# Fast and Robust Compressive Summarization with Dual Decomposition and Multi-Task Learning

**Miguel B. Almeida**[*†] **André F. T. Martins**[*†]
[*]Priberam Labs, Alameda D. Afonso Henriques, 41, 2º, 1000-123 Lisboa, Portugal
[†]Instituto de Telecomunicações, Instituto Superior Técnico, 1049-001 Lisboa, Portugal
{mba,atm}@priberam.pt

## Abstract

We present a dual decomposition framework for multi-document summarization, using a model that jointly extracts and compresses sentences. Compared with previous work based on integer linear programming, our approach does not require external solvers, is significantly faster, and is modular in the three qualities a summary should have: conciseness, informativeness, and grammaticality. In addition, we propose a multi-task learning framework to take advantage of existing data for extractive summarization and sentence compression. Experiments in the TAC-2008 dataset yield the highest published ROUGE scores to date, with runtimes that rival those of extractive summarizers.

## 1 Introduction

Automatic text summarization is a seminal problem in information retrieval and natural language processing (Luhn, 1958; Baxendale, 1958; Edmundson, 1969). Today, with the overwhelming amount of information available on the Web, the demand for fast, robust, and scalable summarization systems is stronger than ever.

Up to now, **extractive systems** have been the most popular in multi-document summarization. These systems produce a summary by extracting a representative set of sentences from the original documents (Kupiec et al., 1995; Carbonell and Goldstein, 1998; Radev et al., 2000; Gillick et al., 2008). This approach has obvious advantages: it reduces the search space by letting decisions be made for each sentence as a whole (avoiding fine-grained text generation), and it ensures a grammatical summary, assuming the original sentences are well-formed. The typical trade-offs in these models (maximizing relevance, and penalizing redundancy) lead to submodular optimization problems (Lin and Bilmes, 2010), which are NP-hard but approximable through greedy algorithms; learning is possible with standard structured prediction algorithms (Sipos et al., 2012; Lin and Bilmes, 2012). Probabilistic models have also been proposed to capture the problem structure, such as determinantal point processes (Gillenwater et al., 2012).

However, extractive systems are rather limited in the summaries they can produce. Long, partly relevant sentences tend not to appear in the summary, or to block the inclusion of other sentences. This has motivated research in **compressive summarization** (Lin, 2003; Zajic et al., 2006; Daumé, 2006), where summaries are formed by compressed sentences (Knight and Marcu, 2000), not necessarily extracts. While promising results have been achieved by models that simultaneously extract and compress (Martins and Smith, 2009; Woodsend and Lapata, 2010; Berg-Kirkpatrick et al., 2011), there are still obstacles that need to be surmounted for these systems to enjoy wide adoption. All approaches above are based on **integer linear programming** (ILP), suffering from slow runtimes, when compared to extractive systems. For example, Woodsend and Lapata (2012) report 55 seconds on average to produce a summary; Berg-Kirkpatrick et al. (2011) report substantially faster runtimes, but fewer compressions are allowed. Having a compressive summarizer which is both fast and expressive remains an open problem. A second inconvenience of ILP-based approaches is that they do not exploit the modularity of the problem, since the declarative specification required by ILP solvers discards important structural information. For example, such solvers are unable to take advantage of efficient dynamic programming routines for sentence compression (McDonald, 2006).

This paper makes progress in two fronts:

- We derive a **dual decomposition** framework for extractive and compressive summarization (§2–3). Not only is this framework orders of magnitude more efficient than the ILP-based approaches, it also allows the three well-known metrics of summaries—conciseness, informativeness, and grammaticality—to be treated separately in a modular fashion (see Figure 1). We also contribute with a novel **knapsack factor**, along with a linear-time algorithm for the corresponding dual decomposition subproblem.
- We propose **multi-task learning** (§4) as a principled way to train compressive summarizers, using auxiliary data for extractive summarization and sentence compression. To this end, we adapt the framework of Evgeniou and Pontil (2004) and Daumé (2007) to train structured predictors that share some of their parts.

Experiments on TAC data (§5) yield state-of-the-art results, with runtimes similar to that of extractive systems. To our best knowledge, this had never been achieved by compressive summarizers.

## 2 Extractive Summarization

In **extractive summarization**, we are given a set of sentences $\mathcal{D} := \{s_1, \ldots, s_N\}$ belonging to one or more documents, and the goal is to extract a subset $\mathcal{S} \subseteq \mathcal{D}$ that conveys a good summary of $\mathcal{D}$ and whose total number of words does not exceed a prespecified budget $B$.

We use an indicator vector $\boldsymbol{y} := \langle y_n \rangle_{n=1}^N$ to represent an extractive summary, where $y_n = 1$ if $s_n \in \mathcal{S}$, and $y_n = 0$ otherwise. Let $L_n$ be the number of words of the $n$th sentence. By designing a quality score function $g : \{0,1\}^N \to \mathbb{R}$, this can be cast as a global optimization problem with a knapsack constraint:

$$\begin{array}{rl} \text{maximize} & g(\boldsymbol{y}) \\ \text{w.r.t.} & \boldsymbol{y} \in \{0,1\}^N \\ \text{s.t.} & \sum_{n=1}^N L_n y_n \leq B. \end{array} \qquad (1)$$

Intuitively, a good summary is one which selects sentences that individually convey "relevant" information, while collectively having small "redundancy." This trade-off was explicitly modeled in early works through the notion of **maximal marginal relevance** (Carbonell and Goldstein, 1998; McDonald, 2007). An alternative are **coverage-based models** (§2.1; Filatova and Hatzivassiloglou, 2004; Yih et al., 2007; Gillick et al., 2008), which seek a set of sentences that covers as many diverse "concepts" as possible; redundancy is automatically penalized since redundant sentences cover fewer concepts. Both models can be framed under the framework of submodular optimization (Lin and Bilmes, 2010), leading to greedy algorithms that have approximation guarantees. However, extending these models to allow for sentence compression (as will be detailed in §3) breaks the diminishing returns property, making submodular optimization no longer applicable.

