Lagrangian Based Approaches for Lexicalized Tree Adjoining Grammar Parsing

Caio Corro

Université Paris 13

École doctorale Galilée

Lagrangian Based Approaches for Lexicalized Tree Adjoining Grammar Parsing

Présentée par **Caio Filippo Corro**
et soutenue publiquement le 9 mars 2018

Adeline Nazarenko	Professeure, Université Paris 13	Directrice
Joseph Le Roux	Maître de conférences, Université Paris 13	Encadrant
Alexis Nasr	Professeur, Aix-Marseille Université	Rapporteur
Leo Liberti	Directeur de recherche, CNRS	Rapporteur
Isabelle Tellier	Professeure, Université Paris 3	Présidente du jury
André Martins	Research scientist, Unbabel	Examinateur
Roberto Wolfler Calvo	Professeur, Université Paris 13	Examinateur
Benoît Crabbé	Maître de conférences, Université Paris Diderot	Invité du jury
Mathieu Lacroix	Maître de conférences, Université Paris 13	Invité du jury

LIPN – UMR CNRS 7030 / ID 0148
Université Paris 13
99, avenue J.B. Clément
93430 Villetaneuse

Abstract

In linguistics and Natural Language Processing (NLP), syntax is the study of the structure of sentences in a given language. Two approaches have mainly been considered to describe them: dependency structures and phrase-structures. A dependency links a pair of words together with its relation type whereas a phrase-structure describe a sentence by means of a hierarchy of word sets called constituents. In this thesis, we focus on phrase-structure parsing, that is the computation of the constituency structure of a given sentence. Context-Free Grammars (CFGs) have been widely adopted by the NLP community due to their simplicity and the low complexity of their parsing algorithms. However, CFGs are too limited in order to describe all phenomena observed in natural language structures. Therefore, Lexicalized Tree Adjoining Grammars (LTAGs) have been widely studied as a plausible alternative, among others. They are more expressive than CFGs but can also be parsed in polynomial time. Unfortunately, the best known algorithm has a $\mathcal{O}(n^7)$ time complexity with n the length of the input sentence. Thus, in practice most algorithms are based on greedy methods which require fairly strong independence assumptions. The main approach in the literature, called supertagging, filters the search space in a pre-processing step while ignoring long distance relationships, one of the main motivation for LTAGs.

In the past years, combinatorial optimization techniques have been successfully applied to computationally challenging NLP tasks. We follow this line of work in the case of LTAG parsing. More precisely, in our setting, a given NLP problem is reduced to a subgraph selection problem. As such, it has a generic form which may interest other research communities. Then we formulate the generic graph problem as an Integer Linear Program. Integer Linear Programing has been widely studied and many optimization methods exist. We focus on Lagrangian relaxation which previously received much attention from the NLP community. Interestingly, the proposed algorithms can be parametrized to fit a range of different data without impacting efficiency.

Our first contribution is a novel pipeline for LTAG parsing. Contrary to the supertagging approach, we propose a pre-processing step which takes into account relationships between words: well-nested dependency parsing with 2-bounded block degree. An algorithm with a $\mathcal{O}(n^7)$ time complexity has been proposed for this problem in the literature, which is similar to the standard LTAG parser complexity. In order to tackle the complexity challenge, we show that it can be reduced to a subgraph selection problem which can be expressed

via a generic ILP. With our algorithm, the well-nested constraint can easily be toggled off and the block degree bound can be changed. Thus, as an example, it can be used for parsing problems related to other lexicalized grammars. We experiment on several problems showing the efficiency and usefulness of our method.

Our second contribution is a novel approach for discontinuous constituent parsing. We introduce a variant of LTAG for this task. Parsing is then equivalent to the joint tagging and non-projective dependency parsing problem. We show that it can be reduced to the Generalized Maximum Spanning Arborescence problem which has been previously studied in the combinatorial optimization literature. A novel resolution algorithm based on Lagrangian relaxation is proposed. We experiment on two standard discontinuous constituent datasets and obtain state-of-the-art results alongside competitive decoding speed.

Key words: Parsing, Tree Adjoining Grammars, Lagrangian relaxation, Generalized Maximum Spanning Arborescence

Français

Titre: Approches fondées sur la relaxation Lagrangienne pour l'analyse syntaxique avec grammaires d'arbres adjoints

Résumé: Ces dernières années, des méthodes issues de l'optimisation combinatoire ont été appliquées avec succès pour résoudre des problèmes algorithmiques difficiles en Traitement Automatique des Langues (TAL). Nous suivons cette méthodologie dans le cadre de l'analyse syntaxique avec des Grammaires d'Arbres Adjoints Lexicalisées. Plus précisément, un problème d'analyse est d'abord réduit à un problème de sélection de sous-graphe. Ensuite, nous formulons ce dernier sous forme de Programme Linéaire en Nombres Entiers. Beaucoup d'algorithmes ont été proposés pour ces formulations. Nous nous concentrons sur la relaxation Lagrangienne qui a reçu beaucoup d'attention de la part de la communauté du TAL. La particularité de notre méthode réside dans le fait que nos algorithmes résolvent des problèmes généraux et peuvent donc être testés sur différentes données.

Mots clés: Analyse Syntaxique, Grammaires d'Arbres Adjoints, Relaxation Lagrangienne, Arborescence généralisée de poids maximum

Acknowledgments

First of all, I would like to thank my supervisor, Joseph Le Roux, for his invaluable mentorship and support during these 3 years. One of the strong point of this thesis is its interdisciplinary nature which would have not been possible without the great help of Mathieu Lacroix. I would also like to thank Adeline Nazarenko for her trust with this Ph.D. work. The LIPN and especially the RCLN team has been a pleasant work environment. In particular, I would like to thank Emmanuel Cartier for the opportunity to teach and Jorge Garcia Flores for the development of the sound mapping website. Also, I would like to thank Thomas Rubiano, Nadi Tomeh and Yann Chevaleyre for the endless discussions we shared. Finally, I would like to thank Antoine Rozenknop and Roberto Wolfler Calvo for their help at the beginning of this work.

I thank Laura Kallmeyer and Kilian Evang for providing us with the script for the discontinuous PTB. I also thank Xavier Carreras for providing us his script for extracting a spinal TAG from the Penn Treebank.

On a personal note, these 3 years have been a difficult time but I have been lucky enough to have a lot of encouragement from my friends and family. I would never have started a Ph.D. without the support (and almost obligation!) of Mathilde: thank you very much. I give a special thank to my mother and my sister who always believed in me. Thanks to my ex-roomates David, Lysanne, Mariecke, Joanna and Thomas who brightened my stay in Paris. Thanks to my friends Angèle, Anne, Benjamine, Baptiste, Corel, David, Fabrice, Gaël, Gino, Maxime and Simon (sorry for ones I have forgotten) for all the 8.6 and Picon drinks we shared through the years. Let's hope there will be many more.

My Ph.D. studies were supported by a public grant overseen by the French National Research Agency (ANR) as part of the Investissements d'Avenir program (ANR-10-LABX-0083).

Contents

Part I

Introduction

Chapter 1

Introduction

A formal language is a possibly infinite set of strings specified via rules defining how symbols can be combined. In general, formal languages are described thanks to either rule-rewriting or tree-based systems. The most popular rule-rewriting formalism is **Context-Free Grammar** (CFG), which is expressive enough to describe programming languages, among others. In this thesis, though, we focus on natural languages where symbols are words and strings are sentences. Joshi [1985] argued that CFGs are not well suited to natural languages, leading to the development of **mildly context-sensitive grammars**, including **Tree Adjoining Grammars** (TAGs). Following research in linguistics confirmed the ability of TAGs to model composition rules observed in natural language sentences [Abeillé, 1988, Abeillé et al., 1990]. In other words, grammars are generators of well-formed sentences. A concrete example of TAGs attractiveness over CFGs is their ability to model cross-serial dependencies. In Dutch, relative clauses like "`dat Jan de kinderen zag zwemmen`", meaning "`that Jan saw the children swim`", are composed by first enumerating all the subjects and then all the verbs. This phenomenon, certifying that the number of verbs is equal to the number of subjects alongside dependencies from the k^{th} verb to the k^{th} subject, is roughly equivalent to the copy language problem, that is building a grammar which models the set of strings $\{ww | \in \Sigma^*\}$, where Σ is a set of symbols. It is well-known that the latter problem cannot be modelled with CFGs.

The other way around, the grammatical analysis of a given sentence, called the parse, exposes relationships between phrases that may not be directly represented in the word sequence. Both analogies are useful in Natural Language Processing (NLP): encoding and decoding messages are obviously critical problems for the development of human language interfaces. Usually, grammatical analysis rely on data that have been annotated by experts. Because these linguistic structures often look like trees, they are called treebanks. However, parsing with TAGs automatically extracted from annotated treebanks has received little attention from the community even though Chiang [2000] experimentally demonstrated their accuracy. Importantly, an alternative line of work is based on splittable TAGs [Schabes and C. Waters, 1995, Carreras et al., 2008, Kasai et al., 2017] which have efficient parsing algorithms but cannot directly encode properties that make TAGs linguistically plausible, including cross-serial depen-

dencies. We believe the unattractiveness of mildly context-sensitive formalisms is due to their long decoding time. The aim of this thesis is to propose an efficient parsing framework for Lexicalized Tree Adjoining Grammars (LTAGs), a variant of TAGs where each tree must have exactly one lexical anchor, that is a leaf with a word [Schabes et al., 1988].

1.1 Lexicalized Tree Adjoining Grammars

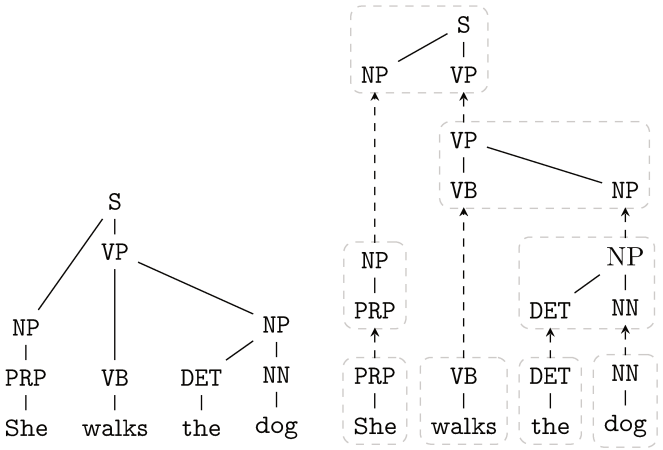
1.1.1 Motivation

A grammar is a finite set of rules which constrain the combination of symbols (words) in order to generate the possibly infinite set of valid strings (sentences) in a given language. As an example, the informal simple rule *a subject followed by a verb is a valid clause* validates the English sentence "`She walks`". Traditionally, coherent subsets of words, called constituents,[1] are grouped with respect to their hierarchical relationships resulting in a tree called phrase-structure [Chomsky, 1957]. The sentence "`She walks the dog`" contains 7 constituents : `She`, `walks`, `the`, `dog`, `the dog`, `walks the dog`, `She walks the dog`.[2] Relationships include the noun phrase "`the dog`" with determinant "`the`" and noun "`dog`". Figure 1.1 exposes the full phrase-structure of the sentence alongside two plausible constructions from fragments called **elementary trees**.[3] Thus, a sentence can be generated by combining fragments representing local syntactic units via **combination operations**. As of now, we only introduce the **substitution** operation which merges the root node of a tree fragment with the leaf of another if the labels are equal. Such generative grammars are qualified as **shallow** because they only certify the grammatical correctness of a sentence without imposing any semantic coherence. As reported by Noam Chomsky, the sentence "`Colorless green ideas sleep furiously`" is thus valid despite being meaningless. This issue can be partially addressed thanks to lexicalization. A grammar is **lexicalized** if and only if each one of its elementary trees contain exactly one lexical anchor, that is a word. Figure 1.2 exhibits two elementary trees associated with the word "`walks`", either used as an intransitive or transitive verb. Moreover, note that the sentence "`She walks the dog`" is semantically correct but "`She walks the river`" is not. Because elementary trees are lexicalized, we can trivially add a substitution constraint: the object associated with a transitive realization `walks` can only be in a subset of allowed words, including `dog` but not `river`. As such, the substitution operation in a lexicalized grammar directly encodes semantic information: we qualify this family of formalisms as **deep**. Although beyond the scope of this work, note that **unification grammars**, which combine surface and semantic structures, found a widespread intestest in the linguistic community and rely upon TAGs, among others [Abeillé, 2007].

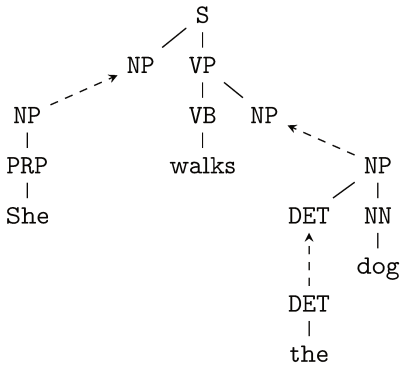
[1] We abuse definitions and ignore the difference between part of speech and constituent for simplicity.

[2] Unless otherwise specified, all the examples follow the Penn Treebank annotation scheme [Marcus et al., 1993].

[3] Note that, when the tree fragments are of depth 1, the formalism is equivalent to standard CFGs. Moreover, readers familiar with the TAG formalism may be offended by the usage of substitution for the determinant: our goal is to introduce the formalism, not to propose a linguistically plausible grammar.

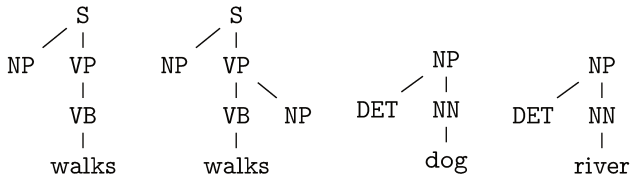


(a) Phrase-structure (b) Tree fragments



(c) Lexicalized tree fragments

Figure 1.1: (a) Phrase-structure of the sentence "She walks the dog" according to the Penn Treebank annotation guide. (b) Example of a possible construction using elementary trees of depth 1 only. Boxes delimit elementary trees and dashed arcs substitution operations. (c) Alternative with a lexicalized grammar. The number of elementary trees used is equal to the number of words.



(a) Intransitive (b) Transitive (c) Dog (d) River

Figure 1.2: Examples of lexicalized elementary trees. (a) In its intransitive form, the verb "walks" only requires a subject (NP node). (b) When used as a transitive verb, an object is required (right NP node). (c)-(d) Noun phrase elementary trees.

1.1.2 Parsing

Given a sentence, **parsing** is the computation of its syntactic structure according to a grammar. Several properties of natural languages make this a challenging task. First, a sentence may be ambiguous: in the sentence "`She bought a house on the hill`", the location may either refer to the transaction place or to the building location. Disambiguation may rely on the context or the most common usage. Note that if the location was a boat, both analyses would be allowed only under a shallow formalism but not under a deep one. Secondly, a grammar is always constructed with respect to a strict subset of the possible infinite number of sentences. Several syntactic structures may not be observed beforehand. As an example, a parser must be able to deduce the use of "`walks`" as a transitive word even if it was an unknown phenomenon at the design step. Even with extensive hand-engineering effort, one cannot take into account every possible linguistic incongruity. Among other phenomena, we can point out that dialects and figures of speech may abuse the standard morpho-syntactic coherence of a sentence. Finally and obviously, spelling and grammatical errors are commonly observed. The key solution resides in **weighted parsing**. Likelihood scores are given to elementary trees and combination operations. In order to account for unobserved constructions, it is common to construct elementary trees from templates: each observed tree fragment can be modified by changing its lexical anchor. Nowadays, likelihood scores are computed via machine learning. Given a grammar and a scoring model, the **optimal** syntactic structure is the highest scoring analysis.

There are many challenges involved in the development of a parsing algorithm, notably accuracy and speed. The first issue has led to the production of hand-annotated data. Constituency parsers are often built with a grammar automatically extracted from a subset of this data and evaluated against held-out sentences. Moreover, real-world NLP applications often require fast analysis. There are two schools of thought on this subject. First, parsers often rely on the context-free assumption [Carreras et al., 2008] because mildly context-sensitive grammars have high space and time complexities. Second, greedy and beam-search based algorithms have recently achieved state-of-the art results in constituent parsing [Zhu et al., 2013, Coavoux and Crabbé, 2017]. These algorithms read the sentence from left to right, at each step taking a (non-deterministic) decision on the structure to predict. This method is similar to the (deterministic) left-to-right parsing algorithm used by compilers [Aho et al., 1986].

1.1.3 Previous work

Mildly context-sensitive weighted grammars are attractive for concrete applications outside the research scope. However they suffer from high parsing complexities: the standard LTAG algorithm has a $\mathcal{O}(n^8)$ time and $\mathcal{O}(n^5)$ space complexities with n the length of the input sentence. This last dynamic programming algorithm is based on a variant of the Cocke-Younger-Kasami (CYK) algorithm [Kasami, 1965, Younger, 1967, Cocke, 1970] where the phrase-structure is constructed bottom-up while ensuring feasibility according to the grammar. The program must keep in memory all possible subtrees given a subset of words

and attempt every possible elementary tree inclusion and attachment, resulting in those non-tractable complexities, see Section 3.3. Previous work on tractable LTAG parsers mainly focused on three alternatives. First, an asymptotically faster $\mathcal{O}(n^7)$ algorithm has been proposed by Eisner and Satta [2000]. However, it remains too slow for practical applications. Secondly, Left-to-Right parsers have been proposed for TAGs [Nederhof, 1998, A. Prolo, 2002, Shen and Joshi, 2005]. A possible approach with this technique is to keep in memory only a beam[4] of candidates at each step, similarly to transition-based parsers. Their main drawback is their failure to guarantee the optimality of a returned phrase-structure or even an assessment of its quality. Thirdly and finally, a popular approach is to include a preliminary step called **supertagging**: only a subset of elementary trees per word are retained as candidates [Joshi and Srinivas, 1994]. These subsets can be singletons. In practice, supertagging is efficient for short sentences but fails on long ones as it does not impact the asymptotic worst time complexity. Additionally, it often relies on strong independence assumptions. Most supertagging models are uni-gram or bi-gram, that is every elementary tree is predicted independently or only dependencies between two consecutive trees are considered. However, in LTAGs, relationships may be of long distance which will be hardly captured by these independence assumptions. In the sentence "`She walks, despite her hatred for quadruped mammals, the dog`", capturing the transitive nature of the first verb is difficult without further analysis.

To avoid these problems, recent work of Kasai et al. [2017] proposed a pipeline system with a neural supertagger built upon dense vector representation of elementary trees followed by a transition-based algorithm. They argue that the recurrent neural architecture implicitly captures long distance relationships. However, their transition based parser cannot generate mildly context-sensitive structures. We propose an alternative approach by:

- Explicitly representing long distance relationships,
- Permitting the complete generative capacity of LTAGs.

Unlike most of the previous work, we do not rely on strong elementary tree filtering or greedy approaches in order to ensure parsing speed but on combinatorial optimization.

Even beyond the scope of mildly context-sensitive grammars, efficient constituency parsing is often regarded as a more difficult task than another syntactic formalism called dependency parsing [Tesnière, 1959, Melčuk, 1988]. In the latter, relations between words are expressed via bi-lexical dependencies, with each dependency defining the **head** and the **modifier** of the relation. As an example, in the sentence "`She walks the dog`", "`She`" is the modifier of the word "`walks`" in the analysis given in Figure 1.3. Dependencies may be labeled with the type of the relation, like *subject of*. Lexical Functional Grammars incorporate both syntactic representations in a shared structure [Kaplan and Bresnan, 1982]. However, in practice, many dependency parsers simply ignore

[4] We call beam search any heuristic procedure where only the most promising candidates are explored at a given step. This set of candidates is called beam.

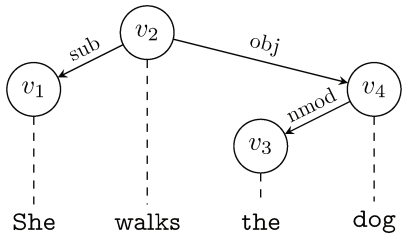


Figure 1.3: Example of a dependency parse. An arrow represents a labeled head → modifier relation.

the constituency structure. As such, their major benefit is that they are not built upon a grammar but only impose constraints on the structure.[5] Simply speaking, the most common approach called **projective dependency parsing** forbids crossing dependencies (Figure 1.4). Interestingly, even if both projective dependency parsing and CFG parsing have the same asymptotic time complexity with respect to the sentence length, the latter incorporate a non-negligible multiplicative constant due to the grammar.[6] Thus, dependency parsing has received a lot of attention from the research community. State-of-the-art results are constantly improving, in regard of both accuracy and parsing time. Historically, several dependency treebanks were built by converting constituent trees via hand-crafted transformation rules called head-percolation tables [Yamada and Matsumoto, 2003, Johansson and Nugues, 2007, Seeker and Kuhn, 2012]. Because of this, several authors incorporated dependency-based filtering in constituency parsers. Carreras et al. [2008] extracted a splittable LTAG grammar according to a head-percolation table. Then, they filtered the constituent parser search space using dependency marginal scores. Recently, Kong et al. [2015] proposed an algorithm which transforms a dependency parse to a constituency one. This task is not deterministic thus they relied on a weighted CFG grammar. Their pipeline parser has a quadratic worst-time complexity but, surprisingly, a linear observable running time in practice. Fernández-González and Martins [2015] reduced the constituency parsing problem to a labelled dependency parsing problem. Interestingly, LTAGs naturally encode bi-lexical dependencies which represent combination operations between elementary trees. It seems therefore appropriate to apply a similar methodology and examine the LTAG parsing problem as a constraint dependency parsing task, which is precisely what we propose in this thesis.

1.2 Contributions

The primary concern of this thesis is efficient LTAG parsing. Following the line of work linking dependency and constituency parsing, we study LTAG parsing as a dependency parsing task. Indeed, a **LTAG derivation tree** is a graph

[5] There exists grammar based dependency formalisms. However, they have received little attention from the NLP community.

[6] Theoretically, algorithms with a lower complexity have been proposed for CFG parsing, relying on fast matrix multiplication methods [Valiant, 1975]. However, they are rarely used in practice.

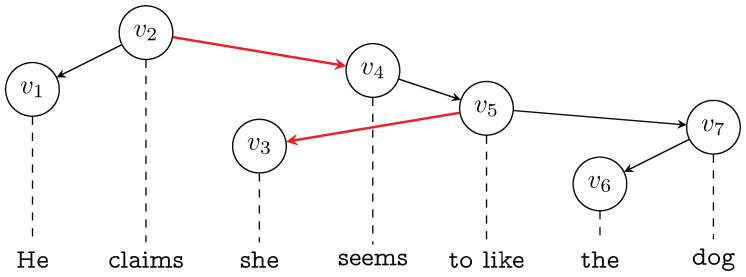


Figure 1.4: This example is a non-projective dependency parse because of the two red arcs. This non-projective dependency structure is motivated by TAG parsing [Kallmeyer and Kuhlmann, 2012].

where each word is represented by a vertex labeled with its elementary tree. A combination operation is represented by an arc labeled with the address of the operation site. The derivation tree is therefore a **dependency structure**: each vertex has at most one predecessor, the graph is acyclic and has a unique root, i.e. a unique vertex without predecessor. This structure and its labels are highly constrained by the grammar. For example, the label on an arc must ensure that it encodes a combination between similar syntactic units. An example is given in Figure 1.5. However, we depart from previously cited work as we do not consider projective syntactic dependencies obtained by head-percolation tables but dependencies induced from LTAG combination operations. The latter are known to be of a non-trivial type of structure [Bodirsky et al., 2005]. Unfortunately, there are no annotated treebanks which exploit the expressive power of LTAGs. To the best of our knowledge, previous experimental work only considered LTAGs restricted to context-free languages [Chiang, 2000, Carreras et al., 2008, Kasai et al., 2017]. These grammars have a cubic parsing time [Schabes and C. Waters, 1995]. The problem is challenging: we want to develop fast decoding algorithms without any test data. Moreover, Satta [1994] showed that developing an asymptotically faster algorithm for TAG parsing is as difficult as boolean matrix multiplication, a well studied problem in the literature. To overcome these difficulties, we formulate the critical parts of LTAG parsing as subgraph selection problems that are generic enough to have other applications. Then, we formulate these graph problems as Integer Linear Programs (ILPs). We propose resolution algorithms for the ILPs and test their efficiency on existing benchmarks. Thus, we are able to show that our parsing algorithms are practically efficient.

1.2.1 Derivation tree parsing

In order to tackle the complexity challenge of LTAG parsing, Joshi and Srinivas [1994] proposed to rely on a two step pipeline:

1. A tagger assigns one elementary tree to each word.

2. The standard LTAG algorithm is run using previous information as a filter.

They motivate their work by the fact that an elementary tree is a rich description of a lexical item (for example intransitive verb), i.e. a specialization of a

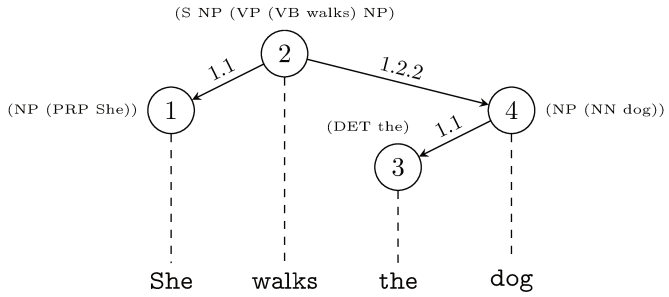


Figure 1.5: Derivation tree representation of a parse. Labels on vertices (respectively arcs) represent elementary trees (respectively operation site addresses).

part of speech tag (for example verb). However, taggers commonly rely on local evidences only, which may not be sufficient for recursive structures and long distance relationships that can be modeled with LTAGs. Moreover, supertagging does not reduce the asymptotic complexity of LTAG parsing. Our first contribution, in Chapter 5, is a novel pipeline system:

1. A constraint dependency parser assigns head-modifier relations.
2. A dynamic program assigns elementary trees and operation sites.

As such, we consider the derivation tree structure instead of the resulting phrase-structure and follow the common pipeline practice of dependency structure parsing. Elementary trees are assigned with respect to non local context as their attachment sites are fixed. The first step is known to have a $\mathcal{O}(n^7)$ time complexity [Gómez-Rodríguez et al., 2009] and we propose an algorithm with linear time complexity for the second one (Section 5.6). Unfortunately, the dependency parser of Gómez-Rodríguez et al. [2009] is not efficient enough on sentences longer than ≈ 20 words. Indeed, LTAG dependencies have an intricate structure: they are well-nested arborescences with 2-bounded block degree [Bodirsky et al., 2005]. In order to tackle this complexity challenge, we propose to rely on combinatorial optimization. We introduce two novel Integer Linear Programming (ILP) formulations of this problem. Interestingly, our ILPs allow to easily toggle the well-nested property and change the block degree bound. They can therefore model dependency structures induced by other lexicalized grammars [Kuhlmann, 2007].

Although LTAGs are linguistically motivated, they can only be used in the case of continuous phrase-structures, that is phrase-structures where every constituent form a group of contiguous words. However, several treebanks have been annoted with discontinuous constituents [Brants et al., 2004, Evang and Kallmeyer, 2011]. Also, Bunt et al. [1987] argued that discontinuity is inevitable in order to bridge the gap between syntax and semantics. In Chapter 6, we show that the discontinuous phrase-structure parsing problem can be tackled through a variant of LTAGs: we relax the intricate structure of the derivation tree. Then, the construction of the phrase-structure can introduce discontinuity. We reduce the joint problem of assigning head-modifier relations and elementary trees to a graph problem called the Generalized Maximum Spanning Arborescence. The

latter is NP-complete and has already been studied by the optimization community [Myung et al., 1995, Pop, 2009]. Thus, once again, we can rely on ILP optimization techniques in order to ensure practical efficiency.

1.2.2 Efficient parsing via Lagrangian Relaxation

Fast decoding of linguistic structures has been a major concern in the NLP community. Unfortunately, some problems are known to have a high complexity. Several authors proposed to formulate them as ILPs which are optimized using appropriate techniques. Koo et al. [2010] applied dual decomposition to high-order dependency parsing which is known to be a NP-hard problem. Riedel and Clarke [2006] enforce the linguistic plausibility of a dependency parse by lazily generating violated constraints.[7] We take a similar approach since the CPLEX software, commonly used as a baseline method for solving ILPs, is inefficient with our programs. We rely on Lagrangian relaxation, a method that was previously successfully applied to NLP[8] [Rush and Collins, 2012, Sontag et al., 2010]. Simply speaking, in a given maximization problem, we identify a set of complicating constraints: if they were absent, then the program could have been solved using an efficient polynomial time algorithm. These difficult constraints are removed and introduced as penalties in the objective. The resulting program, called the Lagrangian dual, is a parametrizable upper-bound on the original one. By minimizing on its parameters, called the Lagrangian multipliers, we tighten this bound. We rely on subgradient descent for the dual optimization, an iterative method: at each step, the relaxed problem is solved with updated multipliers. The optimal solution of the original problem may be obtained under certain conditions.

For our first problem, decoding the maximum well-nested arborescence with 2-bounded block degree, we observe that (1) we can relax the LTAG specific constraints to obtain a quadratic problem and (2) the number of relaxed constraints is high. Moreover, given an arborescence, it is easy to check if it is well-nested or if it has a 2-bounded block degree. Thus, we mix Lagrangian relaxation with lazy constraint generation (Section 5.4). The dual optimization process starts with an empty set of multipliers. Then, at each step of the subgradient descent, we seek for violated relaxed constraints before updating the multipliers. This technique is called Non Delayed Relax-and-Cut [Lucena, 2005]. As part of our pipeline proposal, the decoded arborescence is used as a filter for elementary trees and attachment sites assignment. So we have to return the best possible structure that satisfies the constraints in order to limit error propagation and ensure the feasibility of the second step. Unfortunately, the global optimum of the Lagrangian dual cannot deliver such a certificate in the general case, even if it often happens in practice. Thus, we propose to rely on the Branch-and-Bound algorithm, an exhaustive search procedure. Variables are recursively fixed to 0 or 1 in a search tree. A portion of the search space, that is sub-parts of the search tree, can be safely removed if the Lagrangian relaxation returns an upper bound which is less than the best known solution.

[7] This technique is called the cutting-plane method in the optimization vocabulary.

[8] In order to avoid confusion, we stress that, in this thesis, NLP always stands for Natural Language Processing and never for Non-Linear Programming.

The second problem consists of the joint assignment of tags (elementary trees) and dependencies. We show that this task is equivalent to the Maximum Generalized Spanning Arborescence problem. Unfortunately, it has received little attention from the community and only in cases of graphs with symmetric arc weights. Myung et al. [1995] proposed a dual ascent optimization method embedded in a Branch-and-Bound procedure. We propose an alternative dual optimization algorithm more in tune with the current trend in the NLP community and which is able to produce a certificate of optimality in 99% of cases without relying on an exhaustive search procedure (Section 6.4). In the ILP we propose, there are local and global constraints. Local constraints link dependencies with tags. Global constraints ensure the arborescence structure. We propose to reformulate the ILP in order to rely on dual decomposition, a special flavor of Lagrangian relaxation where the dual is decomposed in a set of independent subproblems. In our case, there are two subproblems which can be run in quadratic time. The first one is a tagging problem with minimal dependency interaction and the second one is an arborescence decoding problem. The obtained Lagrangian dual can again be efficiently optimized thanks to the subgradient descent algorithm.

Chapter 2

Publications[1]

Méthode lagrangienne pour les arborescences couvrantes avec application en traitement automatique des langues

Authors: Caio Corro, Joseph Le Roux, Mathieu Lacroix, Antoine Rozenknop, Roberto Wolfler Calvo

Conference: 17eme congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision

Publication data: February 2016

Abstract: Nous nous intéressons au calcul des arborescences couvrantes de poids maximum avec deux contraintes structurelles : degré de bloc (block degree) et bonne imbrication (well-nestedness). Ces contraintes sont motivées par des problèmes d'analyse syntaxique en traitement automatique des langues (TAL) dans lesquels une phrase est représentée sous forme d'une arborescence couvrante dans un graphe orienté. Nous proposons une formulation du problème en PLNE ainsi qu'une relaxation lagrangienne de celle-ci. Le problème relâché correspond à l'arborescence couvrante de poids maximal pouvant être efficacement calculée grâce à l'algorithme d'Edmonds.

Dependency Parsing with Bounded Block Degree and Well-nestedness via Lagrangian Relaxation and Branch-and-Bound

Authors: Caio Corro, Joseph Le Roux, Mathieu Lacroix, Antoine Rozenknop, Roberto Wolfler Calvo

Conference: 54th Annual Meeting of the Association for Computational Linguistics

Publication date: August 2016

[1] Sorted by publication date.

Abstract: We present a novel dependency parsing method which enforces two structural properties on dependency trees: bounded block degree and well-nestedness. These properties are useful to better represent the set of admissible dependency structures in treebanks and connect dependency parsing to context-sensitive grammatical formalisms. We cast this problem as an Integer Linear Program that we solve with Lagrangian Relaxation from which we derive a heuristic and an exact method based on a Branch-and-Bound search. Experimentally, we see that these methods are efficient and competitive compared to a baseline unconstrained parser, while enforcing structural properties in all cases.

Transforming Dependency Structures to LTAG Derivation Trees

Authors: Caio Corro, Joseph Le Roux

Workshop: 13th International Workshop on Tree Adjoining Grammars and Related Formalisms

Publication date: September 2017

Abstract: We propose a new algorithm for parsing Lexicalized Tree Adjoining Grammars (LTAGs) which uses pre-assigned bilexical dependency relations as a filter. That is, given a sentence and its corresponding well-formed dependency structure, the parser assigns elementary trees to words of the sentence and return attachment sites compatible with these elementary trees and predefined dependencies. Moreover, we prove that this algorithm has a linear-time complexity in the input length. This algorithm returns all compatible derivation trees as a packed forest. This result is of practical interest to the development of efficient weighted LTAG parsers based on derivation tree decoding.

Efficient Discontinuous Phrase-Structure Parsing via the Generalized Maximum Spanning Arborescence

Authors: Caio Corro, Joseph Le Roux, Mathieu Lacroix

Conference: Conference on Empirical Methods in Natural Language Processing 2017

Publication date: September 2017

Abstract: We present a new method for the joint task of tagging and non-projective dependency parsing. We demonstrate its usefulness with an application to discontinuous phrase-structure parsing where decoding lexicalized spines and syntactic derivations is performed jointly. The main contributions of this paper are (1) a reduction from joint tagging and non-projective dependency parsing to the Generalized Maximum Spanning Arborescence problem, and (2) a novel decoding algorithm for this problem through Lagrangian relaxation. We evaluate this model and obtain state-of-the-art results despite strong independence assumptions.

Part II

Background

Chapter 3

Lexicalized Tree Adjoining Grammar

A sentence in a natural language is not a sequence of words randomly generated one after the other. The linguistic research community investigates the structure of natural languages which are described by the means of grammars. Several theories have been proposed to describe a grammar. In this chapter, we introduce the phrase-structure formalism which hierarchically decomposes a sentence in coherent word subsets (Section 3.1). Then, we show how Lexicalized Tree Adjoining Grammars (LTAGs) can be used in order to describe the well-formedness of sentences by imposing constraints on the phrase-structure construction (Section 3.2). Although language generation is an interesting and challenging task, we focus on a different one called parsing (Section 3.3): given a sentence, what is its associated phrase-structure? Unfortunately, the algorithm has an intractable complexity in practice. In order to rely on combinatorial optimization, we seek a simple graph representation of the parsing problem. Thus, we conclude by introducing the LTAG derivation tree structure, a dependency structure describing the construction process of the phrase-structure via a LTAG (Section 3.4).

3.1 Phrase-structure

In this section, we introduce the syntactic phrase-structure formalism, or, more precisely, constituent analysis [Chomsky, 1957]. Syntactic analysis is an active research area in linguistics. We do not claim to give a complete picture of the field. Definitions related to syntax are borrowed form Brinton [2000]. Examples are in English and follow the annotation conventions of the Penn Treebank [Marcus et al., 1993], a widely used corpus. We first introduce the notion of constituency. Then, we describe how to generate well-formed natural language sentences by imposing rules on the hierarchy of its constituents. The resulting generation process gives a phrase-structure. In practice, this procedure relies on formal grammars. We introduce Context-Free Grammars for these first examples. Finally, we explain relations among constituents at the same level of the hierarchy, which will lead to the concept of lexicalization and Lexicalized

Tree Adjoining Grammars in Section 3.2.

> **Definition 3.1: Constituent**
> A **constituent** is a set of words defining a syntactic unit in a hierarchical syntactic structure. As such, it is a part of a sentence that can be moved, modified or deleted alongside agreement adjustments.

We distinguish two types of constituents. On the one hand, the grammatical category of a single word which we refer to as the **part of speech** tag. Each word is assigned a single tag representing its function in the phrase-structure, for example determiner `DET`, verb `VB`, adjective `JJ` or noun `NN`, among others. They constitute the bottom level of the hierarchy, i.e. nodes adjacent to leaves. Note that a word can be ambiguous: it can have different part of speech tags assigned in different contexts. As an example, the word "`orange`" can either be an adjective or a noun. On the other hand, other constituents are non-lexical: they identify a (possibly unary) set of words, and can be decomposed into a sequence of lower-level constituents. As an example, a simple declarative clause `S` can be decomposed into a nominal phrase `NP` (the subject) followed by a verbal phrase `VP` (the predicate). In this thesis, examples rely on the part-of-speech tags and constitutency labels that were used for the Penn Treebank [Marcus et al., 1993], which we report in Appendix A.

We define a simple set of rewriting rules using a Context-Free Grammar (CFG) as follows:

- `NP` → `DT NN`
- `NP` → `DT JJ NN`
- `NP` → `NP CC NP`

The left-hand side of a rule is a unique constituent, called a non-terminal symbol: the one being rewritten. The right-hand side is the sequence of symbols that the former can be rewritten into: it is a sequence of non-terminal and terminal (words) symbols. In the case of phrase-structures, only part of speech tags can be rewritten into lexical items:

- `DET` → `the`
- `NN` → `dog`

We can generate sentences using these rules, see Figure 3.1. The generated phrase-structure is called the **derived tree**.

> **Definition 3.2: Derivation**
> A derivation in a CFG is a sequence of rule applications which rewrites the start symbol, usually `S`, into a string. A derivation ends when the resulting string contains terminal symbols only.

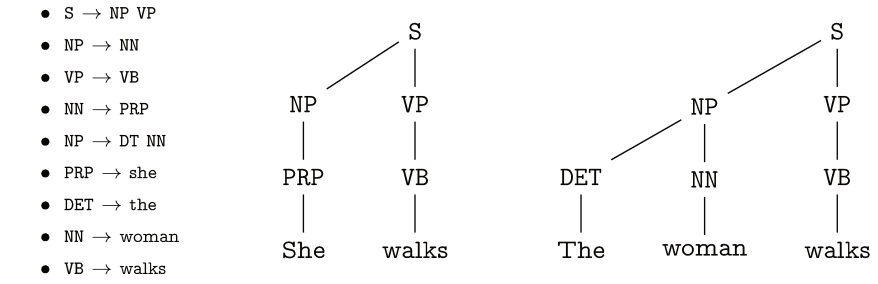


Figure 3.1: Two examples of derived tree that can be built using the grammar on the left of the figure.

Definition 3.3: Derived tree
The derived tree is a representation of a derivation process as a graph. Nodes represent symbols and edges derivation steps.

CFGs are not powerful enough to describe every construction observed in natural languages. First, derivation constraints are local to a constituent and its direct children only. As such, they cannot forbid "`the river`" to be the object of the verb "`walk`" without duplicating the number of non-terminals. As an example, we could have defined a specialization of the rule `VB` → `VB NP`, meaning a verb followed by a noun phrase: `VB-walks` → `VB-walks NP-dog`. However, the resulting number of rules would lead to an impractical grammar. Second, some linguistic structures cannot be constrained using CFGs. In Dutch, a relative clause can be constructed by a sequence of subjects followed by a sequence of verbs: there must be the same number of subject and verbs, the first verb being associated with the first subject, and so on. This kind of structure is called cross-serial dependencies and is roughly equivalent to the copy-language problem, a well-known limitation of CFGs. Third and finally, CFGs can generate continuous constituents only, or, in other words, constituents that are derived into a contiguous sequence of words only.

Definition 3.4: Constituent yield
The **yield** of a constituent is the set of terminal symbols it has been rewritten into during the derivation process.

The derivation process induces an ordering between words (leafs in the derived tree). Indeed, in a CFG, the symbols on the right-hand side of a rule are ordered. Given a rule `S` → `NP VP`, all the terminals in the yield of the `NP` symbol in a derived tree will be preceding the ones in the yield of the `VP` symbol.

Definition 3.5: Continuity
A constituent is continuous if its yield is a contiguous sequence of terminals in the string obtained via derivation. If not, the constituent is discontinuous.

