol{w}}, \boldsymbol{w}) \triangleq \Omega(\widehat{\boldsymbol{w}}) - \Omega(\boldsymbol{w})$. We have successively:

$$\begin{aligned} \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t+1}\|^2 &\leq^{\text{(i)}} \|\widehat{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{t+1}\|^2 \\ &\leq^{\text{(ii)}} \|\widehat{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{t}\|^2 + 2\eta_t u(\widehat{\boldsymbol{w}}, \tilde{\boldsymbol{w}}^{t+1}) \\ &\leq^{\text{(iii)}} \|\widehat{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{t}\|^2 + 2\eta_t u(\widehat{\boldsymbol{w}}, \boldsymbol{w}^{t+1}) \\ &= \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t}\|^2 + \|\boldsymbol{w}^t - \tilde{\boldsymbol{w}}^{t}\|^2 + 2(\widehat{\boldsymbol{w}} - \boldsymbol{w}^t)^\top (\boldsymbol{w}^t - \tilde{\boldsymbol{w}}^t) + 2\eta_t u(\widehat{\boldsymbol{w}}, \boldsymbol{w}^{t+1}) \\ &= \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t}\|^2 + \eta_t^2 \|\boldsymbol{g}\|^2 + 2\eta_t (\bar{\boldsymbol{w}} - \boldsymbol{w}^t)^\top \boldsymbol{g} + 2\eta_t u(\widehat{\boldsymbol{w}}, \boldsymbol{w}^{t+1}) \\ &\leq^{\text{(iv)}} \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t}\|^2 + \eta_t^2 \|\boldsymbol{g}\|^2 + 2\eta_t (L(\widehat{\boldsymbol{w}}) - L(\boldsymbol{w}^t)) + 2\eta_t u(\widehat{\boldsymbol{w}}, \boldsymbol{w}^{t+1}) \\ &\leq \|\widehat{\boldsymbol{w}} - \boldsymbol{w}^{t}\|^2 + \eta_t^2 G^2 + 2\eta_t (L(\widehat{\boldsymbol{w}}) - L(\boldsymbol{w}^t)) + 2\eta_t u(\widehat{\boldsymbol{w}}, \boldsymbol{w}^{t+1}), \end{aligned} \tag{F.9}$$

where the inequality (i) is due to the nonexpansiveness of the projection operator, (ii) follows from Proposition 9.5, (iii) results from the fact that $\Omega(\tilde{\boldsymbol{w}}^{t+1}) \geq \Omega(\Pi_{\mathcal{W}}(\tilde{\boldsymbol{w}}^{t+1}))$, and (iv) results from the subgradient inequality of convex functions, which has an extra term $\frac{\sigma}{2}\|\widehat{\boldsymbol{w}} - \boldsymbol{w}^t\|^2$ if L is σ-strongly convex.

F.6 Proof of Proposition 9.7

Invoking Lemma 9.6 and summing for $t = 1, \dots, T$ gives

$$\begin{aligned}
\mathrm{Reg}_T &= \sum_{t=1}^{T} \left(L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) + \lambda \Omega(\boldsymbol{w}^t) \right) - \sum_{t=1}^{T} \left(L(\boldsymbol{w}^*; x^{n(t)}, y^{n(t)}) + \lambda \Omega(\boldsymbol{w}^*) \right) \\
&\leq^{\text{(i)}} \sum_{t=1}^{T} \left(L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) + \lambda \Omega(\boldsymbol{w}^{t+1}) \right) - \sum_{t=1}^{T} \left(L(\boldsymbol{w}^*; x^{n(t)}, y^{n(t)}) + \lambda \Omega(\boldsymbol{w}^*) \right) \\
&\leq \frac{G^2}{2} \sum_{t=1}^{T} \eta_t + \sum_{t=1}^{T} \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^t\|^2 - \|\boldsymbol{w}^* - \boldsymbol{w}^{t+1}\|^2}{2\eta_t} \\
&= \frac{G^2}{2} \sum_{t=1}^{T} \eta_t + \frac{1}{2} \sum_{t=2}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} \right) \|\boldsymbol{w}^* - \boldsymbol{w}^t\|^2 + \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^1\|^2}{2\eta_1} - \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^{t+1}\|^2}{2\eta_T}.
\end{aligned} \tag{F.10}$$

where the inequality (i) is due to the fact that $\boldsymbol{w}^1 = \boldsymbol{0}$, whence

$$\begin{aligned}
\sum_{t=1}^{T} \left(L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) + \Omega(\boldsymbol{w}^t) \right) &= \sum_{t=1}^{T} \left(L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) + \Omega(\boldsymbol{w}^{t+1}) \right) - (\Omega(\boldsymbol{w}^{t+1}) - \Omega(\boldsymbol{w}^1)) \\
&\leq \sum_{t=1}^{T} \left(L(\boldsymbol{w}^t; x^{n(t)}, y^{n(t)}) + \Omega(\boldsymbol{w}^{t+1}) \right).
\end{aligned} \tag{F.11}$$

Noting that the second term vanishes for a constant learning rate and that the last term is non-positive suffices to prove the first part. For the second part, we continue as:

$$\begin{aligned}
\mathrm{Reg}_T &\leq \frac{G^2}{2} \sum_{t=1}^{T} \eta_t + \frac{F^2}{2} \sum_{t=2}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} \right) + \frac{F^2}{2\eta_1} = \frac{G^2}{2} \sum_{t=1}^{T} \eta_t + \frac{F^2}{2\eta_T} \\
&\leq^{\text{(ii)}} G^2 \eta_0 (\sqrt{T} - 1/2) + \frac{F^2 \sqrt{T}}{2\eta_0} \leq \left(G^2 \eta_0 + \frac{F^2}{2\eta_0} \right) \sqrt{T},
\end{aligned} \tag{F.12}$$

where equality (ii) is due to the fact that $\sum_{t=1}^{T} \frac{1}{\sqrt{t}} \leq 2\sqrt{T} - 1$. For the third part, continue after inequality (i) as:

$$\begin{aligned}
\mathrm{Reg}_T &\leq \frac{G^2}{2} \sum_{t=1}^{T} \eta_t + \sum_{t=2}^{T} \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}} - \sigma \right) \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^t\|^2}{2} + \left(\frac{1}{\eta_1} - \sigma \right) \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^1\|^2}{2} - \frac{\|\boldsymbol{w}^* - \boldsymbol{w}^{t+1}\|^2}{2\eta_T} \\
&= \frac{G^2}{2\sigma} \sum_{t=1}^{T} \frac{1}{t} - \frac{\sigma T}{2} \cdot \|\boldsymbol{w}^* - \boldsymbol{w}^{t+1}\|^2 \leq \frac{G^2}{2\sigma} \sum_{t=1}^{T} \frac{1}{t} \leq^{\text{(iii)}} \frac{G^2}{2\sigma} (1 + \log T),
\end{aligned} \tag{F.13}$$

where the equality (iii) is due to the fact that $\sum_{t=1}^{T} \frac{1}{t} \leq 1 + \log T$.

F.7 Lipschitz Constants of Some Loss Functions

Let $\boldsymbol{w}^*$ be a solution of the problem (9.17) with $\mathcal{W} = \mathcal{H}$. For certain loss functions, we may obtain bounds of the form $\|\boldsymbol{w}^*\| \leq \gamma$ for some $\gamma > 0$, as the next proposition illustrates. Therefore, we may redefine $\mathcal{W} = \{\boldsymbol{w} \in \mathcal{H} \mid \|\boldsymbol{w}\| \leq \gamma\}$ (a vacuous constraint) without affecting the solution of (9.17).

Proposition F.1 *Let $\Omega(\boldsymbol{w}) = \frac{\lambda}{2}(\sum_{m=1}^{M} \|\boldsymbol{w}_m\|)^2$. Let L_{SSVM} and L_{CRF} be the structured hinge and logistic losses (9.5). Assume that the average cost function (in the SVM case) or the average entropy (in the CRF case) are bounded by some $\Lambda \geq 0$, i.e.,*[1]

$$\frac{1}{N}\sum_{i=1}^{N} \max_{y' \in \mathcal{Y}(x^i)} \rho(y'; y_i) \leq \Lambda \quad \text{or} \quad \frac{1}{N}\sum_{i=1}^{N} H(Y^i) \leq \Lambda. \tag{F.14}$$

Then:

1. *The solution of (9.17) with $\mathcal{W} = \mathcal{H}$ satisfies $\|\boldsymbol{w}^*\| \leq \sqrt{2\Lambda/\lambda}$.*
2. *L is G-Lipschitz on $\mathcal{H}$, with $G = 2\max_{u \in \mathcal{U}} \|\boldsymbol{f}(u)\|$.*
3. *Consider the following problem obtained from (9.17) by adding a quadratic term:*

$$\min_{\boldsymbol{w}} \frac{\sigma}{2}\|\boldsymbol{w}\|^2 + \Omega(\boldsymbol{w}) + \frac{1}{N}\sum_{i=1}^{N} L(\boldsymbol{w}; x^i, y^i). \tag{F.15}$$

   The solution of this problem satisfies $\|\boldsymbol{w}^*\| \leq \sqrt{2\Lambda/(\lambda+\sigma)}$.
4. *The modified loss $\tilde{L} = L + \frac{\sigma}{2}\|.\|^2$ is $\tilde{G}$-Lipschitz on $\left\{\boldsymbol{w} \mid \|\boldsymbol{w}\| \leq \sqrt{2\Lambda/(\lambda+\sigma)}\right\}$, where $\tilde{G} = G + \sqrt{2\sigma^2\Lambda/(\lambda+\sigma)}$.*

Proof. Let $F_{\text{SSVM}}(\boldsymbol{w})$ and $F_{\text{CRF}}(\boldsymbol{w})$ be the objectives of (9.17) for the SVM and CRF cases. We have

$$F_{\text{SVM}}(\boldsymbol{0}) = \Omega(\boldsymbol{0}) + \frac{1}{N}\sum_{i=1}^{N} L_{\text{SSVM}}(\boldsymbol{0}; x^i, y^i) = \frac{1}{N}\sum_{i=1}^{N} \max_{y' \in \mathcal{Y}(x^i)} \rho(y'; y^i) \leq \Lambda_{\text{SSVM}} \tag{F.16}$$

$$F_{\text{CRF}}(\boldsymbol{0}) = \Omega(\boldsymbol{0}) + \frac{1}{N}\sum_{i=1}^{N} L_{\text{CRF}}(\boldsymbol{0}; x^i, y^i) = \frac{1}{N}\sum_{i=1}^{N} \log|\mathcal{Y}(x^i)| \leq \Lambda_{\text{CRF}} \tag{F.17}$$

Using the facts that $F(\boldsymbol{w}^*) \leq F(\boldsymbol{0})$, that the losses are non-negative, and that $(\sum_i |a_i|)^2 \geq \sum_i a_i^2$, we obtain $\frac{\lambda}{2}\|\boldsymbol{w}^*\|^2 \leq \Omega(\boldsymbol{w}^*) \leq F(\boldsymbol{w}^*) \leq F(\boldsymbol{0})$, which proves the first statement.

