Alexandre Tachard Passos

"Combinatorial algorithms and linear programming for inference in natural language processing."

"Algoritmos combinatórios e programação linear para inferência em processamento de linguagem natural."

CAMPINAS
2013

University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Alexandre Tachard Passos

"Combinatorial algorithms and linear programming for inference in natural language processing."

Supervisor: Prof. Dr. Jacques Wainer
Orientador(a):

"Algoritmos combinatórios e programação linear para inferência em processamento de linguagem natural."

PhD Thesis presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a Doctor degree in Computer Science.

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.

This volume corresponds to the final version of the Thesis defended by Alexandre Tachard Passos, under the supervision of Prof. Dr. Jacques Wainer.

Este exemplar corresponde à versão final da Tese defendida por Alexandre Tachard Passos, sob orientação de Prof. Dr. Jacques Wainer.

[illegible]

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2013

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

P268c

Passos, Alexandre Tachard, 1986-
Combinatorial algorithms and linear programming for inference in natural language processing / Alexandre Tachard Passos. – Campinas, SP : [s.n.], 2013.

Orientador: Jacques Wainer.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Aprendizado de máquina. 2. Processamento da linguagem natural (Computação). 3. Algoritmos. 4. Programação linear. 5. Análise combinatória. I. Wainer, Jacques,1958-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Algoritmos combinatórios e de programação linear para inferência em processamento de linguagem natural
Palavras-chave em inglês:
Machine learning
Natural language processing (Computer science)
Algorithms
Linear programming
Combinatorial analysis
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Jacques Wainer [Orientador]
Andrew Kachites McCallum
Sebastian Robert Riedel
Siome Klein Goldenstein
Eduardo Alves do Valle Júnior
Data de defesa: 28-08-2013
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Tese Defendida e Aprovada em 16 de Agosto de 2013, pela Banca examinadora composta pelos Professores Doutores:

[illegible]

Prof. Dr. Andrew Kachites McCallum
UMASS / AMHERST

[illegible]

Prof. Dr. Sebastian Robert Riedel
DCS - University College, London

[illegible]

Prof. Dr. Siome Klein Goldenstein
IC / UNICAMP

Eduardo A. do Valle Jr.

Prof. Dr. Eduardo Alves do Valle Júnior
FEEC / UNICAMP

[illegible]

Prof. Dr. Jacques Wainer
IC / UNICAMP

Institute of Computing / *Instituto de Computação*
University of Campinas / *Universidade Estadual de Campinas*

Combinatorial algorithms and linear programming for inference in natural language processing.

Alexandre Tachard Passos

August 28, 2013

Abstract

In natural language processing, and in general machine learning, probabilistic graphical models (and more generally structured linear models) are commonly used. Although these models are convenient, allowing the expression of complex relationships between many random variables one wants to predict given a document or sentence, most learning and prediction algorithms for general models are inefficient. Hence there has recently been interest in using linear programming relaxations for the inference tasks necessary when learning or applying these models.

This thesis presents two contributions to the theory and practice of linear programming relaxations for inference in structured linear models. First we present a new algorithm, based on column generation (a technique which is dual to the cutting planes method) to accelerate the Viterbi algorithm, the most popular exact inference technique for linear-chain graphical models. The method is also applicable to tree graphical models and hypergraph models. Then we present a new linear programming relaxation for the problem of joint inference, when one has many submodels and wants to predict using all of them at once. In general joint inference is NP-complete, but algorithms based on dual decomposition have proven to be efficiently applicable for the case when the joint model can be expressed as many separate models plus linear equality constraints. This thesis proposes an extension to dual decomposition which allows also the presence of factors which score parts that belong in different submodels, improving the expressivity of dual decomposition at no extra computational cost.

Resumo

Em processamento de linguagem natural, e em aprendizado de máquina em geral, é comum o uso de modelos gráficos probabilísticos (probabilistic graphical models). Embora estes modelos sejam muito convenientes, possibilitando a expressão de relações complexas entre várias variáveis que se deseja prever dado uma sentença ou um documento, algoritmos comuns de aprendizado e de previsão utilizando estes modelos são frequentemente ineficientes. Por isso têm-se explorado recentemente o uso de relaxações usando programação linear deste problema de inferência.

Esta tese apresenta duas contribuições para a teoria e prática de relaxações de programação linear para inferência em modelos probabilísticos gráficos. Primeiro, apresentamos um novo algoritmo, baseado na técnica de geração de colunas (dual à técnica dos planos de corte) que acelera a execução do algoritmo de Viterbi, a técnica mais utilizada para inferência em modelos lineares. O algoritmo apresentado também se aplica em modelos que são árvores e em hipergrafos. Em segundo mostramos uma nova relaxação linear para o problema de inferência conjunta, quando se quer acoplar vários modelos, em cada qual inferência é eficiente mas em cuja junção inferência é NP-completa. Esta tese propõe uma extensão à técnica de decomposição dual (dual decomposition) que permite além de juntar vários modelos a adição de fatores que tocam mais de um submodelo eficientemente.

Iara,

Acknowledgements

First to my advisor, Jacques Wainer, who supported me throughout this very odd process. I'd also like to specially thank Andrew McCallum, for treating me as his own student, and teaching me more than I could have hoped for. Looking back over the past two years I've learned undeniably much, and most of it from him.

There were many others who held advisory roles for me, and whom I want to thank: Pedro Kröger, George Lima, Siome Goldenstein, Joseph Turian, Aria Haghighi, Hanna Wallach, Sebastian Riedel, Jurgen van Gael, Tushar Chandra, and others.

Most of this thesis would not have been possible without the collaboration of my colleagues. First David Belanger, who co-authored many of my papers, and on whom I bounced many ideas in the past years. Special thanks to Daniel Duckworth for having read earlier drafts of this thesis and provided substantial comments on it. One learns the most from one's peers, and I learned much from the others at IESL: Sam Anzaroot, Anton Bakalov, Jinho Choi, Laura Dietz, Gregory Druck, Ari Kobren, Vineet Kumar, Brian Martin, David Mimno, Harshal Pandya, Sameer Singh, David Soergel, Luke Vilnis, Michael Wick, Limin Yao, and Jiaping Zheng; and at UNICAMP: Daniel Cason, Murilo de Lima, Jefferson Moisés, Robson Peixoto, Heitor Nicoliello, and others.

During the PhD process I had the good fortune of doing two internships in research labs: Microsoft Research Cambridge and Google Research. Both experiences were very enlightening, and substantially changed my world view. I'd like to thank the people I worked with, at Microsoft: Khalid El-Arini, Ali Eslami, Michael Gartrell, Thore Graepel, Ralf Herbrich, Tom Minka, Ulrich Paquet, Matthew Smith, David Stern, Martin Szummer, and others; and at Google: Kevin Canini, Rafael Frongillo, Jason Gauci, Eugene Ie, Kristen LeFevre, Indraneel Mukherjee, Fernando Pereira, Yoram Singer, Tal Shaked, and others. I also would like to thank CNPQ and CAPES for partially funding my research.

I'd like to thank other people in the field with whom I haven't yet overlapped institutionally—and some haven't even met in person—but who managed to help me regardless: John Myles White, Richard Socher, Yoav Goldberg, Neil Parikh, André Martins, Mark Reid, Nikete Della Penna, Joe Reisinger, Leon Palafox, Hal Daumé III, David Warde-Farley, Mikio Braun, Rob Zinkov, Tiberio Caetano, Yaroslav Bulatov, and others. I had the

opportunity to write a paper with Piyush Rai, which is not in this thesis only because of cohesion. I learned much from software design and machine learning from the scikit-learn folks: Gael Varoquaux, Olivier Grisel, Alexandre Gramfort, Fabian Pedregosa, Andreas Mueller, Mathieu Blondei, Nelle Varoquaux, and others. Also thanks to the `metaoptimize.com/qa` community, for answering my questions.

If not for friends, old and new, I'd have come apart long ago. I'd like to thank first Emma Tosch, Manuela Borzone, and Luisa Galindo, for helping me keep myself together in my time in the US. To my old friends, Aline Bessa, Lola, Ivana Pristed, Nara Almeida, Manuel Sá, Túlio Miranda, Olmo Lacerda, Laís Mendonça, Alais de Hoogh, we may be apart physically but you are still very important to me. There are also the friends in the Amherst area: Jeff Dalton, Cibele Freire, Fabricio Murai, Andrea Vu, Tommy Boucher, John Foley, and others.

Finally my family, Jorgina Ribeiro Tachard, Bruno Tachard Passos, and José Leandro Pereira Passos, who have been very understanding of the distance and constraints of this weird journey.

And last, but not least, to Iara, for whom no simple acknowledgment is enough.

I do not thank the scala compiler ;-).

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In recent years Natural Language Processing (NLP) seems to have matured enough to be profitably applicable in many endeavors. A recent high-profile example is the Watson system from IBM, which won a televised game of Jeopardy! against two of the best known human players [19]. Google uses its large-scale Knowledge Graph [88], built by automatically refining and expanding Freebase [4], a semantic web version of wikipedia, in most of its search queries, which are reportedly tagged and parsed with special-purpose algorithms.

Coupled with the very large corpora easily acquired online, there is now a need to routinely analyze, with a standard NLP pipeline, millions or billions of tokens, not just in industry applications but also in academic endeavors, such as the Knowledge Base Population (KBP) challenges in the Text Analysis Conference (TAC) [60, 29].

This need for fast analysis of large corpora highlights a tension in state-of-the-art approaches to NLP problems, which mostly fall into one of two camps: they are either sequences of classifiers, which are fast at training and prediction time, but are prone to overfitting and often don't achieve the best accuracies, or they are structured linear models, which achieve very high accuracies at the expense of a more computationally intensive training and prediction process. The main challenge in structured linear models is the problem of *inference*, which is determining, out of all possible output structures, either the probability of each structure or which structure is most compatible with the input data.

While in general the problem of inference is intractable, and either $\#P$-hard or NP-hard [36, 106, 59], there are known polynomial-time inference algorithms for most practical models in NLP. Most of these algorithms are of combinatorial nature, based on dynamic programming. That said, these algorithms are still often orders of magnitude slower than simpler approaches based on greedy search or a sequence of classifiers. Moreover, these polynomial-time algorithms do not generally compose, which make the problem of joint inference, both across tasks in a single document, and across documents in a corpus, intractable.

Recently much progress has been made on the tractability of inference in progressively more general structured linear models, much of it using the tools of linear programming relaxations [91]. The same linear programming relaxations allow one to efficiently reduce the problem of joint inference to a sequence of inference problems in reweighted models, using the technique of dual decomposition [37].

In this thesis, instead of focusing on turning previously-intractable problems into tractable ones, we focus on accelerating the known combinatorial algorithms for inference in these problems using techniques from linear programming. In chapter 5 we show how to adapt the technique of column generation, described in section 3.4.2, to accelerate the well-known Viterbi algorithm for decoding in linear chain graphical models, in a way which generalizes to more complex models and allows for a finer grained control of the tradeoffs between prediction time and model accuracy. In chapter 6 we extend the domain on which dual decomposition is applicable, presenting simpler and faster algorithms for the case where different submodels are coupled by more than linear equality constraints, while still reducing the joint problem to a sequence of independent problems solvable with known combinatorial algorithms.

Throughout these chapters we'll look at the connections between the combinatorial and linear programming views of inference in structured linear models, and how these views can be brought together to accelerate state-of-the-art approaches to solving natural language processing problems.

Chapter 2

Natural Language Processing

In this chapter we will look at basic terminology, definitions, and tasks for natural language processing (NLP).

Natural language processing can be broadly defined as the computational study of language and its manifestations. While linguists have long studied language, identifying many universalities across languages, their structures, and the forms of individual variation of some concepts across linguistic groups and individuals, having computational tools available allows for quantitative analysis at a scale which is not made possible by small-scale experiments. A large fraction of linguistic knowledge can be seen as mapping from observed representations of language, such as text, to latent structures, which highlight the commonalities across different pieces of text as well as different languages entirely. One of the basic tools of linguistics is a grammar, which is a concise way of describing valid sentences in a language. Grammars are concise because they abstract away from surface syntactic and lexical phenomena and are expressed as compatibility rules over latent structures.

One of the goals of natural language processing, then, is to build systems that take language data as input and produce some kind of linguistic structure as output, mimicking the analytical work which can be performed by linguists. The structure can be interesting in and of itself, as is the case of machine translation or automatic knowledge base generation, but it might also be useful as a building block for other tasks which extract more complex structures from text. Often many natural language processing systems are chained together in a pipeline, with one system's output serving as input to the next system, until finally the end result is obtained. Figure 2.1 shows an example of an NLP pipeline for extracting named entities from text.

In this thesis we will consider only a few natural language processing problems, when the input is text already encoded as a sequence of characters and the processing is done one sentence at a time.

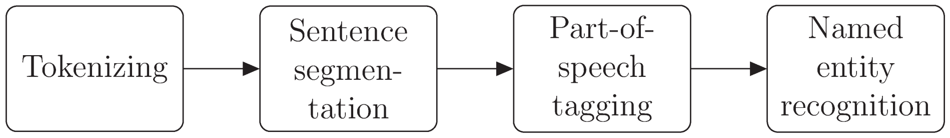


Figure 2.1: An example natural language processing pipeline for extracting named entities from text.

[Doonesbury] [creator's] [union] [troubles] [are] [no] [laughing] [matter] [.]

Figure 2.2: An example sentence, with the token boundaries delineated by []s.

A sentence is a sequence of characters, which are split into tokens, which are usually contiguous sequences of characters separated by space or punctuation on either side. Figure 2.2 shows an example sentence, with the word tokens marked.

We distinguish between a word token, which is a specific sequence of characters in a specific sentence, and its word type, which is the sequence of characters when removed from context. For example, “river” is a word type, while its occurrence in the sentence “she rowed down the river” is the last word token in that sentence.

While historically NLP systems were mostly rule-based or logic-based until the late 1980s, most modern techniques are statistical in nature. We refer the reader to Manning and Schütze [47] for a discussion of some early techniques in statistical natural language processing or Jurafsky and Martin [31] for a more modern treatment. Today most state-of-the-art NLP systems are statistical, with a few exceptions, such as the stanford within-document coreference resolver [43].

The statistical, or corpus-based approach to natural language processing proceeds roughly as follows. First, a large amount of text is annotated by linguistically-trained experts. This is referred to as a corpus. The main corpus used throughout this thesis is the Wall Street Journal section of the Penn Treebank [48], in which close to one million word tokens were annotated with phrase structure parse trees and part of speech tags. Then, an NLP practitioner comes up with a statistical or learning-based model which can, given a sentence, make a prediction of what would the linguistic structure of that sentence look like. The parameters of this model are estimated on a subset of the corpus and the quality of the predictions of the model are evaluated on the remaining section of the corpus. A system is declared to be state-of-the-art if its evaluation is competitive with the best known published results.

There are, of course, alternative scenarios. The most relevant one for this thesis is the domain adaptation scenario, in which a model is trained in a large corpus and evaluated on a different, most likely smaller, corpus comprising data from a different domain. For example, the Penn Treebank does not contain many questions, so a small

[Doonesbury]$_{\text{NOUN}}$ [creator's]$_{\text{NOUN}}$ [union]$_{\text{NOUN}}$ [troubles]$_{\text{NOUN}}$ [are]$_{\text{VERB}}$ [no]$_{\text{DET}}$ [laughing]$_{\text{ADJ}}$ [matter]$_{\text{NOUN}}$ [.]$_{.}$

Figure 2.3: A sentence annotated with universal parts of speech.

corpus of questions was annotated [30], and we will later perform some experiments on applying models trained in the Penn Treebank on the Question Treebank.

Throughout this thesis we will mostly look at two natural language processing problems: predicting parts-of-speech of word tokens and parsing sentences with a dependency grammar.

2.1 Part-of-speech tagging

Often in language the specific type of a given word token matters less to how it fits grammatically in sentences than what is commonly referred to as its part-of-speech. For example, in English, most common nouns can be used written after determiners or adjectives, but are not often used after adverbs or before determiners. Table 2.1 lists 12 parts of speech which can be found in most languages, and comprise the universal part of speech tagset [73]. More commonly, however, individual corpora for specific languages use much finer parts of speech. The Penn Treebank, for example, distinguishes between plural and singular, proper and non proper nouns, among others.

The task of part-of-speech tagging is to tag each token in a sentence with its part-of-speech in that context. Figure 2.3 shows an example sentence from the Penn Treebank tagged with its ground-truth parts of speech.

While most word types have a clear predominant part of speech (and, indeed, predicting for each word type its most frequent part of speech in the Penn Treebank test set is almost as accurate as many state-of-the-art systems), any part of speech tagging system is bound to encounter at test time words it did not see at training time. Also, for the cases in which there is ambiguity, the parts-of-speech of tokens surrounding any given token provide strong evidence as to how that token might be tagged.

The accuracy of state of the art part-of-speech tagging systems on the Penn Treebank, using the standard training, development, and testing splits, is around 97% [46]. Most state of the art approaches either use conditional random fields [41] or independent sequential classification of each token [8].

In this thesis we will mostly concern ourselves with conditional random field models of part-of-speech tagging.

Table 2.1: A list of 12 universal parts of speech.

VERB	Verbs (all tenses and modes)
NOUN	Nouns (common and proper)
PRON	Pronouns
ADJ	Adjectives
ADV	Adverbs
ADP	Adpositions (prepositions and postpositions)
CONJ	Conjunctions
DET	Determiners
NUM	Cardinal numbers
PRT	Particles or other function words
X	Other: foreign words, typos, abbreviations
.	Punctuation

2.2 Dependency parsing

While very useful, parts of speech capture very shallow information at best about the syntactic structure of a sentence. Very often one desires a richer notion of syntactic structure, including phrases, subject-verb-object, which words are being modified by which other words, etc.

While at first most NLP research focused on statistical parsing with context-free grammars, more recently the community has been focusing on parsing with bilexicalized, or dependency, grammars. There are many reasons for this switch, but the main ones are that while context-free annotations are very expressive they are also very difficult to precisely specify and annotate, and much of their structure does not generalize well across languages [67]. Moreover, many other NLP tasks don't require the full detailed structure of a context-free parse, requiring only detection of the phrase boundaries, identification of the main verb of a sentence, the main noun phrases, the subject/object, etc, and all these structures are cheaply obtained from a dependency representation.

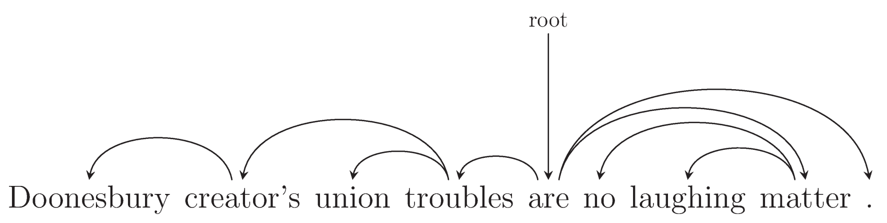

root
Doonesbury creator's union troubles are no laughing matter .

Figure 2.4: An example of a dependency tree. Edges are drawn from heads to modifiers.

A dependency parse is an assignment of a "head word" to each token in a sentence such that the graph formed by these head-to-modifier edges is a directed tree, with one (or sometimes more) words having no "head words". Moreover, for many languages, including English, dependency trees can often assumed to be projective [58]; that is, when drawing dependency edges above the words the edges should not cross (equivalently, if a span is defined by all tokens between a given token's leftmost and rightmost descendants, given any two spans either one strictly contains the other or they do not overlap). Figure 2.4 shows an example of a dependency tree. The head word token of the sentence is the verb "are", its modifiers to the left, headed by "troubles", form the subject, while its modifiers to the right, headed by "matter", form the object.

Most state of the art dependency parsers can be roughly classified into one of two main families: graph-based and transition-based approaches.

Graph-based approaches [58, 56, 38, 17] learn a model that can assign a score to any dependency tree, assuming this score factors as a sum of scores of parts of the tree, and then at test time searches for the highest-scoring tree for any given sentence. For non-projective dependency parsing a first-order model, whose score depends only on the presence or absence of individual edges, can be solved with a directed maximum spanning tree algorithm, but higher-order models, whose scores can depend on the presence or absence of pairs or triples of edges, are known to be intractable [57]. For projective dependency parsing it is possible to use higher-order models with polynomial-time algorithms on the length of the sentence [38], though it is arguable whether measuring asymptotic complexity as sentence length grows is the right approach, as most sentences in most languages tend to be of relatively short length.

Transition-based approaches [68, 24, 7, 26], on the other hand, learn a classifier that will act as a variant of a shift-reduce automaton on each sentence at test time. This family of algorithms has linear or expected linear complexity on the length of a sentence, and are in practice much faster than most graph-based approaches. While the basic shift-reduce model can only produce projective trees there are known extensions of it that can produce any non projective tree. The search for a parse, however, is approximate in transition-based models, and most practical algorithms are known to not find the overall best sequence of state transitions. This also creates some difficulties when justifying parameter estimation for these models [24].

In this thesis we'll mostly concern ourselves with graph-based projective dependency parsing.

Chapter 3

Optimization basics

In this chapter we'll do a brief overview of the main concepts in linear programming and convex optimization.

Let x be a vector in R^n. We write the dot product between two vectors as $x^T y$ or $\langle x, y\rangle$.

In this chapter let $f(x)$ be a function from R^n to $R \cup \{\infty\}$. If it is differentiable let $\nabla f(x)$ be its gradient at the vector x. The set of points on which f is finite, $D_f = \{x | f(x) < \infty\}$ is said to be the domain of f.

Definition 1. *A function $\|\cdot\|$: $R^n \to R$ is a norm it it satisfies the following properties:*

- *triangle inequality, $\|x + y\| \leq \|x\| + \|y\|$ for any vectors x and y;*
- *homogeneity, $\|\alpha x\| = |\alpha| \|x\|$ for any scalar α and vector x; and*
- *separation, $\|x\| = 0$ if and only if $x = 0$.*

Definition 2. *Given a norm $\|\cdot\|$, its dual norm $\|\cdot\|_*$ is defined as*

$$\|y\|_* = \max_{\|x\|=1} x^T y. \tag{3.1}$$

It is easy to see that the two-norm $\|x\|_2 = \sqrt{x^T x}$ is a norm, and its its own dual norm. The sum of the absolute values of the coordinates of a vector is also a norm, $\|x\|_1 = \sum_i |x_i|$, and its dual is the max-norm or infinity-norm, $\|x\|_\infty = \max_i |x_i|$.

Definition 3. *A function f is said to be Lipschitz continuous with constant L under a norm $\|\cdot\|$ if the difference in function value between two points is bounded by a constant times the norm of the difference of the points,*

$$|f(x) - f(y)| \leq L \|x - y\|. \tag{3.2}$$

Definition 4. *The simplex in R^n is the set $\Delta = \{x | x_i \geq 0, \sum_i x_i = 1\}$ of vectors in which all coordinates are positive and sum to one.*

A necessary condition for a point x^* to be a solution to the generic unconstrained optimization problem with a differentiable objective,

$$\min_x f(x), \tag{3.3}$$

is that $\nabla f(x^*) = 0$. This is called the first-order optimality condition for unconstrained optimization problems. It is not, however, a sufficient condition unless more assumptions are made about the problem.

3.1 Constrained optimization and Lagrange duality

In this section we will look at first-order optimality conditions for constrained optimization problems. This presentation follows Boyd and Vandenberghe [5].

Definition 5. *Let $g(x) : R^n \to R^m$ and $h(x) : R^n \to R^k$ be vector-valued functions. A constrained optimization problem is*

$$\begin{aligned} \textbf{\textit{min.}} \quad & f(x) \\ \textbf{\textit{s.t.}} \quad & g_i(x) = 0 \\ & h_j(x) \leq 0 \end{aligned} \tag{3.4}$$

The value of this constrained problem is the smallest possible $f(x^*)$ which can be attained such that x^* satisfies the equality and inequality constraints above. A point which satisfies these constraints is said to be feasible.

We refer to this problem as the primal optimization problem

From the specification of this problem we can construct a family of parametric lower bounds called the dual function.

Definition 6. *Given an optimization problem, we define its Lagrangian as*

$$L(x, \lambda, \gamma) = f(x) + \lambda^T g(x) + \gamma^T h(x). \tag{3.5}$$

Note that minimizing the Lagrangian with respect to x, for any value of λ and for any non-negative value of γ, lower-bounds the value of the constrained optimization problem, as it is always possible to select a feasible x, and then the second term in the Lagrangian is zero and the third term is non negative.

Definition 7. *We refer to the result of this minimization, as a function of λ and γ, as the Lagrange dual function,*

$$D(\lambda, \gamma) = \min_x L(x, \lambda, \gamma), \tag{3.6}$$

and we refer to the problem of maximizing the dual function as the dual optimization problem

$$\mathbf{max.}_{\lambda,\gamma} D(\lambda, \gamma). \tag{3.7}$$

Under some conditions one can prove strong duality; that is, that the maximum value of the dual function, over all possible λ and positive γ, is equal to the minimum of the constrained objective function.

Then, for an optimal feasible point x^* of the primal problem and an optimal feasible point λ^*, γ^* of the dual problem, we have that

$$\begin{aligned} f(x^*) &= D(\lambda^*, \gamma^*) && (3.8)\\ &= \min_x f(x) + \lambda^{*T} g(x) + \gamma^{*T} h(x) && (3.9)\\ &\leq f(x^*) + \lambda^{*T} g(x^*) + \gamma^{*T} h(x^*) && (3.10)\\ &\leq f(x^*) + \gamma^{*T} h(x^*) && (3.11)\\ 0 &\leq \gamma^{*T} h(x^*) && (3.12) \end{aligned}$$

in which the first equality is strong duality, the second is the definition of the dual function, the third follows from the fact that the minimizer has lower value than any point, and the fourth from the fact that feasibility of x^* implies that $g(x^*) = 0$.

Then, because we know that γ_j^* is positive and $h_j(x^*)$ is non positive, we have that, for any primal-dual optimal setting x^*, λ^*, γ^*,

$$\gamma_j^* h_j(x^*) = 0, \tag{3.13}$$

or, equivalently, that either $\gamma_j^* = 0$ or $h_j(x^*) = 0$ or both. This is referred to as the *complimentary slackness condition.*

Moreover, since x^* minimizes the Lagrangian, we must have that

$$\nabla f(x^*) + \lambda^{*T} \nabla g(x^*) + \gamma^{*T} \nabla h(x^*) = 0. \tag{3.14}$$

This is referred to as the *stationarity condition.*

Together with *feasibility* (i.e., that $g(x) = 0$ and $h(x) \leq 0$) these form the Karush-Kuhn-Tucker necessary conditions for optimality of optimization problems. For a broader discussion of Lagrange duality, how it relates to optimization, the sufficient counterparts of the KKT conditions, and other issues we refer the reader to Boyd and Vandenberghe [5].

3.2 Convex optimization

In this section we'll see how the general results from section 3.1 apply to specifically convex functions. We'll state many general results from convex analysis without proof, and we refer the reader to Boyd and Vandenberghe [5] for the proofs.

Definition 8. *Given a vector $\alpha \in \Delta$ in which each coordinate is non negative and they all sum to one, a convex combination of a set of vectors x_i is defined as $\sum_i \alpha_i x_i$. A set is said to be convex if given n points in the set, their convex combination is also in the set.*

Definition 9. *From any function f we can define its epigraph as the set $(x, y) : f(x) \leq y$.*

A function is said to be convex if its epigraph is convex. It is easy to see that this implies Jensen's inequality

$$f(E[x]) \leq E[f(x)]. \tag{3.15}$$

For any two convex sets S_1 and S_2 there exists a separating hyperplane h such that $h^T x_1 \geq h^T x_2$ for all $x_1 \in S_1, x_2 \in S_2$. This holds even for a point x in the boundary of a convex set and the interior of the set. For any convex function, then, for all points x in its domain, there exists a hyperplane g such that

$$f(y) \geq f(x) + g^T(y - x). \tag{3.16}$$

We refer to g as a point in the *subgradient* of f, which we write as $g \in \partial f(x)$. It is possible to prove that if f is differentiable at x there is only one such g and it is equal to the gradient of f at x. Similarly to the gradient, if a function is Lipschitz with constant L then the norm of all of its subgradients is bounded by L.

Subgradients can be constructed from essentially the same rules as gradients (chain rule, linearity, etc), with a few additional rules. An important one is that for a function $f(x) = \max_i f_i(x)$, any subgradient of one maximizing $f_i(x)$ is a subgradient of $f(x)$,

$$\partial_x f(x) = \{\partial_x f_j(x) | f_j(x) = \max_i f_i(x)\}. \tag{3.17}$$

An optimization problem is said to be convex if it can be written as

$$\begin{aligned} \textbf{min.} \quad & f(x) \\ \textbf{s.t.} \quad & g_i(x) \leq 0 \\ & Ax = b \end{aligned} \tag{3.18}$$

in which f and g_i are convex functions.

For convex optimization problems, the KKT conditions described in section 3.1 are necessary and sufficient for optimality, as long as there is an open set of feasible points (this is known as Slater's condition). If the function is not differentiable optimality can be proven as long as there is a subgradient of f and g which satisfies the KKT conditions.

