# Advanced Structured Prediction

Editors:

**Sebastian Nowozin** Sebastian.Nowozin@microsoft.com
*Microsoft Research*
*Cambridge, CB1 2FB, United Kingdom*

**Peter V. Gehler** pgehler@tuebingen.mpg.de
*Max Planck Insitute for Intelligent Systems*
*72076 Tübingen, Germany*

**Jeremy Jancsary** jermyj@microsoft.com
*Microsoft Research*
*Cambridge, CB1 2FB, United Kingdom*

**Christoph Lampert** chl@ist.ac.at
*IST Austria*
*A-3400 Klosterneuburg, Austria*

This is a draft version of the author chapter.

The MIT Press
Cambridge, Massachusetts
London, England

# 1 AD$^3$: A Fast Decoder for Structured Prediction

**André F. T. Martins** atm@priberam.pt
*Priberam Labs & Instituto de Telecomunicações*
*Lisbon, Portugal*

*In this chapter, we present AD$^3$, a new algorithm for approximate maximum a posteriori (MAP) decoding on factor graphs, based on the alternating directions method of multipliers. AD$^3$ can handle many models often encountered in natural language processing (NLP) and information retrieval (IR), such as models with first-order logic constraints, budget and knapsack constraints, and combinations of structured models that are hard to decode jointly. Like other dual decomposition algorithms, AD$^3$ has a modular architecture, where local subproblems are solved independently, and their solutions are gathered to compute a global update. The key characteristic of AD$^3$ is that each local subproblem has a quadratic regularizer, leading to faster convergence, both theoretically and in practice. To solve these AD$^3$ subproblems, we present an active set method that requires only a local MAP decoder (the same requirement as subgradient-based dual decomposition), making AD$^3$ applicable to a wide range of problems. We discuss two recent applications of AD$^3$ in NLP problems: dependency parsing and compressive summarization.*

## 1.1 Introduction

Graphical models enable compact representations of probability distributions, being widely used in natural language processing (NLP), computer vision, signal processing, and computational biology (Pearl, 1988; Lauritzen, 1996; Koller and Friedman, 2009). Given a graphical model, a central prob-

lem is that of decoding the most probable (a.k.a. *maximum a posteriori* – MAP) configuration. Unfortunately, exact MAP decoding is intractable for many graphical models of interest in applications, such as those involving non-local features or structural constraints. This fact has motivated a significant research effort on approximate techniques.

In this chapter, we turn our attention to *LP-MAP decoding* (Schlesinger, 1976), a linear relaxation that underlies recent message-passing and dual decomposition algorithms (Wainwright et al., 2005; Kolmogorov, 2006; Werner, 2007; Komodakis et al., 2007; Globerson and Jaakkola, 2008; Jojic et al., 2010).[1] All these algorithms have a similar consensus-based architecture: they repeatedly perform certain "local" operations in the graph (as outlined in Table 1.1), until some form of local agreement is achieved. The simplest example is the *projected subgradient dual decomposition* (PSDD) algorithm of Komodakis et al. (2007), which has recently enjoyed great success in NLP applications (see Rush and Collins 2012 and references therein). The major drawback of PSDD is that it is too slow to achieve consensus in large problems, requiring $O(1/\epsilon^2)$ iterations for an $\epsilon$-accurate solution.

Here, we introduce an algorithm called AD$^3$ (*alternating directions dual decomposition*), which allies the modularity of dual decomposition with the effectiveness of augmented Lagrangian optimization, via the *alternating directions method of multipliers* (Glowinski and Marroco, 1975; Gabay and Mercier, 1976). AD$^3$ has an iteration bound of $O(1/\epsilon)$, an order of magnitude better than the PSDD algorithm. Like PSDD, AD$^3$ alternates between a *broadcast* operation, where subproblems are assigned to local workers, and a *gather* operation, where the local solutions are assembled by a controller, which produces an estimate of the global solution. The key difference is that AD$^3$ regularizes their local subproblems toward these global estimate, which has the effect of speeding up consensus. In many cases of interest, there are closed-form solutions or efficient procedures for solving the AD$^3$ local subproblems (which are quadratic). For factors lacking such a solution, we introduce an *active set method* which requires only a local MAP decoder (the same requirement as in PSDD). This paves the way for using AD$^3$ with dense or structured factors. AD$^3$ was originally proposed in Martins et al. (2011a) and has been successfully applied to several NLP problems (Martins et al., 2011b, 2013; Das et al., 2012; Almeida and Martins, 2013). An open-source implementation is available at `http://www.ark.cs.cmu.edu/AD3`.

This chapter is organized as follows. We provide background on factor

1. See Chapter **??** of this volume for further details about LP-MAP decoding, along with a generalization of the TRW-S algorithm of Wainwright et al. (2005); Kolmogorov (2006).

| Algorithm | Local Operation |
|---|---|
| Loopy BP (Pearl, 1988) | max-marginals |
| TRW-S (Wainwright et al., 2005; Kolmogorov, 2006) | max-marginals |
| MPLP (Globerson and Jaakkola, 2008) | max-marginals |
| PSDD (Komodakis et al., 2007) | MAP |
| ADD (Jojic et al., 2010) | marginals |
| AD$^3$ (Martins et al., 2011a) | QP/MAP |

**Table 1.1**: Several approximate MAP decoders and the kind of the local operations they need to perform at the factors to pass messages and compute beliefs. With the exception of loopy BP, all algorithms in this table are instances of LP-MAP decoders. In Section 1.6, we will see that the quadratic problems (QP) required by AD$^3$ can be solved as a sequence of local MAP problems.

graphs and the MAP decoding problem in Section 1.2. In Section 1.3, we discuss two important NLP applications, dependency parsing and summarization, along with factor graph representations. In Section 1.4, we discuss the LP-MAP relaxation and the PSDD algorithm. In Section 1.5, we derive AD$^3$ and analyze its convergence properties. The AD$^3$ subproblems are addressed in Section 1.6, where we present the active set method. Section 1.7 reports experiments using AD$^3$ for the two NLP applications above. Finally, related work is discussed in Section 1.8, and Section 1.9 concludes.

## 1.2 Factor Graphs and MAP Decoding

### 1.2.1 Factor Graphs

Let $Y_1, \ldots, Y_M$ be random variables describing a structured output, with each $Y_i$ taking values in a finite set $\mathcal{Y}_i$. We follow the common assumption in structured prediction that some of these variables have strong statistical dependencies. In this chapter, we use *factor graphs* (Tanner, 1981; Kschischang et al., 2001), a convenient way of representing such dependencies that captures directly the factorization assumptions in a model.

**Definition 1.1** (Factor graph)**.** *A factor graph is a bipartite graph* $G := (V, F, E)$*, comprised of:*

- variable nodes $V := \{1, \ldots, M\}$*, corresponding to the variables* $Y_1, \ldots, Y_M$*;*
- factor nodes $F$ *(disjoint from* $V$*);*
- edges $E \subseteq V \times F$ *linking variable nodes to factor nodes.*

We denote by $F(i) := \{\alpha \in F \mid (i, \alpha) \in E\}$ the set of factors linked to

the $i$th variable, and conversely by $V(\alpha) := \{i \in V \mid (i, \alpha) \in E\}$ the set of variables appearing in factor $\alpha$. We use the short notation $\boldsymbol{Y}_\alpha := (Y_i)_{i \in V(\alpha)}$ to refer to tuples of random variables, which take values on the product set $\mathcal{Y}_\alpha := \prod_{i \in V(\alpha)} \mathcal{Y}_i$. We say that the joint probability distribution of $Y_1, \ldots, Y_M$ factors according to $G = (V, F, E)$ if it can be written as

$$\mathbb{P}(Y_1 = y_1, \ldots, Y_M = y_M) \;\propto\; \exp\left(\sum_{i \in V} \boldsymbol{\theta}_i(y_i) + \sum_{\alpha \in F} \boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha)\right), \qquad (1.1)$$

where $\boldsymbol{\theta}_i(\cdot)$ and $\boldsymbol{\theta}_\alpha(\cdot)$ are, respectively, *unary* and *higher-order* log-potential functions. These functions define "local" scores for the variable nodes and for configurations of variables within the factor nodes. To accommodate hard constraints (discussed in Section 1.2.3), we allow these functions to take values in $\bar{\mathbb{R}} := \mathbb{R} \cup \{-\infty\}$.

### 1.2.2 MAP Decoding

Given a probability distribution specified as in (1.1), an important task is that of finding an assignment with maximal probability (the so-called *MAP assignment/configuration*):

$$\widehat{\boldsymbol{y}}_1, \ldots, \widehat{\boldsymbol{y}}_M \;\in\; \operatorname*{argmax}_{y_1, \ldots, y_M} \sum_{i \in V} \boldsymbol{\theta}_i(y_i) + \sum_{\alpha \in F} \boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha). \qquad (1.2)$$

A solver for problem (1.2) is called a *MAP decoder*. In fact, this problem is not specific to probabilistic models: other models, *e.g.*, trained to maximize margin, also lead to maximizations of the form above. Unfortunately, for a general factor graph $G$, this combinatorial problem is NP-hard (Koller and Friedman, 2009), so one must resort to approximations.

In this chapter, we will address a class of approximations based on linear programming relaxations, which will be described formally in Section 1.4.

### 1.2.3 Dense, Hard, and Structured Factors

Along this chapter, we will consider factors of three kinds: *dense factors*, *hard constraint factors*, and *structured factors*.

A dense factor is one whose log-potential function satisfies $\boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha) > -\infty$ for every $\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha$. To represent a dense factor computationally, we need to store $O(|\mathcal{Y}_\alpha|)$ real numbers (one per factor configuration). Since $|\mathcal{Y}_\alpha|$ grows exponentially with the number of variables linked to the factor, dense factors are only tractable for small values of $|V(\alpha)|$.

