Optimal Search Algorithms for Structured Problems in Natural Language Processing

by

Adam David Pauls

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy
in

Computer Science

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Dan Klein, Chair
Professor Stuart Russell
Professor Tom Griffiths

Fall 2012

Optimal Search Algorithms for Structured Problems in Natural Language Processing

Abstract

Optimal Search Algorithms for Structured Problems in Natural Language Processing

by

Adam David Pauls

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Dan Klein, Chair

Many tasks in Natural Language Processing (NLP) can be formulated as the assignment of a label to an input. Often, the set of possible labels can be unmanageably large and even enumerating the set would be intractable. For example, the set of possible labels for machine translation might be the set of all sentences in the output language, which can in principle be infinite. Fortunately, these large label sets typically exhibit some kind of structure – that is, they are composed of smaller parts. Because they are composed of parts, it is possible to construct labels piece by piece. The problem of predicting such structured labels is referred to as **structured prediction**, and the process of incrementally constructing the best structured label(s) is called **search**.

In this thesis, we consider problems for which it is feasible to perform exact or optimal search – that is, search that is guaranteed to find the best possible label according to some model. Many such problems can be formulated as a search for the best path in a weighted directed hypergraph. For example, monolingual parsing, bitext parsing, and syntactic machine translation can all be formulated in this way. We will discuss both known and novel algorithms that can find the best path without considering all hyperedges in the hypergraph, and hence can speed up search without sacrificing search quality. We will provide simplified proofs of correctness for these algorithms. We also propose two novel algorithms that permit extraction of the k-best paths instead of the single best. We compare these approaches both against exhaustive search, and against approximate search techniques which speed up search by sacrificing optimality guarantees.

Acknowledgements

I sometimes find it difficult to imagine what graduate school would have been like without the people in my life who made it the experience it was. I am truly blessed to have been part of a talented research group, at a prestigious school, headed by a brilliant adviser, all while surrounded by caring and encouraging friends and family.

My adviser, Dan Klein, has been an incredible mentor. He was an excellent research role model, and through him I have learned much about the art of quickly formulating, assessing, and presenting (or rejecting) research ideas. He is also one of the smartest people I have ever met, but then again, I have lost count of the number of one-of-the-smartest-people I have met during my graduate career. What truly sets Dan apart is his unwavering dedication to and passion for the things that he does. He cares deeply and genuinely about his students, both in his classes and in his research group, and displays a level of commitment to his teaching and advising that I am convinced requires at least 23 waking hours per day to maintain.

Dan's advising prowess also has the fortunate side effect of attracting top-notch students, so that some of the best collaborators in the world were just a few desks away. John DeNero, Aria Haghighi, Alexandre Bouchard-Côté, Slav Petrov, Percy Liang, David Burkett, Mohit Bansal, John Blitzer, Dave Golland, Taylor Berg-Kirkpatrick, Greg Durrett and Jonathan Kummerfeld were all incredibly (and humblingly) smart, fun, and talented people. I consider myself fortunate to have the opportunity to continue working with some of them when I join Google Research.

I also had the fortune of working with great researchers at other institutions. I learned a lot about machine translation – and even wrote a handout or two – during my internship with Kevin Knight and David Chiang at ISI. There, I also met USC students Jon May, Dirk Hovy, Victoria Fossum, Jason Riesa, Sujith Ravi, Ashish Vaswani, Zornitsa Kozareva, Oana Nicolov, Ulf Hermjakob and fellow interns Michael Auli, Paramveer Dhillon, and Erica Greene, who helped make the summer fun both inside and outside the lab. I had the chance to witness translation in the real world at Google working with Wolfgang Macherey, Franz Och, and a list of interns and employees too long to remember. I collaborated remotely with Chris Quirk and Microsoft Research.