### 2.1 Coverage-Based Summarization

Coverage-based extractive summarization can be formalized as follows. Let $\mathcal{C}(\mathcal{D}) := \{c_1, \ldots, c_M\}$ be a set of relevant **concept types** which are present in the original documents $\mathcal{D}$.[1] Let $\sigma_m$ be a relevance score assigned to the $m$th concept, and let the set $\mathcal{I}_m \subseteq \{1, \ldots, N\}$ contain the indices of the sentences in which this concept occurs. Then, the following quality score function is defined:

$$g(\boldsymbol{y}) = \textstyle\sum_{m=1}^M \sigma_m u_m(\boldsymbol{y}), \qquad (2)$$

where $u_m(\boldsymbol{y}) := \bigvee_{n \in \mathcal{I}_m} y_n$ is a Boolean function that indicates whether the $m$th concept is present in the summary. Plugging this into Eq. 1, one obtains the following Boolean optimization problem:

$$\begin{array}{rl} \text{maximize} & \sum_{m=1}^M \sigma_m u_m \\ \text{w.r.t.} & \boldsymbol{y} \in \{0,1\}^N, \ \boldsymbol{u} \in \{0,1\}^M \\ \text{s.t.} & u_m = \bigvee_{n \in \mathcal{I}_m} y_n, \ \forall m \in [M] \\ & \sum_{n=1}^N L_n y_n \leq B, \end{array} \qquad (3)$$

where we used the notation $[M] := \{1, \ldots, M\}$. This can be converted into an ILP and addressed with off-the-shelf solvers (Gillick et al., 2008). A drawback of this approach is that solving an ILP exactly is NP-hard. Even though existing commercial solvers can solve most instances with a moderate speed, they still exhibit poor worst-case behaviour; this is exacerbated when there is the need to combine an extractive component with other modules, as in compressive summarization (§3).

[1]Previous work has modeled concepts as events (Filatova and Hatzivassiloglou, 2004), salient words (Lin and Bilmes, 2010), and word bigrams (Gillick et al., 2008). In the sequel, we assume concepts are word $k$-grams, but our model can handle other representations, such as phrases or predicate-argument structures.

## 2.2 A Dual Decomposition Formulation

We next describe how the problem in Eq. 3 can be addressed with **dual decomposition**, a class of optimization techniques that tackle the dual of combinatorial problems in a modular, extensible, and parallelizable manner (Komodakis et al., 2007; Rush et al., 2010). In particular, we employ **alternating directions dual decomposition** (AD$^3$; Martins et al., 2011a, 2012) for solving a linear relaxation of Eq. 3. AD$^3$ resembles the subgradient-based algorithm of Rush et al. (2010), but it enjoys a faster convergence rate. Both algorithms split the original problem into several **components**, and then iterate between solving independent **local subproblems** at each component and adjusting multipliers to promote an agreement.[2] The difference between the two methods is that the AD$^3$ local subproblems, instead of requiring the computation of a locally optimal configuration, require solving a local *quadratic* problem. Martins et al. (2011b) provided linear-time solutions for several logic constraints, with applications to syntax and frame-semantic parsing (Das et al., 2012). We will see that AD$^3$ can also handle budget and knapsack constraints efficiently.

To tackle Eq. 3 with dual decomposition, we split the coverage-based summarizer into the following $M + 1$ components (one per constraint):

1. For each of the $M$ concepts in $\mathcal{C}(\mathcal{D})$, one component for imposing the logic constraint in Eq. 3. This corresponds to the OR-WITH-OUTPUT factor described by Martins et al. (2011b); the AD$^3$ subproblem for the $m$th factor can be solved in time $O(|\mathcal{I}_m|)$.
2. Another component for the knapsack constraint. This corresponds to a (novel) KNAPSACK factor, whose AD$^3$ subproblem is solvable in time $O(N)$. The actual algorithm is described in the appendix (Algorithm 1).[3]

## 3 Compressive Summarization

We now turn to **compressive summarization**, which does not limit the summary sentences to be verbatim extracts from the original documents; instead, it allows the extraction of *compressed* sentences where some words can be deleted.

Formally, let us express each sentence of $\mathcal{D}$ as a sequence of word tokens, $s_n := \langle t_{n,\ell}\rangle_{\ell=0}^{L_n}$, where $t_{n,0} \equiv \$$ is a dummy symbol. We represent a **compression** of $s_n$ as an indicator vector $\boldsymbol{z}_n := \langle z_{n,\ell}\rangle_{\ell=0}^{L_n}$, where $z_{n,\ell} = 1$ if the $\ell$th word is included in the compression. By convention, the dummy symbol is included if and only if the remaining compression is non-empty. A compressive summary can then be represented by an indicator vector $\boldsymbol{z}$ which is the concatenation of $N$ such vectors, $\boldsymbol{z} = \langle \boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\rangle$; each position in this indicator vector is indexed by a sentence $n \in [N]$ and a word position $\ell \in \{0\} \cup [L_n]$.

Models for compressive summarization were proposed by Martins and Smith (2009) and Berg-Kirkpatrick et al. (2011) by combining extraction and compression scores. Here, we follow the latter work, by combining a coverage score function $g$ with sentence-level compression score functions $h_1, \ldots, h_N$. This yields the decoding problem:

$$\begin{array}{rl} \text{maximize} & g(\boldsymbol{z}) + \sum_{n=1}^{N} h_n(\boldsymbol{z}_n) \\ \text{w.r.t.} & \boldsymbol{z}_n \in \{0,1\}^{L_n},\ \forall n \in [N] \\ \text{s.t.} & \sum_{n=1}^{N} \sum_{\ell=1}^{L_n} z_{n,\ell} \leq B. \end{array} \tag{4}$$

### 3.1 Coverage Model

We use a coverage function similar to Eq. 2, but taking a compressive summary $\boldsymbol{z}$ as argument:

$$g(\boldsymbol{z}) = \sum_{m=1}^{M} \sigma_m u_m(\boldsymbol{z}), \tag{5}$$

where we redefine $u_m$ as follows. First, we parametrize each occurrence of the $m$th concept (assumed to be a $k$-gram) as a triple $\langle n, \ell_s, \ell_e\rangle$, where $n$ indexes a sentence, $\ell_s$ indexes a start position within the sentence, and $\ell_e$ indexes the end position. We denote by $\mathcal{T}_m$ the set of triples representing all occurrences of the $m$th concept in the original text, and we associate an indicator variable $z_{n,\ell_s:\ell_e}$ to each member of this set. We then define $u_m(\boldsymbol{z})$ via the following logic constraints:

- A **concept type** is selected if *some* of its $k$-gram tokens are selected:

$$u_m(\boldsymbol{y}) := \bigvee_{\langle n,\ell_s,\ell_e\rangle \in \mathcal{T}_m} z_{n,\ell_s:\ell_e}. \tag{6}$$

- A $k$-gram **concept token** is selected if *all* its words are selected:

$$z_{n,\ell_s:\ell_e} := \bigwedge_{\ell=\ell_s}^{\ell_e} z_{n,\ell}. \tag{7}$$

[2] For details about dual decomposition and Lagrangian relaxation, see the recent tutorial by Rush and Collins (2012).