Hard constraint factors have special log-potential functions that are indi-

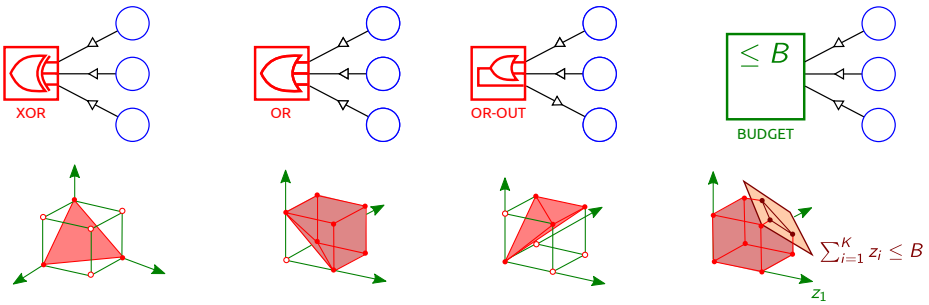


**Figure 1.1**: Hard constraint factors and their marginal polytopes; the AD$^3$ subproblems (1.20) are projections onto these polytopes. The three factors on the left represent one-hot XOR, OR, and OR-with-output logical constraints. The rightmost one is a budget factor, requiring at most $B$ variables to be active.

cators of an *acceptance set* $S_\alpha \subseteq \mathcal{Y}_\alpha$. In other words, $\boldsymbol{\theta}_\alpha(.)$ is of the form:

$$\boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha) := \begin{cases} 0, & \text{if } \boldsymbol{y}_\alpha \in S_\alpha, \\ -\infty, & \text{otherwise.} \end{cases} \tag{1.3}$$

These factors ensure that any configuration not in the acceptance set will have zero probability. This has applications in error-correcting decoding (Richardson and Urbanke, 2008), bipartite graph matching (Duchi et al., 2007), computer vision (Nowozin and Lampert, 2009), and various problems in NLP (Sutton, 2004; Smith and Eisner, 2008; Das et al., 2012). An example is *logic factors* (Martins et al., 2011b), which compute logic functions over binary variables, serving as building blocks for expressing declarative constraints in *first-order logic*. Such constraints are very useful to inject domain knowledge into a problem (Richardson and Domingos, 2006; Chang et al., 2008). Figure 1.1 shows examples of logic factors; see Martins (2012) for how to perform computations with those factors.

Finally, structured factors have log-potential functions with an additional layer of structure, which allows them to be represented in a compact way. For example, they can form a chain model, or a tree, or can even have an

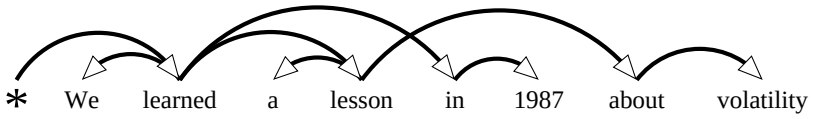


**Figure 1.2**: Example of a sentence (input) and its dependency parse tree (output to be predicted); this is a directed spanning tree where each arc $(h, m)$ represent a syntactic relationships between a *head* word $h$ and the a *modifier* word $m$.

internal factor graph representation[2] themselves, such as

$$\boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha) := \sum_{i \in V(\alpha)} \xi_i(y_i) + \sum_{\beta \in F_\alpha} \xi_\beta(\boldsymbol{y}_\beta), \tag{1.4}$$

where $F_\alpha \subseteq 2^{V(\alpha)}$. Since they have compact representations, structured factors are able to scale to large values of $|V(\alpha)|$.

## 1.3 Factor Graphs for Natural Language Processing

Many problems in NLP can be cast as structured prediction (Smith, 2011). In this section, we describe factor graph representations for two concrete NLP problems: *dependency parsing* and *compressive summarization*. Later, in Section 1.7, we will show experimental results for these two tasks.

### 1.3.1 "Turbo" Parsing

Dependency parsing is an important problem in NLP (Kübler et al., 2009). Given a sentence with $L$ words, to which a special root symbol $*$ is prepended, the goal is to find a directed spanning tree, where each arc represents a syntactic function, linking a *head word* to a *modifier word*.[3] For example, in Figure 1.2, three arcs depart from the head word "learned": one pointing to the modifier "We" (the subject), another to "lesson" (the object), and another to "in" (which introduces a prepositional phrase).

The simplest dependency parsing models are *arc-factored*. In these models,

2. In the literature, structured factors are often not considered as factors on their own, but instead as subgraphs of a larger graph that includes their internal factorization. Here, we consider structured factors explicitly, abstracting away their internal structure. This allows to deal with combinatorial structures that are not directly represented as a graphical model, such as probabilistic context-free grammars.

3. This is the definition of *nonprojective* dependency parsing, the one used throughout this chapter. An formalism is *projective* parsing, which in addition constrains the arcs to be nested. See Kübler et al. (2009) for more details.

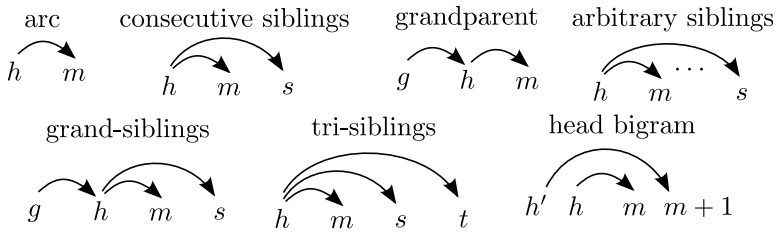


**Figure 1.3**: The parts used in our model. First-order models factor over arcs (Eisner, 1996; McDonald et al., 2005), and second-order models include also consecutive siblings and grandparents (Carreras, 2007). Our parsers add also *arbitrary* siblings (not necessarily consecutive) and head bigrams, as in Martins et al. (2011b), in addition to third-order features for grand- and tri-siblings (Koo and Collins, 2010).

one has $O(L^2)$ scores—one score for every possible arc linking two words—and the goal is to pick the spanning tree maximizing the sum of the scores in its arcs. This problem can be solved efficiently with an algorithm for finding maximal spanning trees (Chu and Liu, 1965; Edmonds, 1967; McDonald et al., 2005). Unfortunately, arc-factored parsers perform poorly, since they do not capture enough interactions among the dependency arcs; and extending the model to include larger substructures (*e.g.* pairs of arcs) renders the decoding problem NP-hard (McDonald and Satta, 2007).

In Martins et al. (2010b), we introduced the concept of "turbo parser," which encapsulates several top-performing approximate dependency parsers (Smith and Eisner, 2008; Martins et al., 2009; Koo et al., 2010). The name designates parsers that decode factor graphs by ignoring the effects caused by the cycles of the graph.[4] These parsers trade off exact decoding by the ability to include scores for more complex substructures, resulting on more expressive models and higher accuracies. Figure 1.3 shows an example of such substructures (or *parts*) that constitute a recent third-order parser (Martins et al., 2013). These parts can be represented by a factor graph that mixes dense, hard constraint, and structured factors, which we next define.

The variable nodes of the factor graph stand for the $O(L^2)$ possible dependency arcs linking each pair of words, each associated with a binary variable. The factors are the following:

- A hard-constraint factor, linked to all the variables, imposing that they jointly define a well-formed tree.
- Head automata factors modeling the left and right sequences of consec-

4. 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).

utive siblings, grandparents, grand-siblings, and tri-siblings. Each of these structured factors is a chain model that, for a given head word, assigns a score for the sequence of modifiers and (eventually) the grandparent. These factors have internal structure relying on horizontal and vertical Markov assumptions.

- Binary pairwise factors (also called Ising factors) for every possible pair of siblings. These are simple dense factors.
- A structured factor modeling the sequence of heads, with scores depending on the heads of consecutive words. This is also a chain model.

We will see in the sequel that a crucial operation to be performed at each factor is computing local MAP configurations. For all factors listed above, this can be done efficiently, as described in Martins et al. (2013).

### 1.3.2 Compressive Summarization

Our second NLP application is compressive summarization (Almeida and Martins, 2013). We are given a collection of documents about a given topic, and the goal is to generate a summary with less than $B$ words, where $B$ is a budget. Let $N$ be the total number of sentences in the collection, where the $n$th sentence has $L_n$ words. We constrain our summary's sentences to be *compressions* of the original sentences, obtained by deleting some words.[5]

We define one binary variable for every word of every sentence, yielding vectors $\boldsymbol{z}_n := (z_{n,\ell})_{\ell=1}^{L_n}$. A good summary is typically a trade-off between three important qualities: *conciseness*, *informativeness*, and *grammaticality*. This can be made formal by defining the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & g(\boldsymbol{z}) + \sum_{n=1}^{N} h_n(\boldsymbol{z}_n) \\ \text{w.r.t.} \quad & \boldsymbol{z}_n \in \{0,1\}^{L_n},\ n = 1, \ldots, N \\ \text{s.t.} \quad & \sum_{n=1}^{N} \sum_{\ell=1}^{L_n} z_{n,\ell} \leq B, \end{aligned} \tag{1.5}$$

where $g(\boldsymbol{z})$ is a *global informativeness score* based on how many "concepts"[6] are covered by the summary $\boldsymbol{z}$, and the $h_n(\boldsymbol{z}_n)$ are *local grammaticality*

5. This constraint is substantially weaker than that of *extractive summarizers*, which form a summary by picking full sentences. Our motivation is to make a better use of the budget, using shorter snippets to accommodate more information.
6. For simplicity, we assume concepts are *word bigrams* (as in Gillick et al. 2008), but other representations would be possible, such as phrases or predicate-argument structures.

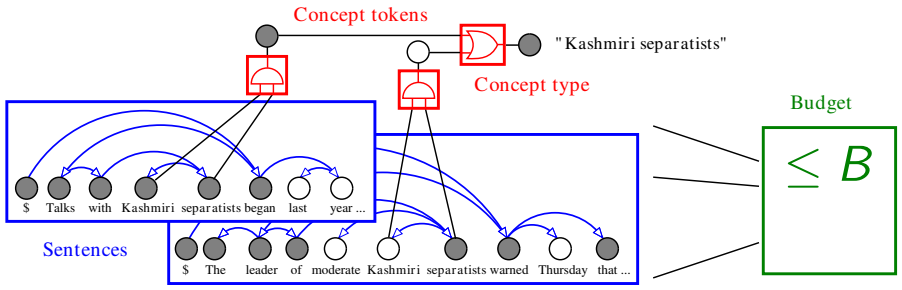


**Figure 1.4**: Components of our compressive summarizer. Factors depicted in blue belong to the compression model, and promote grammaticality. The logic factors in red form the coverage component. Finally, the budget factor, in green, is connected to the word nodes; it ensures that the summary fits the word limit.

*scores*, computed at sentence level.

Figure 1.4 shows a factor graph representation for this task, where circles represent variables, and rectangles represent factors. We introduce a binary variable for each word in the original sentences; to handle concepts, we distinguish between *concept types* (a list of word bigram types that appear frequently in the collection) and *concept tokens* (occurrences of those bigrams in the text), and we introduce also binary variables for those. Note that types and tokens are logically related: a concept type is present in the summary if at least one concept token is present. This is captured by the logic factors in Figure 1.4; this part of the graph takes care of the *informativeness score* $g(\boldsymbol{z})$. Then, for each sentence in the collection, we introduce a structured factor which computes a *grammaticality score* $h_n(\boldsymbol{z}_n)$, inspired by prior work in sentence compression (Knight and Marcu, 2000). The sentence is first parsed, yielding a dependency tree like in Figure 1.2; $h_n(\boldsymbol{z}_n)$ is then defined as a sum of penalties for the arcs deleted in the compression (see Almeida and Martins 2013). The internal structure of this factor is given by a tractable tree-shaped model. Finally, we have a *budget factor* imposing that at most $B$ words can be selected, which takes care of conciseness.