To prove the second statement for the SVM case, note that a subgradient of L_{SSVM} at $\boldsymbol{w}$ is $\boldsymbol{g}_{\text{SSVM}} = \boldsymbol{f}(x, \widehat{y}) - \boldsymbol{f}(x, y)$, where $\widehat{y} = \arg\max_{y' \in \mathcal{Y}(x)} \boldsymbol{w}^\top(\boldsymbol{f}(x, y') - \boldsymbol{f}(x, y)) + \rho(y'; y)$; and that the gradient of L_{CRF} at $\boldsymbol{w}$ is $\boldsymbol{g}_{\text{CRF}} = \mathbb{E}_{\boldsymbol{w}}\boldsymbol{f}(x, Y) - \boldsymbol{f}(x, y)$. Applying Jensen's inequality, we have that $\|\boldsymbol{g}_{\text{CRF}}\| \leq \mathbb{E}_{\boldsymbol{w}}\|\boldsymbol{f}(x, Y) - \boldsymbol{f}(x, y)\|$. Therefore, both $\|\boldsymbol{g}_{\text{SSVM}}\|$ and $\|\boldsymbol{g}_{\text{CRF}}\|$ are upper bounded by $\max_{x \in \mathcal{X}, y, y' \in \mathcal{Y}(x)} \|\boldsymbol{f}(x, y') - \boldsymbol{f}(x, y)\| \leq 2\max_{u \in \mathcal{U}} \|\boldsymbol{f}(u)\|$.

The same rationale can be used to prove the third and fourth statements. ∎

F.8 Computing the proximity operator of the squared L_1

We present an algorithm (Algorithm 19) that computes the Moreau projection of the *squared, weighted* L_1-norm. Denote by $\odot$ the Hadamard product, $[\boldsymbol{a} \odot \boldsymbol{b}]_k = a_k b_k$. Letting $\lambda \geq 0, \boldsymbol{d} \geq \boldsymbol{0}$,

[1] In sequence binary labeling, we have $\Lambda = \bar{P}$ for the CRF case and for the SVM case with a Hamming cost function, where $\bar{P}$ is the average sequence length. Observe that the entropy of a distribution over labelings of a sequence of length P is upper bounded by $\log 2^P = P$.

Algorithm 19 Moreau projection for the squared weighted L_1-norm

Input: A vector $\boldsymbol{x}_0 \in \mathbb{R}^M$, a weight vector $\boldsymbol{d} \geq 0$, and a parameter $\lambda > 0$
Set $u_{0m} = |x_{0m}|/d_m$ and $a_m = d_m^2$ for each $m = 1, \ldots, M$
Sort $\boldsymbol{u}_0$: $u_{0(1)} \geq \ldots \geq u_{0(M)}$
Find $\rho = \max \left\{ j \in \{1, \ldots, M\} \mid u_{0(j)} - \frac{\lambda}{1+\lambda \sum_{r=1}^{j} a_{(r)}} \sum_{r=1}^{j} a_{(r)} u_{0(r)} > 0 \right\}$
Compute $\boldsymbol{u} = \text{soft}(\boldsymbol{u}_0, \tau)$, where $\tau = \frac{\lambda}{1+\lambda \sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)}$
Output: $\boldsymbol{x}$ s.t. $x_r = \text{sign}(x_{0r}) d_r u_r$.

and $\phi_{\boldsymbol{d}}(\boldsymbol{x}) \triangleq \frac{1}{2}\|\boldsymbol{d} \odot \boldsymbol{x}\|_1^2$, the underlying optimization problem is:

$$M_{\lambda\phi_{\boldsymbol{d}}}(\boldsymbol{x}_0) \triangleq \min_{\boldsymbol{x} \in \mathbb{R}^M} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{x}_0\|^2 + \frac{\lambda}{2} \left(\sum_{m=1}^{M} d_m |x_m| \right)^2. \tag{F.18}$$

This includes the squared L_1-norm as a particular case, when $\boldsymbol{d} = \mathbf{1}$ (the case addressed in Algorithm 15). The proof is somewhat technical and follows the same procedure employed by Duchi et al. (2008) to derive an algorithm for projecting onto the L_1-ball. The runtime is $O(M \log M)$ (the amount of time that is necessary to sort the vector), but a trick similar to the one described by (Duchi et al., 2008) can be employed to yield $O(M)$ runtime.

Lemma F.2 *Let $\boldsymbol{x}^* = \text{prox}_{\lambda\phi_{\boldsymbol{d}}}(\boldsymbol{x}_0)$ be the solution of (F.18). Then:*

1. *$\boldsymbol{x}^*$ agrees in sign with $\boldsymbol{x}_0$,* i.e., *each component satisfies $x_{0i} \cdot x_i^* \geq 0$.*
2. *Let $\boldsymbol{\sigma} \in \{-1, 1\}^M$. Then $\text{prox}_{\lambda\phi_{\boldsymbol{d}}}(\boldsymbol{\sigma} \odot \boldsymbol{x}_0) = \boldsymbol{\sigma} \odot \text{prox}_{\lambda\phi_{\boldsymbol{d}}}(\boldsymbol{x}_0)$,* i.e., *flipping a sign in $\boldsymbol{x}_0$ produces a $\boldsymbol{x}^*$ with the same sign flipped.*

Proof. Suppose that $x_{0i} \cdot x_i^* < 0$ for some i. Then, $\boldsymbol{x}$ defined by $x_j = x_j^*$ for $j \neq i$ and $x_i = -x_i^*$ achieves a lower objective value than $\boldsymbol{x}^*$, since $\phi_{\mathbf{d}}(\mathbf{x}) = \phi_{\boldsymbol{d}}(\boldsymbol{x}^*)$ and $(x_i - x_{0i})^2 < (x_i^* - x_{0i})^2$; this contradicts the optimality of $\boldsymbol{x}^*$. The second statement is a simple consequence of the first one and that $\phi_{\boldsymbol{d},\lambda}(\boldsymbol{\sigma} \odot \boldsymbol{x}) = \phi_{\boldsymbol{d},\lambda}(\boldsymbol{\sigma} \odot \boldsymbol{x}^*)$. ∎

Lemma F.2 enables reducing the problem to the non-negative orthant, by writing $\boldsymbol{x}_0 = \boldsymbol{\sigma} \cdot \tilde{\boldsymbol{x}}_0$, with $\tilde{\boldsymbol{x}}_0 \geq \mathbf{0}$, obtaining a solution $\tilde{\boldsymbol{x}}^*$ and then recovering the true solution as $\boldsymbol{x}^* = \boldsymbol{\sigma} \cdot \tilde{\boldsymbol{x}}^*$. It therefore suffices to solve (F.18) with the constraint $\boldsymbol{x} \geq \mathbf{0}$, which in turn can be transformed into:

$$\min_{\boldsymbol{u} \geq \mathbf{0}} F(\boldsymbol{u}) \triangleq \frac{1}{2} \sum_{m=1}^{M} a_m (u_m - u_{0m})^2 + \frac{\lambda}{2} \left(\sum_{m=1}^{M} a_m u_m \right)^2, \tag{F.19}$$

where we made the change of variables $a_m \triangleq d_m^2$, $u_{0m} \triangleq x_{0m}/d_m$ and $u_m \triangleq x_m/d_m$.

The Lagrangian of (F.19) is $\mathcal{L}(\boldsymbol{u}, \boldsymbol{\xi}) = \frac{1}{2}\sum_{m=1}^{M} a_m (u_m - u_{0m})^2 + \frac{\lambda}{2}\left(\sum_{m=1}^{M} a_m u_m\right)^2 - \boldsymbol{\xi}^\top \boldsymbol{u}$, where $\boldsymbol{\xi} \geq \mathbf{0}$ are Lagrange multipliers. Equating the gradient (w.r.t. $\boldsymbol{u}$) to zero gives

$$\boldsymbol{a} \odot (\boldsymbol{u} - \boldsymbol{u}_0) + \lambda \sum_{m=1}^{M} a_m u_m \boldsymbol{a} - \boldsymbol{\xi} = \mathbf{0}. \tag{F.20}$$

From the complementary slackness condition, $u_j > 0$ implies $\xi_j = 0$, which in turn implies

$$a_j(u_j - u_{0j}) + \lambda a_j \sum_{m=1}^{M} a_m u_m = 0. \tag{F.21}$$

Thus, if $u_j > 0$, the solution is of the form $u_j = u_{0j} - \tau$, with $\tau = \lambda \sum_{m=1}^{M} a_m u_m$. The next lemma shows the existence of a split point below which some coordinates vanish.