A satisfying life outside of work can be just as important to one's success in a PhD. The many current, former, and honorary members of my beloved Woolsey house were all great friends and made for an enjoyable place to come home to. The Wednesday night dinner crowd provided highly intelligent humor, as well as much needed culinary variety to my otherwise burrito- and sandwich-heavy diet. Richard Liang and Jacob Becklund were always there if I just wanted to eat some ham and play video games. My friends back home in Canada were surprisingly good at making me feel like I had never left when I visited; Jon Woodward, Tim Louman-Gardiner, Reka Pataky, and Sarah Ramey in particular provided me with a place to stay more often that I can probably ever return the favor.

My family has been an unlimited supply of support. My mother has always made sure I know just how much she brags about my accomplishments, no matter how mundane they seem to me. My brother was always down to chat online and make me feel jealous about his latest traveling adventures. My father, probably more than anyone, encouraged the curiosity and work ethic that made want to get a PhD, and few things sadden me more than that he was not there to see me graduate.

Jessica Kirkpatrick, my partner in crime, made it all worthwhile. She was a relentless cheerleader, amazing listener, surprisingly willing proofreader, and possibly the only person in the world who enjoys a good sleep-in as much as I do. I shudder to think of spending these six years in graduate school without her.

Thank you one last time to everyone who I've known in my time at Berkeley. I couldn't have done it without you.

For my dad, who always told me to
look it up in my Funk and Wagnalls.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Structured Prediction

At a high-level, the goal of research in Natural Language Processing (NLP) is to get computers to understand and produce human language. Of course, this is a lofty and ill-defined goal – what exactly does it mean for a computer to "understand" human language? As such, much of NLP research focuses on specific tasks that can be concretely formulated. For example, **machine translation** is the task of taking a sentence in one language and producing a grammatically correct and semantically faithful rendering of it in another; **parsing** is the task of producing a syntactic or semantic analysis of an input sentence in some agreed-upon grammatical formalism; and **text categorization** is the task of assigning one of a pre-defined set of categories such as "sports" or "politics" to a document.

A large number of these tasks can be formulated as the assignment of the "best" label to an input, where the notion of "best" is captured by a score assigned by a model. In some cases, like text categorization, the set of labels is fixed and usually small, and explicitly scoring all possible labels is trivial. However, in many cases, the set of possible labels can be unmanageably large and even enumerating the set would be intractable. For example, the set of possible labels for machine translation might be the set of all sentences in the output language, which can in principle be infinite. Fortunately, these large label sets typically exhibit some kind of **structure** – that is, they are composed of smaller parts. Because they are composed of parts, it possible to construct labels piece-by-piece. The problem of predicting such structured labels is referred to as **structured prediction**, and the process of incrementally constructing the best structured label(s) is called **search** (or sometimes **decoding**).

1.2 Optimal Search

In this thesis, we consider problems for which it is feasible to perform **exact** or **optimal** search – that is, search that is guaranteed to find the best possible label according to a model. In particular, we will study problems that can be formulated as a search for the best path in a weighted directed hypergraph (Gallo et al., 1993). Many problems can be formulated in this way, including parsing (Klein and Manning, 2001b) with a probabilistic context-free grammar (PCFG), bitext parsing (Wu, 1997), and syntactic machine translation (Galley et al., 2006). This formalism also includes weighted directed graphs as a special case, allowing us to treat search in sequence models using the same framework.

In general, optimal search in a hypergraph can be done using dynamic programming in time linear in the number of hyperedges it contains, though the number of hyperedges is not necessarily linear in the size of the input. We will restrict ourselves to problems where the number of hyperedges is polynomial in the size of the input – meaning optimal search is tractable – but large, so that exhaustive dynamic programming is cumbersome. We will discuss algorithms that can find the best path without considering all hyperedges in the hypergraph, and hence can speed up search without sacrificing search quality. We will also propose search algorithms that can extract the k-best paths instead of the single best. We will compare these approaches both against exhaustive search, and against approximate search techniques which speed up search by sacrificing optimality guarantees.

1.2.1 Why Worry about Optimality?

In many practical settings, finding a solution that is guaranteed to be optimal is not a primary concern. Many practical search algorithms focus on providing good trade-offs between search speed and quality, and it is often possible to achieve significant gains in search speed by incurring a relatively small number of search errors. Moreover, there is no guarantee the model-optimal solution is actually the best according some external evaluation metric, and it is not uncommon for researchers to find that introducing search errors actually *increases* task-level accuracy!