## 1.4 LP-MAP Decoding

We now describe a class of methods for approximate MAP decoding based on a *linear programming relaxation* of the problem (1.2), known in the literature

as Schlesinger's linear relaxation (Schlesinger, 1976; Werner, 2007).[7]

Let us start by representing the log-potential functions in (1.1) in vector notation, $\boldsymbol{\theta}_i := (\boldsymbol{\theta}_i(y_i))_{y_i \in \mathcal{Y}_i} \in \bar{\mathbb{R}}^{|\mathcal{Y}_i|}$ and $\boldsymbol{\theta}_\alpha := (\boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha))_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \in \bar{\mathbb{R}}^{|\mathcal{Y}_\alpha|}$. We introduce "local" probability distributions over the variables and factors, which we represent as vectors of the same size as $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_\alpha$:

$$\boldsymbol{p}_i \in \Delta^{|\mathcal{Y}_i|}, \ \forall i \in V \quad \text{and} \quad \boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|}, \ \forall \alpha \in F, \tag{1.6}$$

where $\Delta^K := \{\boldsymbol{u} \in \mathbb{R}^K \mid \boldsymbol{u} \geq \boldsymbol{0}, \ \boldsymbol{1} \cdot \boldsymbol{u} = 1\}$ denotes the $K$-dimensional probability simplex. We stack these distributions into vectors $\boldsymbol{p}$ and $\boldsymbol{q}$, with dimensions $P := \sum_{i \in V} |\mathcal{Y}_i|$ and $Q := \sum_{\alpha \in F} |\mathcal{Y}_\alpha|$, respectively. If these local probability distributions are "valid" marginal probabilities (*i.e.*, marginals realizable by some global probability distribution $\mathbb{P}(Y_1, \ldots, Y_M)$), then a necessary (but not sufficient) condition is that they are *locally consistent*. In other words, they must satisfy the following *calibration equations*:

$$\sum_{\boldsymbol{y}_\alpha \sim y_i} q_\alpha(\boldsymbol{y}_\alpha) = p_i(y_i), \qquad \forall y_i \in \mathcal{Y}_i, \ \forall (i, \alpha) \in E, \tag{1.7}$$

where the notation $\sim$ means that the summation is over all configurations $\boldsymbol{y}_\alpha$ whose $i$th element equals $y_i$. Equation (1.7) can be written in vector notation as $\mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha = \boldsymbol{p}_i, \ \forall (i, \alpha) \in E$, where we define *consistency matrices*

$$\mathbf{M}_{i\alpha}(y_i, \boldsymbol{y}_\alpha) = \begin{cases} 1, & \text{if } \boldsymbol{y}_\alpha \sim y_i \\ 0, & \text{otherwise.} \end{cases} \tag{1.8}$$

The set of locally consistent distributions forms the *local polytope*:

$$\mathcal{L}(G) = \left\{ (\boldsymbol{p}, \boldsymbol{q}) \in \mathbb{R}^{P+Q} \;\middle|\; \begin{array}{ll} \boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|}, & \forall \alpha \in F \\ \mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha = \boldsymbol{p}_i, & \forall (i, \alpha) \in E \end{array} \right\}. \tag{1.9}$$

We consider the following linear program (the *LP-MAP decoding problem*):

$$\begin{array}{lll} \textbf{LP-MAP:} & \text{maximize} & \displaystyle\sum_{\alpha \in F} \boldsymbol{\theta}_\alpha \cdot \boldsymbol{q}_\alpha + \sum_{i \in V} \boldsymbol{\theta}_i \cdot \boldsymbol{p}_i \\ & \text{with respect to} & (\boldsymbol{p}, \boldsymbol{q}) \in \mathcal{L}(G). \end{array} \tag{1.10}$$

If the solution $(\boldsymbol{p}^*, \boldsymbol{q}^*)$ of (1.10) happens to be integral, then each $\boldsymbol{p}_i^*$ and $\boldsymbol{q}_\alpha^*$ will be at corners of the simplex, *i.e.*, they will be indicator vectors of local configurations $y_i^*$ and $\boldsymbol{y}_\alpha^*$, in which case the output $(y_i^*)_{i \in V}$ is guaranteed to be a solution of the MAP decoding problem (1.2). Under

7. Chapter **??** of the current volume gives a detailed overview on what is known about this linear relaxation, including some hardness results.

certain conditions—for example, when the factor graph $G$ does not have cycles—(1.10) is guaranteed to have integral solutions. In general, however, the LP-MAP decoding problem (1.10) is a relaxation of (1.2). Geometrically, $\mathcal{L}(G)$ is an outer approximation of the *marginal polytope*, defined as the set of valid marginals (Wainwright and Jordan, 2008).

### 1.4.1 Dual Decomposition

While any off-the-shelf LP solver can be used for solving (1.10), specialized algorithms have been designed to exploit the graph structure, achieving superior performance on several benchmarks (Yanover et al., 2006). An example is the *projected subgradient dual decomposition* (PSDD) algorithm, proposed by Komodakis et al. (2007), which has old roots in optimization (Dantzig and Wolfe, 1960; Everett III, 1963; Guignard and Kim, 1987). As we will see in Section 1.5, there is a strong affinity between PSDD and the main focus of this chapter, AD$^3$.

Let us first express (1.10) as a consensus problem. For each edge $(i, \alpha) \in E$, we define a potential function $\boldsymbol{\theta}_{i\alpha} := (\boldsymbol{\theta}_{i\alpha}(y_i))_{y_i \in \mathcal{Y}_i}$ that satisfies $\sum_{\alpha \in F(i)} \boldsymbol{\theta}_{i\alpha} = \boldsymbol{\theta}_i$; a trivial choice is $\boldsymbol{\theta}_{i\alpha} = |F(i)|^{-1}\boldsymbol{\theta}_i$, which spreads the unary potentials evenly across the factors. Since we have a equality constraint $\boldsymbol{p}_i = \mathbf{M}_{i\alpha}\boldsymbol{q}_\alpha$, (1.10) is equivalent to the following *primal formulation*:

$$\begin{aligned} \textbf{LP-MAP-P:} \qquad \text{maximize} \quad & \sum_{\alpha \in F} \left( \boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top \boldsymbol{\theta}_{i\alpha} \right) \cdot \boldsymbol{q}_\alpha \\ \text{with respect to} \quad & \boldsymbol{p} \in \mathbb{R}^P, \quad \boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|}, \forall \alpha \in F, \\ \text{subject to} \quad & \mathbf{M}_{i\alpha}\boldsymbol{q}_\alpha = \boldsymbol{p}_i, \ \forall (i, \alpha) \in E. \end{aligned} \tag{1.11}$$

Note that, although the $\boldsymbol{p}$-variables do not appear in the objective of (1.11), they play a fundamental role through the constraints in the last line. Indeed, it is this set of constraints that complicate the optimization problem, which would otherwise be separable into independent subproblems, one per factor. Introducing Lagrange multipliers $\boldsymbol{\lambda}_{i\alpha} := (\lambda_{i\alpha}(y_i))_{y_i \in \mathcal{Y}_i}$ for these equality constraints, we arrive at the following *dual formulation*:

$$\begin{aligned} \textbf{LP-MAP-D:} \qquad \text{minimize} \quad & g(\boldsymbol{\lambda}) := \sum_{\alpha \in F} g_\alpha(\boldsymbol{\lambda}) \\ \text{with respect to} \quad & \boldsymbol{\lambda} \in \Lambda, \end{aligned} \tag{1.12}$$

where $\Lambda := \left\{ \boldsymbol{\lambda} \mid \sum_{\alpha \in F(i)} \boldsymbol{\lambda}_{i\alpha} = \mathbf{0}, \ \forall i \in V \right\}$ is a linear subspace, and each

$g_\alpha(\boldsymbol{\lambda})$ is the solution of a *local subproblem*:

$$g_\alpha(\boldsymbol{\lambda}) := \max_{\boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|}} \left( \boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top (\boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha}) \right) \cdot \boldsymbol{q}_\alpha \tag{1.13}$$

$$= \max_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \left( \boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha) + \sum_{i \in V(\alpha)} (\boldsymbol{\theta}_{i\alpha}(y_i) + \lambda_{i\alpha}(y_i)) \right). \tag{1.14}$$

The last equality is justified by the fact that maximizing a linear objective over the probability simplex gives the largest component of the score vector.

Note that the local subproblem (1.14) can be solved by a ComputeMAP procedure, which receives unary potentials $\xi_{i\alpha}(y_i) := \boldsymbol{\theta}_{i\alpha}(y_i) + \lambda_{i\alpha}(y_i)$ and factor potentials $\boldsymbol{\theta}_\alpha(\boldsymbol{y}_\alpha)$ (eventually structured) and returns the MAP $\widehat{\boldsymbol{y}}_\alpha$.

Problem (1.12) is often referred to as the *master* or *controller*, and each local subproblem (1.14) as a *slave* or *worker*. The master problem (1.12) can be solved with a *projected subgradient algorithm*.[8] By Danskin's rule (Bertsekas, 1999, p. 717), a subgradient of $g_\alpha$ is readily given by

$$\frac{\partial g_\alpha(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}_{i\alpha}} = \mathbf{M}_{i\alpha} \widehat{\boldsymbol{q}}_\alpha, \quad \forall (i, \alpha) \in E; \tag{1.15}$$

and the projection onto $\Lambda$ amounts to a centering operation. Putting these pieces together yields Algorithm 1.1. At each iteration, the algorithm broadcasts the current Lagrange multipliers to all the factors. Each factor adjusts its internal unary log-potentials (line 6) and invokes the ComputeMap procedure (line 7). The solutions achieved by each factor are then gathered and averaged (line 10), and the Lagrange multipliers are updated with step size $\eta_t$ (line 11).

The two following propositions, proved in Rush and Collins (2012), establish the convergence properties of Algorithm 1.1.

**Proposition 1.1** (Convergence rate)**.** *If the non-negative step size sequence $(\eta_t)_{t \in \mathbb{N}}$ is diminishing and nonsummable ($\lim \eta_t = 0$ and $\sum_{t=1}^{\infty} \eta_t = \infty$), then Algorithm 1.1 converges to the solution $\boldsymbol{\lambda}^*$ of LP-MAP-D* (1.12)*. Furthermore, after $T = O(1/\epsilon^2)$ iterations, we have $g(\boldsymbol{\lambda}^{(T)}) - g(\boldsymbol{\lambda}^*) \leq \epsilon$.*

**Proposition 1.2** (Certificate of optimality)**.** *If, at some iteration of Algorithm 1.1, all the local subproblems are in agreement (*i.e.*, if $\widehat{\boldsymbol{q}}_{i\alpha} = \boldsymbol{p}_i$ after line 10, for all $i \in V$), then: (i) $\boldsymbol{\lambda}$ is a solution of LP-MAP-D* (1.12)*; (ii) $\boldsymbol{p}$ is binary-valued and a solution of both LP-MAP-P and MAP.*

8. A slightly different formulation is presented by Sontag et al. (2011) which yields a subgradient algorithm with no projection.