Lemma F.3 *Let $\boldsymbol{u}^*$ be the solution of (F.19). If $u_k^* = 0$ and $u_{0j} < u_{0k}$, then we must have $u_j^* = 0$.*

Proof. Suppose that $u_j^* = \epsilon > 0$. We will construct a $\tilde{\boldsymbol{u}}$ whose objective value is lower than $F(\boldsymbol{u}^*)$, which contradicts the optimality of $\boldsymbol{u}^*$: set $\tilde{u}_l = u_l^*$ for $l \notin \{j,k\}$, $\tilde{u}_k = \epsilon c$, and $\tilde{u}_j = \epsilon\left(1 - c a_k/a_j\right)$, where $c = \min\{a_j/a_k, 1\}$. We have $\sum_{m=1}^{M} a_m u_m^* = \sum_{m=1}^{M} a_m \tilde{u}_m$, and therefore

$$\begin{aligned} 2(F(\tilde{\boldsymbol{u}}) - F(\boldsymbol{u}^*)) &= \sum_{m=1}^{M} a_m(\tilde{u}_m - u_{0m})^2 - \sum_{m=1}^{M} a_m(u_m^* - u_{0m})^2 \\ &= a_j(\tilde{u}_j - u_{0j})^2 - a_j(u_j^* - u_{0j})^2 + a_k(\tilde{u}_k - u_{0k})^2 - a_k(u_k^* - u_{0k})^2. \end{aligned} \tag{F.22}$$

Consider the following two cases: (i) if $a_j \le a_k$, then $\tilde{u}_k = \epsilon a_j/a_k$ and $\tilde{u}_j = 0$. Substituting in (F.22), we obtain $2(F(\tilde{\boldsymbol{u}}) - F(\boldsymbol{u}^*)) = \epsilon^2\left(a_j^2/a_k - a_j\right) \le 0$, which leads to the contradiction $F(\tilde{\boldsymbol{u}}) \le F(\boldsymbol{u}^*)$. If (ii) $a_j > a_k$, then $\tilde{u}_k = \epsilon$ and $\tilde{u}_j = \epsilon\left(1 - a_k/a_j\right)$. Substituting in (F.22), we obtain $2(F(\tilde{\boldsymbol{u}}) - F(\boldsymbol{u}^*)) = a_j\epsilon^2\left(1 - a_k/a_j\right)^2 + 2a_k\epsilon u_{0j} - 2a_k\epsilon u_{0k} + a_k\epsilon^2 - a_j\epsilon^2 < a_k^2/a_j\epsilon^2 - 2a_k\epsilon^2 + a_k\epsilon^2 = \epsilon^2\left(a_k^2/a_j - a_k\right) < 0$, which also leads to a contradiction. ∎

Let $u_{0(1)} \ge \ldots \ge u_{0(M)}$ be the entries of $\boldsymbol{u}_0$ sorted in decreasing order, and let $u_{(1)}^*, \ldots, u_{(M)}^*$ be the entries of $\boldsymbol{u}^*$ under the same permutation. Let ρ be the number of nonzero entries in $\boldsymbol{u}^*$, i.e., $u_{(\rho)}^* > 0$, and, if $\rho < M$, $u_{(\rho+1)}^* = 0$. Summing (F.21) for $(j) = 1, \ldots, \rho$, we get

$$\sum_{r=1}^{\rho} a_{(r)} u_{(r)}^* - \sum_{r=1}^{\rho} a_{(r)} u_{0(r)} + \left(\sum_{r=1}^{\rho} a_{(r)}\right) \lambda \sum_{r=1}^{\rho} a_{(r)} u_{(r)}^* = 0, \tag{F.23}$$

which implies

$$\sum_{m=1}^{M} u_m^* = \sum_{r=1}^{\rho} u_{(r)}^* = \frac{1}{1 + \lambda \sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)}, \tag{F.24}$$

and therefore $\tau = \frac{\lambda}{1+\lambda \sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)}$. The complementary slackness conditions for $r = \rho$ and $r = \rho + 1$ imply

$$u_{(\rho)}^* - u_{0(\rho)} + \lambda \sum_{r=1}^{\rho} a_{(r)} u_{(r)}^* = 0 \quad \text{and} \quad -u_{0(\rho+1)}^* + \lambda \sum_{r=1}^{\rho} a_{(r)} u_{(r)}^* = \xi_{(\rho+1)} \ge 0; \tag{F.25}$$

therefore $u_{0(\rho)} > u_{0(\rho)} - u_{(\rho)}^* = \tau \ge u_{0(\rho+1)}$. This implies that ρ is such that

$$u_{0(\rho)} > \frac{\lambda}{1 + \lambda \sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)} \ge u_{0(\rho+1)}. \tag{F.26}$$

The next proposition goes farther by exactly determining ρ.

Proposition F.4 *The quantity ρ can be determined via:*

$$\rho = \max\left\{ j \in \{1,\dots,M\} \;\middle|\; u_{0(j)} - \frac{\lambda}{1+\lambda\sum_{r=1}^{j} a_{(r)}} \sum_{r=1}^{j} a_{(r)} u_{0(r)} > 0 \right\}. \tag{F.27}$$

Proof. Let $\rho^* = \max\{j | u^*_{(j)} > 0\}$. We have that $u^*_{(r)} = u_{0(r)} - \tau^*$ for $r \le \rho^*$, where $\tau^* = \frac{\lambda}{1+\lambda\sum_{r=1}^{\rho^*} a_{(r)}} \sum_{r=1}^{\rho^*} a_{(r)} u_{0(r)}$, and therefore $\rho \ge \rho^*$. We need to prove that $\rho \le \rho^*$, which we will do by contradiction. Assume that $\rho > \rho^*$. Let $\mathbf{u}$ be the vector induced by the choice of ρ, i.e., $u_{(r)} = 0$ for $r > \rho$ and $u_{(r)} = u_{0(r)} - \tau$ for $r \le \rho$, where $\tau = \frac{\lambda}{1+\lambda\sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)}$. From the definition of ρ, we have $u_{(\rho)} = u_{0(\rho)} - \tau > 0$, which implies $u_{(r)} = u_{0(r)} - \tau > 0$ for each $r \le \rho$. In addition,

$$\begin{aligned}
\sum_{r=1}^{M} a_r u_r &= \sum_{r=1}^{\rho} a_{(r)} u_{0(r)} - \sum_{r=1}^{\rho} a_{(r)} \tau = \left(1 - \frac{\lambda\sum_{r=1}^{\rho} a_{(r)}}{1+\lambda\sum_{r=1}^{\rho} a_{(r)}}\right) \sum_{r=1}^{\rho} a_{(r)} u_{0(r)} \\
&= \frac{1}{1+\lambda\sum_{r=1}^{\rho} a_{(r)}} \sum_{r=1}^{\rho} a_{(r)} u_{0(r)} = \frac{\tau}{\lambda}, \qquad \text{(F.28)} \\
\sum_{r=1}^{M} a_r (u_r - u_{0r})^2 &= \sum_{r=1}^{\rho^*} a_{(r)} \tau^2 + \sum_{r=\rho^*+1}^{\rho} a_{(r)} \tau^2 + \sum_{r=\rho+1}^{M} a_{(r)} u^2_{0(r)} \\
&< sum_{r=1}^{\rho^*} a_{(r)} \tau^2 + \sum_{r=\rho^*+1}^{M} a_{(r)} u^2_{0(r)}. \qquad \text{(F.29)}
\end{aligned}$$

We next consider two cases:

1. $\boxed{\tau^* \ge \tau.}$ From (F.29), we have that $\sum_{r=1}^{M} a_r (u_r - u_{0r})^2 < \sum_{r=1}^{\rho^*} a_{(r)} \tau^2 + \sum_{r=\rho^*+1}^{M} a_{(r)} u^2_{0(r)} \le \sum_{r=1}^{\rho^*} a_{(r)} (\tau^*)^2 + \sum_{r=\rho^*+1}^{M} a_{(r)} u^2_{0(r)} = \sum_{r=1}^{M} a_r (u^*_r - u_{0r})^2$. From (F.28), we have that $\left(\sum_{r=1}^{M} a_r u_r\right)^2 = \tau^2/\lambda^2 \le (\tau^*)^2/\lambda^2$. Summing the two inequalities, we get $F(\boldsymbol{u}) < F(\boldsymbol{u}^*)$, which leads to a contradiction.

2. $\boxed{\tau^* < \tau.}$ We will construct a vector $\tilde{\boldsymbol{u}}$ from $\boldsymbol{u}^*$ and show that $F(\tilde{\boldsymbol{u}}) < F(\boldsymbol{u}^*)$. Define

$$\tilde{u}_{(r)} = \begin{cases} u^*_{(\rho^*)} - \frac{2a_{(\rho^*+1)}}{a_{(\rho^*)}+a_{(\rho^*+1)}}\epsilon, & \text{if } r = \rho^* \\ \frac{2a_{(\rho^*)}}{a_{(\rho^*)}+a_{(\rho^*+1)}}\epsilon, & \text{if } r = \rho^*+1 \\ u^*_{(r)} & \text{otherwise,} \end{cases} \tag{F.30}$$

where $\epsilon = (u_{0(\rho^*+1)} - \tau^*)/2$. Note that $\sum_{r=1}^{M} a_r \tilde{u}_r = \sum_{r=1}^{M} a_r u^*_r$. From the assumptions that $\tau^* < \tau$ and $\rho^* < \rho$, we have that $u^*_{(\rho^*+1)} = u_{0(\rho^*+1)} - \tau > 0$, which implies that

$\tilde{u}_{(\rho^*+1)} = \frac{a_{(\rho^*)}(u_{0(\rho^*+1)}-\tau^*)}{a_{(\rho^*)}+a_{(\rho^*+1)}} > \frac{a_{(\rho^*)}(u_{0(\rho^*+1)}-\tau)}{a_{(\rho^*)}+a_{(\rho^*+1)}} = \frac{a_{(\rho^*)}u^*_{(\rho^*+1)}}{a_{(\rho^*)}+a_{(\rho^*+1)}} > 0$, and that

$$\begin{aligned}
u^*_{(\rho^*)} &= u_{0(\rho^*)} - \tau^* - \frac{a_{(\rho^*+1)}(u_{0(\rho^*+1)} - \tau^*)}{a_{(\rho^*)} + a_{(\rho^*+1)}} \\
&= u_{0(\rho^*)} - \frac{a_{(\rho^*+1)}u_{0(\rho^*+1)}}{a_{(\rho^*)} + a_{(\rho^*+1)}} - \left(1 - \frac{a_{(\rho^*+1)}}{a_{(\rho^*)} + a_{(\rho^*+1)}}\right)\tau^* \\
&>^{\text{(i)}} \left(1 - \frac{a_{(\rho^*+1)}}{a_{(\rho^*)} + a_{(\rho^*+1)}}\right)(u_{0(\rho^*+1)} - \tau) \\
&= \left(1 - \frac{a_{(\rho^*+1)}}{a_{(\rho^*)} + a_{(\rho^*+1)}}\right)(u^*_{(\rho^*+1)}) > 0,
\end{aligned} \tag{F.31}$$

where inequality (i) is justified by the facts that $u_{0(\rho^*)} \geq u_{0(\rho^*+1)}$ and $\tau > \tau^*$. This ensures that $\tilde{\boldsymbol{u}}$ is well defined. We have:

$$\begin{aligned}
&2(F(\boldsymbol{u}^*) - F(\tilde{\boldsymbol{u}})) \\
&= \sum_{r=1}^{M} a_r(u^*_r - u_{0r})^2 - \sum_{r=1}^{M} a_r(\tilde{u}_r - u_{0r})^2 \\
&= a_{(\rho^*)}(\tau^*)^2 + a_{(\rho^*+1)}u^2_{0(\rho^*+1)} - a_{(\rho^*)}\left(\tau^* + \frac{2a_{(\rho^*+1)}\epsilon}{a_{(\rho^*)} + a_{(\rho^*+1)}}\right)^2 \\
&\quad - a_{(\rho^*+1)}\left(u_{0(\rho^*+1)} - \frac{2a_{(\rho^*)}\epsilon}{a_{(\rho^*)} + a_{(\rho^*+1)}}\right)^2 \\
&= -\frac{4a_{(\rho^*)}a_{(\rho^*+1)}\epsilon}{a_{(\rho^*)} + a_{(\rho^*+1)}} \underbrace{(\tau^* - u_{0(\rho^*+1)})}_{-2\epsilon} - \frac{4a_{(\rho^*)}a^2_{(\rho^*+1)}\epsilon^2}{\left(a_{(\rho^*)} + a_{(\rho^*+1)}\right)^2} - \frac{4a^2_{(\rho^*)}a_{(\rho^*+1)}\epsilon^2}{\left(a_{(\rho^*)} + a_{(\rho^*+1)}\right)^2} \\
&= \frac{4a_{(\rho^*)}a_{(\rho^*+1)}\epsilon^2}{a_{(\rho^*)} + a_{(\rho^*+1)}} \geq 0,
\end{aligned} \tag{F.32}$$

which leads to a contradiction and completes the proof.