[3] The AD$^3$ subproblem in this case corresponds to computing an Euclidean projection onto the knapsack polytope (Eq. 11). Others addressed the related, but much harder, integer quadratic knapsack problem (McDonald, 2007).

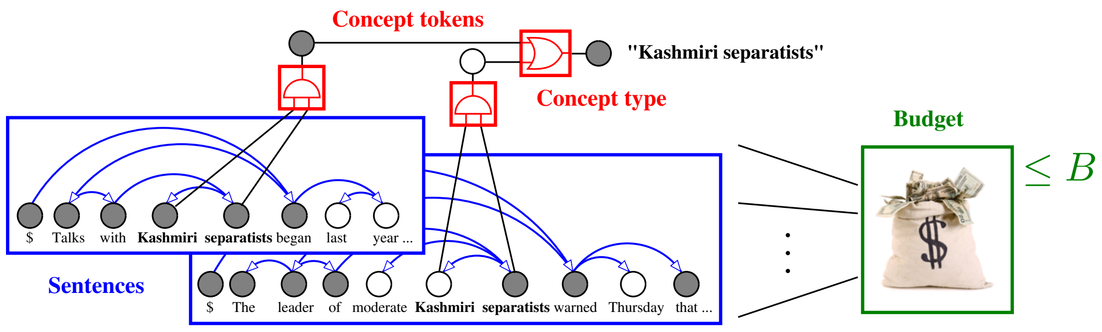
Concept tokens
"Kashmiri separatists"
Concept type
Budget
$\leq B$
Sentences
$ Talks with Kashmiri separatists began last year ...
$ The leader of moderate Kashmiri separatists warned Thursday that ...

Figure 1: Components of our compressive summarizer. Factors depicted in blue belong to the compression model, and aim to enforce grammaticality. The logic factors in red form the coverage component. Finally, the budget factor, in green, is connected to the word nodes; it ensures that the summary fits the word limit. Shaded circles represent active variables while white circles represent inactive variables.

We set concept scores as $\sigma_m := \boldsymbol{w} \cdot \Phi_{\text{cov}}(\mathcal{D}, c_m)$, where $\Phi_{\text{cov}}(\mathcal{D}, c_m)$ is a vector of features (described in §3.5) and $\boldsymbol{w}$ the corresponding weights.

## 3.2 Compression Model

For the compression score function, we follow Martins and Smith (2009) and decompose it as a sum of local score functions $\rho_{n,\ell}$ defined on dependency arcs:

$$h_n(\boldsymbol{z}_n) := \textstyle\sum_{\ell=1}^{L_n} \rho_{n,\ell}(z_{n,\ell}, z_{n,\pi(\ell)}), \qquad (8)$$

where $\pi(\ell)$ denotes the index of the word which is the parent of the $\ell$th word in the dependency tree (by convention, the root of the tree is the dummy symbol). To model the event that an arc is "cut" by disconnecting a child from its head, we define **arc-deletion scores** $\rho_{n,\ell}(0,1) := \boldsymbol{w} \cdot \Phi_{\text{comp}}(s_n, \ell, \pi(\ell))$, where $\Phi_{\text{comp}}$ is a feature map, which is described in detail in §3.5. We set $\rho_{n,\ell}(0,0) = \rho_{n,\ell}(1,1) = 0$, and $\rho_{n,\ell}(1,0) = -\infty$, to allow only the deletion of entire subtrees.

A crucial fact is that one can maximize Eq. 8 efficiently with dynamic programming (using the Viterbi algorithm for trees); the total cost is linear in $L_n$. We will exploit this fact in the dual decomposition framework described next.[4]

## 3.3 A Dual Decomposition Formulation

In previous work, the optimization problem in Eq. 4 was converted into an ILP and fed to an off-the-shelf solver (Martins and Smith, 2009; Berg-Kirkpatrick et al., 2011; Woodsend and Lapata, 2012). Here, we employ the AD$^3$ algorithm, in a similar manner as described in §2, but with an additional component for the sentence compressor, and slight modifications in the other components. We have the following $N + M + \sum_{m=1}^{M} |\mathcal{T}_m| + 1$ components in total, illustrated in Figure 1:

1. For each of the $N$ sentences, one component for the **compression model**. The AD$^3$ quadratic subproblem for this factor can be addressed by solving a *sequence of linear subproblems*, as described by Martins et al. (2012). Each of these subproblems corresponds to maximizing an objective function of the same form as Eq. 8; this can be done in $O(L_n)$ time with dynamic programming, as discussed in §3.2.

2. For each of the $M$ concept types in $\mathcal{C}(\mathcal{D})$, one OR-WITH-OUTPUT factor for the logic constraint in Eq. 6. This is analogous to the one described for the extractive case.

3. For each $k$-gram concept token in $\mathcal{T}_m$, one AND-WITH-OUTPUT factor that imposes the constraint in Eq. 7. This factor was described by Martins et al. (2011b) and its AD$^3$ subproblem can be solved in time linear in $k$.

4. Another component linked to all the words imposing that at most $B$ words can be selected; this is done via a BUDGET factor, a particular case of KNAPSACK. The runtime of this AD$^3$ subproblem is linear in the number of words.

In addition, we found it useful to add a second BUDGET factor limiting the number of sentences that can be selected to a prescribed value $K$. We set $K = 6$ in our experiments.

[4]The same framework can be readily adapted to other compression models that are efficiently decodable, such as the semi-Markov model of McDonald (2006), which would allow incorporating a language model for the compression.

### 3.4 Rounding Strategy

Recall that the problem in Eq. 4 is NP-hard, and that AD$^3$ is solving a linear relaxation. While there are ways of wrapping AD$^3$ in an exact search algorithm (Das et al., 2012), such strategies work best when the solution of the relaxation has few fractional components, which is typical of parsing and translation problems (Rush et al., 2010; Chang and Collins, 2011), and attractive networks (Taskar et al., 2004). Unfortunately, this is not the case in summarization, where concepts "compete" with each other for inclusion in the summary, leading to frustrated cycles. We chose instead to adopt a fast and simple rounding procedure for obtaining a summary from a fractional solution.