So one might ask, why worry about optimality at all? There are several reasons. First, even in cases where one is willing to trade search quality for speed, it is not possible to know exactly how many search errors are caused by inexact search procedures unless one has access to an optimal search procedure. Second, there are cases in which optimal search can be faster and more accurate than sub-optimal search – we discuss such a case in Section 3.3.3. Finally, optimal search techniques often have relatively clear sub-optimal variants, and the development of more powerful optimal search techniques can lead to new and more successful sub-optimal algorithms. For example, the k-best list extraction technique we discuss in Chapter 5 can be employed with inadmissible heuristics to yield search that is inexact, but still employs a heuristic to speed up search in a fashion which is to our knowledge novel.

1.3 What This Thesis Is Not About

This thesis is not about learning models for structured prediction. We will always assume a fixed model, and concern ourselves only with the task of discovering the best (or k-best) labels for some input given that model. For a comprehensive review of approaches to learning such models, we refer the reader to Smith (2011).

This thesis always makes the assumption that the notion of the "best" label under a model denotes the single highest scoring label (usually called the **Viterbi** label). Other notions of best are used in the literature. For example, in Minimum Bayes Risk decoding (Kumar and Byrne, 2004), the goal is to find the label which minimizes the expected loss for some user-defined loss function; note that the Viterbi label is the Minimum Bayes Risk label if the 0-1 loss is used.

1.4 Outline

This thesis is structured as follows. Chapter 2 introduces notation and provides relevant background for search algorithms for structured problems. In particular, we discuss dynamic programming, agenda-based search, and non-optimal search strategies. Chapter 3 describes the Hierarchical A* algorithm of Felzenszwalb and McAllester (2007) and provides comparisons against other optimal and non-optimal search algorithms on the task of parsing. Chapter 4 describes a novel modification of Hierarchical A* called Bridging Hierarchical A*, which demonstrates more robustness to different choices of hierarchies. In Chapter 5, we propose two methods for extracting k-best lists using A* search algorithms. We conclude in Chapter 6.

1.5 Contributions

The contributions of this thesis are:

- An empirical comparison of optimal and non-optimal hierarchical search strategies for parsing (Chapter 3). This chapter is largely based on work previously published in Pauls and Klein (2009b).
- Bridging Hierarchical A*, a novel variant of Hierarchical A* (Chapter 4). This chapter is largely based on work previously published in Pauls and Klein (2010a).
- K-best A*, a generalization of A* that allows the extraction of k-best lists. This chapter is largely based on work previously published in Pauls and Klein (2009a) and Pauls and Klein (2010b).
- A novel proof of correctness for Hierarchical A* that, with minor modification, also proves the correctness of Bridging Hierarchical A* (Appendices A.2 and A.3). This proof also generalizes Hierarchical A* (and hence also standard A*) to allow negative weights.

Chapter 2

Structured Search

2.1 Hypergraph Basics and Problem Statement

A weighted directed hypergraph is a combinatorial data structure that generalizes the notion of a weighted directed graph. A hypergraph G is composed of a set of vertices V and directed (hyper)edges E, where each edge $e \in E$ has a list of tail vertices $T(e)$, head vertices $H(e)$, and weight $w(e) \in \mathbb{R}$. A graph is the special case when both T and H always have cardinality of at most one. A B-hypergraph is a hypergraph in which H must have cardinality one, but the cardinality of T is not constrained; see Figure 2.1(a) for an example B-hypergraph. We will implicitly assume that hypergraphs are B-hypergraphs for the remainder of the thesis, and use $h(e)$ to denote the single head of a hyperedge. We will often write a hyperedge in the form

$$v_1 \ldots v_n \xrightarrow{w} v$$

where $T(e) = \{v_1, \ldots, v_n\}$, $h(e) = v$, and $w = w(e)$.