■

Bibliography

Abney, S. (1996). Statistical methods and linguistics. In *The balancing act: Combining symbolic and statistical approaches to language*, pages 1–26. MIT Press, Cambridge, MA. 1

Ackley, D., Hinton, G., and Sejnowski, T. (1985). A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169. 51

Afonso, M., Bioucas-Dias, J., and Figueiredo, M. (2010). Fast image recovery using variable splitting and constrained optimization. *IEEE Transactions on Image Processing*, 19. 105, 107, 219, 227

Aji, S. and McEliece, R. (2000). The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343. 48

Alshawi, H. (1996). Head automata and bilingual tiling: Translation with minimal representations. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 167–176. 21, 147

Altun, Y., Tsochantaridis, I., and Hofmann, T. (2003). Hidden Markov support vector machines. In *Proc. of International Conference of Machine Learning*. 1, 4, 27, 37, 174, 203, 206

Amari, S. and Nagaoka, H. (2001). *Methods of Information Geometry*. Oxford University Press. 51

Andrew, G. and Gao, J. (2007). Scalable training of L1-regularized log-linear models. In *Proc. of International Conference of Machine Learning*. 211

Anthony, M. and Bartlett, P. (2009). *Neural network learning: Theoretical foundations*. Cambridge University Press. 29

Auli, M. and Lopez, A. (2011). A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 61, 130, 159

Bach, F. (2008a). Consistency of the group Lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9:1179–1225. 189, 194

Bach, F. (2008b). Exploring large feature spaces with hierarchical multiple kernel learning. *Neural Information Processing Systems*, 21. 186, 193, 195, 203, 218

Bach, F., Jenatton, R., Mairal, J., and Obozinski., G. (2011). Convex optimization with sparsity-inducing norms. In *Optimization for Machine Learning*. MIT Press. 4, 206, 207, 211

Bach, F., Lanckriet, G., and Jordan, M. (2004). Multiple kernel learning, conic duality, and the SMO algorithm. In *International Conference of Machine Learning*. 186, 189, 190

Baker, J. (1979). Trainable grammars for speech recognition. *The Journal of the Acoustical Society of America*, 65:S132. 1, 16

Bakin, S. (1999). *Adaptive regression and model selection in data mining problems*. PhD thesis, Australian National University. 188, 203, 207, 218

Bakır, G., Hofmann, T., Schölkopf, B., Smola, A., Taskar, B., and Vishwanathan, S. (2007). *Predicting Structured Data*. The MIT Press. 1, 27

Bar-Hillel, Y., Perles, M., and Shamir, E. (1964). On formal properties of simple phrase structure grammars. *Language and Information: Selected Essays on their Theory and Application*, pages 116–150. 159

Barman, S., Liu, X., Draper, S., and Recht, B. (2011). Decomposition methods for large scale LP decoding. In *49th Annual Allerton Conference on Communication, Control, and Computing*, pages 253–260. IEEE. 132, 133

Baum, L. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563. 13, 48

Baum, L., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171. 13

Bauschke, H. and Combettes, P. (2011). *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. Springer Verlag. 185, 235

Bayati, M., Shah, D., and Sharma, M. (2005). Maximum weight matching via max-product belief propagation. In *Proceedings of International Symposium on Information Theory*, pages 1763–1767. IEEE. 50, 103

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202. 197, 219, 226

Berg-Kirkpatrick, T., Gillick, D., and Klein, D. (2011). Jointly learning to extract and compress. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 168, 225

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near Shannon limit error-correcting coding and decoding. In *Proc. of International Conference on Communications*, volume 93, pages 1064–1070. 135

Bertsekas, D., Hager, W., and Mangasarian, O. (1999). *Nonlinear programming*. Athena Scientific. 61, 63, 106, 109, 235, 237

Bertsekas, D., Nedic, A., and Ozdaglar, A. (2003). *Convex analysis and optimization*. Athena Scientific. 260

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 192–236. 46

Bethe, H. (1935). Statistical theory of superlattices. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 552–575. 57

Bicego, M., Martins, A., Murino, V., Aguiar, P., and Figueiredo, M. (2010a). 2d shape recognition using information theoretic kernels. In *Proc. of International Conference on Pattern Recognition*, pages 25–28. IEEE. 5

Bicego, M., Perina, A., Murino, V., Martins, A., Aguiar, P., and Figueiredo, M. (2010b). Combining free energy score spaces with information theoretic kernels: Application to scene classification. In *Proc. of IEEE International Conference on Image Processing*, pages 2661–2664. 5

Bioucas-Dias, J. and Figueiredo, M. (2007). A new TwIST: two-step iterative shrinkage/thresholding algorithms for image restoration. *IEEE Transactions on Image Processing*, 16(12):2992–3004. 219

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer New York. 1, 28

Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R., and Roukos, S. (1993). Towards history-based grammars: Using richer models for probabilistic parsing. In *Proc. of the Annual Meeting on Association for Computational Linguistics*, pages 31–37. 19, 34

Booth, T. and Thompson, R. (1973). Applying probability measures to abstract languages. *IEEE Transactions on Computers*, 100(5):442–450. 16

Borchardt, C. (1860). Über eine interpolationsformel für eine art symmetrischer functionen und über deren anwendung. *Abhandlungen der Königlichen Akademie der Wissenschaften zu Berlin*, pages 1–20. 33

Boros, E. and Hammer, P. (2002). Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225. 81, 145

Bottou, L. (1991a). Stochastic gradient learning in neural networks. In *Proc. of Neuro-Nîmes*. 4, 39

Bottou, L. (1991b). *Une Approche Theorique de l'Apprentissage Connexionniste: Applications a la Reconnaissance de la Parole*. PhD thesis, Université de Paris XI. 186, 192

Bottou, L. (2004). Stochastic learning. In Bousquet, O. and von Luxburg, U., editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag. 39

Bottou, L. and Bousquet, O. (2007). The tradeoffs of large scale learning. *Neural Information Processing Systems*, 20. 38

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Publishers. 105, 107, 109, 111, 114, 129, 130, 255

Boyd, S. P. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press. 31, 235

Boyle, J. and Dykstra, R. (1986). A method for finding projections onto the intersections of convex sets in Hilbert spaces. In *Advances in order restricted statistical inference*, pages 28–47. Springer Verlag. 261

Bradley, P. and Mangasarian, O. (1998). Feature selection via concave minimization and support vector machines. In *Proc. of International Conference of Machine Learning*, pages 82–90. 33

Bresnan, J. (2001). *Lexical-functional syntax*. Wiley-Blackwell. 22

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on multilingual dependency parsing. In *International Conference on Natural Language Learning*. 151, 162, 182, 216

Carpenter, B. (1997). *Type-logical semantics*. The MIT Press. 225

Carpenter, B. (2008). Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Technical report, Alias-i. 211, 213

Carreras, X. (2007). Experiments with a higher-order projective dependency parser. In *International Conference on Natural Language Learning*. 24, 137, 145, 162

Carreras, X., Collins, M., and Koo, T. (2008). TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-rich Parsing. In *International Conference on Natural Language Learning*. 162

Caruana, R. (1997). Multitask learning. *Machine Learning*, 28(1):41–75. 208, 218

Casella, G. and Robert, C. (1999). *Monte Carlo statistical methods*. Springer-Verlag, New York. 50

Cayley, A. (1889). A theorem on trees. *Quarterly Journal of Mathematics*, 23(376-378):69. 33

Censor, Y. and Zenios, S. A. (1997). *Parallel optimization: Theory, algorithms, and applications*. Oxford University Press. 181

Cesa-Bianchi, N., Conconi, A., and Gentile, C. (2004). On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057. 39, 197

Chang, M., Ratinov, L., and Roth, D. (2007). Guiding semi-supervision with constraint-driven learning. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 69

Chang, M., Ratinov, L., and Roth, D. (2008). Constraints as prior knowledge. In *International Conference of Machine Learning: Workshop on Prior Knowledge for Text and Language Processing*. 68, 73, 137

Chang, Y.-W. and Collins, M. (2011). Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proc. of Empirical Methods for Natural Language Processing*. 61, 130, 159, 225

Chapelle, O. and Rakotomamonjy, A. (2008). Second order optimization of kernel parameters. In *Neural Information Processing Systems: Workshop on Kernel Learning and Automatic Selection of Optimal Kernels*. 186, 190, 203

Charniak, E. (1996a). *Statistical language learning*. The MIT Press. 11, 19

Charniak, E. (1996b). Tree-bank grammars. In *Proc. of the National Conference on Artificial Intelligence*, pages 1031–1036. 1, 19

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. In *Proc. of the National Conference on Artificial Intelligence*, pages 598–603. 19, 35, 136

Charniak, E. and Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 173–180. 162

Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., et al. (2006). Multilevel coarse-to-fine pcfg parsing. In *Proc. of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 168–175. 20, 225

Chen, S. and Rosenfeld, R. (1999). A Gaussian prior for smoothing maximum entropy models. Technical report, CMU-CS-99-108. 31