**Algorithm 1.1** PSDD Algorithm (Komodakis et al., 2007)

1: **input:** graph $G$, parameters $\boldsymbol{\theta}$, maximum number of iterations $T$, stepsizes $(\eta_t)_{t=1}^T$
2: for each $(i,\alpha) \in E$, choose $\boldsymbol{\theta}_{i\alpha}$ such that $\sum_{\alpha \in F(i)} \boldsymbol{\theta}_{i\alpha} = \boldsymbol{\theta}_i$
3: initialize $\boldsymbol{\lambda} = \mathbf{0}$
4: **for** $t = 1$ **to** $T$ **do**
5: **for each** factor $\alpha \in F$ **do**
6: set unary log-potentials $\boldsymbol{\xi}_{i\alpha} := \boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha}$, for $i \in V(\alpha)$
7: set $\widehat{\boldsymbol{q}}_\alpha := \textsc{ComputeMap}(\boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top \boldsymbol{\xi}_{i\alpha})$
8: set $\widehat{\boldsymbol{q}}_{i\alpha} := \mathbf{M}_{i\alpha} \widehat{\boldsymbol{q}}_\alpha$, for $i \in V(\alpha)$
9: **end for**
10: compute average $\boldsymbol{p}_i := |F(i)|^{-1} \sum_{\alpha \in F(i)} \widehat{\boldsymbol{q}}_{i\alpha}$ for each $i \in V$
11: update $\boldsymbol{\lambda}_{i\alpha} := \boldsymbol{\lambda}_{i\alpha} - \eta_t \, (\widehat{\boldsymbol{q}}_{i\alpha} - \boldsymbol{p}_i)$ for each $(i,\alpha) \in E$
12: **end for**
13: **output:** dual variable $\boldsymbol{\lambda}$ and upper bound $g(\boldsymbol{\lambda})$

Propositions 1.1–1.2 imply that, if the LP-MAP relaxation is tight, then Algorithm 1.1 will eventually yield the exact MAP configuration along with a certificate of optimality. Unfortunately, in large graphs with many overlapping factors, it has been observed that convergence can be quite slow in practice (Martins et al., 2011b). This is not surprising, given that it attempts to reach a consensus among all overlapping components; the larger this number, the harder it is to achieve consensus. We describe in the next section another LP-MAP decoder (AD$^3$) with a faster convergence rate.

## 1.5 Alternating Directions Dual Decomposition (AD$^3$)

AD$^3$ obviates some of the weaknesses of PSDD by replacing the subgradient method with the *alternating directions method of multipliers* (ADMM).[9]

Before going into a formal derivation, let us go back to the PSDD algorithm to pinpoint the crux of their weaknesses. It resides on two aspects:

1. *The dual objective function $g(\boldsymbol{\lambda})$ is non-smooth.* This is circumvented by using "subgradients" rather than "gradients." Yet, non-smooth optimization lacks some of the good properties of its smooth counterpart. Ensuring convergence requires diminishing step sizes, leading to slow convergence rates such as the $O(1/\epsilon^2)$ iteration bound stated in Proposition 1.1.

2. *Consensus is promoted solely by the Lagrange multipliers* (see line 6 in Algorithm 1.1). These can be regarded as "price adjustments" that are made at each iteration and lead to a reallocation of resources. However,

9. See Boyd et al. (2011, Section 3.1) for a historical perspective on ADMM as a faster alternative to subgradient methods.

no "memory" exists about past allocations or adjustments, so the workers never know how far they are from consensus. One may suspect that a smarter use of these quantities may accelerate convergence.

The first of these aspects has been addressed by the accelerated dual decomposition method of Jojic et al. (2010), which improves the iteration bound to $O(1/\epsilon)$; we discuss that work further in Section 1.8. We will see that AD$^3$ also yields a $O(1/\epsilon)$ iteration bound with some additional advantages. The second aspect is addressed by AD$^3$ by broadcasting *the current global solution* in addition to the Lagrange multipliers, allowing the workers to regularize their subproblems toward that solution.

### 1.5.1 Augmented Lagrangians, ADMM, and AD$^3$

Augmented Lagrangian methods have a rich and long-standing history in optimization (Hestenes, 1969; Powell, 1969; Bertsekas, 1999, Section 4.2). Given an optimization problem with equality constraints (such as the one in (1.11)), the *augmented Lagrangian function* is the sum of the Lagrangian with a quadratic constraint violation penalty. For (1.11), it is

$$\begin{aligned} L_\eta(\boldsymbol{q}, \boldsymbol{p}, \boldsymbol{\lambda}) \;\; = \;\; & \sum_{\alpha \in F} \left( \boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top (\boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha}) \right) \cdot \boldsymbol{q}_\alpha - \sum_{(i,\alpha) \in E} \boldsymbol{\lambda}_{i\alpha} \cdot \boldsymbol{p}_i \\ & - \frac{\eta}{2} \sum_{(i,\alpha) \in E} \|\mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha - \boldsymbol{p}_i\|^2, \end{aligned} \tag{1.16}$$

where the scalar $\eta > 0$ controls the weight of the penalty.

The most famous augmented Lagrangian method is the *method of multipliers*, which alternates between maximizing the augmented Lagrangian function with respect to the primal variables (in our problem, a joint maximization of $L_\eta(\boldsymbol{q}, \boldsymbol{p}, \boldsymbol{\lambda})$ w.r.t. $\boldsymbol{q}$ and $\boldsymbol{p}$), followed by an update of the Lagrange multipliers. Unfortunately, the quadratic term in (1.16) couples together the variables $\boldsymbol{q}$ and $\boldsymbol{p}$, making their joint maximization unappealing. ADMM (Glowinski and Marroco, 1975; Gabay and Mercier, 1976) avoids this shortcoming, by replacing this joint maximization by a single block Gauss-Seidel-type step. This yields the following updates (where we have

defined the product of simplices $\Delta_F^{\times} := \prod_{\alpha \in F} \Delta^{|\mathcal{Y}_\alpha|}$):

$$\textbf{Broadcast:} \quad \boldsymbol{q}^{(t+1)} := \operatorname*{argmax}_{\boldsymbol{q} \in \Delta_F^{\times}} L_\eta(\boldsymbol{q}, \boldsymbol{p}^{(t)}, \boldsymbol{\lambda}^{(t)}), \tag{1.17}$$

$$\textbf{Gather:} \quad \boldsymbol{p}^{(t+1)} := \operatorname*{argmax}_{\boldsymbol{p} \in \mathbb{R}^P} L_\eta(\boldsymbol{q}^{(t+1)}, \boldsymbol{p}, \boldsymbol{\lambda}^{(t)}), \tag{1.18}$$

$$\textbf{Multiplier update:} \quad \boldsymbol{\lambda}_{i\alpha}^{(t+1)} := \boldsymbol{\lambda}_{i\alpha}^{(t)} - \eta \left( \mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha^{(t)} - \boldsymbol{p}_i^{(t)} \right), \forall (i, \alpha) \in E. \tag{1.19}$$

We next analyze the broadcast and gather steps, and prove a proposition about the multiplier update.

**Broadcast step.** The maximization (1.17) can be carried out in parallel at the factors, as in PSDD. The only difference is that, instead of a local MAP computation, each worker needs to solve a *quadratic program* of the form:

$$\max_{\boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|}} \left( \boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top (\boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha}) \right) \cdot \boldsymbol{q}_\alpha - \frac{\eta}{2} \sum_{i \in V(\alpha)} \|\mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha - \boldsymbol{p}_i\|^2. \tag{1.20}$$

The subproblem (1.20) differs from the linear subproblem (1.13)–(1.14) in the PSDD algorithm by including an Euclidean penalty term, which penalizes deviations from the global consensus. In Section 1.6, we will give efficient procedures to solve these local subproblems.

**Gather step.** The solution of (1.18) has a closed form. (1.18) is separable into independent optimizations for each $i \in V$; defining $\boldsymbol{q}_{i\alpha} := \mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha$,

$$\begin{aligned} \boldsymbol{p}_i^{(t+1)} &:= \arg \min_{\boldsymbol{p}_i \in \mathbb{R}^{|\mathcal{Y}_i|}} \sum_{\alpha \in F(i)} \left( \boldsymbol{\lambda}_{i\alpha} \cdot \boldsymbol{p}_i + \frac{\eta}{2} \|\boldsymbol{q}_{i\alpha} - \boldsymbol{p}_i\|^2 \right) \\ &= |F(i)|^{-1} \sum_{\alpha \in F(i)} \left( \boldsymbol{q}_{i\alpha} - \eta^{-1} \boldsymbol{\lambda}_{i\alpha} \right) \\ &= |F(i)|^{-1} \sum_{\alpha \in F(i)} \boldsymbol{q}_{i\alpha}. \end{aligned} \tag{1.21}$$

The equality in the last line is due to the following proposition:

**Proposition 1.3.** *The sequence $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}, \ldots$ produced by (1.17)–(1.19) is dual feasible, i.e., we have $\boldsymbol{\lambda}^{(t)} \in \Lambda$ for every $t$, with $\Lambda$ as in (1.12).*

*Proof.* We have $\sum_{\alpha \in F(i)} \boldsymbol{\lambda}_{i\alpha}^{(t+1)} = \sum_{\alpha \in F(i)} \boldsymbol{\lambda}_{i\alpha}^{(t)} - \eta (\sum_{\alpha \in F(i)} \boldsymbol{q}_{i\alpha}^{(t+1)} - |F(i)| \boldsymbol{p}_i^{(t+1)}) = \sum_{\alpha \in F(i)} \boldsymbol{\lambda}_{i\alpha}^{(t)} - \eta (\sum_{\alpha \in F(i)} \boldsymbol{q}_{i\alpha}^{(t+1)} - \sum_{\alpha \in F(i)} (\boldsymbol{q}_{i\alpha}^{(t+1)} - \eta^{-1} \boldsymbol{\lambda}_{i\alpha}^{(t)})) = \mathbf{0}$. □