The Fenchel dual of a convex function is defined as

$$f^*(y) = \max_x y^T x - f(x). \tag{3.19}$$

The function and its dual obey the Fenchel-Young inequality,

$$y^T x \leq f(x) + f^*(y), \tag{3.20}$$

and also the gradient mapping properties,

$$\begin{aligned} \nabla f(\nabla f^*(y)) &= y & (3.21)\\ \nabla f^*(\nabla f(x)) &= x. & (3.22) \end{aligned}$$

An example of a function which is its own Fenchel conjugate is the two-norm squared, $f(x) = \frac{1}{2}\|x\|^2$.

3.2.1 The subgradient method

One general-purpose method for solving unconstrained convex optimization problems is the subgradient method. One starts with $x_0 = 0$, and at each iteration one sets $x_{t+1} = x_t - \eta_t \partial f(x_t)$; that is, points are perturbed by adding a learning rate η_t times a subgradient of the function at the current iterate. Assuming that the subgradients have a norm bounded by L and that the norm of the optimal point x^* is less than R, one has that

$$\begin{aligned} \|x_{t+1} - x^*\|^2 &= \|x_t - \eta_t \partial f(x_t) - x^*\|^2 & (3.23)\\ &= \|x_t - x^*\|^2 - 2\eta_t \partial f(x_t)^T (x_t - x^*) + \eta_t^2 \|\partial f(x_t)\|^2 & (3.24)\\ &\leq \|x_t - x^*\|^2 - 2\eta_t \partial f(x_t)^T (x_t - x^*) + \eta_t^2 L^2 & (3.25)\\ &\leq \|x_t - x^*\|^2 - 2\eta_t (f(x_t) - f(x^*)) + \eta_t^2 L^2 & (3.26)\\ &\leq \|x^*\|^2 - 2\sum_i \eta_i (f(x_i) - f(x^*)) + \sum_i \eta_t^2 L^2 & (3.27)\\ 2\sum_i \eta_i (f(x_i) - f(x^*)) &\leq \|x^*\|^2 + \sum_i \eta_t^2 L^2 & (3.28)\\ (\min_i f(x_i) - f(x^*)) &\leq \frac{R^2 + \sum_i \eta_t^2 L^2}{2\sum_i \eta_i} & (3.29) \end{aligned}$$

And hence any schedule for the learning rates η_i for which $\lim_{i\to\infty} \frac{\sum_i \eta_i^2}{\sum_i \eta_i} = 0$ will converge to the optimal solution; moreover if the learning rates are set to $\frac{1}{\sqrt{T}}$ then the error after T iterations is on the order of $O\left(\frac{1}{\sqrt{T}}\right)$.

There is a matching lower bound stating that any first-order method for a non differentiable convex function will have this rate in the worst case, though there are other more efficient methods if more assumptions are made about the function. See Nesterov [65] for more details and a broader discussion of different optimization methods and their convergence rates.

3.3 Dual decomposition

While there exist general-purpose optimization methods, which can solve any convex optimization problem, there are specialized algorithms which exploit the properties of specific function families, and hence can be more efficient. For example, as will be seen in section 4.3, most dynamic programming algorithms correspond to a linear program, and hence can be solved by entering the linear program into a general-purpose LP solver such as Gurobi [28]. This is, however, impractical, as simply writing down all the constraints of a dynamic program is as expensive as directly solving it.

While the set of functions for which a special-purpose algorithm is known is fairly restricted, many problems can be decomposed into subproblems which can be solved using specialized algorithms. Dual decomposition is one of many techniques that allow one to exploit such a decomposition.

In the simplest case, consider the following optimization problem,

$$\textbf{min}. f(x) + g(x) \tag{3.30}$$

where f and g are convex and can be efficiently solved with specialized algorithms. One can then rewrite this problem as follows

$$\begin{aligned} \textbf{min.} \quad & f(x) + g(y) \\ \textbf{s.t.} \quad & x = y \end{aligned} \tag{3.31}$$

and then look at its corresponding dual problem,

$$\textbf{max.}_\lambda \quad D(\lambda) = (\min_x f(x) - \lambda^T x) + (\min_y g(y) + \lambda^T y). \tag{3.32}$$

This dual problem can then be solved with the subgradient method, as described in section 3.2.1. Note that a subgradient of this objective with respect to lambda is the difference between a minimizing x for the first part and a minimizing y for the second one, as a consequence of equation (3.17). Also note that when a minimizer x^* of the first subproblem is identical to a minimizer y^* of the second subproblem—that is, the primal constraint is satisfied—the value of the optimization problem does not depend on λ.

The fully general case of dual decomposition is as follows. There are n problems $f_i(x_i)$ which one wants to optimize, with the linear agreement constraint that $\sum_i A_i x_i = 0$. One can write the primal as

$$\begin{aligned} \textbf{min.} \quad & \sum_i f_i(x_i) \\ \textbf{s.t.} \quad & \sum_i A_i x_i = 0 \end{aligned} \tag{3.33}$$

Adding a dual variable λ to account for the constraint we can write the Lagrangian as

$$L(\mathbf{x}, \lambda) = \sum_i f_i(x_i) + \lambda^T \left(\sum_i A_i x_i \right). \tag{3.34}$$

Regrouping the terms we can write the dual problem as

$$\max_\lambda D(\lambda) = \sum_i \min_{x_i} (f_i(x_i) + \lambda^T A_i x_i) \tag{3.35}$$

Let x_i^* be a maximizer of $f_i(x_i) + \lambda^T A_i x_i$. Then a subgradient algorithm for the dual problem updates λ as follows:

$$\lambda^{(t+1)} = \lambda^{(t)} + \eta_t \sum_i A_i x_i^*. \tag{3.36}$$

Note that the matrices A_i are unrestricted. They can, for example, enforce that only certain coordinates of the vectors x_i have to agree, or project differently-sized x_i and x_j vectors into a shared space on which agreement is enforced.

3.4 Linear programming

In this section we'll restrict ourselves to the case in which $f(x)$, $g(x)$, and $h(x)$ are all linear functions of x. As any linear equality $g_i^T x = b$ can be expressed as two linear inequalities $g_i^T x \leq b$ and $-g_i^T x \leq b$ we'll talk about the general linear optimization problem, or *linear program* (LP) as

$$\begin{aligned} \textbf{min.} \quad & c^T x \\ \textbf{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned} \tag{3.37}$$

where $Ax \leq b$ is shorthand for stating that for each dimension i of the vector Ax, $(Ax)_i \leq b_i$.

We can write the Lagrangian for the linear program as

$$L(x, \lambda) = c^T x - \lambda^T (Ax - b), \tag{3.38}$$

with the constraints that x and λ are positive. Rearranging to pull out λ gives us

$$L(x, \lambda) = b^T \lambda - x^T (A^T \lambda - c), \tag{3.39}$$

which is the (negated) Lagrangian of the following linear program

$$\begin{aligned} \textbf{max.} \quad & b^T \lambda \\ \textbf{s.t.} \quad & A^T \lambda \geq c \\ & \lambda \geq 0 \end{aligned} \tag{3.40}$$

Note that each variable in the primal program in equation (3.37) became a constraint in the dual program, and likewise each constraint in the primal program is a dual variable in the dual program. It is also easy to see that if one allows variables to be negative (which can be represented by replacing x_i in the program with the difference between two nonnegative variables $x_i^+ - x_i^-$) the corresponding dual constraint has to be satisfied with equality. The dual of a dual problem, it's easy to see, is again the primal problem in the canonical form.

Because linear programs have linear objective and linear inequality constraints and linear functions are convex, they are convex optimization problems, and hence as was seen in section 3.2 the KKT conditions are sufficient for optimality, so it is enough to produce a pair of primal and dual variables which satisfy the stationarity, feasibility, and complimentary slackness conditions to show that one has solved the linear program exactly. Alternatively, presenting a feasible setting of the primal variables and a feasible setting of the dual variables which have the same objective value is a sufficient condition for optimality.

It is often useful to talk about an *integer linear program* (ILP); that is, a linear program in which the variables x are restricted to be integral.

There are many algorithms for solving linear programs, though most practical general-purpose techniques are based on either the simplex algorithm or interior point methods. For more information on linear programming we refer the reader to Dantzig [13].

3.4.1 Linear programming for combinatorial optimization

Many important combinatorial optimization problems can be framed as linear programs. Moreover, there is a close connection between popular dynamic programming algorithms for combinatorial problems and their linear programming representation.

As an illustration, in this section we'll look at the problem of shortest paths in a directed acyclic graph. A directed acyclic graph G is defined as a tuple (V, E), where V is a set of vertices (or nodes) and E is a set of edges. Each $e \in E$ is itself a tuple (i_e, j_e, w_e), where $i \in V, j \in V$ are the source and destination of the edge and w_e is its weight. A sequence $s, \ldots, t$ of nodes is said to be a path if there is an edge between any two adjacent nodes in the path. A graph is said to be acyclic if there is no non-empty path which starts and ends at the same node.

The problem of finding the shortest path between any two nodes s and t in a directed acyclic graph can be framed as the following linear problem, with one variable e_i indicating whether edge i is in the path and one constraint per node:

$$\begin{aligned} \textbf{min.} \quad & \sum_i w_{e_i} e_i \\ \textbf{s.t.} \quad & \sum_{e_i : j_{e_i} = t} e_i = 1 \\ & \sum_{j_{e_j} = n} e_j - \sum_{i_{e_i} = n} e_i = 0 \quad \forall n, n \neq s, n \neq t \end{aligned} \tag{3.41}$$

The dual of this LP, then, has one variable per node and one constraint per edge in the graph, and is as follows

$$\begin{aligned} \textbf{max.} \quad & V_t \\ \textbf{s.t.} \quad & V_{j_e} - V_{i_e} \leq w_e \quad \forall e, i_e \neq s \\ & V_s = 0 \end{aligned} \tag{3.42}$$

where the last constraint exists just because there was no constraint for the source variable in the primal problem, so it shouldn't have a dual variable, and will show up with value 0 in all constraints it appears in.

These constraints then can be rewritten as saying that the value of each node is less or equal to the minimum, over all its incoming edges, of the weight of the edge plus the value of the node from which it comes, as in

$$V_n \leq \min_{e : j_e = n} w_e + V_{i_e}. \tag{3.43}$$

Then, because the graph is acyclic, we can order the nodes such that all nodes with an edge entering node n come before n in the ordering, and set the variables to make the inequalities in equation (3.43) tight. This is referred to as *dual coordinate ascent*, as each dual variable is optimized in turn to satisfy all the constraints. This is guaranteed to produce a dual feasible solution because the graph is acyclic. At the same time, if you store the maximizers from equation (3.43) you can reconstruct a primal solution by setting the primal variables corresponding to those edges to 1 and all others to zero. Doing so with backtracking is guaranteed to maintain primal feasibility and it will have the same objective value as the one obtained by dual coordinate ascent, so they form a primal-dual optimal pair.

The algorithm we just derived by reasoning about this linear program is the standard breadth-first-search algorithm for finding shortest paths in acyclic graphs.

3.4.2 Cutting planes and column generation

In this section we will describe the dual techniques of cutting planes and column generation for solving linear programs with a large number of constraints or variables. These techniques are very general, and can be extended to solving general convex problems. For a more thorough survey we refer the reader to Mitchell [62] or other operations research texts.

Historically these techniques seem to have been proposed around the ellipsoid algorithm [33, 64] and the Dantzig-Wolfe decomposition [12], as a way of reducing the solution of exponentially large linear programs to the repeated solution of tractable linear programs. While the precise structure of the ellipsoid algorithm or the Dantzig-Wolfe decomposition are not relevant to this thesis, many of their usages in practice involve the meta algorithms of cutting planes and column generation, which are central to chapter 5.

Assuming a linear program is feasible, the linear inequality constraints then define a polytope, which is a high-dimensional extension of a polygon. An important property in solving linear programs with more constraints than variables is that if there is a solution then a solution has to lie in a corner of the polytope, and a corner of an n-dimensional polytope is the point in which n inequality constraints are satisfied with equality. Figure 3.1 illustrates this property in two dimensions. Note that by complimentary slackness, then, only these n dual variables can have nonzero values.

This suggests that if one has a linear program with a very large number of constraints, it should be possible to solve it more efficiently by at first ignoring most of these constraints, solving this *relaxed problem* exactly, finding violated constraints, and adding them to the relaxed problem. This process then is repeated until no constraints are violated. This is referred to as the *cutting planes algorithm*, and the pseudocode is displayed

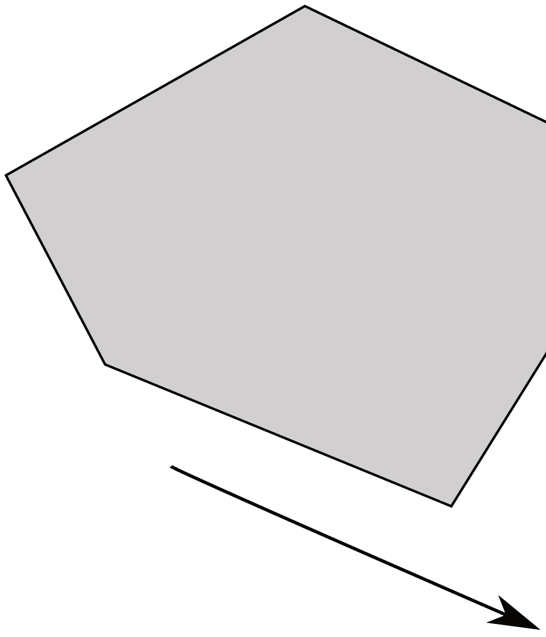


Figure 3.1: An example of a polytope and linear objective.

Algorithm 3.1 The cutting planes algorithm.

```
1: function CuttingPlanes
2:     Initialize and solve the restricted problem
3:     while there is a violated constraint $c$ do
4:         Add $c$ to the restricted problem
5:         Solve the restricted problem
```

in Algorithm 3.1. A compelling property of the cutting plane algorithm, when combined with the ellipsoid algorithm for solving linear programs, is that even if the original problem has an exponential number of constraints it is possible to solve it in polynomial time as long as it is possible to find a violating constraint, if there is one, in polynomial time. An algorithm that takes a point and finds such constraints is often referred to as a violation oracle.

It is often important to talk about the amount a given constraint is violated. If a constraint is

$$A_i^T x \leq b_i, \tag{3.44}$$

its violation is defined as

$$A_i^T x - b_i, \tag{3.45}$$

and if the violation is positive the constraint is said to be violated. Many cutting plane algorithms assume the existence of an oracle which returns the most violated constraint

Algorithm 3.2 The column generation algorithm.

```
1: function ColumnGeneration
2:     Initialize and solve the restricted problem
3:     while there is a variable $x$ with positive reduced cost do
4:         Add $x$ to the restricted problem
5:         Solve the restricted problem
```

in the set.

One can, of course, apply the cutting planes method to the dual of a linear program. Then the same polytope argument suggests that if there are more primal variables than constraints there should be a solution in which most such variables are set to zero. Then one can run the *column generation algorithm* as long as one has a *reduced cost oracle*, which finds the primal variable x_i with largest reduced cost,

$$R_i = \lambda^T A_i - c_i, \tag{3.46}$$

which is the amount by which the corresponding dual constraint is being violated. Algorithm 3.2 shows the pseudocode for column generation. Another way of interpreting the reduced cost of a variable is that it is its (negated) coefficient in the Lagrangian of equation (3.39). Hence if the reduced cost is positive it is possible to improve the value of the Lagrangian by allowing that variable to be nonzero. Note that the KKT conditions of stationarity and complimentary slackness on the LP dual imply that all reduced costs of the variables have to be nonpositive.

Note that while the intermediate iterates of the cutting planes method do not satisfy all the primal constraints, they do satisfy all the dual constraints and monotonically improve the dual objective. Likewise, the iterates of column generation maintain primal feasibility, which means that column generation can be stopped early to return approximate solutions, and the optimality gap can be bounded by the sum all positive reduced costs of the variables which haven't been yet considered.

Chapter 4

Structured prediction

In many learning tasks, such as the ones described in chapter 2, one observes an input $o \in \mathcal{O}$ and wants to predict a class label $x \in \mathcal{U}$, where the set $\mathcal{U}$ of valid possible labels can be exponentially large (in the size of o). In this setting normal multiclass classification techniques fail, because most such techniques can't predict a label that has never been seen in the training data and often learn a per-label set of parameters. To generalize multiclass classification to this exponential-sized setting, then, one must assume that the set of labels is somehow structured, and exploit this structure to make learning and classification possible.

Structured prediction problems often occur in natural language processing and computer vision, as in these fields it is often convenient to frame a prediction problem as aggregating, from local information, a globally consistent picture of what the labels should look like.

For example, in part-of-speech tagging, a natural language processing system receives a sentence and has to output an assignment of a part of speech for each word token in it. While it is possible to model this problem as independent classification of each word token, with its label being its part-of-speech, the parts of speech of the tokens surrounding a token contain very relevant information, which can help disambiguate unseen or ambiguous word types. For example, if an unseen word type is between words tagged as adjective and verb it is fair to assume it is a noun, while if it is between a noun and a verb it is fair to assume it is an adjective. Indeed, one of the earliest successful part-of-speech tagging systems, described in Klein and Simmons [34], used such rules to tag unknown words without requiring a dictionary.

There are many different approaches to solving structured prediction problems, but most can be classified by their position in two roughly orthogonal axes: whether search for an output is performed incrementally or exactly, and whether the parameters are estimated to directly optimize structured accuracy. If the parameters are not directly

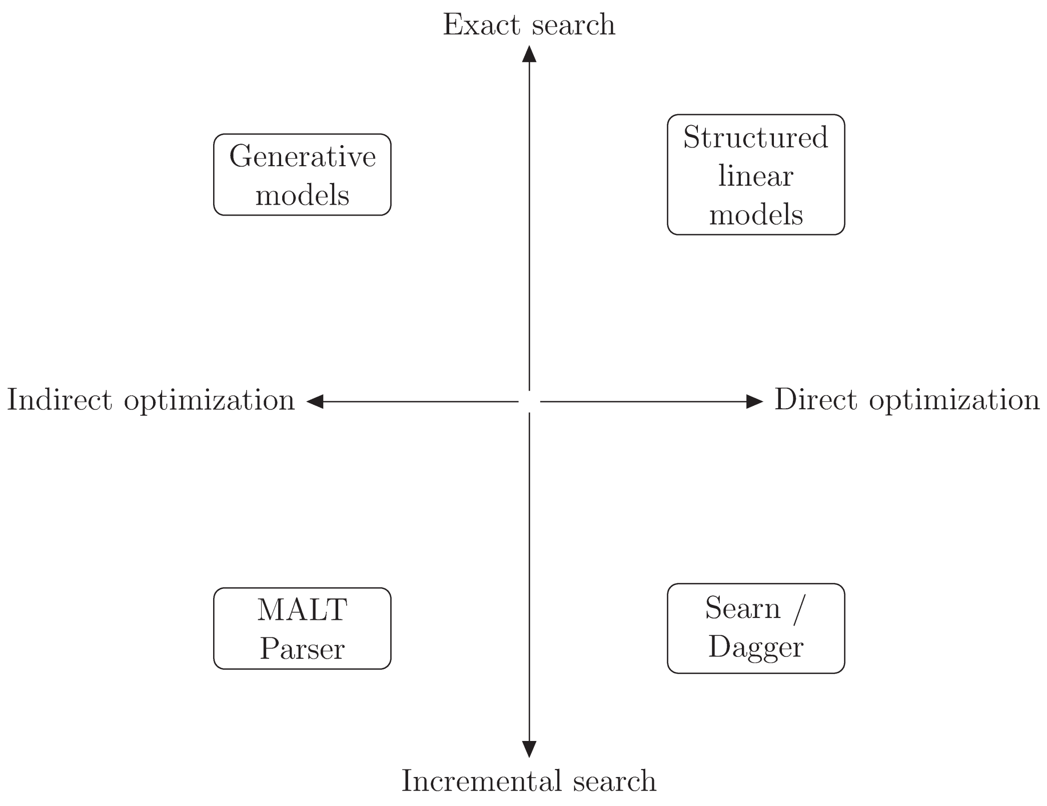


Figure 4.1: Paradigms for structured prediction and some example techniques.

optimized for accuracy they can be maximizing generative log-likelihood, or accuracy of a local classifier which is then applied repeatedly. Models with exact search generally have a higher accuracy than models with approximate search, at the expense of speed, and likewise models with direct optimization tend to generalize better to unseen examples.

On the incremental search with direct training corner we have algorithms such as Searn[14] or Dagger [81], which treat structured prediction as a reinforcement learning problem, in which the predictor makes a sequence of decisions, each pruning the search space, until one single output is left, which is its prediction. Training is done by reducing the problem to classification and carefully ensuring that the classifier's distribution over training examples is close to the distribution over test examples that will be induced by the search process.

On the incremental search with indirect training corner we have transition-based dependency parsers (such as the MALT parser [68]), and the ClearNLP tagger [8], which perform search at test time using a classifier that was trained to predict how to make correct local decisions at test time assuming that the decisions it has made so far were correct. Bridging the gap between direct and indirect training we have approaches such as bootstrap [7] or dynamic oracles [24], which can be used to train a Dagger-style model. Another family of approaches that perform approximate search with somewhat direct training is the neural network approaches to structured prediction. The Senna system [11] jointly learns the parameters of a neural network for many prediction tasks, and searches incrementally at test time for the best output. Recursive neural networks [90] learn a composition operator to minimize reconstruction error of phrases, and then use this operator with incremental search to parse sentences.

With exact search and indirect training there are, for example, generative models such as the Collins parser [10] or the Berkeley parser [72], though in practice these parsers use inexact search at test time for speed reasons.

Finally, in this thesis we will mostly concern ourselves with models in the corner where search is performed exactly and training is done to directly optimize test-time accuracy. Examples of this are conditional random field models for part-of-speech tagging [41] and maximum spanning tree parsers [58, 56]. While there are many families of approaches in this corner, in this thesis we'll concern ourselves mostly with structured linear models.

4.1 Structured linear models

Structured linear models [41, 102, 97, 9] are a very popular class of algorithms for structured prediction. The idea unifying structured linear models is to represent each valid label as a binary vector $\mathbf{x}$, each dimension of which represents whether a specific "part" is present or absent in that label. The score of each label is expressed as a dot product $\langle \mathbf{w}, \mathbf{x} \rangle$, and hence it is linear in the parts. Finally, and this is where the structure comes from, not all binary vectors are valid, and the set of valid vectors is denoted as $\mathcal{U}$.

There are many classes of structured linear models, and perhaps the most popular of these is graphical models, which will be described in more detail in section 4.2. A graphical model describes a probability distribution over a given set of random variables by defining "factors", each of which assigns a score to a setting of its neighboring random variables. Each such setting, then, defines a part. The valid settings of the parts vector $\mathbf{x}$ in a graphical model are those where different factors agree about the values of the variables which they both touch.

The main motivation for structured linear models is structured prediction. While in multiclass classification the size of the set of possible labelings is small, in structured prediction it is often exponentially-sized as a function of the observation. Then naively solving the prediction problem

$$\max_{\mathbf{x} \in \mathcal{U}} \mathbf{w}^T \mathbf{x} \tag{4.1}$$

by enumerating all possible outputs is intractable, and one has to restrict oneself to sets where this maximization can be performed efficiently. Similarly, learning one independent score for each possible label is intractable, as most labels—assignments of tag sequences to tokens, or assignments of parse trees to sentences, for example—will never be seen in the training data. By decoupling the scoring of a label into the scoring of its parts, which will hopefully be seen in the training data, and structuring the parts such that finding the best possible label can be done efficiently, structured linear models exploit the structure in the parts to effectively learn predictors for structured prediction problems.

Structured linear models can easily be adapted to handle the case where the set of possible labellings itself varies across different training examples, as long as differently-sized labels share parts. In dependency parsing, for example, a labeling is an assignment of parents to each token in a sentence, so sentences with different sizes will have different sets of possible labelings. Because of the local properties of structured linear models, which can, for example, decompose the score of a parse tree as a sum of the scores of each edge in it, it is still possible to learn and predict using such models.

A general representation of a structured linear model, then, is a function of an observation o, as $(\mathbf{w}, \mathcal{U})$, which specifies the set of parts (and their scores) for this observation as well as the set of valid labelings. Commonly the score of a part is expressed as a dot product between some features of the observation and a learned parameter vector θ, as in

$$\mathbf{w}^T\mathbf{x} = \theta^T\mathbf{f}(o, \mathbf{x}) = \theta^T \sum_i \mathbf{x}_i f_i(o), \tag{4.2}$$

where $f_i(o)$ is a feature vector for part i in training example o, and $\mathbf{f}(o, \mathbf{x})$ is the sum of the feature vectors for all parts active in $\mathbf{x}$. We'll alternate between these notations as convenient.

4.1.1 MAP and Marginal Inference

Structured linear models are used for prediction and for defining probability distributions. Intuitively, a structured linear model is a redundant representation of the possible labels of an observation o, and a way to score each label. If all possible binary vectors were in the set $\mathcal{U}$ then intuitively each part with a positive score would be active when predicting and each part with a negative score wouldn't. The constraints in $\mathcal{U}$ then help incorporate redundant scores of different parts into one cohesive labeling of an observation.

There are then two natural problems with structured linear models. Computing the best valid labeling, or *MAP inference*, and computing the expected labeling, or *marginal inference*.

For both problems, a key quantity involved is the log partition function,

$$\log Z_{t,o}(\theta) = t \log \sum_{\mathbf{x}\in\mathcal{U}} \exp\left(\frac{1}{t}\theta^T\mathbf{f}(o, \mathbf{x})\right), \tag{4.3}$$

where t is a positive real number referred to as the temperature. When omitted, assume that $t = 1$.

Note that the limit of the log partition function as the temperature goes to zero is the maximum possible score of any label $\mathbf{x}$,

$$\lim_{t \to 0} \log Z_{t,o}(\theta) = \max_{\mathbf{x} \in \mathcal{U}} \theta^T \mathbf{f}(o, \mathbf{x}). \tag{4.4}$$

This means that a subgradient of the partition function at zero temperature is a vector $\mathbf{f}(o, \mathbf{x}^*)$ with maximum compatibility. Using the chain rule one can see that for $t = 1$ the gradient of the partition function is

$$\nabla \log Z_{1,o}(\theta) = \frac{1}{\sum_{\mathbf{x} \in \mathcal{U}} \exp\left(\theta^T \mathbf{f}(o, \mathbf{x})\right)} \sum_{\mathbf{x} \in \mathcal{U}} \exp\left(\theta^T \mathbf{f}(o, \mathbf{x})\right) \mathbf{f}(o, \mathbf{x}), \tag{4.5}$$

which can be written as an the expectation of the feature vector $\mathbf{f}(o, \mathbf{x})$,

$$\nabla \log Z_{1,o}(\theta) = E_\theta[\mathbf{f}(o, \mathbf{x})], \tag{4.6}$$

in which the probability of each configuration is proportional to the exponential of its compatibility,

$$P_\theta(\mathbf{x}|o) \propto \exp(\theta^T \mathbf{f}(o, \mathbf{x})). \tag{4.7}$$

In this thesis we define the problem of computing $\log Z_{0,o}(\theta)$ and one of its subgradients as *MAP inference* and the problem of computing $\log Z_{1,x}(\theta)$ and one such gradient as *marginal inference*. A way of conceptualizing MAP and marginal inference is that MAP inference finds an $\mathbf{x}^*$ which is a mode of the distribution $P_\theta(\mathbf{x}|o)$ while marginal inference finds the mean $E[\mathbf{x}]$ of that distribution.

The acronym MAP stands for maximum a-posteriori. This is so because historically in structured linear models in which $\theta^T \mathbf{f}(o, \mathbf{x})$ is defined as $\log P(o|\mathbf{x}) + \log P(\mathbf{x})$, which is the logarithm of the product between a prior distribution over structures $\mathbf{x}$ and a likelihood function of observations o given a structure. MAP inference then computes a maximum a-posteriori structure $\mathbf{x}$ according to this prior and likelihood. Models with this factorization in their parameters are called *generative models*, because they can be interpreted as describing a process in which the variables $\mathbf{x}$ and o are generated by a stochastic process.

The problem of MAP inference will be the main focus of this thesis.

4.1.2 Joint inference and dual decomposition

It is often the case that while one does not have an efficient inference method for a given structured linear model, its parts can be partitioned into sets (referred to as submodels) such that inference is tractable in each such set and the global validity constraint can be expressed as linear constraints. MAP inference in such models is often referred to as *joint inference* [79, 84, 82, 20].

In that case the dual decomposition algorithm from section 3.3 can be used to construct a MAP inference algorithm.

The general joint inference problem can be written as

$$\begin{aligned} \mathbf{max}._{\mathbf{x}} \quad & \sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle \\ \mathbf{s.t.} \quad & \forall i \quad \mathbf{x}_i \in \mathcal{U}_i \\ & \sum_i \mathbf{A}_i \mathbf{x}_i = 0 \end{aligned} \tag{4.8}$$

where each $\mathbf{x}_i$ represents the vector of parts of a specific structured linear model and each matrix $\mathbf{A}_i$ projects each $\mathbf{x}_i$ into the constraint space. If the linear constraint did not exist the problem would be easy, and equivalent to independence inference in each submodel.