The vertex set V contains a distinguished root vertex $\bar{v}$ and leaf vertices $\underline{V}$. The root vertex is the "goal" or "sink" of G, while the leaf vertices are the "start" or "source" vertices. Leaf vertices are never the head of any edge, and the root vertex is never in the tail of any edge.

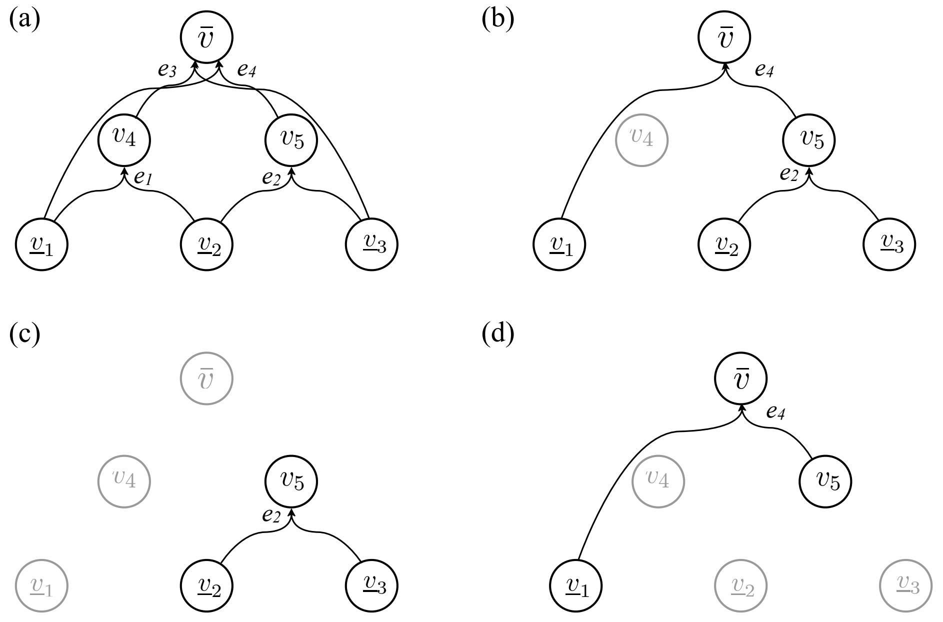

Figure 2.1. Basic hypergraph concepts. (a) A simple hypergraph with edge set $E = \{e_1, e_2, e_3, e_4\}$ and vertex set $V = \{\underline{v}_1, \underline{v}_2, \underline{v}_3, v_4, v_5, \bar{v}\}$, leaves $\underline{V} = \{\underline{v}_1, \underline{v}_2, \underline{v}_3\}$, and root $\bar{v}$. (b) A complete path $\bar{\rho} = \{e_2, e_4\}$. (c) An (inside) path $\rho(v_5) = \{e_2\}$. (d) An outside path $\ddot{\rho}(v_5) = \{e_4\}$.

An **inside path** (or simply **path**) $\rho(v)$ to a target vertex v is a set of hyperedges that satisfies the following recursive property: if $v \in \underline{V}$, then the set is empty; otherwise, the set is the union of exactly one hyperedge e with $h(e) = v$ and paths $\rho(v_i)$ for each $v_i \in T(e)$. See Figure 2.1(c) for an example. We call a path to the root $\bar{v}$ a **complete path** (Figure 2.1(b)), which we denote as $\bar{\rho}$.[1] The weight $w(\rho(v))$ of a path $\rho(v)$ is $\sum_{e \in \rho(v)} w(e)$. We will take the convention that weights are costs to be minimized, so that the "best" or "optimal" path to v is $\arg\min_{\rho(v)} w(\rho(v))$. We will assume that the hypergraphs that we deal with do not have negative cycles, so that the weight of the best path can not be arbitrarily negative.