Assembling all these pieces together leads to AD$^3$ (Algorithm 1.2). Notice

**Algorithm 1.2** Alternating Directions Dual Decomposition (AD$^3$)

1: **input:** graph $G$, parameters $\boldsymbol{\theta}$, penalty constant $\eta$
2: initialize $\boldsymbol{p}$ uniformly (*i.e.*, $p_i(y_i) = 1/|\mathcal{Y}_i|$, $\forall i \in V, y_i \in \mathcal{Y}_i$)
3: initialize $\boldsymbol{\lambda} = \mathbf{0}$
4: **repeat**
5: **for each** factor $\alpha \in F$ **do**
6: set unary log-potentials $\boldsymbol{\xi}_{i\alpha} := \boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha}$, for $i \in V(\alpha)$
7: set $\widehat{\boldsymbol{q}}_\alpha := \textsc{SolveQP}\left(\boldsymbol{\theta}_\alpha + \sum_{i \in V(\alpha)} \mathbf{M}_{i\alpha}^\top \boldsymbol{\xi}_{i\alpha},\ (\boldsymbol{p}_i)_{i \in V(\alpha)}\right)$, the problem (1.20)
8: set $\widehat{\boldsymbol{q}}_{i\alpha} := \mathbf{M}_{i\alpha} \widehat{\boldsymbol{q}}_\alpha$, for $i \in V(\alpha)$
9: **end for**
10: compute average $\boldsymbol{p}_i := |F(i)|^{-1} \sum_{\alpha \in F(i)} \widehat{\boldsymbol{q}}_{i\alpha}$ for each $i \in V$
11: update $\boldsymbol{\lambda}_{i\alpha} := \boldsymbol{\lambda}_{i\alpha} - \eta\,(\widehat{\boldsymbol{q}}_{i\alpha} - \boldsymbol{p}_i)$ for each $(i, \alpha) \in E$
12: **until** convergence
13: **output:** primal variables $\boldsymbol{p}$ and $\boldsymbol{q}$, dual variable $\boldsymbol{\lambda}$, upper bound $g(\boldsymbol{\lambda})$

that AD$^3$ retains the modular structure of PSDD (Algorithm 1.1). The key difference is that AD$^3$ also broadcasts the current global solution to the workers, allowing them to regularize their subproblems toward that solution, thus speeding up the consensus. This is embodied in the procedure SolveQP (line 7), which replaces ComputeMAP of Algorithm 1.1.

### 1.5.2 Convergence Analysis

Convergence of AD$^3$ follows directly from the general convergence properties of ADMM. Unlike PSDD (Algorithm 1.1), convergence is ensured with a fixed step size $\eta$, therefore no annealing is required.

**Proposition 1.4** (Convergence)**.** *Let $(\boldsymbol{q}^{(t)}, \boldsymbol{p}^{(t)}, \boldsymbol{\lambda}^{(t)})_t$ be the sequence of iterates produced by Algorithm 1.2. Then the following holds:*

1. *The primal sequence $(\boldsymbol{p}^{(t)}, \boldsymbol{q}^{(t)})_{t \in \mathbb{N}}$ converges to a solution of the LP-MAP-P* (1.11)*; as a consequence, primal feasibility of LP-MAP-P is achieved in the limit,* i.e.*, $\|\mathbf{M}_{i\alpha} \boldsymbol{q}_\alpha^{(t)} - \boldsymbol{p}_i^{(t)}\| \to \mathbf{0}, \quad \forall (i, \alpha) \in E$;*
2. *The dual sequence $(\boldsymbol{\lambda}^{(t)})_{t \in \mathbb{N}}$ converges to a solution of LP-MAP-D* (1.12)*; moreover, this sequence is always dual feasible,* i.e.*, it is contained in $\Lambda$.*

*Proof.* 1 and the first part of 2 are general properties of ADMM (Glowinski and Le Tallec, 1989, Theorem 4.2). The second part of statement 2 stems from Proposition 1.3. □

The next proposition states the $O(1/\epsilon)$ iteration bound of AD$^3$, which is better than the $O(1/\epsilon^2)$ bound of PSDD.

**Proposition 1.5** (Convergence rate)**.** *Let $\boldsymbol{\lambda}^*$ be a solution of LP-MAP-D* (1.12)*, $\bar{\boldsymbol{\lambda}}_T := \frac{1}{T} \sum_{t=1}^T \boldsymbol{\lambda}^{(t)}$ be the "averaged" Lagrange multipliers after $T$*

*iterations of AD$^3$, and $g(\bar{\boldsymbol{\lambda}}_T)$ the corresponding estimate of the dual objective (an upper bound). Then, $g(\bar{\boldsymbol{\lambda}}_T) - g(\boldsymbol{\lambda}^*) \leq \epsilon$ after $T \leq C/\epsilon$ iterations, where*

$$C \;\leq\; \frac{5\eta}{2}|E| + \frac{5}{2\eta}\|\boldsymbol{\lambda}^*\|^2. \tag{1.22}$$

*Proof.* A detailed proof is presented in Martins et al. (2012), adapting a recent result of Wang and Banerjee (2012) concerning convergence of ADMM in a variational inequality setting. □

As expected, the bound (1.22) increases with the number of overlapping variables, quantified by the number of edges $|E|$, and the magnitude of the optimal dual vector $\boldsymbol{\lambda}^*$. Note that if there is a good estimate of $\|\boldsymbol{\lambda}^*\|$, then (1.22) can be used to choose a step size $\eta$ that minimizes the bound.[10] The optimal stepsize is $\eta = \|\boldsymbol{\lambda}^*\| \times |E|^{-1/2}$, which would lead to $T \leq 5\epsilon^{-1}|E|^{1/2}$.

### 1.5.3 Other Details and Extensions

**Primal and dual residuals.** Since the AD$^3$ iterates are dual feasible, it is also possible to check the conditions in Proposition 1.2 to obtain optimality certificates, as in PSDD. Moreover, even when the LP-MAP relaxation is not tight, AD$^3$ can provide stopping conditions by keeping track of primal and dual residuals (an important advantage over PSDD), as described in Boyd et al. (2011). The *primal residual* $r_P^{(t)}$ is the amount by which the agreement constraints are violated,

$$r_P^{(t)} = \frac{\sum_{(i,\alpha)\in E} \|\mathbf{M}_{i\alpha}\boldsymbol{q}_\alpha^{(t)} - \boldsymbol{p}_i^{(t)}\|^2}{\sum_{(i,\alpha)\in E} |\mathcal{Y}_i|} \in [0,1]. \tag{1.23}$$

The *dual residual* $r_D^{(t)}$ is the amount by which a dual optimality condition is violated,

$$r_D^{(t)} = \frac{\sum_{(i,\alpha)\in E} \|\boldsymbol{p}_i^{(t)} - \boldsymbol{p}_i^{(t-1)}\|^2}{\sum_{(i,\alpha)\in E} |\mathcal{Y}_i|} \in [0,1]. \tag{1.24}$$

We adopt as stopping criterion that these two residuals fall below $10^{-6}$.

**Approximate solutions of the local subproblems.** The next proposition states that convergence may still hold, even if the local subproblems are only solved approximately. The importance of this result will be clear in

10. Although Proposition 1.4 guarantees convergence for any choice of $\eta$, this parameter may strongly impact the behavior of the algorithm. In our experiments, we dynamically adjust $\eta$ in earlier iterations using the heuristic described in Boyd et al. (2011, Section 3.4.1).

Section 1.6.1, where we describe a general iterative algorithm for solving the local quadratic subproblems. Essentially, Proposition 1.6 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 AD$^3$ iterations.

**Proposition 1.6** (Eckstein and Bertsekas, 1992)**.** *For every iteration $t$, let $\widehat{\boldsymbol{q}}^{(t)}$ contain the exact solutions of* (1.20)*, and $\tilde{\boldsymbol{q}}^{(t)}$ those produced by an approximate algorithm. Then Proposition 1.4 still holds, provided that the sequence of errors is summable,* i.e.*, $\sum_{t=1}^{\infty} \|\widehat{\boldsymbol{q}}^{(t)} - \tilde{\boldsymbol{q}}^{(t)}\| < \infty$.*

**Caching the subproblems.** In practice, considerable speed-ups can be achieved by *caching* the subproblems, following a strategy proposed for PSDD by Koo et al. (2010). After a few iterations, many variables $\boldsymbol{p}_i$ reach a consensus (*i.e.*, $\boldsymbol{p}_i^{(t)} = \boldsymbol{q}_{i\alpha}^{(t+1)}, \forall \alpha \in F(i)$) and enter an idle state: they are left unchanged by the $\boldsymbol{p}$-update (line 10), and so do the Lagrange variables $\boldsymbol{\lambda}_{i\alpha}^{(t+1)}$ (line 11). If at iteration $t$ all variables in a subproblem at factor $\alpha$ are idle, then $\boldsymbol{q}_\alpha^{(t+1)} = \boldsymbol{q}_\alpha^{(t)}$, hence the corresponding subproblem does not need to be solved. Typically, many variables and subproblems enter this idle state after the first few rounds. We will show the practical benefits of caching in the experimental section (Section 1.7.1, Figure 1.5).

**Exact decoding with branch-and-bound.** Recall that AD$^3$, as just described, solves the LP-MAP relaxation, which is not only always tight. Das et al. (2012) propose to wrap AD$^3$ into a *branch-and-bound* search procedure to find the *exact MAP*, using the property that the optimal objective value of LP-MAP is an *upper bound* of MAP, and that AD$^3$, along its execution, keeps track of a sequence of feasible dual points (as guaranteed by Proposition 1.4, item 2), building tighter and tighter upper bounds. Although this procedure has worst-case exponential runtime, it was found empirically quite effective in problems for which the relaxations are near-exact.

## 1.6 Local Subproblems in AD$^3$

We now turn our attention to the AD$^3$ local subproblems (1.20). In many important cases, these QPs can be solved efficiently. Some examples are:

- Ising factors (Martins et al., 2011a), which can be solved in constant time.
- The logic constraint factors in Section 1.2.3, useful as building blocks for expressing soft and hard constraints in first-order logic (Martins et al.,

2011a); these can be solved in linear time with respect to the number of variables linked to the factor.[11]

- Budget and knapsack factors, which are important for dealing with cardinality-based potentials, or to promote diversity (Tarlow et al., 2010; Almeida and Martins, 2013). Runtime is again linear in the size of the factor.
- Parity-check factors (Barman et al., 2011), used in low-density parity check decoding. Runtime is $O(L \log L)$, where $L$ is the number of parity check bits.