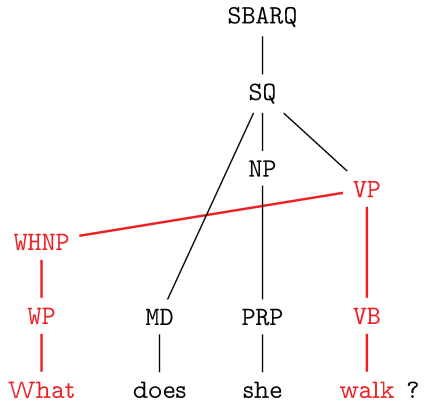


Figure 3.2: Example of a discontinuous phrase-structure in English due to wh-movement. The yield of the VP constituent is not a contiguous set of words.

A phrase-structure or derived tree is continuous if each one of its constituents is continuous. Similarly, a phrase-structure is discontinuous if at least one of its constituents is discontinuous.

CFGs can only generate continuous derived trees. Several treebanks are annotated with discontinuous constituents, even in English [Evang and Kallmeyer, 2011] (see Figure 3.2). Linear Context-Free Rewriting Systems (LCFRS), which generalize the CFG formalism, are capable of deriving discontinuous phrase-structures. Unfortunately, the LCFRS parsing algorithm complexity is prohibitive for NLP applications. As an example, the parsing algorithm of Gómez-Rodríguez et al. [2010] has $\mathcal{O}(n^6)$ complexity in the most restricted case.[1]

In CFGs and derived trees previously described, there is no hierarchical relationship defined between the direct children of a constituent, which we call siblings. We introduce two kinds of relationship that can exist between siblings. This will lead us to the notion of lexicalization.

Definition 3.6: Head-modifier relationship
Given two words in the same constituent, their relationship is qualified as head-modifier if the modifier word is optional and can thus be removed. For example, in the sentence "`She owns a red car`", `red` is a modifier of `car`. Indeed, the sentence "`She owns a car`" is grammatically correct.

Definition 3.7: Governor-complement relationship
Given two words in the same constituent, their relationship is qualified as governor-complement if both words are mutually dependent: one cannot occur without the other. This is the case for the predicate-subject relationship.

[1] Note that this complexity is lower the LTAG parser one but it is for unlexicalized LCFRS only. The unlexicalized TAG parser has also a $\mathcal{O}(n^6)$ complexity.

For example, in the simple sentence "She walks", neither She nor walks can be removed. The direction of the relation is given by their semantic relation: She is a semantic argument of walks, therefore She is the complement of walks.

Definition 3.8: Lexicalized constituent
A constituent is lexicalized if it emphasizes which one of its children contains the head or governor word in its yield.

Unless otherwise specified, we will not distinguish between the two kinds of relationship and refer to both as head-modifier. We mark the head or governor of a relation with the sign * in the right-hand side of a CFG rule. For example, in the rule S $\rightarrow$ NP VP*, the verbal phrase VP is the head of the noun phrase NP. The lexical head of a constituent can be retrieved by moving down to a lexical leaf going through constituents marked as head only. Figure 3.3 shows an example of a derived tree built using a CFG which marks heads. Lexicalized rules are appealing as they can be used to enforce semantic constraints. In the rule VP $\rightarrow$ VB* NP, if the lexical head of the VB constituent is walks, we can restrict allowed lexical heads of the NP constituent to a subset of words containing dog but not river. Thus, with the toy grammar in Figure 3.3, we can avoid the generation of the grammatically correct but semantically incorrect sentence "She walks the river". However, imposing such restriction in a CFG is not natural and impacts the size of the grammar and the parsing algorithm complexity. In the next section, we introduce the Lexicalized Tree Adjoining Grammar formalism which naturally encodes lexicalization. Moreover, we show that it is more expressive than CFGs and thus is a better model for several common linguistic constructions.

3.2 Definitions

Tree Adjoining Grammar (TAG) is a tree-based formalism where elementary trees are combined together in order to build the derived tree [Joshi, 1985, 1987]. Specifically, Lexicalized TAGs (LTAGs) are TAGs where each elementary tree contains exactly one lexical leaf [Schabes et al., 1988]. Contrary to a CFG, a LTAG can naturally encode semantic constraints. We first describe a simpler formalism called Lexicalized Tree Substitution Grammar (Subsection 3.2.1) and then complete with the adjunction operation (Subsection 3.2.2).

3.2.1 Lexicalized Tree Substitution Grammar

The Lexicalized Tree Substitution Grammar (LTSG) formalism is a restricted version of the LTAG one. It only has one type of elementary trees, **initial trees**, and one combination operation, **substitution**.

- S → NP VP*
- VP → VB* NP
- NP → DET NN*
- PRP → She*
- VB → walks*
- DET → the*
- NN → dog*

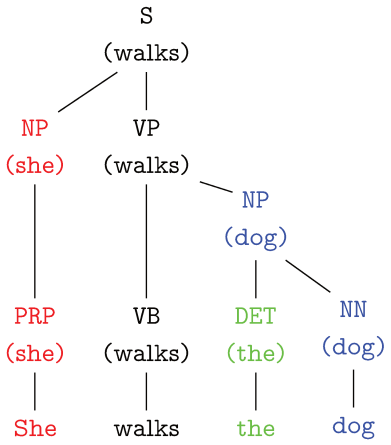


Figure 3.3: (Left) Lexicalized CFG. Each rule identifies its head with the star symbol. (Right) Example of lexicalized constituency structure. The constituent is the same color as its head word.

Definition 3.9: Lexicalized Tree Substitution Grammar
A Lexicalized Tree Substitution Grammar is a tuple $\langle N, T, \Gamma^I, S, f_{SS} \rangle$ where:

- N is a set of non-terminal symbols;
- T is a set of terminal symbols disjoint from N;
- Γ^I is a set of initial trees built with symbols in $N \cup T$;
- $\texttt{S} \in N$ is the start symbol;
- $f_{SS} : \Gamma^I \times \mathbb{Z}^+ \to \mathcal{P}(\Gamma^I)$ is the function that represents combination constraints.

Each elementary tree in Γ^I must have exactly one node labeled with a terminal which must be a leaf. This specific node is called the **lexical anchor** of the tree. Moreover, the root node of a tree must be labeled with a non-terminal.

In practice, non-terminals are constituent labels (`VP`, `NP`, ...) and terminals are words (`she`, `walks`, ...). Without loss of generality, the derived tree is valid only if its root node is the specific non-terminal `S`.[2] An elementary tree assigns a role to a word in a sentence. As an example, two different elementary trees associated with the word `walks` define two different uses of this verb: either as an intransitive or as a transitive verb (see Figure 3.4). In the intransitive case, the verb only requires one argument on its left side, the `NP` leaf, which will be its subject. In the transitive case, the verb also requires an argument (the object) on its right side, hence the rightmost `NP` leaf. Therefore, an elementary

[2]It is trivial to extend this constraint to a subset of non-terminals instead of a single one.

(a) Intransitive (b) Transitive

Figure 3.4: Elementary trees can be understood as part of speech specialization. This example illustrates the use of `walks` as an intransitive or a transitive verb. In both cases, the same part of speech tag is used.

tree could be understood as a specialization of a part of speech tag. In order to distinguish between initial trees and part of speech tags, the former are often called supertags [Joshi and Srinivas, 1994]. Assigning supertags is called supertagging instead of simply tagging.[3]

Definition 3.10: Substitution
Let $G = (V, E)$ be an undirected graph (see Section 4.1 for a formal definition of graph) with finite set of nodes V labeled with elements of $N \cup T$ and finite set of edges E. Moreover, we suppose that G is connected and is a tree. Let $\tau \in \Gamma^I$ be an initial tree and $v \in V$ a node. Substituting τ into v is defined as:

- if v is not a leaf or if v and the root of τ do not have the same label: the operation is undefined;
- otherwise, a new graph is defined by merging the root of τ with node v.

Due to the structure of initial trees, substitution can only happen to nodes labeled with non-terminals.

An example of substitution is given in Figure 3.5. This operation can be constrained thanks to the $f_{SS} : \Gamma^I \times \mathbb{Z}^+ \to \mathcal{P}(\Gamma^I)$ function, with $\mathcal{P}$ the powerset. Given an initial tree $\tau \in \Gamma^I$ and a node address[4] $i \in \mathbb{Z}^+$, $f_{SS}(\tau, i)$ is the set of initial trees that can be substituted into this node. Thus, we can restrict the set of object modifiers of the verb `walks`: we only include the subset of elementary trees with valid lexical anchors, for example `dog`, in the image of $f_{SS}(\tau, i)$ where τ is a tree representing a transitive use of `walks` and i is the address of its object substitution site. A sentence generation is completed when all leafs are labeled with non-terminals. The LTSG formalism we introduced in this subsection is more appealing to describe phrase-structure constraints than the CFG one. Moreover, elementary trees, which are also called supertags, define the precise function of each word in the sentence. In the next section,

[3]The word supertag has also been used in other grammatical formalisms such as Combinatory Categorial Grammars.

[4] We use Gorn addresses which will be introduced in Section 3.3.

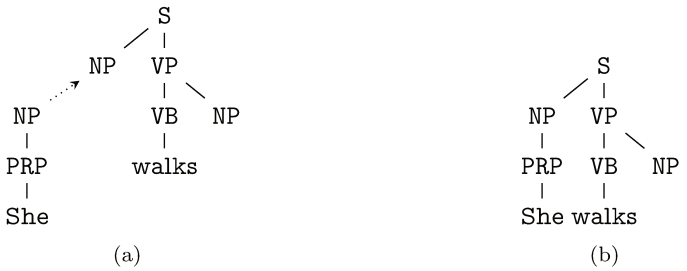


Figure 3.5: (a) The dotted arc depicts a substitution from the root node of the initial tree with lexical anchor `She` into another node. (b) The resulting structure.

we extend the formalism with auxiliary trees combined through adjunctions, leading to LTAGs.

3.2.2 The adjunction operation

In Dutch, relative clauses like "`dat Jan de kinderen zag zwemmen`", meaning "`that Jan saw the children swim`", are composed by first enumerating all the subjects and then all the verbs. Theoretically, the number of subject-verb couples can be infinite: "`dat Jan Piet de kinderen zag helpen zwemmen`", and so on. A possible construction using a LTSG is given in Figure 3.6. Note that the elementary tree anchored with subject `Jan` must then be substituted into the elementary tree anchored with `zwemmen`. However, `Jan` is the subject of `zag`. Thus, this construction breaks our motivation for lexicalized grammars as the combination operation does not represent a head-modifier relationship in this example. This phenomenon is called cross-serial dependencies: the first (respectively second, ...) subject is a modifier of the first (respectively second, ...) verb. This is roughly equivalent to the copy language $\{ww|w \in \Sigma^*\}$, where Σ is a set of symbols, which is not a context-free language.[5] However, it is a mildly context-sensitive language that can be described thanks to a Lexicalized Tree Adjoining Grammar (LTAG).

Definition 3.11: Mildly Context-Sensitive Language
A set of languages is mildly context-sensitive [Joshi, 1985, Kallmeyer, 2010] if:

- it contains all context-free languages,
- it can describe limited cross-serial dependencies,
- it can be parsed in polynomial time,
- it has the constant growth property.

The constant growth property imposes that for all strings longer than a fixed constant in the language, there exists a string in the language which has the

[5]The copy language defined over symbols $\Sigma = \{a, b\}$ contains, among others, the following strings: $aa, bb, abab, abbaabba$.

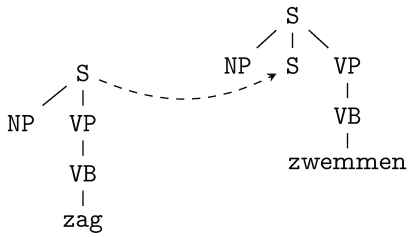


Figure 3.6: Dutch relative clause construction using a TSG.

same length plus a positive constant value [Weir, 1988]. Limited cross-serial dependencies means that the size of the dependent chains that interleave in cross-serial dependencies is bounded by a constant. For example, in TAGs this constant is two, meaning that the language $\{ww|w \in \Sigma^*\}$ can be described with a TAG but not $\{www|w \in \Sigma^*\}$. The set of languages described by TAGs is mildly context sensitive.

Definition 3.12: Lexicalized Tree Adjunction Grammar
A LTAG is a tuple $\langle N, T, \Gamma^I, \Gamma^A, S, f_{SS}, f_{SA}, f_{OA} \rangle$ where:

- N is a set of non-terminal symbols;
- T is a set of terminal symbols disjoint from N;
- Γ^I is a set of initial trees built with symbols in $N \cup T$;
- Γ^A is a set of auxiliary trees built with symbols in $N \cup T$;
- $\mathtt{S} \in N$ is the start symbol;
- $f_{SS} : \Gamma^I \times \mathbb{Z}^+ \rightarrow \mathcal{P}(\Gamma^I)$ is the function that represents substitution constraints;
- $f_{SA} : \Gamma \times \mathbb{Z}^+ \rightarrow \mathcal{P}(\Gamma^A)$ and $f_{OA} : \Gamma \times \mathbb{Z}^+ \rightarrow \mathcal{P}(\Gamma^A)$ are functions that represent adjunction constraints.

The set $\Gamma \triangleq \Gamma^I \cup \Gamma^A$ is the set of **elementary trees**. LTAGs share many similarities with LTSGs (Definition 3.9). The root of an auxiliary tree must be labeled with a non-terminal symbol. Moreover, each auxiliary tree in Γ^A must have:

- exactly one node labeled with a terminal which must be a leaf (lexical anchor);
- exactly one foot node.

The foot node must be a leaf and labeled with the same symbol as the root node. It is identified with symbol *.

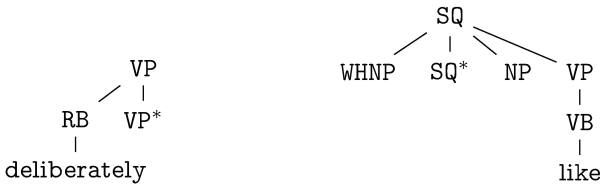


Figure 3.7: Examples of auxiliary trees. The adjective deliberately modifies a verbal phrase. The auxiliary tree anchored with like can be used to construct a question: the interrogative word must be substituted at the leftmost leaf of the tree.

The specificity of LTAGs is that auxiliary trees can be combined through the adjunction operation. Two examples of auxiliary trees are given in Figure 3.7. Contrary to substitution, where a tree is merged at the frontier of another tree, an adjunction breaks the destination tree in two.

Definition 3.13: Adjunction
Let $G = (V, E)$ be an undirected graph with finite set of nodes V labeled with elements of $N \cup T$ and finite set of edges E. Moreover, we suppose that G is connected and is a tree. Let $\tau \in \Gamma^A$ be an auxiliary tree and $v \in V$ a node. Adjoining τ into v is defined as:

- if v and the root of τ do not have the same label then the operation is undefined;
- otherwise, a new graph is defined by replacing v with the root of τ and merging v with the foot node of τ.

Due to the structure of auxiliary trees, adjunction can only happen to nodes labeled with non-terminals.

An example of adjunction is given in Figure 3.8. The example in Figure 3.9 shows a LTAG grammar that solves a simple cross-serial dependencies problem. Similarly to substitution, this operation can be constrained thanks to the $f_{SA} : \Gamma \times \mathbb{Z}^+ \to \mathcal{P}(\Gamma^A)$ function. Moreover, we can force adjunction to happen at a given node with the function $f_{OA} : \Gamma \times \mathbb{Z}^+ \to \mathbb{B}$ set to true if adjunction is obligatory. In the standard LTAG formalism, maximum one adjunction per node is allowed. Multiple adjunctions have been proposed, but we do not consider them as they bring additional issues regarding the order in which combination operations are applied [Schabes and Shieber, 1992, Gardent and Narayan, 2015].

In this section, we introduced the LTAG formalism in order to generate natural language sentences. These grammars are able to encode some semantic constraints. Moreover, LTAGs are more expressive than CFG and allow to correctly model some cross-serial dependencies. Finally, note that the adjunction operation can be used to describe recursive or optional structures without increasing the grammar size. It is customary to use substitution and obligatory

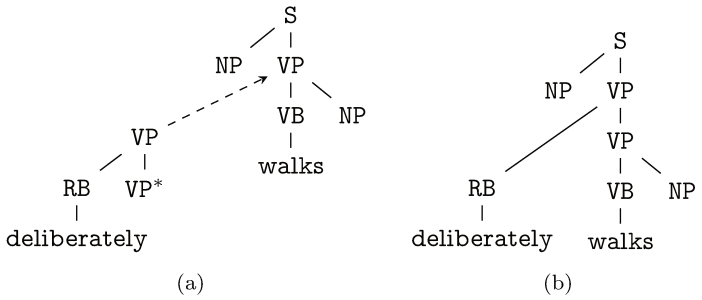


Figure 3.8: (a) The dashed arc depicts an ajdunction operation. (b) The resulting structure. If necessary, the two VP nodes may be merged in a post-processing step in order to reflect the treebank annotation style.

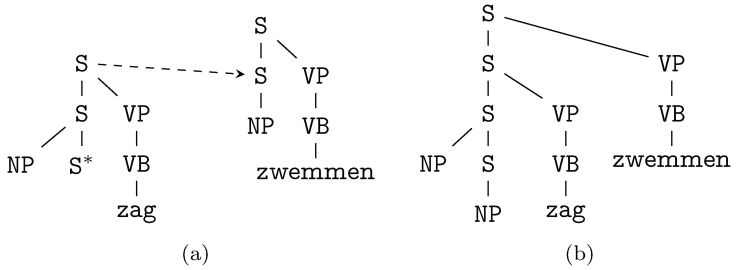


Figure 3.9: (a) Simplified example of cross-serial dependencies via the adjunction operation [Kroch and Santorini, 1987]. (b) The resulting structure. After following substitutions, the subject of zag will be before the subject of zwemmen in the generated sentence.

adjunction to model governor-complement relationships and non-obligatory adjunction to model head-modifier ones [Abeillé, 1988, Abeillé et al., 1990, Gardent and Kallmeyer, 2003] Finally, note that LTAGs can only induce continuous phrase-structures.

3.3 Parsing

In the previous section, we introduced LTAGs for natural language sentence generation. We now turn to the opposite problem: given a sentence and a grammar, how was it generated? This problem is called **parsing** and is the main focus of this thesis. The resulting constituency tree may be of interest for a subsequent task as it exposes the underlying structure of the sequence of words, see Nesson et al. [2006] and Li et al. [2017], among others. For example, Li et al. [2017] recently showed how constituent analysis can be used in order to improve an automatic translation system. Importantly, constituents can easily be augmented with their function, as subject or object, when parsing with LTAGs: an elementary tree defines the function of a word (supertag) and its operation sites define types of relation of modifiers.

Note that the parsing term may typically refer to three different but strongly related tasks via the notion of semi-ring parsing [Goodman, 1999]. First, recognition: can this sentence be generated by a given grammar? Second, derivation forest parsing: decoding the set of all possible derivation trees. This is due to the fact that natural languages are ambiguous, so a given sentence can have several meanings and thus several grammatical analyses. For example, this is the case of the sentence `"She bought a house on the hill"`. Third, weighted disambiguation: what is the best parse in the derivation forest? In this last task, one must rely on a scoring function which evaluates the plausibility of a given phrase-structure.

We first introduce a generic framework to describe parsing algorithms based on logical deduction rules (Subsection 3.3.1). We show how it can be augmented with weights in order to compute the highest scoring parse if the weight function decomposes nicely with respect to the deduction rules. Then, we describe a LTAG parser in Subsection 3.3.2 and discuss the LTAG parsing complexity in Subsection 3.3.3. We focus on the bottom-up algorithm commonly called the CYK-like parser in reference to the CFG parser [Kasami, 1965, Younger, 1967, Cocke, 1970]. Other parsing algorithms have been proposed for non-lexicalized TAGs including Earley-types [Schabes and Joshi, 1988, Joshi and Schabes, 1997] and a LR-types [Nederhof, 1998, A. Prolo, 2002, Shen and Joshi, 2005] which could be upgraded in a LTAG perspective.[6] Earley-type parsers build parse trees in a left-to-right fashion using top-down predictions in order to reduce the search space.[7] LR-type parsers build offline top-down predictions of the Earley-type parsers that are not dependent of the input. Thus, the resulting parser only needs to read the sentence from left to right. A LR parser combined with beam-search results a so called transition-based parser where only a subset

[6] Worst-case complexities may not be preserved.

[7] However, the asymptotic complexity is equivalent to the bottom-up approach.

(or beam) of the possible candidates are kept leading to incomplete[8] but fast parsing.[9]

3.3.1 Parsing as deduction

Describing an algorithm via pseudo-code has several drawbacks. Many decisions in the pseudo-code do not belong to the parsing strategy but to the actual implementation: in which order are the intermediate results constructed? How are they stored? Moreover, space and time complexities may be tedious to infer because of data structures and their accessors, among others. Finally, proving soundness and correctness is not straightforward. Pereira and Warren [1983] followed by Shieber et al. [1995] proposed to rely on a deduction-based framework: we start with a set of axioms and a goal item must be reached thanks to a set of deduction rules. In this framework, the space complexity is readily available from item definitions and time complexity from deduction rules, regardless of the actual implementation. We quickly introduce this formalism and show how any deduction-based parser can be implemented using the agenda-based framework [Kay, 1986].

We introduce the parsing formalism with CFGs in a slighlty variant form of the Chomsky normal form [Chomsky, 1959]: each rewriting rule must have in its right-hand side either exactly one symbol or exactly two non-terminal symbols. Let $s = s_1 \dots s_n$ be the input sentence and R the set of production rules of the grammar. The item set is defined as triplets $[\texttt{A}, i, j]$ with:

- $\texttt{A}$ a non-terminal symbol of the grammar;
- $1 \leq i \leq j \leq n$ two integers defining the yield span of the rule.

The semantics of a rule is simple: a constituent labeled $\texttt{A}$ dominates the substring from the word at index i to the word at index j in the input sentence. Or, in other words, we successfully build a constituent $\texttt{A}$ that yields the word sequence $s_i \dots s_j$. Let $\texttt{S}$ be the start symbol, meaning the root constituent of a parse tree must be $\texttt{S}$ in order to be valid. Thus, the goal of a CFG parser is to produce the following item:

Deduction rule: Goal

$$\frac{}{[\texttt{S}, 1, n]}$$

The axioms are defined as follows:

[8] There is no guarantee to build the full parse forest neither to retrieve the highest scoring parse, hence the use of the term incomplete.

[9] A practical transition-based parser should however also include a back-tracking technique in order to ensure that at least one feasible parse is constructed.

	Item	Rule
1.	$[\texttt{PRP}, 1, 1]$	Lex scan (`He`)
2.	$[\texttt{VB}, 2, 2]$	Lex scan (`walks`)
3.	$[\texttt{NP}, 1, 1]$	Move unary with 1
4.	$[\texttt{VP}, 2, 2]$	Move unary with 2
5.	$[\texttt{S}, 1, 2]$	Move binary with 3 and 4

Figure 3.10: Parsing trace of the sentence "`He walks`" with the grammar given in Figure 3.1. The last item is a goal item so the sentence has been correctly parsed.

Deduction rule: Lex scan

$$\frac{}{[\texttt{A}, i, i]}\ \texttt{A} \to s_i \in R$$

The condition on the right-hand side of the bar defines constraints on axiom creation. Basically, we create an item for each word s_i of the input sentence with each non-terminal `A` such that a $\texttt{A} \to s_i$ exists in the grammar. We now turn to rules that deduce new items from existing ones. First, we investigate rules with a single non-terminal in the right-hand side (unary rules):

Deduction rule: Move unary

$$\frac{[\texttt{B}, i, j]}{[\texttt{A}, i, j]}\ \texttt{A} \to \texttt{B} \in R$$

The item at the top is called the **antecedent** and the item at the bottom the **consequent**. Second, we need two antecedents with consecutive yields when the right-hand side is composed of two non-terminals (binary rules):

Deduction rule: Move binary

$$\frac{[\texttt{B}, i^1, j^1] \qquad [\texttt{C}, i^2, j^2]}{[\texttt{A}, i^1, j^2]}\ \texttt{A} \to \texttt{B C} \in R \wedge j^1 + 1 = i^2$$

Given a rule $\texttt{A} \to \texttt{B C} \in R$, we can deduce an item with non-terminal `A` if and only if we have two items with non-terminals `B` and `C` with consecutive yields. Figure 3.10 gives an example of parsing with the grammar in Figure 3.1.

Let $g = |R|$ be the size of the grammar, t the number of non-terminal symbols and n the input sentence length. The space complexity of a deduction-based

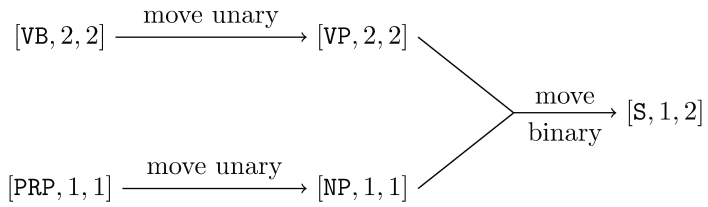


Figure 3.11: Back-pointers are stored as a directed hyper-graph, that is a graph where arcs can have multiple sources: vertices represent items and hyper-arcs represent deduction rule applications.

algorithm is bounded by the maximum number of items. Our items are indexed by a non-terminal and a yield span, thus the space complexity of CFG parsing is $\mathcal{O}(n^2t)$, that is asymptotically quadratic with respect to the input sentence size. The time complexity is bounded by the rule which has the maximum number of free variables in its antecedents. For CFG parsing, the **move binary** rule has six free variables: $\mathtt{A}, \mathtt{B}, \mathtt{C}, i^1, j^1, j^2$. Indeed, i^2 is constrained to be equal to $j^1 + 1$. Moreover, $\mathtt{A}, \mathtt{B}, \mathtt{C}$ are non-terminals but are constrained by the grammar and i^1, j^1 and j^2 are indices in the input sentence. Thus, the time complexity of this CFG parser is $\mathcal{O}(n^3g)$ or asymptotically cubic with respect to the input sentence length.

So, what do we do with the goal item? As such, the only information therein is correctness of the input sentence with respect to the given grammar: this sentence could have been generated with the grammar. But how? In order to retrieve the set of all possible derivation trees for the input sentence, we have to keep back-pointers in memory in order to retrieve the derivation process. If we define the items as vertices in a directed hyper-graph, back-pointers will be hyper-arcs from the set of antecedents to the consequent of deduction rule applications. A single vertex can have several incoming hyper-arcs as it could have been created using different rules and/or antecedent if the grammar is ambiguous. Figure 3.11 gives the back-pointers graph for the parsing example in Figure 3.10. Thus, given the goal item, we can follow back-pointers in the reverse order in order to build one or several derived tree.

We now turn to weighted parsing. In this task, we wish to compute the derivation tree in the parse forest which maximizes a weighting function. The function must decompose nicely so this task is tractable. We assign weights to axioms and deduction rules. Then, the weight of a parse is the sum of the deduction rule weights it used. Maximizing over the parse forest is trivial with the following construction in the back-pointer hyper-graph:

1. Add a root vertex.

2. Add arcs from the root vertex to each axiom vertex. The weights of these arcs are the weights associated with axioms.

3. Assign weights to hyper-arcs corresponding to their deduction rules.

Then, the task reduces to computing the path with maximum weight from the root vertex to the goal vertex. Note however that we do not need to realize this computation as a post-processing step: when adding a vertex to the back-pointer hyper-graph, add the incoming hyper-arc which maximizes the score from the root vertex to the added vertex only. This change does not impact the complexity of the parser.

Let us finally discuss a generic implementation of deduction-based parsers. A common practice is to rely on dynamic programming in order to deduce items: a chart is created and then items are added to it ordered by yield span lengths, from shorter ones to longer ones. This is easy for CFG parsing but is more challenging for more complex grammars. The agenda-based framework is an alternative to dynamic programming [Kay, 1986]. Axioms are placed in an agenda and an empty chart is created. While the agenda is not empty, an item is randomly popped, added to the chart and tested as an antecedent of each deduction rule. If the rule is binary, all possible compatible antecedents available in the chart are used to build a consequent. A deduced item is added to the agenda if it is not already present in the chart. The space and time complexities of these algorithms are no worse than the ones induced by item and rule structures.[10]

3.3.2 CYK-like algorithm for LTAGs

In this subsection, we describe the deduction-based LTAG parser [Schabes et al., 1988] using notations inspired by [Kallmeyer and Satta, 2009]. In the following, we suppose a LTAG $\langle N, T, \Gamma^I, \Gamma^A, S, f_{SS}, f_{SA}, f_{OA} \rangle$ (see Subsection 3.2.2) and an input sentence $s = s_1 \dots s_n$. Moreover, without loss of generality, we suppose that every elementary tree is binary, i.e. every node must have at most two children.

Item definition

We index nodes in an elementary tree $\tau \in \Gamma$ using Gorn addresses. A Gorn address is a sequence of integers from $\mathbb{Z}^+$ indicating a path beginning at the root node of the tree, see Figure 3.12. Given a Gorn address $p \in \mathbb{Z}^+$ and an elementary tree $\tau \in \Gamma$, the predicate $p \in \tau$ is true if and only if a node exists in τ at address p. Items are 8-tuples of the form $[h, \tau, p, c, i, k, l, j]$ with:

1. **Lexical anchor index** $1 \leq h \leq n$;
2. **Elementary tree** $\tau \in \Gamma$;
3. **Gorn address** p of a node in τ;
4. **Combination flag** $c \in \{\top, \bot\}$ indicating if we have already checked adjunction $\top$ or not $\bot$ at the node at position p;
5. **Yield span** $1 \leq i \leq j \leq n$;

[10] Note however that, in order to ensure the tightest possible upper bound on time complexity, the algorithm must rely on efficient data containers. However, this problem is beyond the scope of this thesis.

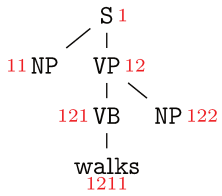


Figure 3.12: An elementary tree with Gorn address of nodes in red.

6. **Gap span** $i \leq k \leq l \leq j$ if the sub-analysis represented by the item has a gap, $k = l = -$ otherwise.

Axioms and goal

There are two families of axioms. First, the **lex scan** rule creates items with position at the lexical anchor if the elementary tree lexical anchor is equal to the word at this position in the sentence:

Deduction rule: Lex scan

$$\frac{}{[h, \tau, p, \top, h, -, -, h]} \; \tau(p) = s_h$$

Second, the **foot predict** rule predicts the span of the content that will be added below the foot node after adjunction. Given the function $foot : \Gamma^A \rightarrow \mathbb{Z}^+$ which returns the foot node position of an auxiliary tree, the creation of these axioms is defined as:

Deduction rule: Foot predict

$$\frac{}{[h, \tau, p, \top, k, k, l, l]} \; \tau \in \Gamma^A \wedge p = foot(\tau) \wedge k \leq l \wedge (h < k \vee l < h)$$

Basically, these items predict the possible yield spans of the non-terminal where the auxiliary tree τ anchored at word position h will be adjoined into.[11] We now turn to deduction rules that combine items in order to create new ones.

Traversal rules

We start with traversal operations which move up inside an elementary tree. The **move unary** rule can be applied to nodes which do not have any sibling:

[11] To reduce the search space, we could check here that the lexical anchor of the auxiliary tree is equal to the word s_h.

Deduction rule: Move unary

$$\frac{[h, \tau, p \cdot 1, \top, i, k, l, j]}{[h, \tau, p, \bot, i, k, l, j]}\ p \cdot 2 \notin \tau$$

Note that the combination operation must be $\top$ in the antecedent, meaning we already checked for adjunction. Moving up to a node with two children is trickier as we have to take into account that only one of them can contain a gap in its yield span. First, we start with the case where none of them has a gap:

Deduction rule: Move binary no gap

$$\frac{[h, \tau, p \cdot 1, \top, i^1, -, -, j^1] \qquad [h, \tau, p \cdot 2, \top, i^2, -, -, j^2]}{[\tau, p, \bot, i^1, -, -, j^2]}\ j^1 + 1 = i^2$$

The condition checks that both yield spans are contiguous. Then, suppose that only the left child has a gap:

Deduction rule: Move binary left gap

$$\frac{[h, \tau, p \cdot 1, \top, i^1, k, l, j^1] \qquad [h, \tau, p \cdot 2, \top, i^2, -, -, j^2]}{[\tau, p, \bot, i^1, k, l, j^2]}\ j^1 + 1 = i^2 \wedge k \neq -$$

Finally, a similar rule takes care of a gap inherited from the right child:

Deduction rule: Move binary right gap

$$\frac{[h, \tau, p \cdot 1, \top, i^1, -, -, j^1] \qquad [h, \tau, p \cdot 2, \top, i^2, k, l, j^2]}{[\tau, p, \bot, i^1, k, l, j^2]}\ j^1 + 1 = i^2 \wedge k \neq -$$

Combination rules

Elementary trees can be combined via either substitution or adjunction operations. The former is used to attach initial trees and the latter auxiliary trees. An initial tree can be substituted if the item is at the root node. Moreover, the destination node must be a leaf and both node labels must match:

Deduction rule: Substitute

$$\frac{[m, \tau', 1, \top, i, -, -, j]}{[h, \tau, p, \top, i, -, -, j]} \; p \cdot 1 \notin \tau \wedge \tau(p) = \tau'(1) \wedge \tau' \in \Gamma^I \cap f_{SS}(\tau, p)$$

An adjunction can be realized if and only if the current yield span of the destination node can fill the gap span of the modifier.

Deduction rule: Adjoin

$$\frac{[m, \tau', 1, \top, i^1, k^1, l^1, j^1] \qquad [h, \tau, p, \bot, k^1, k^2, l^2, l^1]}{[h, \tau, p, \top, i^1, k^2, l^2, j^1]} \; \tau(p) = \tau'(1) \wedge \tau' \in \Gamma^A \cap f_{SA}(\tau, p)$$

Finally, the last rules allow ajdunction to be skipped at nodes which do not have the obligatory adjunction constraint:

Deduction rule: Null adjoin

$$\frac{[h, \tau, p, \bot, i, k, l, j]}{[h, \tau, p, \top, i, k, l, j]} \; \neg f_{OA}(\tau, p)$$

An example of LTAG parsing with these rules and the phrase-structure in Figure 3.13 is given in Table 3.1.

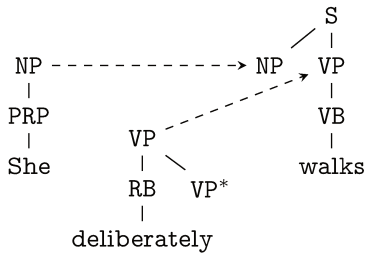
3.3.3 Complexity

As explained previously, space and time complexities can be directly inferred from item structures and deduction rules, respectively. The first and last four elements of an item are word indices in the sentence, thus they can take at most n^5 different values. Thus, the space complexity is $\mathcal{O}(n^5 gt)$ with g the maximum ambiguity (second element) and t the maximum number of nodes in an elementary tree (third element).[12] The maximum ambiguity is the maximum number of elementary trees sharing the same lexical anchor. The **Adjoin** rule is the one with the most free variables in its antecedents: 8 word positions, two elementary trees and one Gorn address. Thus, the time complexity of this LTAG parser is $\mathcal{O}(n^8 g^2 t)$.

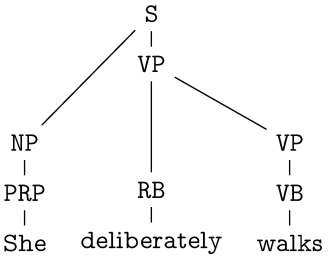
[12]The combination flag can take two different values and thus is a constant that does not appear in the big $\mathcal{O}$ notation.

	Item	Rule
1	$[1, \tau^1, 111, \top, 1, -, -, 1]$	Lex scan
2	$[1, \tau^1, 11, \bot, 1, -, -, 1]$	Move unary
3	$[1, \tau^1, 11, \top, 1, -, -, 1]$	Null adjoin
4	$[1, \tau^1, 1, \bot, 1, -, -, 1]$	Move unary
5	$[1, \tau^1, 1, \top, 1, -, -, 1]$	Null adjoin
6	$[2, \tau^2, 111, \top, 2, -, -, 2]$	Lex scan
7	$[2, \tau^2, 11, \bot, 2, -, -, 2]$	Move unary
8	$[2, \tau^2, 11, \top, 2, -, -, 2]$	Null adjoin
9	$[2, \tau^2, 121, \top, 3, 3, 3, 3]$	Foot scan
10	$[2, \tau^2, 1, \bot, 2, 3, 3, 3]$	Move binary right gap with 8
11	$[2, \tau^2, 1, \top, 2, 3, 3, 3]$	Null adjoin
12	$[3, \tau^3, 1211, \top, 3, -, -, 3]$	Lex scan
13	$[3, \tau^3, 121, \bot, 3, -, -, 3]$	Move unary
14	$[3, \tau^3, 121, \top, 3, -, -, 3]$	Null adjoin
15	$[3, \tau^3, 12, \bot, 3, -, -, 3]$	Move unary
16	$[3, \tau^3, 12, \top, 2, -, -, 3]$	Adjoin with 11
17	$[3, \tau^3, 11, \top, 1, -, -, 1]$	Substitute with 5
18	$[3, \tau^3, 1, \bot, 1, -, -, 3]$	Move binary no gap with 16
19	$[3, \tau^3, 1, \top, 1, -, -, 3]$	Null adjoin

Table 3.1: Parse trace of the sentence `She deliberately walks`. The antecedent of an unary rule is its previous line. For rules with several antecedents, we refer their line numbers. The phrase-structure generation process of this sentence using a LTAG is given in Figure 3.13. We denote τ^1 the elementary tree selected for the first word, τ^2 for the second and so on. We only display items necessary to build the goal. With a more complicated grammar and an ambiguous sentence, several derivations would have been possible.



(a) LTAG analysis, the OA in subscript indicates an obligatory adjunction site



(b) Phrase-structure/derived tree

Figure 3.13: (a) Construction example with a LTAG. (b) Phrase-structure.

An asymptoticaly faster algorithm with a $\mathcal{O}(n^6 \max(n, g)gt)$ time complexity has been proposed by [Eisner and Satta, 2000]. They observed that the **Adjoin** rule simultaneously carries out two independent tests: (1) that the modifier can be adjoined into the head node and (2) that the modifier correctly wraps around the current head node yield span. Thus, they break the operation in 3 different rules leading to a lower time complexity. However, it remains too high for practical use cases where fast analysis is expected: a $\mathcal{O}(n^7)$ factor is a major bottleneck for long sentences.

3.4 Derivation tree

The deduction rule framework is a convenient tool to study parsing algorithms. However, it is unclear if a lower upper bound on LTAG parsing time complexity can be achieved. Satta [1994] reduced boolean matrix multiplication to TAG parsing and argued that obtaining a lower complexity bound for the latter problem is thus likely to be as difficult as to the former one.[13] We propose to rely on combinatorial optimization in order to develop an efficient LTAG parser. Unfortunately, the bottom-up derived tree construction methodology does not seem convenient for this task. Indeed, optimization techniques are often applied to graph problems. It is well known that any dynamic programming algorithm can be reduced to the shortest path problem on an acyclic hypergraph [Martin et al., 1990] enabling the use of optimization in order to ensure efficient decoding, see Clautiaux et al. [2016], among others. However, it happens that LTAG parses have a convenient graph representation. Rambow and Joshi [1997] first

[13] The LTAG and TAG parsing algorithms are slightly different but share many similarities.

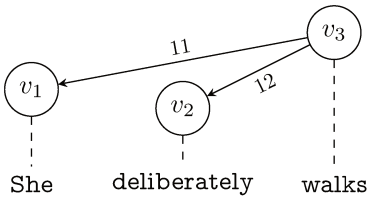


Figure 3.14: Derivation tree of the sentence `She deliberately walks` with respect to the construction in Figure 3.13.

noticed that they can be represented as labeled dependency structures called derivation trees which we introduce in this section.

Definition 3.14: Derivation tree
Let $G = \langle N, T, \Gamma^I, \Gamma^A, S, f_{SS}, f_{SA}, f_{OA} \rangle$ be a LTAG grammar and $s = s_1 \dots s_n$ be a sentence. A LTAG derivation tree for sentence s is a directed graph $D = \langle V, A \rangle$ with $V = \{v_1 \dots v_n\}$ the set of vertices and A the set of arcs where each node $v_i \in V$ corresponds to word s_i. Each vertex $v_i \in V$ is labeled with a single elementary tree in $\Gamma^I \cup \Gamma^A$ which must have word s_i as lexical anchor. An arc $(v_h, v_m) \in A$ represents a combination operation, either a substitution or an adjunction, from the elementary tree anchored at s_m into the elementary tree anchored at s_h. The Gorn address of the operation site is labeled on the arc.

An example of a LTAG derivation tree is given in Figure 3.14. This structure is an arborescence: each vertex has at most one incoming arc, it does not contain a cycle and it is connected. This structure is highly constrained by the grammar: it must describe a valid LTAG derived tree. Combination operations must be correct. For example, only auxiliary trees may be adjoined and operation sites must share the same non-terminal symbol. Besides these grammar constraints, Bodirsky et al. [2005] demonstrated that the arborescence itself has a special structure: it is well-nested and is of 2-bounded block degree. We defer the formal definitions of these properties to Chapter 5.