Hypergraphs provide a language for compactly encoding a large number of possible weighted tree structures or **derivations** over the leaves $\underline{V}$. This formalism is very powerful, and can be used to encode possible labels for a wide variety of problems already discussed in the introduction. Throughout this thesis, we will assume that we are given a **grammar** $\mathcal{G}$ (for example, a PCFG) that tells us how to construct a hypergraph

[1] Note that this definition does not require that a complete path contains an outgoing hyperedge for every leaf vertex. In practice, this is a problem-specific but very common requirement. For example, in parsing, each vertex is associated with a range of leaf vertices, and hyperedges can only combine vertices that cover adjacent ranges. However, we stress that this requirement can be encoded with an appropriate choice of vertices V and hyperedges E, and does not require modification of the general hypergraph framework.

$G = (V, E)$ for a given input (for example, a sentence). Our goal in this thesis is to provide efficient algorithms for extracting the 1-best or k-best paths through G.

2.2 Optimal Search Algorithms

2.2.1 Dynamic Programming

In general, it is possible to construct the best path through a hypergraph G using an $O(|E|)$ dynamic program that visits each hyperedge in the hypergraph exactly once. Because paths through hypergraphs are trees, they exhibit the optimal substructure property: the best path to a vertex v must be constructed from some edge e with $h(e) = v$ and the best paths to each of the children of e. This allows us to express the weight of best path to v, which we will denote with $\beta(v)$, with a simple recursion:

$$\beta(v) = \min_{e:h(e)=v} w(e) + \sum_{v' \in T(e)} \beta(v')$$

with base case $\beta(\underline{v}) = 0 \ \forall \underline{v} \in \underline{V}$. The weight $\beta(v)$ is called the **Viterbi inside cost**. We can efficiently calculate $\beta(v)$ for all v in a bottom-up fashion, where the notion of "bottom-up" is captured by a topological ordering on the vertices (see Huang (1998) for more details). Not all hypergraphs have a topological ordering, though most grammars of interest produce hypergraphs that either have a topological ordering, or can be transformed into hypergraphs that do. For example, PCFGs that contain unary cycles can have cycles removed by computing a unary closure. After computing $\beta(v)$ for all v, we can recover the weight of the best complete path in the graph from $\beta(\bar{v})$, and the best path itself can be recovered by maintaining backpointers that remember which hyperedge e obtained the minimum for each v.

This general dynamic program includes the Viterbi algorithm for sequence models and the CKY algorithm for PCFGs as special cases, among others. Although this algorithm is linear in the size of the hypergraph, it is not necessarily linear in the size of the input. For example, for PCFGs, the number of hyperedges in the hypergraph for a sentence of length n is $O(n^3)$. Even worse, the number of hyperedges also depends on a multiplicative factor which, although constant with respect to the size of the input, can nevertheless be quite large. For example, for sequence models, the number of hyperedges is $O(n \cdot |S|^2)$, where n is the length of the sequence and $|S|$ is the number of possible symbols at each step in the sequence. This factor is often called the **grammar constant**, and can be prohibitively large for many grammars.[2]

Viterbi outside costs

A similar dynamic program can also be used to calculate the **Viterbi outside cost** $\alpha(v)$, which is the best possible weight of extending a path to v all the way to the root $\bar{v}$. More formally, an **outside path**, denoted $\ddot{\rho}(v)$, is a set of vertices given by $\bar{\rho}(v) - \rho(v)$ for some (inside) path $\rho(v)$ and a complete path $\bar{\rho}(v)$ passing through v, where $\rho(v) \subseteq \bar{\rho}(v)$; see Figure 2.1(d). The Viterbi outside cost of v is the minimum over all $\ddot{\rho}(v)$.

The Viterbi outside cost satisfies the recursion

$$\alpha(v) = \min_{e:v \in T(e)} w(e) + \alpha(h(e)) + \sum_{v' \in T(e) - v} \beta(v')$$

and can be computed by proceeding through the vertices in reverse topological order, with base case $\alpha(\bar{v}) = 0$.

The sum of Viterbi inside and outside costs for a vertex v give us the **max-marginal** $\mu(v)$, which is the cost of the best complete path $\bar{\rho}(v)$ that contains v. That is, $\beta(v) + \alpha(v) = \mu(v)$.