In fact, for hard-constraint factors, the AD$^3$ subproblems have a nice geometric interpretation as an Euclidean projection onto the marginal polytope of the factor (as illustrated in Figure 1.1). In this section, we complement these results with a general *active-set procedure* for solving the AD$^3$ subproblems for *arbitrary* factors, the only requirement being a black-box MAP solver—the same as the PSDD algorithm. This makes AD$^3$ applicable to a wide range of problems. In particular, it makes possible to handle *structured factors* (Section 1.2.3), by invoking specialized MAP decoders.[12]

### 1.6.1 Active Set Method

Our active set method is based on Nocedal and Wright (1999, Section 16.4); it is an iterative algorithm that addresses the AD$^3$ subproblems (1.20) by solving a sequence of linear problems. By subtracting a constant, re-scaling, and flipping signs, (1.20) can be written more compactly as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{M}\boldsymbol{q}_\alpha - \boldsymbol{a}\|^2 - \boldsymbol{b} \cdot \boldsymbol{q}_\alpha \\ \text{with respect to} \quad & \boldsymbol{q}_\alpha \in \mathbb{R}^{|\mathcal{Y}_\alpha|} \\ \text{subject to} \quad & \mathbf{1} \cdot \boldsymbol{q}_\alpha = 1, \quad \boldsymbol{q}_\alpha \geq \mathbf{0}, \end{aligned} \tag{1.25}$$

where $\boldsymbol{a} := (\boldsymbol{a}_i)_{i \in V(\alpha)}$, with $\boldsymbol{a}_i := \boldsymbol{p}_i + \eta^{-1}(\boldsymbol{\theta}_{i\alpha} + \boldsymbol{\lambda}_{i\alpha})$; $\boldsymbol{b} := \eta^{-1}\boldsymbol{\theta}_\alpha$; and $\mathbf{M} := (\mathbf{M}_{i\alpha})_{i \in V(\alpha)}$ denotes a matrix with $\sum_i |\mathcal{Y}_i|$ rows and $|\mathcal{Y}_\alpha|$ columns.

The next crucial proposition (proved in Martins et al. 2012) states that problem (1.25) always admits a *sparse solution*.

**Proposition 1.7.** *Problem* (1.25) *admits a solution* $\boldsymbol{q}_\alpha^* \in \mathbb{R}^{|\mathcal{Y}_\alpha|}$ *with at most*

11. This is also the asymptotic complexity for computing the MAP for those factors. While the complexities reported by Martins et al. (2011a) have an extra logarithmic term, in all cases the runtime can drop to linear by using linear-time selection algorithms. We refer the interested reader to Almeida and Martins (2013) for details.

12. Past work (Martins et al., 2011a) suggested another strategy (*graph binarization*) to deal with dense and structured factors. However, this is outperformed, in practice, by the active set method we next present (see Martins 2012, Figure 6.3, for a comparison).

$\sum_{i \in V(\alpha)} |\mathcal{Y}_i| - V(\alpha) + 1$ *non-zero components.*

The fact that the solution lies in a low dimensional subspace makes active set methods appealing, since they only keep track of an *active set* of variables, that is, the non-zero components of $\boldsymbol{q}_\alpha$. Proposition 1.7 tells us that such an algorithm only needs 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. Our active set method seeks to identify the low-dimensional support of the solution $\boldsymbol{q}_\alpha^*$, by generating sparse iterates $\boldsymbol{q}_\alpha^{(1)}, \boldsymbol{q}_\alpha^{(2)}, \ldots$, while it maintains a working set $W \subseteq \mathcal{Y}_\alpha$ with the inequality constraints of (1.25) that are *inactive* along the way (*i.e.*, those $\boldsymbol{y}_\alpha$ for which $q_\alpha(\boldsymbol{y}_\alpha) > 0$ holds strictly). Each iteration adds or removes elements from the working set while it monotonically decreases the objective of Equation (1.25).[13]

**Lagrangian and KKT conditions.** Let $\tau$ and $\boldsymbol{\mu}$ be dual variables associated with the equality and inequality constraints of (1.25), respectively. The Lagrangian function is

$$L(\boldsymbol{q}_\alpha, \tau, \boldsymbol{\mu}) = \frac{1}{2}\|\mathbf{M}\boldsymbol{q}_\alpha - \boldsymbol{a}\|^2 - \boldsymbol{b} \cdot \boldsymbol{q}_\alpha - \tau(1 - \mathbf{1} \cdot \boldsymbol{q}_\alpha) - \boldsymbol{\mu} \cdot \boldsymbol{q}_\alpha. \tag{1.26}$$

This gives rise to the following Karush-Kuhn-Tucker (KKT) conditions:

$$\mathbf{M}^\top(\boldsymbol{a} - \mathbf{M}\boldsymbol{q}_\alpha) + \boldsymbol{b} = \tau\mathbf{1} - \boldsymbol{\mu} \quad (\nabla_{\boldsymbol{q}_\alpha} L = \mathbf{0}) \tag{1.27}$$

$$\mathbf{1} \cdot \boldsymbol{q}_\alpha = 1,\ \boldsymbol{q}_\alpha \geq \mathbf{0},\ \boldsymbol{\mu} \geq \mathbf{0} \quad \text{(Primal/dual feasibility)} \tag{1.28}$$

$$\boldsymbol{\mu} \cdot \boldsymbol{q}_\alpha = \mathbf{0} \quad \text{(Complementary slackness).} \tag{1.29}$$

The method works at follows. At each iteration $s$, it first checks if the current iterate $\boldsymbol{q}_\alpha^{(s)}$ is a *subspace minimizer*, *i.e.*, if it optimizes the objective of (1.25) in the sparse subspace defined by the working set $W$, $\{\boldsymbol{q}_\alpha \in \Delta^{|\mathcal{Y}_\alpha|} \mid q_\alpha(\boldsymbol{y}_\alpha) = 0, \forall \boldsymbol{y}_\alpha \notin W\}$. This check can be made by first solving a relaxation where the inequality constraints are ignored. Since in this subspace the components of $\boldsymbol{q}_\alpha$ not in $W$ will be zeros, one can simply delete those entries from $\boldsymbol{q}_\alpha$ and $\boldsymbol{b}$ and the corresponding columns in $\mathbf{M}$; we use a horizontal bar to denote

13. Our description differs from Nocedal and Wright (1999) in which their working set contains *active* constraints rather than the inactive ones. In our case, most constraints are active for the optimal $\boldsymbol{q}_\alpha^*$, therefore it is appealing to store the ones that are not.

these truncated $\mathbb{R}^{|W|}$-vectors. The problem can be written as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\bar{\mathbf{M}}\bar{\boldsymbol{q}}_\alpha - \boldsymbol{a}\|^2 - \bar{\boldsymbol{b}} \cdot \bar{\boldsymbol{q}}_\alpha \\ \text{with respect to} \quad & \bar{\boldsymbol{q}}_\alpha \in \mathbb{R}^{|W|} \\ \text{subject to} \quad & \mathbf{1} \cdot \bar{\boldsymbol{q}}_\alpha = 1. \end{aligned} \tag{1.30}$$

The solution of this equality-constrained QP can be found by solving a system of KKT equations:[14]

$$\begin{bmatrix} \bar{\mathbf{M}}^\top \bar{\mathbf{M}} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{q}}_\alpha \\ \tau \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{M}}^\top \boldsymbol{a} + \bar{\boldsymbol{b}} \\ 1 \end{bmatrix}. \tag{1.31}$$

The solution of (1.31) will give $(\widehat{\boldsymbol{q}}_\alpha, \widehat{\tau})$, where $\widehat{\boldsymbol{q}}_\alpha \in \mathbb{R}^{|\mathcal{Y}_\alpha|}$ is padded back with zeros. If it happens that $\widehat{\boldsymbol{q}}_\alpha = \boldsymbol{q}_\alpha^{(s)}$, then this means that the current iterate $\boldsymbol{q}_\alpha^{(s)}$ is a subspace minimizer; otherwise a new iterate $\boldsymbol{q}_\alpha^{(s+1)}$ will be computed. We next discuss these two events.

**Case 1: $\boldsymbol{q}_\alpha^{(s)}$ is a subspace minimizer.** If this happens, then it may be the case that $\boldsymbol{q}_\alpha^{(s)}$ is the optimal solution of (1.25). By looking at the KKT conditions (1.27)–(1.29), we have that this will happen iff $\mathbf{M}^\top(\boldsymbol{a} - \mathbf{M}\boldsymbol{q}_\alpha^{(s)}) + \boldsymbol{b} \leq \tau^{(s)}\mathbf{1}$. Define $\boldsymbol{w} := \boldsymbol{a} - \mathbf{M}\boldsymbol{q}_\alpha$. The condition above is equivalent to

$$\max_{\boldsymbol{y}_\alpha \in \mathcal{Y}_\alpha} \left( b(\boldsymbol{y}_\alpha) + \sum_{i \in V(\alpha)} w_i(y_i) \right) \leq \tau^{(s)}. \tag{1.32}$$

It turns out that this maximization is precisely a *local MAP decoding problem*, given a vector of unary potentials $\boldsymbol{w}$ and factor potentials $\boldsymbol{b}$. Thus, the maximizer $\widehat{\boldsymbol{y}}_\alpha$ can be computed via the ComputeMAP procedure, which we assume available. If $b(\widehat{\boldsymbol{y}}_\alpha) + \sum_{i \in V(\alpha)} w_i(\widehat{y}_i) \leq \tau^{(s)}$, then the KKT conditions are satisfied and we are done. Otherwise, $\widehat{\boldsymbol{y}}_\alpha$ indicates the most violated condition; we will add it to the active set $W$, and proceed.

**Case 2: $\boldsymbol{q}_\alpha^{(s)}$ is not a subspace minimizer.** If this happens, then we compute a new iterate $\boldsymbol{q}_\alpha^{(s+1)}$ by keeping searching in the same subspace. We have already solved a relaxation in (1.30). If we have $\widehat{q}_\alpha(\boldsymbol{y}_\alpha) \geq 0$ for all $\boldsymbol{y}_\alpha \in W$, then the relaxation is tight, so we just set $\boldsymbol{q}_\alpha^{(s+1)} := \widehat{\boldsymbol{q}}_\alpha$ and proceed. Otherwise, we move as much as possible in the direction of $\widehat{\boldsymbol{q}}_\alpha$ while keeping

14. Note that this is a low-dimensional problem, since we are working in a sparse working set. By caching the inverse of the matrix in the left-hand side, this system can be solved in time $O(|W|^2)$ at each iteration. See Martins et al. (2012) for further details.

feasibility, by defining $\boldsymbol{q}_\alpha^{(s+1)} := (1-\beta)\boldsymbol{q}_\alpha^{(s)} + \beta\widehat{\boldsymbol{q}}_\alpha$—as described in Nocedal and Wright (1999), the value of $\beta \in [0,1]$ can be computed in closed form:

$$\beta = \min\left\{1, \min_{\boldsymbol{y}_\alpha \in W \,:\, q_\alpha^{(s)}(\boldsymbol{y}_\alpha) > \widehat{q}_\alpha(\boldsymbol{y}_\alpha)} \frac{q_\alpha^{(s)}(\boldsymbol{y}_\alpha)}{q_\alpha^{(s)}(\boldsymbol{y}_\alpha) - \widehat{q}_\alpha(\boldsymbol{y}_\alpha)}\right\}. \tag{1.33}$$

If $\beta < 1$, this update will have the effect of making one of the constraints active, by zeroing out $q_\alpha^{(s+1)}(\boldsymbol{y}_\alpha)$ for the minimizing $\boldsymbol{y}_\alpha$ above. This so-called "blocking constraint" is thus be removed from the working set $W$.