The dual decomposition algorithm solves this problem by dualizing the linear constraint, forming the Lagrangian

$$\sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle + \boldsymbol{\lambda}^T \left(\sum_i \mathbf{A}_i \mathbf{x}_i \right), \tag{4.9}$$

and regrouping its terms and maximizing over the primal variables to form the dual problem

$$\min_{\boldsymbol{\lambda}} D(\boldsymbol{\lambda}) = \sum_i \max_{\mathbf{x}_i \in \mathcal{U}_i} \left\langle \mathbf{w}_i + \boldsymbol{\lambda}^T \mathbf{A}_i, \mathbf{x}_i \right\rangle. \tag{4.10}$$

This problem is convex, as it is the sum of the supremum of linear functions of $\boldsymbol{\lambda}$, and hence can be solved with any convex optimization technique [5].

Any subgradient of the dual function with respect to $\boldsymbol{\lambda}$ can be written as

$$\partial D(\boldsymbol{\lambda}) = \sum_i \mathbf{A}_i \mathbf{x}_i^*, \tag{4.11}$$

for some $\mathbf{x}_i^*$ which is a maximizer of the reweighted MAP inference problem

$$\max_{\mathbf{x}_i \in \mathcal{U}_i} \left\langle \mathbf{w}_i + \boldsymbol{\lambda}^T \mathbf{A}_i, \mathbf{x}_i \right\rangle. \tag{4.12}$$

Given a subgradient, one can then apply the subgradient method to solve the dual problem. As seen in section 3.2.1, the subgradient method is an iterative algorithm which

Algorithm 4.1 The subgradient method for dual decomposition.

```
1: $\boldsymbol{\lambda}_{sc} \leftarrow \mathbf{0}$
2: while has not converged do
3:     for submodel $i$ do
4:         $\mathbf{x}_i^* \leftarrow \max_{\mathbf{x}_i \in \mathcal{U}_i} \langle \mathbf{w}_i + \boldsymbol{\lambda}^T \mathbf{A}_i, \mathbf{x}_i \rangle$
5:     $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \eta^{(t)} \sum_i \mathbf{A}_i \mathbf{x}_i^*$
```

at each iteration updates the dual variables by subtracting from them the subgradient of the objective function, multiplied by a per-iteration step-size $\eta^{(t)}$:

$$\mathbf{x}_i^{(t)*} = \max_{\mathbf{x}_i \in \mathcal{U}_i} \langle \mathbf{w}_i + \boldsymbol{\lambda}^T \mathbf{A}_i, \mathbf{x}_i \rangle \tag{4.13}$$

$$\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}^{(t-1)} - \eta^{(t)} \sum_i \mathbf{A}_i \mathbf{x}_i^{(t)*} \tag{4.14}$$

As long as the sequence of step sizes $\eta^{(t)}$ satisfies some simple conditions the subgradient method is guaranteed to converge to the optimal solution [65].

Applying the subgradient method to optimizing $D(\boldsymbol{\lambda})$ consists of alternatively doing MAP inference on the subproblems and updating their weights accordingly, until the primal constraint is satisfied, at which point the dual objective is minimized. Algorithm 4.1 shows the pseudocode for joint inference with dual decomposition.

4.1.3 Projection variables and max-marginals

While the representation of a structured linear model in terms of parts is convenient, as it captures the dependencies used during inference and learning, one often wants to refer to a function of a labeling which is not a part itself.

Definition 10. *Given a labeling* $\mathbf{x}$*, a matrix* $\mathbf{A}$ *is said to define a projection variable if, for every valid* $\mathbf{x} \in \mathcal{U}$*,*

$$\mathbf{A}\mathbf{x} = e_j \tag{4.15}$$

for some direction j*.*

That is, a projection variable is a matrix such that, for any valid setting $\mathbf{x}$ of the parts in a labeling, the vector $\mathbf{A}\mathbf{x}$ is zero everywhere but in exactly one coordinate.

For projection variables defined by binary matrices, given a row of $\mathbf{A}$, the parts for which it is nonzero can never be active at the same time in a labeling. Each column of $\mathbf{A}$, conversely, can only have one nonzero element.

An interesting projection variable is, in a pairwise graphical model, whose parts score settings of pairs of variables, the value of a single variable. A matrix $\mathbf{A}$ which defines such a projection variable can then have in each row the value 1 for all parts in a mutually exclusive set for which that variable has that value. Another projection variable is simply whether that variable in a graphical model has one out of a few specific values (for example, whether the part-of-speech tag for a specific token in a sentence defines a noun).

Projection variables are relevant when doing joint inference, as they allow the expression of linear constraints which connect parts in different models, as these parts might split the set of possible labelings in different ways.

Given a projection variable it is often important to talk about the way the model behaves conditioned on it taking one of its possible values.

Definition 11. *The max-marginal* $\mathbf{m}^A$ *of projection variable* $\mathbf{A}$ *is a vector such that*

$$\mathbf{m}^A(i) = \max_{\mathbf{x}\in\mathcal{U}} \langle \mathbf{w}, \mathbf{x} \rangle \quad \mathbf{s.t.}\ \mathbf{A}\mathbf{x} = e_i. \tag{4.16}$$

The max-marginals of projection variable then represent how good each value of the projection variable is.

A key property of max-marginals is that they are linear over changes to the weights in the direction of their projection variable. That is,

$$\mathbf{m}^A_{\mathbf{w}+\alpha^T\mathbf{A}}(i) = \mathbf{m}^A_{\mathbf{w}}(i) + \alpha(i) \tag{4.17}$$

This can be verified by substituting the expression on the left in their definition:

$$\begin{aligned}
\mathbf{m}^A_{w+\alpha^T\mathbf{A}|}(i) &= \max_{\mathbf{x}\in\mathcal{U}} \left\langle \mathbf{w} + \alpha^T\mathbf{A}, \mathbf{x} \right\rangle \quad \mathbf{s.t.}\ \mathbf{A}\mathbf{x} = e_i & (4.18)\\
&= \max_{\mathbf{x}\in\mathcal{U}} \langle \mathbf{w}, \mathbf{x} \rangle + \left\langle \alpha^T\mathbf{A}, \mathbf{x} \right\rangle \quad \mathbf{s.t.}\ \mathbf{A}\mathbf{x} = e_i & (4.19)\\
&= \max_{\mathbf{x}\in\mathcal{U}} \langle \mathbf{w}, \mathbf{x} \rangle + \alpha(i) \quad \mathbf{s.t.}\ \mathbf{A}\mathbf{x} = e_i & (4.20)\\
&= \mathbf{m}^A_{\mathbf{w}}(i) + \alpha(i). & (4.21)
\end{aligned}$$

This property is essential for many usages of max-marginals.

Computing max marginals

It is sometimes more natural to define max-marginals over parts. A max-marginal over parts is a vector $\mathbf{m}$ with one coordinate per part such that in each coordinate its value is the best possible score of a setting which uses that part,

$$\mathbf{m}(i) = \max_{\mathbf{x}\in\mathcal{U}} \langle \mathbf{w}, \mathbf{x} \rangle \quad \mathbf{s.t.}\ \langle e_i, \mathbf{x} \rangle = 1. \tag{4.22}$$

Many known efficient MAP inference algorithms can compute max-marginals over parts (that is, the best possible model score such that a given part is in the solution) at very little additional cost. For example, for tree-structured graphical models one can use the forward-backward algorithm [36], and for hypergraph models one can use its generalization, inside-outside [89]. Even in structured linear models for which efficient algorithms for computing max-marginals over parts aren't known, such as minimum spanning tree for projective dependency parsing, it is always possible to compute them by setting the score of each part, in turn, to ∞ and running normal MAP inference, at an extra cost proportional to the number of parts, if so desired, though probably if this is necessary then it is unlikely that large gains in efficiency will be observed by using methods which depend on max-marginals.

Given max-marginals for parts one can easily compute max-marginals for projection variables. Given the constraints on the matrices $\mathbf{A}$ defining projection variables—that for any valid $\mathbf{x}$ we have that $\mathbf{A}\mathbf{x} = e_j$ for some j, and given that $\mathbf{x}$ is restricted to be a binary vector—it is always possible to express each such matrix as a binary matrix. Then, given max-marginals for all parts in $\mathbf{x}$, one can compute max-marginals for $\mathbf{A}\mathbf{x}$ by, for each row of $\mathbf{A}$, selecting the highest-valued max-marginal of any part for which that row is 1:

$$\mathbf{m}^A(i) = \max_{j:A_{ij}=1} \mathbf{m}(i). \tag{4.23}$$

Note that because each part will only have a value of 1 in a single row of $\mathbf{A}$, computing the max-marginals of arbitrary projection variables has linear complexity on the number of parts.

4.1.4 Block coordinate descent for dual decomposition

One disadvantage of the subgradient method for optimizing the dual decomposition objective from section 4.1.2 is that a step size schedule has to be selected in hindsight, the speed of convergence depends strongly on the step size schedule selected, and no step-size schedule is uniformly good. In our experiments we found variations of more than one order of magnitude of performance across different subgradient schedules, and often there was no single schedule which performed among the best in hindsight for all problems. Intuitively it should be possible to adjust the step size of a subgradient update locally if one has access to some information about how "confident" each submodel is in its answer.

In this section we will look at the update rule of MPLP [23], an inference algorithm for graphical models which reduces the problem of MAP inference to joint inference where each clique is a separate model and each model is constrained to assign the same value to overlapping variables, and optimizes this dual decomposition objective using block coordinate descent.

Block coordinate descent methods for optimization work by selecting a subset of the coordinates in the problem's vector space and exactly minimizing the objective with respect to those coordinates, leaving the others fixed. To apply block coordinate descent to the dual decomposition objective one needs to be able to exactly minimize it with respect to some coordinates, and to enable that we need to make more assumptions about the constraint structure of the problem.

First, instead of a single vector constraint $\sum_i \mathbf{A}_i \mathbf{x}_i = 0$ we will break this down into many constraints over pairs of models, which enforce equality:

$$\begin{aligned} \mathbf{max.}_{\mathbf{x}} \quad & \sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle \\ \mathbf{s.t.} \quad & \forall i \quad \mathbf{x}_i \in \mathcal{U}_i \\ & \forall (\mathbf{A}_{c_1}, \mathbf{A}_{c_2}, c_1, c_2) \in \mathcal{C} \quad \mathbf{A}_{c_1}\mathbf{x}_{c_1} = \mathbf{A}_{c_2}\mathbf{x}_{c_2} \end{aligned} \tag{4.24}$$

where $\mathcal{C}$ is a set of constraints, each constraint being represented by two projection matrices and A_{c_1} and A_{c_2} and two indices into models c_1 and c_2.

In a graphical model in the normal representation (where each factor defines a part for each settings of its neighboring variables) projection variables can map full labelings of the graph to the settings of individual variables in the graphical model. In general they can refer to any set of mutually exclusive states of a structured linear model, such as the dependency parents of a token, whether a certain set of superpixels is marked as a segment, etc.

Since the max-marginals of a projection variable represent the best possible score achievable by each value of the variable, if these max-marginals have a unique maximizer that will be the value the variable is set to in any optimal solution. With this property and linearity, to satisfy a constraint of the form

$$\mathbf{A}_{c_1}\mathbf{x}_{c_1} = \mathbf{A}_{c_2}\mathbf{x}_{c_2}, \tag{4.25}$$

which arises by requiring that 0 is in the subgradient of the dual objective for $\boldsymbol{\lambda}_c$, all that is necessary is to set the dual variables $\boldsymbol{\lambda}_c$ such that there exists a coordinate i which is the maximizer of both readjusted max-marginals; that is, for all j,

$$m^{\mathbf{A}_{c_1}}(i) + \boldsymbol{\lambda}_c(i) \geq m^{\mathbf{A}_{c_1}}(j) + \boldsymbol{\lambda}_c(j) \tag{4.26}$$

$$m^{\mathbf{A}_{c_2}}(i) - \boldsymbol{\lambda}_c(i) \geq m^{\mathbf{A}_{c_2}}(j) - \boldsymbol{\lambda}_c(j), \tag{4.27}$$

because of the linearity of max-marginals discussed in section 4.1.3.

An easy way to ensure that there exists a single coordinate which maximizes both max-marginals is to set $\boldsymbol{\lambda}_c$ such that the max-marginals of both projection variables are equal in all coordinates, that is

$$\mathbf{m}^{\mathbf{A}_{c_1}} + \boldsymbol{\lambda}_c = \mathbf{m}^{\mathbf{A}_{c_2}} - \boldsymbol{\lambda}_c. \tag{4.28}$$

Algorithm 4.2 MPLP for solving the dual decomposition objective.

```
1: $\boldsymbol{\lambda} \leftarrow \mathbf{0}$
2: while has not converged do
3:     for equality constraint $c$ do
4:         $\mathbf{m}^{\mathbf{A}_{c_1}} \leftarrow \text{MaxMargs}\left(\mathbf{w}_{c_1} + \sum_{c':c'_1=c_1} \boldsymbol{\lambda}_{c'}^T \mathbf{A}_{c'_1} - \sum_{c':c'_2=c_1} \boldsymbol{\lambda}_{c'}^T \mathbf{A}_{c'_2}\right)$
5:         $\mathbf{m}^{\mathbf{A}_{c_2}} \leftarrow \text{MaxMargs}\left(\mathbf{w}_{c_2} + \sum_{c':c'_1=c_2} \boldsymbol{\lambda}_{c'}^T \mathbf{A}_{c'_1} - \sum_{c':c'_2=c_2} \boldsymbol{\lambda}_{c'}^T \mathbf{A}_{c'_2}\right)$
6:         $\boldsymbol{\lambda}_c \leftarrow \frac{1}{2}\left(\mathbf{m}^{\mathbf{A}_{c_1}} - \mathbf{m}^{\mathbf{A}_{c_2}}\right)$
```

Solving the above for $\boldsymbol{\lambda}_c$ leads to the MPLP updates,

$$\boldsymbol{\lambda}_c = \frac{1}{2}(\mathbf{m}^{\mathbf{A}_{c_2}} - \mathbf{m}^{\mathbf{A}_{c_1}}). \tag{4.29}$$

Algorithm 4.2 shows how to apply MPLP to solve the dual decomposition objective when max-marginals are available. It consists of alternately computing max-marginals and updating the dual variables according to equation (4.29).

Previous research has shown that while block coordinate descent methods are not guaranteed to converge for non-strongly-convex non-smooth optimization problems MPLP often does converge when optimizing the dual decomposition objective, and often does so faster and to better solutions than the subgradient method [92]. There are many alternative convergent block coordinate descent algorithms which optimize dual functions similar to dual decomposition's, and with better convergence properties [61, 86, 77]. There are many other convergent relaxation-based message-passing algorithms for graphical models with similar structure [94, 105].

4.1.5 Estimating the parameters of structured linear models

While a structured linear model coupled with either MAP or marginal inference is a recipe for going from an observation o to a high-scoring label $\mathbf{x}$ given a parameter vector θ, by scoring each part according to equation (4.1), it is not obvious a priori what would be a good value for θ.

In the standard learning setting, one is given *training data*, a set of samples $(o_i, \mathbf{x}_i) \sim D$ from some data distribution D and one wants to estimate a parameter vector θ such that when given new *test data* o_j also sampled from D the structured linear model will return the appropriate $\mathbf{x}_j$.

There are many forms of parameter estimation in structured linear models. Since marginal inference defines a likelihood function $P_{\mathbf{w}}(\mathbf{x}|o)$ it is natural to maximize the likelihood of the true labels when estimating the parameters [41]. Similarly, since MAP inference can be seen as returning the vector $\mathbf{x}^*$ which lies closest to a given corner of instance space, it is possible to define a reasonable notion of a margin between correct and incorrect answers and generalize the perceptron [9] and support vector machine [102] algorithms to apply to structured linear models. All these algorithms can be seen as optimizing some regularized convex loss functions of the given $(o_i, \mathbf{x}_i)$ pairs.

Regularized optimization problems look like

$$\mathbf{min.}_\theta \quad \sum_i \ell(o_i, \mathbf{x}_i^*, \theta) + \|\theta\| \tag{4.30}$$

for some norm-like function $\|\cdot\|$, which is usually either the 2-norm squared $\|\theta\|_2^2$ or the 1-norm $\|\theta\|_1$, where the latter is preferred when one desires *sparsity*, due to its property of producing minimizers θ^* which have most coordinates identically set to zero [98]. There are known results in learning theory that state that solutions to such regularized optimization problems should generalize to unseen examples [103, 66, 45].

A simple loss function is known as the negative log-likelihood of the true output variables $\mathbf{x}^*$ given the observed input o and the current parameters, and is defined as

$$\ell_t(o, \mathbf{x}^*, \theta) = \log Z_{t,o}(\theta) - \theta^T \mathbf{f}(o, \mathbf{x}^*). \tag{4.31}$$

Note that the gradient of this loss with respect to θ is just the difference between observed and expected features f under the model,

$$\nabla \ell_1(o, \mathbf{x}^*, \theta) = E_\theta[\mathbf{f}(o, \mathbf{x})] - \mathbf{f}(o, \mathbf{x}^*), \tag{4.32}$$

and a subgradient for the zero-temperature case is the difference between one maximizing feature vector and the feature vector of the true label,

$$\nabla \ell_0(o, \mathbf{x}^*, \theta) = \mathbf{f}(o, \arg\max_{\mathbf{x}} \theta^T \mathbf{f}(o, \mathbf{x})) - \mathbf{f}(o, \mathbf{x}^*). \tag{4.33}$$

Using stochastic gradient descent using the above gradient is the structured perceptron algorithm [9].

Note that at the unregularized optimum of the above-specified loss functions the sum of the gradients of all training examples is zero. This means that, for each feature, the model will predict that it occurrs in the same fraction of parts as it occurrs in the true data. Hence structured linear models make *feature engineering* easy: if one can find features which are consistently over-represented or under-represented in the parts output by a model, adding them to the optimization problem is expected to fix this bias, leading to the common process where an engineer looks at the mispredictions of a model, hypothesizes features which correlate with the errors, and adds them to the model.

Both the above loss function gradients say that essentially the same loss is incurred whenever a wrong $\mathbf{x}$ is predicted instead of $\mathbf{x}^*$. This assumption, however, might be too restrictive and unrealistic; in part-of-speech tagging, for example, a wrong output that is incorrect in just a single token is better than a wrong output that is incorrect on all tokens. If you can define the loss of predicting $\mathbf{x}$ when $\mathbf{x}^*$ is the ground truth as a linear function $w_{\mathbf{x}^*}^T \mathbf{f}(o, \mathbf{x})$, then you can define the structured SVM loss [102]

$$\ell_{svm}(o, \mathbf{x}^*, \theta) = \max_{\mathbf{x}} \left\{ (\theta + w_{\mathbf{x}^*})^T \mathbf{f}(o, \mathbf{x}) \right\} - \mathbf{f}(o, \mathbf{x}^*). \tag{4.34}$$

A subgradient of this loss function then is just the difference between one maximizing feature vector and the ground truth feature vector,

$$\nabla \ell_{svm}(o, \mathbf{x}^*, \theta) = \mathbf{f}(o, \arg\max_{\mathbf{x}} (\theta + w_{\mathbf{x}^*})^T \mathbf{f}(o, \mathbf{x})) - \mathbf{f}(o, \mathbf{x}^*). \tag{4.35}$$

While it is possible to define an equivalent marginal version of the structured SVM loss [22] as the difference between $\log Z_{1,x}(\theta + w_{\mathbf{x}^*})$ and the score of the true label, it hasn't been frequently used in practice. An attractive property of the structured SVM loss is that upper bounds the loss $w_{\mathbf{x}^*}^T \mathbf{f}(o, \mathbf{x})$ for all training examples $\mathbf{x}_i$, so it is possible to transform optimization guarantees into the generalization guarantees for w on test data.

We will consider two different loss functions $w_{\mathbf{x}^*}$ in this thesis. One, the 0/1 loss, is defined such that

$$w_{\mathbf{x}^*}^T \mathbf{f}(o, \mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} = \mathbf{x}^* \\ \\ 1 & \text{otherwise} \end{cases} \tag{4.36}$$

While this in general does not decompose over the representation of parts implied by the structured linear model at hand, and hence loss-augmented inference can be hard, its gradient can still be computed if one can find the second-best solution to the MAP inference problem. Another common loss is the Hamming loss, which is defined as

$$w_{\mathbf{x}^*}^T \mathbf{f}(o, \mathbf{x}) = \sum_i \max(0, \mathbf{x}_i - \mathbf{x}_i^*). \tag{4.37}$$

Hence it counts the number of parts on which $\mathbf{x}$ and $\mathbf{x}^*$ differ.

Finally, there is also the SampleRank [109] loss function, which for every two valid configurations $\mathbf{x}$ and $\mathbf{x}'$ such that some downstream objective assigns a higher score to $\mathbf{x}$ than to $\mathbf{x}'$, penalizes the model by an amount of margin violation between $\mathbf{x}$ and $\mathbf{x}'$,

$$\ell_{\text{SampleRank},\mathbf{x}^*}(o, \theta) = \max(0, (\theta + w_{\mathbf{x}^*})^T (\mathbf{f}(o, \mathbf{x}') - \mathbf{f}(o, \mathbf{x})) + 1). \tag{4.38}$$

Pairs of $\mathbf{x}, \mathbf{x}'$ configurations are commonly found by stochastic random search of settings that are somewhat close to a ground truth setting, and it is possible to give some guarantees of convergence for this algorithm, as it strictly generalizes the structured SVM loss.

4.1.6 Optimality Theory

There are many similarities between structured linear models, as presented here, and optimality theory, a family of linguistic models of phonetics and some other parts of linguistics [74].

Optimality Theory (OT) is a way of modeling the linguistic process by which a latent universal structure gets transformed into the observed surface structure which is language-specific. OT models the transformative process not as a sequence of transformations but as a selection of one optimal candidate across many possibilities. This allows a linguist to represent each language's process not as a sequence of transformations but as a set of preferences over universal constraints and the selection of the surface structure which violates less of these constraints.

Formally, the process of OT is as follows. Given a latent structure, a universal process called GEN generates a (possibly infinite, or very large) set of candidate surface representations. Then each candidate is evaluated according to a universal set CON of constraints, most of which are expected to be violated in most candidate representations. Finally, a language-specific process called EVAL ranks the constraints, and selects the surface representation which violates less highly-ranked constraints. A simplified version of linguistic work then is to discover universal constraints in CON and to discover their ranks in individual languages by finding example structures which can elicit this difference. The process is rather tricky and cannot be done in isolation: given constraints A and B and two possible surface structures x which violates A and y which violates B, observing that one given language prefers x over Y might suggest that in this language eval ranks B as more important than A, but if there is a third higher-ranked constraint C which y violates but x doesn't this might also explain the phenomenon.

It is easy to make parallels between structured linear models and OT. The set of valid part assignments $\mathcal{U}$ is a natural analogue to GEN, and likewise each part is analogous to each constraint in CON. The main difference, then, is that instead of eval selecting the output $\mathbf{x}$ which violates the least number of higher-ranked constraints a structured linear model defines a linear notion of compatibility between inputs and outputs via the parameter vector θ. This means that it is possible, given certain parameters θ, for many lower-ranked constraints to dominate a higher-ranking one. On the other hand, the scores can be used to define a probability distribution over outputs, in which the difference between the scores of two output structures corresponds to the logarithm of the ratio of their frequencies, allowing the model to explain some amount of individual variation. The most important difference, however, is that while eliciting the language-specific rankings in EVAL is a hard and subjective problem, estimating the parameter vector θ of a structured linear model is a simple convex optimization problem assuming one has a list of observed $(o_i, \mathbf{x}_i)$ input-output pairs.

Similarly, much of the criticism of OT and structured linear models is the same: while the formalism is rich, it leaves the computational problem of selecting the best constraint unspecified, and without making assumptions about the set $\mathcal{U}$ and the feature representation $\mathbf{f}(o, \mathbf{x})$ it does not seem possible for efficient inference to be done in general.

For further discussion of OT we refer the reader to McCarthy [55].

4.2 Graphical models

Graphical models are perhaps the most commonly used structured linear models. Here we'll restrict our presentation to discrete Markov random fields and conditional random fields from the perspective of factor graphs. We will ignore considerations of conditional independence, d-separation, Bayesian networks, and others that were foundational in studies of graphical models, and for a more complete picture we refer the reader to Koller and Friedman [36] or Wainwright and Jordan [106].

A graphical model is a structured linear model defined by two elements: a set of variables and a set of factors. Each output $\mathbf{x}$ is a set of assignments to the variables and each factor defines a set of parts, one for each possible setting of its neighboring variables.

In a linear-chain model for part-of-speech tagging [41], for example, each variable is a set of part-of-speech assignments to each observed word token i. A factor is defined neighboring each individual token's label, and also each pair of adjacent labels in a sentence. These factors' feature vectors then can compute counts of the form "'part-of-speech p occurred attached to a word token of type w", or "tokens i and $i + 1$ had parts-of-speech p and p' assigned to them".

Since each factor depends only on a subset of the output variables it is possible to draw a bipartite graph in which variables and factors are nodes and there is an edge between a variable and a factor if that factor depends on the value of that variable. Note that it's always possible to represent two factors that touch the same variables as a single factor without loss of generality. This graph is called a factor graph, and figure 4.2 shows an example of a factor graph.

A graphical model whose factors touch at most two variables is said to be pairwise, and when talking about pairwise graphical models we'll refer to factors as edges, as each factor will connect two variables in the factor graph.

We can write the score of a setting $\mathbf{x}$ of the variables in a pairwise graphical model, then, as

$$\mathbf{w}^T\mathbf{x} = \sum_{i,j\in E} \sum_{X_i,X_j} \mu_{ij}(X_i, X_j) w_{ij}(X_i, X_j) + \sum_i \sum_{X_i} \mu_i(X_i) w_i(X_i), \tag{4.39}$$

where the first sum is over all pairs of variables i, j which are connected by one factor, the second sum is over all values X_i, X_j which the variables i and j can take, $w_{ij}(X_i, X_j)$ is

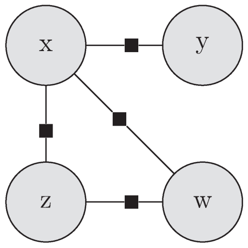


Figure 4.2: A factor graph. Variables are named and represented as circles, factors are represented as squares.

the score of the part corresponding to the factor between variables i and j when they are set to X_i and X_j, and likewise $\mu_i(X_i)$ and $w_i(X_i)$ represent the factors that touch only variable i.

So one can write the MAP score of this graphical model as

$$\log Z_{0,o}(\theta) = \max_{X} \sum_{i,j \in E} \sum_{X_i, X_j} \mu_{ij}(X_i, X_j)\theta_{ij}(X_i, X_j) + \sum_i \sum_{X_i} \mu_i(X_i)\theta_i(X_i). \tag{4.40}$$

4.2.1 MAP inference in acyclic pairwise factor graphs

The key property of graphical models is that if the factor graph is acyclic there are polynomial-time dynamic programming algorithms for MAP and marginal inference, which work by what is called variable elimination.

Variable elimination is a process which, when applied to an acyclic factor graph, returns a factor graph with one variable removed with the property that its $\log Z_{t,o}(\mathbf{w})$ is unchanged. We'll present here the variable elimination algorithm for MAP inference (that is, when $t = 0$), but the algorithm for marginal inference is the same, with basic operations taken from a different ring [2]. We'll assume that the factor graph is connected and that all factors touch at most two variables, though it is easy to generalize this procedure for when these conditions do not apply.

Let i' be the index of a variable that has only one other neighboring variable (let's call it j'), then we can write the MAP score as

$$\begin{aligned}\log Z_{0,o}(\theta) = \max_{X, X_{i'}} \Bigg\{ &\sum_{i,j \in E} \sum_{X_i, X_j} \mu_{ij}(X_i, X_j) w_{ij}(X_i, X_j) + \sum_i \sum_{X_i} \mu_i(X_i) w_i(X_i) \\ &+ \sum_{X_{i'}, X_{j'}} \mu_{i'j'}(X_{i'}, X_{j'}) w_{i'j'}(X_{i'}, X_{j'}) + \sum_{X_{i'}} \mu_{i'}(X_{i'}) w_{i'}(X_{i'}) \Bigg\}\end{aligned} \tag{4.41}$$

Algorithm 4.3 The MAP variable elimination algorithm.

```
1: function MAP($\theta, x$)
2:     for Variable $x_i$ in topological order do
3:         for Value $X_p$ which parent $x_p$ can take do
4:             $\alpha_i(X_p) \leftarrow \max_{X_i} \sum_{x_j \text{child of} x_i} \alpha_j(X_i) + \theta_{pi}(X_p, X_i)$
```

Algorithm 4.4 The variable elimination algorithm for computing max-marginals.

```
1: function MAP($\theta, x$)
2:     for Variable $x_i$ in topological order do
3:         for Value $X_p$ which parent $x_p$ can take do
4:             $\alpha_i(X_p) \leftarrow \max_{X_i} \sum_{x_j \text{child of} x_i} \alpha_j(X_i) + \theta_{pi}(X_p, X_i)$
5:     for Variable $x_i$ in reverse topological order do
6:         for Child $x_c$ do
7:             for Value $X_c$ which $x_c$ can take do
8:                 $\beta_i^c(X_c) \leftarrow \max_{X_i} \beta_i^p(X_i) + \theta_{ic}(X_i, X_c) + \sum_{siblingc'} \alpha_{c'}(X_i)$
```

and then analytically "maximize out" the variable i' by folding its potentials into the local potentials of its neighboring variable j'. That is, define

$$\mathbf{w}'_{j'}(X_{j'}) = w'_{j'}(X_{j'}) + \max_{X_{i'}}\{w_{i'}(X_{i'}) + w_{i'j'}(X_{i'}, X_{j'})\}, \tag{4.42}$$

and $w'_i(X_i) = w_i(X_i)$ otherwise. Let's refer to the right-hand part of the above equation as $\alpha_{i'j'}(X_{j'})$. We can then say that

$$\log Z_{0,o}(\mathbf{w}) = \log Z'_{0,o}(\mathbf{w}') \tag{4.43}$$

where Z' is computed over a model without variable i'.