Computing the Viterbi outside costs or max-marginals is not necessary for performing search – in particular, this computation requires that search has already been performed by computing $\beta(v)$ for all v. However, outside scores and max-marginals will prove useful in several of the techniques we discuss in this thesis.

2.2.2 Weighted Deduction Rules and Agenda-Driven Search

An alternative to dynamic programming for search in hypergraphs is an agenda-driven search using weighted deduction rules (Shieber et al., 1995; Nederhof, 2003). Weighted deduction rules provide a lan-

[2] In fact, in many cases, although the grammar constant is asymptotically constant for large n, it is not necessarily constant for practical values of n (Klein and Manning, 2001a).

guage for specifying search **states** or **items** along with rules that declare how states can be combined together to form new states. A deduction rule ψ has the form

$$\phi_1 : w_1 \; \ldots \; \phi_n : w_n \xrightarrow{p_\psi(w_1,\ldots,w_n)} \phi_0 : g_\psi(w_1, \ldots, w_n)$$

where $\phi_1, \ldots, \phi_n$ are the **antecedent** states of the deduction rule and ϕ_0 is the **conclusion** state. A deduction rule states that, given the antecedents $\phi_1, \ldots, \phi_n$ with weights $w_1, \ldots, w_n$, the conclusion ϕ_0 can be produced with weight $g_\psi(w_1, \ldots, w_n)$ and priority $p_\psi(w_1, \ldots, w_n)$. For the algorithms we discuss in this paper, states will be associated with vertices in the hypergraph and rules will be associated with hyperedges, though the association will not necessarily be one-to-one.

Deduction rules are "executed" within a generic agenda-driven algorithm that constructs states in a prioritized fashion. The algorithm maintains an **agenda** (a priority queue of unprocessed states), as well as a **chart** or **closed list** of states already processed. The agenda is initialized with a set of **initial** states. The fundamental operation of the algorithm is to pop the highest (minimum) priority state ϕ from the agenda, put it into the chart with its current weight, and form using deduction rules any states that can be built by combining ϕ with other states already in the chart. If the conclusion state is new, or is already on the agenda but with a worse priority, it is put on the agenda with priority given by $p_\psi(\cdot)$ and weight $g_\psi(\cdot)$. The search proceeds until a state that satisfies a **goal test** is reached.

This framework defines a declarative way for specifying several important search algorithms. For example, Dijkstra's algorithm (Dijkstra, 1959), uniform cost search, and A* search (Hart et al., 1968) can all be expressed as particular choices of $p_\psi(\cdot)$ and $g_\psi(\cdot)$. In fact, even exhaustive dynamic programming can be described using deduction rules by simply assigning priorities to states that yield the topological ordering in the hypergraph. The algorithms in Chapters 3, 4, and 5 will all be expressed using weighted deduction rules.

A* Search

The A* algorithm of Hart et al. (1968) was originally designed for search in graphs, and was extended by Klein and Manning (2003c) for search in hypergraphs. Several algorithms in this thesis are generalizations of this basic algorithm, and we describe it here as an example usage of weighted deduction rules.

We can construct weighted deduction rules that describe the A* algorithm for a particular hypergraph G as follows: for each hyperedge $e \in G$, we construct a deduction rule ψ whose antecedent states represent the tail vertices $T(e)$ and whose conclusion state represents the head vertex $h(e)$. The functions $g_\psi(\cdot)$ and $p_\psi(\cdot)$ are given by

$$g_\psi(w_1, \ldots, w_n) = w(e) + \sum_{i=1}^{n} w_i$$

and

$$p_\psi(w_1, \ldots, w_n) = g_\psi(w_1, \ldots, w_n) + \hat{\alpha}(h(e))$$

where $\hat{\alpha}(\cdot)$ is a **heuristic**, which is an estimate of the Viterbi outside score. The search is initialized by placing states for each leaf vertex $\underline{v} \in \underline{V}$ on the agenda, and proceeds until a state representing the root vertex is removed from the agenda.