Algorithm 1.3 describes the complete procedure. The active set $W$ is initialized arbitrarily: a strategy that works well in practice is, in the first AD$^3$ iteration, initialize $W := \{\widehat{\boldsymbol{y}}_\alpha\}$, where $\widehat{\boldsymbol{y}}_\alpha$ is the MAP configuration given log-potentials $\boldsymbol{a}$ and $\boldsymbol{b}$; and in subsequent AD$^3$ iterations, warm-start $W$ with the support of the solution obtained in the previous iteration.

Each iteration of Algorithm 1.3 improves the objective of (1.25), and the algorithm is guaranteed to stop after a finite number of steps (Nocedal and Wright, 1999, Theorem 16.5). In practice, since it is run as a subroutine of AD$^3$, Algorithm 1.3 does not need to be run to optimality, which is convenient in early iterations of AD$^3$ (this is supported by Proposition 1.6). The ability to warm-start with the solution from 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 for the correct active set to be identified. We will see some empirical evidence in Section 1.7.1.

## 1.7 Experiments

We next provide empirical results using AD$^3$ for the NLP applications described in Section 1.3. We refer the interested reader to Martins et al. (2012) for a broader empirical evaluation in other tasks and datasets.

### 1.7.1 Turbo Parsing

We applied AD$^3$ to multilingual dependency parsing, using third-order models and the factor graph representation described in Section 1.3.1.[15]

For English, we used two datasets: English-I was derived from the Penn Treebank (Marcus et al., 1993), converted to a dependency treebank by applying the head rules of Yamada and Matsumoto (2003). For this dataset,

15. The resulting parser is released as free software under the name `TurboParser 2.1`, and available for download at `http://www.ark.cs.cmu.edu/TurboParser`.

**Algorithm 1.3** Active Set Algorithm for Solving a General AD$^3$ Subproblem

**input:** Parameters $\boldsymbol{a}, \boldsymbol{b}, \mathbf{M}$, starting point $\boldsymbol{q}_\alpha^{(0)}$
initialize $W^{(0)}$ as the support of $\boldsymbol{q}_\alpha^{(0)}$
**for** $s = 0, 1, 2, \ldots$ **do**
  solve the KKT system and obtain $\widehat{\boldsymbol{q}}_\alpha$ and $\widehat{\tau}$ (Eq. 1.31)
  **if** $\widehat{\boldsymbol{q}}_\alpha = \boldsymbol{q}_\alpha^{(s)}$ **then**
    compute $\boldsymbol{w} := \boldsymbol{a} - \mathbf{M}\widehat{\boldsymbol{q}}_\alpha$
    obtain the tighter constraint $\widehat{\boldsymbol{y}}_\alpha$ via $\boldsymbol{e}_{\widehat{\boldsymbol{y}}_\alpha} = \textsc{ComputeMAP}(\boldsymbol{b} + \mathbf{M}^\top \boldsymbol{w})$
    **if** $b(\widehat{\boldsymbol{y}}_\alpha) + \sum_{i \in V(\alpha)} w_i(\widehat{y}_i) \leq \widehat{\tau}$ **then**
      return solution $\widehat{\boldsymbol{q}}_\alpha$
    **else**
      add the most violated constraint to the active set: $W^{(s+1)} := W^{(s)} \cup \{\widehat{\boldsymbol{y}}_\alpha\}$
    **end if**
  **else**
    compute the interpolation constant $\beta$ as in (1.33)
    set $\boldsymbol{q}_\alpha^{(s+1)} := (1 - \beta)\boldsymbol{q}_\alpha^{(s)} + \beta\widehat{\boldsymbol{q}}_\alpha$
    **if** if $\beta < 1$ **then**
      pick the blocking constraint $\widehat{\boldsymbol{y}}_\alpha$ in (1.33)
      remove $\widehat{\boldsymbol{y}}_\alpha$ from the active set: $W^{(s+1)} := W^{(s)} \setminus \{\widehat{\boldsymbol{y}}_\alpha\}$
    **end if**
  **end if**
**end for**
**output:** $\widehat{\boldsymbol{q}}_\alpha$

the resulting dependencies are all *projective*—i.e., all the dependency arcs are nested (see footnote 3). English-II, on the other hand, contains non-projective dependencies and is the dataset used in the CoNLL 2008 shared task (Surdeanu et al., 2008). Non-projective dependencies are particularly appropriate for languages with a more flexible word order, such as Czech, Dutch, and German; we report here results for those languages using the CoNLL-X datasets (Buchholz and Marsi, 2006) with the standard splits.[16]

For each language, we trained linear models using various features decomposing over the parts in Figure 1.3, depending on words, part-of-speech tags, and arc direction and length. To ensure valid parse trees at test time, we rounded fractional solutions as in Martins et al. (2009)—yet, solutions were integral $\approx 95\%$ of the time. Figure 1.5 shows average runtimes of AD$^3$ and PSDD on this task, as a function of the sentence length. Clearly, AD$^3$ scales better, even though it needs to solve more involved local subproblems. It turns out that the caching and warm-starting procedures are crucial in achieving these runtimes, as illustrated in the right part of the figure.

Table 1.2 shows results for several languages, comparing accuracies and speeds with those of state-of-the-art parsers, including projective parsers

16. Experiments with more languages and baselines are reported in Martins et al. (2013).

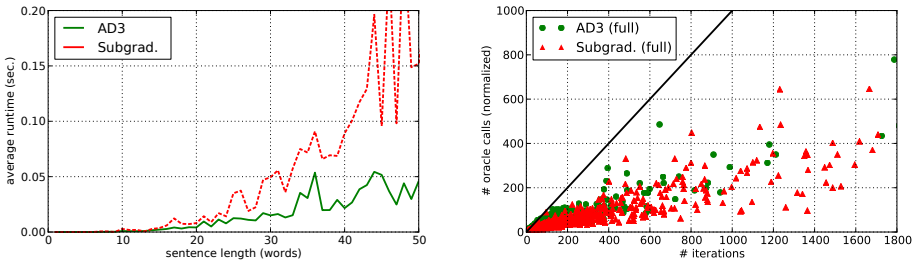


**Figure 1.5**: Left: averaged runtimes in the English-I dev-set as a function of the sentence length. For PSDD, we chose for each sentence the most favorable stepsize in $\{0.001, 0.01, 0.1, 1\}$. Right: number of calls to ComputeMAP for AD$^3$ and PSDD, as a function of the number of iterations (the number of calls is normalized by dividing by the number of factors). In PSDD, this number would equal the number of iterations if there was no caching (black line); in AD$^3$ it would be even higher, since ComputeMAP is called multiple times in the active set algorithm. Yet, both algorithms make significantly fewer calls. Remarkably, after just a few iterations, the number of calls made by AD$^3$ and PSDD is comparable, as the number of active set iterations is quickly amortized during the execution of AD$^3$.

such as Rush and Petrov (2012) and Zhang and McDonald (2012). For English-I, these parsers are in advantage, since they constrain their predicted trees to be projective and therefore easier to decode. Yet, even in this dataset, our parser's accuracy does not lag behind these strong competitors, being faster than the system of Zhang and McDonald (2012) based on beam search; but considerable slower than the highly optimized vine cascade approach of Rush and Petrov (2012), which permits exact decoding with dynamic programming. For the other four datasets, `TurboParser` achieved the best reported scores, with speeds that are the highest reported among higher-order non-projective parsers. More details about these experiments are presented in Martins et al. (2013).

### 1.7.2 Summarization

Our second task is compressive summarization, where we applied AD$^3$ to the factor graph in Section 1.3.2. The model was trained in a multi-task setting, as described in Almeida and Martins (2013), leveraging datasets for compressive summarization (Berg-Kirkpatrick et al., 2011), manual abstracts from DUC/TAC tasks, and a dataset collected by Woodsend and Lapata (2011) containing 4,481 sentence pairs from the English and Simple English Wikipedias. Evaluation is on the non-update portion of the TAC-2008 dataset, containing 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.

| | AD$^3$ | K$^+$10 | M$^+$11 | RP12 | ZM12 |
|---|---|---|---|---|---|
| **English-I** | **93.07**/735 | 92.46/112 | 92.53/66 | 92.7–/4,460* | 93.06/220 |
| **English-II** | **93.22**/785 | 92.57/131 | 92.68/– | | |
| **Dutch** | **86.19**/599 | 85.81/121 | 85.53/– | | |
| **German** | **92.41**/965 | | 91.89/– | 90.8–/2,880 | 91.35/– |
| **Czech** | **90.32**/501 | | 89.46/– | | |

**Table 1.2**: Dependency parsing results for different datasets. Reported are unlabeled attachment scores ignoring punctuation (left), and parsing speeds in tokens per second (right). K$^+$10 is Koo et al. (2010); M$^+$11 is Martins et al. (2011b); RP12 is Rush and Petrov (2012); ZM12 is Zhang and McDonald (2012). Our speeds include the time necessary for pruning, evaluating features, and decoding, as measured on a Intel Core i7 processor @3.4 GHz. The others are speeds reported in the cited papers, converted to tokens per second. The RP12 results on English I, marked with an asterisk, are not directly comparable, since a different dependency conversion scheme was used (see Martins et al. 2013 for details).

We applied a simple rounding procedure to obtain valid summaries from fractional LP-MAP solutions; see Almeida and Martins (2013) for details.

Table 1.3 summarizes the results. The three leftmost columns refer to strong baselines: ICSI-1, the best performing system in the TAC-2008 evaluation (which is extractive); and two recent compressive summarizers (Berg-Kirkpatrick et al., 2011; Woodsend and Lapata, 2012). All these systems use off-the-shelf ILP solvers. The remaining columns show the results achieved by our implementation of a pure extractive system and a compressive summarizer, decoded with GLPK (the ILP solver used by Berg-Kirkpatrick et al. 2011) and AD$^3$. The ROUGE scores show that the compressive summarizers yield considerable benefits in content coverage over extractive systems, confirming previous findings (Berg-Kirkpatrick et al., 2011). Our gains with respect to the compressive baselines come from our multi-task training procedure. Regarding the runtimes, we see that AD$^3$ is orders of magnitude faster than the (exact) ILP solver, with a similar accuracy level; and it is competitive with the speed of an extractive summarizer. To our knowledge, this is the first time a compressive summarizer achieves such a favorable accuracy/speed tradeoff. Figure 1.6 shows an example summary.