Rambow [2010] argued that one must distinguish between the syntactic structure and the representation type. The problem we focus on is syntactic phrase-structure parsing. This syntactic content is naturally expressed as a hierarchical structure tree. However, a LTAG derivation tree contains exactly the same syntactic content as a derived tree despite its structural difference. If attachment operations are encoded in node labels of the derived tree, one representation can be converted to the other without losing information. Thus, instead of parsing a sentence by constructing the derived tree in a bottom-up fashion, like in the CYK-type algorithm presented in the previous subsection, we propose to parse the derivation tree explicitly. This task is a dependency parsing problem which can be formalized as a graph decoding problem. As such, it is natural to tackle the complexity challenge using the combinatorial optimization techniques introduced in the next chapter.

Chapter 4

Efficient structure decoding

In the previous chapter, we introduced the linguistic motivation for mildly context-sensitive parsing as well as practical algorithms. We argued that dealing with these non context-free grammars directly is unfortunately not efficient enough for practical applications because of the time complexity of the chart-based parsing algorithm. There has been some attempt to develop more efficient alternatives. Eisner and Satta [2000] proposed a $\mathcal{O}(n^7)$ algorithm for Lexicalized Tree-Adjoining Grammars (LTAGs). Unfortunately, it remains too inefficient for practical use. Schabes and C. Waters [1995] proposed a LTAG inspired formalism called Lexicalized Tree Insertion Grammar (LTIG) with a cubic time parsing algorithm. However, LTIGs do not have the expressive power that makes LTAGs interesting. Thus, it appears that in the current literature one has to choose between fast decoding and expressiveness. We propose to study LTAG parsing as a subgraph selection task, along the same line as the work of Kuhlmann and Jonsson [2015] for semantic parsing. Through this approach, we develop algorithms for derivation tree parsing using tools from combinatorial optimization.

Combinatorial optimization techniques have been successfully applied in NLP. As an example, before the rise of neural networks, dependency parsers achieved state of the art results in terms of accuracy and speed by relying on Integer Linear Programming formulations and well-known optimization techniques like linear relaxation [Martins et al., 2009a], Lagrangian relaxation [Koo et al., 2010], column generation [Riedel et al., 2012] and branch-and-bound [Qian and Liu, 2013]. Rush et al. [2010] combined a constituency parser and a part of speech tagger. Similarly, Le Roux et al. [2013] combined different constituency parsers which must reach an agreement. As the problems are not tractable, they rely on heuristics based on Lagrangian relaxation. However, in the NLP works listed above, optimization techniques are necessary because of the inclusion of high-order likelihood weights or the combination of several systems. On the contrary, the problems we study in this thesis have an intricate structure even in the simplest case. This is closer to the work of Riedel and Clarke [2006] who include linguistically motivated constraints in dependency parsing via lazy constraint generation and the Combinatory Categorial Grammar parser based on Lagrangian relaxation of Auli and Lopez [2011].

In this chapter, we describe a decoding framework relying on Integer Linear Programming (ILP) and Lagrangian relaxation. First, in Section 4.1, we introduce basic notions and notations from graph theory that we will use through this work. As a running example, we will show how non-projective dependency parsing can be naturally formulated as a constrained subgraph selection problem for which there exists a simple combinatorial algorithm. However, if we add additional constraints on the structure, this may not be true anymore. Thus, in Section 4.2, we introduce ILP. Many graph problems can be naturally formulated with this framework. ILP is not tied to a solving method and, in general, optimizing one is NP-hard. Lagrangian relaxation (Section 4.3) combined with subgradient descent (Section 4.4) has been widely adopted by both NLP and optimization communities in order to solve huge and intricate problems. Intuitively, we exploit the structure of our programs to identify a set of difficult constraints. We remove them and introduce them as penalties in the score, leading to an efficient heuristic method. We emphasize that Lagrangian relaxation may produce a certificate if an optimal solution is obtained. Finally, we describe problem reduction in Section 4.5, a method to efficiently prune the search space, and Branch-and-Bound in Section 4.6, a generic algorithm used to solve an ILP exactly.

4.1 Graph-based structure decoding

Graph theory is an appealing tool as it is a *lingua franca* among computer scientists: it is unlikely that a postgraduate student did not take at least one class using graphs before obtaining his/her Master's degree. Moreover, visual representations are straightforward and many definitions, like path and cycle, are intuitive and do not need an in-depth knowledge to be understood. Thus, reducing an intricate parsing problem to a graph problem seems like a reasonable choice. In this section, we explain the notion of structure decoding and its relation with subgraph selection. Important notations that will be used throughout this thesis are introduced. We will use non-projective dependency parsing as an example to illustrate the content.

Definition 4.1: Structure decoding
Given an input s, **decoding** aims at finding the best element inside a set of **output candidates** O^s with respect to a weight function $f^s : O^s \to \mathbb{R}$. In other words, the goal is to find $o \in O^s$ that maximizes f^s. In this work, $s = s_1 \dots s_n$ will always be a sentence in a natural language, i.e. a sequence of tokens, with possibly accompanying features, like morphological information. The decoding process is said to be **structured** because:

1. the set of output candidates is specific to a given input in contrast with the simplest forms of regression or classification,
2. the set of output candidates can have an exponential size with respect to the input,
3. an output is composed of a set of highly constrained **components**.

Other works in the literature have similar definition of structured output spaces [Smith, 2011, Martins, 2012].

Remember that natural languages are ambiguous (Section 3.1). Given a sentence s, one can define O^s as the set of all grammatically correct phrase-structures of s. In this setting, components may be hierarchical dominance of constituents. As an example, a noun-phrase with yield span $[s_i, s_j]$ can contain a determinant followed by a noun. Constituents are obviously constrained: in English, the determinant will always be at the left of the noun. In the Lexicalized Tree-Adjoining Grammar formalism, O^s can be the set of valid derivation trees. That is, components are elementary trees assignments and attachments operations. Finally, in dependency parsing, candidates are valid syntactic structures which are composed of bi-lexical relations between words. In Definition 4.1 above, the weight function f^s may be estimated using machine learning. In that case, structure decoding is also called structured prediction.

Two questions arise from Definition 4.1: given an input, what is the size of the search space? and how is a weight function formalized? Contrary to what one may think, these are related. Regarding the first question, the search space may be very large. Indeed, in practice, grammars are automatically extracted from treebanks. In order to ensure experimental robustness, they are often highly ambiguous, thus allowing linguistically absurd analyses. For example, in LTAG parsing, a common practice is to replace elementary trees with templates: any elementary tree can anchor any word. In dependency parsing, state of the art parsers allow every bi-lexical relation between every couple of words in a sentence. Thus, the weight function must be carefully designed in order to allow efficient maximization in the search space. In general, it is defined as a sum of weights of subset of components of candidate o. In dependency parsing, the subsets may be singletons, each one containing a bi-lexical relation. It is then obvious that in the sentence "`She walks the dog`", `the` is unlikely to be the head of `dog`: this single dependency must be associated with a low weight. A more complicated parser may also introduce subsets of sibling relations, for example subsets containing all relations sharing the same head. In order to ensure computational tractability, the number of subsets of a candidate o is often polynomially bounded by the size of the input s. However, given a search space O^s and a weight function f^s, there may not exist a tractable maximization algorithm. We now introduce the graph theory vocabulary that will allow us to describe more formally O^s and f^s for a given problem.

Definition 4.2: Directed graph
A **directed graph** is a tuple $G = (V, A)$ with $V = \{v_0 \dots v_{|V-1|}\}$ a finite set of vertices and $A \subseteq V \times V$ a finite set of arcs. Arcs are couples where the first element is the **source** and the second the **destination**. For convenience, an arc (v_i, v_j) will mainly be written as $v_i \to v_j$. In this document, unless otherwise specified, we simply refer to **graph** for a directed graph.

An undirected graph is a graph where arcs are unordered couples, that is sets of two vertices. Traditionally, the terminologies node and edge are used to refer to vertex and arc, respectively, in undirected graphs.

Given a vertex subset $W \subseteq V$, we note $\delta^{out}(W)$ and $\delta^{in}(W)$ its sets of outgoing and incoming arcs, respectively. The set $\delta(W) = \delta^{out}(W) \cup \delta^{in}(W)$ is

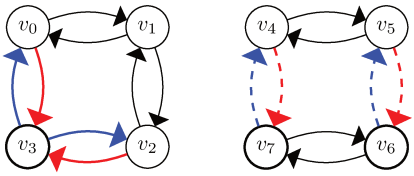


Figure 4.1: Solid red and blue arcs are sets $\delta^{in}(v_3)$ and $\delta^{out}(v_3)$, respectively. The cut-set of v_3 contains these four arcs. Dashed red and blue arcs are sets $\delta^{in}(\{v_6; v_7\})$ and $\delta^{out}(\{v_6; v_7\})$, respectively. Similarly, $\delta(\{v_6; v_7\})$ contains the four dashed arcs. Note that arcs between v_6 and v_7 are not part of these sets.

the **cut-set** of W, see Figure 4.1. If W is a singleton, we drop the set-defining braces. Given a subset of arcs $T \subseteq A$, we note $V[T]$ its **cover**, that is the set of incident vertices, or in other words the set of vertices that are source or destination of at least one arc in T: $V[T] = \{v \in V | \delta(v) \cap T \neq \emptyset\}$. If $V[T] = V$, then the structure defined by arcs T is a **spanning** structure.

Definition 4.3: Arborescence
Let $G = (V, A)$ be a graph. An **arborescence** $T \subseteq A$ is a set of arcs inducing a connected graph with no circuit and where the incident vertices have at most one incoming arc in T. The root of the arborescence T is the single vertex of $V[T]$ with no incoming arc. It is easy to show that, in an arborescence, each vertex is a descendant of the root one.

A **spanning arborescence** is an arborescence with $V[T] = V$. See Figure 4.2 for an example.

Given an arborescence T and a vertex $v \in V[T]$, the **sub-arborescence** T' rooted at v is the maximum subset of arcs $T' \in T$ forming a v-rooted arborescence.

Given an input s, a structured output space O^s and a scoring function f^s, we reduce the decoding to a subgraph selection task by defining how to build:

1. a graph $G^s = (V, A)$;
2. the set of subgraph candidates $\mathcal{T}^s \subseteq \mathcal{P}(A)$, with $\mathcal{P}$ the powerset, so that there is a bijection between O^s and $\mathcal{T}^s$.

We assume that we can define the scoring function on the reduced problem search space $f^s : \mathcal{T}^s \to \mathbb{R}$. Thus, under the graph formulation, components of candidate $T \in \mathcal{T}^x$ are arcs. The simplest weight function decomposes as $f(T) = \sum_{a \in T} w_a$, with w_a the weight of arc $a \in A$. This weighting model is called **arc-factored**. More elaborated models are called **higher-order** models.[1]

Definition 4.4: Maximum Spanning Arborescence
Given a graph $G = (V, A)$, the Maximum Spanning Arborescence (MSA) problem aims at computing the spanning arborescence $T \subseteq A$ rooted at a vertex $v \in V$ that maximizes an arc-factored weighting model.

[1] Note that a weighting function could also take into account vertices in $V[T]$.

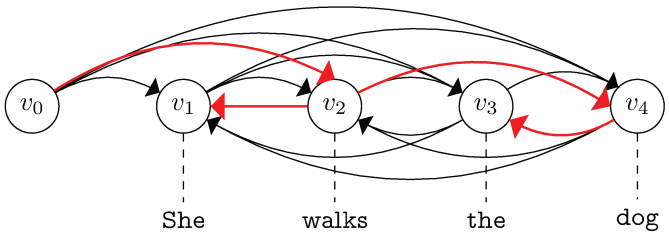


Figure 4.2: Example of non-projective dependency parsing via graph formulation with the sentence "`She walks the dog`". Arcs are dependency candidates, thick red arcs describe a v_0-rooted arborescence.

Example 4.5: Non-projective dependency parsing
We illustrate the decoding process with the arc-factored model for non-projective dependency parsing of McDonald et al. [2005]. In dependency parsing, given a sentence of length n, an output candidate is a bi-lexical analysis of a sentence defined as follows. An artificial root word is added. Every word, except the root:

- is assigned exactly one head;
- must be a descendant of the root word.

As such, components are head-modifier relationships and an output candidate must satisfy the previous rules. Using the graph formalism, vertices will represent words and arcs head-modifier relations.

Given an input sentence $s = s_1 \dots s_n$, we build a graph $G = (V, A)$ with $n + 1$ vertices where v_m is associated with the word s_m, plus an artificial root vertex v_0. A bi-lexical dependency candidate with head-word s_h and modifier-word s_m is represented by an arc $v_h \to v_m$. Then, the set of valid dependency parses $\mathcal{T}$ is the set of v_0-rooted spanning arborescences of G. See Figure 4.2 for an example. The arc-factored weight of a candidate is defined as:

$$f(T \in \mathcal{T}) = \sum_{a \in T} w_a$$

where w_a is a likelihood weight of including the bi-lexical dependency represented by arc $a \in A$. Note that, nowadays, this weight is learned with machine learning.

Enumerating candidates T is intractable. Indeed, given a complete graph with n vertices, there exists $(n+1)^{n-1}$ spanning arborescences on it [Cayley, 1889]. However, under an arc-factored model, maximization can be done efficiently with the Maximum Spanning Arborescence (MSA) algorithim [Chu and Liu, 1965, Edmonds, 1967, Tarjan, 1977]. This algorithm has a quadratic complexity with respect to the number of vertices.

Before the rise of deep neural networks in the NLP community, dependency parsers as the one previously described were enhanced thanks to high-order models. That is, the weight function would decompose into subsets including several arcs. For example, the sibling weighting model can be formulated as:

$$f(T \in \mathcal{T}) = \sum_{a \in T} w_a + \sum_{v \in V} \sum_{a_1 \in T \cap \delta^+(v)} \sum_{a_2 \in T \cap \delta^+(v) \setminus \{a_1\}} w'_{a_1, a_2}$$

where w'_{a_1,a_2} is the sibling weight for including both arcs a_1 and a_2. Unfortunately, McDonald and Satta [2007] proved that computing the spanning arborescence of maximum weight for this family of models is NP-hard. A breakthrough in non-projective dependency parsing occurred when the problem was formulated thanks to an Integer Linear Program, enabling the use of efficient techniques borrowed from the combinatorial optimization community [Martins et al., 2009a, Koo et al., 2010, Riedel et al., 2012]. We follow this line of work but we focus on graph problems which are inherently complex, even in the case of an arc-factored weighting model.

4.2 Integer Linear Programming

In this section, we describe how a subgraph selection problem can be modeled as an Integer Linear Program (ILP). Intuitively, components of the graph like arcs and vertices will be represented by binary variables. Then, constraints are used to prohibit variable assignments that do not represent valid structures in the search space. Although simple, programs of this family have been highly studied in the literature [Genova and Guliashki, 2011]. Many methods have been proposed to tackle problems with intractable complexities including the traveling salesman [Bellmore and Nemhauser, 1968], the set covering [Christofides and Korman, 1975] and the Steiner tree [Gordeev and Tarastsov, 1993] problems, to name only a few. In this Section, we introduce ILPs alongside important properties. Various methods which exploit the structure of a specific program have been proposed like column and row generation, relaxation and decomposition. An important asset is their ability to provide a certificate of optimality when the best solution is obtained. This is in contrast with non-optimal greedy or beam-search methods, like transition parsers. We focus on Lagrangian relaxation which will be introduced in Section 4.3.

4.2.1 Introduction

Given a set of vectors X, optimization seeks to find one of its elements that maximizes[2] an **objective function** $f : D \to \mathbb{R}$ with $X \subseteq D$. Formally, we define an **optimization program** as follows:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & f(x) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

A **feasible solution** is any element $x \in X$. Given the **optimal score** $\hat{f} = \max_{x \in X} f(x)$, an **optimal solution** is noted $\hat{x} \in \{x \in X | f(x) = \hat{f}\}$.

[2] Readers familiar with the optimization literature may be more used to minimization problems. We follow the trend in NLP instead.

An optimization program can be classified according to the nature of the objective function and the constraints. A **linear program** (LP) is a program for which the objective function is affine and the feasible set is a polytope described by a finite set of linear constraints. Using vector notation, we write a LP as:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

where $x \in \mathbb{R}^c$ is the vector of c variables (columns), $w \in \mathbb{R}^c$ is the weight vector and e a constant defining the affine objective function. Note that e can be omitted without changing the set of optimal solutions, hence the name linear program. As will be shown below, it is however convenient to keep this constant term. The feasible solution set is $X = \{x \in \mathbb{R}^c | Ax \leq b\}$ where $A \in \mathbb{R}^{r \times c}$ is a matrix and $b \in \mathbb{R}^r$ a vector defining r constraints (rows). One can observe that each constraint defines a hyperplane delimiting the set of feasible solutions in the c-dimensional solution space. If we furthermore constrain some components of x to be integers, the program is called a **Mixed-Integer Linear Program**. In this work, we will focus on pure **Integer Linear Programs** (ILPs), that is where x is restricted to be an integer vector. Moreover, we restrict each element of x to be binary, which leads to Binary Linear Programs (BLP) of the form:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + c \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{B}^{|x|} \end{aligned}$$

with $\mathbb{B} = \{0, 1\}$. In the graph formulation introduced in the previous section, components, that is arcs, are defined as binary elements of vector x. Thus, the set of constraints delimits the set of valid subgraphs. Note that many other families of programs have been studied in the literature including, *inter alia*, conic, quadratic, non-linear and disjunctive ones. The families which an optimization problem belongs to are not necessary exclusive but their study is far beyond the scope of this thesis.

Example 4.6: ILP formulation of the MSA
We now give a formulation of non-projective dependency parsing as an ILP. Recall that, given a sentence $s_1 \dots s_n$, we build a graph $G = (V, A)$ with $V = \{v_0 \dots v_n\}$ and $A = V \times V$ (Example 4.5). Given an arc-factored scoring function $f(T \subseteq A) = \sum_{a \in T} w_a$, we search a v_0-rooted spanning arborescence T on G which maximizes f. Let $x \in \mathbb{B}^A$ be a vector of $|A|$ binary variables indexed by arcs. Given an assignment of x, an arc $a \in A$ belongs to the arborescence if and only if $x_a = 1$. Then, the MSA can be

computed thanks to the following program [Schrijver, 2003]:

$$\begin{aligned}
\underset{x}{\text{argmax}} \quad & \sum_{a \in A} x_a w_a && && (4.1)\\
\text{s.t.} \quad & x(\delta^{in}(v_0)) = 0 && && (4.2)\\
& x(\delta^{in}(v)) = 1 && \forall v \in V^+ && (4.3)\\
& x(\delta^{in}(W)) \geq 1 && \forall W \subseteq V^+ && (4.4)\\
& x \in \mathbb{B}^A && && (4.5)
\end{aligned}$$

with $V^+ = V \setminus \{v_o\}$. The objective (4.1) maximizes the arc-factored score of the arborescence. Constraint (4.2) ensures that the root vertex has no incoming arc while constraints (4.3) ensure that other vertices have exactly one. Finally, the decoded structure is acyclic because of the previous constraints together with inequalities 4.4: for each subset of vertices, there must be at least one incoming arc.

Searching the optimal solution of this ILP with a generic algorithm would probably lead to poor performance as there are an exponential number of inequalities (4.4). However, it is well-known that an algorithm for the MSA with a $O(|V|^2)$ time-complexity exists [Chu and Liu, 1965, Edmonds, 1967, Tarjan, 1977]. This property will be used throughout this thesis: if an ILP can be cast to the MSA program (4.1)-(4.5), then it can be solved efficiently. On a side note, formulations of this problem with a polynomial number of constraints have also been proposed [Magnanti and Wolsey, 1995, Martins et al., 2009a]. They rely on a flow formulation where commodities are sent to vertices through arcs. We discuss this model in Subsection 5.2.2.

The term **solving** defines the search of an optimal solution of an optimization program and such a procedure is called a **solver**. The time complexity of a solver is defined with respect to the number of constraints and variables in the input program. A solver is said to deliver a **certificate of optimality** if it provably returns an optimal solution for a given problem. Otherwise, it is qualified as a **heuristic procedure**. Efficiently solving an optimization program is a difficult task which keeps a large community of researchers busy. Commonly, solvers are either general enough for a family of programs, that is where the variables and constraints are not known beforehand, or specialized for a single program, that is variables and constraints must obey a generic predefined structure. This does not necessarily imply that the numbers of variables and constraints are fixed. We qualify implementations of the first type of solvers as **off-the-shelf**. The most famous solver for Linear Programs is the **simplex** algorithm. Starting from a feasible solution, the simplex iteratively moves to better feasible solutions and is able to provide a certificate when an optimal solution is reached. It has an exponential worst-case complexity but a polynomial amortized complexity. It is known to be experimentally efficient. Another solver for Linear Programs is the interior-point algorithm which has a polynomial complexity. The most famous off-the-shelf solver implementation for LP is CPLEX.[3] It is often used

[3] CPLEX can solve other families of optimization programs like ILPs and quadratic programs. `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

as a baseline for comparison against specialized procedures. However, we need to clarify that CPLEX is not a pure implementation of a solver like the simplex. Instead, it uses an intricate in-house recipe which relies on specialized procedures depending on the structure of the input program.

Because solving an optimization program, called the **primal**, may be difficult, it is sometimes appealing to consider alternatives to the original formulation which expose interesting properties. Note that as we are studying the maximization of a primal problem, any primal feasible solution score is a **lower bound** on the primal optimal score. A **relaxation** of a LP is an alternative to the primal program which contains at least the same search space, and which assigns weights greater or equal to the primal feasible solution original weights [Geoffrion, 1974]. It is easy to see that the optimal score of a relaxed problem is an upper bound to the primal optimal score. An obvious way to relax an ILP is to remove one or several constraints. Another alternative to the primal formulation is the **dual**. A dual formulation of a well-formed[4] primal maximization problem is a problem which yields **upper bounds** to the primal one [Boyd and Vandenberghe, 2004]. However, there is no obligation that any primal and dual optimal solutions match, and neither their scores. The difference between optimal scores of both problems is known as the **duality gap**. One motivation for the use of a dual formulation is the composition of systems which iteratively improve the quality of the lower and the upper bound by passing information from one to the other. We will discuss a construction method relying both on relaxation and duality, called Lagrangian relaxation, in Section 4.3.

4.2.2 Properties

As previously stated, we will only focus on boolean ILPs of the following form:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{B}^{|x|} \end{aligned}$$

In this section, we will introduce general properties of such a program. For simplicity, we suppose that the inequality constraints always describe a polytope, or, in other words, that the problem is bounded and that an optimal solution exists.

Theorem 4.7: NP-hardness of ILP solving
Finding a primal feasible solution given an ILP is NP-hard.

Proof. The minimum vertex cover is a known NP-hard problem [Dinur and Safra, 2005] and can be expressed as a compact ILP. □

[4] Importantly, with a finite optimal score.

Theorem 4.8: NP-completeness of ILP satisfiability
Given an ILP, computing a feasible solution is a NP-complete problem. Given a variable assignment, it is possible to check in polynomial time that it is a feasible solution.

Proof. See Karp [1972]. □

From theorem 4.7, we can suspect an ILP formulation of a parsing problem will not lead to an efficient optimization procedure if we use an off-the-shelf solver. By theorem 4.8, even finding a feasible solution or an approximate solution is not straightforward. However, checking if a variable assignment is a feasible solution is straightforward. A common workaround is to solve the linear relaxation of an ILP, that is an ILP where the integrality constraint has been removed. Now, we will show that even if solving a LP can be realized with a polynomial time complexity, it only provides little information about the ILP optimal solution aside from an upper bound.

Theorem 4.9: LP solving is in P
Given a LP, a feasible solution can be found with a polynomial-time complexity.

Proof. See Khachiyan [1980], Karmarkar [1984]. □

However, deducing an ILP's optimal solution from its linear relaxation solution is not a trivial task. This can be easily shown with the following artificial program with variables x_1 and x_2:

$$\begin{aligned}
\underset{x_1,x_2}{\text{argmax}} \quad & x_1 && (4.6)\\
\text{s.t.} \quad & -0.1x_1 - x_2 \geq -1 && (4.7)\\
& -0.09x_1 + x_2 \geq 0.5 && (4.8)\\
& x_1 \geq 0 && (4.9)\\
& x_2 \geq 0 && (4.10)\\
& x_1, x_2 \in \mathbb{Z} && (4.11)
\end{aligned}$$

The single optimal solution of this problem is $\hat{x}_1 = 5, \hat{x}_2 = 1$. However, as can be observed in Figure 4.3, the optimal solution of the linear relaxation, that is the same program but with constraint (4.11) removed, is different. Moreover, the distance between these two points can be arbitrarily large: it is easy to change the two constraints (4.7)-(4.8) in order to increase it while maintaining the same set of integer feasible solutions. For the sake of completeness, we would like to point out that previous work proposed a characterisation of the distance between optimal solutions of an ILP and its linear relaxation [Blair and Jeroslow, 1979, Cook et al., 1986], where the distance is bounded with respect to the coefficients in matrix A. However, as showed with the example, even if the distance is bounded, it does not mean that it is close nor easy to reach. More

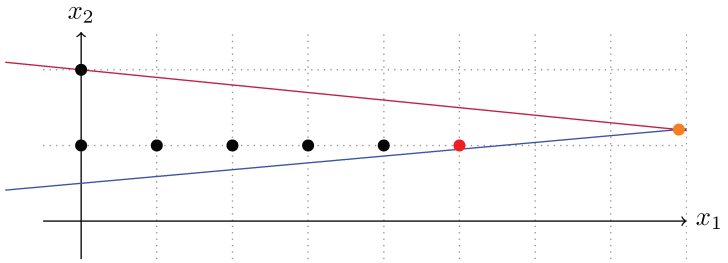


Figure 4.3: Graphical representation of the search of the ILP (4.6)-(4.11). The purple (respectively blue) line depicts constraint (4.7) (respectively (4.8)). Black dots are feasible solutions and the red dot is the single optimal solution. The orange dot is the optimal solution of the linear relaxation.

recently, Martins et al. [2009b] studied a similar bound in the case of linear relaxation of boolean variables only.

Definition 4.10: Integrality property
A LP has the integrality property if all the vertices of the polytope described by the constraints have integer coordinates only.

It is straightforward to see that the optimal solution of a LP is always a vertex of the polytope described by its constraints. Thus, by Definition 4.10, given an ILP, if its linear relaxation has the integrality property, then both problems have the same optimal solution set. The program defined in Example 4.6 has the integrality property. Thus, solving the MSA is equivalent to solving its linear relaxation.[5] This will be of special interest in the following section where we will draw connections between linear and Lagrangian relaxation. Proving that a LP has the integrality property is beyond the scope of this thesis. Several methods exist, including proving that the matrix A is totally unimodular and defining the facets of the convex hull of integer solutions.

Definition 4.11: Standard dual program
Given a primal LP of the form:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + c \\ \text{s.t.} \quad & Ax \leq b \end{aligned}$$

its **standard dual program** is given by:

$$\begin{aligned} \underset{\lambda}{\text{argmin}} \quad & b^\top \lambda + c \\ \text{s.t.} \quad & A^\top \lambda = w \\ & \lambda \geq 0 \end{aligned}$$

[5] However, this is not true for all the formulations of the problem. For example, the uni-commododity formulation of the MSA proposed in Martins et al. [2009a] does not have this property.

where $\lambda \in \mathbb{R}^{|\lambda|}$ is the vector of dual variables.

The dual program of Definition 4.11 has many applications. As an example, one variant of the simplex algorithm constantly swaps between the primal and standard dual formulations. Note that the number of constraints and the number of variables are swapped when transforming one formulation into the other. One can easily observe this because the matrix A is transposed, meaning its number of rows, that is the number of constraints (respectively variables), becomes the number of columns, that is the number of variables (respectively constraints), in the standard dual. A well-known and interesting fact of the LP standard dual formulation is that both programs have the same optimal score, that is $w^\top \hat{x} = b^\top \hat{\lambda}$. Moreover, by means of the **complementary slackness** theorem, an optimal solution of one formulation can be deduced from an optimal solution of the other. However, we are interested in ILPs. If the ILP has the integrality property, it shares similar properties with its standard dual. However, defining such a program is a difficult task, commonly referred to as polyhedral study. If it has not the integrality property, there exists a **dual gap**, or in other words $w^\top \hat{x}$ is strictly lower than $b^\top \hat{\lambda}$. In the next section, we present an alternative formulation called the **Lagrangian dual** based on **Lagrangian relaxation**.

4.3 Lagrangian relaxation

Lagrangian relaxation [Geoffrion, 1974, Beasley, 1993, Lemaréchal, 2001, Guignard, 2003] is an optimization technique that has been successfully used by both optimization [Held and Karp, 1970, 1971, Ceria et al., 1998] and NLP communities [Koo et al., 2010, Le Roux et al., 2013, Almeida and Martins, 2013, Das et al., 2012]. Intuitively, given an ILP[6], a set of constraints making the program difficult to solve is identified. Observing that if these constraints were absent then the program would be easy to solve, they are removed and introduced as penalties in the objective.

4.3.1 Definitions

Let a primal ILP be of the following form:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e && (4.12)\\ \text{s.t.} \quad & Ax \leq b && (4.13)\\ & Cx \leq d && (4.14)\\ & x \in \mathbb{B}^{|x|} && (4.15) \end{aligned}$$

where a single constraint matrix $\begin{bmatrix} A \\ C \end{bmatrix}$ has been horizontally divided into two matrices A and C, and similarly for vectors b and d. We suppose that (4.13) and (4.14) are the sets of **easy** and **hard** constraints, respectively. Formally,

[6] Lagrangian relaxation can be applied to optimization programs of other families, but we will restrict our presentation to ILPs.

we qualify a set of constraints as easy if and only if solving an affine objective with respect to them can be done efficiently thanks to a polynomial algorithm. Then, **hard** constraints can be interpreted as supplementary constraints that prevent the use of this algorithm. Thus, one aims to remove them in order to easily solve the problem. Lagrangian relaxation provides a framework which achieves this goal while exposing attractive properties.

Definition 4.12: Lagrangian relaxation
Given an ILP defined as (4.12)-(4.15), we relax the constraint set (4.14) and introduce them as penalties in the objective:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e - \lambda^\top (Cx - d) && (4.16) \\ \text{s.t.} \quad & Ax \leq b && (4.17) \\ & x \in \mathbb{B}^{|x|} && (4.18) \end{aligned}$$

where $\lambda \geq \mathbf{0}$ is an arbitrary vector of **Lagrangian multipliers**.

One of the main motivations for using Lagrangian relaxation is that the newly defined objective can be formulated as a new ILP with reparameterized weights. Moreover, it can be easily solved because of the premise on easy constraints:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w'^\top x + e' \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{B} \end{aligned}$$

where w' and e' are the reparameterized vectors:

$$\begin{aligned} w' &= w + \lambda^\top C \\ e' &= e - \lambda^\top d \end{aligned}$$

It is a relaxation because, for any set of multipliers $\lambda \geq \mathbf{0}$, its optimal score is an upper bound to the primal optimal score, see Section 4.3.2 for the proof. As of now, we will consider that solving the Lagrangian relaxation is a closed problem, removing the need of an optimization program formulation. Thus, we define $X = \{x | Ax \leq b \text{ and } x \in \mathbb{B}^{|x|}\}$ the set of feasible solutions of the Lagrangian relaxation and $f(x; \lambda) = w'^\top x + e'$ the reparameterized score function. The optimal solution score with respect to the set of Lagrangian multipliers can then simply be noted:

$$L(\lambda) = \max_{x \in X} f(x; \lambda)$$

Thus, in order to find the tightest upper bound as possible, we aim to search the λ assignment that minimizes $L(\lambda)$.

Definition 4.13: Lagrangian dual
Given a Lagrangian relaxation following Definition 4.12, the **Lagrangian dual problem** is:

$$\begin{aligned} \underset{\lambda}{\text{argmin}} \quad & L(\lambda) \\ & \lambda \in \mathbb{R}^+ \end{aligned}$$

Thus, $L(\lambda)$ is called the **Lagrangian dual objective**.

Example 4.14: MSA with out-degree constraints
Given a graph $G = (V, A)$, we wish to compute the MSA where each vertex must have at most d outgoing arcs. This problem can be formulated thanks to the following ILP:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & \sum_{a \in A} x_a w_a & & (4.19) \\ \text{s.t.} \quad & x(\delta^{in}(v_0)) = 0 & & (4.20) \\ & x(\delta^{in}(v)) = 1 & \forall v \in V^+ & (4.21) \\ & x(\delta^{in}(W)) \geq 1 & \forall W \subseteq V^+ & (4.22) \\ & x(\delta^{out}(v)) \leq d & \forall v \subseteq V & (4.23) \\ & x \in \mathbb{B}^c & & (4.24) \end{aligned}$$

where inequalities (4.23) are the out-degree constraints. Obviously, if the latter were absent, the program would be easy to solve: they are ideal candidates for relaxation. We build the following Lagrangian relaxation or Lagrangian dual objective:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & \sum_{a \in A} x_a w_a - \sum_{v \in V} \lambda_v (x(\delta^{out}(v)) - d) & & (4.25) \\ \text{s.t.} \quad & (4.20) - (4.22) & & (4.26) \\ & x \in \mathbb{B}^c & & (4.27) \end{aligned}$$

with $\lambda \geq \mathbf{0}$ the vector of Lagrangian multipliers indexed by vertices. The objective (4.25) can be rewritten as:

$$\begin{aligned} & \sum_{a \in A} x_a w_a - \sum_{v \in V} \lambda_v (x(\delta^{out}(v)) - d) \\ = \quad & \sum_{a \in A} x_a w_a - \sum_{v \in V} \lambda_v x(\delta^{out}(v)) + \sum_{v \in V} \lambda_v d \end{aligned}$$

By definition of the notation, we have $x(\delta^{out}(v)) = \sum_{v \to v' \in \delta^{out}(v)} x_{v \to v'}$. Let $e' = \sum_{v \in V} \lambda_v d$ be the reparameterized constant:

$$= \quad \sum_{a \in A} x_a w_a - \sum_{v \in V} \sum_{v \to v' \in \delta^{out}(v)} \lambda_v x_{v \to v'} + e'$$

The double sum can be rewritten as:

$$\begin{aligned} &= \sum_{a \in A} x_a w_a - \sum_{v \to v' \in A} \lambda_v x_{v \to v'} + e' \\ &= \sum_{v \to v' \in A} x_{v \to v'} (w_{v \to v'} - \lambda_v) + e' \end{aligned}$$

Thus, we can define the reparameterized weight vector w' with $w'_{v \to v'} = w_{v \to v'} - \lambda_v$:

$$= x^\top w' + e'$$

Thus, the Lagrangian dual objective can be computed with the standard MSA algorithm with updated weights. Building the Lagrangian dual is then straightforward.

Historically, this relaxation with $d = 1$ has been successfully used as a heuristic to solve the travelling salesman problem [Held and Karp, 1970, 1971].

In the next section, we will study various properties of the Lagrangian dual. An optimization method for problems of this form called **subgradient descent** is introduced in section 4.4.

4.3.2 Properties

In this section, we expose attractive properties of the Lagrangian dual. First, we formally prove that the Lagrangian relaxation is indeed a relaxation of the primal. Then, we give necessary and sufficient conditions under which the Lagrangian relaxation optimal solution is also the optimal primal solution. Thus, if we solve the Lagrangian dual by iteratively solving the Lagrangian relaxation with different multipliers, we can easily check if a solution of the relaxation solution is optimal. Moreover, we will prove that the Lagrangian dual is a non-differentiable but convex function. This leads us to the use of the subgradient descent algorithm (Section 4.4) as a solving method [Shor, 1985].

Theorem 4.15: Weak Lagrangian duality
For any vector of multipliers $\lambda \geq \mathbf{0}$, the dual objective $L(\lambda)$ is an upper bound to the primal optimal score, that is $w^\top \hat{x} + e \leq L(\lambda)$.

Proof. The proof of theorem 4.15 is straightforward:

$$\begin{aligned} w \cdot \hat{x} + e &\leq w\hat{x} + e + \lambda^\top (C\hat{x} - d) \\ &\leq f(\hat{x}; \lambda) \\ &\leq \max_{x \in X} f(x; \lambda) \\ &\leq L(\lambda) \end{aligned}$$

The first line holds because $\hat{x}$ is a primal feasible solution, thus $C\hat{x} - d \leq 0$, and $\lambda \geq \mathbf{0}$. Moreover, maximizing $f(;\lambda)$ over X may only provide higher or equal scores than the assignment $\hat{x} \in X$. □

Theorem 4.16: Strong Lagrangian duality
If, for any λ and $\bar{x} = \text{argmax}_{x \in X} f(x;\lambda)$, it holds that $\lambda^\top (C\bar{x} - d) = 0$ and $C\bar{x} \leq d$, then $\bar{x} = \hat{x}$ is a primal optimal solution. The first condition is called **complementary slackness** and the second **primal feasibility**.

Proof. By theorem 4.15, we know that $w^\top \hat{x} + e \leq f(\bar{x};\lambda)$. Moreover, from prerequisites we have $w^\top \hat{x} + e \leq w^\top \bar{x} + e$. Because $B\bar{x} \leq d$, it is a primal feasible solution so $w^\top \hat{x} + e \geq w^\top \bar{x} + e$. Thus, trivially, $w^\top \hat{x} + e = w^\top \bar{x} + e$.□

Example 4.17: Optimal Lagrangian multipliers
We apply the Lagrangian relaxation in Example 4.14 to the toy graph defined in Figure 4.4a. The out-degree is fixed to $d = 1$, i.e. each node must have at most one outgoing arc in the spanning arborescence. First, we set λ to be a null vector. The Lagrangian relaxation solution is then simply the MSA with original weights, as depicted on Figure 4.4a.

Intuitively, we want to penalize outgoing arcs from v_0 because the solution contains two of them. Thus, we set the following multipliers:

$$\begin{aligned} \lambda_{v_0} &= 2 \\ \lambda_{v_1} &= 0 \\ \lambda_{v_2} &= 0 \end{aligned}$$

The reparameterized weight of arc $v_0 \to v_1$ is then (see Example 4.14):

$$w'_{v_0,v_1} = w_{v_0,v_1} - \lambda_{v_0} = 1 - 2 = -1$$

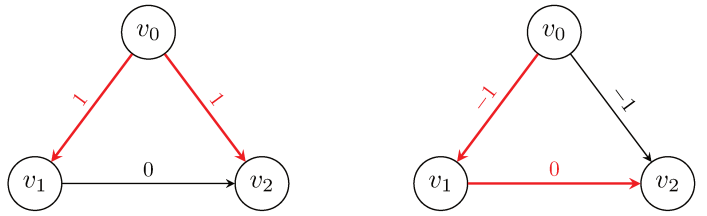
and similarly for arc $v_0 \to v_2$. The weight of arc $v_1 \to v_2$ is left unchanged. For now, we omit the bias e' as it does not impact the optimal solution. The MSA solution with reparameterized weights is depecited in Figure 4.4b.

This solution is optimal if and only if complementary slackness and primal feasibility hold (Theorem 4.16). Primal feasibility is trivial to check: each vertex has indeed at most one outgoing arc. We are left with complementary slackness:

$$\begin{aligned} \lambda^\top (Cx - d) &= \sum_{v \in V} \lambda_v (x(\delta^{\text{out}}(v)) - d) \\ &= \lambda_{v_0}(1-1) + \lambda_{v_1}(1-1) + \lambda_{v_0}(0-1) \\ &= 0 \end{aligned}$$

proving that the solution in Figure 4.4b is a primal optimal solution.

Theorems 4.15 and 4.16 state that the Lagrangian relaxation provides an upper bound to the primal problem and that there may exist an assignment



(a) Graph with original weights

(b) Graph with reparameterized weights

Figure 4.4: Red bold arcs denotes MSAs.

of multipliers so that the solution of the relaxed problem is equal to the primal optimal solution, respectively. Unfortunately, Lagrangian relaxations can introduce a dual gap, meaning the primal and dual optimal solutions do not match. Thus, whether or not to choose Lagrangian relaxation instead of linear relaxation is a matter of computational cost and dual tightness. For example, in Example 4.14, the dual objective has a exponential number of constraints but we nerveless efficiently optimize it with the Chu-Liu-Edmonds' algorithm. In general, if the relaxed problem can be solved thanks to a specialized polynomial time algorithm, then it is worth trying to apply Lagrangian relaxation. In the next subsection, we discuss the specific case of equality constraints relaxation. A generic and experimentally efficient method to minimize the Lagrangian dual, called subgradient descent, is introduced in Section 4.4.