If the heuristic is **consistent**, meaning it satisfies $\hat{\alpha}(\bar{v}) = 0$ and

$$w(\rho(v)) + \hat{\alpha}(v) \leq \sum_{v' \in T(e)} w(\rho(v')) + \hat{\alpha}(h(e)) + w(e)$$

for all edges $e \in G$ with $v \in T(e)$ and all paths $\rho(v)$ and $\rho(v')$ to vertices $v, v' \in T(e)$, then A* search guarantees that when a state is removed from the agenda, its weight will be equal to the Viterbi inside cost $\beta(v)$ of the vertex v it represents. We refer to this guarantee as a **correctness guarantee**. As a special case, this guarantee ensures that when the root state is reached, its weight will be the weight of the best path to the root $\bar{v}$. The actual path itself can be constructed by maintaining backpointers. A* also provides a **monotonicity guarantee**, which states that vertices are removed from the agenda in increasing order of $\beta(v) + \hat{\alpha}(v)$.

A consistent heuristic is also **admissible**, meaning that $\hat{\alpha}(v)$ is a lower bound on the cost of any outside paths $\ddot{\rho}(v)$ from v, i.e. $\hat{\alpha}(v) \leq \alpha(v)$ for all v. The more tightly $\hat{\alpha}(\cdot)$ approximates $\alpha(\cdot)$, the quicker the search will proceed. In particular, the monotonicity guarantee implies that A* will explore all vertices that satisfy

$$\beta(v) + \hat{\alpha}(v) \leq \beta(\bar{v})$$

In other words, A* explores all vertices whose estimated best path cost $\beta(v) + \hat{\alpha}(v)$ is better than the true best path through the graph. As a special case, if heuristic is **perfect**, meaning $\hat{\alpha}(v) = \alpha(v)$ for all v, then A* will only explore vertices that are on the best path (up to ties).

In addition to allowing exact search without exploring the entire hypergraph, A* also has the advantage that it does not require that a topological ordering on the vertices exists since it handles cycles implicitly; see Huang (1998) for more discussion.[3]

2.3 Approximate Search Algorithms

The previous section described well-known search techniques for hypergraphs that are provably optimal, and gave an example of a search technique (A*) that is optimal despite not exploring all edges in a hypergraph. In this section, we describe search techniques that avoid exploration of the entire hypergraph in ways that do not guarantee optimality, but are often used in practice because they can be made arbitrarily fast, though at the cost of arbitrarily large potential for search errors. These techniques will be useful in determining how much efficiency we lose by using algorithms that guarantee optimality.

2.3.1 Beam Search

Beam search (Lowerre, 1976) is a pruning procedure that introduces competition among groups of vertices, and keeps only the best set of vertices in each group. The notion of a "group" is captured by some projection function $\pi(v)$. In most cases, there are some quite natural projections: for example, in HMMs, where beam search was first employed, vertices that represent states at a particular time step are projected into the same group; in parsing, it is common to project vertices that represent states that cover a particular span of a sentence to the same group. There are many different strategies for choosing the "best set" of

[3]Although the hypergraphs may be cyclic, the optimal solutions cannot. Note that this is a vacuous condition because in the absence of negative cycles, a cycle can only increase the cost of a solution and thus cannot be part of an optimal solution.

vertices in each group. Two common choices are to keep a fixed number of vertices, or to keep all vertices within some threshold of the best vertex found for each group.

By pruning vertices early on in the search, beam search reduces the number of hyperedges explored because any hyperedge reachable only by a pruned vertex will never be considered. This strategy can lead to search errors because the optimal path may involve a vertex v that has sufficiently low Viterbi inside score $\beta(v)$ relative to other vertices in its group that it is pruned from consideration. Even worse, beam search has the peculiar property that it is possible for it to find solutions with *higher* costs when the beams are made larger. Nevertheless, its simplicity makes it a common choice in practice.