Chen, S. F. and Rosenfeld, R. (2000). A survey of smoothing techniques for maximum entropy models. *IEEE Transactions on Speech and Audio Processing*, 8(1):37–50. 31

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124. 15

Chomsky, N. (1965). *Aspects of the Theory of Syntax*, volume 119. The MIT press. 5, 15, 16, 21

Chomsky, N. (1993). *Lectures on government and binding: The Pisa lectures*, volume 9. Walter de Gruyter. 22

Chomsky, N. (1995). *The minimalist program*, volume 28. Cambridge Univ Press. 22

Chu, Y. J. and Liu, T. H. (1965). On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400. 25, 144

Clarke, J. and Lapata, M. (2008). Global Inference for Sentence Compression An Integer Linear Programming Approach. *Journal of Artificial Intelligence Research*, 31:399–429. 69, 137, 168, 225

Cocke, J. and Schwartz, J. (1970). Programming languages and their compilers; preliminary notes. Technical report, Courant Institute of Mathematical Sciences, NYU. 19

Collins, M. (1999). *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania. 19, 35

Collins, M. (2002a). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proc. of Empirical Methods for Natural Language Processing*. 1, 4, 27, 41, 42, 174, 186, 203, 206

Collins, M. (2002b). Ranking algorithms for named-entity extraction: Boosting and the voted perceptron. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 181, 215

Collins, M., Globerson, A., Koo, T., Carreras, X., and Bartlett, P. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *Journal of Machine Learning Research*, 9:1775–1822. 38, 181, 186

Collobert, R., Sinz, F., Weston, J., and Bottou, L. (2006). Trading convexity for scalability. In *International Conference of Machine Learning*. 175

Combettes, P. and Wajs, V. (2006). Signal recovery by proximal forward-backward splitting. *Multiscale Modeling and Simulation*, 4(4):1168–1200. 191, 192, 203

Coppersmith, D. and Winograd, S. (1990). Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280. 25

Covington, M. (1990). Parsing discontinuous constituents in dependency grammar. *Computational Linguistics*, 16(4):234–236. 21

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research*, 7:551–585. 4, 42, 43, 162, 173, 174, 177, 180, 181, 186, 215, 224

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292. 32

Crammer, K. and Singer, Y. (2003). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991. 42, 173, 174, 224

Culotta, A. and Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Annual Meeting of the Association for Computational Linguistics*. 5, 21

Culotta, A., Wick, M., Hall, R., and McCallum, A. (2007). First-order probabilistic models for coreference resolution. In *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, pages 81–88. 69

Danskin, J. (1966). The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, pages 641–664. 237

Dantzig, G. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8(1):101–111. 61

Darmois, G. (1935). Sur les lois de probabilitéa estimation exhaustive. *CR Académie des Sciences de Paris*, 260:1265–1266. 51

Das, D. (2012). *Semi-Supervised and Latent-Variable Models of Natural Language Semantics*. PhD thesis, Carnegie Mellon University. 129, 130, 225

Daumé, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325. 34, 50

Dawid, A. (1992). Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2(1):25–36. 48

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41–85. 48

Dechter, R. and Mateescu, R. (2007). And/or search spaces for graphical models. *Artificial intelligence*, 171(2):73–106. 101, 224

Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–393. 206

Dempster, A., Laird, N., and Rubin, D. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38. 13

Denis, P. and Baldridge, J. (2007). Joint Determination of Anaphoricity and Coreference Resolution using Integer Programming. In *Proc. of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 69, 137, 225

Ding, Y. and Palmer, M. (2005). Machine translation using probabilistic synchronous dependency insertion grammar. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 5, 21

Donoho, D. and Johnstone, J. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3):425. 237

Dreyer, M. and Eisner, J. (2006). Better Informed Training of Latent Syntactic Features. In *Proc. of Empirical Methods in Natural Language Processing*, pages 317–326. 20

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the L1-ball for learning in high dimensions. In *Proc. of International Conference of Machine Learning*. 118, 268

Duchi, J. and Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2873–2908. 186, 193, 196, 211

Duchi, J., Tarlow, D., Elidan, G., and Koller, D. (2007). Using combinatorial optimization within max-product belief propagation. *Advances in Neural Information Processing Systems*, 19. 3, 68, 69, 82

Duda, R., Hart, P., and Stork, D. (2001). *Pattern classification*, volume 2. Wiley New York. 28

Eckstein, J. and Bertsekas, D. (1992). On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318. 105, 109, 115, 132

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240. 25, 144

Eisenstein, J., Smith, N. A., and Xing, E. P. (2011). Discovering sociolinguistic associations with structured sparsity. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 206, 208, 218

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proc. of International Conference on Computational Linguistics*, pages 340–345. 5, 19, 24, 35, 136, 146, 147

Eisner, J. (2000). Bilexical grammars and their cubic-time parsing algorithms. *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62. 24

Eisner, J. and Satta, G. (1999). Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 24

Everett III, H. (1963). Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417. 61

Facchinei, F. and Pang, J. (2003). *Finite-dimensional variational inequalities and complementarity problems*, volume 1. Springer Verlag. 252

Feldman, J., Wainwright, M., and Karger, D. (2005). Using linear programming to decode binary linear codes. *IEEE Transactions on Information Theory*, 51(3):954–972. 77

Figueiredo, M. (2002). Adaptive sparseness using Jeffreys' prior. *Advances in Neural Information Processing Systems*. 210

Fillmore, C. (1976). Frame Semantics and the Nature of Language. *Annals of the New York Academy of Sciences*, 280(1):20–32. 128

Finkel, J., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 363–370. 50

Finkel, J., Kleeman, A., and Manning, C. (2008). Efficient, feature-based, conditional random field parsing. *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 959–967. 19, 20, 36, 136

Finley, T. and Joachims, T. (2008). Training structural SVMs when exact inference is intractable. In *Proc. of International Conference of Machine Learning*. 184, 226

Forney, G. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278. 13, 48

Fortin, M. and Glowinski, R. (1983). *Augmented Lagrangian Methods*, volume 15. Elsevier. 109

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110. 121

Friedman, J., Hastie, T., and Tibshirani, R. (2010). A note on the group lasso and a sparse group lasso. Technical report, Stanford University. 193, 195, 203, 209, 218

Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309. 3, 68, 71

Gabay, D. and Mercier, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2(1):17–40. 105, 107, 109, 132

Gabow, H., Galil, Z., Spencer, T., and Tarjan, R. (1986). Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122. 25, 144

Gaifman, H. (1964). Concerning measures in first order calculi. *Israel journal of mathematics*, 2(1):1–18. 68

Gaifman, H. (1965). Dependency systems and phrase-structure systems. *Information and control*, 8(3):304–337. 22

Gallager, R. (1962). Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28. 77

Galley, M. (2006). A skip-chain conditional random field for ranking meeting utterances by importance. In *Proc. of Empirical Methods in Natural Language Processing*, pages 364–372. 14

Ganchev, K., Graca, J., Gillenwater, J., and Taskar, B. (2010). Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049. 101, 225

Gao, J., Andrew, G., Johnson, M., and Toutanova, K. (2007). A comparative study of parameter estimation methods for statistical natural language processing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 31, 207

Gelfand, A. and Smith, A. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, pages 398–409. 50

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741. 50

Germann, U., Jahr, M., Knight, K., Marcu, D., and Yamada, K. (2001). Fast decoding and optimal decoding for machine translation. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 228–235. 137

Giménez, J. and Marquez, L. (2004). SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proc. of International Conference on Language Resources and Evaluation*. 34, 162

Gimpel, K. and Smith, N. A. (2010). Softmax-Margin CRFs: Training Log-Linear Models with Loss Functions. In *NAACL*. 175

Globerson, A. and Jaakkola, T. (2008). Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. *Neural Information Processing Systems*, 20. 3, 59, 60, 61, 93, 106, 126, 127, 233

Glowinski, R. and Le Tallec, P. (1989). *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. Society for Industrial Mathematics. 105, 113, 132

Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par penalisation-dualité, d'une classe de problèmes de Dirichlet non linéaires. *Rev. Franc. Automat. Inform. Rech. Operat.*, 9:41–76. 105, 107, 109, 132

Goldberg, Y. and Elhadad, M. (2010). An efficient algorithm for easy-first non-directional dependency parsing. In *Proc. of Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. 34

Goldfarb, D., Ma, S., and Scheinberg, K. (2010). Fast alternating linearization methods for minimizing the sum of two convex functions. Technical report, UCLA CAM 10-02. 105, 107

Goodman, J. (1999). Semiring parsing. *Computational Linguistics*, 25(4):573–605. 48

Goodman, J. (2004). Exponential priors for maximum entropy models. In *Proc. of Annual Meeting of the North American Chapter of the Association for Computational Linguistics*. 31

Graça, J., Ganchev, K., Taskar, B., and Pereira, F. (2009). Posterior vs. parameter sparsity in latent variable models. *Advances in Neural Information Processing Systems*. 218

Grave, E., Obozinski, G., and Bach, F. (2011). Trace lasso: a trace norm regularization for correlated designs. *Arxiv preprint arXiv:1109.1990*. 219

Greig, D., Porteous, B., and Seheult, A. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B (Methodological)*, pages 271–279. 52

Grünbaum, B. (2003). *Convex polytopes*, volume 221. Springer Verlag. 226

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182. 206

Halevy, A., Norvig, P., and Pereira, F. (2009). The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, 24(2):8–12. 1

Halpern, J. (1990). An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350. 68

Hammersley, J. and Clifford, P. (1971). Markov field on finite graphs and lattices. Unpublished manuscript. 46

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. New York: Springer-Verlag. 29

Hazan, E., Agarwal, A., and Kale, S. (2007). Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2):169–192. 39, 195

Hazan, T. and Shashua, A. (2010). Norm-product belief propagation: Primal-dual message-passing for approximate inference. *IEEE Transactions on Information Theory*, 56(12):6294–6316. 59, 93, 101, 225

He, B. and Yuan, X. (2011). On the $O(1/t)$ convergence rate of alternating direction method. *SIAM Journal of Numerical Analysis (to appear)*. 105, 115, 132, 251, 252

Henderson, J. (2003). Inducing history representations for broad coverage statistical parsing. In *Proc. of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 24–31. 20

Hestenes, M. (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:302–320. 107, 109

Hofmann, T., Scholkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171. 188

Huang, B. and Jebara, T. (2007). Loopy belief propagation for bipartite maximum weight b-matching. *International Conference of Artificial Intelligence and Statistics*. 50, 103