The procedure works as follows. First, solve the LP relaxation using AD$^3$, as described above. This yields a solution $\boldsymbol{z}^*$, where each component lies in the unit interval $[0, 1]$. If these components are all integer, then we have a certificate that this is the optimal solution. Otherwise, we collect the $K$ sentences with the highest values of $z^*_{n,0}$ ("posteriors" on sentences), and seek the feasible summary which is the closest (in Euclidean distance) to $\boldsymbol{z}^*$, while only containing those sentences. This can be computed exactly in time $O(B \sum_{k=1}^{K} L_{n_k})$, through dynamic programming.[5]

### 3.5 Features and Hard Constraints

As Berg-Kirkpatrick et al. (2011), we used stemmed word bigrams as concepts, to which we associate the following concept features ($\Phi_{\text{cov}}$): indicators for document counts, features indicating if each of the words in the bigram is a stopword, the earliest position in a document each concept occurs, as well as two and three-way conjunctions of these features.

For the compression model, we include the following arc-deletion features ($\Phi_{\text{comp}}$):

- the dependency label of the arc being deleted, as well as its conjunction with the part-of-speech tag of the head, of the modifier, and of both;
- the dependency labels of the arc being deleted and of its parent arc;
- the modifier tag, if the modifier is a function word modifying a verb ;
- a feature indicating whether the modifier or any of its descendants is a negation word;
- indicators of whether the modifier is a temporal word (*e.g.*, *Friday*) or a preposition pointing to a temporal word (*e.g.*, *on Friday*).

In addition, we included hard constraints to prevent the deletion of certain arcs, following previous work in sentence compression (Clarke and Lapata, 2008). We never delete arcs whose dependency label is `SUB`, `OBJ`, `PMOD`, `SBAR`, `VC`, or `PRD` (this makes sure we preserve subjects and objects of verbs, arcs departing from prepositions or complementizers, and that we do not break verb chains or predicative complements); arcs linking to a conjunction word or siblings of such arcs (to prevent inconsistencies in handling coordinative conjunctions); arcs linking verbs to other verbs, to adjectives (*e.g.*, *make available*), to verb particles (*e.g.*, *settle down*), to the word *that* (*e.g.*, *said that*), or to the word *to* if it is a leaf (*e.g.*, *allowed to come*); arcs pointing to negation words, cardinal numbers, or determiners; and arcs connecting two proper nouns or words within quotation marks.

## 4 Multi-Task Learning

We next turn to the problem of learning the model from training data. Prior work in compressive summarization has followed one of two strategies: Martins and Smith (2009) and Woodsend and Lapata (2012) learn the extraction and compression models separately, and then post-combine them, circumventing the lack of fully annotated data. Berg-Kirkpatrick et al. (2011) gathered a small dataset of manually compressed summaries, and trained with full supervision. While the latter approach is statistically more principled, it has the disadvantage of requiring fully annotated data, which is difficult to obtain in large quantities. On the other hand, there is plenty of data containing manually written abstracts (from the DUC and TAC conferences) and user-generated text (from Wikipedia) that may provide useful weak supervision.

With this in mind, we put together a **multi-task learning** framework for compressive summarization (which we name task #1). The goal is to take advantage of existing data for related tasks, such as extractive summarization (task #2), and sentence compression (task #3). The three tasks are instances of **structured predictors** (Bakır et

[5]Briefly, if we link the roots of the $K$ sentences to a super-root node, the problem above can be transformed into that of finding the best configuration in the resulting binary tree subject to a budget constraint. We omit details for space.

| Tasks | Features | Decoder |
|---|---|---|
| Comp. summ. (#1) | $\Phi_{\text{cov}}$, $\Phi_{\text{comp}}$ | AD$^3$ (solve Eq. 4) |
| Extr. summ. (#2) | $\Phi_{\text{cov}}$ | AD$^3$ (solve Eq. 3) |
| Sent. comp. (#3) | $\Phi_{\text{comp}}$ | dyn. prg. (max. Eq. 8) |

Table 1: Features and decoders used for each task.

al., 2007), and for all of them we assume feature-based models that decompose over "parts":

- For the compressive summarization task, the parts correspond to **concept features** (§3.1) and to **arc-deletion features** (§3.2).
- For the extractive summarization task, there are parts for **concept features** only.
- For the sentence compression task, the parts correspond to **arc-deletion features** only.

This is summarized in Table 1. Features for the three tasks are populated into feature vectors $\Phi_1(x, y)$, $\Phi_2(x, y)$, and $\Phi_3(x, y)$, respectively, where $\langle x, y\rangle$ denotes a task-specific input-output pair. We assume the feature vectors are all $D$ dimensional, where we place zeros in entries corresponding to parts that are absent. Note that this setting is very general and applies to arbitrary structured prediction problems (not just summarization), the only assumption being that some parts are shared between different tasks.

Next, we associate weight vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \in \mathbb{R}^D$ to each task, along with a "shared" vector $\boldsymbol{w}$. Each task makes predictions according to the rule:

$$\widehat{y} := \arg\max_y \; (\boldsymbol{w} + \boldsymbol{v}_k) \cdot \Phi_k(x, y), \qquad (9)$$

where $k \in \{1, 2, 3\}$. This setting is equivalent to the approach of Daumé (2007) for domain adaptation, which consists in splitting each feature into task-component features and a shared feature; but here we do not duplicate features explicitly. To learn the weights, we regularize the weight vectors separately, and assume that each task has its own loss function $\mathcal{L}_k$, so that the total loss $\mathcal{L}$ is a weighted sum $\mathcal{L}(\boldsymbol{w}, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3) := \sum_{k=1}^{3} \sigma_k \mathcal{L}_k(\boldsymbol{w} + \boldsymbol{v}_k)$. This yields the following objective function to be minimized:

$$F(\boldsymbol{w}, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3) = \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{k=1}^{3} \frac{\lambda_k}{2}\|\boldsymbol{v}_k\|^2 + \frac{1}{N}\sum_{k=1}^{3} \sigma_k \mathcal{L}_k(\boldsymbol{w} + \boldsymbol{v}_k), \qquad (10)$$

where $\lambda$ and the $\lambda_k$'s are regularization constants, and $N$ is the total number of training instances.[6] In our experiments (§5), we let the $\mathcal{L}_k$'s be structured hinge losses (Taskar et al., 2003; Tsochantaridis et al., 2004), where the corresponding cost functions are concept recall (for task #2), precision of arc deletions (for task #3), and a combination thereof (for task #1).[7] These losses were normalized, and we set $\sigma_k = N/N_k$, where $N_k$ is the number of training instances for the $k$th task. This ensures all tasks are weighted evenly. We used the same rationale to set $\lambda = \lambda_1 = \lambda_2 = \lambda_3$, choosing this value through cross-validation in the dev set.