4.3.3 Equalities and dual decomposition

Let a primal be of the form:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e && (4.28)\\ \text{s.t.} \quad & Ax \leq b && (4.29)\\ & Cx = d && (4.30)\\ & x \in \mathbb{B}^c && (4.31) \end{aligned}$$

where (4.30) is the set of hard constraints. We obtain the following program by relaxing constraints (4.30):

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e + \lambda^\top (Cx - d) && (4.32)\\ \text{s.t.} \quad & Ax \leq b && (4.33)\\ & x \in \mathbb{B}^c && (4.34) \end{aligned}$$

with λ an **unconstrained** set of Lagrangian multipliers. To understand why the Lagrangian multipliers are unconstrained, suppose we equivalently replace the equalities with $Cx \leq d$ and $-Cx \leq -d$ and relax them:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & w^\top x + e - \lambda^1 (Cx - d) - \lambda^2 (-Cx + d) && (4.35)\\ \text{s.t.} \quad & Ax \leq b && (4.36)\\ & x \in \mathbb{B}^c && (4.37) \end{aligned}$$

with λ^1 and λ^2 two sets of positive Lagrangian multipliers. The objective function can be rewritten as:

$$\begin{aligned} & w^\top x + e - \lambda^1(Cx - d) - \lambda^2(-Cx + d) \\ = \; & w^\top x + e - \lambda^1(Cx - d) + \lambda^2(Cx - d) \\ = \; & w^\top x + e + (\lambda^2 - \lambda^1)(Cx - d) \end{aligned}$$

Thus, $\lambda = \lambda^2 - \lambda^1$ is an unconstrained set of multipliers.

Note that when relaxing equalities only, primal feasibility implies complementary slackness. Indeed, it is easy to see that if $Bx = d$, then $\lambda(Bx - d) = 0$ for any λ.

Definition 4.18: Dual decomposition
A **dual decomposition** is a Lagrangian dual with a dual objective that decomposes in several independent subproblems.

Let (P) be a primal program defined as:

$$\begin{aligned} (P) \qquad \underset{x}{\text{argmax}} \quad & w^\top x \\ \text{s.t.} \quad & A^i x \leq b^i && 1 \leq i \leq l \\ & x \in \mathbb{B}^{|x|} \end{aligned}$$

with A^i and b^i matrices and vectors defining l sets of constraints. We suppose that maximizing x with respect to only one set of constraints $A^i x \leq b^i$ can be computed efficiently. In order to perform dual decomposition, we first need to reformulate the program. The vector of variables x is replaced by distinct copy vectors $x^i \in \mathbb{B}^{|x|}, 1 \leq i \leq l$. We introduce a vector of variables z called the witness vector that is used to ensure that all copy vectors are equal. Let w^i, $1 \leq i \leq l$, be weight vectors such that $\sum_{i=1}^{l} w^i = w$. Then, the problem can be reformulated as:

$$\begin{aligned} (P') \qquad \underset{x}{\text{argmax}} \quad & \sum_{i=1}^{l} w^{i\top} x^i && && (4.38) \\ \text{s.t.} \quad & A^i x^i \leq b^i && 1 \leq i \leq l && (4.39) \\ & z = x^i && 1 \leq i \leq l && (4.40) \\ & x^i \in \mathbb{B}^{|x|} && 1 \leq i \leq l && (4.41) \\ & z \in \mathbb{R}^{|x|} && && (4.42) \end{aligned}$$

Variables z in constraint set (4.40) ensure that all copies $\{x^i\}$ are equal. ILPs (P) and (P') are obviously equivalent. By relaxing constraints (4.40), we build

the following Lagrangian relaxation:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & \sum_{i=1}^{l} w^{i\top} x^i + \sum_{i=1}^{l} \lambda^i (z - x^i) & \\ \text{s.t.} \quad & A^i x^i \leq b^i & 1 \leq i \leq l \\ & x^i \in \mathbb{B}^{|x|} & 1 \leq i \leq l \\ & z \in \mathbb{R}^{|x|} & \end{aligned}$$

Note that z is unconstrained. If the set of Lagrangian multipliers is defined such as $\sum_{i=1}^{l} \lambda^i \neq 0$, then the term $\sum_{i=1}^{l} \lambda^i z$ makes the objective unbounded. Thus, we restrict the domain of the Lagrangian multipliers to $\Lambda = \{\{\lambda^i\} | \sum_i \lambda^i = 0\}$. The Lagrangian relaxation then becomes:

$$\begin{aligned} \mathcal{L}^*(\{\lambda^i\}) = \max_{x} \quad & \sum_{i=1}^{l} w'^{i\top} x^i & \\ \text{s.t.} \quad & A^i x^i \leq b^i & 1 \leq i \leq l \\ & x^i \in \mathbb{B}^c & 1 \leq i \leq l \end{aligned}$$

with $w'^i = w^i + \lambda^i$ the reparameterized weight vectors. Note that z has disappeared from the objective. Given a set of multipliers $\{\lambda^i\}$, computing the Lagrangian objective amounts to solving l distinct subproblems:

$$\begin{aligned} (P_i) \qquad \max_{x^i} \quad & w'^{i\top} x^i \\ \text{s.t.} \quad & A^i x^i \leq b^i \\ & x^i \in \mathbb{B}^{|x|} \end{aligned}$$

If the solution of each subproblem P_i can be easily computed, then computing the Lagrangian objective is easy. Moreover, the subproblems can be parallelized.

The Lagrangian dual of the previous problem is simply:

$$\min_{\{\lambda^i\} \in \Lambda} \quad \mathcal{L}^*(\{\lambda^i\})$$

Note that the set of dual feasible solutions lies in Λ instead of the set of real vectors (or instead of the set of positive real if we compare to inequalities relaxation).

Example 4.19: MSA with out-degree constraints (2)
We propose a different relaxation of Example 4.14 via dual decomposition. Remember that, given a graph $G = (V, A)$, we wish to compute the MSA where each vertex must have at most d outgoing arcs. This problem can be formulated thanks to the following ILP:

$$\begin{aligned} \underset{x}{\text{argmax}} \quad & \sum_{a \in A} x_a w_a & & (4.43) \\ \text{s.t.} \quad & x \in \mathcal{X} & & (4.44) \\ & x(\delta^{out}(v)) \leq d & \forall v \subseteq V & (4.45) \\ & x \in \mathbb{B}^{|x|} & & (4.46) \end{aligned}$$

with $\mathcal{X}$ the arborescence polytope as defined by constraints (4.20)-(4.22). Let $\{x^i\} = \{x^1, x^2\}$ be the copy variables, $\{w^i\} = \{w^1, w^2\}$ be any weight vectors such that $\sum_i w^i = w$ and $z \in \mathbb{R}^{|x|}$ be the witness vector. Then, the problem can be reformulated as:

$$\begin{aligned}
\underset{x}{\text{argmax}} \quad & \sum_i w^{i\top} x^i && && (4.47)\\
\text{s.t.} \quad & x^1 \in \mathcal{X} && && (4.48)\\
& x^2(\delta^{out}(v)) \leq d && \forall v \subseteq V && (4.49)\\
& z = x^i && \forall i \in \{1,2\} && (4.50)\\
& x^1, x^2 \in \mathbb{B}^{|x|} && && (4.51)
\end{aligned}$$

We build the following Dual objective by relaxing constraints (4.50):

$$\begin{aligned}
\underset{x}{\text{argmax}} \quad & \sum_i w'^{i\top} x^i && && (4.52)\\
\text{s.t.} \quad & x^1 \in \mathcal{X} && && (4.53)\\
& x^2(\delta^{out}(v)) \leq d && \forall v \subseteq V && (4.54)\\
& x^1, x^2 \in \mathbb{B}^c && && (4.55)
\end{aligned}$$

with $w'^i = w^i + \lambda^i$ and $\{\lambda^i\} \in \Lambda$ is the set of Lagrangian multipliers. Maximizing this program reduces to computing two distinct subproblems:

$$\begin{array}{llll}
(P_1) & \underset{x}{\text{argmax}} \quad w'^{1\top} x^1 & (P_2) & \underset{x}{\text{argmax}} \quad w'^{2\top} x^2 \\
& \text{s.t.} \quad x^1 \in \mathcal{X} & & \text{s.t.} \quad x^2(\delta^{out}(v)) \leq d \quad \forall v \subseteq V \\
& \qquad\quad x^1 \in \mathbb{B}^{|x|} & & \qquad\quad x^2 \in \mathbb{B}^{|x|}
\end{array}$$

Subproblem P_1 can be computed thanks to the MSA algorithm with updated weights. Subproblem P_2 searches, for earch vertex, the d outgoing arcs with maximum positive weights. With a convenient data structure, this can be computed in quadratic time, similar to the MSA.

4.4 Subgradient descent

Subgradient descent is an optimization technique which aims at finding the minimum value of a non-differentiable convex function.[7] The technique is supported by the subgradient theory, a generalization of the gradient (Section 4.4.1). In particular, we will show that for the family of functions defined by Lagrangian duals of ILPs, it this straightforward to compute a subgradient at any point. The pseudo-code of the algorithm is given in Section 4.4.2.

[7] If we search the maximum of a non-differentiable concave function, the similar method is called supergradient ascent. We choose to focus on subgradient descent in this section because the Lagrangian dual of a maximization problem is a minimization problem.

4.4.1 Theory

Gradient descent is a well-known optimization technique for differentiable functions [Boyd and Vandenberghe, 2004]. Given a differentiable function, at any point of the domain, the opposite of the gradient at this point is the fastest decreasing direction. Thus, a function is iteratively minimized by moving in that direction with a given stepsize. A good stepsize can be computed via any line search or trust region technique. Moreover, when the gradient is null, the given point is a stationary point (a local minimum, a local maximum or an inflection) or, if the function is convex, a global minimum. A notable use of gradient descent based optimization is the backpropagation algorithm for neural network parameters estimation [Linnainmaa, 1976, LeCun et al., 2012]. But what if the studied function is not differentiable? If it is nonetheless convex, a common alternative is the subgradient descent method [Shor, 1985]. Given a point in the domain of a function, its subgradient set is the set of supporting hyperplanes of this function. Notably, if the function is differentiable at a given point, the subgradient set at this point is the singleton containing the gradient. Thus, a similar iterative algorithm can be used. However, subgradient descent misses many of the interesting properties of gradient descent: a subgradient is not necessarily a direction of decreasing and several frequent line search and trust region methods are not applicable. In this section, we introduce the theory that supports the subgradient descent algorithm. Experimentally, despite its drawbacks, it is a simple method that has proven to be efficient in many applications.

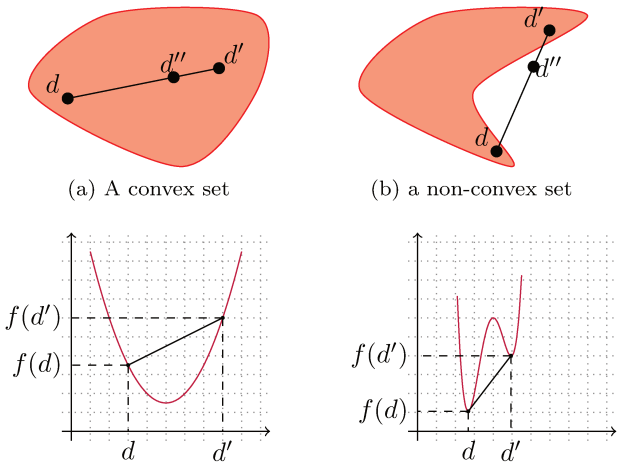
Definition 4.20: Convex set
A set D is convex if and only if:

$$\forall d, d' \in D, \epsilon \in [0,1]: \quad \epsilon d + (1-\epsilon)d' \in D$$

Definition 4.21: Convex function
Given a function $f : D \to \mathbb{R}$, f is convex if and only if D is a convex set and:

$$\forall d, d' \in D, \epsilon \in [0,1]: \quad f(\epsilon d + (1-\epsilon)d') \leq \epsilon f(d) + (1-\epsilon) f(d')$$

A set is convex if, given two arbitrary elements of it, any convex combination of them is also included in the former set. Figure 4.5a illustrates a convex set: if d'' is a convex combination of $d, d' \in D$, then $d'' \in D$ is always true. In Figure 4.5b, the set is non-convex, meaning there exists a point d'' which is a convex combination of two $d, d' \in \mathcal{D}$ with $d'' \notin \mathcal{D}$. The similar definition for a function can be understood as, for any two given points of a function with a convex domain, all their convex combinations must lie *above* the function (see Figure 4.5c). If not, the function is non-convex (See Figure 4.5d). We will now develop this last point in order to obtain a formal definition of a subgradient.



(a) A convex set (b) a non-convex set

(c) A convex function (d) A non-convex function

Figure 4.5: Convexity

Definition 4.22: Graph
Given a function $f : D \to \mathbb{R}$, its graph is the set of tuples in $D\mathbb{R}$ defined by:

$$\{(d, f(d))|d \in D\}$$

Definition 4.23: Epigraph
Given a function $f : D \to \mathbb{R}$, its epigraph is the set of tuple in $D\mathbb{R}$ defined by:

$$\mathbf{epi}\, f = \{(d, r)|d \in \mathcal{D} \text{ and } f(d) \leq r\}$$

Definition 4.24: Supporting hyperplane
Given a set S, a supporting hyperplane of S is a hyperplane such that:

- S is entirely contained in one of the half-spaces separated by the hyperplane;
- S has at least one boundary point with the hyperplane.

Definition 4.25: Subgradient
Given a function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, a subgradient at $d \in D$ is a vector $g \in \mathbb{R}^n$ such that:

$$\forall d' \in \mathcal{D} : \quad f(d') \geq f(d) + g^\top (d' - d)$$

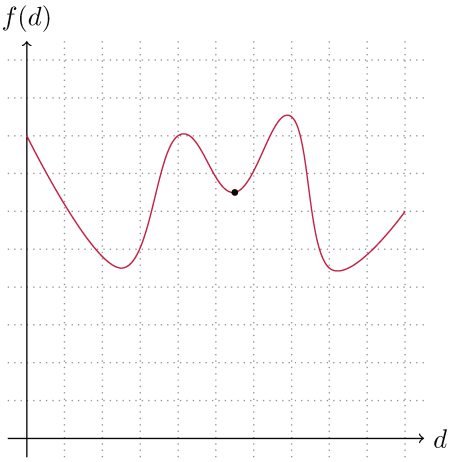


Figure 4.6: No supporting hyperlane to the epigraph of the function exists at the black dot. Thus, the subgradient is undefined at this point.

The set of subgradients at point d is called the **subdifferential** and is denoted $\partial f(d)$.

The graph of f can be understood as the set of points which are usually used in order to draw it. The epigraph of f is the set of points which are, roughly speaking, *above* the graph. Thus, equivalently to Definition 4.21, a function is convex if its domain and its epigraph are convex sets. Let D be a convex set and $f : D \to \mathbb{R}$ be a convex function. Given a point $d \in D$, a subgradient $g \in \partial f(d)$ is a supporting hyperplane of **epi** f with $(d, f(d))$ being a boundary point. Alternatively and equivalently, the graph $\{(d', f(d)+g^\top(d'-d))|d' \in \mathcal{D}\}$ defines a supporting hyperplane of f. It is easy to see that the function $f'(d'; g) = f(d) + g^\top(d' - d)$ is a global subestimator of the function. As such, moving along the subgradient of a given point may be a good idea in order to minimize a function. However, such an approach is appealing only if a subgradient exists at all points of the domain of f. Figure 4.6 shows an example of a non-convex function which has at least one point where its subgradient is undefined. It can be proven that a subgradient exists at any point of a convex function via the hyperplane separation theorem [Boyd and Vandenberghe, 2004]. We focus on the specific case of Lagrangian duals in Subsection 4.4.2.

So far, we showed that a subgradient is a global subestimator of a convex function and we suppose that there exists at least one subgradient at a given point of the domain. The main motivation behind subgradient descent is that if we want to find the minimum of f, it may not be a bad idea to move along the direction of descent of one of its subgradients. Unlike the gradient descent case, there is no guarantee that following this direction always results in a diminution of f. However, it always results in the decrease of the distance to a point which attains the minimum of f. If, at first glance, this difference may not seem important, it has a practical downside: if, after moving with a given stepsize,

we obtain a point with a higher value, we cannot certify that this is because the stepsize was too important. This may be counter-intuitive because it is not observable in a function which has a 2 dimensions graph.

Theorem 4.26: Descent direction
Given a convex function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, $d \in D$, $g \in \partial f(d)$ a subgradient at d and $\hat{d} = \text{argmin}_{d \in D} f(d)$, $\exists \epsilon \in \mathbb{R}$ such that:

$$\|(d - \epsilon g) - \hat{d}\| \leq \|d - \hat{d}\|$$

Thus, the subgradient at a given point of a convex function is a descent direction.

Proof. We suppose that d is not a minimizer of f.

$$\|(d - \epsilon g) - \hat{d}\|^2 = \|d - \hat{d}\|^2 + \|\epsilon g\|^2 - 2\epsilon g^\top (d - \hat{d})$$

By Definition 4.25 of the subgradient $f(d) - f(\hat{d}) \geq g^\top (d - \hat{d})$

$$\begin{aligned} &\leq \|d - \hat{d}\|^2 + \|\epsilon g\|^2 - 2\epsilon (f(d) - f(\hat{d})) \\ &\leq \|d - \hat{d}\|^2 + \epsilon^2 \|g\|^2 - 2\epsilon (f(d) - f(\hat{d})) \end{aligned}$$

Therefore, we have $\|(d - \epsilon g) - \hat{d}\|^2 < \|d - \hat{d}\|^2$ when $\epsilon^2 \|g\|^2 - 2\epsilon (f(d) - f(\hat{d}) < 0$. Let $a = \|g\|^2$ and $b = 2(f(\hat{d}) - f(d))$. We now have to prove that there exists at least one ϵ value for which $a\epsilon^2 - b\epsilon < 0$. Note that the left side roots are $\frac{b - \sqrt{b^2}}{2a} = 0$ and $\frac{b + \sqrt{b^2}}{2a} = \frac{b}{a}$. We have $a > 0$ and $b > 0$ because d is not a minimizer of f, thus the gradient cannot be null and $f(\hat{d}) < f(d)$. Thus, $a\epsilon^2 - b\epsilon < 0$ has at least one solution because there exists two distinct left-side roots. Thus:

$$\leq \|d - \hat{d}\|^2$$

meaning there always exists a stepsize for which the subgradient is a descent direction. □

Definition 4.27: Projection
A function $p : D \to D'$ is a projection from set D to set D' if and only if, for any $d \in D$ and $d' \in D'$:

$$\|d - p(d)\| \leq \|d - d'\|$$

That is, $p(d)$ is the closest point to d in D'.

Theorem 4.28: Projected descent direction
Given a convex function $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, $d \in D$, $g \in \partial f(d)$ a subgradient at d, $\hat{d} = \text{argmin}_{d \in D} f(d)$ and a projection function $p : \mathbb{R}^n \to D$, $\exists \epsilon \in \mathbb{R}$ such that:

$$\|p\,(d - \epsilon g) - \hat{d}\| \leq \|d - \hat{d}\|$$

Thus, the subgradient at a given point of a convex function is a descent direction after projection in the domain.

Proof. From Theorem 4.26, we have:

$$\|(d - \epsilon g) - \hat{d}\| \leq \|d - \hat{d}\|$$

By definition, the projection of a point into D is closer to the original than any other point in D. That is $\|p\,(d - \epsilon g) - \hat{d}\| \leq \|(d - \epsilon g) - \hat{d}\|$. Thus:

$$\|p\,(d - \epsilon g) - \hat{d}\| \leq \|d - \hat{d}\|$$

which ends the proof. □

4.4.2 Algorithm

In the previous section, we theoretically motivated the use of subgradient information in order to minimize a non-differentiable convex function. We showed that moving along the opposite direction of a subgradient at a given point brings closer to an optimal point. In this section, we apply subgradient descent to Lagrangian duals (see Section 4.3). We describe how a subgradient can be easily computed at any point of the Lagrangian dual objective. Moreover, we write down the complete minimization algorithm.

A Lagrangian dual is a function of the form:

$$\min_{\lambda \in \Lambda} \quad L(\lambda)$$

with Λ the domain of Lagrangian multipliers and L the Lagrangian dual objective. It is unlikely that a closed form solution for this problem exists in the general case. However, L is non-differentiable but convex (Theorem 4.29).

Theorem 4.29: Lagrangian dual objective convexity
Let $L(\lambda \in \Lambda) = \max_{x \in X} f(x; \lambda)$ be a Lagrangian dual objective with Λ a convex set and $f(x; \lambda) = h(x) + \lambda^\top l(x)$. Function h is the original primal score function and function l is the contribution of relaxed constraints. Then, L is convex but non-differentiable.

Proof. Except in trivial cases, the function is obviously non-differentiable because of the max operator. In order to prove that L is convex, we have to show that for any λ, $\lambda' \in \Lambda$ and $\epsilon \in [0, 1]$:

$$L(\epsilon\lambda + (1 - \epsilon)\lambda') \leq L(\epsilon\lambda) + L((1 - \epsilon)\lambda')$$

Let $\lambda^* = \epsilon\lambda + (1-\epsilon)\lambda'$ and $x^* = \text{argmax}_{x \in X} f(x; \lambda^*)$. Obviously, as L maximizes over x, we have:

$$f(x^*; \lambda) \leq L(\lambda)$$
$$f(x^*; \lambda') \leq L(\lambda')$$

Multiplying and then summing both inequalities, we obtain:

$$\epsilon f(x^*;\lambda) + (1-\epsilon) f(x^*;\lambda') \leq \epsilon L(\lambda) + (1-\epsilon) L(\lambda')$$

The left side can be reformulated as:

$$\begin{aligned}
\epsilon f(x^*;\lambda) + (1-\epsilon) f(x^*;\lambda') &= \epsilon(h(x^*) + \lambda^\top l(x^*)) + (1-\epsilon)(h(x^*) + \lambda'^\top l(x^*)) \\
&= \epsilon h(x^*) + \epsilon\lambda^\top l(x^*) + (1-\epsilon)h(x^*) + (1-\epsilon)\lambda'^\top l(x^*) \\
&= h(x^*) + (\epsilon\lambda + (1-\epsilon)\lambda')^\top l(x^*) \\
&= f(x^*;\lambda^*) \\
&= L(\lambda^*)
\end{aligned}$$

Thus $L(\epsilon\lambda + (1-\epsilon)\lambda') \leq L(\epsilon\lambda) + L((1-\epsilon)\lambda')$, meaning the Lagrangian dual objective is convex. Note that in this proof, we did not use any property of the search space X, the primal objective function f and neither the relaxed constraints. This is a major property that motivated the use of Lagrangian relaxation beyond the scope of linear problems [Bertsekas, 1999]. □

Theorem 4.30: Subgradient of the dual objective
Let $L(\lambda \in \Lambda) = \max_{x\in X} f(x;\lambda)$ with Λ a convex set and $f(x) = w^\top x + e + \lambda^\top(Cx - d)$ be a Lagrangian dual objective, where primal constraints $Cx \leq d$ have been relaxed. Given a set of multipliers $\lambda \in \Lambda$, let $\hat{x} = \operatorname{argmax}_{x\in X} f(x;\lambda)$. Then:

$$C\hat{x} - d \in \partial L(\lambda)$$

is a subgradient of L at λ.

Proof. Definition 4.25 states that $C\hat{x} - d$ is a subgradient at λ if and only if:

$$L(\lambda') \geq L(\lambda) + (C\hat{x} - d)^\top(\lambda' - \lambda)$$

for every $\lambda' \in \Lambda$. Starting with the right side of the inequality:

$$\begin{aligned}
& L(\lambda) + (C\hat{x} - d)^\top(\lambda' - \lambda) \\
= \;& \max_{x\in X} \left(w^\top x + e + \lambda^\top(Cx - d)\right) + (C\hat{x} - d)^\top(\lambda' - \lambda) \\
= \;& w^\top\hat{x} + e + \lambda^\top(C\hat{x} - d) + \lambda'^\top(C\hat{x} - d) - \lambda^\top(C\hat{x} - d) \\
= \;& w^\top\hat{x} + e + \lambda'^\top(C\hat{x} - d) \\
= \;& f(\hat{x};\lambda') \\
\leq \;& \max_{x\in X} f(x;\lambda') \\
\leq \;& L(\lambda')
\end{aligned}$$

Thus $C\hat{x} - d \in \partial L(\lambda)$. □

By Theorem 4.30, we can easily compute a subgradient for any given Lagrangian multiplier. We compute the dual objective solution and then $C\hat{x} - d$ is a subgradient, which basically corresponds to the quantity of violation of relaxed constraints. The Lagrangian multipliers can then be updated with the following rule:

$$\lambda = p\left(\lambda - \epsilon^t(C\hat{x} - d)\right)$$

with p a projection operator into Λ and ϵ the stepsize.[8] The stepsize can be fixed with the rule of Polyak [1987]:

$$\epsilon = \frac{L(\lambda) - LB}{\|B\hat{x} - d\|^2}$$

where LB is the best known lower bound at the current iteration. Pseudo-code of the algorithm is given in Algorithm 1. We suppose that we can use a **primal heuristic**, that is an algorithm which computes a primal solution with respect to the reparameterized weights. Importantly, the algorithm delivers a certificate of optimality if complementary slackness holds or the lower and upper bounds are equal.

Algorithm 1 The subgradient descent algorithm minimizing a Lagrangian dual.

```
lb ← −∞                                          ▷ Initialize lower bound
ub ← +∞                                          ▷ Initialize upper bound
p ← −                                            ▷ Will store the best found primal solution
λ = 0                                            ▷ Initialize Lagrangian multipliers
for 1 ≤ t ≤ T do
    x̂ = argmax_{x∈X} f(x; λ)                     ▷ Dual objective
    ub = min(ub, f(x̂; λ))
    x̄ = primal(λ)                                ▷ Primal heuristic
    if f(x̄; λ) > lb then
        lb = f(x̄; λ)
        p ← x̄
    if Cx̂ ≤ d then                               ▷ Test primal feasibility
        if λ^⊤(Cx̂ − d) = 0 then                   ▷ Test complementary slackness
            return x̂
        if f(x̂; λ) > lb then                     ▷ Update lower bound ?
            lb = f(x̂; λ)
            p ← x̂
    if lb = ub then                              ▷ If both bounds match, then p is optimal
        return p
    λ = p(λ − ε^t(Cx̂ − d))                       ▷ Update and project multipliers
return p      ▷ Return the possibly non-optimal best known primal feasible
```

4.5 Problem reduction

In the previous section, we introduced the subgradient descent algorithm in order to minimize a Lagrangian dual problem. We supposed that there exists

[8] The projection operator can be removed if the multipliers are unconstrainted.

an efficient polynomial time algorithm to solve the Lagrangian dual objective as it must be iteratively computed. But even in that case it is appealing to prune the search space when it is big so the running time of each iteration is minimized. Problem reduction [Beasley, 1993] consists in fixing certain variables while maintaining the possibility to deliver an optimality certificate: a variable is fixed to 1 (respectively 0) if it is guaranteed to be (respectively not to be) in an optimal solution.

Let (P) and (L) respectively be a primal problem and a Lagrangian dual objective of the form:

$$(P) \quad \begin{aligned} \underset{x}{\text{argmax}} \quad & h(x) \\ \text{s.t.} \quad & x \in X \\ & x \in X' \end{aligned} \qquad\qquad (L) \quad \begin{aligned} \underset{x}{\text{argmax}} \quad & f(x;\lambda) \\ \text{s.t.} \quad & x \in X \end{aligned}$$

where the constraints describing the set X' have been relaxed. The value $h(x)$ of any primal feasible solution $x \in X$ is a lower bound to the primal optimal score and, by definition, the value of $\max_{x \in X} f(x;\lambda)$ for any dual feasible solution $\lambda \in \Lambda$ is an upper bound to the primal optimal score. Consider the following Lagrangian dual objective with an additional constraint on a single variable x_i:

$$(L') \quad \begin{aligned} \underset{x}{\text{argmax}} \quad & f(x;\lambda) \\ \text{s.t.} \quad & x \in X \\ & x_i = 1 \end{aligned}$$

It is straightforward to prove that if the optimal solution score of this problem (L') is lower than the best known lower bound of (P), then we can fix $x_i = 0$ in (P) and (L'). Alternatively, if we impose an additional constraint $x_i = 0$ in (L) and observe a similar result, then we can fix $x_i = 1$. Thus, solving (L') can be used to prune the search space.

In practice, we want to rely on the structure of (L) so (L') can be computed efficiently for each variable which has not been fixed yet by modifying the optimal solution of (L). The best case scenario is a constant time algorithm and thus a linear time problem reduction algorithm.

4.6 Branch-and-bound

The primal and Lagrangian dual optimal scores may not match if there exists a dual gap. Moreover, the subgradient descent algorithm is executed for a fixed number of iterations in practice. Thus, the algorithm may not converge even if the dual is tight. The Branch-and-Bound algorithm is an exhaustive search procedure which uses the upper bound given by a dual formulation in order to prune the search space [Land and Doig, 1960]. The algorithm has two different steps which are successively executed:

1. the **branching** step splits the search space into two subproblems with disjoint solution sets;

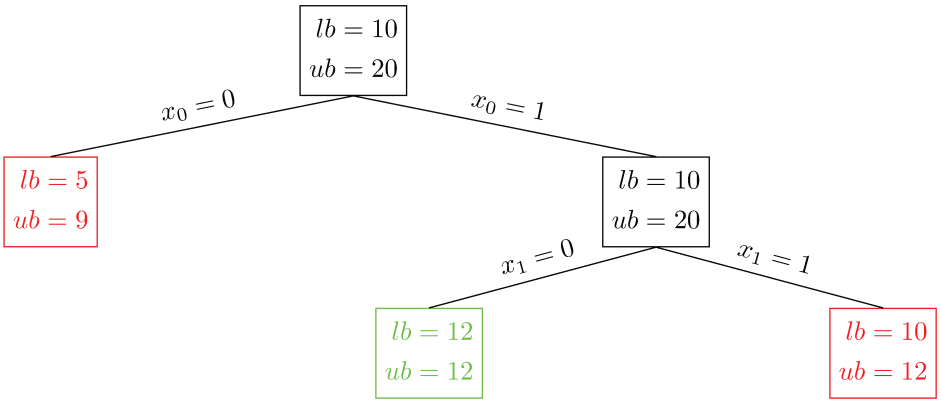


Figure 4.7: Example of a branch-and-bound search tree.

2. the **bounding** step computes lower and upper bounds with the subgradient descent algorithm.

Any subproblem which has an upper bound lower to the best known (global) lower bound can be safely removed.

Figure 4.7 shows a graphical example of a Branch-and-Bound search where lb and ub respectively denote the lower and upper bounds computed at each node. In the topmost node, we compute bounds on the complete search space. Then, as the subgradient descent algorithm did not converge, we branch into two subproblems according to variable x_0. By fixing $x_0 = 0$ we obtain an upper bound that is less than the best known lower bound, so we can safely stop the search procedure at this node. We branch again on the other node but with variable x_1. One of the nodes has converged and the other cannot improve the best known solution. So the primal optimal solution score is 12.

4.7 Conclusion

In this chapter, we explained the Lagrangian relaxation heuristic for solving ILPs. The resulting Lagrangian dual can be efficiently minimized via subgradient descent. In the examples, the maximum spanning arborescence must be computed at each iteration of the subgradient descent on a graph with reparameterized weights. Fortunately, this can be efficiently computed in quadratic time with the Chu–Liu/Edmonds' algorithm.

In Chapter 3, we showed that the Lexicalized Tree Adjoining Grammar parsing problem can be formulated as a graph problem via the derivation tree representation. The best known algorithm for this problem has a $\mathcal{O}(n^7)$ complexity with n the input sentence length. In order to tackle this complexity challenge, we propose to rely on Lagrangian relaxation. Thus, in the following we introduce novel ILPs for this parsing problem. Again, solving the Lagrangian dual objective will require to compute the maximum spanning arborescence.

Part III

Contributions

Chapter 5

Derivation tree parsing via the Yield Restricted Maximum Spanning Arborescence

In Chapter 3, we introduced motivation of phrase-structure parsing with Lexicalized Tree Adjoining Grammars (LTAGs). Algorithms which have been proposed in the literature either have a too high complexity or make unreasonable assumptions. As a reminder, the best known parser, which was proposed by Eisner and Satta [2000], is a $\mathcal{O}(n^7)$ method based on deduction rules. Moreover, supertagging as a pre-processing step cannot account for long range dependencies explicitly while transition based methods lack of optimality guarantee. Our original contribution is a new two steps pipeline for LTAG derivation tree parsing (Section 5.1).

In the first step (Section 5.1), we assign LTAG-compatible bi-lexical relations, that is we parse a well-nested arborescence with a 2-bounded block degree [Bodirsky et al., 2005]. The resulting structure is an unlabeled derivation tree. Interestingly, this step is not dependent of a specific LTAG. However, the best known parsing algorithm for this dependency parsing problem has a $\mathcal{O}(n^7)$ complexity [Gómez-Rodríguez et al., 2009]. Thus, we introduce a novel characterization of the problem in terms of graph and propose two new ILPs in Section 5.2. In Section 5.3, we show that one of the ILPs can be reduced to a more general problem that we call the Yield Restricted Maximum Spanning Arborescence. This problem has an exponential number of difficult constraints. These constraints can be relaxed resulting and introduced in the objective. However, computing the objective value and updating all Lagrangian multipliers are time consuming tasks. Thus, in Section 5.4, we propose to tackle this challenge by lazy generation of multipliers, a technique called Non-delayed Relax-and-Cut in the literature [Lucena, 2005]. We experimentally show the efficiency of our method on several datasets with different types of constraints in Section 5.5: different block degree limits and with or without the well-nestedness require-

ment.

The second step of the pipeline is a parse tree labeler: elementary trees and operations sites are assigned to the dependency structure. We propose an algorithm for this problem in Section 5.6 using the deduction rule formalism. Moreover, we prove that this step has a linear time complexity with respect to the sentence size.

5.1 Motivations

Dependency parsing has been argued to be an easier task than phrase-structure parsing, leading to the development of many parsers for the former task. We can single out two major reasons for this claim. Firstly, dependency parsers usually do not rely on a grammar. Many systems only enforce structural constraints on dependency trees. The famous dependency parser of Eisner [2000] enforcing projectivity (see Section 5.2) is still one of the most commonly used by the NLP community. Indeed, several treebanks, including the PTB conversion using the head-percolation table of Collins [2003], contain only projective dependency trees. However, other datasets contain more intricate structures. Thus, several algorithms have been proposed in order to achieve a high-coverage of dependency treebanks while maintaining a polynomial complexity [Gómez-Rodríguez et al., 2009, 2011, Pitler et al., 2012, 2013, Satta and Kuhlmann, 2014]. McDonald et al. [2005] showed that unconstrained dependency parsing, usually referred to as non-projective parsing, can be realized with a quadratic complexity. This approach can generate dependency trees which do not respect properties observed in treebanks but achieves high-accuracy results in practice [Ma and Hovy, 2017]. Secondly, most approaches achieve state of the art results while relying on a pipeline system. While it is possible to assign head-modifier relationships together with grammatical relations, most approaches rely on a post-processing parse tree labeler for the latter (see Figure 5.1).

In order to reduce LTAG parsing to a dependency parsing task, we need to incorporate grammar based constraints into the dependency framework while maintaining short parsing time. This is a challenging task. Previous work of Riedel and Clarke [2006] incorporated linguistically motivated constraints into a dependency parser. They rely on optimization methods in order to provide tractable parsing time. We follow a similar process.

An LTAG derivation tree has a highly constrained structure. Obviously, substitution sites must be filled. Moreover, a given site in an elementary tree may forbid adjunction, impose mandatory adjunction and/or provide fine-grained constraints on allowed adjunctions. For substitution and adjunction operations, non-terminal labels must match. Finally, the derivation tree must correctly transform into a continuous derived tree with respect to the word order, that is the derived tree must not contain crossing edges. Note that, contrary to the derived tree, the derivation tree may contain crossing arcs due to wrapping adjunctions, which will be discussed in the next section. These arcs appear conjointly with correct auxiliary trees, heads, operation sites and yields to ensure the derived tree correctness. Note that removing this requirement, which will be

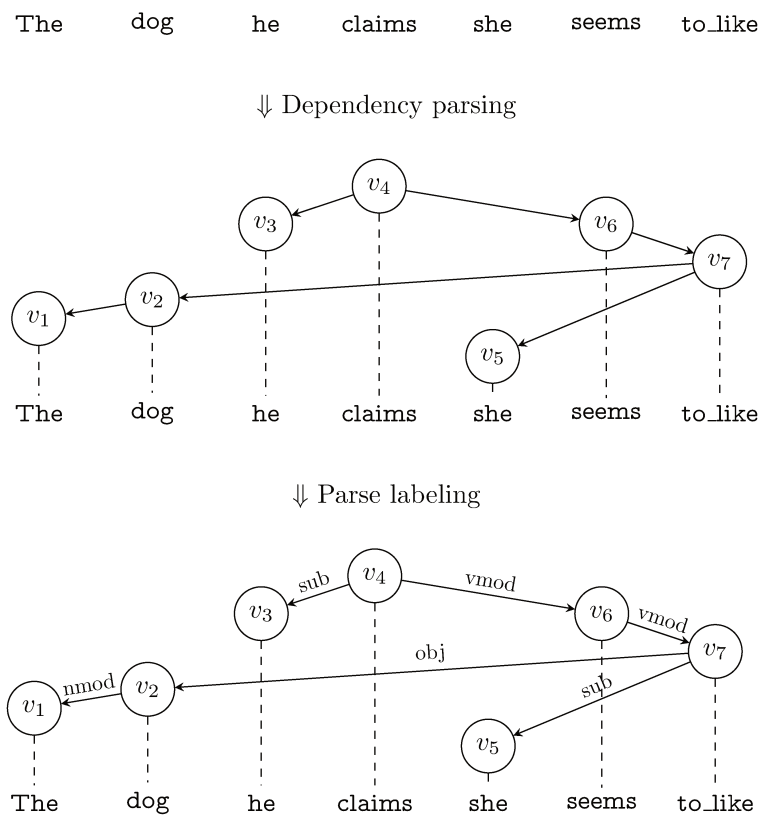


Figure 5.1: The dependency parsing pipeline. We follow the convention of chunking to like as a single lexical item [Kallmeyer and Kuhlmann, 2012].

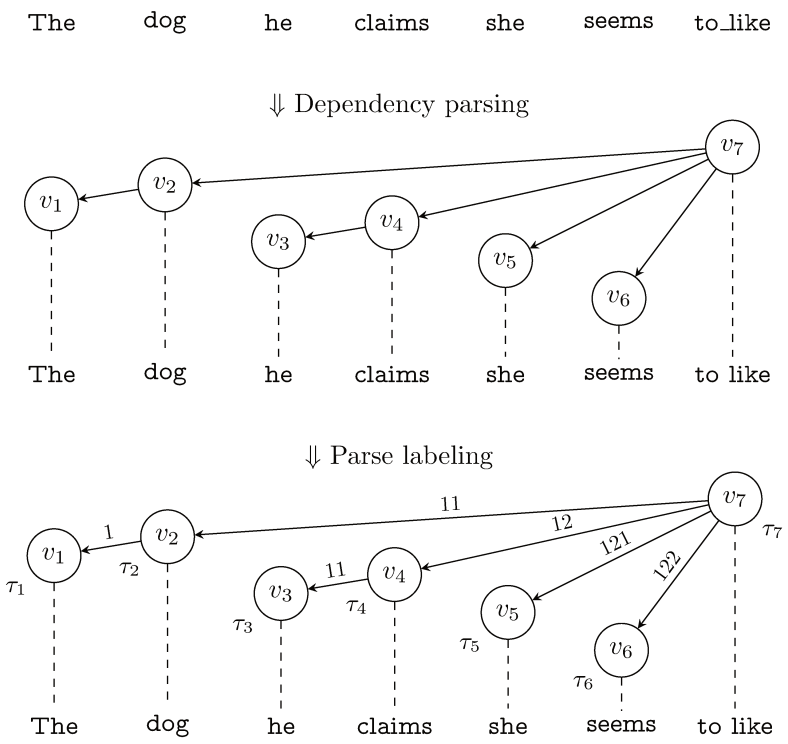


Figure 5.2: The proposed TAG parsing pipeline.

discussed in Chapter 6, does not induce an easy parsing problem. However, one may want to break this problem into two: on one side, heads assignment, on the other elementary trees and operation sites assignment. As such, the first one does not directly take into account the grammar. This is even more important given that lexicalized grammars are highly ambiguous in practice in order to ensure generalization. Thus, we propose the following pipeline:

1. A dependency parser which outputs LTAG compatible unlabeled derivation trees;

2. A parse tree labeler which assigns elementary trees to words and operation sites to head-modifier relations.

See Figure 5.2 for an illustration of the process.