It is easy to see that iterating this procedure will eliminate all variables, leaving us with the MAP score of the model. Algorithm 4.3 shows how this procedure defines a dynamic programming algorithm.

This algorithm is referred to as a *message-passing algorithm* because it can be represented by each variable in the graph sending a message to its parent variable containing the best possible score, for each setting of the parent variable, of the subtree of which the variable is the root.

Variable elimination can also be used to get max-marginals. Just note that a byproduct of the variable elimination messages is the max-marginals of the "root" node. Because the graph is acyclic at a cost identical to the cost of eliminating all variables it is possible to compute what would the variable elimination algorithm look like assuming each variable was the "root", by continuing the variable elimination process eliminating all variables on "the other side". This can be done by defining two types of messages: up messages, which go from leaves to the root, and down messages, which go from the root to the leaf. The up messages are those from normal variable elimination, while the down messages sent by a parent variable to each child variable are the sum of its parent's down message plus its score plus the up messages of the siblings of that variable. Algorithm 4.4 shows the variable elimination algorithm for computing max-marginals, where the up messages to variable x_i are denoted as $\alpha_i(X_i)$ and the down messages from variable x_i to child variable x_c are denoted as $\beta_i^c(X_c)$. The max-marginals of a variable x_i then are the sum of the up message of its children with the down message of its parent,

$$\mathbf{m}^{x_i}(X_i) = \beta_p^i(X_i) + \sum_{\text{child } x_c} \alpha_c(X_i). \tag{4.44}$$

4.2.2 MAP inference in general discrete graphical models

In the general case MAP inference for graphical models is known to be NP-complete [36]. For discrete pairwise models, the MAP inference problem can be naturally written as the following integer linear program,

$$\begin{aligned}
\mathbf{max}. \quad & \sum_{i,j \in E} \sum_{X_i, X_j} \mu_{ij}(X_i, X_j) w_{ij}(X_i, X_j) \\
\mathbf{s.t}. \quad & \sum_{X_i, X_j} \mu_{ij}(X_i, X_j) = 1 \\
& \sum_{X_i} \mu_{ij}(X_i, X_j) = \sum_{X_k} \mu_{jk}(X_j, X_k) \\
& \mu_{ij}(X_i, X_j) \in \{0, 1\}
\end{aligned} \tag{4.45}$$

in which the local potentials $w_i(Y_i)$ have been arbitrarily folded into the pairwise potentials for notational convenience. Each binary variable in the above ILP represents whether each part in the structured linear model is active in the solution, and it is easy to see that the above mentioned constraints are sufficient to specify exactly the set of vectors which can be obtained from valid settings of the variables in the graphical model.

4.2.3 The LP relaxation

Since the integer linear program formulation can't be solved efficiently in general, it is natural to look at its linear programming relaxation, obtained by relaxing the constraints that the indicator variables $\mu_{ij}(X_i, X_j)$ should be binary, and allowing them to take any real number between 0 and 1.

The LP relaxation, however, is known to not be tight in general. If the graph has cycles it is possible to have solutions of the LP relaxation which are not solutions to the ILP (a simple example is the three-node binary cycle graph with identical repulsive potentials, for which the LP solution is fractional). This is because the constraints in the relaxation do not perfectly encode the convex hull of all integral vertices of the ILP, which is known as the marginal polytope [106].

One interesting thing about the dual of the LP relaxation is that it provides another way to derive algorithm 4.3. If the LP relaxation is written as

$$\begin{aligned}
\mathbf{max}. \quad & \sum_{i,j \in E} \sum_{X_i, X_j} \mu_{ij}(X_i, X_j) w_{ij}(X_i, X_j) \\
\mathbf{s.t.} \quad & \sum_{X_0, X_1} \mu_{01}(X_0, X_1) = 1 \\
& \sum_{X_i} \mu_{ij}(X_i, X_j) = \sum_{X_k} \mu_{jk}(X_j, X_k) \\
& \mu_{ij}(X_i, X_j) \in [0, 1]
\end{aligned} \tag{4.46}$$

its LP dual is

$$\begin{aligned}
\mathbf{min}. \quad & T \\
\mathbf{s.t.} \quad & \alpha_0(x_0) - T \leq 0 \\
& \sum_{i:(i,j)\in E} \alpha_{ij}(X_i) - \sum_{k:(j,k)\in E} \alpha_{jk}(X_j) \leq w_{ij}(X_i, X_j)
\end{aligned} \tag{4.47}$$

Note that if an ordering is chosen such that each variable i has a unique parent and the model is a tree, each constraint states that

$$\alpha_j(X_j) \geq \sum_{i:(i,j)\in E} \alpha_i(X_i) + w_{ij}(X_i, X_j), \tag{4.48}$$

which is identical to the second term of equation (4.42).

So one can think of variable elimination as a way of solving the LP relaxation of inference in a graphical model by dual coordinate ascent, with a schedule that is guaranteed to converge. Note the similarity between this linear program and the linear program for shortest paths shown in section 3.4.1.

4.2.4 MPLP for inference in graphical models

While it is possible to solve the LP relaxation of MAP inference in graphical models by entering it into an off-the-shelf LP solver such as Gurobi [28] or CPLEX [27], the cost of simply writing out the linear program is often prohibitive, making it desirable to find special-purpose methods for solving this LP relaxation directly.

In this section we will talk about the MPLP algorithm [23], one of many message-passing algorithms that solves the LP relaxation of MAP inference. MPLP works by reducing the problem of MAP inference in a graphical model to joint inference on many independent smaller models (one per factor), using dual decomposition to enforce agreement between these submodels (see section 3.3 and section 4.1.2 for an explanation) and block-coordinate descent to minimize the dual decomposition objective, as seen in section 4.1.4.

We'll describe the MPLP algorithm for a pairwise graphical model but the generalization for higher-order factors is simple.

Let e be a factor connecting variables i and j with score $x_e(x_i, x_j)$. Let's then rewrite the LP relaxation in equation (4.47) by creating an indicator variable for each variable in each factor and constraining all indicator copies for the same variable to be identical as follows

$$\begin{aligned}
\textbf{max.} \quad & \sum_{e,X_i,X_j} \mu_e(X_i, X_j) w_e(X_i, X_j) \\
\textbf{s.t.} \quad & \sum_{X_i} \mu_e(X_i, X_j) = \mu_{e,j}(X_j) \qquad \forall e, X_j \\
& \sum_{X_i,X_j} \mu_e(X_i, X_j) = 1 \qquad \forall e \qquad \mu_{e,i}(X_i) = \mu_{0,i}(X_i) \quad \forall e, i, Y_i
\end{aligned} \tag{4.49}$$

Adding Lagrange multipliers to the last set of constraints we get the following Lagrangian (omitting the other constraints),

$$L(\mu, \lambda) = \sum_{e,X_i,X_j} \mu_e(X_i, X_j) w_e(X_i, X_j) + \sum_{e,i,X_i} \lambda_{e,i,X_i}(\mu_{0,i}(X_i) - \mu_{e,i}(X_i)). \tag{4.50}$$

Reordering the terms and maximizing over the primal variables gives us the following LP dual

$$\begin{aligned}
\textbf{min.} \quad & \sum_e \max_{\mu_e(X_i,X_j)} \mu_e(X_i, X_j) \left\{ w_e(X_i, X_j) - \lambda_{e,i}(X_i) - \lambda_{e,j}(X_j) \right\} \\
& + \sum_i \max_{\mu_{0,i}(X_i)} \mu_{0,i}(X_i) \sum_e \lambda_{e,i}(X_i)
\end{aligned} \tag{4.51}$$

where all the maximizations are implicitly over the simplex Δ.

This is the dual decomposition objective for doing inference independently in each factor and constraining all copies of each variable to be identical. As such it can be solved with the subgradient method but it can be more efficiently solved with block coordinate descent, a method which, even though it does not guarantee convergence, is often faster in practice than the subgradient method. To derive the block coordinate descent method note that a sufficient condition for optimality of this dual objective is that its subgradient is 0, which, for any $\lambda_{e,i}$ means that the maximizer X_i^* of factor e's assignment to variable i is the same as the maximizing assignment of the consensus copy of variable i.

One way to ensure that this will happen for all copies of variable i is to make sure that each factor's max-marginals for variable i are identical. It is easy to see that setting

$$\lambda_{e,i}(X_i) = -\sum_{e' \neq e} \max_{X_j} \theta_{e'}(X_i, X_j) - \lambda_{e',j}(X_j) \tag{4.52}$$

is enough to do so, as it makes all the max-marginals and the consensus variable's max-marginals identical.

Iterating these updates usually converges to the optimum of the LP relaxation more quickly than the subgradient method, as seen in the experiments in the MPLP paper [23] and in Sontag et al's dual decomposition tutorial [92]. They can, however, not converge to the optimal solution of the LP relaxation, as is common in coordinate descent methods for non-smooth functions. There are, however, smoothed versions of MPLP which are guaranteed to converge to the optimal solution [50, 61]

MPLP is not the only message-passing algorithm that approximately solves an objective derived from the LP relaxation of graphical model inference. It is, however, representative, and readers should look at the references in this paragraph for further discussion of these methods, their convergence, and how they relate to each other [16, 50, 105, 37, 94, 86, 77, 61]

4.2.5 Bayesian networks and continuous graphical models

Most of the presentation of graphical models so far has been nonstandard, in that it covers mostly Markov random fields from the perspective of factor graphs, and with discrete variables only. Another form of graphical model is the *Bayesian network*, which is a Markov random field in which each variable has a "parent factor", and the scores the parent factor assigns to each value of the variable, when exponentiated, have to sum to one. When using Bayesian networks, however, commonly the observations o are also represented as variables being part of the same network, and MAP or marginal inference is done only varying the variables in the label $\mathbf{x}$. As Bayesian networks can be represented with factor graphs we don't need to describe learning differently, except by mentioning that when using generative bayesian networks it is often possible to estimate the parameters θ by estimating the frequencies of substructures in the data.

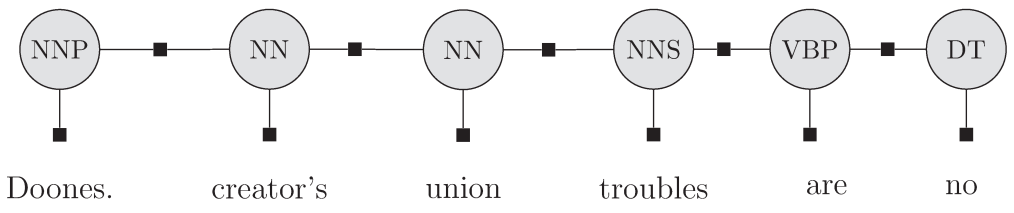


Figure 4.3: The linear chain graphical model used for POS tagging for a fragment of our example sentence.

Many models in the literature have *latent variables* z, which are variables in the model which are neither a part of the observation o nor of the predicted label $\mathbf{x}$. A latent-variable structured linear model assigns scores by computing the dot product $\theta^T \mathbf{f}(o, \mathbf{x}, z)$. Since one is interested in predicting $\mathbf{x}$ from o, the latent variables need to be dealt with somehow, either by summing over all their possible values or by choosing their values which maximize the score of $\mathbf{x}$. Learning in these models is nonconvex, as is the loss functions we discussed all become the difference between two nonlinear convex functions, and there are many approaches to do so in the literature. We refer the reader to Koller and Friedman [36] for a discussion of graphical models with latent variables, and inference and learning in those settings.

Finally, many graphical models in the literature have continuous variables x_i instead of the discrete variables we've restricted ourselves to considering so far. We note that as long as the factors are parametrized as the logarithm of distributions from the exponential family then the model is still a structured linear model, as the natural parametrization is linear, and most of the arguments we make here can be generalized by replacing sums with integrals and taking proper care to ensure measurability of the underlying distributions.

4.2.6 Part-of-speech tagging graphical model

In this section we'll look at a basic way of solving the part of speech tagging problem with a graphical model. As we saw in section 2.1, the most relevant pieces of information about the part of speech tag we need to assign to a token are its word type and the part of speech tags of the tokens around it.

Figure 4.3 shows an example of a graphical model with factors that capture these two sources of information for part of speech tagging.

This form of graphical model is referred to as a linear chain, because it is possible to put the variables in a sequence such that each variable only has factors touching the other variables immediately next to it. The MAP inference algorithm based on variable elimination for linear chain graphical models is known as the Viterbi algorithm [104]. The same linear chain structure is used in other tagging tasks, and even some sequence segmentation tasks, such as noun phrase chunking [87] or named-entity recognition [53].

The specific form of the factors used in part of speech tagging models vary, but most models have pairwise factors whose features are only the values of the two neighboring variables and local factors whose features are cross-products of each variable's value and the following features of the observed sequences

1. the exact word type used at each position;
2. lemmatized, lower-cased, and otherwise canonicalized versions of the word types;
3. character prefixes and suffixes of the word tokens;
4. whether a token is entirely uppercase, is numbers, or is comprised only of punctuation;
5. dictionary lookups for common part of speech tags of the word type;
6. the above features applied at the tokens surrounding any given token, and conjunctions of these.

Moreover, some state of the art models have factors touching triples of adjacent tags, instead of pairs.

4.3 Hypergraph models

While graphical models can be very intuitive to reason about, efficient inference is only possible in special cases with very restricted structure. This means that when a structured linear model has global constraints—like projectivity in dependency parsing—it can be impossible or very difficult to represent efficiently as a graphical model. In this section we will look at a specific generalization of acyclic factor graphs called hypergraph models, grammars, or weighted logic programs [63, 89].

A hypergraph model is a directed hypergraph [21] with scores on the hyperedges. It is represented by a tuple (N, S, t, E), in which N is a set of nodes, $S \subset N$ is a set of nodes named as sources, $t \in N$ is the target node, and E is a set of hyperedge, in which each edge $e \in E$ is a tuple (I_e, o_e, w_e), where $I_e \subset N$ is the set of input nodes of hyperedge e, o_e is its output node, and w_e is its score. All these sets are finite.

When viewed as a structured linear model, the parts in a hypergraph model are the edges which are active in any given hyperpath. An output label $\mathbf{x}$ is a set of hyperedges which forms a valid path, in that the target node is the output node of one hyperedge in the set and for each hyperedge in the set each of its input nodes is either a source node or the output node of exactly one other hyperedge in the set. Similarly to directed graphs, then, an ordering of a set of hyperedges is said to be a topological sort if for any node n all hyperedges with n as output appear before any hyperedge with n as input. If there is a topological sort for a set of hyperedges then the hypergraph is said to be acyclic.

A hypergraph generalizes a directed graph by letting each edge have more than one input. Most concepts carry over from graphs to hypergraphs.

It is easy to see that an acyclic graphical model can be represented as a hypergraph model: create a node for each (variable,value) pair, pick a "root" variable , create score 0 hyperedges connecting all its values to a target node, label all nodes corresponding to values of degree-one variables as sources, and create a hyperedge for each setting of a pairwise factor with the appropriate score. It is also easy to see that many dynamic programming algorithms can be said to represent a hypergraph model: each dynamic programming cell is a node in the hypergraph, and for each dynamic programming hyperedge that says that the value of each cell has to be greater or equal to the sum of the values in other cells plus a constant becomes a hyperedge in the hypergraph, and the property of recursive substructures is just stating that the hypergraph is acyclic. Finally, context-free grammars in Chomsky normal form are easily seen to be hypergraph models, by defining a node for each contiguous span of characters in a sentence and each nonterminal in the grammar, and creating hyperedges for all grammar productions.

The main reason why hypergraph models generalize graphical models for which inference is efficient is that paths in the hypergraph support a richer notion of mutual exclusivity than variables in graphical models. In a graphical model the one restriction on the set of possible assignments is that each variable can take only one value. In general hypergraphs, however, the restriction is that flow is conserved along nodes, which allows for non-local mutual exclusion constraints. For example, a parsing hypergraph can be seen as a binary graphical model with one variable per span of consecutive word tokens in a sentence, whose value is one if that span is a constituent and zero otherwise. To represent the consistency constraints in a graphical model would require factors connecting all variables representing overlapping spans to enforce nestedness, which would make inference by variable elimination intractable. In a hypergraph model, however, by properly adjusting the edges it is easy to ensure that spans are nested.

The most useful property about hypergraph models is that while MAP inference and marginal inference are easily formulated in them, and solvable with dynamic programming algorithms, they have more expressive power than acyclic graphical models. For example, the Eisner algorithm [17, 56] is a way of encoding first-order projective dependency parsing in a hypergraph model, which would not be possible with an acyclic graphical model (though note that each hypergraph model is trivially a graphical model with one variable per hyperedge and one factor touching all such variables that ensures they form a valid path).

4.3.1 MAP inference in hypergraph models

A set of hyperedges P is said to form a hyperpath if the following conditions hold for any any edge $e \in P$,

1. for all input nodes $i \in I_e$, either $i \in S$ or there is an edge $e' \in P$ such that $o_{e'} = i$; and
2. its output o_e is either the target t or there is another edge $e' \in P$ such that $o_e \in I_{e'}$.

Equivalently, these constraints can be framed in terms of the nodes in the hypergraph as follows. For each node n in the hypergraph which is not a source or target, the size of the set of edges leaving the node is identical to the size of the set of edges entering the node. Exactly one node enters the target node, and the source nodes are unconstrained.

The key property of hypergraph paths is that finding the maximum-scoring path can be cast as the following LP, with one indicator variable x_e for each hyperedge e:

$$\begin{aligned} \textbf{max}.\quad & \sum_e x_e w_e \\ \textbf{s.t}.\quad & \sum_{o_e = t} x_e = 1 \\ & \sum_{n \in I_e} x_e - \sum_{o_e = n} x_e = 0 \quad \forall n \notin \{S \cup \{t\}\} \end{aligned} \tag{4.53}$$

Its dual LP then will have one constraint per hyperedge and one variable V_n per hypernode, and can be written as follows

$$\begin{aligned} \textbf{min}.\quad & T \\ \textbf{s.t}.\quad & T - \sum_{n \in I_e} V_n \geq w_e \qquad \forall e : o_e = t \\ & V_{o_e} - \sum_{n \in I_e} V_n \geq w_e \quad \forall e : o_e \neq t \end{aligned} \tag{4.54}$$

It is easy to see that if the hypergraph is acyclic there is an order over the variables V_n such that if n' comes after n then the value of $V_{n'}$ is not used, directly or indirectly, in computing the value of V_n, and hence this LP can be solved with dual coordinate ascent, which in this case is identical to dynamic programming. The time complexity of this algorithm is $O(|E|)$ and the space complexity is $O(|N|)$. Algorithm 4.5 shows the dynamic programming algorithm in pseudocode.

Algorithm 4.5 The algorithm for MAP in a hypergraph.

```
1: function InsideMAP($(N, E)$)
2:     Initialize $V_s = 0$ for source $s$, $V_n = -\infty$ otherwise
3:     for Hyperedge $e$ in topological order do
4:         $V_{o_e} \leftarrow \max\left(V_{o,e}, w_e + \sum_{n' \in I_e} V_{n'}\right)$
5:     return $V$
```

Algorithm 4.6 The inside-outside algorithm for max-marginals in a hypergraph.

```
1: function OutsideMAP($(N, E)$)
2:     $V \leftarrow$ InsideMAP($(N, E)$)
3:     Initialize $O_t = 0$ for the target $t$, $O_n = -\infty$ otherwise
4:     for Hyperedge $e$ in reverse topological order do
5:         for $n \in I_e$ do
6:             $O_n \leftarrow \max\left(O_n, O_{o_e} + w_e + \sum_{n' \in I_e, n' \neq n} V_{n'}\right)$
7:     return $O$, $V$
```

This algorithm can be adapted for marginal inference by replacing all sums with products and all maximizations with sums, and returning the logarithm of T instead of T.

To compute max marginals, a simple extension of algorithm 4.5 can be used. It is commonly referred to as the inside-outside algorithm [42], originally proposed as a way of performing the E step in the EM algorithm for context-free grammar parsing. It works by keeping an auxiliary table, O_n such that $V_n + O_n$ is the score of the best path that includes node n. Algorithm 4.6 shows the inside-outside algorithm. It works by, after running algorithm 4.5, iterating over the edges in the reverse order while ensuring that, for each node n, for each edge e such that $n \in I_e$, the outside score of n is bigger than the sum of the inside scores of its neighboring inputs plus the outside score of the target, or

$$O_n = \max_{e : n \in I_e} O_t + \sum_{n' \in I_e \wedge n' \neq n} V_{n'} + w_e. \tag{4.55}$$

This algorithm, algorithm 4.6, is equivalent to the max-marginal algorithm for graphical models, and it can be seen as efficiently considering all possible “rotations” of each edge in the hypergraph. Its complexity is $O(Ed^2)$, where E is the number of edges in the hypergraph and d is the maximum degree of an edge.

Note that because it is possible to compute max-marginals in hypergraph models it is possible to apply the MPLP algorithm described in section 4.2.4 to inference in problems for which individual factors can be expressed as hypergraph models.

4.3.2 Projective dependency parsing as a hypergraph model

As seen in section 2.2, dependency parsing is that task of detecting, for each word token in a sentence, which other token heads the phrase containing it. For an example of a dependency parse see Figure 2.4. As a requirement, the head relationships have to be hierarchical; that is, each token has a single head and the directed graph induced by this relationship is acyclic. Additionally, one is sometimes interested in adding the constraint that the trees have to be projective; that is that when drawing the edges above the word tokens two lines cannot cross or, equivalently, that the set of all tokens that are directly or indirectly headed by a given token is a contiguous span of the sentence.

Projectivity means that it is possible to tackle the parsing problem with dynamic programming. In this section we will look at the first-order model of Eisner and Satta [17] and one third-order model by Koo and Collins [38].

Both these algorithms reduce the problem of projective dependency parsing to parsing with a context-free grammar that can produce all dependency trees and is easily represented as a hypergraph.

The key idea behind this specific grammar which makes it more efficient than the naive alternative of explicitly keeping track of all possible head tokens for all contiguous sub spans of the sentence is to, instead, construct the left and the right children of each token, and "gluing" these pieces together only when constructing the left and right children of its head.

The grammar includes four types of half-spans: left- and right-headed complete and incomplete span. A left-headed span is a pair of indices such that all tokens in that span are directly or indirectly headed by its leftmost token (and likewise for a right-headed span). A left-headed complete span is a pair of indices such that the rightmost token in the span cannot have any more dependents, while a left-headed incomplete span is such that the rightmost token can still receive more dependents.

Spans are constructed as follows: a left-headed incomplete span can be combined with a left-headed complete span immediately to its right to form a left-headed complete span. In doing so the rightmost word token of the incomplete span acquires all its dependents. Likewise, a left-headed complete span can be combined with a right-headed complete span immediately to its right to form either a left-headed or a right-headed incomplete span, by making the head token of either complete span take the head token of the other span as a modifier. Table 4.1 shows all the edges in the Eisner algorithm's hypergraph.

It is easy to see that these rules can be combined in a hypergraph: each span is a node and each production rule is a hyperedge. Then MAP and max-marginals can be computed using the generic algorithms in algorithm 4.5 and 4.6. As the hypergraph model allows one to assign scores to each rule in the grammar one can use this model to assign a score to each possible dependency edge in the sentence by assigning this score to all hyperedges

Table 4.1: The edges used in the Eisner algorithm's hypergraph. LC(h,m) is a left-headed complete span with head h and modifier m, LI are left-headed incomplete spans, etc.

Target	**Inputs**	**Score**	**Constraints**
LC(h, m)	LI(h, s), LC(s, m)	0	$h < s \leq m$
RC(m, h)	RC(m, s), RI(s, h)	0	$m \leq s < h$
LI(h, m)	LC(h, s), RC(s+1, m)	ScoreEdge(h,m)	$h \leq s < m$
RI(h, m)	LC(h, s), RC(s+1, m)	ScoreEdge(m,h)	$h \leq s < m$

whose outputs are the incomplete spans in which those dependency edges are added and letting all other hyperedges have a score of zero. For this reason, because scores depend on single dependency edges, this model is referred to as a first-order model. It is easy to see that there are $O(n^2)$ spans, and each span has $O(n)$ incoming hyperedges, so the total complexity of this algorithm is cubic on the length of the sentence.

This general idea can be extended to higher-order models by enriching the notion of spans. Now we'll present Model 1 from Koo and Collins [38], which is a third-order model that allows scoring of some pairs of neighboring edges in a dependency tree as well as triples of edges of the form $(p- > n), (n- > s_0), (n- > s_1)$, where s_0 and s_1 are both adjacent left children or right children of the node n.

This is accomplished by first creating the additional notion of a "sibling span", which is the connection of a left and a right complete spans which will be siblings (instead of one heading the other as is the case in incomplete spans). Incomplete spans are grown then by a word acquiring its first dependency as in the Eisner algorithm and subsequently adding sibling spans to the modifier side. This allows the model to score pairs of dependency edges which have the same parent word. The second difference is augmenting each span with a "grandparent" index which lies outside it, and only gluing spans for which the grandparent index matches. Table 4.2 shows the grammar used by the Model 1 algorithm. Similarly to the Eisner algorithm, it is easy to see that there are $O(n^3)$ nodes in the hypergraph and each node has $O(n)$ in-degree, so total time parsing complexity is $O(n^4)$ on the length of the sentence.

Table 4.2: The edges in the Koo et al Model 1's hypergraph. LC(g, h, m) is a left-complete span with grandparent g, head h, and modifier m. SI are sibling spans. The grandparent index g is always constrained to lie outside the span.

Target	**Inputs**	**Score**	**Constraints**
LC(g, h, m)	LI(g, h, s), LC(h, s, m)	0	$h < s \leq m$
RC(g, m, h)	RC(h, m, s), RI(g, s, h)	0	$m \leq s < h$
LI(g, h, m)	LI(g, h, s), SI(h, s, m)	TRIPLE(g, h, s, m)	$h < s \leq m$
LI(g, h, m)	LC(g, h, h), RC(h, h+1, m)	TRIPLE(g, h, -1, m)	$h < m$
RI(g, m, h)	SI(h, m, s), RI(g, s, h)	TRIPLE(g, h, s, m)	$m \leq s < h$
RI(g, m, h)	LC(h, m, h-1), RC(g, h, h)	TRIPLE(g, h, -1, m)	$m < h$
SI(h, l, r)	LC(h, l, s), RC(h, s+1, r)	0	$l \leq s < r$

Chapter 5

Faster MAP inference with column generation

As was seen in section 4.2.3, there is a connection between the standard message-passing algorithm for MAP inference in tree graphical models and a linear program. This begs the following question: is it possible, then, to use tools developed for linear programming to improve the understanding and the efficiency of message-passing algorithms for MAP inference in chains and trees?

At first the answer to such a question might seem negative: simply enumerating all variables and constraints of the linear program in topological order has the same asymptotic time complexity as solving the problem by message-passing in the first place, for models for which message-passing is possible. This observation rules out using LP solving tools as black boxes for replacing the dynamic programming techniques. However, this does not rule out using higher-level LP algorithms such as cutting planes or column generation, which only require as black boxes the ability to solve smaller, restricted, versions of the linear programs at hand.

At the same time, for many NLP tasks simply making decisions locally can be sufficiently accurate. In the Penn Treebank, for example, selecting the majority POS tag for each word type (including the test set) has a higher than 90% accuracy. In this setting intuition suggests that it should be possible to find the optimal labelling of a sentence without directly considering all possible alternatives for all possible token pairs. An ideal algorithm would guess a labelling and refine it only in places where some ambiguity makes it difficult to prove that the initial guess is correct.

In this chapter we'll look at an algorithm for MAP inference in acyclic graphical models which is based on column generation. While this algorithm's asymptotic complexity is no better than simple message-passing, we'll look at experimental evidence which suggests that if the parametrization of the graphical model is favorable it is possible to achieve substantial speedups. We will not look at inference in loopy graphs as there is no exact tractable message-passing algorithm for inference in these models.