We optimize Eq. 10 with stochastic subgradient descent. This leads to update rules of the form

$$\begin{aligned} \boldsymbol{w} &\leftarrow (1 - \eta_t \lambda)\boldsymbol{w} - \eta_t \sigma_k \tilde{\nabla} \mathcal{L}_k(\boldsymbol{w} + \boldsymbol{v}_k) \\ \boldsymbol{v}_j &\leftarrow (1 - \eta_t \lambda_j)\boldsymbol{v}_j - \eta_t \delta_{jk} \sigma_k \tilde{\nabla} \mathcal{L}_k(\boldsymbol{w} + \boldsymbol{v}_k), \end{aligned}$$

where $\tilde{\nabla}\mathcal{L}_k$ are stochastic subgradients for the $k$th task, that take only a single instance into account, and $\delta_{jk} = 1$ if and only if $j = k$. Stochastic subgradients can be computed via cost-augmented decoding (see footnote 7).

Interestingly, Eq. 10 subsumes previous approaches to train compressive summarizers. The limit $\lambda \to \infty$ (keeping the $\lambda_k$'s fixed) forces $\boldsymbol{w} \to 0$, decoupling all the tasks. In this limit, inference for task #1 (compressive summarization) is based solely on the model learned from that task's data, recovering the approach of Berg-Kirkpatrick et al. (2011). In the other extreme, setting $\sigma_1 = 0$ simply ignores task #1's training data. As a result, the optimal $\boldsymbol{v}_1$ will be a vector of zeros; since tasks #2 and #3 have no parts in common, the objective will decouple into a sum of two independent terms

[6]Note that, by substituting $\boldsymbol{u}_k := \boldsymbol{w} + \boldsymbol{v}_k$ and solving for $\boldsymbol{w}$, the problem in Eq. 10 becomes that of minimizing the sum of the losses with a penalty for the (weighted) variance of the vectors $\{\boldsymbol{0}, \boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3\}$, regularizing the difference towards their average, as in Evgeniou and Pontil (2004). This is similar to the hierarchical joint learning approach of Finkel and Manning (2010), except that our goal is to learn a new task (compressive summarization) instead of combining tasks.

[7]Let $\mathcal{Y}_k$ denote the output set for the $k$th task. Given a task-specific cost function $\Delta_k : \mathcal{Y}_k \times \mathcal{Y}_k \to \mathbb{R}$, and letting $\langle x_t, y_t\rangle_{t=1}^{T}$ be the labeled dataset for this task, the structured hinge loss takes the form $\mathcal{L}_k(\boldsymbol{u}_k) := \sum_t \max_{y' \in \mathcal{Y}_k}(\boldsymbol{u}_k \cdot (\Phi_k(x_t, y') - \Phi_k(x_t, y_t)) + \Delta_k(y', y_t))$. The inner maximization over $y'$ is called the *cost-augmented decoding problem*: it differs from Eq. 9 by the inclusion of the cost term $\Delta_k(y', y_t)$. Our costs decompose over the model's factors, hence any decoder for Eq. 9 can be used for the maximization above: for tasks #1–#2, we solve a relaxation by running AD$^3$ without rounding, and for task #3 we use dynamic programming; see Table 1.

involving $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$, which is equivalent to training the two tasks separately and post-combining the models, as Martins and Smith (2009) did.

## 5 Experiments

### 5.1 Experimental setup

We evaluated our compressive summarizers on data from the Text Analysis Conference (TAC) evaluations. We use the same splits as previous work (Berg-Kirkpatrick et al., 2011; Woodsend and Lapata, 2012): the non-update portions of TAC-2009 for training and TAC-2008 for testing. In addition, we reserved TAC-2010 as a devset. The test partition contains 48 multi-document summarization problems; each provides 10 related news articles as input, and asks for a summary with up to 100 words, which is evaluated against four manually written abstracts. We ignored all the query information present in the TAC datasets.

**Single-Task Learning.** In the single-task experiments, we trained a compressive summarizer on the dataset disclosed by Berg-Kirkpatrick et al. (2011), which contains manual compressive summaries for the TAC-2009 data. We trained a structured SVM with stochastic subgradient descent; the cost-augmented inference problems are relaxed and solved with AD$^3$, as described in §3.3.[8] We followed the procedure described in Berg-Kirkpatrick et al. (2011) to reduce the number of candidate sentences: scores were defined for each sentence (the sum of the scores of the concepts they cover), and the best-scored sentences were greedily selected up to a limit of 1,000 words. We then tagged and parsed the selected sentences with TurboParser.[9] Our choice of a dependency parser was motivated by our will for a fast system; in particular, TurboParser attains top accuracies at a rate of 1,200 words per second, keeping parsing times below 1 second for each summarization problem.

**Multi-Task Learning.** For the multi-task experiments, we also used the dataset of Berg-Kirkpatrick et al. (2011), but we augmented the training data with extractive summarization and sentence compression datasets, to help train the compressive summarizer. For extractive summarization, we used the DUC 2003 and 2004 datasets (a total of 80 multi-document summarization problems). We generated oracle extracts by maximizing bigram recall with respect to the manual abstracts, as described in Berg-Kirkpatrick et al. (2011). For sentence compression, we adapted the Simple English Wikipedia dataset of Woodsend and Lapata (2011), containing aligned sentences for 15,000 articles from the English and Simple English Wikipedias. We kept only the 4,481 sentence pairs corresponding to deletion-based compressions.