Huang, J., Huang, X., and Metaxas, D. (2009). Learning with dynamic group sparsity. In *Proc. International Conference of Computer Vision*, pages 64–71. 219

Huang, L. (2008). Forest reranking: Discriminative parsing with non-local features. In *ACL*. 136

Huang, L. and Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. 5, 24, 34, 163, 166

Hudson, R. (1984). *Word grammar*. Blackwell Oxford. 5, 21

Ising, E. (1925). Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik A Hadrons and Nuclei*, 31(1):253–258. 51

Jaakkola, T. and Haussler, D. (1998). Exploiting generative models in discriminative classifiers. Technical report, Dept. of Computer Science, Univ. of California. 32

Jebara, T., Kondor, R., and Howard, A. (2004). Probability product kernels. *Journal of Machine Learning Research*, 5:819–844. 32

Jelinek, F. (1997). *Statistical methods for speech recognition*. the MIT Press. 1, 12

Jenatton, R., Audibert, J.-Y., and Bach, F. (2009). Structured variable selection with sparsity-inducing norms. Technical report, arXiv:0904.3523. 4, 186, 195, 203, 206, 207, 209, 218

Jenatton, R., Mairal, J., Obozinski, G., and Bach, F. (2010). Proximal methods for sparse hierarchical dictionary learning. In *Proc. of International Conference of Machine Learning*. 206, 211, 218

Jie, L., Orabona, F., Fornoni, M., Caputo, B., and Cesa-Bianchi, N. (2010). OM-2: An online multi-class Multi-Kernel Learning algorithm. In *Proc. of the 4th IEEE Online Learning for Computer Vision Workshop*, pages 43–50. IEEE. 203

Johnson, J. (2008). Equivalence of Entropy Regularization and Relative-Entropy Proximal Method. Unpublished manuscript. 107

Johnson, J., Malioutov, D., and Willsky, A. (2007). Lagrangian relaxation for MAP estimation in graphical models. In *45th Annual Allerton Conference on Communication, Control and Computing*. 61, 106, 107

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632. 19, 35, 136

Jojic, V., Gould, S., and Koller, D. (2010). Accelerated dual decomposition for MAP inference. In *International Conference of Machine Learning*. 107, 115, 126, 127, 130, 132, 256

Jurafsky, D. and Martin, J. (2000). *Speech & Language Processing*. Prentice Hall. 11

Kahane, S., Nasr, A., and Rambow, O. (1998). Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *International Conference on Computational Linguistics*. 22

Kakade, S. and Shalev-Shwartz, S. (2008). Mind the Duality Gap: Logarithmic regret algorithms for online optimization. In *Neural Information Processing Systems*. 44, 174, 177, 178

Kasami, T. (1966). An efficient recognition and syntax-analysis algorithm for context-free languages. Technical report, Air Force Cambridge Research Lab (Scientific report AFCRL-65-758). 19

Kazama, J. and Torisawa, K. (2007). A new perceptron algorithm for sequence labeling with non-local features. In *Proc. of Empirical Methods for Natural Language Processing*. 181

Kazama, J. and Tsujii, J. (2003). Evaluation and extension of maximum entropy models with inequality constraints. In *Proc. of Empirical Methods for Natural Language Processing*. 31

Kim, S. and Xing, E. (2010). Tree-guided group lasso for multi-task regression with structured sparsity. In *Proc. of International Conference of Machine Learning*. 206, 218

Kimeldorf, G. and Wahba, G. (1971). Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95. 33

Kirchhoff, G. (1847). Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508. 25

Klein, D. and Manning, C. (2003). Accurate unlexicalized parsing. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 423–430. 19

Kloft, M., Brefeld, U., Sonnenburg, S., and Zien, A. (2010). Non-Sparse Regularization and Efficient Training with Multiple Kernels. *Arxiv preprint arXiv:1003.0079*. 186, 189, 190, 191, 194, 200, 203

Koehn, P. (2010). *Statistical Machine Translation*, volume 9. Cambridge University Press. 11

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press. 3, 45, 46, 50

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1568–1583. 3, 59, 93, 106

Komodakis, N. and Paragios, N. (2009). Beyond pairwise energies: Efficient optimization for higher-order MRFs. In *Proc. of Conference on Computer Vision and Pattern Recognition*. 106

Komodakis, N., Paragios, N., and Tziritas, G. (2007). MRF optimization via dual decomposition: Message-passing revisited. In *Proc. of International Conference on Computer Vision*. 3, 61, 63, 106, 126, 223, 225, 256

Koo, T. and Collins, M. (2010). Efficient third-order dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 1–11. 5, 24, 130, 163, 164

Koo, T., Globerson, A., Carreras, X., and Collins, M. (2007). Structured prediction models via the matrix-tree theorem. In *Empirical Methods for Natural Language Processing*. 25, 177

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual decomposition for parsing with non-projective head automata. In *Proc. of Empirical Methods for Natural Language Processing*. 5, 26, 61, 114, 135, 147, 149, 151, 152, 153, 159, 160, 163, 164, 165, 166, 167, 224, 226

Koopman, B. (1936). On distributions admitting a sufficient statistic. *Transactions of the American Mathematical Society*, 39(399):10. 51

Kovalevsky, V. and Koval, V. (1975). A diffusion algorithm for decreasing energy of max-sum labeling problem. Technical report, Glushkov Institute of Cybernetics, Kiev, USSR. 60, 126

Kowalski, M. and Torrésani, B. (2009). Structured sparsity: From mixed norms to structured shrinkage. In *Workshop on Signal Processing with Adaptive Sparse Structured Representations*. 194

Kschischang, F. R., Frey, B. J., and Loeliger, H. A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47. 45, 47

Kulesza, A. and Pereira, F. (2007). Structured Learning with Approximate Inference. *Neural Information Processing Systems*. 114, 184, 226

Lacoste-Julien, S., Taskar, B., Klein, D., and Jordan, M. I. (2006). Word alignment via quadratic assignment. In *Proc. of Annual Meeting of the North American Chapter of the Association for Computational Linguistics*. 137

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of International Conference of Machine Learning*. 1, 4, 13, 27, 35, 36, 174, 186, 203, 206

Lafferty, J., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the 1992 AAAI Fall Symposium on Probabilistic Approaches to Natural Language*, pages 89–97. 21

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., and Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72. 185, 186, 188, 190, 203, 218

Langford, J., Li, L., and Zhang, T. (2009). Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801. 193, 196, 211, 212, 213

Lauritzen, S. (1996). *Graphical Models*. Clarendon Press, Oxford. 45, 50

Lavergne, T., Cappé, O., and Yvon, F. (2010). Practical very large scale CRFs. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 31, 207, 211

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 174, 181

Liang, P. and Jordan, M. (2008). An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proc. of International Conference on Machine learning*, pages 584–591. ACM. 32

Liang, P., Jordan, M., and Klein, D. (2011). Learning dependency-based compositional semantics. In *Proc. fo Annual Meeting of the Association for Computational Linguistics (ACL)*. 168, 225

Liao, L., Fox, D., and Kautz, H. (2005). Location-based activity recognition. In *Advances in Neural Information Processing Systems*. 102

Lions, P. and Mercier, B. (1979). Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979. 203

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45:503–528. 182

Liu, J. and Ye, J. (2010a). Fast Overlapping Group Lasso. Technical report, arXiv:1009.0306. 187, 195

Liu, J. and Ye, J. (2010b). Moreau-Yosida regularization for grouped tree structure learning. In *Advances in Neural Information Processing Systems*. 187, 195, 207, 211, 218

Lorbert, A., Eis, D., Kostina, V., Blei, D., and Ramadge, P. (2010). Exploiting covariate similarity in sparse regression via the pairwise elastic net. In *Proc. of International Conference on Artificial Intelligence and Statistics*, volume 9, pages 477–484. 219

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. (2010). Graphlab: A new parallel framework for machine learning. In *International Conference on Uncertainty in Artificial Intelligence*. 131

Magerman, D. (1995). Statistical decision-tree models for parsing. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 276–283. 19, 34, 35

Magnanti, T. and Wolsey, L. (1994). Optimal Trees. Technical Report 290-94, Massachusetts Institute of Technology, Operations Research Center. 140, 141, 142, 143

Mairal, J., Jenatton, R., Obozinski, G., and Bach, F. (2010). Network flow algorithms for structured sparsity. In *Advances in Neural Information Processing Systems*. 187, 195, 218

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA. 1, 11, 19

Marcus, M., Marcinkiewicz, M., and Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330. 1, 4, 18

Marroquin, J., Mitter, S., and Poggio, T. (1987). Probabilistic solution of ill-posed problems in computational vision. *Journal of the American Statistical Association*, pages 76–89. 50

Martin, R., Rardin, R., and Campbell, B. (1990). Polyhedral characterization of discrete dynamic programming. *Operations research*, pages 127–138. 102

Martins, A., Bicego, M., Murino, V., Aguiar, P., and Figueiredo, M. (2010a). Information theoretical kernels for generative embeddings based on hidden markov models. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 463–472. 5

Martins, A. F. T., Das, D., Smith, N. A., and Xing, E. P. (2008a). Stacking Dependency Parsers. In *Proc. of Empirical Methods for Natural Language Processing*. 5, 163, 164

Martins, A. F. T., Figueiredo, M. A. T., and Aguiar, P. M. Q. (2008b). Tsallis Kernels on Measures. In *IEEE Information Theory Workshop*. 5

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2008c). Nonextensive Entropic Kernels. In *Proc. of International Conference of Machine Learning*. 5

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2009a). Nonextensive Information Theoretic Kernels on Measures. *Journal of Machine Learning Research*, 10:935–975. 5, 32

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2010b). Online MKL for Structured Prediction. *Arxiv preprint arXiv:1010.2770*. 185

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2011a). An Augmented Lagrangian Approach to Constrained MAP Inference. In *Proc. of International Conference on Machine Learning*. 105, 168, 225

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2011b). Online learning of structured predictors with multiple kernels. In Gordon, G., Dunson, D., and Dudík, M., editors, *International Conference on Artificial Intelligence and Statistics*, volume 15. JMLR W&CP. 185, 205, 218

Martins, A. F. T., Gimpel, K., Smith, N. A., Xing, E. P., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2010c). Learning Structured Classifiers with Dual Coordinate Ascent. Technical report, CMU-ML-10-109. 173, 181