Our approach benefits from several advantages. The first step does not take into account the grammar, obviously meaning that its asymptotic complexity will not incorporate a (potentially big) grammar constant. Moreover, long range dependencies are taken into account from the beginning. Long range dependencies are problematic in the standard pipeline approach as most supertaggers predict elementary trees independently. This can be problematic, as an example, in the presence of long parenthetical elements (see Figure 5.3). We note that a more sophisticated supertagger may prevent this kind of error at the expense of increasing the complexity [Bonfante et al., 2014]. However, we are not

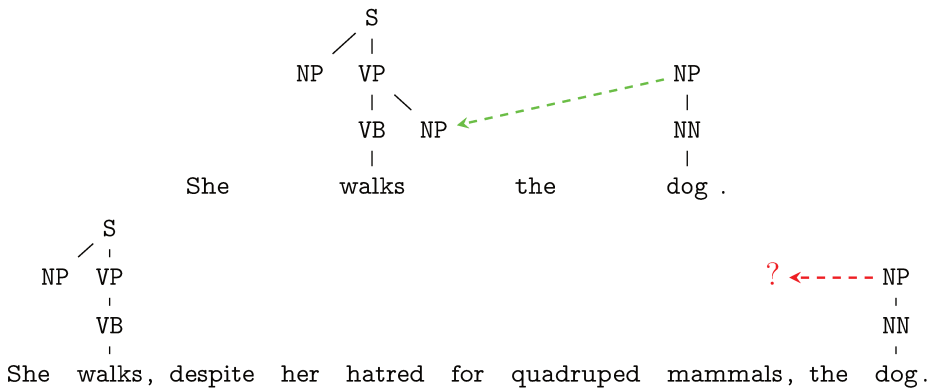


Figure 5.3: In the top example, the pipeline algorithm correctly assigned the transitive verb supertag to `walks`. In the bottom example, due to the long range dependency, it fails: thus, we cannot assign a head to `dog`.

aware of any experimental result based on such an approach. Finally, we rely on subproblems which are actively studied by the NLP community: dependency parsing and tree tagging. As such, any improvement in the parameterization of dependency parsers can be directly incorporated in our work. As we will see in Section 5.6, tree parse labeling is equivalent to constrained CYK parsing. Thus, similarly, any improvement in the parameterization of tree taggers or linear chain taggers can also be incorporated in our work.

5.2 Parsing well-nested arborescences with 2-bounded block degree

A combination operation in a LTAG encodes a head-modifier relationship between two words. Bodirsky et al. [2005] showed that the resulting dependency structure can be characterized using a restricted class of drawings, that is totally ordered forests. In Subsection 5.2.1, we formally introduce these structures but using arborescence properties: well-nested arborescences with 2-bounded block degree (2-BBD). Parsing well-nested arborescences with 2-BBD is a difficult task. Gómez-Rodríguez et al. [2009] proposed a $\mathcal{O}(n^7)$ time complexity algorithm, that is with an asymptotic complexity similar to LTAG parsing. An obvious question at this stage is to ask whether our pipeline approach provides any practical benefit as its first step is as difficult as the original parsing algorithm? We argue that this dependency parsing problem which is not dependent on a specific LTAG has two main advantages. First, even if the asymptotic complexity of both algorithms seems similar, the grammar constant in the LTAG parser complexity may induce prohibitive parsing time even for short sentences. This is a particularly important problem since, in real-world applications, the grammar may be highly ambiguous in order to be robust to noisy inputs (grammatical errors, unknown words, ...). Second, the parsing problem becomes simpler and can be expressed as a graph decoding problem as we do not need

to check many constraints from the grammar.[1]

Our characterization using arborescence properties naturally leads to the development of parsing algorithms by adding constraints on existing ILP formulations for arborescence decoding. In Subsection 5.2.2 we propose an ILP with a polynomial number of constraints. However, the decoding time using a generic solver is unsatisfactory. Moreover, the intricate structure of the program prevents the use of Lagrangian relaxation. Thus, we propose an alternative ILP in Subsection 5.2.3 which has an exponential number of constraints but exposes an appealing structure.

5.2.1 Definitions

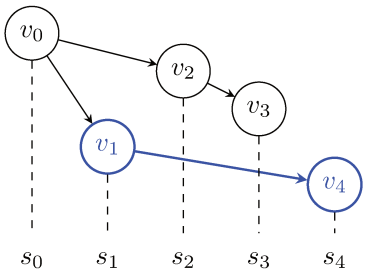
In this subsection, we propose a formal definition of k-BBD and well-nestedness as properties of an arborescence. This is not, of course, fundamentally different from existing definitions but we stress that it simplifies the exposition of the parsing problem as a graph optimization problem. We prove that we can constrain an arborescence to be well-nested and have a k-BBD thanks to constraints on arc sets. Intuitively, this will be of a major interest when solving the problem via Lagrangian relaxation. Indeed, if the relaxed problem is the unconstrained maximum spanning arborescence, then it can be solved efficiently via the Chu–Liu/Edmonds' algorithm (see Example 4.6). However, this is possible only if penalties from relaxed constraints violation are defined on arc weights. The polynomial program proposed in Subection 5.2.2 does not fulfill this requirement contrary to the exponential one in Subection 5.2.3.

Let $G = (V, A)$ be a graph with $V = \{v_0, \dots v_n\}$ the finite set of vertices and $A \subseteq V \times V$ the set of arcs. The block degree and well-nestedness properties suppose a total order on vertices. Without loss of generality, we rely on the common integer order via vertex indices, i.e. $v_0 < v_1 < \cdots < v_{n-1} < v_n$. The **predecessor** (respectively **successor**) of vertex v_k is v_{k-1} (respectively v_{k+1}).

Definition 5.1: Yield of a vertex
Let $T \subseteq A$ be a set of arcs. The **yield** of a vertex $v \in V$ with respect to T corresponds to the set of vertices reachable from v with respect to T.

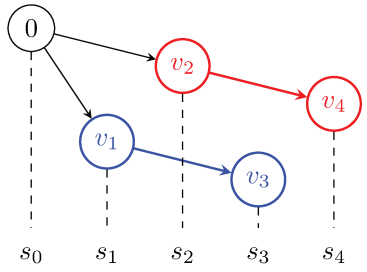
Definition 5.2: Block degree
The **block degree** of a vertex set $W \subseteq V$ is the number of vertices of W without a predecessor inside W. Given an arborescence T, the block degree of a vertex is the block degree of its yield and the block degree of T is the maximum block degree of its incident vertices. An arborescence **satisfies the k-BBD condition** if its block degree is less than or equal to k. This definition is illustrated in Figure 5.4.

[1] The LTAG parsing problem can be formulated as a graph decoding problem using the generic construction of Martin et al. [1990] which reduces a dynamic program to a shortest path problem in an acyclic hypergraph. However, this formulation would need a graph with a $\mathcal{O}(n^7)$ number of vertices.



Vertex	Yield	Block-degree
v_0	$\{\boldsymbol{v_0}, v_1, v_2, v_3, v_4\}$	1
v_1	$\{\boldsymbol{v_1}, \boldsymbol{v_4}\}$	2
v_2	$\{\boldsymbol{v_2}, v_3\}$	1
v_3	$\{\boldsymbol{v_3}\}$	1
v_4	$\{\boldsymbol{v_4}\}$	1

Figure 5.4: Example of a 2-BBD arborescence. The table shows yields and block-degree of each vertex. Vertices in bold do not have their predecessor in the set. The vertex with maximum block-degree is v_1, meaning the arborescence has a block-degree equal to two. It has a 2-BBD, 3-BBD, ...



Vertex	Yield
v_0	$\{v_0, v_1, v_2, v_3, v_4\}$
v_1	$\{v_1, v_3\}$
v_2	$\{v_2, v_4\}$
v_3	$\{v_3\}$
v_4	$\{v_4\}$

Figure 5.5: Example of an ill-nested arborescence: yields of v_1 and v_2 are disjoints, i.e. $\{v_1, v_3\} \cap \{v_2, v_4\} = \emptyset$, and $v_1 < v_2 < v_3 < v_4$ with v_1, v_2 in the yield of v_1 and v_2, v_4 in the yield of v_2.

Definition 5.3: Well-nestedness
Two disjoint subsets $I_1, I_2 \subseteq V$ **interleave** if there exist $i, j \in I_1$ and $k, l \in I_2$ such that $i < k < j < l$. An arborescence is **well-nested** if it is not incident to two vertices whose yields interleave. Otherwise, it is said to be ill-nested, see Figure 5.5.

Let $T \subseteq A$ be a subset of arcs inducing a v_0-rooted spanning arborescence. In the following, without loss of generality, we define arborescence properties on arborescence T. From Definition 5.2 and Definition 5.3, we can derive necessary and sufficient conditions so that T is a well-nested arborescence with 2-BBD:

- For each vertex $v_k \in V$, the block degree of its yield with respect to arborescence T is at most 2;
- For each couple of vertices $v_k, v_l \in V$ such that $v_k \neq v_l$, their yields with respect to T are either not disjoint or do not interleave.

The 2-BBD condition can be constrained by collecting the yield of each vertex $v_k \in V$, that is checking the existence of a path from v_k to other vertices via

arcs in T, and forcing them to have at most two vertices without predecessor. Similarly, the well-nestedness condition can be checked using paths. This is exactly how the polynomial program in Section 5.2.2 will be derived.

Note that arcs are not directly used in the definition of block degree and well-nestedness. However, these conditions require to compute, among other, the yield of each vertex. This is something that may be inefficient in optimization programs because it introduces additional variables and intricate constraints. Alternatively, we propose to define properties of T via subsets of arcs in T only.

Definition 5.4: Family of bounded block degree vertex subsets
Given a vertex set V and an integer $k \geq 0$, we denote $\mathcal{V}^{\geq k}$ the family of vertex subsets of V with block degree greater or equal to k.

Theorem 5.5: k-bounded block degree arborescence
An arborescence T is not k-BBD if and only if there exists a vertex subset $W \in \mathcal{V}^{\geq k+1}$ whose cutset $\delta(W)$ contains a single arc in T, that is $|\delta(W) \cap T| = 1$.

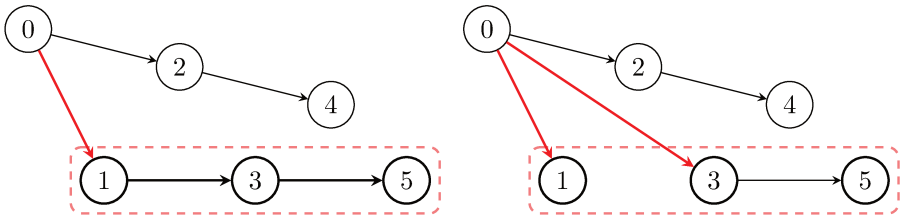
Proof. By definition of block degree, a dependency tree is not k-BBD if and only if it is incident with a vertex whose yield W belongs to $\mathcal{V}^{\geq k+1}$. In other word, T is not k-BBD if it contains a subarborescence T' such that $V[T']$ equals $W \in \mathcal{V}^{\geq k+1}$. This holds if and only if W has exactly one entering arc (since $v_0 \notin W$) and no leaving arc belonging to T. □

Definition 5.6: Family of ill-nested vertex subsets
Given a vertex set V, we denote $\mathcal{V}^{\text{ill}}$ the family of couples of disjoint interleaving vertex subsets of V.

Theorem 5.7: Well-nestedness arborescence
An arborescence T is not well-nested if and only if there exists a couple $(I_1, I_2) \in \mathcal{V}^{\text{ill}}$ such that $\delta(I_1) \cap T$ and $\delta(I_2) \cap T$ are singletons.

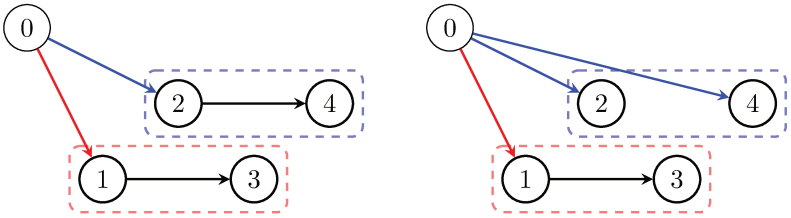
Proof. $\delta(I_1)$ and $\delta(I_2)$ both intersect T only once if and only if T contains two subarborescences T_1 and T_2 such that $V[T_1] = I_1$ and $V[T_2] = I_2$. This means that T is incident with two vertices whose yields are I_1 and I_2, respectively. Result follows from the definition of $\mathcal{V}^{\text{ill}}$ and well-nested arborescences. □

By Theorem 5.5, we can check if an arborescence satisfies the 2-BBD property via conditions on the cutset of subsets of vertices only, see Figure 5.6. Obviously, the size of $\mathcal{V}^{\geq 3}$ grows faster than V. Similarly, the well-nested property can be checked via conditions on cuts of subset couples of vertices only, see Figure 5.7. We derive from these theorems an ILP with an exponential number of constraints in Section 5.2.3.



(a) Not 2-BBD arborescence (b) 2-BBD arborescence

Figure 5.6: If there is only one adjacent arc to the arc set $\{v_1, v_3, v_5\}$, which is of block degree 3, then the arborescence is not 2-BBD.



(a) Ill-nested arborescence (b) Well-nested arborescence

Figure 5.7: If there is only one adjacent arc to subset $\{v_1, v_3\}$ and one adjacent arc to subset $\{v_2, v_4\}$, then the arborescence is ill-nested.

5.2.2 A polynomial ILP formulation

We now derive a compact ILP formulation, that is a program with a polynomial number of variables and constraints, for computing the maximum well-nested and 2-BBD arborescence. First, we explain the multi-flow commodity formulation introduced by Martins et al. [2009a] for computing the Maximum Spanning Arborescence. Contrary to the ILP formulation of Schrijver [2003], which was based on arc-variables only (see Example 4.6), it introduces additional variables but it is compact. Moreover, the flow variables can be used to compute the yield of vertices. Then, we propose constraints in order to ensure well-nestedness and bounded block degree. Unfortunately, they will need the introduction of new variables in the program. Finally, we study the complexity of this ILP and its downsides.

Multi-flow commodity formulation of the MSA

Let $G = (V, A)$ be a directed graph with $n+1$ vertices, $V = \{v_0 \dots v_n\}$, and $\mathcal{T}$ the set of all v_0-rooted spanning arborescences on G. A spanning arborescence $T \in \mathcal{T}$ is defined as a subset of arcs $T \subseteq A$. We seek to compute the v_0-rooted maximum spanning arborescence. We assume that the weighting model is arc-factored: the weight of a spanning arborescence T can be written as $f(T \in \mathcal{T}) = \sum_{a \in T} w_a$ with w_a the weight of arc $a \in A$. In the multi-flow commodity formulation of the MSA, the root v_0 sends one commodity to each vertex. A vertex must consume its own commodity exactly once. Commodities flow through selected arcs and with respect to their direction only. Moreover, only n arcs can be selected, meaning that the induced structure is connected. Thus it correctly models the set of v_0-rooted arborescences $\mathcal{T}$.

Let $x \in \mathbb{B}^A$ be a vector of boolean variables indexed by the set of arcs A, with $x_a = 1$ meaning that arc a is selected. Let $y \in \mathbb{R}^{V^+ \times A}$ be the vector of flow variables indexed by vertices and arcs: $y_{v,a}$ is the quantity of commodities which must be delivered to vertex $v \in V^+$ that goes through arc $a \in A$. Given a subset of arcs $B \subseteq A$, $x(B)$ corresponds to $\sum_{a \in B} x_a$. We abuse this notation and define $y_v(B) \triangleq \sum_{a \in B} y_{v,a}$ for flow variables. The MSA problem can be formulated thanks to the following ILP:

$$\max_x \quad \sum_{a \in A} x_a w_a \tag{5.1}$$

$$\text{s.t.} \quad y_v(\delta^+(0)) = 1 \qquad \forall v \in V^+ \tag{5.2}$$

$$y_u(\delta^-(v)) - y_u(\delta^+(v)) = \mathbb{1}[v = u] \qquad \forall v, u \in V^+ \tag{5.3}$$

$$y_{v,a} \leq x_a \qquad \forall a \in A, v \in V^+ \tag{5.4}$$

$$x(A) = n \tag{5.5}$$

$$x \in \mathbb{B}^A \tag{5.6}$$

$$y \in \mathbb{R}^{V^+ \times A} \tag{5.7}$$

with $V^+ = V \setminus \{v_0\}$ and $\mathbb{1}$ the Kronecker delta, that is $\mathbb{1}[v = u]$ is equal to 1 if $v = u$ and 0 otherwise. Constraints (5.2) ensure that the root vertex sends one unit of commodity to each vertex. Each vertex consumes exactly one unity of its commodity because of constraints (5.3). An arc can only carry flow if it is selected thanks to constraints (5.4) and constraints (5.5) ensure that there are exactly n selected arcs. This formulation has the integrality property, meaning that relaxing the integrality constraint on arc variables (5.6) to $x \in [0, 1]^A$ leaves the optimal solution unchanged [Martins, 2012].

Bounded block degree

The yield of vertex $v_i \in V$ is denoted $\blacktriangle(v_i)$. The block degree of vertex v_i is equal to the number of vertices in $\blacktriangle(v_i)$ which do not have their predecessor in $\blacktriangle(v_i)$. Let $b \in \mathbb{B}^{V^+} \times \mathbb{B}^{V^+}$ be a matrix of boolean variables. We constrain b so that $b_{v_i,v_j} = 1$ if and only if vertex $v_j \in V$ is in the yield of $v_i \in V$ and predecessor v_{j-1} is not. In other words, $b_{v_i,v_j} = 1$ if and only if v_j is the beginning of a block in the yield of v_i. To achieve this, we basically need to linearize the boolean operation $p = \neg q \wedge r$ [Boros and Hammer, 2002]:

$$p \leq 1 - q \tag{5.8}$$

$$p \leq r \tag{5.9}$$

$$p \geq (1 - q) + r - 1 \tag{5.10}$$

In the following, we use this boolean constraint as a linear constraint. If v_j is in the yield of v_i, then the commodity at destination of v_j must flow through the incoming arc of v_i. Thus, we build the following constraints. The vertex v_1 is always the beginning of a block in its own yield:

$$b_{v_1,v_1} = 1 \tag{5.11}$$

Obviously, the root vertex v_0 cannot be in the yield of any other vertex. Thus, checking if v_1 is the beginning of a block in the yield of u is simply:

$$b_{u,v_1} = y_u(\delta^{in}(v_1)) \qquad \forall u \in V^+ \setminus \{v_1\} \tag{5.12}$$

Because a vertex is always in its own yield, checking if it is a beginning of a block is equivalent to checking if its previous vertex is not in the yield. Let *pred* and *succ* be the predecessor and successor predicates, respectively, that is $pred(v_i) = v_{i-1}$ and $succ(v_i) = v_{i+1}$. Then, we can define the following constraints:

$$b_{u,u} = 1 - y_{pred(u)}(\delta^{in}(u)) \qquad \forall u \in V^+ \setminus \{v_1\} \tag{5.13}$$

Finally, in other cases, we use the previously defined boolean equation:

$$b_{u,v} = \neg y_{pred(v)}(\delta^{in}(u)) \wedge y_v(\delta^{in}(u)) \qquad \forall u \in V^+, v \in V^+ \setminus \{v_1, u, succ(u)\} \tag{5.14}$$

Thus, thanks to constraints (5.11)-(5.14), $b_{u,v}$ is equal to one if and only if vertex v is the beginning of a block in the yield of vertex u. The block degree of an arborescence is the maximum block degree of its incident vertices. In order to ensure that an arborescence has a k-BBD, we must ensure that each of its incident vertices has a block degree less or equal to k. To this end, we sum the beginning of block indicators:

$$\sum_{v \in V^+} b_{u,v} \leq k \qquad \forall u \in V^+ \tag{5.15}$$

Well-nestedness

We now propose constraints to enforce well-nestedness. From Definition 5.3, an arborescence is ill-nested if there exist two vertices such that:

- their yields W^1 and W^2 are disjoint, i.e. $W^1 \cap W^2 = \emptyset$;
- and there exist $u, u' \in W^1$ and $v, v' \in W^1$ such that $u < v < u' < v'$.

However, it is difficult to directly enforce well-nestedness using this condition. Havelka [2007] introduced the notion of ill-nested set of an arc: an arborescence is well-nested if and only if the ill-nested set of each one of its arcs is empty.

Definition 5.8: Ill-nested set
Let $G = (V, A)$ be a graph and $T \subseteq A$ be a subset of arcs inducing an arborescence. Given two vertices $v_i, v_j \in V[T]$ such that $v_i < v_j$ and $v_i \to v_j \in T$ (respectively $v_j \to v_i \in T$), the **ill-nested set** of arc $v_i \to v_j \in T$ (respectively $v_j \to v_i \in T$) is the set:

$$\begin{aligned} \{v_k \to v_l \in T \quad | \quad & \exists (I^1, I^2) \in \mathcal{V}^{\text{ill}} \text{ such that} \\ & v_i, v_j \in I^1 \wedge v_k, v_l \in I^2 \wedge \\ & v_i \to^* v_l \notin T \wedge v_j \to^* v_l \notin T \wedge \\ & v_k \to^* v_i \notin T \wedge v_l \to^* v_i \notin T\} \end{aligned}$$

where $v_i \rightarrow^* v_j \in T$ denotes the existence of a path from vertex v_i to vertex v_j using arcs of T only.

Theorem 5.9: Well-nestness and ill-nested sets
Let $G = (V, A)$ be a graph and $T \subseteq A$ an arborescence. T is well-nested if and only if the ill-nested set of each arc in the arborescence $a \in T$ is empty.

Proof. Intuitively, the theorem states that any ill-nested couple of vertex set must not define the yields of two distinct sub-arborescences in the arborescence T. See Havelka [2007] for the complete proof. □

In order to ensure well-nestedness, if an arc is selected in the arborescence then its ill-nested set must be empty. If we take into account each arc in the set independendly, this condition can be written as:

$$
\begin{aligned}
& v_i \rightarrow v_j \in T \\
& \Rightarrow \neg \\
& (\\
& \quad v_k \rightarrow v_l \in T \\
& \quad \wedge v_i \rightarrow^* v_l \notin T \\
& \quad \wedge v_j \rightarrow^* v_l \notin T \\
& \quad \wedge v_k \rightarrow^* v_i \notin T \\
& \quad \wedge v_l \rightarrow^* v_i \notin T \\
&)
\end{aligned}
$$

for each quadruple $v_i, v_j, v_k, v_l \in V$ such that there exists a couple $(I^1, I^2) \in \mathcal{V}^{\text{ill}}$ with $v_i, v_j \in I^1$ and $v_k, v_l \in I^2$. Using the boolean equivalence $a \Rightarrow b = \neg a \vee b$, we have:

$$
\begin{aligned}
= \quad & \neg v_i \rightarrow v_j \in T \\
& \vee \neg \\
& (\\
& \quad v_k \rightarrow v_l \in T \\
& \quad \wedge v_i \rightarrow^* v_l \notin T \\
& \quad \wedge v_j \rightarrow^* v_l \notin T \\
& \quad \wedge v_k \rightarrow^* v_i \notin T \\
& \quad \wedge v_l \rightarrow^* v_i \notin T \\
&)
\end{aligned}
$$

Because $\neg(a \wedge b) = \neg a \vee \neg b$, we can reformulate the condition:

$$
\begin{aligned}
= \quad & \neg v_i \rightarrow v_j \in T \vee \neg v_k \rightarrow v_l \in T \\
& \vee v_i \rightarrow^* v_l \in T \vee v_j \rightarrow^* v_l \in T \\
& \vee v_k \rightarrow^* v_i \in T \vee v_l \rightarrow^* v_i \in T
\end{aligned}
$$

Thus, the following linear constraint set ensures well-nestedness:

$$
\begin{aligned}
&(1 - x_{v_i \to v_j}) + (1 - x_{v_k \to v_l}) \\
&+ y_{v_l}(\delta^{in}(v_i)) + y_{v_l}(\delta^{in}(v_j)) + y_{v_i}(\delta^{in}(v_k)) + y_{v_i}(\delta^{in}(v_l)) \geq 1 \\
\forall \quad & v_i, v_j, v_k, v_l \in V \text{ such that there exists a couple } (I^1, I^2) \in \mathcal{V}^{\text{ill}} \\
& \text{with } v_i, v_j \in I^1 \text{ and } v_k, v_l \in I^2
\end{aligned} \tag{5.16}
$$

Discussion

The following ILP can be used to compute the well-nested arborescence with k-BBD:

$$
\begin{aligned}
\underset{x}{\text{argmax}} \quad & \sum_{a \in A} x_a w_a & \\
\text{s.t.} \quad & \text{(5.2)-(5.6)} & \text{(arborescence)} \\
& \text{(5.11)-(5.15)} & (k\text{-BBD}) \\
& \text{(5.16)} & \text{(well-nestedness)}
\end{aligned}
$$

This formulation is compact because it has a polynomial number of variables and constraints with respect to the input graph. The number of arc variables x is bounded by $\mathcal{O}(|V|^2)$ and the number of flow variables y by $\mathcal{O}(|V|^3)$. The number of constraints (5.2)-(5.6) is bounded by $\mathcal{O}(|V|^3)$. To bound the block degree of the predicted arborescence, we need to introduce $\mathcal{O}(|V|^2)$ variables b to compute the beginning of block in the yield of each vertex. There are $\mathcal{O}(|V|^2)$ constraints (5.11)-(5.15). The most expensive constraint set is (5.16) ensuring well-nestedness as it is defined on quadruples of vertices, thus its size is bounded by $\mathcal{O}(|V|^4)$.

Although the multi-commodity flow formulation of the MSA problem has the integrality property, the additional constraints no longer guarantee that solving the LP relaxation will give an integral optimal solution anymore. Thus, solving the problem with a generic solver relies on an algorithm with an exponential complexity. In practice, with CPLEX, we observed intractable running times even with small graphs.

As a final note, we discuss the use of Lagrangian relaxation. One wishes to relax bounded block degree and well-nestedness constraints so that the polynomial time algorithm for the MSA problem can be used at each iteration of the subgradient descent procedure that minimizes the Lagrangian dual. However, both constraint sets rely on paths via flow variables. Penalizing flow variables cannot be incorporated in the Chu–Liu/Edmonds' algorithm as it assumes an arc-factored model. Interestingly, we proposed a compact formulation of the problem but which seems difficult to use in order to build a fast parsing method. We observe that if we want to rely on the MSA as a subproblem in the Lagrangian relaxation framework, the relaxed constraints must penalize arcs independently. We propose a formulation which satisfies this requirement in the next section.

5.2.3 An exponential ILP formulation

Again, let $x \in \mathbb{B}^A$ be a vector of boolean variables indexed by the set of arcs A, with $x_a = 1$ meaning that arc a is selected. Our exponential formulation does not rely on flow variables: we can rely on the arborescence polytope introduced in Example 4.6. To simplify the program, we denote $\mathcal{X}$ the set of assignments of x inducing v_0-rooted spanning arborescences only (see Example 4.6). The arc-factored MSA problem is then simply:

$$\max_x \quad \sum_{a \in A} w_a x_a \tag{5.17}$$

$$\text{s.t.} \quad x \in \mathcal{X} \tag{5.18}$$

which can be solved via the Chu–Liu/Edmonds' algorithm.

We ensure that arborescence x is of k-BBD thanks to Theorem 5.5. For all subsets of vertices $W \in \mathcal{V}^{\geq k}$, there must be at least two arcs in the cutset of W:

$$x(\delta(W)) \geq 2 \qquad \forall W \in \mathcal{V}^{\geq k} \tag{5.19}$$

Similarly, from Theorem 5.7, an arborescence is well-nested if and only if for all couples of vertex subsets $(I^1, I^2) \in \mathcal{V}^{\text{ill}}$, there are at least two arcs adjacent to the cutset of I^1 or to the cutset of I^2. Because we already constrained the structure to be an arborescence, both sets have at least one incoming arc. Thus, we introduce the following constraint set:

$$x(\delta(I^1)) + x(\delta(I^2)) \geq 3 \qquad \forall (I^1, I^2) \in \mathcal{V}^{\text{ill}} \tag{5.20}$$

It is easy to see that the size of sets $\mathcal{V}^{\geq k}$ and $\mathcal{V}^{\text{ill}}$ grows rapidly with respect to the size of V. Thus, a program including these constraints is totally inappropriate to be optimized using a generic solver.

Our exponential formulation is:

$$\begin{aligned} \max_x \quad & \sum_{a \in A} w_a x_a & \\ \text{s.t.} \quad & x \in \mathcal{X} & \text{(arborescence)} \\ & (5.19) & (k\text{-BBD}) \\ & (5.20) & \text{(well-nestedness)} \end{aligned}$$

We obviously identify constraint sets (5.19) and (5.20) as the difficult ones: if they were absent, the problem would reduce to the MSA problem which has a quadratic time complexity. Thus, they are ideal relaxation candidates. However, relaxing an exponentially large set of constraints is also a technical challenge. In the next section, we generalize this program and then derive an efficient heuristic based on Lagrangian relaxation. The proposed procedure relies on a black box in order to lazily compute violated constraints.

5.3 Reduction to the Yield Restricted Maximum Spanning Arborescence problem

A k-BBD arborescence is an arborescence where no vertex has a yield with a block degree strictly greater than k. In other words, there is a set of yields which are forbidden. Similarly, well-nestedness is a constraint on pairs of disjoint sub-arborescences or, in other words, pairs of yields. This naturally leads to a more general problem where we want to compute an arborescence with resctrictions on the yields of its sub-arborescences: the Yield Restricted Maximum Spanning Arborescence problem.

5.3.1 Definition

We introduce the Yield Restricted Maximum Spanning Arborescence (YRMSA) problem, which, to the best of our knowledge, has not been studied in the literature yet. Without loss of generality, we assume v_0-rooted arborescences only.

Definition 5.10: Yield Restricted Arborescence
Let $G = (V, A)$ be a graph and $\mathcal{F}$ be a set of forbidden yield sets. Each $F \in \mathcal{F}$ is a set of yields $F = (W^1, \dots, W^{|F|})$ such that:

- Items are subsets of vertices, $W^k \subseteq V^+$;
- Items are mutually disjoint, $W^k \cap W^l = \emptyset$ for all $k \neq l$.

A $\mathcal{F}$ **Yield Restricted Arborescence** is an arborescence T such that there is no set of sub-arborescences in T which have yields $F \in \mathcal{F}$.

Thus, the well-nested MSA with 2-BBD is equivalent to the $\mathcal{F}$ YRMSA where[2] $\mathcal{F} = \mathcal{V}^{\text{ill}} \cup \mathcal{V}^{\geq 3}$.

5.3.2 ILP formulation

We first introduce Theorem 5.11, from which we will derive our ILP formulation. Given a set of forbidden yield sets $\mathcal{F}$, we want to prevent an arborescence from having $|F|$ distinct subarborescences with yields F for every $F \in \mathcal{F}$. If for any $F \in \mathcal{F}$, each yield in F has exactly one incident arc, then this constraint is violated. This theorem is similar to the one we introduce for well-nestedness (Theorem 5.7).

Theorem 5.11: Yield restricted arborescence
An arborescence T is not $\mathcal{F}$-yield restricted if and only if there exists a set of yields $F \in \mathcal{F}$ such that $\delta(F_k) \cap T$ is a singleton for every yield $F_k \in F$.

Proof. Each $F_k \in F$ intersects T only once if and only if T contains $|F|$ arborescences $T_1 \dots T_{|F|}$ such that $V[T_1] = F_1 \wedge \dots \wedge V[T_{|F|}] = F_{|F|}$. This means that T is incident with $|F|$ vertices whose yields are $F_1 \dots F_{|F|}$. □

[2] We treat elements of $\mathcal{V}^{\geq 3}$ as sets of size 1.

Let $G = (V, A)$ be a graph and $\mathcal{F}$ a set of forbidden yields. We introduce a variable vector x indexed by arcs A. Then, the YRMSA problem can be formulated as the following ILP:

$$\begin{aligned}
\underset{x}{\text{argmax}} \quad & x^\top w && && (5.21)\\
\text{s.t.} \quad & x(\delta^{in}(v_0)) = 0 && && (5.22)\\
& x(\delta^{in}(v)) = 1 && \forall v \in V^+ && (5.23)\\
& x(\delta^{in}(W)) \geq 1 && \forall W \subseteq V^+ && (5.24)\\
& \sum_{i=1}^{|F|} x(\delta(F_i)) \geq |F| + 1 && \forall F \in \mathcal{F} && (5.25)\\
& x \in \mathbb{B}^{|A|} && && (5.26)
\end{aligned}$$

As usual, the objective (5.21) maximizes the decoded structure objective under an arc-factored model and constraints (5.22)-(5.24) defines the spanning arborescence polytope. Moreover, constraint set (5.25) ensures that there is no set of sub-arborescences with yields equal to any set $F \in \mathcal{F}$. This formulation does not have the integrality property in the general case, so we need to enforce x to contain boolean values via constraint (5.26). Note that this program is not compact because of the size of constraint set (5.24), regardless of the size of $\mathcal{F}$.[3] However, alternatively, we could have defined the arborescence polytope thanks to the multi-commodity flow formulation which has a polynomial number of variables and constraints (see Section 5.2.2), but is does not address the size of $\mathcal{F}$.

5.3.3 Lagrangian Relaxation

In this section, we propose a Lagrangian relaxation formulation of the program defined by (5.21)-(5.26). It is easy to see that constraint set (5.25) is a set of difficult constraints: if they were absent, then the program would reduce to the MSA problem, which can be computed in quadratic time thanks to the Chu–Liu/Edmonds' algorithm. Thus, we relax them to build the following Lagrangian dual objective:

$$\begin{aligned}
\max_{x} \quad & L(x; \lambda) && (5.27)\\
\text{s.t.} \quad & x \in \mathcal{X} && (5.28)
\end{aligned}$$

with λ the vector of Lagrangian multipliers associated with forbidden yield constraints and $\mathcal{X}$ the arborescence polytope. The dual objective function is:

$$\begin{aligned}
L(x; \lambda) &= w^\top x + \sum_{F \in \mathcal{F}} \lambda_F \left(\sum_{i=1}^{|F|} \delta(F_i) - (|F| + 1) \right)\\
&= \sum_{a \in A} w_a x_a + \sum_{a \in A} \sum_{F \in \mathcal{F}} x_a \lambda_F \mathbb{1}[a \in \delta(F)]\\
&\quad - \sum_{F \in \mathcal{F}} \lambda_F \left(|F| + 1 \right)
\end{aligned}$$

[3] In our application, $\mathcal{F}$ has indeed an exponential size.

First, note that the last line contains constant terms only, so it does not influence the optimal solution. Other terms can be rearranged to build reparameterized weight function:

$$w'_a = w_a + \sum_{F \in \mathcal{F}} \lambda_F \mathbb{1}[a \in \delta(F)]$$

Thus, the Lagrangian dual objective can be efficiently computed by running the Chu–Liu/Edmonds' algorithm on a graph with updated weights. However, the size of $\mathcal{F}$ impacts the computation of reparameterized weights.

The Lagrangian multiplier vector has a very high cardinality. When the graph becomes large, this may involve several downsides regarding memory usage. Moreover, updating them all at each iteration is computationally inefficient. However, when searching for the well-nested maximum spanning arborescence of k-BBD, computing sub-arborescences that violate the bounded block degree and well-nestedness constraints given an arborescence is a simple quadratic time problem [Havelka, 2007], see Appendix C. An elegant option would be to lazily generate the Lagrangian multipliers: at each iteration, we look for violated constraints and add the associated multipliers to the existing ones. This is exactly how we tackle the problem in the following section. The proposed procedure relies on a black box in order to lazily compute violated constraints and can thus be applied to other problems as well.

5.4 Efficient decoding via Non Delayed Relax-and-Cut

In this section, we describe our decoding method for the YRMSA problem based on the Non Delayed Relax-and-Cut algorithm [Lucena, 2005]. This method has been proposed in order to optimize Lagrangian duals with many relaxed constraints: only a subset of Lagrangian multipliers is lazily generated at each iteration by computing violated constraints. This technique is closely related to column and row generation in linear programming.[4] We experimentally validate our method on several datasets and using different constraint sets (see Section 5.5).

5.4.1 Motivations

A common practice to optimize a Lagrangian dual is to rely on the projected subgradient descent algorithm (Section 4.4). It is an iterative algorithm which updates the Lagrangian multipliers by extracting information from the Lagrangian dual objective. Let λ^t be the vector of Lagrangian multipliers at iteration t and let p a projection function to the domain of multipliers. The multipliers are updated as follows:

$$\lambda^{t+1} = p(\lambda^t - s^t g^t) \tag{5.29}$$

[4] We lazily generate relaxed constraints whereas row generation lazily generate hard constraints.

where s^t is the stepsize and g^t is the subgradient at iteration t. Obviously, memory usage is an issue when the vector of multipliers is exponentially large with respect to the input. Moreover, and more importantly, computing the gradient becomes an intractable task. This is all the more surprising given that, in the ILP proposed in the previous section, we showed that the Lagrangian dual objective of our problem can be computed in polynomial time.

In order to tackle this computational challenge, Lucena [2005] proposed an alternative optimization framework based on subgradient descent called **Non Delayed Relax-and-Cut**. Instead of relying on the full vector of multipliers, the latter is lazily constructed, ignoring provably null elements. The element associated with a single relaxed constraint is added only once it has been violated at a given iteration. First, we recall that the domain of multipliers is the non-negative orthant as we relaxed inequalities (5.25) only in order to build our Lagrangian dual (see Definition 4.12). Thus, the projection operator p is the element-wise operation:

$$p(\lambda) = \max(0, \lambda)$$

In other words, negative elements are set to null. If the Lagrangian multipliers are initialized to $\lambda^1 = \mathbf{0}$, then, after the projection, constraints that have never been violated up to iteration t will have a null element in λ^t. This is straighforward to observe in the update rule (5.29): the stepsize is strictly positive and a subgradient element associated with a relaxed constraint $f(x) \geq 0$ is $f(x)$, thus positive or null when the constraint is satisfied (see Theorem 4.30). Second, a common stepsize when optimizating Lagrangian duals [Fisher, 1981] is:

$$s^t = \frac{L^*(\lambda^t) - LB^t}{\|g^t\|^2}$$

where LB^t is the best known lower bound at iteration t. As reported by Beasley [1993], when dealing with many constraints, even if polynomially bounded, the $\|g^t\|^2$ term can result in each Lagrangian update being almost null, meaning the multipliers are nearly stagnant. Because of the projection operator p, elements of λ^t which are null and have positive or null associated subgradient element will have null values in λ^{t+1}. Thus, a good practice is to modify the subgradient so that, for any element i, if $g_i^t > 0$ and $\lambda_i^t = 0$, then we set $g_i^t = 0$. This has the same effect on the multipliers as the projection, but it prevents the stepsize from becoming too small. Hence, instead of generating a full subgradient at each iteration, which is an expensive operation because we would need to consider all multipliers associated with constraints, we process only a subpart, namely the one associated with constraints that have been violated in the current or previous iterations. We refer the reader to [Lucena, 2005, 2006] for an in-depth survey of Lagrangian dual optimization via Non Delayed Relax-and-Cut.

5.4.2 Algorithm

Let CV^t be the set of relaxed constraints violated at iteration t and PV^t the set of constraints which have been violated in at least one previous iteration.[5]

[5] Lucena [2005] proposed to remove constraints from PV^t if they have not been violated in the last i iterations, where i is a predefined constant. We do not follow this advice.

These sets are not necessarily disjoint. At each iteration, we need to compute the set of violated constraints CV^t. In our generic framework for solving the YRMSA problem, this is a parameterizable black-box. In the particular case of the well-nested MSA with k-BBD problem, this is an easy task. Finding pairs of ill-nested sub-arborescences and sub-arborescences with block degree strictly greater than k are both quadratic time problems, see Havelka [2007] and Möhl [2006], respectively. We give pseudo-code for both algorithms in Appendix C. Note that there may be many overlapping constraints inside CV^t, that is arcs may appear in several constraints. This can lead to exaggeration in the weight reparameterization of arcs. Thus, in practice, it may be more efficient to only add a subset of violated constraints inside CV^t. The strategy is left to the user as it can be directly implemented inside the black-box. The subgradient g^t and the updated multipliers λ^{t+1} are only computed for elements associated with constraints $CV^t \cup PV^t$. Then, the set of previously violated constraints is trivially updated as $PV^{t+1} = CV^t \cup PV^t$. The full Non Delayed Relax-and-Cut algorithm is exposed in Algorithm 2.

Algorithm 2 The Non Delayed Relax-and-cut algorithm mimizing our YRMSA Lagrangian dual.

$lb \leftarrow -\infty$ ▷ Initialize lower bound
$ub \leftarrow +\infty$ ▷ Initialize upper bound
$p \leftarrow$ undef ▷ Will store the best found primal solution
$\lambda^1 = 0$ ▷ Initialize Lagrangian multipliers
$PV^0 = \emptyset$ ▷ Initialize the set of previously violated constraints
for $1 \leq t \leq T$ **do**
  $\hat{x} = \max_{x \in X} f(x; \lambda)$ ▷ Dual objective
  $ub = \min(ub, f(\hat{x}; \lambda))$
  $CV^t \leftarrow blackbox(\hat{x})$ ▷ Compute violated constraints
  $PV^t \leftarrow PV^{t-1} \cup CV^t$
  if $|CV^t| = 0$ **then** ▷ Test primal feasibility
    if $\lambda^\top (B\hat{x} - d) = 0$ **then**
      ▷ Only check for the set of relaxed constraints in PV^t
      return $\hat{x}$
    if $f(\hat{x}; \lambda) > lb$ **then** ▷ Update lower bound ?
      $lb = f(\hat{x}; \lambda)$
      $p \leftarrow \hat{x}$
  if $lb = ub$ **then** ▷ If both bound match, then p is optimal
    return p
  $\lambda = p\,(\lambda - \epsilon^t \times (B\hat{x} - d))$
    ▷ Only update for the set of relaxed constraints in PV^t
return p ▷ Return the possibly non-optimal best known primal feasible

5.4.3 Problem Reduction

Problem reduction is an efficient technique in order to prune the search space while maintaining the ability to deliver a certificate of optimality, contrary to beam pruning (see Section 4.5). Moreover, the efficiency of a Branch-and-Bound procedure crucially depends on the number of free variables. Thus, early prun-

ing is an important benefit in order to guarantee efficient decoding. In most approaches in the literature, problem reduction relies on the fact that computing a new solution from the objective dual solution by adding an equality constraint on a single variable is a low time complexity problem. As an example, updating a Lagrangian dual objective solution can be realized in constant time when the relaxed problem is unconstrained [Beasley, 1987], or in linear time if the relaxed problem is the Maximum Spanning Tree problem[6] [Beasley, 1989]. Unfortunately, such an efficient method, relying on a small update of an optimal solution, is not known for the MSA problem. We propose two different procedures in order to guarantee that a variable must, or must not, be in the optimal solution. That is, we are able to fix some arc variables to 1 or 0. The first one requires to re-compute the MSA on an updated graph. The second one is based on a different relaxation of the spanning arborescence problem.