### 5.2 Results

Table 2 shows the results. The top rows refer to three strong baselines: the ICSI-1 extractive coverage-based system of Gillick et al. (2008), which achieved the best ROUGE scores in the TAC-2008 evaluation; the compressive summarizer of Berg-Kirkpatrick et al. (2011), denoted BGK'11; and the multi-aspect compressive summarizer of Woodsend and Lapata (2012), denoted WL'12. All these systems require ILP solvers. The bottom rows show the results achieved by our implementation of a pure *extractive* system (similar to the learned extractive summarizer of Berg-Kirkpatrick et al., 2011); a system that *post-combines* extraction and compression components trained separately, as in Martins and Smith (2009); and our compressive summarizer trained as a *single task*, and in the *multi-task* setting.

The ROUGE and Pyramid scores show that the compressive summarizers (when properly trained) yield considerable benefits in content coverage over extractive systems, confirming the results of Berg-Kirkpatrick et al. (2011). Comparing the two bottom rows, we see a clear benefit by training in the multi-task setting, with a consistent gain in both coverage and linguistic quality. Our ROUGE-2 score (12.30%) is, to our knowledge, the highest reported on the TAC-2008 dataset, with little harm in grammaticality with respect to an extractive system that preserves the original sentences. Figure 2 shows an example summary.

### 5.3 Runtimes

We conducted another set of experiments to compare the runtime of our compressive summarizer based on AD$^3$ with the runtimes achieved by GLPK, the ILP solver used by Berg-Kirkpatrick et al. (2011). We varied the maximum number of it-

[8] We use the AD$^3$ implementation in `http://www.ark.cs.cmu.edu/AD3`, setting the maximum number of iterations to 200 at training time and 1000 at test time. We extended the code to handle the knapsack and budget factors; the modified code will be part of the next release (AD$^3$ 2.1).

[9] `http://www.ark.cs.cmu.edu/TurboParser`

| System | R-2 | R-SU4 | Pyr | LQ |
|---|---|---|---|---|
| ICSI-1 | 11.03 | 13.96 | $34.5^\dagger$ | – |
| BGK'11 | 11.71 | 14.47 | $41.3^\dagger$ | – |
| WL'12 | 11.37 | 14.47 | – | – |
| Extractive | 11.16 | 14.07 | 36.0 | **4.6** |
| Post-comb. | 11.07 | 13.85 | 38.4 | 4.1 |
| Single-task | 11.88 | 14.86 | 41.0 | 3.8 |
| Multi-task | **12.30** | **15.18** | **42.6** | 4.2 |

Table 2: Results for compressive summarization. Shown are the ROUGE-2 and ROUGE SU-4 recalls with the default options from the ROUGE toolkit (Lin, 2004); Pyramid scores (Nenkova and Passonneau, 2004); and linguistic quality scores, scored between 1 (very bad) to 5 (very good). For Pyramid, the evaluation was performed by two annotators, each evaluating half of the problems; scores marked with $^\dagger$ were computed by different annotators and are not directly comparable. Linguistic quality was evaluated by two linguists; we show the average of the reported scores.

| Solver | Runtime (sec.) | ROUGE-2 |
|---|---|---|
| ILP Exact | 10.394 | 12.40 |
| LP-Relax. | 2.265 | 12.38 |
| $\text{AD}^3$-5000 | 0.952 | 12.38 |
| $\text{AD}^3$-1000 | 0.406 | 12.30 |
| $\text{AD}^3$-200 | 0.159 | 12.15 |
| Extractive (ILP) | 0.265 | 11.16 |

Table 3: Runtimes of several decoders on a Intel Core i7 processor @2.8 GHz, with 8GB RAM. For each decoder, we show the average time taken to solve a summarization problem in TAC-2008. The reported runtimes of $\text{AD}^3$ and LP-Relax include the time taken to round the solution (§3.4), which is 0.029 seconds on average.

erations of $\text{AD}^3$ in $\{200, 1000, 5000\}$, and clocked the time spent by GLPK to solve the exact ILPs and their relaxations. Table 3 depicts the results.[10]

We see that our proposed configuration ($\text{AD}^3$-1000) is orders of magnitude faster than the ILP solver, and 5 times faster than its relaxed variant, while keeping similar accuracy levels.[11] The gain when the number of iterations in $\text{AD}^3$ is increased to 5000 is small, given that the runtime is more than doubled; accuracy starts to suffer, however, if the number of iterations is reduced too much. In practice, we observed that the final rounding procedure was crucial, as only 2 out of the 48 test problems had integral solutions (arguably because of the "repulsive" nature of the network, as hinted in §3.4). For comparison, we also report in the bottom row the average runtime of the learned extractive baseline. We can see that our system's runtime is competitive with this baseline. To our knowledge, this is the first time a compressive summarizer achieves such a favorable accuracy/speed tradeoff.

[10]Within dual decomposition algorithms, we verified experimentally that $\text{AD}^3$ is substantially faster than the subgradient algorithm, which is consistent with previous findings (Martins et al., 2011b).

[11]The runtimes obtained with the exact ILP solver seem slower than those reported by Berg-Kirkpatrick et al. (2011). (around 1.5 sec. on average, according to their Fig. 3). We conjecture that this difference is due to the restricted set of subtrees that can be deleted by Berg-Kirkpatrick et al. (2011), which greatly reduces their search space.

Japan dispatched four military ships to help Russia rescue seven crew members aboard a small submarine trapped on the seabed in the Far East. The Russian Pacific Fleet said the crew had 120 hours of oxygen reserves *on board when the submarine submerged at midday Thursday (2300 GMT Wednesday) off the Kamchatka peninsula, the stretch of Far Eastern Russia facing the Bering Sea*. The submarine*, used in rescue, research and intelligence-gathering missions,* became stuck at the bottom of the Bay of Berezovaya off Russia's Far East coast when its propeller was caught *in a fishing net*. The Russian submarine had been tending an underwater antenna mounted to the sea floor *when it became snagged on a wire helping to stabilize a ventilation cable attached to the antenna*. Rescue crews lowered a British remote-controlled underwater vehicle to a Russian mini-submarine trapped *deep* under the Pacific Ocean, hoping to free the vessel and its seven trapped crewmen *before their air supply ran out*.

Figure 2: Example summary from our compressive system. Removed text is *grayed out*.

## 6 Conclusions

We presented a multi-task learning framework for compressive summarization, leveraging data for related tasks in a principled manner. We decode with $\text{AD}^3$, a fast and modular dual decomposition algorithm which is orders of magnitude faster than ILP-based approaches. Results show that the state of the art is improved in automatic and manual metrics, with speeds close to extractive systems.