For clarity of exposition we will focus mainly on linear chain graphical models, such as those in Figure 4.3, in which the graph is connected and each variable has at most two neighbors. Sections 5.8 and 5.9 show extensions of these techniques for tree-structured graphical models and hypergraph models. Let's also assume the factors decompose into local factors, which assign scores to each setting of each variable in the chain in isolation, and pairwise compatibility factors, which assign compatibility scores to all pairs of adjacent labels. Letting $\mu_i(x_i, x_{i+1})$ be the indicator on whether the i-th pair of adjacent variables in the chain is set to values x_i, x_{i+1}, and likewise for $\theta_i(x_i)$ and $\theta_i(x_i, x_{i+1})$ representing the scores of such assignments, we can write a linear program for MAP inference as follows,

$$\begin{aligned} \textbf{max.} \quad & \sum_i \sum_{x_i, x_{i+1}} \mu_i(x_i, x_{i+1})(\theta_i(x_i, x_{i+1}) + \theta_i(x_i)) \\ \textbf{s.t.} \quad & \sum_{x_{n-1}, x_n} \mu_{n-1}(x_{n-1}, x_n) = 1 \\ & \sum_{x_{i-1}} \mu_{i-1}(x_{i-1}, x_i) = \sum_{x_{i+1}} \mu_i(x_i, x_{i+1}) \end{aligned} \tag{5.1}$$

in which all variables are assumed to be positive. It is easy to see that this LP has one variable per setting of adjacent variables in the chain, or $O(nK^2)$ variables, where K is the number of values each variable can take, but only one constraint per label of each graphical model variable, or $O(nK)$ constraints. This suggests that it should be possible to apply column generation to this problem, as the larger number of variables than constraints suggests that there are optimal solutions in which most of these variables are set to zero. Of course, it is possible to arrive at this conclusion by simply noting that a valid output y in a graphical model assigns exactly one value to each variable, and with this mapping in which each pair of graphical model variables is mapped to K^2 LP indicator variables most of their values have to be zero.

We can write the dual of this linear program as

$$\begin{aligned} \textbf{min.} \quad & T \\ \textbf{s.t.} \quad & T - \alpha_n(x_n) \geq \theta_n(x_n) \\ & \alpha_{i+1}(x_{i+1}) - \alpha_i(x_i) \geq \theta_i(x_i, x_{i+1}) + \theta_i(x_i) \end{aligned} \tag{5.2}$$

Therefore the amount by which each dual constraint is violated is the following reduced cost

$$R_i(x_i, x_{i+1}) = \theta_i(x_i, x_{i+1}) + \theta_i(x_i) + \alpha_i(x_i) - \alpha_{i+1}(x_{i+1}), \tag{5.3}$$

Algorithm 5.1 The beam search algorithm, with a beam of size one.

```
1: function Beam($\theta, y$)
2:     $\alpha_0 \leftarrow 0$
3:     $x_0^* \leftarrow 0$
4:     for Position $i$ in the chain do
5:         for value $x_i$ do
6:             $\alpha_i(x_i) \leftarrow \alpha_{i-1} + \theta_{i-1}(x_{i-1}^*, x_i)$
7:         $\alpha_i \leftarrow \max_{x_i} \alpha_i(x_i)$
```

and if this quantity is positive then a dual constraint is violated, and it might be possible to achieve a larger primal objective by inserting primal variable $\mu_i(x_i, x_{i+1})$ in the basis.

A column generation algorithm (see section 3.4.2) would then start with an arbitrary restricted version of the primal problem, and alternatively solve it by message-passing (which amounts to normal MAP message-passing where some messages are not sent) and find variables with positive reduced cost.

The content of this chapter is an extended version of prior work which appeared in NIPS 2012 [3].

5.1 A detour: beam search for approximate MAP inference

Perhaps the most popular approximate inference algorithm for acyclic graphical models is *beam search*. It works similarly to variable elimination, with one key difference: when computing the local factors for the neighbor of an eliminated variable it only considers the d best-scoring settings for that variable, ignoring the future pairwise compatibility scores. The value d is known as the width of the beam. If $d = K$ then beam search is exactly the Viterbi algorithm, and finds the exact solution, while for smaller beam widths it is approximate, with no guarantees on its performance. The pseudocode for beam search with a beam of size one can be seen in Algorithm 5.1. The time complexity of beam search is $O(ndK)$, and the space complexity is $O(nK)$.

The intuition behind beam search, and explanation for it being mostly successful and widely used, is that local information is more than often enough to select the best answer, and in the cases when it isn't the pairwise compatibility with the immediately preceding state often will.

Beam search can also be interpreted as solving a restricted version of the MAP inference linear program in equation (5.2) by coordinate ascent, in which the set of constraints to include is dynamically chosen as the algorithm progresses. Because not all constraints are eventually considered the beam search algorithm is not exact. The techniques described in this chapter allow one to bound the suboptimality gap of beam search and modify it such that convergence to the exact solution is guaranteed at a small additional computational cost.

5.2 An efficient reduced-cost oracle

Generally, the key to an efficient column generation algorithm is the ability to exploit the structure of the problem to find LP variables with positive reduced costs in less time than it would take to simply enumerate all variables directly. In our specific case, while the reduced cost expression from equation (5.3) is technically sufficient for a column generation algorithm, a simple enumeration of all LP variables and computation of their reduced costs is itself as expensive as the Viterbi algorithm. In this section we will see how exploiting the structure of the problem allows us to design an efficient reduced-cost oracle that, while having worst-case performance no better than explicitly enumerating all LP variables, in practice finds an answer in much less time.

Revisiting the reduced-cost expression, we have that at position i in the chain, for variable assignments x_i and x_{i+1}, the reduced cost is the following expression,

$$R_i(x_i, x_{i+1}) = \theta_i(x_i, x_{i+1}) + \theta_i(x_i) + \alpha_i(x_i) - \alpha_{i+1}(x_{i+1}). \tag{5.4}$$

A sufficient condition for the algorithm to have converged is that the maximum reduced-cost is no greater than zero, or

$$\max_{i, x_i, x_{i+1}} R_i(x_i, x_{i+1}) \leq 0, \tag{5.5}$$

as if this is the case then we have dual feasibility and then optimality.

To obtain a fast column-generation algorithm then a good strategy is to devise upper-bounds on the reduced cost which can be efficiently computed and are sufficiently tight. Here we'll look at upper-bounding the reduced costs for all LP variables at any given position i in the chain. One simple way to do so is to, if possible, precompute upper bounds on the rows and columns of the transition matrices $\theta_i(x_i, x_{i+1})$,

$$\begin{aligned} T^+(x_i) &\geq \max_{x_{i+1}} \theta_i(x_i, x_{i+1}) && (5.6) \\ T^-(x_{i+1}) &\geq \max_{x_i} \theta_i(x_i, x_{i+1}). && (5.7) \end{aligned}$$

Note that for many linear chain graphical models, such as the models used in these experiments, these bounds can be computed a priori, as the transition weights do not depend on the observations. Even when they do it is often possible to bound their

Algorithm 5.2 An efficient reduced-cost oracle algorithm.

```
1: function ReducedCostOracle($i, T^+, T^-, S^+, S^-$)
2:     $U^+ \leftarrow \max_{x_i} S^+(x_i) + T^+(x_i)$
3:     $C^- \leftarrow \{x_{i+1} | S^-(x_{i+1}) + T^-(x_{i+1}) + U^+ > 0\}$
4:     $U^- \leftarrow \max_{x_{i+1} \in C^-} S^-(x_{i+1}) + T^-(x_{i+1})$
5:     $C^+ \leftarrow \{x_i | S^+(x_i) + T^+(x_i) + U^- > 0\}$
6:     Return $\{(x_i, x_{i+1}) | x_i \in C^+, \quad x_{i+1} \in C^-, \quad R_i(x_i, x_{i+1}) > 0\}$
```

magnitude by computing how many features can be active for a given observation and inspecting the model's parameters.

Moreover, let's define for convenience

$$S^+(x_i) = \theta_i(x_i) + \alpha_i(x_i) \tag{5.8}$$

$$S^-(x_{i+1}) = -\alpha_{i+1}(x_{i+1}) \tag{5.9}$$

to be the parts of the reduced cost which depend only on the left and right variable value, respectively.

Given these upper bounds, then, one can compute an upper bound on the value of any reduced cost for an LP variable at position i by

$$\max_{x_i, x_{i+1}} R(x_i, x_{i+1}) \le \max_{x_i} \left(S^+(x_i) + \frac{1}{2} T^+(x_i) \right) + \max_{x_{i+1}} \left(S^-(x_{i+1}) + \frac{1}{2} T^-(x_{i+1}) \right), \tag{5.10}$$

which can be computed in $O(K)$ by independently maximizing over x_i and x_{i+1}. It is easy to compute, given an upper-bound on the terms which depend only on x_i, the set of values of x_{i+1} for which the upper bound is positive, and vice versa. Then only this pruned set of values needs to be examined for dual constraint violations, hence even when there are variables with positive reduced cost it is possible to find them in subquadratic time.

Algorithm 5.2 shows how to use these upper bounds to compute an efficient reduced-cost oracle, which will, for any position in the chain, return all LP variables which have positive reduced cost. These variables can then be added to the restricted primal problem in a column generation algorithm.

5.3 Better reduced costs from forward and backward messages

While we technically already have enough ingredients for a successful column generation algorithm, the situation is still unsatisfying for a few reasons. The main one is that, inspecting the reduced cost in equation (5.3), it is easy to see that it will be nonzero if there is a better path for arriving at any given x_{i+1} than one which is in the restricted primal problem. This is intuitively wasteful, because it might be that even the best possible way of arriving at state x_{i+1} still leaves it with a score low enough to never be a part of the final solution, so there is no need to consider LP variables which assign that value to that position in the chain. This suggests we could somehow try to incorporate information from both sides of the chain in the expression for the reduced cost.

The usual dynamic programming algorithm for forward-backward MAP inference in chains, seen in algorithm 4.4, can be converted to the following LP:

$$\begin{aligned}
\textbf{min.}\quad & F + B \\
\textbf{s.t.}\quad & F - \mu_{n-1}(x_{n-1}, x_n) \geq \theta_{n-1}(x_{n-1}, x_n) + \theta_n(x_n) \\
& \alpha_{i+1}(x_{i+1}) - \alpha_i(x_i) \geq \theta_i(x_i, x_{i+1}) + \theta_i(x_i) \\
& B - \mu_0(x_0, x_1) \geq \theta_0(x_0, x_1) \\
& \beta_{i-1}(x_{i-1}) - \beta_i(x_i) \geq \theta_{i-1}(x_{i-1}, x_i) + \theta_i(x_i)
\end{aligned} \tag{5.11}$$

This LP is simply the LP corresponding to the forward-backward dynamic programming algorithm for message passing, shown in algorithm 4.4. Note that the value of this LP is the sum of the best forward and backward scores, thereby being twice the value of the graphical model.

This LP corresponds to the dual of the following LP,

$$\begin{aligned}
\textbf{max.}\quad & \sum_i \mu_i^f(x_i, x_{i+1})(\theta_i(x_i, x_{i+1}) + \theta_i(x_i)) + \sum_i \mu_i^b(x_{i-1}, x_i)(\theta_{i-1}(x_{i-1}, x_i) + \theta_i(x_i)) \\
\textbf{s.t.}\quad & \sum_{x_{n-1}, x_n} \mu_{n-1}^f(x_{n-1}, x_n) = 1 \\
& \sum_{x_0, x_1} \mu_0^b(x_0, x_1) = 1 \\
& \sum_{x_{i-1}} \mu_{i-1}^f(x_{i-1}, x_i) = \sum_{x_{i+1}} \mu_i^f(x_i, x_{i+1}) \\
& \sum_{x_{i+1}} \mu_i^b(x_i, x_{i+1}) = \sum_{x_{i-1}} \mu_{i-1}^b(x_{i-1}, x_i)
\end{aligned} \tag{5.12}$$

It is easy to see, however, that any set of edges which is active in a solution which is optimal for the "backward" part of the LP is also an optimal solution for the "forward" part of the LP, and hence given any optimal solution one can set $\mu^f = \mu^b$ and still have an optimal solution to the problem. Enforcing this constraint directly leads to the following LP

$$
\begin{aligned}
\textbf{max.} \quad & \sum_i \mu_i(x_i, x_{i+1})(2\theta_i(x_i, x_{i+1}) + \theta_i(x_i) + \theta_{x_{i+1}}) \\
\textbf{s.t.} \quad & \sum_{x_{n-1}, x_n} \mu_{n-1}(x_{n-1}, x_n) = 1 \\
& \sum_{x_0, x_1} \mu_0(x_0, x_1) = 1 \\
& \sum_{x_{i-1}} \mu_{i-1}(x_{i-1}, x_i) = \sum_{x_{i+1}} \mu_i(x_i, x_{i+1}) \\
& \sum_{x_{i+1}} \mu_i(x_i, x_{i+1}) = \sum_{x_{i-1}} \mu_{i-1}(x_{i-1}, x_i)
\end{aligned}
\tag{5.13}
$$

which when dualized leads to the following LP

$$
\begin{aligned}
\textbf{min.} \quad & F + B \\
\textbf{s.t.} \quad & F - \mu_{n-1}(x_{n-1}, x_n) \geq \theta_{n-1}(x_{n-1}, x_n) + \theta_n(x_n) \\
& B - \mu_0(x_0, x_1) \geq \theta_0(x_0, x_1) \\
& \alpha_{i+1}(x_{i+1}) - \alpha_i(x_i) + \beta_i(x_i) - \beta_{i+1}(x_{i+1}) \geq 2\theta_i(x_i, x_{i+1}) + \theta_i(x_i) + \theta_{i+1}(x_{i+1})
\end{aligned}
\tag{5.14}
$$

The main difference between the LP in equation (5.14) and the LP in equation (5.11) is that the forward and backward constraints have been "merged". It is easy to see that a solution which is feasible in the original LP is also feasible in this forward-backward LP, but the converse is not true. Intuitively, what the merged constraint states is that along any given primal LP variable $\mu_i(x_i, x_{i+1})$ even if it could provide a better way at "arriving" at $\alpha_{i+1}(x_{i+1})$ when going forward in the chain it need not be in the solution if it cannot also provide a better way of "arriving" at $\beta_i(x_i)$ when going backwards in the chain.

5.4 The main column generation algorithm

As seen in section 3.4.2, the main structure of a column generation algorithm is shown in Algorithm 3.2. Its main ingredients are a restricted primal problem, how to solve it, and how to select variables with positive reduced cost to add to it. We will tackle each item in turn.

First, our restricted primal problem is defined as follows. For each position i in the linear chain we keep a domain D_i, a set of the possible labels for that token which are active. This domain is initialized, at first, to be the label with the best local score. The set of primal variables in the restricted problem is defined to be all primal variables involving labels in the domain for each position in the chain. When adding a primal variable to the restricted cost we simply add the label for each of its endpoints to the corresponding domain. While this adds more primal variables than strictly necessary we found that after adding one variable other variables involving the same endpoints would soon be added, and maintaining an explicit list of variables in each position with size potentially quadratic in the label set to be more costly than considering these unnecessary primal variables.

We also keep for each position i a set C_i of candidate labels. The label with the highest local score x_i^* is a candidate label, and a label x_i is not a candidate only if

$$\theta_i(x_i) + T_i^+(x_i) + T_i^-(x_i) \leq \theta_i(x_i^*) + L^+(x_i^*) + L^-(x_i^*), \tag{5.15}$$

where

$$\begin{aligned} L^+(x_i) &\leq \min_{x_{i+1}} \theta_i(x_i, x_{i+1}) & (5.16)\\ L^-(x_{i+1}) &\leq \min_{x_i} \theta_i(x_i, x_{i+1}). & (5.17)\end{aligned}$$

That is, a label is filtered from the candidate set if its possible to upper-bound its score to be less than a lower bound of the score of another label. Hence labels which are not in the candidate set cannot possibly be a part of an optimal solution. As far as we are aware this pruning method is novel, and as the experiments show it can be substantially faster than naively applying max-product. This pruning technique is referred to as Viterbi+P in the experiments.

When solving the restricted primal problem, dual variables $\alpha_i(x_i), \beta_i(x_i)$ for all candidate labels $x_i \in C_i$ are computed, but their maximizations are made only with respect to variables in the domain. That is,

$$\begin{aligned} \alpha_{i+1}(x_{i+1}) &= \max_{x_i \in D_i} \alpha_i(x_i) + \theta_i(x_i, x_{i+1}) + \theta_i(x_i) & (5.18)\\ \beta_{i-1}(x_{io1}) &= \max_{x_i \in D_i} \beta_i(x_i) + \theta_{i-1}(x_{i-1}, x_i) + \theta_i(x_i). & (5.19)\end{aligned}$$

This solves the dual of the restricted LP because it enforces the dual constraints corresponding to all primal variables in the LP, in this case all primal variables touching labels which are in the domains of each position in the chain.

Algorithm 5.3 The main column generation algorithm.

```
1: for $i \leftarrow 1 \rightarrow n$ do
2:     $D_i \leftarrow \{\arg\max \theta_i(x_i)\}$
3: while domains have not converged do
4:     $(\alpha, \beta) \leftarrow GetMessages(D, \theta)$
5:     for $i \leftarrow 1 \rightarrow n$ do
6:         $D_i^*, D_{i+1}^* \leftarrow$ ReducedCostOracle($i$)
7:         $D_i \leftarrow D_i \cup D_i^*$
8:         $D_{i+1} \leftarrow D_{i+1} \cup D_{i+1}^*$
```

While this representation is more wasteful, as dual constraints are being enforced for primal variables which were never found to have a positive reduced cost, we found experimentally that keeping a list of active edges instead of active values for each variable can be memory-intensive and take longer to converge, as often adding one edge touching one value would be followed by adding neighboring edges.

Note that this restricted problem assigns a value to all dual variables, even to those whose constraints are unlikely to be violated in a dynamic programming solution. While this seems wasteful, guessing values for these dual variables in a way that preserves feasibility, necessary for reasoning in terms of a primal-dual optimal as in section 5.3, and allows the efficient computation of a reduced-cost oracle proved difficult.

To find all edges with a positive reduced cost we can use the generic oracle from section 5.2.

Algorithm 5.3 shows the main column generation algorithm, and algorithm 5.2 shows the reduced-cost oracle.

5.5 Estimating the accuracy gap of beam search

It is possible to use the reduced costs to upper-bound the score of any primal solution.

Theorem 1. *Given a primal feasible solution to a restricted primal problem with score S, and given its dual variables, the score of any other solution can be bounded as*

$$S^* \leq S + \sum_i \max_{x_i, x_{i+1}} R_i(x_i, x_{i+1}). \tag{5.20}$$

We can sketch a proof of this as follows. Intuitively, because our indicator variables are bound to be between 0 and 1, the maximum increase in score obtainable by adding a primal variable to the model is its reduced cost. Hence if one has a partial solution, obtained from beam search or from one of the variants of column generation, one can stop search early and quantify the duality gap between the current solution and the best possible solution.

Note that naively computing the maximizations in equation (5.20) has the same worst-case complexity as exact inference, but can be accelerated in practice using the reduced cost oracle from Algorithm 5.2.

5.6 Loss-augmented inference for the 0/1 loss

As described in section 4.1.5, a popular loss function for estimating the parameters of structured linear models is the structured SVM with the 0/1 loss function, whose gradient is defined as

$$\nabla \ell_{svm}(o, \mathbf{x}^*, \theta) = \mathbf{f}(o, \arg\max_{\mathbf{x}} \theta^T \mathbf{f}(o, \mathbf{x}) + \delta(\mathbf{x} \neq \mathbf{x}^*)) - \mathbf{f}(o, \mathbf{x}^*). \tag{5.21}$$

To compute this gradient, then, it is necessary to find any solution with score greater than the score of the true label minus 1, if one exists. If the ground truth solution is not the solution with a highest score MAP inference will return a margin violation. However, if the ground truth solution is the optimal this reduces to finding the second-best-scoring solution, but only if there is such a solution with score less than 1 smaller than the score of the ground truth solution.

The column-generation algorithm can be adapted to this task as follows. If a primal variable is part of a solution with score at most γ smaller than the score of the best solution then its reduced cost is also lower bounded by $-\gamma$. Then if one finds the set of primal variables with reduced cost greater than $-\gamma$ and runs any algorithm of choice for two-best inference restricted to those primal variables one is guaranteed to find a margin violation if there is one.

The following theorem ensures the validity of this algorithm:

Theorem 2. *If we add to the restricted LP all primal variables with reduced cost larger than $-\gamma$ in an optimal solution, k-best Viterbi inference using only the primal variables in the restricted problem is guaranteed to return all of the top-k assignments with score equal to $MAP - \gamma$ or larger, where MAP is the score of the MAP assignment.*

Proof. While the notion of 2-best setting is not well-defined in terms of the LP (because it can take on non-integral values), the one-to-one correspondence between settings of the graphical model variables and feasible integral settings of the LP variables allows us to reason about the properties of specific parts of a two-best solution.

Let μ^*, λ^* be a primal-dual optimal pair for a MAP inference LP, and let $D_1, \ldots, D_n$ be the restricted domains of the primal variables. Let $S(\mu)$ refer to the LP objective for μ and $L(\mu, \lambda)$ refer to the Lagrangian score for μ, λ.

We'll use the subscript v to index the LP variables. Let $R_v(\mu_v, \lambda)$ refer to the reduced cost of LP variable v w.r.t dual variable vector λ.

Suppose, for the sake of contradiction, that there exists a setting of the graphical model variables such that its corresponding integral LP variable setting $\tilde{\mu}$ is such that $S(\mu^*) - S(\tilde{\mu}) < \gamma$ and some component of $\tilde{\mu}$ isn't in the instantiated pairwise domains. I.e. $\exists v \; s.t. \; \tilde{\mu}_v \notin D_v$. Note that by defining $\tilde{\mu}$ in terms of a setting to the graphical model variables, we are specifying that $\tilde{\mu}$ satisfies the primal constraints by construction.

Since $\tilde{\mu}$ satisfies the primal constraints, $S(\tilde{\mu}) = L(\tilde{\mu}, \lambda^*)$. We also have that $S(\mu^*) = L(\tilde{\mu}, \lambda^*)$ because μ^*, λ^* is a primal-dual pair. Therefore, we have the condition that $L(\tilde{\mu}, \lambda^*) - L(\tilde{\mu}, \lambda^*) < \gamma$. Now, let's rewrite the terms of the Lagrangian in terms of reduced costs. Let V be the index set for the components of μ_v:

$$\sum_{v \in V} R_v(\mu^*_v, \lambda^*)\mu^*_v - \sum_{v \in V} R_v(\tilde{\mu}_v, \lambda^*)\tilde{\mu}_v \leq \gamma \tag{5.22}$$

Since $\tilde{\mu}$ is integral, we know that $\tilde{\mu}_v$ is either 0 or 1. The same is true for μ^*_v because we know the LP is tight. Therefore, we can cancel out the sums over the subset of V where $\tilde{\mu}_v = \mu^*_v$. We next split up this index set of places where they disagree into two sets, S^+, where $\mu^*_v = 1$ and $\tilde{\mu}_v = 0$, and S^-, where $\mu^*_v = 0$ and $\tilde{\mu}_v = 1$:

$$\sum_{v \in S^+} R_v(\mu^*_v, \lambda^*)\mu^*_v - \sum_{v \in S^-} R_v(\tilde{\mu}_v, \lambda^*)\tilde{\mu}_v < \gamma \tag{5.23}$$

By the optimality of μ^*, we know that every term in the left hand sum is zero. Therefore, we have that:

$$\sum_{v \in S^-} R_v(\tilde{\mu}_v, \lambda^*)\tilde{\mu}_v + \gamma \geq 0 \tag{5.24}$$

Next, we split S^- up further into two sets. Let S^-_I refer to the index set of components $\tilde{\mu}_v$ with $\tilde{\mu}_v \in D_v$ and S^-_O refer to components $\tilde{\mu}_v$ with $\tilde{\mathbf{x}}_v \notin D_v$.

By the initial assumptions of the proof, we have that S^-_O is nonempty. We actually have that $|S^-_O| \geq 2$. If $|S^-_O| = 1$, then there's no way that the primal constraints could be satisfied by $\tilde{\mu}$ because LP variables for adjacent primal variables in the graphical model must agree on the setting of the node where they intersect.

For $v \in S^-_I$, we have that $R_v(\tilde{\mu}_v, \lambda^*) < 0$, by the fact that $\tilde{\mu}_v$ isn't used in the LP optimum. For $v \in S^-_O$, we have that $R_v(\tilde{\mu}_v, \lambda^*) \leq -\gamma$, by the assumptions of the proof.

Therefore, we have:

$$\begin{aligned}
\sum_{v\in S^-} R_v(\tilde{\mu}_v,\lambda^*)\tilde{\mu}_v+\gamma &= \sum_{v\in S^-} R_v(\tilde{\mu}_v,\lambda^*)+\gamma && (5.25)\\
&= \sum_{v\in S_I^-} R_v(\tilde{\mu}_v,\lambda^*)+\sum_{v\in S_O^-} R_v(\tilde{\mu}_v,\lambda^*)+\gamma && (5.26)\\
&\leq \sum_{v\in S_O^-} R_v(\tilde{\mu}_v,\lambda^*)+\gamma && (5.27)\\
&\leq -\gamma|S_O^-|+\gamma && (5.28)\\
&\leq -2\gamma+\gamma && (5.29)\\
&\leq 0 && (5.30)
\end{aligned}$$

But we had in equation (5.24) that $\sum_{v\in S^-} R_v(\tilde{\mu}_v,\lambda^*)\tilde{\mu}_v+\gamma\geq 0$. Therefore, we have a contradiction, and the proof is complete. □

Note that this can be extended to k-best inference if one has a parameter γ such that such that all k-best solutions have a score at most γ worse than the optimal score. If γ is not known *a priori* it can be determined by first guessing any value of γ (including 0), running k-best decoding using only the primal variables originally added to the restricted LP, computing the score of the k-th solution returned, and set γ to be the difference between that score and the optimal score, as then one is guaranteed to find the true k best settings possibly using a small subset of the settings of the primal variables in the graphical model.

5.7 Exploiting time vs accuracy tradeoffs

The runtime of the reduced-cost oracle from section 5.2 depends quadratically on the number of candidates x_i or x_{i+1} whose reduced cost cannot be bound away from 0. It is then desirable to tighten these bounds as much as possible to ensure that these sets are small.

Since parameter estimation is structured linear models is often done with ℓ_2 regularization, a convenient way to control the tightness of the bound is to change the strength of the regularizer of the pairwise scores, relatively to the regularizer of the other scores during parameter estimation. This will then learn a different model, possibly with lower accuracy, but on which column generation is a faster inference method.

5.8 A forward-backward reduced cost for tree models

The LP version of the MAP problem in trees, as seen in section 4.2.3, is

$$\begin{aligned}
\textbf{max}.\quad & \sum_{i,x_i} \mu_i(x_i)\theta_i(x_i) + \sum_{ij\in E}\sum_{x_i,x_j} \mu_{ij}(x_i,x_j)\tau_{ij}(x_i,x_j) \\
\textbf{s.t.}\quad & \sum_{x_i} \mu_i(x_i) = 1 \\
& \sum_{x_i} \mu_{ij}(x_i,x_j) = \mu_j(x_j) \\
& \sum_{x_j} \mu_{ij}(x_i,x_j) = \mu_i(x_i)
\end{aligned} \tag{5.31}$$

Following Wainwright and Jordan [106], we choose an arbitrary node as the root of the tree and rewrite 5.31 as

$$\begin{aligned}
\textbf{max}.\quad & \sum_{x_1} \mu_1(x_1)\theta_1(x_1) + \sum_{ij\in E}\sum_{x_i,x_j} \mu_{ij}(x_i,x_j)(\tau_{ij}(x_i,x_j)+\theta_j(x_j)) \\
\textbf{s.t.}\quad & \sum_{x_1} \mu_1(x_1) = 1 \\
& \sum_{x_i} \mu_{ij}(x_i,x_j) = \sum_{x_k} \mu_{jk}(x_j,x_k), \quad \forall k \in N(j)-i, \forall i,j \in E, x_j.
\end{aligned} \tag{5.32}$$

This is done by assigning the local score of each node, besides the root, to its parent transition edge.

From this LP you can derive the following Lagrangian,

$$\begin{aligned}
L(\mu,\lambda,T) = & \sum_{x_1} \mu_1(x_1)\theta_1(x_1) + \sum_{ij\in E}\sum_{x_i,x_j} \mu_{ij}(x_i,x_j)(\tau_{ij}(x_i,x_j)+\theta_j(x_j)) + T\left(\sum_{x_1}\mu_1(x_1) - 1\right) \\
& + \sum_{ij\in E}\sum_{x_j}\sum_{k\in N(j)-i} \lambda_{kj}(x_j)\left(\sum_{x_k}\mu_{jk}(x_j,x_k) - \sum_{x_i}\mu_{ij}(x_i,x_j)\right)
\end{aligned} \tag{5.33}$$

whose terms can be rearranged to give rise to the following expression with reduced costs

$$\begin{aligned}
L(\mu,\lambda,T) = & \sum_{x_1} \mu_1(x_1)\left(\theta_1(x_1) - \sum_{j\in N(1)} \lambda_{ij}(x_1) - T\right) \\
& + \sum_{ij\in E}\sum_{x_i,x_j} \mu_{ij}(x_i,x_j)\left(\tau_{ij}(x_i,x_j) + \theta_j(x_j) + \sum_{k\in N(j)-i} \lambda_{kj}(x_j) - \lambda_{ji}(x_i)\right)
\end{aligned} \tag{5.34}$$

By setting the maximum reduced cost of the pairwise marginals at each edge to zero we get the standard max-product message-passing updates on trees,

$$\lambda_{ji}(x_i) = \max_{x_j} \tau_{ij}(x_i, x_j) + \theta_j(x_j) + \sum_{k \in N(j)-i} \lambda_{kj}(x_j) \tag{5.35}$$

Note that the value of these messages depends only on the fact that the root of the tree is closer to i than it is to j; otherwise the message going across this edge would be analogous, except going in the other direction.