## 1.8 Related Work

A few related methods have appeared in the literature after our earlier work in AD$^3$ (Martins et al., 2010a, 2011a).

| ICSI-1 | BGK'11 | WL'12 | Extr., ILP | Comp., ILP | Comp., $AD^3$ |
|---|---|---|---|---|---|
| 11.03/– | 11.71/– | 11.37/– | 11.16/0.265 | **12.40**/10.394 | **12.30**/0.406 |

**Table 1.3**: Results for compressive summarization. Shown are ROUGE-2 recall scores (Lin, 2004) and averaged runtimes (in seconds) to solve a summarization problem in TAC-2008. The three leftmost baselines are Gillick et al. (2008); Berg-Kirkpatrick et al. (2011); Woodsend and Lapata (2012). The remaining columns are our implementation of ILP-based extractive and compressive systems (decoded with GLPK) and our compressive system decoded with $AD^3$.

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 1.6**: Example summary from our system. Removed text is *grayed out*.

Meshi and Globerson (2011) also applied ADMM to LP-MAP decoding, although addressing the *dual* rather than the primal. Yedidia et al. (2011) proposed a *divide-and-concur algorithm* for low-density parity check (LDPC) decoding, which resembles a non-convex version of $AD^3$. Barman et al. (2011) proposed an algorithm analogous to $AD^3$ for the same LDPC decoding problem; their subproblems correspond to projections onto the parity polytope, for which they have derived an efficient algorithm. More recently, Fu et al. (2013) proposed a Bethe-ADMM procedure resembling $AD^3$, but with a variant of ADMM that makes the subproblems become local marginal computations.

A different strategy was proposed by Jojic et al. (2010) to overcome the limitations of the PSDD algorithm pointed out in Section 1.5. They also achieve a $O(1/\epsilon)$ iteration bound by smoothening the objective of (1.12) with an "entropic" perturbation (controlled by a "temperature" parameter), and applying an accelerated gradient method (Nesterov, 2005). An advantage is that the local subproblems become marginal computations, which are formally simpler than our QPs (1.25). However, there are two drawbacks: first, they need to operate at near-zero temperatures, which leads to numerical instabilities in some structured factors (the $O(1/\epsilon)$ bound

requires setting the temperature to $O(\epsilon)$); second, the solution of the local subproblems is always *dense*, which precludes the benefits of caching.

## 1.9 Conclusions

We introduced AD$^3$, an LP-MAP decoder based on ADMM. AD$^3$ enjoys the modularity of dual decomposition methods, and achieves faster consensus than subgradient algorithms, both theoretically (with its $O(1/\epsilon)$ iteration bound) and in practice.

AD$^3$ can handle hard constraint factors, such as those arising from statements in first-order logic and budget constraints, which are typical in NLP and IR applications. For dense and structured factors, we introduced an *active set method* for solving the AD$^3$ subproblems, which requires only a local MAP decoder as a black-box. We have shown the effectiveness of AD$^3$ in two important NLP applications, dependency parsing and summarization.

## Acknowledgements

I would like to thank Noah Smith, Mário Figueiredo, Eric Xing, Pedro Aguiar, and Miguel Almeida. This work was partially supported by the EU/FEDER programme, QREN/POR Lisboa (Portugal), under the Intelligo project (contract 2012/24803), and by a FCT grant PTDC/EEI-SII/2312/2012.

## 1.10 References

M. B. Almeida and A. F. T. Martins. Fast and robust compressive summarization with dual decomposition and multi-task learning. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 2013.

S. Barman, X. Liu, S. Draper, and B. Recht. Decomposition methods for large scale LP decoding. In *49th Annual Allerton Conference on Communication, Control, and Computing*, pages 253–260. IEEE, 2011.

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

C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding. In *Proc. of International Conference on Communications*, volume 93, pages 1064–1070, 1993.

D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. *Distributed Optimization*

*and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Publishers, 2011.

S. Buchholz and E. Marsi. CoNLL-X shared task on multilingual dependency parsing. In *International Conference on Natural Language Learning*, 2006.

X. Carreras. Experiments with a higher-order projective dependency parser. In *International Conference on Natural Language Learning*, 2007.

M. Chang, L. Ratinov, and D. Roth. Constraints as prior knowledge. In *International Conference of Machine Learning: Workshop on Prior Knowledge for Text and Language Processing*, July 2008.

Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

D. Das, A. Martins, and N. Smith. An exact dual decomposition algorithm for shallow semantic parsing with constraints. In *Proc. of First Joint Conference on Lexical and Computational Semantics (*SEM)*, 2012.

J. Duchi, D. Tarlow, G. Elidan, and D. Koller. Using combinatorial optimization within max-product belief propagation. *Advances in Neural Information Processing Systems*, 19, 2007.

J. Eckstein and D. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55(1):293–318, 1992.

J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of International Conference on Computational Linguistics*, pages 340–345, 1996.

H. Everett III. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

Q. Fu, H. Wang, and A. Banerjee. Bethe-ADMM for tree decomposition based parallel MAP inference. In *Proc. of Uncertainty in Artificial Intelligence*, 2013.

D. Gabay and B. Mercier. A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers and Mathematics with Applications*, 2(1):17–40, 1976.

D. Gillick, B. Favre, and D. Hakkani-Tur. The ICSI summarization system at TAC 2008. In *Proc. of Text Understanding Conference*, 2008.

A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. *Neural Information Processing Systems*, 20, 2008.

R. Glowinski and P. Le Tallec. *Augmented Lagrangian and operator-splitting methods in nonlinear mechanics*. Society for Industrial Mathematics, 1989.

R. Glowinski and A. Marroco. 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, 1975.

M. Guignard and S. Kim. Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Mathematical programming*, 39(2):215–228, 1987.

M. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory*

*and Applications*, 4:302–320, 1969.

V. Jojic, S. Gould, and D. Koller. Accelerated dual decomposition for MAP inference. In *International Conference of Machine Learning*, 2010.

K. Knight and D. Marcu. Statistics-based summarization—step one: Sentence compression. In *Prof. of National Conference on Artificial Intelligence*, 2000.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:1568–1583, 2006.

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

T. Koo and M. Collins. Efficient third-order dependency parsers. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, pages 1–11, 2010.

T. Koo, A. M. Rush, M. Collins, T. Jaakkola, and D. Sontag. Dual decomposition for parsing with non-projective head automata. In *Proc. of Empirical Methods for Natural Language Processing*, 2010.

F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47, 2001.

S. Kübler, R. McDonald, and J. Nivre. *Dependency parsing*. Morgan & Claypool Publishers, 2009.

S. Lauritzen. *Graphical Models*. Clarendon Press, Oxford, 1996.

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

M. Marcus, M. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

A. F. T. Martins. *The Geometry of Constrained Structured Prediction: Applications to Inference and Learning of Natural Language Syntax*. PhD thesis, Carnegie Mellon University and Instituto Superior Técnico, 2012.

A. F. T. Martins, N. A. Smith, and E. P. Xing. Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proc. of Annual Meeting of the Association for Computational Linguistics*, 2009.

A. F. T. Martins, N. A. Smith, E. P. Xing, P. M. Q. Aguiar, and M. A. T. Figueiredo. Augmented Dual Decomposition for MAP Inference. In *Neural Information Processing Systems: Workshop in Optimization for Machine Learning*, 2010a.

A. F. T. Martins, N. A. Smith, E. P. Xing, M. A. T. Figueiredo, and P. M. Q. Aguiar. Turbo Parsers: Dependency Parsing by Approximate Variational Inference. In *Proc. of Empirical Methods for Natural Language Processing*, 2010b.

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

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

A. F. T. Martins, M. A. T. Figueiredo, P. M. Q. Aguiar, N. A. Smith, and E. P. Xing.

Alternating directions dual decomposition, 2012. Arxiv preprint arXiv:1212.6550.

A. F. T. Martins, M. B. Almeida, and N. A. Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 2013.

R. McDonald and G. Satta. On the complexity of non-projective data-driven dependency parsing. In *International Conference on Parsing Technologies*, 2007.

R. T. McDonald, F. Pereira, K. Ribarov, and J. Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of Empirical Methods for Natural Language Processing*, 2005.

R. J. McEliece, D. J. C. MacKay, and J. F. Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2), 1998.

O. Meshi and A. Globerson. An Alternating Direction Method for Dual MAP LP Relaxation. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.

Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

J. Nocedal and S. Wright. *Numerical optimization*. Springer, 1st edition, 1999.

S. Nowozin and C. Lampert. Global connectivity potentials for random field models. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 818–825. IEEE, 2009.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

M. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press, 1969.

M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62 (1):107–136, 2006.

T. Richardson and R. Urbanke. *Modern coding theory*. Cambridge University Press, 2008.

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

A. M. Rush and S. Petrov. Vine pruning for efficient multi-pass dependency parsing. In *Proc. of Conference of the North American Chapter of the Association for Computational Linguistics*, 2012.

M. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, 4:113–130, 1976.

D. Smith and J. Eisner. Dependency parsing by belief propagation. In *Proc. of Empirical Methods for Natural Language Processing*, 2008.

N. A. Smith. *Linguistic Structure Prediction*, volume 13 of *Synthesis Lectures on Human Language Technologies*. Morgan and Claypool, May 2011.

D. Sontag, A. Globerson, and T. Jaakkola. Introduction to dual decomposition for inference. In *Optimization for Machine Learning*. MIT Press, 2011.

M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. *Proc. of International Conference on Natural Language Learning*, 2008.

C. Sutton. Collective segmentation and labeling of distant entities in information extraction. Technical report, DTIC Document, 2004.

R. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on Information Theory*, 27(5):533–547, 1981.

D. Tarlow, I. E. Givoni, and R. S. Zemel. HOP-MAP: Efficient message passing with high order potentials. In *AISTATS*, 2010.

M. Wainwright and M. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers, 2008.

M. Wainwright, T. Jaakkola, and A. Willsky. MAP estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.

H. Wang and A. Banerjee. Online Alternating Direction Method. In *Proc. of International Conference on Machine Learning*, 2012.

T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1165–1179, 2007.

K. Woodsend and M. Lapata. Wikisimple: Automatic simplification of wikipedia articles. In *Proc. of AAAI Conference on Artificial Intelligence*, pages 927–932, 2011.

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

H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proc. of International Conference on Parsing Technologies*, 2003.

C. Yanover, T. Meltzer, and Y. Weiss. Linear programming relaxations and belief propagation–an empirical study. *Journal of Machine Learning Research*, 7:1887–1907, 2006.

J. Yedidia, Y. Wang, and S. Draper. Divide and concur and difference-map BP decoders for LDPC codes. *IEEE Transactions on Information Theory*, 57(2): 786–802, 2011.

H. Zhang and R. McDonald. Generalized higher-order dependency parsing with cube pruning. In *Proc. of Empirical Methods in Natural Language Processing*, 2012.