Our approach is modular and easy to extend. For example, a different compression model could incorporate rewriting rules to enable compressions that go beyond word deletion, as in Cohn and Lapata (2008). Other aspects may be added as additional components in our dual decomposition framework, such as query information (Schilder and Kondadadi, 2008), discourse constraints (Clarke and Lapata, 2007), or lexical preferences (Woodsend and Lapata, 2012). Our multi-task approach may be used to jointly learn parameters for these aspects; the dual decomposition algorithm ensures that optimization remains tractable even with many components.

## A Projection Onto Knapsack

This section describes a linear-time algorithm (Algorithm 1) for solving the following problem:

$$\begin{aligned} &\text{minimize } \|\boldsymbol{z} - \boldsymbol{a}\|^2 \\ &\quad \text{w.r.t. } z_n \in [0,1], \ \forall n \in [N], \\ &\quad\quad \text{s.t. } \textstyle\sum_{n=1}^N L_n z_n \le B, \end{aligned} \tag{11}$$

where $\boldsymbol{a} \in \mathbb{R}^N$ and $L_n \ge 0, \forall n \in [N]$. This includes as special cases the problems of projecting onto a budget constraint ($L_n = 1, \forall n$) and onto the simplex (same, plus $B = 1$).

Let $\text{clip}(t) := \max\{0, \min\{1, t\}\}$. Algorithm 1 starts by clipping $\boldsymbol{a}$ to the unit interval; if that yields a $\boldsymbol{z}$ satisfying $\sum_{n=1}^N L_n z_n \le B$, we are done. Otherwise, the solution of Eq. 11 must satisfy $\sum_{n=1}^N L_n z_n = B$. It can be shown from the KKT conditions that the solution is of the form $z_n^* := \text{clip}(a_n + \tau^* L_n)$ for a constant $\tau^*$ lying in a particular interval of split-points (line 11). To seek this constant, we use an algorithm due to Pardalos and Kovoor (1990) which iteratively shrinks this interval. The algorithm requires computing medians as a subroutine, which can be done in linear time (Blum et al., 1973). The overall complexity in $O(N)$ (Pardalos and Kovoor, 1990).

**Algorithm 1** Projection Onto Knapsack.

```
 1: input: $\boldsymbol{a} := \langle a_n \rangle_{n=1}^N$, costs $\langle L_n \rangle_{n=1}^N$, maximum cost $B$
 2:
 3: {Try to clip into unit interval:}
 4: Set $z_n \leftarrow \text{clip}(a_n)$ for $n \in [N]$
 5: if $\sum_{n=1}^N L_n z_n \le B$ then
 6:     Return $\boldsymbol{z}$ and stop.
 7: end if
 8:
 9: {Run Pardalos and Kovoor (1990)'s algorithm:}
10: Initialize working set $\mathcal{W} \leftarrow \{1, \ldots, K\}$
11: Initialize set of split points:
        $\mathcal{P} \leftarrow \{-a_n/L_n, (1-a_n)/L_n\}_{n=1}^N \cup \{\pm\infty\}$
12: Initialize $\tau_{\text{L}} \leftarrow -\infty$, $\tau_{\text{R}} \leftarrow \infty$, $s_{\text{tight}} \leftarrow 0$, $\xi \leftarrow 0$.
13: while $\mathcal{W} \ne \varnothing$ do
14:     Compute $\tau \leftarrow \text{Median}(\mathcal{P})$
15:     Set $s \leftarrow s_{\text{tight}} + \xi\tau + \sum_{n\in\mathcal{W}} L_n \text{clip}(a_n + \tau L_n)$
16:     If $s \le B$, set $\tau_{\text{L}} \leftarrow \tau$; if $s \ge B$, set $\tau_{\text{R}} \leftarrow \tau$
17:     Reduce set of split points: $\mathcal{P} \leftarrow \mathcal{P} \cap [\tau_{\text{L}}, \tau_{\text{R}}]$
18:     Define the sets:
            $\mathcal{W}_{\text{L}} := \{n \in \mathcal{W} \mid (1-a_n)/L_n < \tau_{\text{L}}\}$
            $\mathcal{W}_{\text{R}} := \{n \in \mathcal{W} \mid -a_n/L_n > \tau_{\text{R}}\}$
            $\mathcal{W}_{\text{M}} := \left\{ n \in \mathcal{W} \,\middle|\, -\frac{a_n}{L_n} \le \tau_{\text{L}} \wedge \frac{1-a_n}{L_n} \ge \tau_{\text{R}} \right\}$
19:     Update working set: $\mathcal{W} \leftarrow \mathcal{W} \setminus (\mathcal{W}_{\text{L}} \cup \mathcal{W}_{\text{R}} \cup \mathcal{W}_{\text{M}})$
20:     Update tight-sum:
            $s_{\text{tight}} \leftarrow s_{\text{tight}} + \sum_{n\in\mathcal{W}_{\text{L}}} L_n(1-a_n) - \sum_{n\in\mathcal{W}_{\text{R}}} L_n a_n$
21:     Update slack-sum: $\xi \leftarrow \xi + \sum_{n\in\mathcal{W}_{\text{M}}} L_n^2$
22: end while
23: Define $\tau^* \leftarrow (B - \sum_{i=1}^N L_i a_i - s_{\text{tight}})/\xi$
24: Set $z_n \leftarrow \text{clip}(a_n + \tau^* L_n), \ \forall n \in [N]$
25: output: $\boldsymbol{z} := \langle z_n \rangle_{n=1}^N$.
```

## Acknowledgments

We thank all reviewers for their insightful comments; Trevor Cohn for helpful discussions about multi-task learning; Taylor Berg-Kirkpatrick for answering questions about their summarizer and for providing code; and Helena Figueira and Pedro Mendes for helping with manual evaluation. This work was partially supported by the EU/FEDER programme, QREN/POR Lisboa (Portugal), under the Discooperio project (contract 2011/18501), and by a FCT grant PTDC/EEI-SII/2312/2012.

## References

G. Bakır, T. Hofmann, B. Schölkopf, A. Smola, B. Taskar, and S. Vishwanathan. 2007. *Predicting Structured Data*. The MIT Press.