This shows that if we picked each leaf in turn as the root, computed the Lagrangian, and averaged them, we'd get, at each edge, only two distinct values for the dual variables associated with it, depending on whether the root is closer to i or to j from that edge. The average Lagrangian, then, would have a value equal to

$$\begin{aligned} L(\mu, \lambda, T) = {} & \frac{1}{N_l} \sum_{\text{leaf } l} \sum_{x_k} \mu_l(x_l) \left(\theta_l(x_l) - \sum_{j \in N(1)} \lambda_{ij}(x_1) - T \right) \\ & + \sum_{ij \in E} \frac{1}{N_i + N_j} \sum_{x_i, x_j} \mu_{ij}(x_i, x_j) \Bigg((N_i + N_j) \tau_{ij}(x_i, x_j) + N_i \theta_i(x_i) N_j \theta_j(x_j) \\ & + N_i \left(\sum_{k \in N(j)-i} \lambda_{kj}(x_j) - \lambda_{ji}(x_i) \right) + N_j \left(\sum_{k \in N(i)-j} \lambda_{ki}(x_i) - \lambda_{ij}(x_j) \right) \Bigg) \end{aligned} \tag{5.36}$$

where N_l is the number of leaves in the tree, N_i the number of leaves reachable from node i.

This is the Lagrangian of the following primal problem

$$\begin{aligned} \textbf{max}. \quad & \frac{1}{N_l} \sum_{l} \sum_{x_l} \mu_l(x_l) \theta_l(x_l) + \sum_{ij} \frac{1}{N_i + N_j} (N_l \tau_{ij}(x_i, x_j) + N_i \theta_i(x_i) + N_j \theta_j(x_j)) \\ \textbf{s.t.} \quad & \sum_{x_l} \mu_l(x_l) = 1, \quad \forall \text{leaf } l \\ & \sum_{k \in N(i)-j} N_k \sum_{x_k} \mu_{ki}(x_k, x_i) = N_i \sum_{x_j} \mu_{ij}(x_i, x_j), \quad \forall i, x_i \end{aligned} \tag{5.37}$$

Note that the constraint here is equivalent to the sum of many marginalization constraints, as

$$\sum_{k \in N(i)-j} N_k = N_i,$$

as the number of leaves reachable through all descendants of i has to be equal to the number of leaves reachable through i.

Also note that this LP is a generalization for trees of the LP for chains presented in equation (5.13).

The reduced cost in such an LP, for each edge ij, is

$$\begin{aligned} N_l R_{ij}(x_i, x_j) = (N_i + N_j)\tau_{ij}(x_i, x_j) + N_i\theta_i(x_i) + N_j(\theta_j(x_j)) \\ + N_i \left(\sum_{k \in N(i)-j} \lambda_{ki}(x_i) - \lambda_{ij}(x_j) \right) \\ + N_j \left(\sum_{k \in N(j)-i} \lambda_{kj}(x_j) - \lambda_{ji}(x_i) \right) \end{aligned} \tag{5.38}$$

And, analogously to the chain case, it is easy to see that setting all dual variables to the fixed points of the max-product messages is a sufficient but not necessary condition for dual feasibility. A column generation algorithm can then be constructed by using an identical reduced-cost oracle to the linear chain case.

5.9 An inside-outside reduced-cost for hypergraph models

As seen in section 4.3.1, it is natural to represent a hypergraph model with the following linear program,

$$\begin{array}{lll} \textbf{max.} & \sum_e \mu_e w_e & \text{for hyperedge } e \\ \textbf{s.t.} & \sum_{e:o_e=n} \mu_e = \sum_{e:n\in I_e} \mu_e & \text{for node } n \text{ apart from source and target} \\ & \sum_{e:o_e=t} \mu_e = 1 & \text{for the target node t} \end{array} \tag{5.39}$$

which asserts that for each node n, the sum of the indicator variables v_e for the edges entering n (that is, edges which have n as the target node o_e) has to be equal to the sum of the indicator variables of edges leaving n (that is, edges which have n as a member of their set of inbound nodes I_e), and the objective scores each edge e by its weight w_e.

The dual of this LP is

$$\begin{aligned}
\textbf{min.} \quad & v_t \\
\textbf{s.t.} \quad & v_{o_e} - \sum_{n \in I_e} v_n \geq w_e \quad \text{for each edge } e
\end{aligned} \tag{5.40}$$

which has a variable storing the value v_n of each node n. Similarly to the graphical models case, coordinate ascent in this dual is equivalent to the standard dynamic programming algorithm.

By defining an outside value o_n for each node n it is possible to define a backwards version of this LP,

$$\begin{aligned}
\textbf{min.} \quad & v_t + \sum_{\text{source } s} o_s \\
\textbf{s.t.} \quad & v_{o_e} - \sum_{n \in I_e} v_n \geq w_e && \text{for each edge } e \\
& o_i - o_{o_e} - \sum_{n \in I_e, n \neq i} v_n \geq w_e && \text{for each edge } e\text{, input } i \in I_e
\end{aligned} \tag{5.41}$$

and its dual

$$\begin{aligned}
\textbf{max.} \quad & \sum_e w_e \left(\mu_e + \sum_{e: n \in I_e} \mu_e^n \right) \\
\textbf{s.t.} \quad & \sum_{e: o_e = n} \mu_e = \sum_{e: n \in I_e} \left(\mu_e + \sum_{n' \neq n, n' \in I_e} \mu_e^{n'} \right) && \text{for node } n \text{ apart from source and target} \\
& \sum_{e: n \in I_e} \mu_e^n = \sum_{e: n = o_e} \sum_{n' \neq n, n' \in I_e} \mu_e^{n'} && \text{for node } n \text{ apart from source and target} \\
& \sum_{e: o_e = t} \mu_e = 1 && \text{for the target node t} \\
& \sum_{e: o_e = s} \mu_e^s = 1 && \text{for all source nodes } s
\end{aligned} \tag{5.42}$$

Note that in this LP a forward indicator variable μ_e can have a value larger than 1, as its value counts how often that edge is used in the forward and backward computations.

Because of the asymmetry between sources and target in hypergraph models it is not possible to merge the primal variables for the forward and backward problems, as was done in the chain and tree cases. That said, conditioned on its neighboring inputs, the relationship between an input and its target is the same as in between adjacent node variables in the chain and tree cases, leading us to the following LP

$$\begin{aligned} \textbf{min.} \quad & v_t + \sum_{\text{source } s} o_s \\ \textbf{s.t.} \quad & v_{o_e} + o_i - o_{o_e} - v_i - 2 \sum_{n \in I_e, n \neq i} v_n \geq 2w_e \quad \text{for each edge } e, \text{ input } i \in I_e \end{aligned} \tag{5.43}$$

which has a reduced-cost expression including forward and backward variables for each input of each edge. While this LP underestimates the value of the source nodes which are not in an optimal path, the same arguments as in the chain and tree cases suffice to prove that it estimates the value of the target node correctly, as for all edges in the path there can be no gap in the "upward" messages to offset a gap in the "downward" messages.

Note, however, that unlike in the chain and tree case, it is not possible to form an efficient oracle for this reduced-cost, as there is not necessarily any clear structure in the hyperedges which can be exploited to efficiently lower-bound and upper-bound the scores of all edges which are not being considered. This might not be an issue, however, as experimentally over 95% of the CPU time of hypergraph-based dependency parsers is spent computing dot products between edge features and weights, and hence upper- and lower-bounding the values of these dot products directly might be a productive approach. Preliminary experiments however failed to obtain speedups on the third-order parsing model described in table 4.2.

5.10 Related work

Column generation has been employed as a way of dramatically speeding up MAP inference problems in Riedel et al [80], which applies it directly to the LP relaxation for dependency parsing with grandparent edges.

There has been substantial prior work on improving the speed of max-product inference in chains by pruning the search process. CarpeDiem [18] relies on an an expression similar to the oriented, left-to-right reduced cost equation (5.44), also with a similar pruning strategy to the one described in section 5.2. Following up, Kaji et al. [32] presented a staggered decoding strategy that similarly attempts to bound the best achievable score using uninstantiated domains, but only used local scores when searching for new candidates. The dual variables obtained in earlier runs were then used to warm-start the inference in later runs. Their techniques obtained similar speed-ups as ours over Viterbi inference. However, their algorithms do not provide extensions to inference in trees, a margin-violation oracle, and approximate inference using a duality gap. Furthermore, Kaji et al. use data-dependent transition scores. This may improve our performance as well, if the transition scores are more sharply peaked. Similarly, Raphael [75] also presents a staggered decoding strategy, but does so in a way that applies to many dynamic programming algorithms.

The strategy of preprocessing data-independent factors to speed up max-product has been previously explored by McAuley and Caetano [51], who showed that if the transition weights are large, savings can be obtained by sorting them offline. Our contributions, on the other hand, are more effective when the transitions are small. The same authors have also explored strategies to reduce the worst-case complexity of message-passing by exploiting faster matrix multiplication algorithms [52].

Alternative methods of leveraging the interplay between fast dynamic programming algorithms and higher-level LP techniques have been explored elsewhere. For example, in dual decomposition [85], inference in joint models is reduced to repeated inference in independent models. Tree block-coordinate descent performs approximate inference in loopy models using exact inference in trees as a subroutine [94]. Column generation is cutting planes in the dual, and cutting planes have been used successfully in various machine learning contexts. See, for example, Sontag et al [93] and Riedel et al [78].

There is a mapping between dynamic programs and shortest path problems [49]. Our reduced cost is an estimate of the desirability of an edge setting, and thus our algorithm is heuristic search in the space of edge settings. With dual feasibility, this heuristic is consistent, and thus our algorithm is iteratively constructing a heuristic such that it can perform A^* search for the final restricted LP [1].

There are other non-LP-related pruning techniques to accelerate MAP inference. For example, Wang and Koller [107] propose a fast message-passing algorithm for inference in graphs which are composed of a single cycle.

5.11 A sample execution of the CG algorithm

In this section we show an example of how the column generation algorithm works on a sample sentence from the corpus. Table 5.1 shows an example sentence, the true part-of-speech tags of its tokens, the model's initial guess, the domains after each column generation iteration, and the model's final output.

The first thing to notice is that most tokens never have their labels questioned, as they are obvious. Secondly, some mistakes are never fixed: the token "Poles" is tagges as a plural proper noun (NNPS) instead of a plural common noun (NNS), which is its correct tag. Then, some ambiguities are considered but are not sufficient to change the model's prediction: for example, the token "do" is first analyzed as a verb in its base form (VB), then considered a verb in a non-third person singular present (VBP), but finally decided to have the correct label, which was the initial guess. Finally, some tokens such as "view" have their labels fixed by the algorithm: even though it functions as a verb in its base

Sentence	True tags	Initial domains	Iteration 1	Iteration 2	Final output
The	DT	DT	DT	DT	DT
Poles	NNS	NNPS	NNPS	NNPS	NNPS
Might	MD	MD	MD PRP	MD PRP	MD
do	VB	VB	VB VBP	VB VBP	VB
better	RBR	VB	VB RB RBR JJR	VB RB RBR JJR . . .	RB
to	TO	TO	TO	TO	TO
view	VB	NN	NN VB	NN VB	VB
it	PRP	PRP	PRP	PRP	PRP
as	IN	IN	IN	IN	IN
a	DT	DT	DT	DT	DT
Trojan	NNP	NNP	NNP	NNP	NNP
Horse	NNP	NNP	NNP	NNP	NNP

Table 5.1: A sample execution of the column generation algorithm. The first column shows the sentence's tokens, the second their true parts of speech. The third column shows the CG algorithm's initial guess for the POS tag of each token, the fourth and fifth columns show the domain of each token after the first and second CG iteration, respectively, and the last column shows the exact answer predicted by CG.

form (VB), the model guesses it is a singular common noun (NN), but column generation eventually considers the true label and the model settles on it.

Note that not all errors can be fixed. The token "better" is a comparative adverb (RBR), but the model initially tags it as a verb in its base form (VB), considers many options (including the true one), and eventually settles on the generic adverb (RB), which is more similar to the true tag than the initial guess. The sizes of the domains of the tokens after the second iteration of column generation are bigger in more ambiguous tokens and smaller on more certain tokens, as is expected of a good pruning algorithm.

5.12 Experiments

In this section we experimentally analyze the proposed algorithms for the case of MAP inference in linear chains. We show that

1. it is possible to accelerate the runtime of MAP inference in linear chains with column generation;
2. the reduced-cost derived in section 5.3 leads to faster algorithms than the reduced costs of other LPs;
3. the 0/1-loss oracle from section 5.6 provides dramatic speedups to two-best inference in the case of large domain sizes; and
4. by changing the regularization strength on the transition weights it is possible to explore time vs accuracy tradeoffs.

Since there are no formal worst-case performance guarantees for column generation, and its performance depends heavily on the particular structure of the model to which it is applied, we avoid experiments on synthethic data, focusing instead on two semi-realistic problems: part-of-speech tagging with data from the Wall Street Journal section of the Penn Treebank [48], and joint part-of-speech tagging and named-entity recognition (NER) trained on the data from the CoNLL-2003 shared task [99]. NER is a sequence segmentation task which is commonly solved by reducing it to sequence tagging, where each token gets tagged as being outside a segment (O), being inside a segment (I-TYPE), or starting a new segment of the same type as the segment of the previous token (B-TYPE). The CoNLL-2003 shared task has four segment types: person, organization, location, and miscellaneous; and all explicitly named entities in the corpus are tagged with their types.

For both tasks we used standard linear chain conditional random fields trained with features similar to those in the Stanford POS tagger [101] and NER [20]. For the joint task inference was performed on a state space where each token was labeled with a tuple of POS tag and NER label. In both datasets there are 45 POS tags, and there are 8 NER labels, so the total size of the label set for the joint task is 360. All models were trained, unless otherwise specified, by optimizing ℓ_2-regularized log likelihood with the LBFGS optimizer [44]. Unless otherwise specified the strength of the ℓ_2 regularizer was chosen to maximize test-set performance.

All the code is implemented in the scala programming language [69] and the timing experiments are on a circa 2010 Macbook Pro. The code uses the Factorie library for graphical models [54], but all relevant code for inference was reimplemented as efficiently as possible for this chapter.

Table 5.2: Comparing inference time and exactness of Column Generation (CG), Viterbi, Viterbi with the final pruning technique of section 5.2 (Viterbi+P), and CG with duality gap termination condition 0.15%(CG+DG), and beam search on POS tagging (left) and joint POS/NER (right).

Algorithm	**% Exact**	**Sent./sec.**
Viterbi	100	3144.6
Viterbi+P	100	4515.3
CG	100	8227.6
CG-DG	98.9	9355.6
Beam-1	57.7	12117.6
Beam-2	92.6	7519.3
Beam-3	98.4	6802.5
Beam-4	99.5	5731.2

Algorithm	**% Exact**	**Sent./sec.**
Viterbi	100	56.9
Viterbi+P	100	498.9
CG	100	779.9
CG-DG	98.4	804
Beam-1	66.6	3717.0
Beam-5	98.5	994.97
Beam-7	99.2	772.8
Beam-10	99.5	575.1

For the first experiment we compare the performance of the following inference algorithms on the two tasks:

- Viterbi, the standard forward message-passing algorithm;
- Viterbi+P, which uses Viterbi but only over the labels for each token which could conceivably be the maximizer, using the pruning technique described in section 5.4;
- CG, which is column generation, algorithm 5.3, using the reduced-cost from section 5.3;
- CG-DG, which is column generation for approximate inference, as described in section 5.5 which terminates if the score is within 0.15% of the best possible answer; and
- Beam-N, which is beam search considering the N best labels for each token.

Note that Viterbi, Viterbi+P, and CG are exact inference algorithms, while CG-DG and Beam-N are approximate.

Table 5.2 shows the performance of the algorithms in both the independent and joint tasks. The conclusions are qualitatively similar: the pruning technique from Viterbi+P

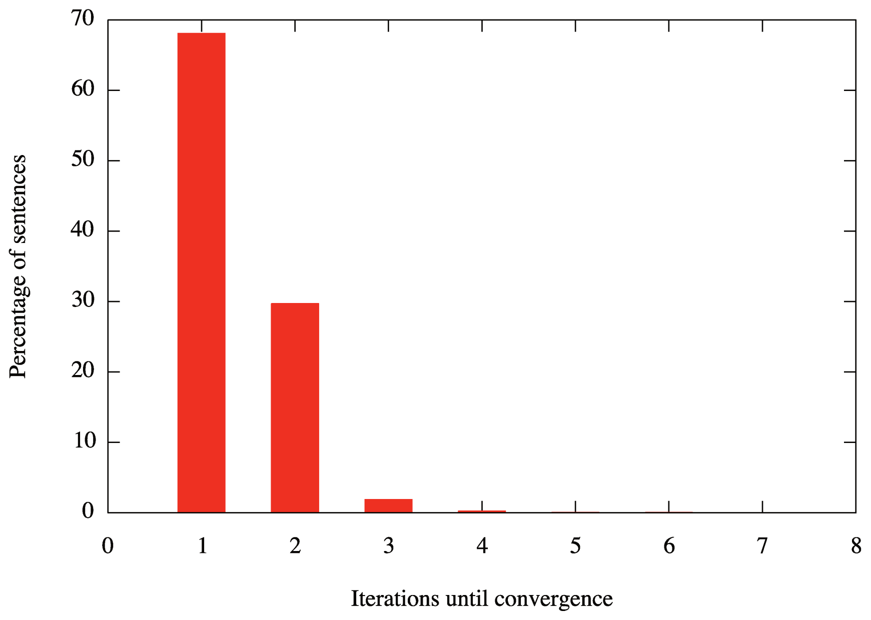


Figure 5.1: A histogram of iterations until convergence of column generation in the POS tagging experiment.

is by itself substantially faster than Viterbi, and column generation is faster still. For the POS tagging experiment CG was faster than beam search with a beam of size two, which was exact only 92.6% of the time, while in the joint experiment it was equivalent to a beam of size seven, which is still exact only 99.2% of the time.

Since column generation might take many iterations until convergence it's surprising that it can be as fast as beam search considering only two labels per token. Figure 5.1 shows why this is true: in almost 70% of the sentences column generation terminates after the first iteration, and only on a small fraction it needs more than three iterations. A similar histogram for the joint POS/NER experiment shows that the mean number of iterations is bigger, reflecting the bigger runtime.

Note that for ease of comparison only the time to actually perform inference was recorded; the local scores and transition scores were precomputed for all inference algorithms. All experiments were performed on a single machine on a single thread with no substantial load, but we do not report precise timing numbers as those are irreproducible.

For the second experiment we compare the performance of column generation on POS tagging using three different expressions for the reduced-cost:

- CG, which uses the reduced-cost from section 5.3;

Table 5.3: The performance of column generation with different reduced-costs.

Reduced Cost	**POS Sent./sec.**
CG	8227.6
CG-α	5125.8
CG-α+θ_{i+1}	4532.1

Table 5.4: The speedups for a 0/1 loss oracle. The table shows sentences per second for two-best inference using Viterbi, pruned Viterbi, and CG.

	POS	POS/NER
Method	**Sent./sec.**	**Sent./sec.**
Viterbi 2-best	56.0	.06
Viterbi+P 2-best	119.6	11.7
CG	85.0	299.9

- CG-α, which uses the following reduced-cost

$$R_i(x_i, x_{i+1}) = \alpha_{i+1}(x_{i+1}) - \alpha_i(x_i) - \theta_i(x_i) - \theta_i(x_i, x_{i+1}), \tag{5.44}$$

  which incorporates only information from the forward dual variables α (and hence no backward pass was performed); and

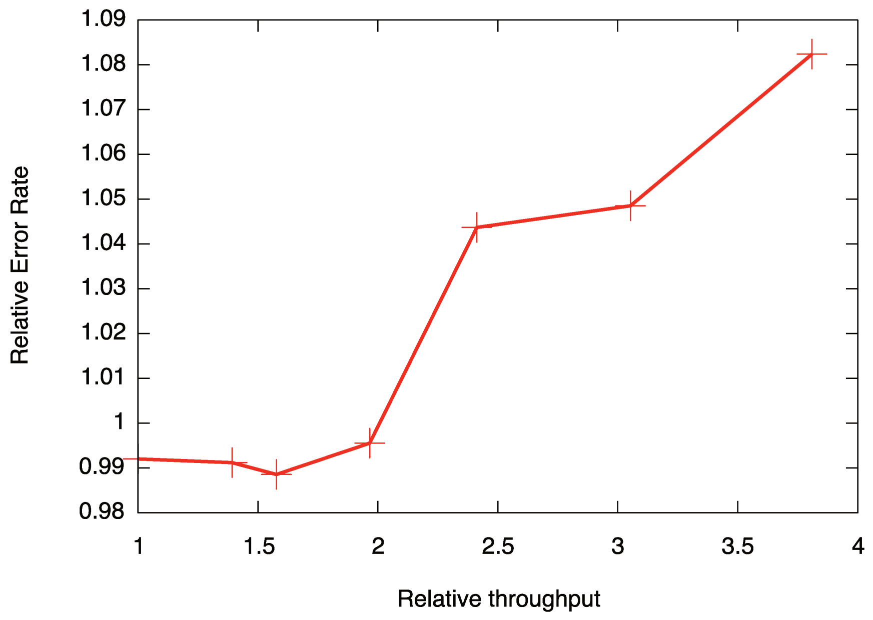
- CG-α+θ, which uses the following reduced-cost

$$R_i(x_i, x_{i+1}) = \alpha_{i+1}(x_{i+1}) - \alpha_i(x_i) - \frac{1}{2}(\theta_i(x_i) + \theta_{i+1}(x_{i+1})) - \theta_i(x_i, x_{i+1}), \tag{5.45}$$

  which also makes exclusive usage of the forward dual variables α but also looks at the local score of the second node in the path (and hence message-passing is performed slightly different to account for the different definitions of the dual variables).

The results are in Table 5.3. As expected, the reduced-cost expression from section 5.3 leads to faster inference algorithms, as it incorporates global information from both sides of the linear chain.

For the third experiment we analyze the speedups obtained by using the 0/1-loss oracle defined in section 5.6. We compare three different algorithms:



t

Figure 5.2: Training-time manipulation of accuracy vs. test throughput for our algorithm.

- Viterbi 2-best, a simple adaptation of viterbi which stores the two best values for each maximization operation;
- Viterbi+P 2-best, which is similar to Viterbi but prunes the labels which are guaranteed to not be a part of any two-best solution; and
- CG, which uses column generation to find a set of edges guaranteed to contain a margin violation if one exists and then runs Viterbi 2-best on those edges.

Table 5.4 shows the results of this experiment. For POS tagging the simple Viterbi+P 2-best outperformed CG, which outperformed Viterbi, but in the joint task these conclusions were reversed, and CG was many hundreds of times faster than Viterbi. This is because there are far more states to prune in the joint task than in the tagging task, and hence a more effective but more expensive pruning method is beneficial.

For the fourth and final experiment we explore the time versus accuracy tradeoffs allowed by the column generation algorithm. In this experiment we train many models for POS tagging, varying the strength of ℓ_2 regularization of the transition weights between 0.1 and 10 times the strength of the local scores, leaving the regularization of the local scores fixed to the same value used in the other experiments. We then plot, for each regularization strength, the relative speedup of column generation, when analyzing all sentences of the test set, and the relative accuracy of this model. Figure 5.2 shows that a 4x gain in speed can be obtained at the expense of an 8% relative decrease in accuracy, which might be preferrable to using approximate inference.

5.13 Conclusions

In this chapter we presented an efficient family of algorithms based on column generation for MAP inference in chains and trees. This algorithm exploits the fact that inference can often rule out many possible values, and we can efficiently expand the set of values on the fly. Depending on the parameter settings it can be twice as fast as Viterbi in WSJ POS tagging and 13x faster in a joint POS-NER task.

The approach of accelerating dynamic programming algorithms with column generation seems fairly generic. It is still an open problem whether more efficient reduced cost formulations can be found for other important model families, such as specific hypergraph models used in parsing.

Chapter 6

Linear programming relaxations for joint inference

All known efficient algorithms for MAP inference need to make assumptions about which structure the parts can have. In graphical models, for example, the time complexity of MAP inference is exponential in the treewidth of the graph, and in non-projective dependency parsing inference over parts which correspond to presence/absence of single edges in a tree leads to a polynomial-time algorithm but inference over parts which can condition on two edges is NP-complete [59].

While efficient inference algorithms are known for many simple structured prediction tasks, if one wants to have parts which don't have the specified structure those algorithms cannot be applied, and one is forced to resort to approximate methods, which can have negative consequences for training a model [40], apart from possibly producing suboptimal results.

When all the parts in a model can be partitioned such that for each partition an efficient inference algorithm is known this problem is referred to as *joint inference*, as it is effectively doing inference jointly on many models. A popular family of algorithms for joint inference, applicable whenever the parts decompose in this way and the validity conditions can be expressed as linear equality constraints across the submodels is dual decomposition [83, 92, 37]. In dual decomposition the linearity of the constraints is exploited by constructing a dual problem which does independent inference in each structured linear model (hereafter referred to as a submodel) separately, with possibly different weights for some parts, and shifts mass between parts in different models until the validity constraint is satisfied.

In this chapter we present a new dual formulation for joint inference, which as well as allowing for the linear equality constraints in dual decomposition also allows the creation of parts which span different submodels. This objective is very similar to the dual decomposition objective, and can be solved with the same subgradient algorithm used for dual decomposition or, if one suitably constrains the structure of these parts, a block coordinate descent algorithm similar to MPLP [23].

We compare this dual formulation with standard dual decomposition in two corpus-wide inference task for natural language processing: joint part-of-speech tagging and joint dependency parsing [84].

The material from this chapter is an extension of a paper which has been submitted to the JMLR [70].

6.1 A novel linear programming formulation of joint inference

As seen in section 4.1.2, the dual decomposition algorithm can solve the inference problem efficiently if each part in a model can be assigned exclusively to one out of many submodels on which inference is efficient, and the validity constraints can be expressed as linear equality constraints.

However, often one wants to assign scores to functions of a labeling that span multiple such submodels. For example, in corpus-wide inference for POS tagging, one wants to encourage different tokens of the same word type to have the same POS tag without forcing them to do so. In general, given many submodels, one might want to add or subtract something to the score whenever submodel a picks part i and submodel b picks part j. This construction can be implemented in dual decomposition, by creating suitable projection variables selecting these parts and adding a new auxiliary model, a graphical model, which has copies of these projection variables and assigns a score to joint settings of them. Then the constraints enforcing that the projection variables have to agree with their copies can be expressed as linear constraints. This approach is popular [84, 35, 6, 108], but while attractive in its conceptual simplicity, introducing a separate submodel connecting other submodels can make inference slower, as for information to flow between two coupled submodels it needs to go through the auxiliary model, and hence it takes two dual variables updates instead of one for changes to happen.

In this chapter we show how it is possible to directly represent some parts which span across submodels and assign them dual variables, thereby omitting the need for these extraneous submodels. We are effectively "optimizing out" inference in these parts, instead of explicitly representing them as other submodels and constraints.

We will consider parts of the form $p = (\mathbf{A}_p, i, \mathbf{B}_p, j, \mathbf{c}^p)$, which assigns a penalty equal to c^p_{mn} to the score of the joint model whenever projection variable $\mathbf{A}_p\mathbf{x}_i$ is set to value m and projection variable $\mathbf{B}_p\mathbf{x}_j$ is not set to value n. These parts can be made to represent arbitrary pairwise scores, as shown in section 6.4.

The abstract form of the joint inference problem is

$$\mathbf{max}_{\cdot \mathbf{x}} \quad \sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle - \sum_P \sum_{mn} c^p_{mn} \max(0, \mathbf{A}_p \mathbf{x}_{p_i}(m) - \mathbf{B}_p \mathbf{x}_{p_j}(n)) \tag{6.1}$$

where inference is performed in many independent submodels with extra scores for the penalties mentioned above.

We can write this as a linear program by introducing the auxiliary variables z^p_{mn}

$$\begin{aligned} \mathbf{max}_{\cdot \mathbf{x},\mathbf{z}} \quad & \sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle - \sum_P \sum_{mn} c^p_{mn} z^p_{mn} \\ \mathbf{s.t.} \quad & z^p_{mn} \geq \mathbf{A}_p \mathbf{x}_{p_i}(m) - \mathbf{B}_p x_{p_j}(n) \\ & z^p_{mn} \geq 0 \end{aligned} \tag{6.2}$$

Note that for this problem to be well-defined we need c^p_{mn} to be non-negative. These two problems have the same optimal value and the same maximizing $\mathbf{x}$ variables.