Fixing Variables to 1

Since a vertex in $V^+ = V \setminus \{v_0\}$ must have exactly one parent, fixing $x_a = 1$ for an arc $a = v_i \to v_j$ greatly reduces the problem size, as it will also fix $x_{v_h \to v_j} = 0$, for all $v_h \to v_j \in A$ such that $h \neq i$. Among all arc variables that can be set to 1, *promising* candidates are either the arcs in a solution of the unconstrained MSA or the arcs obtained in the best dual solution after the subgradient descent. There are exactly $|V^+|$ such arcs in each set of candidates, so we test fixation for at most $2|V^+|$ variables. In this case, we are ready to pay the price of a quadratic computation for each of these arcs. Hence, for each candidate arc we obtain an upper bound by seeking the (unconstrained) MSA on the graph where this arc is removed. If this upper bound is lower than the score of the best solution found so far, we can safely say that this arc is in the optimal solution. Consequently, this procedure has a cubic time complexity.

Fixing Variables to 0

We could apply the same strategy for fixing variables to 0. However, this reduction is less rewarding and there are many more variables set to 0 than 1 in a MSA solution. Instead, we rely on a relaxation of the maximum spanning arborescence problem. We recall that the latter can be formulated thanks to the following ILP:

$$\max_x \quad x^\top w \tag{5.30}$$

$$\text{s.t.} \quad x(\delta^{in}(v_0)) = 0 \tag{5.31}$$

$$x(\delta^{in}(v)) = 1 \qquad \forall v \in V^+ \tag{5.32}$$

$$x(\delta^{in}(W)) \geq 1 \qquad \forall W \subseteq V^+ \tag{5.33}$$

$$x \in \mathbb{B}^c \tag{5.34}$$

We build a relaxation by removing the acyclic constraints 5.33. Then, changing an optimal solution by adding a constraint forcing a single arc $v_i \to v_j \in A$ variable to 1 is a constant time problem with the appropriate data structure: remove the single incoming arc of vertex v_j and add arc $v_i \to v_j$. The score of the updated solution of this new relaxation provides an upper bound on a

[6] This problem is equivalent to the MSA with symetric arc scores.

solution containing arc a. If this upper bound is lower than the score of the best solution found so far, we can fix the variable x_a to 0. This procedure has a quadratic complexity.

5.5 Experimental results

We introduced a novel problem called the Yield Restricted Maximum Spanning Arborescence (YRMSA). This was motivated by the LTAG parsing strategy we proposed in Section 5.1: our algorithm relies on computing the well-nested maximum arborescence with 2-BBD. The only algorithm previously proposed in the literature has an impractical $\mathcal{O}(n^7)$ complexity with n the length of the input sentence [Gómez-Rodríguez et al., 2009]. Instead of developing a specific optimization technique for this problem, we proposed a more general framework. We evaluate our method on several datasets with several constraint families. Interestingly, we are able to efficiently parse k-BBD structures without enforcing well-nestedness, which is a NP-complete problem [Satta, 1992].

5.5.1 Datasets

Several authors argued that the standard classification of dependencies as either projective or non-projective (i.e. unrestricted) might not be precise enough to describe the annotated structures observed in several datasets [Pitler et al., 2012, Satta and Kuhlmann, 2014, Kuhlmann and Nivre, 2006]. They proposed finer-grained descriptions of dependency structures in terms of well-nestedness, bounded block degree and edge degree, among others. We restrict ourselves to the first two properties as they are defined on sub-arborescence yields. We used the following corpora to test our Lagrangian framework:

English: We extracted dependencies from the Wall-Street Journal part of the Penn Treebank (PTB) with additional NP bracketings [Vadas and Curran, 2007] using the LTH converter.[7] In line with common practice, sections 02-21 were used for training, 22 for development and 23 for testing. Part of speech tags were predicted by the Stanford tagger[8] trained with 10-jackkniffing.[9]

German: We used dependencies from the SPMRL dataset [Seddah et al., 2014], with predicted part of speech tags and the official split. Following common practice, we removed sentences of length greater than 100 in the test set.

Dutch, Spanish and Portuguese: We used the Universal Dependency Treebank 1.2 [Van der Beek et al., 2002, McDonald et al., 2013, Afonso et al., 2002] with gold part of speech tags and the official split. Again, we removed sentences of length greater than 100 in the test sets.

The structure distributions according to well-nestedness and block degree are shown in Table 5.1. As expected, we observe that different datasets contain different data distributions with respect to these properties. Gómez-Rodríguez

[7] `http://nlp.cs.lth.se/software/treebank_converter/`
[8] `http://nlp.stanford.edu/software/tagger.shtml`
[9] Prediction precision: 97.40%

	English		**German**		**Dutch**	
	WN	IL	WN	IL	WN	IL
Block degree 1	**92.26**	-	**67.60**	-	**69.13**	-
Block degree 2	**7.58**	0.12	**27.12**	**0.79**	**28.50**	0.08
Block degree 3	0.12	0.01	**3.86**	**0.30**	**2.24**	0.01
Block degree 4	-	-	0.19	<0.01	0.04	-
Block degree > 4	-	-	0.11	<0.01	-	-

	Spanish		**Portuguese**	
	WN	IL	WN	IL
Block degree 1	**93.95**	-	**81.56**	0.05
Block degree 2	**5.99**	0.04	**13.92**	0.02
Block degree 3	0.02	-	**3.76**	-
Block degree 4	-	-	0.54	-
Block degree > 4	-	-	0.14	-

Table 5.1: Distribution of dependency tree characteristics in datasets. Values in bold are the smallest subsets to contain 99% of the sentences in each language.

et al. [2011] introduced several dynamic programming algorithms to ensure k-BBD and well-nestedness. Because decoding ill-nested structures is known to be NP-complete, they proposed an algorithm only for a strict subclass. However, computational costs make their algorithms unattractive for practical use. Moreover, each different combination on structure restrictions involves the development of a different chart-based algorithm. Fortunately, our method allows us to easily change the bounded degree constraint or toggle the well-nestedness one. Thus, for each language, we decided to use the tighter combination of bounded block degree and well-nestedness constraints which covers at least 99% of the data. Therefore, we chose to enforce 2-BBD and well-nestedness for English and Spanish, 3-BBD and well-nestedness for Dutch and Portuguese and 3-BBD only for German, i.e. a decoded arborescence can be ill-nested.

5.5.2 Decoding

In this subsection, we discuss issues related to the decoding process. We tested three decoding strategies. First, the MSA that computes the best unconstrained arborescence. Second, a Lagrangian heuristic based method (LR): we run the subgradient descent algorithm to optimize the Lagrangian dual problem. Then, we return the best found primal valid solution, even if we do not get an optimality certificate. Third and finally, an exact and optimal method based on Branch-and-bound (B&B) where each node of the search tree computes its upper-bound via subgradient descent (see Section 4.6). In the latter two strategies, the subgradient descent is stopped after a fixed maximum number of iterations. We chose 100 for English and 200 for other languages after tuning on the development set.

Our attempt to run the dynamic programming algorithm of Gómez-Rodríguez et al. [2009] was unsuccessful. Even with heavy pruning we were not able to

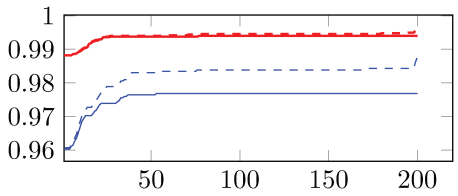


Figure 5.8: Optimality rate (y-axis) vs number of subgradient iterations (x-axis) for English (thin blue) and German (thick red). Solid lines denote the optimal rate with certificate, dashed is without.

	English (96 sentences)			German (59 sentences)		
	MSA	LR	B&B	MSA	LR	B&B
Mean	0.02	0.26	0.53	0.04	0.51	0.71
Std.	0.01	0.20	0.86	0.02	0.41	1.39
Med.	0.02	0.21	0.27	0.03	0.47	0.47
3rd	0.03	0.34	0.53	0.05	0.71	0.80
Total	1.81	25.09	50.52	2.18	30.19	42.20

Figure 5.9: Timings for different parsing algorithms on the test for solutions which do not satisfy constraints after running the MSA. We give (in seconds) average time, standard deviation, median time, time to parse up to the 3rd quartile and total time.

run it on sentences longer than 20 words.[10] We also tried to use CPLEX on the compact ILP formulation we proposed in Section 5.2.2. The decoding time was also prohibitive, approximately one hour for the English dataset. Thus, we did not explore these two decoding methods in our experiments. Obviously, both would have returned similar results to B&B.

We discuss the efficiency of our methods on data for English and German. Other languages give similar results. Optimality rates after the subgradient descent are reported in Figure 5.8. We see that Lagrangian Relaxation often returns optimal solutions but fails to give a certificate of their optimality. Table 5.9 shows parsing times for sentences which require more than one iteration of the subgradient descent. We see that LR and B&B, while slower than MSA, are fast in the majority of cases, below the third quartile. Inevitably, there are some rare cases where a large portion of the search space is explored, and thus their parsing time is high. Let us remark that these algorithms are run only when MSA returns an invalid structure, and so the total time is very acceptable compared to the baseline.

Finally, we stress the importance of the problem reduction techniques proposed in Section 5.4.3 as a pre-processing step for the B&B decoder: after subgradient descent is performed, problem reduction removes an average of 83.97% (respectively 76.59%) of arc variables in the English test set (respectively German test set).

[10] Original authors also reported a similar observation (private communication).

5.5.3 Training

For this experiment, we use linear classifier for arc weights based on hand-crafted features. Feature weights are trained using the averaged structured perceptron [Collins, 2002] with 10 iterations. The best iteration is selected on the development set. We used the same feature set as in TurboParser [Martins et al., 2010], including features for lemma and part of speech tags. For German, we additionally use morpho-syntactic features.

Training with the structured perceptron requires to decode the optimal structure for each instance at each iteration. Unfortunately, relying on the branch-and-bound algorithm results in excessively large training time. Alternatively, one can relies on approximate decoding. However, this must be undertaken with care. Previous work argued that a good practice is to rely on a decoding algorithm which computes the optimal solution in a strictly larger search space, or in other word, which computes the solution of a relaxed problem [Finley and Joachims, 2008, Komodakis, 2011]. Thus, we used the unconstrained MSA problem for training. We also experimented using the Lagrangian relaxation procedure. It did not significantly improve accuracy and made training and decoding slower, similarly to the observation of Martins et al. [2009b].

5.5.4 Parsing Results

Table 5.2 shows attachment score (UAS), percentage of valid dependency trees and relative time to MSA for different systems for our three decoding strategies. We can see that all three decoding methods are almost equivalent in terms of parsing accuracy (UAS). However, we are interested in computing valid structures. The MSA fails to retrieve arborescences which expose the desired properties. Even if for German the gap is small, this means that the decoded structures are useless for a subsequent parse tree labeling task via formal grammars like LTAGs.

In order to see how much well-nested and bounded block-degree structures are missed by a state-of-the-art parser, we compare our results with TurboParser.[11] We run the parser with three different feature sets: arc-factored, standard (second-order features), and full (third-order features). The results are shown in Table 5.3. Our model, by enforcing strict compliance to structural rules (100% valid dependency trees), is closer to the empirical distribution than TurboParser with arc-factored model on all languages but German. Higher-order scoring functions manage to get more similar to the treebank data than our strict thresholds for all languages but Portuguese, at the expense of a significative computational burden.

We interpret this fact as an indication that adding higher order features into our system, or that relying on state-of-the-art neural network parameterization, would make the relaxation method converge more often and faster. Thus, the pipeline we proposed in Chapter 5 for LTAG parsing via derivation tree parsing may be of interest in order to ensure fast parsing. Unfortunately, we were not able to conduct this experiment due to the lack of an annotated dataset.

[11] We used 2.1.0 and all defaults but the feature set.

		MSA	LR	B&B
English	UAS	89.45	**89.54**	89.53
	2-BBD/WN	96.02	–	–
	Relative Time	1	1.8	2.5
German	UAS	**87.79**	87.78	87.78
	3-BBD	98.81	–	–
	Relative Time	1	1.5	1.7
Dutch	UAS	77.30	76.96	**77.40**
	3-BBD/WN	94.82	–	–
	Relative Time	1	1.7	5
Spanish	UAS	83.37	83.37	**83.44**
	2-BBD/WN	92.62	–	–
	Relative Time	1	2.7	3
Portuguese	UAS	83.13	82.99	**83.21**
	3-BBD/WN	87.84	–	–
	Relative Time	1	5.7	19.7

Table 5.2: UAS, percentage of valid structure and decoding time for test data. Time is relative to MSA decoding. The percentage of valid structure is always 100% except for MSA decoding.

Corpus →	**English (99.84)**			**German (99.27)**			**Dutch (99.87)**		
Order ↓	UAS	VDT	RT	UAS	VDT	RT	UAS	VDT	RT
1st	89.29	94.87	1	87.97	98.74	1	76.10	93.26	1
2nd	92.04	99.75	16	89.83	99.28	16	79.05	97.93	18
3rd	92.37	99.75	34	90.35	99.24	36	79.68	97.41	37

Corpus →	**Spanish (99.94)**			**Portuguese (99.24)**		
Order ↓	UAS	VDT	RT	UAS	VDT	RT
1st	83.11	93.43	1	83.53	94.79	1
2nd	86.61	98.54	10	87.35	98.96	15
3rd	87.31	99.64	18	88.09	98.98	32

Table 5.3: UAS, percentage of valid dependency trees (VDT) and relative time (RT) obtained by Turboparser for different score functions on test sets. For each language we give the percentage of valid dependency structures in the data, according to the constraints we postulated.

5.6 Parse tree labeling

We proposed novel formulations of well-nested dependency parsing with 2-BBD as optimization problems. We now turn to the parse tree labeling problem: given a sentence and a LTAG-compatible dependency tree, how can we assign elementary trees and attachment-sites? We introduce a novel algorithm for this task inspired by the CYK LTAG parser and prove that it has a linear time complexity.

5.6.1 Notation

We introduce notations which are used to simplify the algorithm exposition. The input of the algorithm is a sentence $s = s_1 \cdots s_n$ and a graph $G = (V, A)$ with $V = \{v_1 \cdots v_n\}$ the set of vertices, where v_k is the representation of word s_k, and $A \subset V \times V$ the set of arcs describing a spanning arborescence with root v_r. We make the following assumptions on the arborescence described by A:

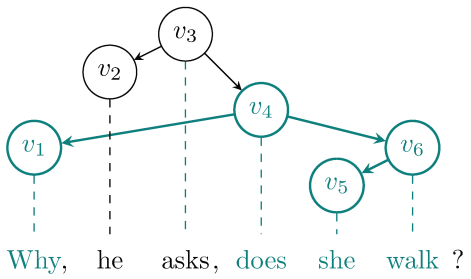
- It is well-nested.
- it has a 2-BBD.

We emphasize that these properties are mandatory: the following parse labeling algorithm is undefined on graphs exposing a different structure. The word s_r is the root word of the sentence, the only one which is assigned an initial tree but which is not attached via substitution to another tree.

Given a vertex $v_i \in V \setminus \{v_r\}$ and $\delta^{in}(v_i) = \{v_j \to v_i\}$, its **parent** is vertex $v_j = \mathit{parent}(v_i)$. The **predecessor** (respectively **successor**) of a vertex $v_i \in V$ is vertex v_{i-1} (respectively v_{i+1}), which is independent of the graph structure. The vertex with the smallest (respectively largest) index in a set is the **leftmost** (respectively **rightmost**) vertex. The **yield** of a vertex $v_k \in V$, written $\blacktriangle(v_k)$, is the set of vertices reachable from v_k with respect to A, including itself. Moreover, we denote $\blacktriangle^{\text{left}}(v_k)$ (respectively $\blacktriangle^{\text{right}}(v_k)$) the leftmost (respectively rightmost) vertex in the yield of vertex v_k. Because the input is constrained to 2-BBD arborescences only, a vertex has at most one gap. The set of vertices in the gap of v_k is $\triangle(v_k)$. If and only if v_k is of block degree 1, then $\triangle(v_k) = \emptyset$. We define $\triangle^{\text{left}}(v_k)$ (respectively $\triangle^{\text{right}}(v_k)$) the leftmost (respectively rightmost) vertex in the gap of node v_k. $\triangle^{\text{left}}(v_k)$ and $\triangle^{\text{right}}(v_k)$ are undefined for vertices with a block degree of 1. The definitions are illustrated in Figure 5.10

Recall that a LTAG is a tuple $\langle N, T, \Gamma^I, \Gamma^A, S, f_{SS}, f_{SA}, f_{OA} \rangle$ with:

- N is a set of non-terminal symbols;
- T is a set of terminal symbols disjoint from N;
- Γ^I is a set of initial trees built with symbols in $N \cup T$;
- Γ^A is a set of auxiliary trees built with symbols in $N \cup T$;
- $\mathtt{S} \in N$ is the start symbol;
- f_{SS} is the function that represents substitution constraints;



Notation	Value
$parent(v_4)$	v_3
$\blacktriangle(v_4)$	$\{v_1, v_4, v_5, v_6\}$
$\blacktriangle^{\text{left}}(v_4)$	v_1
$\blacktriangle^{\text{right}}(v_4)$	v_6
$\triangle(v_4)$	$\{v_2, v_3\}$
$\triangle^{\text{left}}(v_4)$	v_2
$\triangle^{\text{right}}(v_4)$	v_3

Figure 5.10: Example of a 2-BBD and well-nested dependency structure. The right table exposes notations we use for information we can extract about vertex v_4.

- f_{SA} and f_{OA} are functions that represent adjunction constraints.

Elementary trees can be combined via substitution and adjunction operations, see Section 3. An unlabeled parse tree is represented by an arborescence where an arc $v_h \rightarrow v_m$ represents a combination of the elementary tree anchored at word s_m into the elementary tree anchored at word s_h.

5.6.2 Item definition

Given a LTAG $\langle N, T, \Gamma^I, \Gamma^A, \Gamma^S, f_{OA}, f_{SA} \rangle$, a sentence $s = s_1 \dots s_n$ and the corresponding dependency structure $G = (V, A)$, items are 6-tuples of the form $[v_h, \tau, p, c, b_l, b_r]$ with:

1. **considered vertex** $v_h \in V$ in the dependency structure, corresponding to word s_h;
2. **elementary tree** $\tau \in \Gamma$, indicating the association of the anchor of τ with word s_h;
3. **Gorn address** $p \in \tau$ of a node in the elementary tree;
4. **combination flag** $c \in \{\bot, \top\}$ specifying if a combination operation has already been investigated $\top$ or not $\bot$ at node p;
5. **left boundary** $b_l \in V \cup \{l_h, g_h, -\}$ defines the left boundary of the yield of the item, which is discussed in more details below;
6. **right boundary** $b_r \in V \cup \{l_h, g_h, -\}$ defines its right boundary.

In the LTAG parsing algorithm presented in Section 3.3, boundaries of the yield are given using integer indices. Instead, we use either vertices V or special values l_h, g_h and $-$. If the left boundary of an item is node $v_m \in V$, the left (respectively right) boundary **position** is given by $\blacktriangle^{\text{left}}(v_m)$ (respectively $\blacktriangle^{\text{right}}(v_m)$). Special value l_h is used to indicate that the lexical anchor is used as a boundary. The remaining values are used to indicate that the boundary is

determined by the foot node span. The boundary is set to special value g_h if and only if the span of v_h has a gap, i.e. $\triangle(v_h) \neq \emptyset$. Finally, note that vertices which do not have a gap in their span, can still be combined through left or right adjunction. Thus, we can still use g_h in order to qualify the boundary of a sub-analysis which targets to be combined through a left (or right) adjunction. However, in this case, one of the boundaries is undefined, which is specified with the value $-$. Because boundaries are given by means of different kinds of values, we use the following syntactic sugar in order to minimize the number of combination rules:

$$\overleftarrow{v_m} \triangleq \blacktriangle^{\text{left}}(v_m)$$
$$\overrightarrow{v_m} \triangleq \blacktriangle^{\text{right}}(v_m)$$
$$\overleftarrow{l_h} \triangleq v_h$$
$$\overrightarrow{l_h} \triangleq v_h$$
$$\overleftarrow{g_h} \triangleq \begin{cases} \triangle^{\text{left}}(v_h) & \text{if } \triangle(v_h) \neq \emptyset \\ pred(\blacktriangle^{\text{right}}(v_h)) & \text{otherwise} \end{cases}$$
$$\overrightarrow{g_h} \triangleq \begin{cases} \triangle^{\text{right}}(v_h) & \text{if } \triangle(v_h) \neq \emptyset \\ succ(\blacktriangle^{\text{left}}(v_h)) & \text{otherwise} \end{cases}$$

Note that $\overrightarrow{-}$ and $\overleftarrow{-}$ are undefined, meaning that rules which use these values in their side condition cannot be applied.

Tree τ is a candidate for word represented by vertex v_h and its dependents if we can go up at its root node with boundaries equal to the span of v_h. In order to simplify notations, we use intermediate items to represent them. If the item has both boundaries defined, then:

Deduction rule: Full

$$\frac{[v_h, \tau, 1, \top, b_l, b_r]}{[v_h, \tau]} \overleftarrow{b_l} = \blacktriangle^{\text{left}}(v_h) \wedge \overrightarrow{b_r} = \blacktriangle^{\text{right}}(v_h)$$

If $h \neq r$, this item will be a candidate to combination through substitution (respectively adjunction) if $\tau \in T^I$ (respectively $\tau \in T^A$), or similarly, if v_h has no gap (respectively has a gap). Two other intermediate items are used specifically to indicate that they are meaning to be combined through left and right adjunction:

Deduction rule: Full left

$$\frac{[v_h, \tau, 1, \top, b_l, -]}{[v_h, \tau, \leftarrow]} \overleftarrow{b_l} = \blacktriangle^{\text{left}}(v_h)$$

Deduction rule: Full right

$$\frac{[v_h, \tau, 1, \top, -, b_r]}{[v_h, \tau, \rightarrow]} \overrightarrow{b_r} = \blacktriangle^{\text{right}}(v_h)$$

A triplet item ending with the $\leftarrow$ (respectively $\rightarrow$) symbol is a candidate for left (respectively right) adjunction. Obviously, τ in the antecedent of both rules must be an auxiliary tree. This is constrained by the **Foot scan** rule introduced in the following subsection. We observe in antecedents that one of the boundaries is unknown: this is due to the fact that the vertex has no gap, thus we cannot deduce the yield span of the content that will be placed under the foot node of τ in the derived tree.

5.6.3 Axioms and goal

The first axiom is:

Deduction rule: Lex scan

$$\frac{}{[v_h, \tau, p, \top, l_h, l_h]} \; lex(\tau) = p \wedge \tau(p) = s_h$$

meaning that, for each vertex v_h and elementary tree τ, we instantiate items with lexical anchor s_h, starting at the lexical anchor address. Moreover, we create items at the foot position of an auxiliary tree for vertices with a gap in their yield:

Deduction rule: Foot scan

$$\frac{}{[v_h, \tau, p, \top, g_h, g_h]} \; \tau \in \Gamma^A \wedge foot(\tau) = p \wedge \vartriangle(v_h) \neq \emptyset$$

Finally, the last two axioms are used to predict possible trees, on vertices without gap, that will be combined through left or right adjunction:

Deduction rule: Foot scan left

$$\frac{}{[v_h, \tau, p, \top, g_h, -]} \; \tau \in \Gamma^A \wedge foot(\tau) = p \wedge \vartriangle(v_h) = \emptyset$$

Deduction rule: Foot scan right

$$\frac{}{[v_h, \tau, p, \top, -, g_h]}\ \tau \in \Gamma^A \wedge foot(\tau) = p \wedge \vartriangle(v_h) = \emptyset$$

A proof completes if any tree $\tau \in \Gamma^S$ is a candidate for the root vertex v_r of the dependency structure:

Deduction rule: Goal

$$\frac{[v_r, \tau]}{}\ \tau \in \Gamma^S$$

In the rest of this section, we describe rules governing allowed deductions.

5.6.4 Traversal rules

We start with tree traversal.[12] Obviously, the premise of any move operation is that we already checked any potential operation, marked by the $\top$ flag. Given address $p \cdot 1$ in tree τ, we consider two cases. First, if node $p \cdot 1$ does not have any sibling, i.e. $p \cdot 2 \notin \tau$:

Deduction rule: Move unary

$$\frac{[v_h, \tau, p \cdot 1, \top, b_l, b_r]}{[v_h, \tau, p, \bot, b_l, b_r]}\ (p \cdot 2) \notin \tau$$

Secondly, if $p \cdot 2$ exists, the siblings must be contiguous:

Deduction rule: Move binary

$$\frac{[v_h, \tau, p \cdot 1, \top, b_{l1}, b_{r1}] \qquad [v_h, \tau, p \cdot 2, \top, b_{l2}, b_{r2}]}{[v_h, \tau, p, \bot, b_{l1}, b_{r2}]}\ succ(\overrightarrow{b_{r1}}) = \overleftarrow{b_{l2}}$$

[12] As previously, we assume binary elementary trees in the following presentation, but this can be generalized to other tree structures.

5.6.5 Combination rules

Finally, let us concentrate on combination operations. The simplest one is substitution. Substituting initial tree τ' anchored at s_m into site p of elementary tree τ anchored at s_h is formally defined as:

Deduction rule: Substitute

$$\frac{[v_m, \tau']}{[v_h, \tau, p, \top, v_m, v_m]} \; \vartriangle(v_m) = \emptyset \wedge \tau'(1) = \tau(p) \wedge (p \cdot 1) \notin \tau \wedge \tau' \in f_{SS}(\tau, p)$$

The first condition checks that sub-analysis in the antecedent does not have a gap. From the axioms and the fact that the antecedent is not a triplet, it follows that τ' is an initial tree. The second condition checks that both labels match. If τ' is a well-defined tree, then $\tau'(1)$ and $\tau(p)$ must be non-terminals. Finally, the third and last condition checks that the substitution targets a leaf.

The wrapping adjunction combines a modifier with a gap:

Deduction rule: Wrapping adjoin

$$\frac{[v_m, \tau'] \qquad [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, v_m, v_m]} \; \vartriangle^{\text{left}}(v_m) = \overleftarrow{b_l} \wedge \vartriangle^{\text{right}}(v_m) = \overrightarrow{b_r} \wedge \tau' \in f_{SA}(\tau, p)$$

The first two conditions check that the gap of the left antecedent is equal to the yield of the right antecedent. Similarly as for substitution, τ' is constrained to be an auxiliary tree from the axioms. The last condition checks that adjoining τ' into site p in tree τ is allowed. Note that, in the consequent, both boundaries are defined by the adjoined child: it is a wrapping adjunction.

Left and right adjunctions deal with vertices without gap. The adjoined tree is a triplet with third item forcing the direction:

Deduction rule: Left adjoin

$$\frac{[v_m, \tau', \leftarrow] \qquad [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, v_m, b_r]} \; \blacktriangle^{\text{right}}(v_m) + 1 = \overleftarrow{b_l} \wedge \tau' \in f_{SA}(\tau, p)$$

Deduction rule: Right adjoin

$$\frac{[v_m, \tau', \rightarrow] \, [v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, b_l, v_m]} \; \blacktriangle^{\text{left}}(v_m) - 1 = \overrightarrow{b_r} \wedge \tau' \in f_{SA}(\tau, p)$$

Finally, if allowed, adjunction may be skipped at the current site:

Deduction rule: Null adjoin

$$\frac{[v_h, \tau, p, \bot, b_l, b_r]}{[v_h, \tau, p, \top, b_l, b_r]} \neg f_{OA}(\tau, p)$$

5.6.6 Example

We now give a simple example to illustrate our algorithm with the sentence "`What does she deliberately walk?`" and its associated phrase-structure depicted in Figure 5.11a. Furthermore, we suppose a toy grammar that can be used to build this sentence only, see Figure 5.11b. The example includes substitution, wrapping adjunction and left adjunction. The input is the dependency analysis in Figure 5.11c. Observe that the root vertex v_2 is the only vertex with a block degree equal to two is v_5. After the parse tree labeler is run, one obtains the derivation tree in Figure 5.11d which can be used to build the derived tree. In this toy example, the output is a single derivation tree. With an ambiguous input sentence and a more complex grammar, the algorithm would have returned several derivation trees with the same dependency structure. If the input is incompatible with the grammar, then the algorithm fails and outputs an error.

The trace is given in Table 5.4. To simplify the execution, we use a *quaternary-move* operation. Alternatively, we could have binarized all elementary trees, which is a common practice in TAG parsing. It does not impact the asymptotic complexity.

5.6.7 Correctness

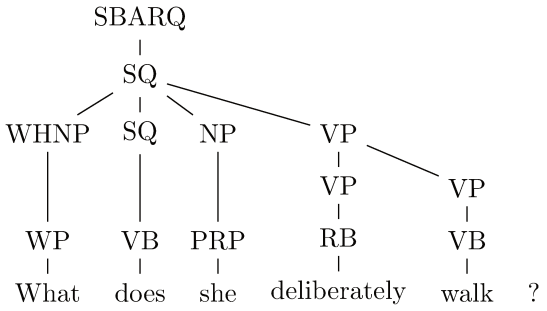
Since we use the deductive parsing framework, proving the correctness of the algorithm is straightforward from the notion of item invariant. Proving that every production rule maintains this invariant gives us soundness. Conversely, completeness can be proven by induction on items. In the following, we explain our invariant.

An item $[v_h, \tau, p, c, b_l, b_r]$ can be deduced from the axioms through the application of deduction rules if and only if, with respect to the input dependency parse, $\tau(p)$ can be derived to generate the subsequence of terminals and foot nodes:

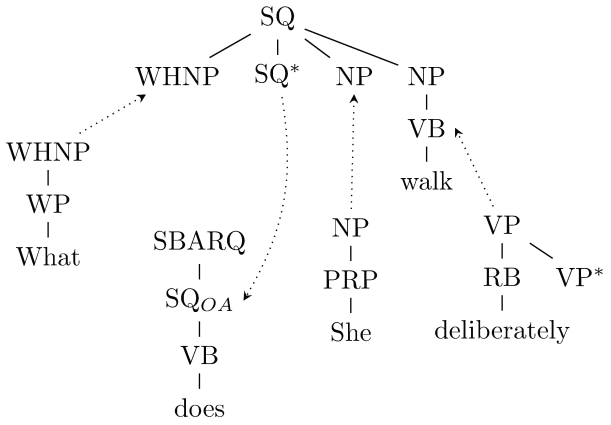
- if address p in τ does not dominate a foot node, the subsequence is $\overleftarrow{b_l} \ldots \overrightarrow{b_r}$;
- if p in τ dominates a foot node and this is not a left nor a right adjunction $b_l, b_r \notin \{\overleftarrow{g_h}, \overrightarrow{g_h}\}$, the subsequence is $s_{\blacktriangle^{\text{left}}(b_l)} \ldots s_{pred(\triangle^{\text{left}}(v_h))} X^* s_{succ(\triangle^{\text{right}}(v_h))} \ldots s_{\blacktriangle^{\text{right}}(b_r)}$;

	Item	Rule
1.	$[v_1, \tau_1, 1.1.1, \top, l_1, l_1]$	lex-scan (What)
2.	$[v_1, \tau_1, 1.1, \bot, l_1, l_1]$	move-unary
3.	$[v_1, \tau_1, 1.1, \top, l_1, l_1]$	null-adjoin
4.	$[v_1, \tau_1, 1, \bot, l_1, l_1]$	move-unary
5.	$[v_1, \tau_1, 1, \top, l_1, l_1]$	null-adjoin
6.	$[v_1, \tau_1]$	full
7.	$[v_3, \tau_3, 1.1.1, \top, l_3, l_3]$	lex-scan (she)
8.	$[v_3, \tau_3, 1.1, \bot, l_3, l_3]$	move-unary
9.	$[v_3, \tau_3, 1.1, \top, l_3, l_3]$	null-adjoin
10.	$[v_3, \tau_3, 1, \bot, l_3, l_3]$	move-unary
11.	$[v_3, \tau_3, 1, \top, l_3, l_3]$	null-adjoin
12.	$[v_3, \tau_3]$	full
13.	$[v_4, \tau_4, 1.1.1, \top, l_4, l_4]$	lex-scan (deliberately)
14.	$[v_4, \tau_4, 1.1, \bot, l_4, l_4]$	move-unary
15.	$[v_4, \tau_4, 1.1, \top, l_4, l_4]$	null-adjoin
16.	$[v_4, \tau_4, 1.2, \top, g_4, -]$	foot-scan-left
17.	$[v_4, \tau_4, 1, \bot, l_4, -]$	move-binary with 15
18.	$[v_4, \tau_4, 1, \top, l_4, -]$	null-adjoin
19.	$[v_4, \tau_4, \leftarrow]$	full-left
20.	$[v_5, \tau_5, 1.1, \top, v_1, v_1]$	substitute with 6
21.	$[v_5, \tau_5, 1.2, \top, g_5, g5]$	foot-predict
22.	$[v_5, \tau_5, 1.3, \top, v_3, v_3]$	substitute with 12
23.	$[v_5, \tau_5, 1.4.1.1, \top, l_5, l_5]$	lex-scan (walk)
24.	$[v_5, \tau_5, 1.4.1, \bot, l_5, l_5]$	move-unary
25.	$[v_5, \tau_5, 1.4.1, \top, l_5, l_5]$	null-adjoin
26.	$[v_5, \tau_5, 1.4, \bot, l_5, l_5]$	move-unary
27.	$[v_5, \tau_5, 1.4, \top, v_4, l_5]$	left-adjoin 19
28.	$[v_5, \tau_5, 1, \bot, v_1, l_5]$	move-quaternary with 20, 21, 22
29.	$[v_5, \tau_5, 1, \top, v_1, l_5]$	null-adjoin
30.	$[v_5, \tau_5]$	full
31.	$[v_2, \tau_2, 1.1.1.1, \top, l_2, l_2]$	lex-scan (does)
32.	$[v_2, \tau_2, 1.1.1, \bot, l_2, l_2]$	move-unary
33.	$[v_2, \tau_2, 1.1.1, \top, l_2, l_2]$	null-adjoin
34.	$[v_2, \tau_2, 1.1, \bot, l_2, l_2]$	move-unary
35.	$[v_2, \tau_2, 1.1, \top, v_1, v_5]$	wrapping-adjoin with 30
36.	$[v_2, \tau_2, 1, \bot, l_2, l_2]$	move-unary
37.	$[v_2, \tau_2, 1, \top, l_2, l_2]$	null-adjoin
38.	$[v_2, \tau_2]$	full

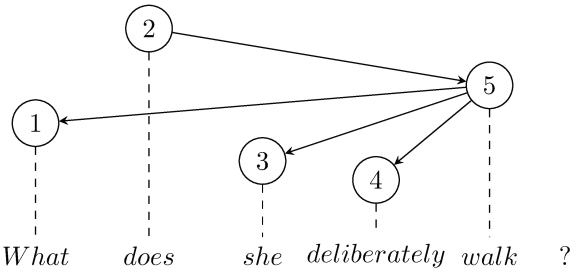
Table 5.4: Trace of the parsing example. The antecedent of a unary rule is its previous line. For rules with several antecedents, we refer their line numbers.



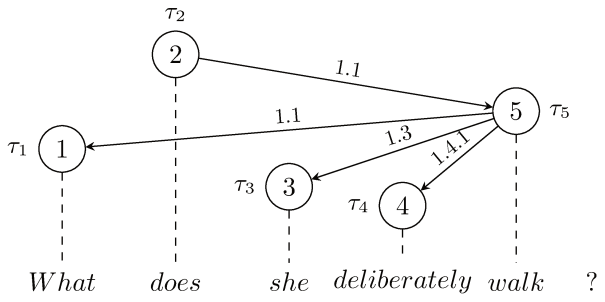
(a) Phrase-structure/derived tree



(b) LTAG analysis, the *OA* in subscript indicates an obligatory adjunction site



(c) Dependency analysis



(d) Derivation tree

Figure 5.11: Phrase-structure of the sentence "`What does she deliberately walks?`" alongside a LTAG analysis which relies on wrapping and left adjunctions.

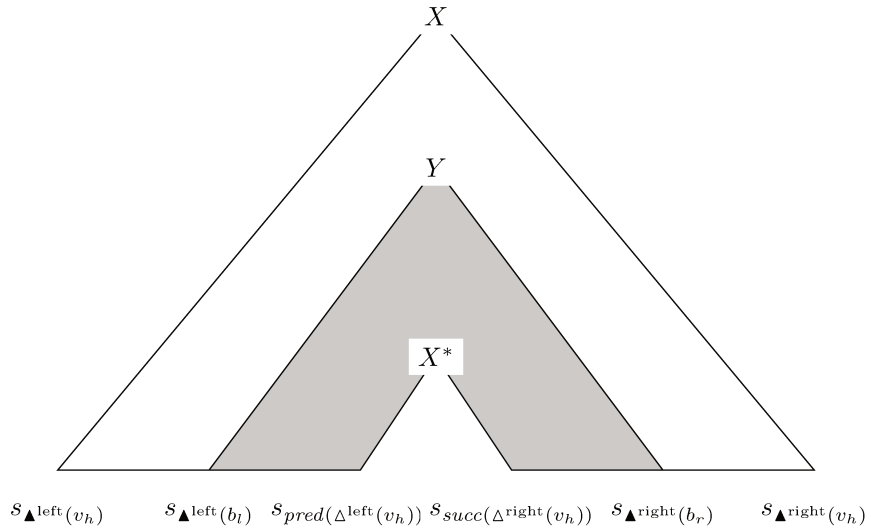


Figure 5.12: Invariant of an item $[v_h, \tau, p, c, b_l, b_r]$ when vertex v_h has a block degree of 2. X is the root node of τ, X^* its foot node and Y the node at address p. Only the gray area has been parsed.

- if p in τ dominates a foot node and $b_l = \overleftarrow{g_h}$, the subsequence is $X^* s_{\blacktriangle^{\text{left}}(v_h)} \dots s_{\blacktriangle^{\text{right}}(b_r)}$;
- if p in τ dominates a foot node and $b_r = \overrightarrow{g_h}$, the subsequence is $s_{\blacktriangle^{\text{left}}(b_l)} \dots s_{\blacktriangle^{\text{right}}(v_h)} X^*$;

with X^* the foot node of τ. A visualization is provided on Figure 5.12.

5.6.8 Complexity

It is common practice to directly deduce space and time complexities from item structures and deduction rules, respectively. However, improving bounds in this setting may lead to algorithms difficult to understand. Thus, we decided to propose a deduction-based algorithm that is simple to understand but which naively exposes a loose upper bound on its underlying complexity. In this section, we prove that the space and time complexities are linear.

In order to simplify the analysis, we suppose an agenda-based implementation [Kay, 1986]. Each deduced item is placed in an agenda. While the agenda is not empty, the main loop pops out an item from it and adds it to the chart. Then, the popped out item is tested as an antecedent, and all deduced consequents are pushed into the agenda if not already present in the chart. See Algorithm 3 for an outline of the algorithm.

Before analysing the algorithm complexity, we observe that the first value of an item, the current vertex, is redundant. Indeed, given the value of the left boundary (or right boundary), we can always retrieve the considered vertex in

Algorithm 3 Outline of the parsing algorithm. Lines 17-20 apply the **move binary** rule.