P. B. Baxendale. 1958. Machine-made index for technical literature—an experiment. *IBM Journal of Research Development*, 2(4):354–361.

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

Manuel Blum, Robert W Floyd, Vaughan Pratt, Ronald L Rivest, and Robert E Tarjan. 1973. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461.

J. Carbonell and J. Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*.

Y.-W. Chang and M. Collins. 2011. Exact decoding of phrase-based translation models through lagrangian relaxation. In *Proc. of Empirical Methods for Natural Language Processing*.

James Clarke and Mirella Lapata. 2007. Modelling compression with discourse constraints. In *Proc. of Empirical Methods in Natural Language Processing*.

J. Clarke and M. Lapata. 2008. Global Inference for Sentence Compression An Integer Linear Programming Approach. *Journal of Artificial Intelligence Research*, 31:399–429.

T. Cohn and M. Lapata. 2008. Sentence compression beyond word deletion. In *Proc. COLING*.

D. Das, A. F. T. Martins, and N. A. Smith. 2012. An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints. In *Proc. of First Joint Conference on Lexical and Computational Semantics (*SEM)*.

H. Daumé. 2006. *Practical Structured Learning Techniques for Natural Language Processing*. Ph.D. thesis, University of Southern California.

H. Daumé. 2007. Frustratingly easy domain adaptation. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

H. P. Edmundson. 1969. New methods in automatic extracting. *Journal of the ACM*, 16(2):264–285.

T. Evgeniou and M. Pontil. 2004. Regularized multi–task learning. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 109–117. ACM.

Elena Filatova and Vasileios Hatzivassiloglou. 2004. A formal model for information selection in multi-sentence text extraction. In *Proc. of International Conference on Computational Linguistics*.

J.R. Finkel and C.D. Manning. 2010. Hierarchical joint learning: Improving joint parsing and named entity recognition with non-jointly labeled data. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

J. Gillenwater, A. Kulesza, and B. Taskar. 2012. Discovering diverse and salient threads in document collections. In *Proc. of Empirical Methods in Natural Language Processing*.

Dan Gillick, Benoit Favre, and Dilek Hakkani-Tur. 2008. The icsi summarization system at tac 2008. In *Proc. of Text Understanding Conference*.

K. Knight and D. Marcu. 2000. Statistics-based summarization—step one: Sentence compression. In *AAAI/IAAI*.

N. Komodakis, N. Paragios, and G. Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *Proc. of International Conference on Computer Vision*.

J. Kupiec, J. Pedersen, and F. Chen. 1995. A trainable document summarizer. In *SIGIR*.

H. Lin and J. Bilmes. 2010. Multi-document summarization via budgeted maximization of submodular functions. In *Proc. of Annual Meeting of the North American chapter of the Association for Computational Linguistics*.

H. Lin and J. Bilmes. 2012. Learning mixtures of submodular shells with application to document summarization. In *Proc. of Uncertainty in Artificial Intelligence*.

C.-Y. Lin. 2003. Improving summarization performance by sentence compression-a pilot study. In *the Int. Workshop on Inf. Ret. with Asian Languages*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, July.

H. P. Luhn. 1958. The automatic creation of literature abstracts. *IBM Journal of Research Development*, 2(2):159–165.

A. F. T. Martins and N. A. Smith. 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*.

A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing. 2011a. An Augmented Lagrangian Approach to Constrained MAP Inference. In *Proc. of International Conference on Machine Learning*.

A. F. T. Martins, N. A. Smith, P. M. Q. Aguiar, and M. A. T. Figueiredo. 2011b. Dual Decomposition with Many Overlapping Components. In *Proc. of Empirical Methods for Natural Language Processing*.

Andre F. T. Martins, Mario A. T. Figueiredo, Pedro M. Q. Aguiar, Noah A. Smith, and Eric P. Xing. 2012. Alternating Directions Dual Decomposition. *Arxiv preprint arXiv:1212.6550*.

R. McDonald. 2006. Discriminative sentence compression with soft syntactic constraints. In *Proc. of Annual Meeting of the European Chapter of the Association for Computational Linguistics*.

R. McDonald. 2007. A study of global inference algorithms in multi-document summarization. In *ECIR*.

A. Nenkova and R. Passonneau. 2004. Evaluating content selection in summarization: The pyramid method. In *Proceedings of NAACL*, pages 145–152.

Panos M. Pardalos and Naina Kovoor. 1990. An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds. *Mathematical Programming*, 46(1):321–328.

D. R. Radev, H. Jing, and M. Budzikowska. 2000. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. In *the NAACL-ANLP Workshop on Automatic Summarization*.

A.M. Rush and M. Collins. 2012. A Tutorial on Dual Decomposition and Lagrangian Relaxation for Inference in Natural Language Processing. *Journal of Artificial Intelligence Research*, 45:305–362.

A. Rush, D. Sontag, M. Collins, and T. Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of Empirical Methods for Natural Language Processing*.

Frank Schilder and Ravikumar Kondadadi. 2008. Fastsum: Fast and accurate query-based multi-document summarization. In *Proc. of Annual Meeting of the Association for Computational Linguistics*.

R. Sipos, P. Shivaswamy, and T. Joachims. 2012. Large-margin learning of submodular summarization models.

B. Taskar, C. Guestrin, and D. Koller. 2003. Max-margin Markov networks. In *Proc. of Neural Information Processing Systems*.

B. Taskar, V. Chatalbashev, and D. Koller. 2004. Learning associative Markov networks. In *Proc. of International Conference of Machine Learning*.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proc. of International Conference of Machine Learning*.

K. Woodsend and M. Lapata. 2010. Automatic generation of story highlights. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 565–574.

Kristian Woodsend and Mirella Lapata. 2011. Learning to simplify sentences with quasi-synchronous grammar and integer programming. In *Proc. of Empirical Methods in Natural Language Processing*.

Kristian Woodsend and Mirella Lapata. 2012. Multiple aspect summarization using integer linear programming. In *Proc. of Empirical Methods in Natural Language Processing*.

Wen-tau Yih, Joshua Goodman, Lucy Vanderwende, and Hisami Suzuki. 2007. Multi-document summarization by maximizing informative content-words. In *Proc. of International Joint Conference on Artifical Intelligence*.

D. Zajic, B. Dorr, J. Lin, and R. Schwartz. 2006. Sentence compression as a component of a multi-document summarization system. In *the ACL DUC Workshop*.