Martins, A. F. T. and Smith, N. A. (2009). Summarization with a Joint Model for Sentence Extraction and Compression. In *North American Chapter of the Association for Computational Linguistics: Workshop on Integer Linear Programming for NLP*. 5, 168, 225

Martins, A. F. T., Smith, N. A., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2011c). Dual Decomposition with Many Overlapping Components. In *Proc. of Empirical Methods for Natural Language Processing*. 135, 136

Martins, A. F. T., Smith, N. A., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2011d). Structured Sparsity in Structured Prediction. In *Proc. of Empirical Methods for Natural Language Processing*. 205

Martins, A. F. T., Smith, N. A., and Xing, E. P. (2009b). Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 135, 136, 145, 147, 150, 151, 152, 155, 158, 162, 163, 177, 182

Martins, A. F. T., Smith, N. A., and Xing, E. P. (2009c). Polyhedral Outer Approximations with Application to Natural Language Parsing. In *Proc. of International Conference of Machine Learning*. 5, 114, 162, 173, 184, 226

Martins, A. F. T., Smith, N. A., Xing, E. P., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2010d). Augmented Dual Decomposition for MAP Inference. In *Neural Information Processing Systems: Workshop in Optimization for Machine Learning*. 105

Martins, A. F. T., Smith, N. A., Xing, E. P., Aguiar, P. M. Q., and Figueiredo, M. A. T. (2010e). Online MKL for Structured Prediction. In *Neural Information Processing Systems: Workshop in New Directions in Multiple Kernel Learning*. 185, 189

Martins, A. F. T., Smith, N. A., Xing, E. P., Figueiredo, M. A. T., and Aguiar, P. M. Q. (2010f). Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proc. of Empirical Methods for Natural Language Processing*. 67, 100, 135, 136, 154, 158, 163, 164, 173

Matsuzaki, T., Miyao, Y., and Tsujii, J. (2005). Probabilistic CFG with latent annotations. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 75–82. 20

McAllester, D., Collins, M., and Pereira, F. (2008). Case-factor diagrams for structured probabilistic modeling. *Journal of Computer and System Sciences*, 74(1):84–96. 98, 101, 224

McCallum, A. (2003). Efficiently inducing features of conditional random fields. In *Proc. of Uncertainty in Artificial Intelligence*. 206

McDonald, R., Crammer, K., and Pereira, F. (2005a). Online large-margin training of dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 42, 162, 173

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual dependency analysis with a two-stage discriminative parser. In *Proc. of International Conference on Natural Language Learning*. 23, 25, 136, 137, 162, 163

McDonald, R. and Satta, G. (2007). On the complexity of non-projective data-driven dependency parsing. In *Proc. of International Conference on Parsing Technologies*. 25, 145, 146, 177

McDonald, R. T., Pereira, F., Ribarov, K., and Hajic, J. (2005b). Non-projective dependency parsing using spanning tree algorithms. In *Proc. of Empirical Methods for Natural Language Processing*. 5, 25, 35, 136, 216

McDonald, R. T. and Pereira, F. C. N. (2006). Online learning of approximate dependency parsing algorithms. In *Proc. of Annual Meeting of the European Chapter of the Association for Computational Linguistics*. 5, 23, 25, 145, 146

McEliece, R. J., MacKay, D. J. C., and Cheng, J. F. (1998). Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2). 135

Mel'čuk, I. (1988). *Dependency syntax: theory and practice*. State University of New York Press. 5, 21

Meshi, O. and Globerson, A. (2011). An Alternating Direction Method for Dual MAP LP Relaxation. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*. 132

Meza-Ruiz, I. and Riedel, S. (2009). Jointly Identifying Predicates, Arguments and Senses using Markov Logic. In *Proc. of North American Chapter of the Association for Computational Linguistics*. 68

Michelot, C. (1986). A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. *Journal of Optimization Theory and Applications*, 50(1):195–200. 118

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D., and Kolobov, A. (2007). BLOG: Probabilistic Models with Unknown Objects. *Statistical relational learning*, page 373. 68

Mitchell, T. (1997). *Machine learning*. McGraw Hill. 1, 28

Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350. 48

Moreau, J. (1962). Fonctions convexes duales et points proximaux dans un espace hilbertien. *CR de l'Académie des Sciences de Paris Série A Mathematics*, 255:2897–2899. 191, 192, 195, 203

Mota, J., Xavier, J., Aguiar, P., and Puschel, M. (2010). Distributed basis pursuit. *IEEE Transactions on Signal Processing*, 99. 105

Murray, M. and Rice, J. (1993). *Differential Geometry and Statistics*. Number 48 in Monographs on Statistics and Applied Probability. Chapman and Hall, 1st edition. 51

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Math. Doklady*, 27:372–376. 39, 107

Nesterov, Y. (2005). Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152. 107, 130

Ng, A. and Jordan, M. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841. 32

Nivre, J. (2009). Non-projective dependency parsing in expected linear time. In *Annual Meeting of the Association for Computational Linguistics*, pages 351–359. 26

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006). Labeled pseudo-projective dependency parsing with support vector machines. In *Procs. of International Conference on Natural Language Learning*. 24, 34, 163

Nivre, J. and McDonald, R. (2008). Integrating graph-based and transition-based dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 163

Nocedal, J. and Wright, S. (1999). *Numerical optimization*. Springer verlag. 121, 123, 124, 125

Nowozin, S. and Lampert, C. (2009). Global connectivity potentials for random field models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 818–825. IEEE. 68

Obozinski, G., Taskar, B., and Jordan, M. (2010). Joint covariate selection and joint subspace selection for multiple classification problems. *Statistics and Computing*, 20(2):231–252. 206, 218

Pal, C., Sutton, C., and McCallum, A. (2006). Sparse forward-backward using minimum divergence beams for fast training of conditional random fields. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 5. IEEE. 34

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann. 3, 45, 48

Pereira, F. (2000). Formal grammar and information theory: together again? *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 358(1769):1239–1253. 1

Pereira, F. and Schabes, Y. (1992). Inside-outside reestimation from partially bracketed corpora. In *Proc. of Annual Meeting on Association for Computational Linguistics*, pages 128–135. 19

Petrov, S. (2009). *Coarse-to-Fine Natural Language Processing*. PhD thesis, University of California at Bekeley, Berkeley, CA, USA. 20

Petrov, S. and Klein, D. (2007). Improved inference for unlexicalized parsing. In *Proc. of Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 404–411. 20

Petrov, S. and Klein, D. (2008a). Discriminative log-linear grammars with latent variables. *Advances in Neural Information Processing Systems*, 20:1153–1160. 20, 207

Petrov, S. and Klein, D. (2008b). Sparse multi-scale grammars for discriminative latent variable parsing. In *Proc. of Empirical Methods for Natural Language Processing*. 20, 207

Pitman, E. (1936). Sufficient statistics and intrinsic accuracy. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 32, pages 567–579. Cambridge Univ Press. 51

Poole, D. (1993). Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1):81–129. 68

Poon, H. and Domingos, P. (2009). Unsupervised semantic parsing. In *Proc. of Empirical Methods in Natural Language Processing*. 68

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *IEEE International Conference on Computer Vision Workshops*, pages 689–690. IEEE. 101, 224

Powell, M. (1969). A method for nonlinear constraints in minimization problems. In Fletcher, R., editor, *Optimization*, pages 283–298. Academic Press. 107, 109

Punyakanok, V., Roth, D., Yih, W., and Zimak, D. (2004). Semantic role labeling via integer linear programming inference. In *Internacional Conference on Computational Linguistics*. 137

Punyakanok, V., Roth, D., Yih, W., and Zimak, D. (2005). Learning and Inference over Constrained Output. In *Proc. of International Joint Conference on Artificial Intelligence*. 68, 69, 73

Quattoni, A., Carreras, X., Collins, M., and Darrell, T. (2009). An efficient projection for $l_{1,\infty}$ regularization. In *Proc. of International Conference of Machine Learning*. 208

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. 1, 12, 13, 35

Rakotomamonjy, A., Bach, F., Canu, S., and Grandvalet, Y. (2008). SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521. 186, 189, 190, 200, 203

Ratliff, N., Bagnell, J., and Zinkevich, M. (2006). Subgradient methods for maximum margin structured learning. In *International Conference of Machine Learning: Workshop on Learning in Structured Outputs Spaces*. 40, 186

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1):151–175. 20, 34

Ravikumar, P., Agarwal, A., and Wainwright, M. (2010). Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *Journal of Machine Learning Research*, 11:1043–1080. 59, 106

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine Learning*, 62(1):107–136. 2, 3, 68, 71, 137, 160, 223

Richardson, T. and Urbanke, R. (2008). *Modern coding theory*. Cambridge University Press. 68, 77

Riedel, S. and Clarke, J. (2006). Incremental integer linear programming for non-projective dependency parsing. In *Proc. of Empirical Methods for Natural Language Processing*. 68, 69, 135, 138, 140, 141, 167, 224

Riezler, S. (1998). *Probabilistic constraint logic programming*. PhD thesis, University of Tübingen. 68

Robbins, H. and Monro, S. (1951). A stochastic approximation model. *Annals of Mathematical Statistics*, 22:400–407. 39

Rockafellar, R. (1970). *Convex Analysis*. Princeton University Press. 54, 139, 235, 236

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. 41, 42

Roth, D. and Yih, W. (2004). A linear programming formulation for global inference in natural language tasks. In *International Conference on Natural Language Learning*. 2, 3, 68, 73, 223

Roth, D. and Yih, W. T. (2005). Integer linear programming inference for conditional random fields. In *International Conference of Machine Learning*. 69, 137

Rush, A., Sontag, D., Collins, M., and Jaakkola, T. (2010). On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of Empirical Methods for Natural Language Processing*. 3, 61, 114, 159, 223, 225

Rush, A. M. and Collins, M. (2011). Exact decoding of syntactic translation models through lagrangian relaxation. In *Proc. of Annual Meeting on Association for Computational Linguistics*. 61, 114, 130, 159, 225

Sag, I., Wasow, T., Bender, E., and Sag, I. (1999). *Syntactic theory: A formal introduction*, volume 92. Center for the Study of Language and Information. 22

Sang, E. (2002). Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *Proc. of International Conference on Natural Language Learning*. 215

Sang, E. and Buchholz, S. (2000). Introduction to the CoNLL-2000 shared task: Chunking. In *Proc. of International Conference on Natural Language Learning*. 214

Sang, E. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of International Conference on Natural Language Learning*. 215