1: $A \leftarrow []$ ▷ Empty Agenda
2: **for** $1 \leq m \leq n$ **do** ▷ Init
3: **for** $\tau \in \Gamma$ **do** ▷ Lex. scan
4: **if** $\tau(lex(\tau)) = s_m$ **then**
5: $A.push([\tau, lex(\tau), \top, l_m, l_m])$
6: **for** $\tau \in \Gamma^A$ **do** ▷ Foot scan
7: **if** $(v_m)_{\rightarrow} \neq -$ **then**
8: $A.push([\tau, foot(\tau), \top, g_m, g_m])$
9: **else**
10: $A.push([\tau, foot(\tau), \top, \overleftarrow{g_m}, \overleftarrow{g_m}])$
11: $A.push([\tau, foot(\tau), \top, \overrightarrow{g_m}, \overrightarrow{g_m}])$
12: $C \leftarrow []$ ▷ Empty Chart
13: **while** $|A| > 0$ **do**
14: $[\tau, p \cdot 1, \top, b_{l1}, b_{r1}] \leftarrow A.pop()$
15: $C.add([\tau, p \cdot 1, \top, b_{l1}, b_{r1}])$
16: ▷ **apply Move binary**
17: Let b_{l2} be the unique boundary with $(b_{r1})_{\Rightarrow} + 1 = (b_{l2})_{\Leftarrow}$
18: **for** $[\tau, p \cdot 2, \top, b_{l2}, b_{r2}] \in C$ **do**
19: **if** $[\tau, p, \top, b_{l1}, b_{r2}] \notin C$ **then**
20: $A.push([\tau, p, \top, b_{l1}, b_{r2}])$
21: ...Apply other rules...

constant time. Obviously, when the boundary is given by a vertex v_m, we have $v_h = (v_m)_{\uparrow}$.[13] For special boundary values, all indexed by a word position h, a similar operation is straightforward. For example, if a boundary is given by l_h then the considered vertex is v_h.

The algorithm has a maximum of two nested loops: the main while loop and for loops matching the second antecedent of binary rules. We first consider the while loop. An item is added to the agenda if and only if it is not present in the chart. Thus, each item is considered exactly once by this loop. We note n the length of the input sentence, t the maximum number of nodes in an elementary tree $\tau \in \Gamma$ and $g = |\Gamma|$ the number of elementary trees.[14] Naively, the number of items is then bounded by $\mathcal{O}(n^2tg)$. However, the number of combination operations which can be applied on an elementary tree is bounded by its number of nodes, provided we dismiss multiple adjunction sites. In this case, each node of an elementary tree may be adjoined or substituted on at most once. Thus, given an elementary tree and a left boundary, the number of values allowed as a right boundary is limited. This leads to a tighter bound on the number of items: $\mathcal{O}(\min(t, n)ntg)$.

[13] We assume that the data structure storing the dependency graph provides such a function in constant time.

[14] Alternatively, g can be the number of elementary trees associated with the most ambiguous word of the vocabulary.

We now investigate time complexity. **Move binary** is the only rule which has two free antecedents. Indeed, it is easy to see that, in the other binary rules, fixing the right antecedent always fixes the left one. For the **Move binary** rule, given the left antecedent, the number of candidates for the right one is naively bounded by $\mathcal{O}(n)$. However, we previously argued that, given a fixed left boundary, the maximum number of right boundary alternatives cannot exceed the number of sites on the current elementary tree. Thus, we can tighten the bound to $\mathcal{O}(\min(t, n))$.

In conclusion, the time complexity of the proposed algorithm is $\mathcal{O}(\min(t, n)^2 ntg)$, that is asymptotically linear with respect to the input sentence length.

5.7 Conclusion

In this chapter, we proposed a novel pipeline algorithm for the LTAG parsing problem. The first step is a dependency parsing task with a complexity roughly similar to LTAG parsing. The second step is a parse tree labeler with linear time complexity. As we do not have a LTAG corpus annotated with wrapping adjunctions, we experimented our dependency parser on different datasets, demonstrating at the same time the efficiency and the usefulness of the proposed approach.

Moreover, we introduced the Yield Restricted Maximum Spanning Arborescence problem. It is a generalization of the well-nested MSA with k-BBD problem. This modelization allows to change the block degree bound easily and to toggle the well-nestedness constraint. We proposed an efficient decoding algorithm based on Lagrangian relaxation. Experimentally, the resulting parser is on par in terms of accuracy scores but returns valid structures quicker than high-order models. More importantly, the high-order models happen to return more valid structures but they do not guarantee it.

Chapter 6

Discontinuous phrase-structure Parsing via the Generalized Maximum Spanning Arborescence

In the previous chapter, we proposed a new pipeline algorithm for LTAG parsing. We identified the constrained dependency structure of LTAG derivation trees as the bottleneck and proposed two novel ILP formulations of the problem. But can we get rid of the well-nestedness and 2-bounded block degree constraints? As such, the first step of the pipeline would reduce to the Maximum Spanning Arborescence (MSA) problem. We propose to remove the structural constraints on the derived tree: any crossing arc is allowed. The generative power of this grammar is in practice useless as it means that the word order is not taken into account. However, it is nonetheless useful for parsing. We motivate our work with discontinuous phrase-structure parsing in Section 6.1. The MSA can be solved in quadratic time, so we propose instead to decode dependencies and supertags jointly.

Given a graph where vertices are partitioned into clusters, a generalized spanning arborescence is a set a of arcs inducing an arborescence adjacent to exactly one vertex per cluster. In Section 6.2, we reduce the joint problem of supertagging and non-projective dependency parsing to Generalized Maximum Spanning Arborescence (GMSA): vertices in a cluster represent supertag candidates for a word. Unfortunately, computing the GMSA given an arc-factored model is a NP-hard problem [Myung et al., 1995]. The problem has received little attention from the combinatorial optimization community [Myung et al., 1995, Feremans et al., 1999, 2002, Pop, 2009]. In particular, it has mainly been studied as an alternative to solve the equivalent undirected problem, the Generalized Maximum Spanning Tree problem, that is on directed graphs with special structure: opposite arcs between two vertices have the same weight. We adapt an ILP

formulation of the problem [Pop, 2009] in Section 6.3 in order to propose a new decoding method for the GMSA based on dual decomposition. As such, we depart from previous techniques proposed in the optimization literature for this problem and follow a recent trend in the NLP community [Koo et al., 2010, Das et al., 2012, Le Roux et al., 2013, Almeida and Martins, 2013]. We describe our decomposition in Section 6.4. In Section 6.5, we introduce a decoding algorithm for our Lagrangian dual heuristic. We experiment its efficiency on a discontinuous phrase-structure parsing task in Section 6.6. Interestingly, we obtain a certificate of optimality for $\approx 99\%$ of the test instances without relying on Branch-and-Bound. Moreover, our approach results in low decoding time and high parsing scores on two standard benchmarks.

6.1 Motivations

Many head-modifier relationships can be directly deduced from the constituency structure. For example, the subject of a verb is attached at the clause `S` level and its object at the verbal phrase `VP` level. Words with non-local dependencies can introduce discontinuous constituency structures, for example in wh-questions: the object of the main verb, attached at the `VP` level, is the first word of the sentence and therefore crosses the subject constituent, see Figure 6.1. However, in the original Penn Treebank [Marcus et al., 1993] all constituents are constrained to be continuous so they can be parsed with CFGs. Thus, the annotation guide allows the introduction of empty anchors with traces that can be used to retrieve these non-trivial head-modifier relationships that cannot be represented with a continuous structure, see Figure 6.2 for an example. In practice, besides a few notable exceptions [Johnson, 2002, Cai et al., 2011, Kummerfeld and Klein, 2017], parsers ignore empty elements and thus are not able to recover traces. Although LTAGs generate continuous constituents only, they can encode non-local head-modifier attachments in the derivation tree, see Figure 6.3. Unfortunately, no large scale automatic or manual annotation that take into account these phenomena have been produced by the community. Evang and Kallmeyer [2011] proposed to automatically transform the continuous Penn Treebank into discontinuous phrase-structures. The resulting constituency tree cannot be parsed using CFGs or LTAGs. A common approach is to rely on Linear Context-Free Rewriting Systems (LCFRSs) [Weir, 1988], a discontinuous generalization of CFGs. However, this formalism has the same problem as CFGs concerning shallow versus deep syntax. Moreover, LCFRS parsers have prohibitive space and time complexities [Gómez-Rodríguez et al., 2010].

In order to combine the deep syntax ability of LTAGs with the discontinuous constituency representation, we propose to relax constraints on the dependency structure of derivation trees. As such, the derivation tree is constrained only with respect to anchors and combination constraints. Note that this relaxed grammar has an odd generative power: there is no constraint on word order. However, we focus on parsing so we simply ignore this problem.

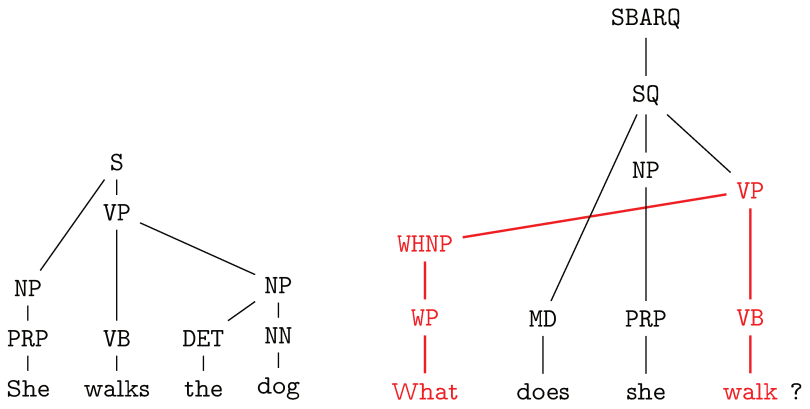


Figure 6.1: The wh-movement introduces a discontinuous constituent.

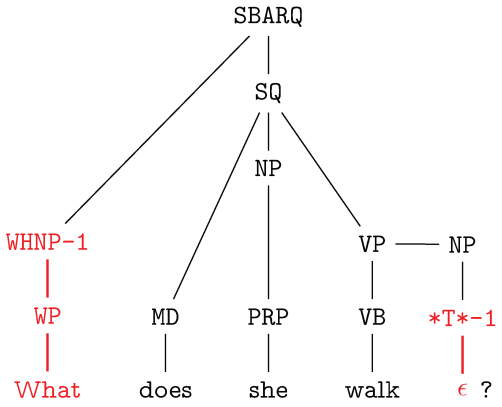


Figure 6.2: The Penn Treebank uses empty elements with traces in order to represent discontinuous constituents as continuous ones.

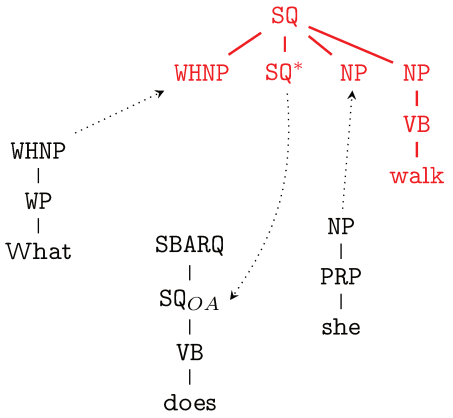
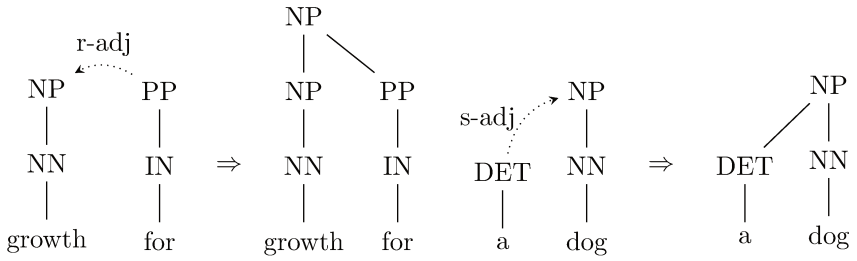


Figure 6.3: The adjunction operation in LTAGs can be used to represent discontinuous constituents in the derivation tree although the derived tree is necessarily continuous.



(a) Regular adjunction (b) Sister adjunction

Figure 6.4: Illustration of the two adjunction operations.

6.2 Reduction to the Generalized Maximum Spanning Arborescence problem

In this section, we show how the parsing problem with a variant of LTAGs can be reduced to the Generalized Maximum Spanning Arborescence (GMSA) problem. We consider spinal TAGs (see definition below) without any constraint on the derivation tree structure. Then, the LTAG parsing problem reduces to the joint supertagging and dependency parsing task:

1. Supertagging: select one elementary tree per word.

2. Dependency parsing: choose combination operations.

An ILP formulation of GMSA problem is described in the next section.

6.2.1 Spinal Tree Adjoining Grammar

In this subsection, we briefly explain the formalism derived from LTAG called Spinal Tree Adjoining Grammar (STAG) that we will use in order to parse discontinuous phrase-structures. Importantly, we do not require derivations to be LTAG compatible (i.e. well-nested dependencies with a 2-bounded block degree). Moreover, elementary tree structures and combination operations differ from LTAG.

A spine is a lexicalized tree where each node labeled with a non terminal has a single child. In other words, in contrast with a LTAG's elementary trees, a spine is a single path from the root node to the lexical anchor. Derived trees are built from two kinds of operations: **regular adjunctions** and **sister adjunctions**. There is no substitution operation. Contrary to auxiliary trees, spines do not have a foot note. A regular adjunction duplicates the target site. A sister adjunction attaches both spines together directly. The two operations are illustrated in Figure 6.4. There is no restriction on operation sites.[1] This formalism and its linguistic relevance has been investigated by Shen [2006]. Figure 6.5 illustrates an example of analysis.

[1]Except, obviously, that we cannot adjoin into a node labeled with a part of speech tag.

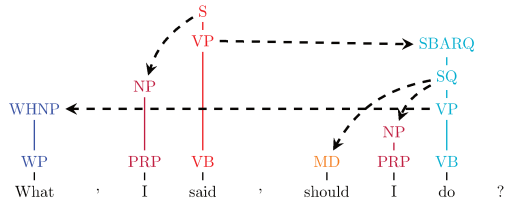


Figure 6.5: A derivation with spines and adjunctions (dashed arrows). The induced dependency tree is non-projective. Each color corresponds to a spine. All operations are sister adjunctions.

6.2.2 Generalized Spanning Arborescence

Let $D = (V, A)$ be a directed graph and $T \subseteq A$ be a subset of arcs. Let $\pi = \{V_0, \ldots, v_n\}$ be a partition into clusters of V. In other words, $\bigcup_{W \in \pi} W = V$ and $W \cap W' = \emptyset$ for any $W, W' \in \pi$ such that $W \neq W'$. We remind that $V[T] \subseteq V$ is the subset of vertices that are adjacent to at least one arc in T. Recall that a subset $T \subseteq A$ of arcs is called an arborescence if the graph $(V[T], T)$ is connected, acyclic and each vertex has at most one entering arc. The vertex with no entering arc is called the root of T.

Definition 6.1: Generalized Spanning Arborescence
A subset of arcs T is a Generalized Spanning Arborescence (GSA) if and only if [Myung et al., 1995]:

- T is an arborescence;
- $V[T]$ contains exactly one vertex in each cluster of π.

The cluster which contains the only vertex without incoming arc is called the root cluster.

Without loss of generality, in the following we assume that we want to compute a GSA rooted at cluster W_0 and that $W_0 = \{v_0\}$ is a singleton. Thus, we call such a GSA a v_0-rooted GSA. Figure 6.6 gives an example of a v_0-rooted GSA. The partition of V is composed of a cluster having one vertex and six clusters having four vertices each. Every cluster is depicted by a hatched area. The GSA is given by the arcs and its adjacent vertices are the orange ones.

The Maximum v_0-rooted Generalized Spanning Arborescence (GMSA) problem consists in computing the v_0-rooted GSA with maximum score according to a weighting function $f : \mathcal{P}(A) \to \mathbb{R}$. As with the MSA problem, a weighting function is said to be arc-factored if and only if it can be written as:

$$f(T) = \sum_{a \in T} w_a$$

where w_a is the score of a single arc. Note that we can include weights associated with vertices in their incoming arcs. Contrary to the arc-factored MSA problem

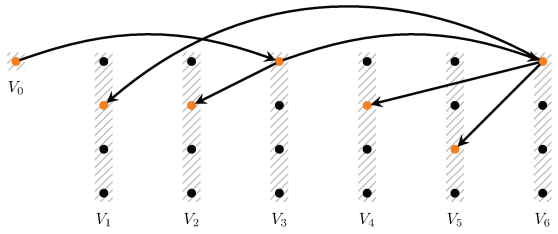


Figure 6.6: Example of a v_0-rooted generalized spanning arborescence. Dashed areas represent clusters.

which has a quatratic-time complexity, the arc-factored GMSA problem is NP-hard [Myung et al., 1995].[2]

6.2.3 Reduction

Given an instance of the STAG problem with sentence $s = s_1 \dots s_n$, we construct an instance of GMSA as follows. With every spine of every word s_k, we associate a vertex v. For $k = 1, \dots, n$, we denote by V_k the set of vertices associated with the spines of s_k. Moreover, we add a set V_0 containing only one vertex and V_0 will now refer both to the cluster and to the vertex it contains depending on the context. Let $\pi = \{V_0, \dots, V_n\}$ and $V = \cup_{k=0}^{n} V_k$. For each couple v_i, v_j of vertices such that $v_i \in V_h$ and $v_j \in V_m$, $h \neq m$ and $m \neq 0$, we associate an arc $v_i \to v_j$ corresponding to the best adjunction of the root of the spine associated with v_j of V_m to the spine associated with vertex v_i of V_h. The weight associated with the arc can be expressed in terms of the modifier spine, the head spine and/or the adjunction operation. This ends the construction of (D, π, w).

There is a 1-to-1 correspondence between the solutions to GMSA and those to the joint supertagging and spine parsing task in which each adjunction is performed with the label maximizing the score of the adjunction. Indeed, the vertices covered by a GSA T with root V_0 correspond to the spines on which the derivation is performed. By definition of GSAs, one spine per word is chosen. Each arc of T corresponds to an adjunction. The score of the arborescence is the sum of the scores of the selected spines plus the sum of the scores of the best adjunctions with respect to T. Hence, one can solve GMSA to perform joint tagging and non-projective dependency parsing.

As an illustration, the GSA depicted in Figure 6.7 represents the derivation tree of Figure 6.5: the vertices of $V \setminus V_0$ covered by the GSA are those associated with the spines of Figure 6.5 and the arcs represent the different adjunctions. For instance the arc from V_3 to V_2 represents the adjunction of spine `NP-PRP` to spine `S-VP-VB` at index 0.

[2] In the EMNLP publication where our work was published, we described the weighting model differently. However, both approaches are equivalent and we stick to arc-factored models to ensure the consistency of this thesis.

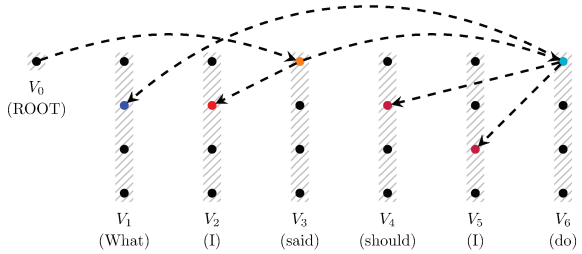


Figure 6.7: The generalized spanning arborescence inducing the derivation tree in Figure 6.5.

6.3 Integer Linear Programming formulation

Let $x \in \mathbb{B}^A$ and $y \in \mathbb{B}^V$ be the variable vectors associated with arcs and vertices. An arc $a \in A$ (respectively a vertex $v \in V$) is selected if and only if $x_a = 1$ (respectively $y_v = 1$). We want to constrain assignments on both vectors such that vector x can only represent a v_0-rooted GSA and y its incident vertices.

Contracting a vertex subset $W \subseteq V$ consists in removing vertices W and adding a vertex w. Every arc $u \to v \in \delta^{in}(W)$ is replaced by an arc $u \to w$ and every arc $v \to u \in \delta^{out}(W)$ by $w \to u$. We denote D^π the graph obtained by contracting each cluster of π in D. Note that a GSA of D and π induces a spanning arborescence in D^π. For instance, contracting each cluster in the graph given by Figure 6.6 leads to a graph D^π having 7 vertices and the set of dashed arcs corresponds to a spanning arborescence of D^π, see Figure 6.8. Thus, y must satisfy the following constraints, adapted from the MSA [Schrijver, 2003] on clusters:

$$x(\delta^-(V_0)) = 0 \tag{6.1}$$

$$x(\delta^-(V_k)) = 1 \qquad \forall 1 \leq k \leq n \tag{6.2}$$

$$x(\delta^-(\bigcup_{V_k \in \pi'} V_k)) \geq 1 \qquad \forall \pi' \subseteq \pi \setminus \{V_0\} \tag{6.3}$$

In order to simplify notation, we define the polytope of spanning arborescences on cluster:

$$\mathcal{X}^\pi = \{x \in \{0,1\}^A | x \text{ satisfies (6.1)-(6.3)}\}$$

Thus, we will use constraint $x \in \mathcal{X}^\pi$ as a linear constraint in the following.

Note that $\mathcal{X}^\pi$ includes invalid structures, see Figure 6.9 for an example. Indeed, we need to add constraints between vectors of variables (adjacency) and force that the structure contains only one vertex per cluster in y. The GMSA

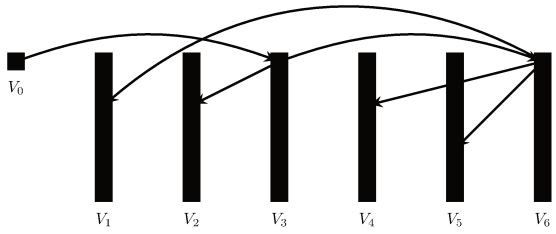


Figure 6.8: If we contract all the clusters of the graph in Figure 6.6, we obtain a spanning arborescence.

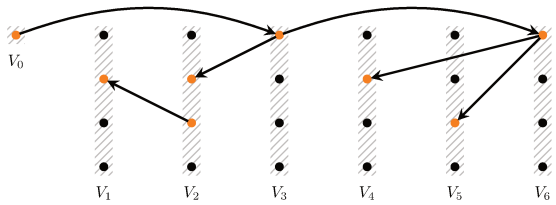


Figure 6.9: Forcing the structure to be an arborescence over clusters is not enough to ensure that it is a GSA. In this example, there are 2 adjacent vertices in cluster V_2.

problem can be formulated with the following ILP:

$$\max_{x,y} \quad w^\top x \tag{6.4}$$

$$\text{s.t.} \quad x \in \mathcal{X}^\pi \tag{6.5}$$

$$y_v \geq x_a \qquad \forall v \in V, a \in \delta(v) \tag{6.6}$$

$$y_v(V_k) = 1 \qquad \forall 0 \leq k \leq n \tag{6.7}$$

$$x \in \mathbb{B}^A, y \in \mathbb{B}^V \tag{6.8}$$

Constraint set (6.5) forces selected arcs to be a spanning arborescence over clusters. Constraint set (6.6) ensures that a vertex is selected if at least one of its adjacency arcs is selected. There is exactly one selected vertex per cluster thanks to constraints (6.7). Note that because of (6.5), (6.6) and (6.7), a vertex is selected if and only if it has an adjacent arc. Finally, the objective function (6.4) seeks to maximize the sum of selected arc weights. Thus, the optimal solution of this ILP is the arc-factored v_0-rooted GMSA of the graph. We propose a new efficient algorithm for computing the GMSA based on a relaxation of this ILP.

6.4 Dual decomposition

To perform the dual decomposition, we first reformulate the ILP (6.4)-(6.8) by introducing copy variables, as explained in Section 4.3.3. For this purpose, we replace the variables x by three copies $\{x^i\} = \{x^1, x^2, x^3\}$, $x^i \in \mathbb{B}^A$ and the witness vector $z \in \mathbb{R}^A$. Let w^1, w^2 and w^3 be weight vectors such that

$\sum_{i=1}^{3} w^i = w$. A simple strategy, which we follow, is to define $w^1 = w^2 = w^3 = \frac{1}{3}w$. The GMSA can then be reformulated as:

$$\max_{\{x^i\},y,z} \quad \sum_{i=1}^{3} w^{i\top} x^i \tag{6.9}$$

$$\text{s.t.} \quad x^1 \in \mathcal{X} \tag{6.10}$$

$$y_v \geq x^2_a \qquad \forall v \in V, a \in \delta^-(v) \tag{6.11}$$

$$y_v \geq x^3_a \qquad \forall v \in V, a \in \delta^+(v) \tag{6.12}$$

$$y_v(V_k) = 1 \qquad \forall 0 \leq k \leq n \tag{6.13}$$

$$x^1, x^2, x^3 \in \mathbb{B}^A, x \in \mathbb{B}^V \tag{6.14}$$

$$z = x^i \qquad \forall 1 \leq i \leq 3 \tag{6.15}$$

Note that each copy only appears in a single constraint set. As usual in dual decomposition, the witness variable vector z only appears in equations (6.15). Their goal is to ensure equality between copies x^1, x^2 and x^3. Thus, this ILP is equivalent to the program (6.4)-(6.8).

We relax constraints (6.15) and build the following Lagrangian dual objective:

$$\mathcal{L}^*(\{\lambda^i\}) = \max_{\{x^i\},y,z} \quad \sum_{i=1}^{3} w^{i\top} x^i + \sum_{i=1}^{3} \lambda^{i\top} \cdot (z - x^i)$$
$$\text{s.t.} \quad (6.10) - (6.14)$$

where $\{\lambda^i\} = \{\lambda^1, \lambda^2, \lambda^3\}$, with $\lambda^1, \lambda^2, \lambda^3 \in \mathbb{R}^A$, is the set of Lagrangian multipliers. We restrict the domain of $\{\lambda^i\}$ in the dual problem to the set $\Lambda = \{\{\lambda^i\} | \sum_{i=1}^{3} \lambda^i = \mathbf{0}\}$ and rewrite the dual objective as (Section 4.3.3):

$$= \max_{\{x^i\},y} \quad \sum_{i=1}^{3} \bar{w}^{i\top} x^i$$
$$\text{s.t.} \quad (6.10) - (6.14)$$

where $\bar{w}^i = w^i - \lambda^i$ are reparameterized weight vectors. It is easy to see that computing the dual objective given the Lagrangian multipliers decomposes in two independent subproblems:

$$P_1(\bar{w^1}) = \max_{x^1} \bar{w}^{1\top} x^1 \qquad P_2(\bar{w^2}, \bar{w^3}) = \max_{x^2,x^3,y} \bar{w}^{2\top} x^2 + \bar{w}^{3\top} \cdot x^3$$
$$\text{s.t.} \quad x^1 \in \mathcal{X}^\pi \qquad \text{s.t.} \quad (6.11) - (6.14)$$

Thus, our Lagrangian decomposition is a dual decomposition method. Efficient algorithms to solve both subproblems are proposed in Section 6.5.1.

The Lagrangian dual is:

$$\min_{\{\lambda^i\} \in \Lambda} L^*(\{\lambda^i\})$$

This problem can be minimized using the standard subgradient algorithm, Section 4.4.2. In the next section, we introduce efficient algorithms to solve both subproblems.

6.5 Efficient decoding

In this section we describe how the Lagrangian dual is efficiently minimized. We begin with quadratic algorithms to solve both subproblems (Subsection 6.5.1). These are sufficient to realize the subgradient descent. However, we then show that we can improve convergence with simple techniques that do not increase the complexity of subproblems (Subsection 6.5.2). Finally, we propose a method to prune the search space via problem reduction (Subsection 6.5.3).

6.5.1 Combinatorial algorithms for subproblems

Obviously, both subproblems could be solved using a generic solver. However, we prefer to rely on efficient combinatorial algorithms. The pseudo-code for both subproblems is described in Appendix D.

Subproblem P_1

Subproblem P_1 is:

$$\begin{aligned}
\max_{x^1} \quad & \bar{w}^{1\top} x^1 \\
\text{s.t.} \quad & x^1(\delta^-(V_0)) = 0 \\
& x^1(\delta^-(V_k)) = 1 && \forall 1 \leq k \leq n \\
& x^1(\delta^-(\bigcup_{V_k \in \pi'} V_k)) \geq 1 && \forall \pi' \subseteq \pi \setminus \{V_0\}
\end{aligned}$$

Recall that these constraints define a spanning arborescence over clusters. On the contracted graph D^π where all parallel arcs are removed except the one with maximum weight, they define a (standard) spanning arborescence. Thus, subproblem P_1 can be solved in quadratic time via the Chu–Liu/Edmonds' algorithm on the contracted graph.

Subproblem P_2

Subproblem P_2 is:

$$\begin{aligned}
\max_{x^2, x^3, y} \quad & \bar{w}^{2\top} x^2 + \bar{w}^{3\top} \cdot x^3 \\
\text{s.t.} \quad & y_v \geq x^2_a && \forall v \in V, a \in \delta^-(v) \\
& y_v \geq x^3_a && \forall v \in V, a \in \delta^+(v) \\
& y_v(V_k) = 1 && \forall 0 \leq k \leq n \\
& x^2, x^3 \in \mathbb{B}^A, y \in \mathbb{B}^V
\end{aligned}$$

Observe that each value of x^2 and x^3 is only constrained by a single value of y. The problem amounts to selecting a vertex for each cluster as well as:

- all its incoming arcs in x_2 with positive weight $\bar{w}^2$;
- all its outgoing arcs in x_3 with positive weight $\bar{w}^3$.

The optimal solution of P_2 is the assignment of variables following these rules with maximum objective value.

For each vertex $v \in V$, we compute its **local weight** w'_v:

$$w'_v = \sum_{a \in \delta^{in}(v)} \max(0, \bar{w}^2) + \sum_{a \in \delta^{out}(v)} \max(0, \bar{w}^3) \qquad \forall v \in V$$

which is the sum of all its incoming arcs with positive $\bar{w}^2$ weight and outgoing arcs with positive $\bar{w}^3$ weight. Let $V^{\max}$ be the set of vertices defined by:

$$V^{\max} = \bigcup_{V_c \in \pi} \operatorname*{argmax}_{v \in V_c} w'_v$$

This set contains the vertex with maximum local weight in each cluster. Let A^2 and A^3 be the sets of arcs such that A^2 (respectively A^3) contains all the arcs with positive weights entering (respectively leaving) a vertex of $V^{\max}$. The optimal solution of P_2 is the assignment of variables such that:

- y corresponds to the incident vector of $V^{\max}$.
- x^2 corresponds to the incident vector of $\delta^{in}(V^{\max})$;
- x^3 corresponds to the incident vector of $\delta^{out}(V^{\max})$;

This procedure can be realized with a $\mathcal{O}(|V|^2)$ time complexity. If the number of vertices per cluster is bounded by a constant, then the time complexity is $\mathcal{O}(|\pi|^2)$.[3]

6.5.2 Lagrangian enhancement

In practice, subgradient descent is proven to converge to the Lagrangian dual minimum in a bounded number of iterations. However, in practice, two problems are often encountered. First, the presence of a dual gap, meaning that we can never guarantee to return the optimal solution without relying on an exhaustive search method like branch-and-bound. Second, a common practice is often to stop the minimization after a number of iterations fixed in advance in order to guarantee short decoding time. Thus, the dual minimum may not be reached. In this section, we propose two improvements of the previous relaxation which aim at increasing the number of delivered certificates while decreasing in practice the number of required subgradient descent iterations.

Constraint tightening

Constraints which would have been redundant in the primal program may be of interest in the Lagrangian dual objective [Beasley, 1993]. We propose to add two sets of constraints to subproblem P_2:

$$\begin{aligned}
\max_{x^2, x^3, y} \quad & \bar{w}^{2\top} x^2 + \bar{w}^{3\top} x^3 \\
\text{s.t.} \quad & (6.11) - (6.14) \\
& x^2(\delta^{in}(V_k)) = 1 && \forall V_l \in \pi^+ && (6.16) \\
& x^3(\delta^{out}(V_k) \cup \delta^{in}(V_l)) \leq 1 && \forall V_k \in \pi, V_k \in \pi^+ \setminus \{V_k\} && (6.17)
\end{aligned}$$

[3] In our application, the number of vertices in a cluster is bounded by the grammar size.

Constraints (6.16) force each cluster, except the root, to have exactly one incoming arc in x^2. Constraints (6.17) ensure that there is at most one selected arc between two clusters. These two properties are obviously required in a GSA primal valid solution and are satisfied by any solution of P_1.

These two constraints can be added without impacting the asymptotic complexity of the algorithm. Intuitively, when computing the local weights, the first one requires to keep the incoming arc with maximum weight instead of all arcs with positive weights. The second one requires to store outgoing arcs in different containers, one per destination cluster. Then, instead of selecting all outgoing arcs with positive weights, we only add the one with maximal weight per destination cluster, if it has a positive weight.

Arc reweighting

Previously introduced constraint set (6.16) forces each cluster, except the root, to have exactly one incoming arc in the incidence vector x^2. Unfortunately, imposing such a constraint on x^3 while maintaining the same decoding algorithm is not straightforward.[4] We propose to change the initial arc weight vector w so P_2 would naturally tend to satisfy this constraint. This change does not impact the primal optimal solution set. In order to keep the same primal optimal score, a constant bias term can be added to the objective. For each cluster $V_k \in \pi \setminus \{V_0\}$, let $m_k = \max_{a \in \delta^{in}(V_k)} w_k$ be the maximal score of its incoming arcs.[5] We modify weights such that less incoming arcs have positive weights. Let ϵ_k be a value such that $m_k - \epsilon_k$ is positive only for arcs in $\delta^{in}(V_k)$ which have weight m_k. Subtracting ϵ_k from the weight w_a of each arc of $\delta^{in}(V_k)$ does not modify the primal optimal solution set because only one entering arc per cluster is selected.

6.5.3 Problem reduction

We propose a problem reduction technique, in order to prune the search space, which is fast to compute. Let $\{\hat{x}^i\}, \hat{y}$ be an optimal solution of $\mathcal{L}^*(\{\lambda^i\})$ computed at any iteration of the subgradient algorithm and LB the best known lower bound. Let $v \in V$ be a vertex which is not in the current dual optimal solution, that is $\hat{y}_v = 0$. If we compute an upper bound to the GMSA with additional constraint $y_v = 1$ that is less than or equal to LB, then y_v cannot be in the primal optimal solution: y_v and its adjacent arcs can be safely removed from the graph. Let u be the vertex such that $\hat{y}_u = 1$ and both v and u are in the same cluster. Then, using the local weights (Subsection 6.5.1), $\mathcal{L}^*(\{\lambda^i\}) - c_u + c_v$ is an upper bound on the weight of any solution to the GMSA with $y_v = 1$. In fact, this is the solution of the Lagrangian dual objective with the same Lagrangian multipliers and the additional constraint. This operation can be realized in constant time with appropriate cache strategy. We can check the whole graph in linear time with respect to the number of vertices. If the number of vertices per cluster is bounded, then the problem reduction algorithm has a $\mathcal{O}(|\pi|)$ time complexity.

[4] We strongly suspect that it will impact the asymptotic complexity.

[5] Several arcs may have the same weight, so there may be more than one arc with maximal score.

6.6 Experimental results

In this section, we run experiments with the GMSA as a framework for discontinuous constituency parsing with STAGs. Arc weights are computed using neural networks (Subsection 6.6.1). We test our model on two datasets (Subsection 6.6.2) and obtain state of the art results while being very fast (Subsection 6.6.3). This demonstrates the relevance of the GMSA as a model for NLP problems.

6.6.1 Neural parameterization

We rely on a conditional probabilistic model for our framework. We implement our probability distributions with neural networks, more specifically we build a neural architecture on top of bidirectional recurrent networks, more precisely bi-LSTMs [Hochreiter and Schmidhuber, 1997], that compute context sensitive representations of words. At each step, the recurrent architecture is given as input a concatenation of word and part of speech embeddings. We refer the reader to Goldberg [2016] for more information.

Given a sentence, we assume a derivation is generated by three distinct tasks: supertagging, non-projective dependency parsing and arc labeling. As previously, we follow the common approach in dependency parsing and assign labels in a post-processing step, although our model is able to incorporate label scores directly.[6] Thus, in the GMSA we are left with jointly decoding a dependency structure and assigning a sequence of spines. We suppose that adjunctions are generated by an arc-factored model, and that a spine prediction depends on both current position and head position. Moreover, we assume independence between probabilities. Then parsing amounts to finding the most probable derivation and can be realized in the log space. A more precise description of our model is given in Appendix E.

6.6.2 Datasets

Discontinuous constituent parsing is not very common. Thus, there are only a few datasets available. We ran a series of experiments on the two main corpora annotated with discontinuous constituents.

English: We used an updated version of the Wall Street Journal part of the Penn Treebank corpus [Marcus et al., 1993] which introduces discontinuity [Evang and Kallmeyer, 2011]. As usual, sections 2-21 are used for training, 22 for developpement and 23 for testing. We used both gold and predicted POS tags by the Stanford tagger,[7] trained with 10-jackknifing.[8] Spines are extracted following the head-percolation table of Collins [1997].

German: We used the Tiger corpus [Brants et al., 2004] with the split defined for the SPMRL 2014 shared task [Maier, 2015, Seddah et al., 2013].

[6] Our motivation for this pipeline is to reduce the number of weights to compute and to mimic the standard dependency parsing architecture.

[7] `http://nlp.stanford.edu/software/tagger.shtml`

[8] Prediction precision: 97.40%

Following Maier [2015] and Coavoux and Crabbé [2017], we removed sentences number 46234 and 50224 as they contain annotation errors. We only used the given gold POS tags. Spines are extracted following the head-percolation table distributed with Tulipa [Kallmeyer et al., 2008].

6.6.3 Results

Regarding decoding, we keep only the 10 spines with maximum marginal probability but allow every possible adjunction. The maximum number of iterations of the subgradient descent is set to 500 and the stepsize is fixed following the rule of Polyak [1987].

Parsing results and timing on short sentences only ($\leq$ 40 words) and full test set using the default discodop[9] eval script are reported on Table 6.1 and Table 6.2.[10] We report labeled recall (LR), precision (LP), F-measure (LF) and time measured in minutes. We also report results published by van Cranenburgh et al. [2016] for the discontinuous PTB and Coavoux and Crabbé [2017] for Tiger. Moreover, dependency unlabeled attachment scores (UAS) and tagging accuracies (Spine acc.) are given on Table 6.3. We achieve significantly better results on the discontinuous PTB, while being roughly 36 times faster together with a low memory footprint.[11] On the Tiger corpus, we achieve on par results. Note however that Coavoux and Crabbé [2017] rely on a greedy parser combined with beam search.

Fast and efficient parsing of discontinuous constituent is a challenging task. Our method can quickly parse the whole test set, without any parallelization or GPU, obtaining an optimality certificate for more than 99% of the sentences in less than 500 iterations of the subgradient descent, partly thanks to the enhancement presented in Subsection 6.5.2 and the problem reduction algorithm in Subsection 6.5.3. Moreover, we obtain state-of-the-art results. Our decoding algorithm is obviously slower than the greedy search method of Coavoux and Crabbé [2017], however, it is way faster than the exact decoder of [van Cranenburgh et al., 2016]. These findings show the benefits of the GMSA formulation for NLP problems.

6.7 Conclusion and future work

In this chapter, we introduced a novel approach to discontinuous phrase-structure parsing based on STAGs, a variant of LTAG. We showed that this problem is equivalent to GMSA. We introduced a novel dual decomposition algorithm for this problem which computes the optimal solution for $\approx$ 99% of the test instances while being fast in practice. We evaluated our decoding algorithm on two datasets and achieve state of the art results.

[9] `https://github.com/andreasvc/disco-dop/`

[10] C2017 processing time is 137.338 seconds plus approximatively 30 seconds for model and corpus loading (personnal communication).

[11] Execution times are not directly comparable because we report our experimental conditions and published results.

	LR	LP	LF	Time
	Short sentences only			
This work	90.63	91.01	90.82	≈ 4
This work†	89.57	90.13	89.85	≈ 4
VC2016†			87.00	≈ 180
	Full test set			
This work	89.89	90.29	90.09	≈ 6.5
This work†	88.90	89.45	89.17	≈ 5.5

Table 6.1: Parsing results and processing time in minutes on the english discontinuous PTB corpus. Results marked with † use predicted part of speech tags. VC2016 indicates results of van Cranenburgh et al. [2016].

	LR	LP	LF	Time
	Short sentences only			
This work	82.69	84.68	83.67	≈ 7.5
	Full test set			
This work	80.66	82.63	81.63	≈ 11
C2017			81.60	≈ 2.5

Table 6.2: Parsing results and processing time in minutes on the german Tiger corpus. C2017 indicates results of Coavoux and Crabbé [2017].

Future work may extend our algorithm to include bi-gram weights on vertices and high-order weights on arcs. This can be achieved with slight modifications in the second subproblem. Moreover, note that graphs with weights given by machine learning techniques may have a distribution which happens to accelerate convergence [Martins et al., 2009b]. Further experiments should investigate the application of our algorithm with datasets from the optimization community and with symmetric graphs in order to solve the undirected version of the problem. Moreover, bounds given by our decomposition should be compared with the ones given by the algorithm of Myung et al. [1995]. Although the GMSA has not been studied a lot, future work may use the TSPLib dataset [Reinelt, 1991] which was used by previous authors to test algorithms for the Generalized (Assymetric) Travelling Salesman problem.

	UAS	Spine acc.
English	93.70	97.32
English†	93.04	96.81
German	92.25	96.49

Table 6.3: Dependency parsing and tagging results. Results marked with † use predicted part of speech tags.