We can write the Lagrangian of this problem as

$$\sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle - \sum_p \sum_{mn} \left\{ c^p_{mn} z^p_{mn} + \lambda^p_{mn}(z^p_{mn} - \mathbf{A}_p \mathbf{x}_{p_i}(m) + \mathbf{B}_p \mathbf{x}_{p_j}(n)) + \mu^p_{mn} z^p_{mn} \right\} \tag{6.3}$$

Using the stationarity KKT condition on z (that 0 is in the subgradient of the Lagrangian with respect to z) gives us

$$\mu^p_{mn} = -c^p_{mn} - \lambda^p_{mn}. \tag{6.4}$$

Substituting these we get

$$\sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle - \sum_p \sum_{mn} \left\{ \lambda^p_{mn} (\mathbf{B}_p \mathbf{x}_{p_j}(n) - \mathbf{A}_p \mathbf{x}_{p_i}(m)) \right\} \tag{6.5}$$

with the constraints that

$$0 \leq \lambda^p_{mn} \leq c^p_{mn}, \tag{6.6}$$

which follow from the non-negativity of the dual variables μ and λ for inequality constraints.

Reordering the sums,

$$\sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle + \sum_p \left\{ \sum_m \mathbf{A}_p \mathbf{x}(m) \sum_n \lambda^p_{mn} - \sum_n \mathbf{B}_p \mathbf{y}(n) \sum_m \lambda^p_{mn} \right\}, \tag{6.7}$$

Algorithm 6.1 The boxed subgradient method for optimizing our objective.

```
1: $\boldsymbol{\lambda}_{sc} \leftarrow \mathbf{0}$
2: while has not converged do
3:     for submodel $i$ do
4:         $\mathbf{x}_i^* \leftarrow \max_{\mathbf{x}_i \in \mathcal{U}_i} \left\langle \mathbf{w}_i + \sum_{p':p'_i=i} \mathbf{1}^T \boldsymbol{\lambda}^{p'T} \mathbf{A}_{p'} - \sum_{p':p'_j=i} \mathbf{1}^T \boldsymbol{\lambda}^{p'} \mathbf{B}_{p'}, \mathbf{x}_i \right\rangle$
5:     for part $p$ do
6:         $\boldsymbol{\lambda}^p(m,n) \leftarrow \max(0, \min(c_{m,n}, \boldsymbol{\lambda}^p - \eta^{(t)}(\mathbf{A}_p \mathbf{x}_i^*(m) - \mathbf{B}_p \mathbf{x}_j^*(n))))$
```

and finally

$$\sum_i \langle \mathbf{w}_i, \mathbf{x}_i \rangle + \sum_p \left\{ \mathbf{x}_i^T \mathbf{A}_p^T \mathbf{1} \boldsymbol{\lambda}^p \mathbf{1} - \mathbf{x}_j^T \mathbf{B}^T \boldsymbol{\lambda}^{pT} \mathbf{1} \right\}. \tag{6.8}$$

Now we can maximize over the primal variables to get the desired dual problem:

$$\begin{aligned} \min_{\lambda} \quad & \sum_i \max_{x_i} \left\langle \mathbf{w}_i + \sum_{p:p_i=i} \mathbf{1}^T \boldsymbol{\lambda}^{pT} \mathbf{A}_p - \sum_{p:p_j=i} \mathbf{1}^T \boldsymbol{\lambda}^p \mathbf{B}_p, \mathbf{x}_i \right\rangle \\ \textbf{s.t.} \quad & 0 \le \boldsymbol{\lambda}^p \le \mathbf{c}^p \end{aligned} \tag{6.9}$$

Note the similarity with the dual problem for dual decomposition, in equation (4.10). The similarity in functional form suggests that similar methods might be used to optimize either problem.

If one interprets the c_{mn} scores as penalizing disagreement instead of enforcing agreement (as in an equality constraint as used in dual decomposition), then the box constraints on the dual variables $\boldsymbol{\lambda}$ can be interpreted as the model trying to force the projection variables to "agree", but only up to a point.

6.2 A projected subgradient algorithm

This can be solved with the projected subgradient method. The subgradient of the dual problem w.r.t. $\lambda(i,j)$ is simply $\mathbf{Ax}(m) - \mathbf{By}(n)$. One step updates the lambdas according to these and then truncates them to fit within the constraints. Algorithm 6.1 shows the algorithm. Note the similarity to algorithm 4.1: in both cases inference in each submodel is performed once per iteration, and the dual variables are updated using a subgradient which is defined as the difference between two projection variables. The projected subgradient method has the same convergence properties as the normal subgradient method [65].

The main differences are that there are more dual variables—there is one dual variable per non-zero penalty c_{mn}—and that there is a projection step. Indeed, if one penalty c_{mn} is set to infinity this algorithm degrades gracefully to a dual decomposition algorithm which enforces the linear equality constraint $\mathbf{Ax}_i(m) = \mathbf{Bx}_j(n)$.

6.3 A block coordinate descent algorithm

In this section we derive a block coordinate descent algorithm for optimizing the dual objective from section 6.1 which has a similar structure to MPLP.

The optimality conditions state that, for each coordinate pair (m, n), one of the following conditions has to be true:

1. $\mathbf{Ax}(m) = \mathbf{By}(n)$;
2. $\mathbf{Ax}(m) = 1$ and $\lambda_{mn} = 0$; or
3. $\mathbf{Ax}(m) = 0$ and $\lambda_{mn} = c_{mn}$.

These conditions can be derived from the KKT conditions of the Lagrange multipliers for the inequality constraints for $0 \leq \lambda_{mn} \leq c_{mn}$. If none of the inequality constraints is tight their dual variables have to be 0 and hence we have condition 1; otherwise either the dual variable for the upper bound or the dual variable for the lower bound is tight, and we get condition 2 or 3.

Setting up a block coordinate descent algorithm similar to MPLP then involves the following. For two projection variables, compute the primal objective of the optimal solution, which is

$$\mathbf{max}_{mn} \;\; \mathbf{m}^A_{w_x}(m) + \mathbf{m}^B_{w_y}(n) + a_{mn}, \tag{6.10}$$

where

$$a_{mn} = -\sum_{n' \neq n} c_{mn'}. \tag{6.11}$$

Then given m^*, n^* which are maximizers of the above expression one sets $\boldsymbol{\lambda}$ such that

$$\mathbf{m}^A_{w_x}(m^*) + \sum_n \lambda(m^*, n) \;\; \geq \;\; \mathbf{m}^A_{w_x}(m) + \sum_n \lambda(m, n) \tag{6.12}$$

$$\mathbf{m}^B_{w_y}(n^*) - \sum_m \lambda(m, n^*) \;\; \geq \;\; \mathbf{m}^B_{w_y}(n) - \sum_m \lambda(m, n). \tag{6.13}$$

Satisfying these constraints ensure that the independent maximizations of the reweighted problems will have the same score as the joint maximization in equation (6.10), constructing a primal-dual optimal pair.

These constraints can be then put into any LP solver, such as Gurobi [28], which will then compute valid solution to these inequalities.

If the problem has more structure, however, it is possible to analytically compute this block coordinate descent step. More specifically, if we define an *agreement factor* to be a set of parts with scores which assign penalties c_{mn} to pairwise settings (m, n) of the projection variables, and are such that

$$c_{mn} = \begin{cases} c_m & \text{if } m = n \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

Given these constraints on the penalties there are effectively only k dual variables, as all others are constrained to be equal to 0 by equation (6.6). We refer to the dual variables then as $\lambda(m)$, and equations (6.12) and (6.13) reduce to

$$\begin{aligned} \mathbf{m}^A_{w_x}(m^*) + \lambda(m^*) &\geq \mathbf{m}^A_{w_x}(m) + \lambda(m) && (6.15) \\ \mathbf{m}^B_{w_y}(n^*) - \lambda(n^*) &\geq \mathbf{m}^B_{w_y}(m) - \lambda(m). && (6.16) \end{aligned}$$

In these inequalities, a dual variable $\lambda(m)$ for $m \neq m^*, n^*$ only appears in two inequalities, one giving it an upper bound and the other giving it a lower bound, both of which depend only on the values of $\lambda(m^*)$ and $\lambda(n^*)$:

$$\begin{aligned} \lambda(m) &\leq \mathbf{m}^A_{w_x}(m^*) + \lambda(m^*) - \mathbf{m}^A_{w_x}(m) && (6.17) \\ \lambda(m) &\geq -\mathbf{m}^B_{w_y}(n^*) + \lambda(n^*) + \mathbf{m}^B_{w_y}(m). && (6.18) \end{aligned}$$

Given these bounds, there are two possible cases:

1. If $m^* \neq n^*$, then $\lambda(m^*) = 0$ and $\lambda(n^*) = c_{n^*}$, according to the optimality conditions.

2. If $m^* = n^*$ then the optimality conditions are indifferent as to the value of $\lambda(m^*)$. One can then, for each $m \neq m^*$, use equations (6.17) and (6.18) to find an upper and lower bound on the value of $\lambda(m^*)$ which still allows the equations to be respected and $\lambda(m)$ to be within its boundaries,

$$\begin{aligned} \lambda(m^*) &\leq c_m + \mathbf{m}^B_{w_y}(m^*) - \mathbf{m}^B_{w_y}(m) && (6.19) \\ \lambda(m^*) &\geq -\mathbf{m}^A_{w_x}(m^*) + \mathbf{m}^A_{w_x}(m). && (6.20) \end{aligned}$$

   Taking the maximum lower bound and the minimum upper bound, then, is guaranteed to give a non-empty interval in which an optimal $\lambda(m^*)$ is guaranteed to lie.

After selecting the values of $\lambda(m^*)$ and $\lambda(n^*)$, equations (6.17) and (6.18) can be used to determine the set of valid values for any other $\lambda(m)$.

Algorithm 6.2 shows the block coordinate descent algorithm which was described in this section. Note that its asymptotic per-step complexity is the same as MPLP, as it

Algorithm 6.2 Box-constrained block coordinate ascent algorithm.

```
1: while has not converged do
2:     for part $p$ do
3:         $\mathbf{m}^{\mathbf{A}_{p_i}} \leftarrow \text{MaxMargs}\left(\mathbf{w}_{c_1} + \sum_{p':p'_i=c_1} \mathbf{1}^T\boldsymbol{\lambda}_{p'}^T\mathbf{A}_{p'_i} - \sum_{p':p'_2=p_1} \mathbf{1}^T\boldsymbol{\lambda}_{p'}^T\mathbf{A}_{c'_2}\right)$
4:         $\mathbf{m}^{\mathbf{A}_{p_j}} \leftarrow \text{MaxMargs}\left(\mathbf{w}_{c_2} + \sum_{p':p'_j=c_2} \mathbf{1}^T\boldsymbol{\lambda}_{p'}^T\mathbf{A}_{p'_j} - \sum_{p':p'_2=p_2} \mathbf{1}^T\boldsymbol{\lambda}_{p'}^T\mathbf{A}_{c'_2}\right)$
5:         $m^*, n^* \leftarrow \arg\max_{mn} \mathbf{m}^{\mathbf{A}_{p_1}}(m) + \mathbf{m}^{\mathbf{A}_{p_2}} - c_m\delta(m{\neq}n)$
6:         if $m^* = n^*$ then
7:             $U \leftarrow \min_{m\neq m^*} -\epsilon + \mathbf{m}^{\mathbf{A}_{p_i}}(i) - \mathbf{m}^{\mathbf{A}_{p_i}}(m^*)$
8:             $L \leftarrow \max_{m\neq m^*} \epsilon - \mathbf{m}^{\mathbf{A}_{p_j}}(m) + \mathbf{m}^{\mathbf{A}_{p_j}}(m^*)$
9:             $\boldsymbol{\lambda}_p(m^*) \leftarrow \frac{1}{2}(U+L)$
10:        else
11:            $\boldsymbol{\lambda}_p(m^*) \leftarrow 0$
12:            $\boldsymbol{\lambda}_p(n^*) \leftarrow c_{n^*}$
13:        for all $m$ such that $m \neq m^*,\ m \neq n^*$ do
14:            $U \leftarrow -\mathbf{m}^{\mathbf{A}_{p_i}}(m) + \mathbf{m}^{\mathbf{A}_{p_i}}(m^*) + \boldsymbol{\lambda}_p(m^*)$
15:            $L \leftarrow -\mathbf{m}^{\mathbf{A}_{p_j}}(n^*) + \boldsymbol{\lambda}_p(n^*) + \mathbf{m}^{\mathbf{A}_{p_j}}(m)$
16:            $\boldsymbol{\lambda}_p(m) \leftarrow \frac{1}{2}(U+L)$
```

is linear in the size of the projection variables, and the maximization in line 5 can often be performed in time linear in the domain, as if all c_m are identical the maximizing pair (m, n) is either the maximizer of the sum of the max-marginals or the maximizer of each max-marginal independently (if the c_m are not identical there can be a situation where there is another maximizer and all possibilities need to be considered).

6.4 Representing arbitrary pairwise scores

The derivation of the objective and algorithms in the previous sections were in terms of penalties, which are defined such that whenever the first variable of the part has value m and the second variable does not have value n a penalty of c_{mn} is subtracted from the overall model score. In most formulations of graphical models, however, scores are defined such that if the first variable has value m and the second has value n a score of a_{mn} is added to the overall model score. In this section we will see how to convert a score-based representation to a penalty-based representation.

Restating the definition of penalties, the value which is added to the joint model's score when the first variable has value m and the second has value n is

$$-\sum_{n'\neq n} c_{mn'}. \tag{6.21}$$

To convert between scores and penalties, then, one sets up the following linear system:

$$a_{mn} = -\sum_{n'\neq n} c_{mn'}. \tag{6.22}$$

To solve it, sum all the equations for a given value of m

$$\sum_n a_{mn} = -(k-1)\sum_n c_{mn}, \tag{6.23}$$

where k is the number of values which the first variable can take, and solve for $\sum_n c_{mn}$,

$$\sum_n c_{mn} = -\frac{1}{k-1}\sum_n a_{mn}. \tag{6.24}$$

Then sum all the equations, for a given m, for all values of n except n', and get

$$\sum_{n\neq n'} a_{mn} = -(k-1)c_{mn'} - (k-2)\sum_n c_{mn}. \tag{6.25}$$

Substituting equation (6.24) and solving for $c_{mn'}$ we get

$$c_{mn'} = \frac{(k-2)a_{mn'} - \sum_{n\neq n'} a_{mn}}{k-1}. \tag{6.26}$$

Substituting this into equation (6.22) suffices to verify correctness.

Note that for this to lead to a valid linear program we need c_{mn} to be non negative. This is always possible to ensure without changing the optimal solution by adding a constant C to all scores a_{mn}, as then we have that

$$c_{mn'} = \frac{(k-2)a_{mn'} - \sum_{n\neq n'} a_{mn} - C}{k-1}, \tag{6.27}$$

so setting C to be sufficiently negative suffices to ensure non-negativity of c_{mn}.

6.5 Application to inference in graphical models

In a similar vein to MPLP [23], which uses dual decomposition and block coordinate descent to solve the problem of inference in general high-treewidth graphical models, the linear relaxation developed in this chapter can also be applied to that problem, with some important differences.

MPLP reduces the problem of inference in graphical models to dual decomposition by letting each factor of the model be its own submodel and enforcing that for each variable all factors which touch it must agree on its value. Since these constraints are linear constraints and max-marginals are easily computed from each single-factor model, the block coordinate descent algorithm can be applied. A disadvantage of MPLP as an inference algorithm is that before convergence one has a dual solution which disagrees on the values of some variables in the graphical model. While there are strategies to obtain primal iterates which converge to a solution [61], they are defined on a smoothed variant of the algorithm.

To apply our objective to inference in graphical models with pairwise factors, however, one can let each variable be its own submodel and express all factors as additional parts. Unlike MPLP block coordinate descent can only be applied if all factors of the graphical model have the structure described in section 6.4, where $c_{mn} = 0$ if $m \neq n$. A conceptual advantage of this algorithm over MPLP is that by its nature one always has a primal feasible solution: optimization proceeds by first setting each variable to its locally optimal value, ignoring the pairwise factors in the model, and then sending dual variable messages between variables to account for the disagreement penalties specified by the model. While there are no anytime guarantees in the algorithm itself (and it might produce intermediary solutions which are worse than previous intermediary solutions) there is no ambiguity as to how to select a valid solution if one wants to terminate optimization early.

6.6 Application to corpus-wide inference in natural language processing

A natural application for many structured linear models with some simple parts which extend across the models is corpus-wide inference in natural language processing. In many settings, specially when the amount of training data is low, it is desirable to share information about many prediction tasks that one needs to perform.

For example, in named-entity recognition a given phrase is usually either a named entity or not, and moreover when it is a named entity it usually has the same type. This means that one can aggregate different occurrences of the same phrase in a corpus and add a penalty to the model for when those phrases don't share the same type or whether they are named entities or not [35, 6]. This effectively allows one to aggregate information from all contexts in which the phrase appears to make a decision, instead of making each decision independently.

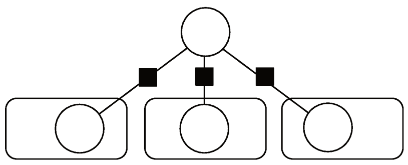


Figure 6.1: A single consensus structure. Rounded rectangles represent sentence-level MAP problems. Dark squares are agreement parts. The bottom circles are single variables in the sentences. The consensus variable is on top.

Likewise in part-of-speech tagging of English and romance languages most rare word types only appear in a corpus with a single part-of-speech tag, and which tag that is might not be obvious from context in all occurrences, so again it is desirable to pool together information from across these occurrences of the rare word type [84].

Of course, a joint model is not the only solution for these problems. State of the art named entity detection systems, for example, can use rather intricate feature sharing schemes, where features are pooled across different occurrences of similar word types which are nearby in a corpus [76]. These schemes to tend to be more brittle than joint inference, and to require more engineering to work equivalently well.

6.6.1 A structure for corpus-wide inference models

In this section we will follow Rush et al [84] and describe a general setup common to corpus-wide inference problems.

In most such problems the specific submodels are already mostly accurate, and the goal of joint inference is to tweak the submodels' decisions in places where information from the training data alone might not be sufficient to ensure good answers. The way the tweaking is done is usually to achieve consensus: the modeller specifies some sets of similar data points and adds parts to the joint model to encourage data points in the same sets to have similar labelings.

There are many ways to achieve this. One could, for example, add parts to the joint model encouraging agreement between all pairs of data points in the same consensus set, but this has the disadvantages that, when the set have different sizes, it becomes hard to choose the scores of the consensus parts in a way that doesn't over count the importance of agreement in the larger sets or undercounts in the smaller sets; at the same time most inference methods scale in complexity at least linearly with the number of parts, and hence having the number of parts in a set proportional to the square of the number of elements in it is undesirable.

Another way is to add a consensus variable to the global model for each agreement

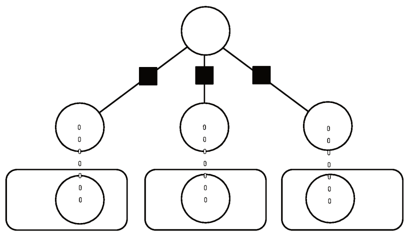


Figure 6.2: The variable-copying version of Fig. 6.1 used by dual decomposition, where dashed lines denote equality constraints.

set, and add parts encouraging each data point to agree with the value of the consensus variable. This leads to a structure, for each set, similar to Figure 6.1. The number of agreement parts scales linearly with the size of each consensus set.

Note that to implement this framework with ordinary dual decomposition one needs to define auxiliary submodels for the consensus structure, define projection variables both in the auxiliary submodels and the normal submodels, and apply either the subgradient method or MPLP. Figure 6.2 shows what the same structure looks like with variable copies. The main advantage of the formulation described in this chapter when compared with the standard formulation of corpus-wide inference with variable copies is that, by sending messages directly across submodels, the inference process should be more efficient. This is validated experimentally in section 6.8.

6.7 Related work

Decoding individual sentences using corpus-level information has proven to be useful in various settings, particularly those involving domain adaptation or limited amounts of labeled training data. Examples of methods for sharing information between sentences include graph-based semi-supervised learning [95], using Gibbs sampling or loopy belief propagation for inference in models that include skip-chain factors [20, 96, 35], using sentence-level models to extract context information and using an LP-solver to directly solved the constrained inference problem [82], and introducing a "global MRF" connecting sentence models, where inference is done by copying variables and using dual decomposition [84, 6, 35].

A further application of dual decomposition has been to avoid cascading errors in an NLP pipeline by instead making all decisions jointly [79]. Dual decomposition has also been applied to a wide variety of NLP problems that do not perform global inference, but instead allow maximization of sentence-level score functions that decompose into sub-functions for which maximization is tractable for each in isolation [39, 71, 83].

There have been limited applications of block coordinate descent for solving the dual decomposition objective in NLP. However, we draw on various algorithmic ideas from the machine learning literature. Our Box-BCD and Box-Subgradient algorithms are examples of efficiently handling factors that have a specific structure [16, 50]. There is also a precedent of constructing message passing schemes by doing block coordinate descent in a dual problem [105, 37, 23, 94, 86, 77].

There is much work in natural language processing using joint inference to achieve good results [20, 82, 39, 79] in individual prediction tasks instead of corpus-wide tasks, with success. Most of these models, however, found no need for parts which span multiple submodels.

6.8 Experiments

In this section we present experimental evidence that optimizing the dual objective presented in this chapter instead of the usual construction based on dual decomposition and variable copies leads to faster convergence, both in terms of the value of the dual problem and in terms of accuracy of the downstream joint model. Measuring accuracy is important because it is conceivable that formulating the dual problem differently could lead to faster decrease in the dual objective and yet slower downstream convergence; in our experiments both metrics are, however, correlated.

We mirror the experimental setup of Rush et al [84] for both tagging and parsing. We use the formulation of the corpus-wide inference problem described in section 6.6.1.

We consider two experimental tasks: part-of-speech tagging and dependency parsing. In both cases we compare algorithms using our proposed dual objective with algorithms which use the variable-copying structure depicted in Figure 6.2.

We compare the following algorithms:

- Subgradient: algorithm 4.1;
- MPLP: algorithm 4.2;
- Box-Subgradient: algorithm 6.1; and
- Box-BCD: algorithm 6.2.

We analyze each algorithm in terms of how quickly accuracy increases and the dual objective decreases. Considering both is important, since they can be uncorrelated, due to inherent error in the model's fit to the data. This is why accuracy curves in the following sections sometimes peak when the dual objective isn't minimized.

Rather than using wall-clock time as the benchmark against which we measure the speed of the algorithms, we compare against the total number of calls to inference in sentences. After the first pass through the corpus, which is shared by all algorithms, we only perform inference on sentence for which some dual variable changed. We normalize the total number of inference operations by the size of the corpus for ease of comparison across experiments. This assumes all calls to inference are equally expensive, which is not strictly true, as subsequent inference in the sentences can be accelerated by pruning and caching.

Considering iterations rather than wall clock time is advantageous because it removes the effect of implementation-level details when comparing the algorithms. While more engineering can conceivably make calls to inference faster, the optimization algorithms will still require the same number of calls to inference to converge. Nevertheless, counting iterations is a reliable proxy for inference time: less than 5% of the total runtime in any of our experiments was spent not doing inference in the sentences.

Our primary motivation for measuring inference calls instead of wall clock time, however, is that it allows us to be generous to the baseline algorithms we seek to outperform. Though 'optimizing out' the consensus structures avoids the cost of performing MAP in these structures, we ignore this cost in order to give optimistic cost measurements for our baselines: Subgradient and MPLP. Second, we seek to demonstrate that our coordinate descent methods are effective alternatives to the subgradient methods, and thus we should properly account for the extra inference required by MPLP and Box-BCD to compute max-marginals. We assign a pessimistic multiplier of two for all inference calls that require max-marginals. In practice this is an exaggeration, especially if the state space is pruned, local scores are cached, and computing the model's scores is expensive.

There are various hyperparameters to tune, and we choose values in the following order. First, we choose basic model hyperparameters such as regularization weights to maximize accuracy for isolated sentence-level inference. Then, we chose the weights of the disagreement factor penalties on the test set independently for each experiment, by maximizing final corpus-wide inference accuracy on the test set. Finally, for each of subgradient methods we choose the best step size schedule in hindsight from the following functional forms: $T^{-1}, T^{-\frac{1}{2}}$, and 0.9^T, where T indicates either the current iteration or the number of iterations in which the dual objective has increased so far, multiplied by logarithmically spaced factors ranging from 10^{-3} to 10. These schedules subsume standard ones used in the machine learning literature, including those used by Rush et al (2012). There was no single step size schedule which was among the best in all problems we've tried.

We selected the best step size schedules in hindsight for two reasons. First, for most individual schedules the relative performance of the two subgradient methods is similar.

Secondly, even choosing the best step size schedule in hindsight the block coordinate descent methods outperform or are competitive with the subgradient methods, highlighting their usefulness.

Even though we used the best stepsize schedule in hindsight, qualitative observations are preserved for most individual schedules; that is Box-Subgradient outperformed Subgradient on most specific schedules.

Likewise, for experiments reporting wall-clock time the conclusions are similar.

Finally, since max-marginals are linear, for both MPLP and Box-BCD it is not necessary to run inference before an update if the last update only touched the dual variable corresponding to that variable. This allows one to, if no sentence appears more than once in a consensus set, do more than one pass in the variables in the same consensus set before proceeding to the next one. We do this in the experiments, doing up to 10 passes in each consensus set, but while it improves results a bit it doesn't lead to qualitative differences versus simply iterating over each consensus set once.

All the code is implemented in the scala programming language [69] and the timing experiments are on a circa 2010 Macbook Pro. The code uses the Factorie library for graphical models [54], but all relevant code for inference was reimplemented as efficiently as possible for this chapter.

6.8.1 The projection variables

There are three types of projection variables used in these experiments: the POS tag of a given token in a linear chain model, the POS tag of a token's parent in a first-order dependency parsing model, and a projection variable which maps the POS tag of a given token into the set of universal POS tags. We will go over each of these in turn.

Constructing a projection matrix $\mathbf{A}$ which projects a parts vector $\mathbf{x}$ from a linear chain graphical model, which has a part per setting of every two neighboring variables, into a projection variable representing the POS tag of a given token is straightforward. Since each row of $\mathbf{A}$ will correspond to one particular POS tag for each token and only one part with that POS tag can be active in any valid solution, each row of $\mathbf{A}$ is 1 for all parts which assign that POS tag to that token, and 0 everywhere else.

Similarly, in first-order projective dependency parsing, as seen in section 4.3.2, each possible assignment of a parent for a token is represented as either a left-incomplete or a right-incomplete span, and each token will only be a child on one of these assignments. Since the part-of-speech of each token is known, a row of a matrix $\mathbf{A}$ which corresponds to the part-of-speech tag of a parent of a specific token is nonzero in all the left-incomplete or right-incomplete spans with that token as a child and a token with the right part-of-speech as a parent. Since these are mutually exclusive this defines a valid projection variable.

For the coarse POS tag projection variable we want a matrix $\mathbf{B}$ such that if $\mathbf{Ax}$ is a projection variable representing a POS tag then $\mathbf{BAx}$ represents the coarse version of that POS tag. This is possible construct since each fine-grained POS tag maps to a single coarse POS tag [73].

Given these matrices we can perform subgradient and block coordinate descent updates as specified by the relevant algorithms.

6.8.2 The modelling parts

In both experiments one has projection variables representing POS tags—of tokens with the same word type, in POS tagging, and of the parents of tokens in similar contexts, in parsing—and the goal is to encourage all POS tags in a given consensus set to agree.

This is implemented by adding a consensus variable for each such consensus set and creating two sets of parts per projection variable: one which penalizes the variable and the consensus variable for having different part-of-speech tags and one which penalizes different coarse part-of-speech tags [73]. Since max-marginals can be efficiently computed from both linear-chain CRFs and projective dependency parsing models, and both these parts are agreement parts, as described in section 6.3, we can apply the subgradient and block coordinate descent methods for optimizing the dual objective.

6.8.3 POS Tagging

In the part-of-speech tagging experiment, following Rush et al [84], we create a consensus set for each rare word type observed in the test set, encouraging all tokens of that word type to have similar part-of-speech tags.

We use a bigram linear-chain conditional random field tagger [41], trained to optimize ℓ_2-regularized maximum likelihood with L-BFGS. Following Rush et al [84], we perform semi-supervised training experiments, learning models on subsets of 50, 100, 200, and 500 sentences from the first chapter of the Penn treebank and testing on the Penn treebank chapters 22-24 [48]. Both are are consistently comparable to those for the Stanford tagger [100] reported in Rush et al [84], Table 4.

Figure 6.3 shows tagging accuracy and dual objective as a function of the number of times inference is performed in the subproblems, for the WSJ-200 experiment, and Table 6.1 shows how much inference is necessary to reach various percentile gains in accuracy and percentile reductions in the dual objective. Overall, Box-BCD is at least 5 times less costly than Subgradient, and MPLP also works well in both metrics.