Schlesinger, M. (1976). Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, 4:113–130. 3, 59

Schmidt, M. and Murphy, K. (2010). Convex structure learning in log-linear models: Beyond pairwise potentials. In *Proc. of International Conference on Artificial Intelligence and Statistics*. 218

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. The MIT Press, Cambridge, MA. 1, 28, 29, 33, 187

Schrijver, A. (2003). *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer. 102, 140, 143, 226

Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proc. of Annual Meeting of the North American Chapter of the Association for Computational Linguistics*. 39

Shafer, G. and Shenoy, P. (1990). Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2(1):327–351. 48

Shalev-Shwartz, S. (2011). Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194. 185

Shalev-Shwartz, S. and Singer, Y. (2006). Online learning meets optimization in the dual. In *Proc. of Conference on Learning Theory*. 44, 173, 174, 177

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *Proc. of International Conference of Machine Learning*. 39, 40, 41, 193, 197, 199, 213

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2010). Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 125. 39, 40, 41, 193

Shor, N. (1985). *Minimization methods for non-differentiable functions*. Springer. 61

Smith, D. and Eisner, J. (2006). Minimum risk annealing for training log-linear models. In *Proc. of Annual Meeting of the Association for Computation Linguistics*. 175

Smith, D. and Eisner, J. (2008). Dependency parsing by belief propagation. In *Proc. of Empirical Methods for Natural Language Processing*. 3, 5, 25, 68, 69, 77, 78, 79, 82, 135, 146, 151, 152, 153, 158, 160, 162, 167, 174, 177, 182, 224

Smith, D. and Smith, N. (2007). Probabilistic models of nonprojective dependency trees. In *Proc. of Empirical Methods for Natural Language Processing*. 25, 177

Smith, N. A. (2011). *Linguistic Structure Prediction*, volume 13 of *Synthesis Lectures on Human Language Technologies*. Morgan and Claypool. 1, 11

Sokolovska, N., Lavergne, T., Cappé, O., and Yvon, F. (2010). Efficient learning of sparse conditional random fields for supervised sequence labelling. *IEEE Journal of Selected Topics in Signal Processing*, 4(6):953–964. 211

Sonnenburg, S., Rätsch, G., Schäfer, C., and Schölkopf, B. (2006). Large scale MKL. *Journal of Machine Learning Research*, 7:1565. 186, 190, 201, 203

Sontag, D. (2010). *Approximate Inference in Graphical Models using LP Relaxations*. PhD thesis, MIT. 50, 59, 226

Sontag, D., Globerson, A., and Jaakkola, T. (2011). Introduction to dual decomposition for inference. In *Optimization for Machine Learning*. MIT Press. 60, 61, 62, 126

Sontag, D. and Jaakkola, T. (2009). Tree block coordinate descent for MAP in graphical models. In *Proc. of International Conference on Artificial Intelligence and Statistics*, volume 8, pages 544–551. Journal of Machine Learning Research: W&CP. 60

Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., and Jaakkola, T. (2008). Tightening LP relaxations for MAP using message-passing. In *Proc. of Uncertainty in Artificial Intelligence*. 3, 61, 128, 131, 225

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. *Proc. of International Conference on Natural Language Learning*. 162, 182

Sutton, C. (2004). Collective segmentation and labeling of distant entities in information extraction. Technical report, DTIC Document. 68

Sutton, C. and McCallum, A. (2006). An introduction to conditional random fields for relational learning. In *Introduction to statistical relational learning*, pages 95–130. MIT press. 14

Suzuki, T. and Tomioka, R. (2009). SpicyMKL. *Arxiv preprint arXiv:0909.5026*. 194, 203

Tanner, R. (1981). A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547. 45

Tarjan, R. (1977). Finding optimum branchings. *Networks*, 7(1):25–36. 25, 144

Taskar, B., Abbeel, P., and Koller, D. (2002). Discriminative probabilistic models for relational data. In *Proc. of Uncertainty in Artificial Intelligence*, pages 895–902. 68

Taskar, B., Chatalbashev, V., and Koller, D. (2004a). Learning associative Markov networks. In *Proc. of International Conference of Machine Learning*. 52

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin Markov networks. In *Proc. of Neural Information Processing Systems*. 1, 4, 15, 27, 37, 174, 186, 189, 199, 200, 203, 206

Taskar, B., Klein, D., Collins, M., Koller, D., and Manning, C. (2004b). Max-margin parsing. In *Proc. of Empirical Methods for Natural Language Processing*, pages 1–8. 19, 20, 98

Taskar, B., Lacoste-Julien, S., and Jordan, M. (2006a). Structured Prediction via the Extragradient Method. In *Neural Information Processing Systems*. 38

Taskar, B., Lacoste-Julien, S., and Jordan, M. I. (2006b). Structured prediction, dual extragradient and Bregman projections. *Journal of Machine Learning Research*, 7:1627–1653. 38, 176

Tesnière, L. (1959). *Eléments de syntaxe structurale*. Libraire C. Klincksieck. 5, 21

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B.*, pages 267–288. 31, 207

Tikhonov, A. (1943). On the stability of inverse problems. In *Doklady Akademii Nauk SSSR*, volume 39, pages 195–198. 31

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *International Conference on Natural Language Learning*. 181

Tomioka, R. and Suzuki, T. (2010). Sparsity-accuracy trade-off in MKL. Technical report, arXiv:1001.2615. 193, 194

Tromble, R. and Eisner, J. (2006). A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proc. of Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 423–430. 159

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proc. of International Conference of Machine Learning*. 1, 4, 27, 37, 174, 186, 203, 206

Tsuruoka, Y., Tsujii, J., and Ananiadou, S. (2009). Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty. In *Proc. of Annual Meeting of the Association for Computational Linguistics*. 211

Tutte, W. (1984). *Graph Theory*. Addison-Wesley, Reading, MA. 25

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142. 39

Vapnik, N. V. (2000). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York. 32

Vapnik, V. (1998). *Statistical Learning Theory*. Wiley. 29

Vazirani, V. (2001). *Approximation Algorithms*. Springer. 184, 226

Vishwanathan, S. V. N., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient methods. In *International Conference of Machine Learning*. 4, 39, 186

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269. 13, 48

Wainwright, M., Jaakkola, T., and Willsky, A. (2005a). MAP estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717. 3, 59, 93, 106

Wainwright, M. and Jordan, M. (2008). *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers. 3, 46, 50, 51, 53, 54, 55, 57, 59, 60, 75, 182

Wainwright, M. J. (2006). Estimating the" Wrong" Graphical Model: Benefits in the Computation-Limited Setting. *Journal of Machine Learning Research*, 7:1829–1859. 184, 226

Wainwright, M. J., Jaakkola, T., and Willsky, A. (2005b). A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335. 59, 93, 101, 225

Wang, H. and Banerjee, A. (2012). Online Alternating Direction Method. In *Proc. of International Conference on Machine Learning*. 105, 115, 132, 251, 252

Wang, M., Smith, N. A., and Mitamura, T. (2007). What is the Jeopardy model? A quasi-synchronous grammar for QA. In *Proceedings of Empirical Methods for Natural Language Processing-CoNLL*. 5, 21

Weiss, D. and Taskar, B. (2010). Structured prediction cascades. In *Proc. of International Conference on Artificial Intelligence and Statistics*, volume 1284. 225

Weiss, Y. and Freeman, W. (2001a). Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural computation*, 13(10):2173–2200. 50, 103

Weiss, Y. and Freeman, W. (2001b). On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744. 50, 103

Weiss, Y., Yanover, C., and Meltzer, T. (2007). Map estimation, linear programming and belief propagation with convex free energies. In *Proc. of Uncertainty in Artificial Intelligence*. 59, 93, 101, 225

Werner, T. (2007). A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1165–1179. 59, 60, 93, 106, 126

Wiegerinck, W. and Heskes, T. (2003). Fractional belief propagation. In *Advances in Neural Information Processing Systems*, volume 15, page 455. 59, 93, 101, 225

Wong, R. (1984). A dual ascent approach for steiner tree problems on a directed graph. *Mathematical programming*, 28(3):271–287. 140

Wright, S., Nowak, R., and Figueiredo, M. A. T. (2009). Sparse reconstruction by separable approximation. *IEEE Transactions on Signal Processing*, 57(7):2479–2493. 191, 203, 208, 214, 219, 226

Xiao, L. (2009). Dual averaging methods for regularized stochastic learning and online optimization. In *Advances in Neural Information Processing Systems*. 211

Xiao, L. (2010). Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596. 219, 227

Xu, Z., Jin, R., King, I., and Lyu, M. (2009). An extended level method for efficient multiple kernel learning. *Neural Information Processing Systems*, 21:1825–1832. 186, 191, 197, 203

Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proc. of International Conference on Parsing Technologies*. 34, 162

Yanover, C., Meltzer, T., and Weiss, Y. (2006). Linear programming relaxations and belief propagation–an empirical study. *Journal of Machine Learning Research*, 7:1887–1907. 59, 128, 129

Yedidia, J., Freeman, W., and Weiss, Y. (2004). Constructing free-energy approximations and generalized belief propagation algorithms. Technical Report TR2004-040, MERL. 74

Yedidia, J., Freeman, W., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312. 59

Yedidia, J., Wang, Y., and Draper, S. (2011). Divide and concur and difference-map BP decoders for LDPC codes. *IEEE Transactions on Information Theory*, 57(2):786–802. 132

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2001). Generalized belief propagation. In *Neural Information Processing Systems*. 58, 59

Younger, D. (1967). Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189–208. 19

Yuan, M. and Lin, Y. (2006). Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society Series B (Statistical Methodology)*, 68(1):49. 188, 203, 207, 218

Zhang, N. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Tenth Canadian Conference on Artificial Intelligence*, pages 171–178. 48

Zhang, Y. and Clark, S. (2008). A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Empirical Methods for Natural Language Processing*. 50

Zhao, P., Rocha, G., and Yu, B. (2009). The composite absolute penalties family for grouped and hierarchical variable selection. *Annals of Statistics*, 37(6A):3468–3497. 4, 186, 193, 195, 203, 206, 208, 218

Zien, A. and Ong, C. (2007). Multiclass multiple kernel learning. In *Proc. of International Conference of Machine Learning*. 186, 203

Zinkevich, M. (2003). Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proc. of International Conference of Machine Learning*. 39, 193, 195

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B(Statistical Methodology)*, 67(2):301–320. 31, 207