Part IV

Conclusion

Chapter 7

Conclusion

Mildly context-sensitive grammars have been introduced in order to ensure both polynomial parsing time and broad linguistic coverage. In particular, the linguistic relevance of Tree Adjoining Grammars (TAGs) has been studied extensively since their introduction. Although polynomial, existing parsing algorithms in the literature are either experimentally intractable due to their $\mathcal{O}(n^7)$ complexity or prohibit wrapping adjunctions, and are therefore restricted to context-free languages. We proposed a novel approach to tackle this kind of challenging NLP problems. First, we reduced a given problem to a subgraph selection task: we reduced LTAG dependency parsing to the Yield Restricted Maximum Spanning Arborescence problem and discontinuous phrase-structure parsing to the Generalized Maximum Spanning Arborescence problem. This reduction led to a generic problem which may have application in other fields. Moreover, graph theory is appealing as it is a *lingua franca* among computer scientists. Second, we introduced an ILP formulation of the graph problem. As such, we could rely on many combinatorial optimization techniques. Third and finally, we applied Lagrangian relaxation to solve the ILPs. In this chapter, we summarize the contributions of this thesis and discuss possible extensions.

7.1 Summary of contributions

Lexicalized Tree Adjoining Grammars (LTAGs) are a variant of TAGs where each elementary tree has exactly one lexical anchor. As such, the derivation tree of a LTAG is a labeled bi-lexical dependency structure. Previous work in the literature characterized it as a well-nested arborescence with 2-bounded block degree. Following previous work in dependency parsing, we rely on a pipeline approach for this problem. First we parse the dependency structure and then we run a parse tree labeler. Interestingly, the computational complexity of LTAG parsing is broken down: the first step has a $\mathcal{O}(n^7)$ complexity, similar to LTAG but without the grammar constant, and the second step has a linear time complexity with respect to the input sentence length but takes into account the grammar. As the first step is the bottleneck of our approach, we introduced two ILPs formulations for decoding the maximum well-nested arborescence with 2-bounded block degree. In particular, we showed that one of them has an exponential number of constraints but an interesting and generalizable structure.

We proposed a decoding algorithm based on Lagrangian relaxation and showed its efficiency on several datasets.

The complexity of LTAG parsing is due to the intricate structure of the induced dependencies. We proposed a novel approach to discontinuous constituent parsing that follows our previously described LTAG parser but:

- We do not impose any constraint on the dependency structure aside from being an arborescence.
- We do not impose structural restriction on combination operations.
- We jointly decode the bi-lexical dependency structure and elementary trees assignment.

We showed that the induced derived tree can then include discontinuous constituents. Moreover, this problem is equivalent to the Generalized Maximum Spanning Arborescence. We proposed a novel decoding algorithm based on dual decomposition and achieved state-of-the-art results on two standard benchmarks.

7.2 Future work

In this section, we discuss possible future research directions.

7.2.1 Lexicalized Tree Adjoining Grammar parsing

In Chapter 5, we proposed a novel LTAG parsing algorithm based on a pipeline method. We tackled the complexity challenge using Lagrangian relaxation on a Integer Linear Programming formulation of the problem. We reported experimental results on the efficiency of our parser with different problems which have a similar structure to LTAG derivation trees, namely well-nested arborescences with k-bounded block degree. Unfortunately, we were not able to test the relevance of our algorithm for phrase-structure parsing, in particular regarding long distance relationships, as we do not have a LTAG annotated corpus. Villemonte de la Clergerie [2010] developed a Tree Adjoining Meta-Grammar which has a wide coverage of French. Moroever, Villemonte De La Clergerie [2013] showed that this symbolic grammar gets strong competitive parsing scores on the French Treebank [Abeillé et al., 2003]. As such, it may be used as the basis for the automatic extraction of a LTAG corpus in order to test our algorithm.[1]

7.2.2 Parsing other lexicalized grammars

Kuhlmann [2007] studied the relation between lexicalized grammars and induced bi-lexical dependencies. We summarize his results in Table 7.1. Lexicalized Linear Context-Free Rewriting Systems (LLCFRS) are particulary appealing as LCFRS extraction is straighforward from discontinuous treebanks [Maier and Søgaard, 2008, Kallmeyer, 2013] and head percolation rules are available

[1] At of the date of this thesis, we are discussing the feasibility of this approach with Eric Villemonte De La Clergerie.

Formalism	k-bounded block degree	Well-nested
Context-Free Grammar	✓	✓
Linear Context-Free Rewriting System	✓	
Coupled Context-Free Grammar	✓	✓
Tree Adjoining Grammar	✓	✓

Table 7.1: Properties of dependencies induced from the Lexicalized alternative of several formalisms

in order to perform lexicalization. Moreover, the general parsing problem is NP-complete if the grammar induces ill-nested dependencies. However, our dependency parser is able to efficiently parse this family of structures and can be used in a similar pipeline to the one we proposed for LTAGs.

7.2.3 Natural Language Processing applications of the Generalized Maximum Spanning Arborescence

The Generalized Maximum Spanning Arborescence (GMSA) problem is equivalent to the joint tagging and non-projective dependency parsing problem. A natural application is the case of part of speech (POS) tags, which was previously explored by Bohnet and Nivre [2012] with a transition system. Many POS taggers rely on bi-gram weights, that is the weight of two consecutive tags. This weighting model can be included in the decoding algorithm we proposed for the GMSA. Indeed, we can solve in polynomial time the second subproblem (Subsection 6.5.1) using a variant of the Viterbi algorithm [Forney, 1973], which includes bi-gram weights.

Regarding dependency parsing, high-order models were commonly considered as mandatory in order to obtain state-of-the-art dependency scores before the rise of neural networks. Pei et al. [2015] also observed a small improvement with high-order neural parameterization thanks to a specific activation function. Note that these weighting models may be highly relevant for small datasets. High-order weights can also be introduced in the second subproblem of our decoding algorithm using a technique similar to Koo et al. [2010].

7.2.4 Simple and accurate derivation tree parsing

Last but not least, we observe that dependency parsers have recently achieved extremely high performances [Dozat and Manning, 2017]. Thus, relying on a pipeline procedure similar to the one we propose for LTAG parsing may produce state-of-the-art results with a continuous and/or discontinuous spinal TAG grammar. Although simple, such an approach may be competitive with transition based parsers both with respect to parsing time and accuracy.

Bibliography

Carlos A. Prolo. Fast LR parsing using rich (tree adjoining) grammars. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

Anne Abeillé. A lexicalized tree adjoining grammar for french: The general framework. Technical report, University of Pennsylvania, 1988.

Anne Abeillé. *Les grammaires d'unification.* Hermès, 2007.

Anne Abeillé, Kathleen Bishop, Sharon Cote, and Yves Schabes. A lexicalized tree adjoining grammar for english. Technical report, University of Pennsylvania, 1990.

Anne Abeillé, Lionel Clément, and François Toussenel. Building a treebank for french. *Treebanks*, pages 165–187, 2003.

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. Floresta sintá(c)tica: A treebank for portuguese. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, 2002.

Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compilers, Principles, Techniques.* Addison wesley, 1986.

Miguel Almeida and Andre Martins. Fast and robust compressive summarization with dual decomposition and multi-task learning. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 196–206, 2013.

Michael Auli and Adam Lopez. A comparison of loopy belief propagation and dual decomposition for integrated ccg supertagging and parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 470–480. Association for Computational Linguistics, 2011.

John Beasley. *Modern heuristic techniques for combinatorial problems*, chapter Lagrangian relaxation, pages 243–303. John Wiley & Sons, Inc., 1993.

John E Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93, 1987.

John E Beasley. An sst-based algorithm for the steiner problem in graphs. *Networks*, 19(1):1–16, 1989.

Mandell Bellmore and George L Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16(3):538–558, 1968.

Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

Charles Eugene Blair and RG Jeroslow. The value function of a mixed integer program: II. *Discrete Mathematics*, 25(1):7–19, 1979.

FC Bock. An algorithm to construct a minimum directed spanning tree in a directed network. *Developments in Operations Research*, 1969.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möohl. Well-nested drawings as models of syntactic structure. In *Proceedings of the 10th Conference on Formal Grammar and Ninth Meeting on Mathematics of Language*, pages 195–203, Edinburgh, UK, 2005.

Bernd Bohnet and Joakim Nivre. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, 2012.

Guillaume Bonfante, Bruno Guillaume, Mathieu Morey, and Guy Perrier. Supertagging with Constraints. In Philippe Blache, Henning Christiansen, Verónica Dahl, Denys Duchier, and Jørgen Villadsen, editors, *Constraint and Language*. Cambridge Scholar Publishing, 2014.

Endre Boros and Peter L Hammer. Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1):155–225, 2002.

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

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. TIGER: Linguistic interpretation of a german corpus. *Research on language and computation*, 2(4):597–620, 2004.

Laurel J Brinton. *The structure of modern English: A linguistic introduction*. John Benjamins Publishing, 2000.

Harry Bunt, Jan Thesingh, and Ko van der Sloot. Discontinuous constituents in trees, rules, and parsing. In *Third Conference of the European Chapter of the Association for Computational Linguistics*, 1987.

Shu Cai, David Chiang, and Yoav Goldberg. Language-independent parsing with empty elements. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 212–216, 2011.

Paolo M Camerini, Luigi Fratta, and Francesco Maffioli. A note on finding optimum branchings. *Networks*, 9(4):309–312, 1979.

Xavier Carreras, Michael Collins, and Terry Koo. *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, chapter TAG, Dynamic Programming, and the Perceptron for Efficient, Feature-Rich Parsing, pages 9–16. Coling 2008 Organizing Committee, 2008.

Arthur Cayley. A theorem on trees. *Quartery Journal of Mathematics*, 23: 376–378, 1889.

Sebastián Ceria, Paolo Nobili, and Antonio Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2): 215–228, 1998.

David Chiang. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, 2000.

Noam Chomsky. *Syntactic structures*. The Hague: Mouton, 1957.

Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137 – 167, 1959.

Nicos Christofides and S Korman. Note - a computational survey of methods for the set covering problem. *Management Science*, 21(5):591–599, 1975.

Yoeng-Jin Chu and Tseng-Hong Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396, 1965.

François Clautiaux, Ruslan Sadykov, François Vanderbeck, and Quentin Viaud. Combining dynamic programming with filtering to solve a four-stage two-dimensional guillotine-cut bounded knapsack problem. Unpublished, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *Proceedings of the 2016 International Conference on Learning Representations*, 2016.

Maximin Coavoux and Benoit Crabbé. Incremental discontinuous phrase structure parsing with the GAP transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, 2017.

John Cocke. *Programming languages and their compilers: Preliminary notes*. Courant Institute Mathematical Sciences, 1970.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Madrid, Spain, July 1997.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms, 2002.

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

William Cook, Albertus MH Gerards, Alexander Schrijver, and Éva Tardos. Sensitivity theorems in integer linear programming. *Mathematical Programming*, 34(3):251–264, 1986.

Dipanjan Das, André FT Martins, and Noah A Smith. An exact dual decomposition algorithm for shallow semantic parsing with constraints. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*, pages 209–217, 2012.

Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, pages 439–485, 2005.

Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. *Proceedings of the 2017 International Conference on Learning Representations*, 2017.

Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71(4):233–240, 1967.

Jason Eisner. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in probabilistic and other parsing technologies*, pages 29–61. Springer, 2000.

Jason Eisner and Giorgio Satta. A faster parsing algorithm for lexicalized tree-adjoining grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks*, pages 79–84, 2000.

Kilian Evang and Laura Kallmeyer. PLCFRS parsing of english discontinuous constituents. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 104–116, Dublin, Ireland, October 2011.

Corinne Feremans, Martine Labbé, and Gilbert Laporte. The generalized minimum spanning tree: Polyhedra and branch-and-cut. *Electronic Notes in Discrete Mathematics*, 3:45–50, 1999.

Corinne Feremans, Martine Labbé, and Gilbert Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39(1):29–34, 2002.

Daniel Fernández-González and T. André F. Martins. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, 2015.

Thomas Finley and Thorsten Joachims. Training structural svms when exact inference is intractable. In *Proceedings of the 25th international conference on Machine learning*, pages 304–311. ACM, 2008.

Marshall L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27, 1981.

G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Claire Gardent and Laura Kallmeyer. Semantic construction in f-tag. In *10th Conference of the European Chapter of the Association for Computational Linguistics*, 2003. URL `http://www.aclweb.org/anthology/E03-1030`.

Claire Gardent and Shashi Narayan. Multiple adjunction in feature-based tree-adjoining grammar. *Computational Linguistics*, 41(1):41–70, 2015.

Krasimira Genova and Vassil Guliashki. Linear integer programming methods and approaches–a survey. *Journal of Cybernetics and Information Technologies*, 11(1), 2011.

Arthur M Geoffrion. Lagrangean relaxation for integer programming. *Approaches to integer programming*, pages 82–114, 1974.

Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.

Carlos Gómez-Rodríguez, David Weir, and John Carroll. Parsing mildly non-projective dependency structures. In *Proceedings of the 12th Conference of the European Chapter of the ACL*, pages 291–299, 2009.

Carlos Gómez-Rodríguez, Marco Kuhlmann, and Giorgio Satta. Efficient parsing of well-nested linear context-free rewriting systems. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 276–284, 2010.

Carlos Gómez-Rodríguez, John Carroll, and David Weir. Dependency parsing schemata and mildly non-projective dependency parsing. *Computational Linguistics*, 37(3), 2011.

Joshua Goodman. Semiring parsing. *Computational Linguistics, Volume 25, Number 4, December 1999*, 1999.

Eduard Nikolaevich Gordeev and OG Tarastsov. The steiner problem: A survey. *Discrete Mathematics and Applications*, 3(4):339–364, 1993.

Monique Guignard. Lagrangean relaxation. *Top*, 11(2):151–200, 2003.

Jirí Havelka. Relationship between non-projective edges, their level types, and well-nestedness. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 61–64, 2007.

Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

Michael Held and Richard M Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical programming*, 1(1):6–25, 1971.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Richard Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for english. In *16th Nordic Conference of Computational Linguistics*, pages 105–112, 2007.

Mark Johnson. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

Aravind K Joshi and Yves Schabes. Tree-adjoining grammars. *Handbook of formal languages*, 3:69–124, 1997.

Aravind K. Joshi and B. Srinivas. Disambiguation of super parts of speech (or supertags): Allmost parsing. In *The 15th International Conference on Computational Linguistics*, 1994.

Aravind Krishna Joshi. Tree adjoining grammars: how much context-sensitivity is required to provide reasonable structural descriptions? In David R. Dowty, Lauri Karttunen, and Arnold Zwicky, editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, 1985.

Aravind Krishna Joshi. *An introduction to tree adjoining grammars.* John Benjamins, mathematics of language edition, 1987.

Laura Kallmeyer. *Parsing beyond context-free grammars.* Springer Science & Business Media, 2010.

Laura Kallmeyer. Linear context-free rewriting systems. *Language and Linguistics Compass*, 7(1):22–38, 2013.

Laura Kallmeyer and Marco Kuhlmann. A formal model for plausible dependencies in lexicalized tree adjoining grammar. In *Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 108–116, 2012.

Laura Kallmeyer and Giorgio Satta. A polynomial-time parsing algorithm for tt-mctag. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 994–1002, 2009.

Laura Kallmeyer, Timm Lichte, Wolfgang Maier, Yannick Parmentier, Johannes Dellert, and Kilian Evang. TuLiPA: Towards a multi-formalism parsing environment for grammar engineering. In *Proceedings of the workshop on Grammar Engineering Across Frameworks*, pages 1–8, Manchester, England, 2008.

Ronald M Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. *Formal Issues in Lexical-Functional Grammar*, pages 29–130, 1982.

Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

Jungo Kasai, Bob Frank, Tom McCoy, Owen Rambow, and Alexis Nasr. TAG parsing with neural networks and vector representations of supertags. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1713–1723, 2017.

Tadao Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report, University of Illinois, 1965.

Martin Kay. *Readings in Natural Language Processing*, chapter Algorithm Schemata and Data Structures in Syntactic Processing, pages 35–70. Morgan Kaufmann Publishers Inc., 1986.

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

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of The International Conference on Learning Representations*, 2015.

Nikos Komodakis. Efficient training for pairwise or higher order crfs via dual decomposition. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1841–1848. IEEE, 2011.

Lingpeng Kong, M. Alexander Rush, and A. Noah Smith. Transforming dependencies into phrase structures. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 788–798, 2015.

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, 2010.

Anthony S Kroch and Beatrice Santorini. *The derived constituent structure of the West Germanic verb raising construction.* University of Pennsylvania, School of Engineering and Applied Science, Department of Computer and Information Science, 1987.

Marco Kuhlmann. *Dependency Structures and Lexicalized Grammars.* PhD thesis, Saarland University, 2007.

Marco Kuhlmann and Peter Jonsson. Parsing to noncrossing dependency graphs. *Transactions of the Association of Computational Linguistics*, 3:559–570, 2015.

Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514, 2006.

Jonathan K. Kummerfeld and Dan Klein. Parsing with traces: An o(n4) algorithm and a structural representation. In *Transactions of the Association for Computational Linguistics*, 2017.

Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

Joseph Le Roux, Antoine Rozenknop, and Jennifer Foster. Combining PCFG-LA Models with Dual Decomposition: A Case Study with Function Labels and Binarization. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, October 2013.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Claude Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697, 2017.

Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160, 1976.

Abilio Lucena. Non delayed relax-and-cut algorithms. *Annals of Operations Research*, 140(1):375–410, 2005.

Abilio Lucena. Lagrangian relax-and-cut algorithms. In *Handbook of Optimization in Telecommunications*, pages 129–145. Springer, 2006.

Xuezhe Ma and Eduard Hovy. Neural probabilistic model for non-projective mst parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 59–69. Asian Federation of Natural Language Processing, 2017. URL `http://aclweb.org/anthology/I17-1007`.

Thomas L Magnanti and Laurence A Wolsey. Optimal trees. *Handbooks in operations research and management science*, 7:503–615, 1995.

Wolfgang Maier. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, 2015.

Wolfgang Maier and Anders Søgaard. Treebanks and mild context-sensitivity. In *Proceedings of Formal Grammar*, page 61, 2008.

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.

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

Andre Martins, Noah Smith, Eric Xing, Pedro Aguiar, and Mario 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, 2010.

André Filipe Torres Martins. *The geometry of constrained structured prediction: applications to inference and learning of natural language syntax*. PhD thesis, Carnegie Mellon University, 2012.

André FT Martins, Noah A Smith, and Eric P Xing. Concise integer linear programming formulations for dependency parsing. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 342–350, 2009a.

André FT Martins, Noah A Smith, and Eric P Xing. Polyhedral outer approximations with application to natural language parsing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 713–720, 2009b.

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, 2007.

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, 2005.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. Universal dependency annotation for multilingual parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, 2013.

Igor Aleksandrovič Melčuk. *Dependency syntax: theory and practice*. Suny Press, 1988.

Mathias Möhl. *Drawings as models of syntactic structure: Theory and algorithms*. PhD thesis, Saarland University, 2006.

Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

Mark-Jan Nederhof. An alternative LR algorithm for TAGs. In *Proceedings of the 17th international conference on Computational linguistics-Volume 2*, pages 946–952, 1998.

Rebecca Nesson, Alexander Rush, and Stuart Shieber. Induction of probabilistic synchronous tree-insertion grammars for machine translation. Association for Machine Translation in the Americas, 2006.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

Wenzhe Pei, Tao Ge, and Baobao Chang. An effective neural network model for graph-based dependency parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 313–322, 2015.

Fernando C. N. Pereira and David H. D. Warren. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, 1983.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. Dynamic programming for higher order parsing of gap-minding trees. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 478–488, 2012.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. Finding optimal 1-endpoint-crossing trees. *Transactions of the Association of Computational Linguistics*, 1:13–24, 2013.

Boris T Polyak. Introduction to optimization. *Optimization Software*, 1987.

Petrica C. Pop. A survey of different integer programming formulations of the generalized minimum spanning tree problem. *Carpathian Journal of Mathematics*, 25(1):104–118, 2009.

Xian Qian and Yang Liu. Branch and bound algorithm for dependency parsing with non-local features. *Transactions of the Association of Computational Linguistics*, 1:37–48, 2013.

Owen Rambow. The simple truth about dependency and phrase structure representations: An opinion piece. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 337–340, 2010.

Owen Rambow and Aravind Joshi. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. *Recent trends in meaning-text theory*, 39:167–190, 1997.

Gerhard Reinelt. TSPLIB - a traveling salesman problem library. *ORSA journal on computing*, 3(4):376–384, 1991.

Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137, 2006.

Sebastian Riedel, David Smith, and Andrew McCallum. Parse, price and cut: delayed column and row generation for graph based parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 732–743, 2012.

Alexander M Rush and Michael Collins. A tutorial on dual decomposition and lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 2012.

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, 2010.

Giorgio Satta. Recognition of linear context-free rewriting systems. In *30th Annual Meeting of the Association for Computational Linguistics*, 1992.

Giorgio Satta. Tree-adjoining grammar parsing and boolean matrix multiplication. *Computational Linguistics*, 20(2), 1994.

Giorgio Satta and Marco Kuhlmann. Efficient parsing for head-split dependency trees. *Transactions of the Association for Computational Linguistics*, 1:267–278, 2014.

Yves Schabes and Richard C. Waters. Tree insertion grammar: A cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics, Volume 21, Number 4, December 1995*, 1995.

Yves Schabes and Aravjnd K. Joshi. An earley-type parsing algorithm for tree adjoining grammars. In *26th Annual Meeting of the Association for Computational Linguistics*, 1988.

Yves Schabes and Stuart M. Shieber. An alternative conception of tree-adjoining derivation. In *30th Annual Meeting of the Association for Computational Linguistics*, 1992.

Yves Schabes, Anne Abeillé, and Aravind Joshi. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*, 1988.

Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency.* Springer, 2003.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, D. Jinho Choi, Richárd Farkas, Jennifer Foster, Iakes Goenaga, Koldo Gojenola Galletebeitia, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiórkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Villemonte Eric de la Clergerie. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, 2013.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages, 2014.

Wolfgang Seeker and Jonas Kuhn. Making ellipses explicit in dependency conversion for a german treebank. In *LREC*, pages 3132–3139, 2012.

Libin Shen. *Statistical LTAG Parsing.* PhD thesis, University of Pennsylvania, 2006.

Libin Shen and Aravind K Joshi. Incremental ltag parsing. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 811–818, 2005.

Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24 (1):3–36, 1995.

Naum Zuselevich Shor. *Minimization methods for non-differentiable functions*. Springer-Verlag, 1985.

Noah A Smith. Linguistic structure prediction. *Synthesis lectures on human language technologies*, 4(2):1–274, 2011.

David Sontag, Amir Globerson, and Tommi Jaakkola. *Optimization in Machine Learning*, chapter Introduction to Dual Decomposition for Inference. MIT Press, 2010.

Robert Endre Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

Lucien Tesnière. *Les éléments de Syntaxe structurale*. Editions Klincksieck, 1959.

David Vadas and James Curran. Adding noun phrase structure to the penn treebank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 240–247, 2007.

Leslie G Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.

Andreas van Cranenburgh, Remko Scha, and Rens Bod. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111, 2016.

Leonoor Van der Beek, Gosse Bouma, Rob Malouf, and Gertjan Van Noord. The alpino dependency treebank. *Language and Computers*, 45(1):8–22, 2002.

Éric Villemonte de la Clergerie. Building factorized TAGs with meta-grammars. In *The 10th International Conference on Tree Adjoining Grammars and Related Formalisms*, 2010.

Éric Villemonte De La Clergerie. Improving a symbolic parser through partially supervised learning. In *The 13th International Conference on Parsing Technologies*, Naria, Japan, 2013.

David Jeremy Weir. Characterizing mildly context-sensitive grammar formalisms. Unpublished, 1988.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206, 2003.

Daniel H Younger. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208, 1967.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, 2013.

Appendix A

Penn Treebank tag set

Figure A.1 and Figure A.2 report the set of constituency labels and part-of-speech tags used in the Penn Treebank [Marcus et al., 1993], respectively.

S	Simple declarative clause
SBAR	Clause introduced by a subordinating conjunction
SBARQ	Direct question introduced by a wh-word or a wh-phrase
SINV	Inverted declarative sentence
SQ	Inverted yes/no question or main clause of a wh-question
ADJP	Adjective phrase
ADVP	Adverb phrase
CONJP	Conjunction phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker
NAC	Not a Constituent
NP	Noun phrase
PP	Prepositional phrase
PRN	Parenthetical
PRT	Particle
QP	Quantifier phrase
RRC	Reduced relative clause
UCP	Unlike coordinated phrase
VP	Vereb phrase
WHADJP	Wh-adjective phrase
WHAVP	Wh-adverb phrase
WHNP	Wh-noun phrase
WHPP	Wh-prepositional phrase
X	Unknown, uncertain, or unbracketable

Table A.1: Constituency labels

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential there
FW	Foreign word
IN	Preposition or subordinating conjunction
JJ	Adjective
JJR	Adjective, comparative
JJS	Adjective, superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol
TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Table A.2: Part-of-speech tags

Appendix B

Maximum Spanning Arborescence

The Maximum Spanning Arborescence[1] (MSA) algorithm has been independently proposed by Edmonds [1967], Chu and Liu [1965], Bock [1969]. The best known worst-case complexity with a complete graph, $\mathcal{O}(n^2)$ with n is the number of vertices, is due to Tarjan [1977]. However, the algorithm in the original publication contains errors which were corrected by Camerini et al. [1979].[2] A discussion of the application of the MSA for dependency parsing alongside examples can be found in McDonald et al. [2005].

We now give a simple explanation of the algorithm. Let $G = (V, A)$ be a directed graph with vertices $V = \{v_0 \dots v_n\}$, arcs $A = V \times V$ and let $w \in \mathbb{R}^A$ be a vector of weights indexed by arcs. Without loss of generality, we assume v_0 is the desired root. Edmonds [1967] proved that, given a subset of vertices $W \subseteq V \setminus \{v_0\}$, if the set of arcs composed of the arc of maximum weight for each vertex $v \in W$ defines a cycle, then the MSA of the graph contains every arc of this cycle except one. Thus, the v_0-rooted MSA can be computed as follows. First, construct arc subset T by adding the best incoming arc for each vertex $v_1 \dots v_n$. If the resulting graph is an arborescence, then T is the MSA. Else, find any cycle in T, contract the cycle and repeat the procedure. This contraction aims at finding which arc in the cycle is the single one that should be replaced. Finally, cycles can be decontracted in order to rebuild the original graph. Note that the arc weights must be updated after contraction. We report the reader to the original publications for a detailed explanation of the algorithm. The pseudo-code is described in Algorithm 4. Because you can make at most n contractions and selecting the best incoming arcs is a $\mathcal{O}(n^2)$ task,[3] the whole procedure has a cubic time complexity. However, Tarjan [1977]

[1] This algorithm is often presented as solving the Minimum Spanning Arborescence problem. We prefer the maximization formulation in order to be more consistent with the NLP literature.

[2] Some of these algorithms focus on an equivalent problem called optimum branching.

[3] A contracted graph has at most n vertices. A single vertex has at most n incoming arcs and selecting the incoming arc for a single vertex is a linear time procedure without ordered data structures. Thus, selecting the best incoming arc for each vertex of a contracted graph is a $\mathcal{O}(n^2)$ problem.

proposed a more efficient algorithm with a $\mathcal{O}(n^2)$ time complexity, relying on a set of priority queues for efficient max and union operations.

Algorithm 4 Pseudo-code for the MSA

function MSA(G, V, w)
  $T \leftarrow \emptyset$ ▷ Output variable (set of arcs)
  for $v \in V^+$ **do**
    ▷ Select the best incoming arc for each vertex in V^+
    $T \leftarrow T \cup \{\text{argmax}_{a \in \delta^{in}(v)} w_a\}$
  if T is acyclic **then**
    return T
  $C \leftarrow$ Any subset of arcs forming a cycle in T
  $V', A', w' \leftarrow$ CONTRACT(V, A, w, C)
  $T \leftarrow$ MSA(V', A', w')
  ▷ a is an arc of the cycle which has been replaced in the arborescence
  $a \in \{v_i \rightarrow v_j \in C | \exists v_k : v_k \rightarrow v_j \in T\}$
  return $T \cup C \setminus \{a\}$

function CONTRACT(V, A, w, C)
  ▷ Any vertex identifier that can be used to represent the contracted cycle
  $v_c \in \{v_c | v_c \notin V\}$
  for $v_i \in V \setminus C$ **do**
    if $\exists v_j \in C : v_j \rightarrow v_i \in A$ **then**
      ▷ Select the best incoming arc for each vertex in the cycle
      $A \leftarrow A \cup \{v_c \rightarrow v_i\}$
      $w_{v_c \rightarrow v_i} \leftarrow \max\{w_{v_j \rightarrow v_i} | v_j \in C\}$
  for $v_i \in V \setminus C$ **do**
    if $\exists v_j \in C : v_i \rightarrow v_j \in A$ **then**
    ▷ Construct outgoing arcs for the new vertex with updated weights
      $A \leftarrow A \cup \{v_i \rightarrow v_c\}$
      $w_{v_i \rightarrow v_c} \leftarrow \max\{w_{v_i \rightarrow v_j} - w_{v_v \rightarrow v_j} + \sum_{a \in C} w_a | v_j \in C \wedge v_k \rightarrow v_j \in C\}$
  $V \leftarrow V \setminus C \cup \{v_c\}$
  $A \leftarrow \{v_i \rightarrow v_j \in A | v_i \in V \wedge v_j \in V\}$
  return V, A, w

Appendix C

Subproblems for the well-nested arborescence with k-bounded block degree

In this appendix, we give the pseudo-code for the two algorithms used for computing the maximum well-nested arborescence with k-bounded block degree in Chapter 5. The inputs are a set of vertices V and a v_0-rooted spanning arborescence T. The first one computes the block degree of an arborescence and the second returns true if a given arborescence is well-nested. In practice, in the black box for our algorithm solving the YRMSA, we must return sets of yields. This modification is straighforward.

C.1 Gap degree of an arborescence

The function COMPUTE-YIELD in Algorithm 5 computes the yield of the vertex v_h with respect to arborescence T. The results is stored in the input vector y: if vertex v_i is in the yield of v_h, then $y[i]$ is set to $True$. Let n be the number of vertices adjacent to T. Then, this algorithm as a $\mathcal{O}(n)$ linear time complexity. Indeed, the recursive call to COMPUTE-YIELD can be only be done n times as T does not contain any cycle.[1]

Algorithm 5 Algorithm which computes the yield of a given vertex.

```
1: function COMPUTE-YIELD(T, v_h, y)
2:     y[v_h] = True
3:     for v_h ← v_m ∈ δ^out(v_h) ∩ T do
4:         COMPUTE-YIELD(T, v_m, y)
```

[1] We assume that $\delta^{\text{out}}(v_h) \cap T$ is a constant time operation. This can be realized by storing T in a convenient data structure.

The function COMPUTE-BLOCK-DEGREE in Algorithm 6 computes the block degree of an spanning arborescence T. For each vertex, we compute its yield and then count the number of vertices that are at the beginning of a block. The block degree the arborescence is the maximum block degree of its vertices. This algorithm has a $\mathcal{O}(n^2)$ time complexity, that is quadratic with respect to the number of vertices.

Algorithm 6 Algorithm which computes the block degree of a spanning arborescence.

```
1:  function COMPUTE-BLOCK-DEGREE(V, T)
2:      max_bd ← 0
3:      for v_i ∈ V do
4:          y ← Vector of size |V| filled with False
5:          COMPUTE-YIELD(T, v_i, y)
6:          if y[0] = True then                    ▷ Does v_0 start a block?
7:              bd ← 1
8:          else
9:              bd ← 0
10:         for v_j ∈ V \ {v_0} do
11:             if y[j − 1] = False ∧ y[j] = True then  ▷ Does v_j start a block?
12:                 bd ← bd + 1
                                  ▷ We return the maximum between all vertices
13:         max_bd ← max(max_bd, bd)
14:     return max_bd
```

C.2 Well-nestedness of an arborescence

In order to test if a spanning arborescence is well-nested, we compute the ill-nested set of each one of its arc (Definition 5.8). If every set is empty, the arborescence is well-nested. The algorithm has two nested for-loops on arcs in the arborescence. A spanning arborescence contains exactly $n - 1$ arcs so the time complexity of the algorithm is $\mathcal{O}(n^2)$. This algorithm was first proposed by Havelka [2007].

Algorithm 7 This algorithm returns true if and only if an arborescence is well-nested.

```
1:  function IS-WELL-NESTED(V, T)
2:      y ← Matrix of size |V| × |V| filled with False
3:      for v_i ∈ V do
4:          COMPUTE-YIELD(T, v_i, y[i])

5:      for v_i → v_j ∈ T do
6:          for v_k → v_l ∈ T \ {v_i → v_j} do
7:                                  ▷ Test if v_k → v_l is in the ill-nested set of v_i → v_j
8:              if min(i, j) < k < max(i, j) ∧ ¬(min(i, j) < l < max(i, j)) then
9:                  if ¬(y[i][l] ∨ y[j][l] ∨ y[k][i] ∨ y[l][i]) then
10:                     return False
11:             if min(i, j) < l < max(i, j) ∧ ¬(min(i, j) < k < max(i, j)) then
12:                 if ¬(y[i][l] ∨ y[j][l] ∨ y[k][i] ∨ y[l][i]) then
13:                     return False
                                        ▷ All sets where empty, so we return True
14:     return True
```

Appendix D

Subproblems for the GMSA

In this appendix, we give the pseudo-code of the two algorithms for computing the dual objective of the Lagrangian relaxtion of the GMSA in Section 6.5.

D.1 Subproblem 1

The first subproblem compute the Maximum Spanning Arborescence (MSA) over clusters. Given a graph (V, A) with arc weight w and clusters π, we build a graph where the vertices are clusters and arcs are the highest scoring arcs between clusters. Then, we run an implementation of the MSA algorithm on this graph. The algorithm is given in Algorithm 8.

Algorithm 8 Function which computes the MSA over clusters.

function SUBPROBLEM1(V, A, w, π)
  $A' \leftarrow \{\}$ ▷ Arcs of the contracted graph
  $w' \leftarrow []$ ▷ Arc weights in the contracted graph
  ▷ Construct the contracted graph
  for $v_h \rightarrow v_m \in A$ **do**
    $i \leftarrow \pi^{-1}(v_h)$ ▷ $\pi^{-1}(v_h)$ returns the cluster index of v_h
    $j \leftarrow \pi^{-1}(v_m)$
    if $v_i \rightarrow v_j \notin A'$ **then**
      $A' \leftarrow A' \cup v_i \rightarrow v_j$
    if $v_i \rightarrow v_j \notin w' \vee w[v_i \rightarrow v_j] > w'[v_i \rightarrow v_j]$ **then**
      $w'[v_i \rightarrow v_j] \leftarrow w[v_i \rightarrow v_j]$
  return MSA(π, A', w')

D.2 Subproblem 2

The second subproblem computes local cluster solutions without respecting the arborescence constraint. The algorithm is given in Algorithm 9.

Algorithm 9 Function which computes the solution of the second subproblem.

```
1:  function SUBPROBLEM2(V, A, w^in, w^out, π)
2:      score ← 0
3:      for V_k ← π \ {V_0} do
4:          score ← score+ GET-BEST-IN-CLUSTER(V_k, w^in, w^out, π)
5:      return score

6:  function GET-BEST-IN-CLUSTER(V, A, w^in, w^out, π)
7:      score = −∞
8:      for v ← V do
9:          score ← max(score, COMPUTE-VERTEX-SCORE(v, A, w^in, w^out, π) )
10:     return score

11: function COMPUTE-VERTEX-SCORE(v, A, w^in, w^out, π)
12:     score_head ← −∞
13:     for a ∈ δ^−(v) do
14:         score_head = max(score_head, w_a^in)

15:     score_modifiers ← 0
16:     for a ∈ δ^+(v) do
17:         if w_a^out > 0 then
18:             score_modifiers ← score_modifiers + w_a^out

19:     return score_head + score_modifiers
```

Appendix E

Probabilistic model

In this appendix, we describe the neural network which estimates the conditional probabilities for the experiments in Chapter 6. Under our parsing model, a parse tree is composed of a set of head attachments $\boldsymbol{h}$ (head words), a set of spine assignments $\boldsymbol{t}$ (supertags), a set of attachment positions $\boldsymbol{i}$ (non-terminal positions) and a set of attachment operations $\boldsymbol{a}$ (sister or regular adjunctions). Given an input sentence s, the probability of a derivation is given by $P(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{i}, \boldsymbol{a}|s)$. By the chain rule, we have:

$$P(\boldsymbol{h}, \boldsymbol{t}, \boldsymbol{i}, \boldsymbol{a}|s) = P(\boldsymbol{h}|s) \times P(\boldsymbol{t}|s, \boldsymbol{h}) \times P(\boldsymbol{i}, \boldsymbol{a}|s, \boldsymbol{h}, \boldsymbol{t})$$

We assume independence between each adjunction operation and that the propability of a spine for a given word depends on its head only. Thus, we decompose the score of an arc as the sum of three distinct log-probabilities:

- $\log P_\alpha$ which is the log-probability of a given word to be attached to a given head;
- $\log P_\nu$ which is the log-probability of the modifier spine represented by the destination vertex of the arc;
- $\log P_\gamma$ which is the log-probability of the label, that is of the position of the adjunction in the head spine and the adjunction type.

The three distributions P_α, P_ν and P_γ can be learned independently. Therefore the training algorithm does not need to compute the GMSA for each instance, which would be a time-consuming task. Note that a richer model could also include the probability of the head spine. In practice, arc labeling (i.e. assigning non-terminal positions and operation types) is realized as a post-processing step, which is a common approach in dependency parsing.

Given a input sentence $s = s_1 \dots s_n$, we first compute context-sensitive word embeddings with a two-stack Bi-LSTM [Hochreiter and Schmidhuber, 1997]. Note that these context-sensitive embedding are shared between the three distributions. The resulting embeddings b_k are then passed through specific feed-forward networks depending on the distribution and role. The result of the feed-forward transformation parameterized by set of parameters ρ of a word

embedding b_s is a vector denoted $b_s^{(\rho)}$. We first define a biaffine attention network weighting dependency relations [Dozat and Manning, 2017]:

$$o_{h,m}^{(\alpha)} = b_m^{(\alpha_1)\top} W^{(\alpha)} b_h^{(\alpha_2)} + V^{(\alpha)} b_h^{(\alpha_2)}$$

where $W^{(\alpha)}$ and $V^{(\alpha)}$ are trainable parameters, m a given modifier index and h a given head candidate index. Moreover, we define a biaffine attention classifier networks for class c as:

$$\begin{aligned} o_{c,h,m}^{(\tau)} &= b_m^{(\tau_1)\top} W^{(\tau_c)} b_h^{(\tau_2)} \\ &+ V^{(\tau_c)} \left(b_m^{(\tau_1)} \oplus b_h^{(\tau_2)} \right) \\ &+ u^{(\tau_c)} \end{aligned}$$

where $\oplus$ is the concatenation. $W^{(\tau_c)}$, $V^{(\tau_c)}$ and $u^{(\tau_c)}$ are trainable parameters. Then, we define the weight of assigning spine t to word at position m with head h as $o_{s,h,m}^{(\nu)}$.

Distributions of dependencies P_α and spines P_ν are parameterized by these biaffine attention networks followed by a softmax layer:

$$\begin{aligned} P_\alpha(h|m, \mathbf{w}) &= \frac{\exp o_{h,m}^{(\alpha)}}{\sum_{h'} \exp o_{h',m}^{(\alpha)}} \\ P_\nu(s|h, m, \mathbf{w}) &= \frac{\exp o_{s,h,m}^{(\nu)}}{\sum_{s'} \exp o_{s',h,m}^{(\nu)}} \end{aligned}$$

Now we move on to the post-processing step predicting arc labels. For each adjunction of spine s at position m to spine t at position h, instead of predicting a site index i, we predict the non-terminal nt at $t[i]$ with a biaffine attention classifier.[1] The probability of the adjunction of spine s at position m to a site labeled with nt on spine t at position h with type $a \in \{\text{regular}, \text{sister}\}$ is:

$$\begin{aligned} P_\gamma(nt, a|h, m) &= P_{\gamma'}(nt|h, m, \mathbf{w}) \\ &\times P_{\gamma''}(a|h, m\mathbf{w}) \end{aligned}$$

P_γ and $P_{\gamma''}$ are again defined as distributions from the exponential family using biaffine attention classifiers:

$$\begin{aligned} P_{\gamma'}(nt|h, m, t) &= \frac{\exp o_{nt,h,m}^{(\gamma')}}{\sum_{nt'} \exp o_{nt,h,m}^{(\gamma')}} \\ P_{\gamma''}(a|h, m, t) &= \frac{\exp o_{t,h,m}^{(\gamma'')}}{\sum_{a'} \exp o_{a',h,m}^{(\gamma'')}} \end{aligned}$$

[1] If a spine contains repeated non-terminal sequences, we select the lowest match.

We use embeddings of size 100 for words and size 50 for parts-of-speech tags. We stack two bidirectional LSTMs with a hidden layer of size 300, resulting in a context sensitive embedding of size 600. All feed-forward networks have a unique elu-activated hidden layer of size 100 [Clevert et al., 2016]. We regularize parameters with a dropout ratio of 0.5 on LSTM inputs. We estimate parameters by maximizing the likelihood of the training data through stochastic subgradient descent using Adam [Kingma and Ba, 2015]. Our implementation uses the Dynet library [Neubig et al., 2017] with default parameters.