In tables 6.2, 6.3, and 6.4 we present the normalized number of runs of inference in order to achieve certain quantiles for the dual objective and accuracy for the POS

Table 6.1: Normalized number of inference runs for each algorithm to attain quantiles of the best dual solution in the WSJ-200 tagging experiment. Best results are in bold. If a quantile was not reached during 100 iterations, we show 'na'.

Accuracy quantile	80%	85%	90%	95%
Subgradient	70	92	na	na
MPLP	22	23	25	30
Box-Subgradient	20	35	40	54
Box-BCD	**8**	**9**	**10**	**10**
Dual Quantile	80%	85%	90%	95%
Subgradient	24	34	56	na
MPLP	21	22	23	35
Box-Subgradient	30	35	40	54
Box-BCD	**7**	**7**	**8**	**9**

experiments trained on 50, 100, and 500 sentences from the first section of the WSJ. They are similar to the results on WSJ-200. Table 6.5 shows the gains in accuracy we obtain from doing corpus-wide inference vs. isolated inference.

6.8.4 Parsing

Following Rush et al [84], each set of POS tags around a token defines a context, and identical contexts are encouraged to have dependency parents with similar POS tags by introducing various consensus structures. We depart from their setup by not using additional corpora of unlabeled data at test time, which might have improved accuracy, but does not change the overall characteristics of the optimization problem. The domain adaptation experiments use the Wall Street Journal section of the Penn treebank (WSJ)[48] and Question treebank (QTB) [30]. We use the same parts as employed in the POS tagging experiment.

This experiment employs a first-order projective arc-factored parser [56] using dynamic programming for inference, which does not perform as well as the second-order projective MST used by Rush et al [84].

Table 6.2: The results of the WSJ-50 tagging experiment.

Accuracy quantile	80%	85%	90%	95%
Subgradient	67	96	108	122
MPLP	40	45	47	57
Box-Subgradient	58	63	81	108
Box-BCD	**12**	**14**	**15**	**27**
Dual Quantile	80%	85%	90%	95%
Subgradient	60	100	102	130
MPLP	44	47	50	75
Box-Subgradient	40	56	105	112
Box-BCD	**12**	**13**	**14**	**19**

Table 6.3: The results of the WSJ-100 tagging experiment.

Accuracy quantile	80%	85%	90%	95%
Subgradient	81	88	105	110
MPLP	29	32	37	42
Box-Subgradient	52	67	71	86
Box-BCD	**11**	**13**	**15**	**19**
Dual Quantile	80%	85%	90%	95%
Subgradient	67	74	76	102
MPLP	29	30	32	34
Box-Subgradient	29	57	65	72
Box-BCD	**9**	**10**	**10**	**12**

Table 6.4: The results of the WSJ-500 tagging experiment. The rows are the total times inference is run to achieve each quantile, normalized by the size of the corpus.

Accuracy quantile	80%	85%	90%	95%
Subgradient	14	18	23	26
MPLP	13	13	13	14
Box-Subgradient	9	9	12	16
Box-BCD	**5**	**5**	**5**	**6**
Dual Quantile	80%	85%	90%	95%
Subgradient	12	16	25	32
MPLP	12	12	13	13
Box-Subgradient	7	9	15	21
Box-BCD	**4**	**5**	**5**	**5**

Table 6.5: Comparing corpus-wide inference vs. isolated inference for the POS experiments.

	Isolated Inference	Corpus-Wide Inference	Accuracy Gain
WSJ-50	79.0	80.1	1.1
WSJ-100	84.8	86.4	1.6
WSJ-200	88.4	89.5	1.1
WSJ-500	91.8	92.1	0.3

In the WSJ to QTB experiment the model is trained on the WSJ standard training split (chapters 1 to 18) and tested on the QTB, while in the QTB to WSJ experiment we train on the QTB and test on the standard WSJ test set (chapters 22 to 24). Both models were trained with stochastic gradient descent using AdaGrad regularized dual averaging [15], and hyperparameters were tuned to maximize test-set accuracy.

Table 6.6 summarizes the experimental results. In the WSJ to QTB experiment the box-constrained algorithms uniformly outperform their respective baselines, and the parameter-free MPLP algorithms perform almost as well as the best subgradient algorithm, even while accounting for the higher cost of obtaining max-marginals. For the QTB to WSJ experiment we were unable to reproduce accuracy increases as reported in Rush et al [84]; none of the optimization algorithms managed to improve the accuracy for any setting of the penalties. This is probably due to our simpler parser and not using additional data. However, regarding dual optimization, each MPLP method outperforms the corresponding subgradient method, and the boxed algorithms outperform the ones which use variable copying.

Figure 6.4 and figure 6.5 show the accuracy and dual objective across runs of the WSJ-to-QTB parsing experiments.

Figure 6.6 and figure 6.7 show the accuracy and dual objective across runs of the QTB-to-WSJ parsing experiments.

Table 6.7 shows the accuracies for the parsing experiments before and after joint inference. As noted earlier, substantially smaller increases were observed than in Rush et al [84], mostly because we use a simpler model (a first-order parser instead of a second-order parser) and we don't use an additional unlabeled corpus.

6.9 Conclusions

In this chapter we presented a new linear programming relaxation for joint inference. It extends the common paradigm of dual decomposition, which allows for independent submodels coupled by constraints, to also allow for scores to be assigned pairs of variables in different submodels. The extra flexibility comes at very little cost in complexity, and a projected subgradient algorithm for this dual objective is very similar to one for dual decomposition. We also present a block coordinate descent algorithm for optimizing this dual objective in the case of parts which just encourage variables to agree.

We validate experimentally this objective by showing that optimizing it leads to faster gains in accuracy or decrease in dual value than optimizing an equivalent construction using dual decomposition which defines auxiliary models for the parts which we can handle directly.

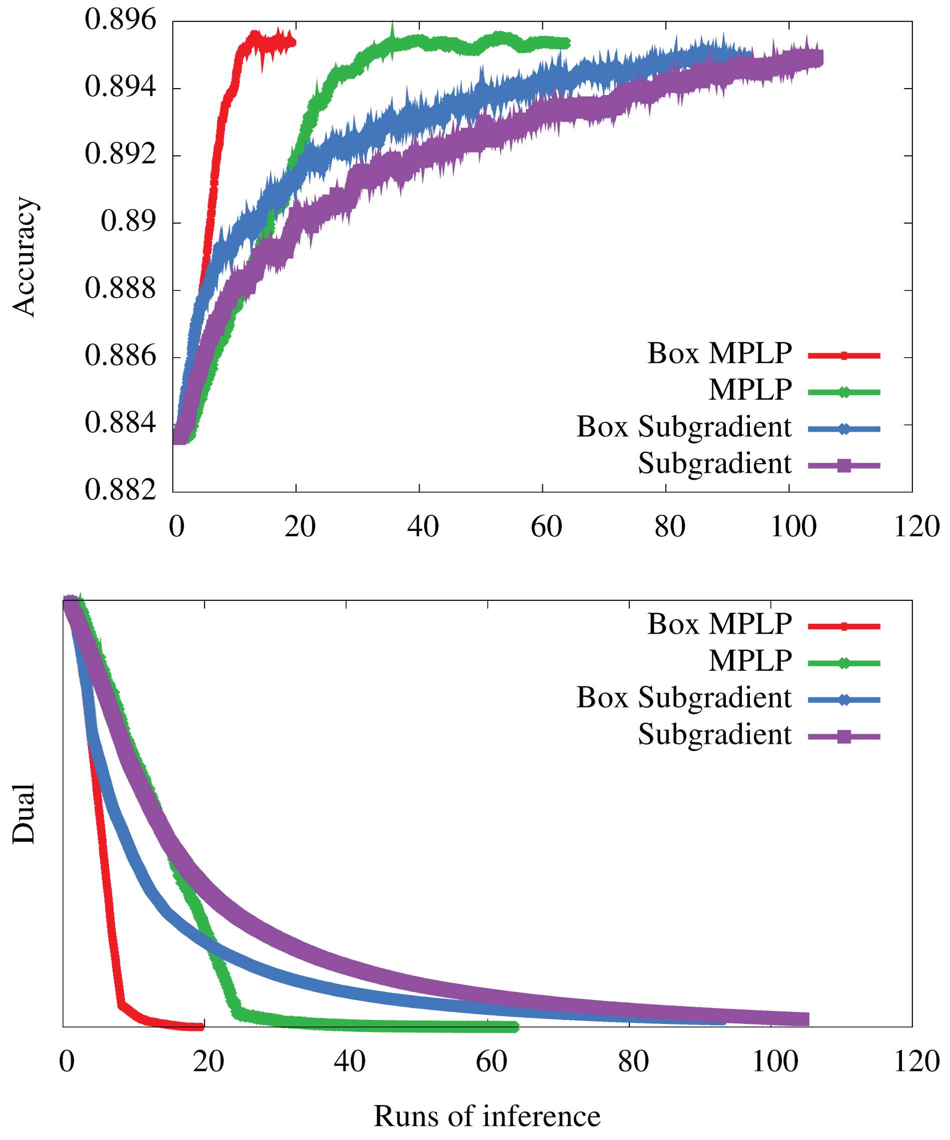


Figure 6.3: Accuracy (top) and dual objective (bottom) v.s. runs of sentence-level inference for the WSJ-200 tagging experiment.

Table 6.6: Normalized number of runs of inference for each algorithm to attain quantiles of the best solution in the parsing experiments. Quantiles not reached during 100 iterations are labelled ‘na’.

WSJ to QTB				
Accuracy quantile	80%	85%	90%	95%
Subgradient	4.1	4.3	5.2	6.1
MPLP	4.3	4.3	4.3	‘na’
Box-Subgradient	**2.1**	**2.1**	**2.4**	**2.8**
Box-BCD	2.6	2.8	3	‘na’
Dual quantile	80%	85%	90%	95%
Subgradient	3.0	3.2	3.4	3.9
MPLP	4.2	4.4	4.9	4.9
Box-Subgradient	**1.6**	**1.7**	**1.8**	**2.0**
Box-BCD	2.5	2.5	2.5	2.6
QTB to WSJ				
Dual quantile	80%	85%	90%	95 %
Subgradient	15	16	18	22
MPLP	14	15	16	17
Box-Subgradient	8.1	9.2	10	12
Box-BCD	**6.9**	**7.4**	**7.9**	**8.6**

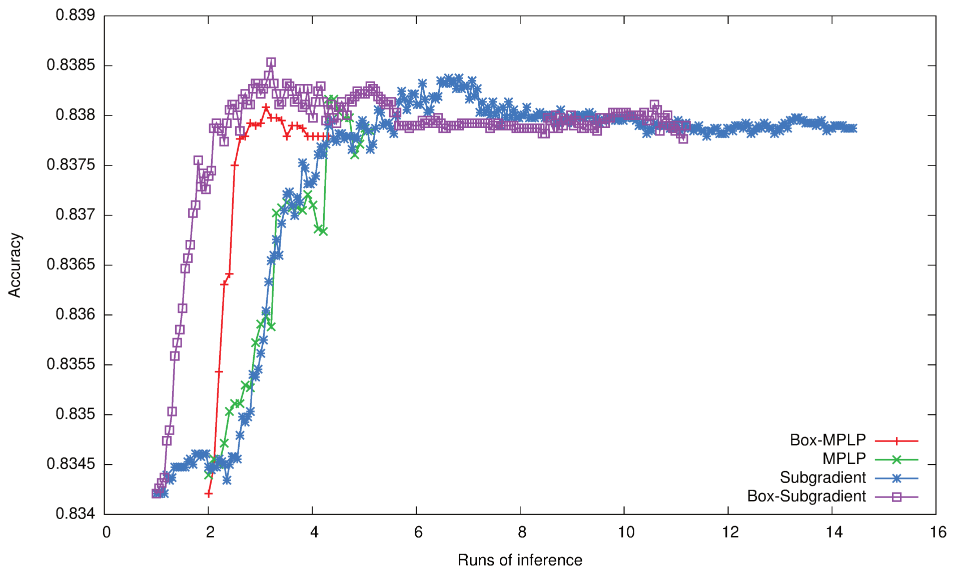


Figure 6.4: Accuracy versus normalized number of runs of inference in individual sentences for the WSJ to QTB parsing experiment.

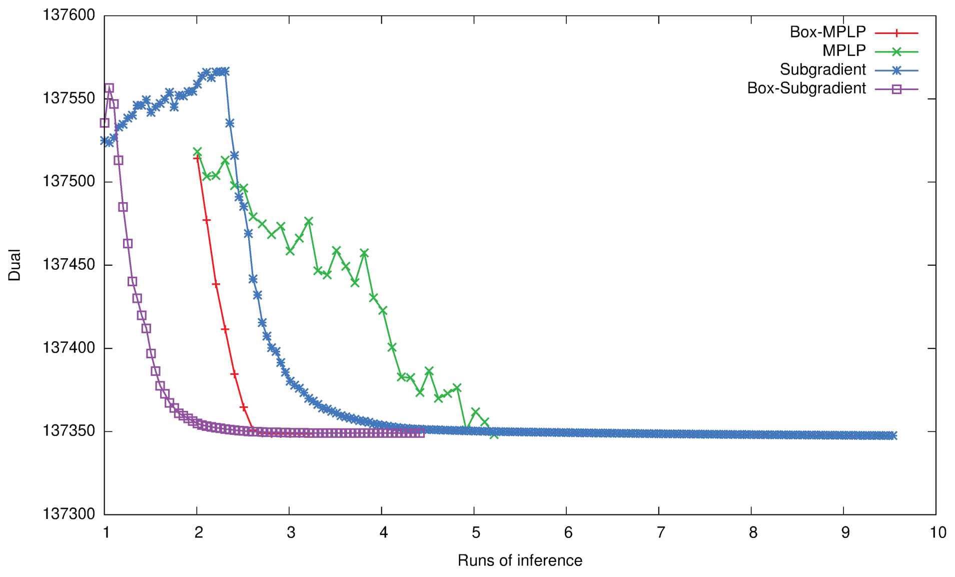


Figure 6.5: Value of the dual objective versus normalized number runs of inference in individual sentences for the WSJ to QTB parsing experiment.

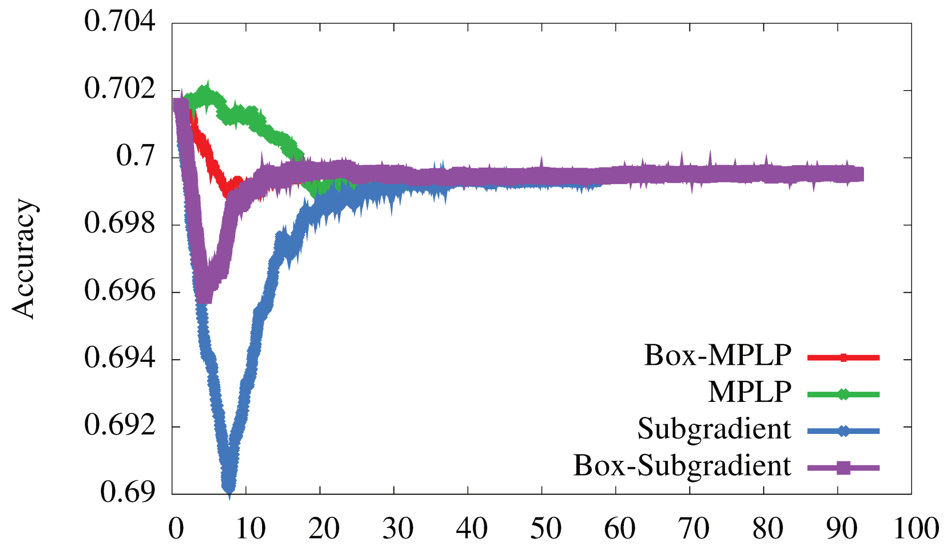


Figure 6.6: Accuracy versus normalized number of runs of inference in individual sentences for the QTB to WSJ parsing experiment.

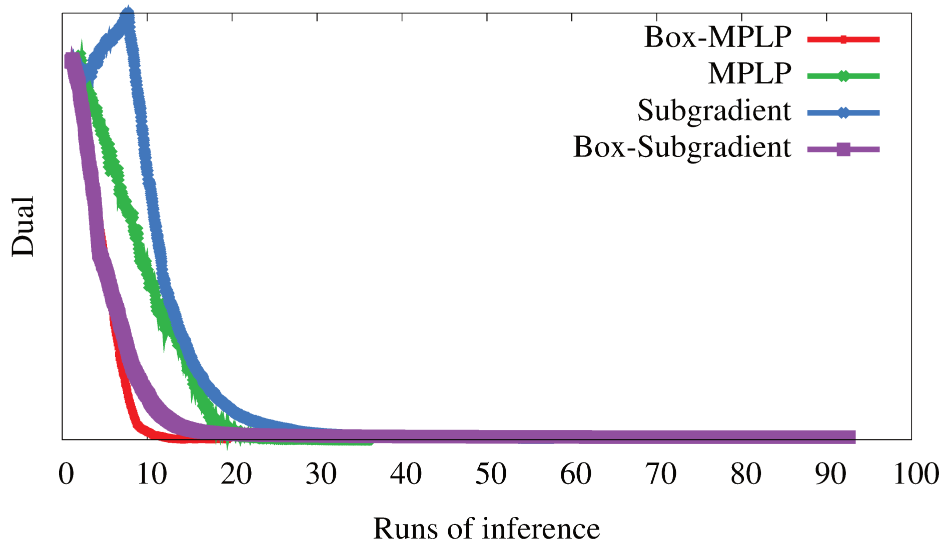


Figure 6.7: Value of the dual objective versus normalized number runs of inference in individual sentences for the QTB to WSJ parsing experiment.

Table 6.7: Unlabeled attachment scores for the parsing experiments.

Experiment	Isolated Inference	Joint Inference
WSJ to QTB	83.43	83.78
QTB to WSJ	70.14	69.93

Chapter 7

Conclusions and future work

In this thesis we explored the relationships between inference in structured linear models, linear programming, and combinatorial algorithms. Chapter 5 showed how an application of a technique from linear programming can speed up the common dynamic programming algorithms for inference in models for which inference is known to be tractable. Chapter 6, on the other hand, showed a new LP formulation for joint inference, more general than dual decomposition, allowing the simple combinatorial algorithms for inference in tractable models to be applied to a broader range of problems.

There are many possible avenues for future work. Generally, chapters 5 and 6 focused on improving the runtime of exact inference methods for structured linear models. In practice, however, for many models of interest, such as grid models in computer vision, there are no practical exact inference methods, and practitioners use a variety of approximate algorithms. It is an interesting open question if the techniques developed in this thesis will generalize to approximate inference methods.

Similarly, there are variational formulations of marginal inference which are also expressable as optimization problems with linear constraints [106], and hence some notion of column generation might be applicable to marginal inference.

While inference is a fundamental task in structured linear models, a model is only as useful as its parameters, and the connections between learning and linear programming haven't been as widely explored as the connections between inference and linear programming. The perceptron algorithm can be seen as solving a linear feasibility problem, whose constraints are specified by the training examples [25], and general linear problems can be reduced to linear feasibility problems. It might be possible, then, to directly use algorithms such as the perceptron to solve inference and learning simultaneously, or alternatively to approach learning as another MAP inference problem solvable with message-passing.

Bibliography

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, and K. Weihe. Network flows: theory, algorithms and applications. *ZOR-Methods and Models of Operations Research*, 41(3):252–254, 1995.

[2] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[3] David Belanger, Alexandre Passos, Sebastian Riedel, and Andrew McCallum. MAP inference in chains using column generation. In *Advances in Neural Information Processing Systems*, pages 1853–1861, 2012.

[4] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250. ACM, 2008.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[6] Hai Leong Chieu and Loo-Nin Teow. Combining local and non-local information with dual decomposition for named entity recognition from text. In *15th International Conference on Information Fusion (FUSION), 2012*, pages 231–238. IEEE, 2012.

[7] Jinho D Choi and Martha Palmer. Getting the most out of transition-based dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 687–692, 2011.

[8] Jinho D Choi and Martha Palmer. Fast and robust part-of-speech tagging using dynamic model selection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 363–367. Association for Computational Linguistics, 2012.

[9] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

[10] Michael Collins. Head-driven statistical models for natural language parsing. *Computational linguistics*, 29(4):589–637, 2003.

[11] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.

[12] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[13] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[14] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2010.

[16] John Duchi, Daniel Tarlow, Gal Elidan, and Daphne Koller. Using combinatorial optimization within max-product belief propagation. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 369–376. MIT Press, Cambridge, MA, 2007.

[17] Jason Eisner and Giorgio Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 457–464. Association for Computational Linguistics, 1999.

[18] Roberto Esposito and Daniele P. Radicioni. Carpediem: an algorithm for the fast evaluation of SSL classifiers. In *Proceedings of the 24th international conference on Machine learning*, pages 257–264. ACM, 2007.

[19] David Ferrucci. Build Watson: an overview of DeepQA for the Jeopardy! challenge. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 1–2. ACM, 2010.

[20] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[21] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete applied mathematics*, 42(2):177–201, 1993.

[22] Kevin Gimpel and Noah A Smith. Softmax-margin crfs: Training log-linear models with cost functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 733–736. Association for Computational Linguistics, 2010.

[23] Amir Globerson and Tommi Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in neural information processing systems*, 2007.

[24] Yoav Goldberg and Joakim Nivre. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics*, 2012.

[25] Elad Hazan and Zohar Karnin. A polylog pivot steps simplex algorithm for classification. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 638–646, 2012.

[26] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086. Association for Computational Linguistics, 2010.

[27] ILOG, Inc. ILOG CPLEX: High-performance software for mathematical programming and optimization, 2006.

[28] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2012.

[29] Heng Ji and Ralph Grishman. Knowledge base population: Successful approaches and challenges. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1148–1158, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[30] John Judge, Aoife Cahill, and Josef Van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 497–504. Association for Computational Linguistics, 2006.

[31] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[32] N. Kaji, Y. Fujiwara, N. Yoshinaga, and M. Kitsuregawa. Efficient staggered decoding for sequence labeling. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 485–494. Association for Computational Linguistics, 2010.

[33] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[34] Sheldon Klein and Robert F Simmons. A computational approach to grammatical coding of english words. *Journal of the ACM (JACM)*, 10(3):334–347, 1963.

[35] Peter Kluegl, Martin Toepfer, Florian Lemmerich, Andreas Hotho, and Frank Puppe. Collective information extraction with context-specific consistencies. *Machine Learning and Knowledge Discovery in Databases*, pages 728–743, 2012.

[36] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

[37] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *IEEE 11th International Conference on Computer Vision (ICCV)*, pages 1–8. IEEE, 2007.

[38] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics, 2010.

[39] Terry Koo, Alexander M Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298. Association for Computational Linguistics, 2010.

[40] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Advances in neural information processing systems*, pages 785–792, 2007.

[41] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, 2001.

[42] Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

[43] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics, 2011.

[44] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[45] Ben London, Bert Huang, Ben Taskar, and Lise Getoor. Collective stability in structured prediction: Generalization from one example. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[46] Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *Computational Linguistics and Intelligent Text Processing*, pages 171–189. Springer, 2011.

[47] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[48] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

[49] R. Kipp Martin, Ronald L. Rardin, and Brian A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations Research*, 38(1):pp. 127–138, 1990.

[50] André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44. Association for Computational Linguistics, 2010.

[51] Julian McAuley and Tibério Caetano. Exploiting data-independence for fast belief-propagation. In *International Conference on Machine Learning 2010*, volume 767, page 774, 2010.

[52] Julian. McAuley and Tibério Caetano. Faster algorithms for max-product message-passing. *Journal of Machine Learning Research*, 12:1349–1388, 2011.

[53] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 188–191. Association for Computational Linguistics, 2003.

[54] Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, pages 1249–1257, 2009.

[55] John J McCarthy. *A thematic guide to Optimality Theory.* Cambridge University Press, 2002.

[56] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98. Association for Computational Linguistics, 2005.

[57] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proceedings of EACL*, volume 6, pages 81–88, 2006.

[58] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[59] Ryan McDonald and Giorgio Satta. On the complexity of non-projective data-driven dependency parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, pages 121–132. Association for Computational Linguistics, 2007.

[60] Paul McNamee, Hoa Trang Dang, Heather Simpson, Patrick Schone, and Stephanie M. Strassel. An evaluation of technologies for knowledge base population. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[61] Ofer Meshi, Tommi Jaakkola, and Amir Globerson. Convergence rate analysis of MAP coordinate minimization algorithms. In P. Bartlett, F.C.N. Pereira, C.J.C.

Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 3023–3031, 2012.

[62] John E Mitchell. Cutting plane methods and subgradient methods. *Tutorials in Operations Research*, pages 34–61, 2009.

[63] Mark-Jan Nederhof. Weighted deductive parsing and Knuth's algorithm. *Computational Linguistics*, 29(1):135–143, 2003.

[64] Arkadići Semenovich Nemirovskići and DB Yudin. *Problem complexity and method efficiency in optimization*. Wiley (Chichester and New York), 1983.

[65] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2003.

[66] Andrew Y Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[67] Joakim Nivre. Dependency grammar and dependency parsing. *MSI report*, 5133(1959):1–32, 2005.

[68] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kubler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95, 2007.

[69] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: a comprehensive step-by-step guide*. Artima Inc, 2008.

[70] Alexandre Passos, David Belanger, Jacques Wainer, and Andrew McCallum. Linear programming relaxations for joint inference. Submitted to the Journal of Machine Learning Research.

[71] Michael Paul and Jason Eisner. Implicitly intersecting weighted automata using dual decomposition. In *Proceedings of NAACL-HLT*, pages 232–242, Montreal, June 2012.

[72] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 433–440. Association for Computational Linguistics, 2006.

[73] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[74] Alan Prince and Paul Smolensky. *Optimality Theory: Constraint interaction in generative grammar*. Wiley-Blackwell, 2008.

[75] Christopher Raphael. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(12):1379–1390, 2001.

[76] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics, 2009.

[77] Pradeep Ravikumar, Alekh Agarwal, and Martin J Wainwright. Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *The Journal of Machine Learning Research*, 11:1043–1080, 2010.

[78] Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for Markov logic. *Proceedings of UAI 2008*, pages 468–475, 2008.

[79] Sebastian Riedel and Andrew McCallum. Fast and robust joint models for biomedical event extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1–12. Association for Computational Linguistics, 2011.

[80] Sebastian Riedel, David Smith, and Andrew McCallum. Parse, price and cut—delayed column and row generation for graph based parsers. *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '12)*, 2012.

[81] Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 627–635, 2011.

[82] Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the 8th Conference on Computational*

Natural Language Learning (CoNLL' 04), pages 1–8. Association for Computational Linguistics, 2004.

[83] Alexander M. Rush and Michael Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artifficial Intelligence Research (JAIR)*, 45:305–362, 2012.

[84] Alexander M Rush, Roi Reichart, Michael Collins, and Amir Globerson. Improved parsing and pos tagging using inter-sentence consistency constraints. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1434–1444, 2012.

[85] Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1–11. Association for Computational Linguistics, 2010.

[86] Alex Schwing, Tamir Hazan, Marc Pollefeys, and Raquel Urtasun. Globally convergent dual MAP LP relaxation solvers using Fenchel-Young margins. In *Advances in Neural Information Processing Systems 25*, pages 2393–2401, 2012.

[87] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003.

[88] Amit Singhal. Introducing the knowledge graph: things, not strings. Official Google Blog, 2012.

[89] Noah A Smith. Linguistic structure prediction. *Synthesis Lectures on Human Language Technologies*, 4(2):1–274, 2011.

[90] Richard Socher, Cliff C. Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2011.

[91] David Sontag. *Approximate Inference in Graphical Models using LP Relaxations*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2010.

[92] David Sontag, Amir Globerson, and Tommi Jaakkola. Introduction to dual decomposition for inference. In Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, editors, *Optimization for Machine Learning*. MIT Press, 2011.

[93] David Sontag and Tommi Jaakkola. New outer bounds on the marginal polytope. In *Advances in Neural Information Processing Systems*, 2007.

[94] David Sontag and Tommi Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[95] Amarnag Subramanya, Slav Petrov, and Fernando Pereira. Efficient graph-based semi-supervised learning of structured tagging models. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 167–176. Association for Computational Linguistics, 2010.

[96] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. In *Introduction to statistical relational learning*. MIT Press, 2006.

[97] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Advances in neural information processing systems*, volume 16, 2003.

[98] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[99] Erik F Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 142–147. Association for Computational Linguistics, 2003.

[100] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[101] Kristina Toutanova and Christopher D Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000.

[102] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces.

In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[103] Vladimir N Vapnik. *Statistical learning theory*. Wiley, 1998.

[104] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[105] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. MAP estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005.

[106] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[107] Huayan Wang and Daphne Koller. A fast and exact energy minimization algorithm for cycle MRFs. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[108] Mengqiu Wang, Wanxiang Che, and Christopher D Manning. Joint word alignment and bilingual named entity recognition using dual decomposition. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2013.

[109] Michael Wick, Khashayar Rohanimanesh, Kedar Bellare, Aron Culotta, and Andrew McCallum. Samplerank: Training factor graphs with atomic gradients. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 777–784, New York, NY, USA, June 2011. ACM.