2.3.2 Coarse-to-Fine

Coarse-to-fine search (Charniak and Johnson, 2005), hereafter CTF, is a modification of exhaustive dynamic programming. Like beam search, CTF introduces competition among groups of vertices specified by a projection function. However, where beam search prunes vertices "on-the-fly" as search proceeds, CTF prunes vertices in advance by first performing dynamic programming in a projected "coarse" grammar, and pruning any vertices whose projections were low-scoring according to the coarse grammar. More formally, given some projection function π, CTF builds a smaller, **projected** hypergraph G^π for a graph G by projecting each edge $e = v_1 \ \ldots \ v_n \xrightarrow{w} v$ to a coarse edge $\pi(e) = e^\pi = \pi(v_1) \ \ldots \ \pi(v_n) \xrightarrow{w^\pi} \pi(v)$ of with w^π chosen to approximate $w(e)$. CTF then computes Viterbi inside and outside scores using dynamic programming, and prunes vertices from G whose projections have with high max-marginal cost $\mu(\pi(v))$ in G^π. More precisely, CTF prunes any vertex $v \in V$ with

$$\mu(\pi(v)) \geq \mu(\pi(\bar{v})) + \tau$$

CTF can prune far more effectively than beam search because it makes pruning decisions based on coarse max-marginals $\mu(\pi(v))$, which incorporate both inside and outside information, where beam search only considers the inside scores $\beta(v)$. However, the performance of CTF is dependent on an appropriate choice of approximating weights $w(e^\pi)$, which is not necessary in standard beam search.

We note the particular algorithm we have described here is a special case of a larger family of algorithms that go by the name of “coarse-to-fine.” In particular, it is common to prune based on sum-marginals instead of max-marginals (Charniak and Johnson, 2005).

2.3.3 Heuristic Search

When we described agenda-based search in Section 2.2.2, we assumed that the priority function $p(\cdot)$ took the form $p(v) = \beta(v) + \hat{\alpha}(v)$ for some consistent heuristic $\hat{\alpha}$, which is sufficient to guarantee optimality. There are several other choices of priority functions that can lead to significant increases in efficiency, though at the cost of optimality guarantees. For example, if $\hat{\alpha}(\cdot)$ is inadmissible (and hence inconsistent), we lose optimality, but it becomes much easier to devise heuristics that tightly approximate the true Viterbi outside costs $\alpha(\cdot)$. Many practical algorithms rely on heuristic search; in particular, most phrase-based machine translation systems rely heavily on inadmissible heuristic costs for efficient search (Koehn et al., 2003).

Chapter 3

Hierarchical Search

In Chapter 2, we reviewed well-known techniques for optimal- and non-optimal search in hypergraphs. In this chapter, we empirically evaluate **hierarchical** search techniques which have become increasingly popular for NLP problems (Charniak et al., 2006; Petrov and Klein, 2007). In particular, we evaluate the Hierarchical A* (HA*) algorithm of Felzenszwalb and McAllester (2007), and compare it against non-hierarchical A* as well as the hierarchical (non-optimal) coarse-to-fine algorithm of Petrov and Klein (2007). We evaluate these techniques on several different parsing problems.

3.1 Hierarchical A*

HA* is described in its most general form in Felzenszwalb and McAllester (2007). In this section, we first give a brief overview of HA* as it applies to hypergraphs in general, and then describe the algorithm in detail as it applies to parsing.

3.1.1 Overview

In standard, non-hierarchical A*, the heuristic $\hat{\alpha}(\cdot)$ is assumed to come from a black box. For example, Klein and Manning (2003c) precomputes most heuristics offline, while Klein and Manning (2003a) solves simpler parsing problems for each sentence. In such cases, the time spent to compute heuristics is often non-trivial. Indeed, it is typical that effective heuristics are themselves expensive search problems. HA* relies on the insight that the heuristic itself can be computed in an agenda-driven way, making it possible speed up the computation of a heuristic using the A* algorithm itself.

Formally, HA* takes as input a sentence and a sequence (or *hierarchy*) of $M + 1$ grammars $\mathcal{G}^0 \dots \mathcal{G}^M$, where $\mathcal{G}^M$ is the *target grammar* and $\mathcal{G}^0 \dots \mathcal{G}^{m-1}$ are *auxiliary grammars*. Each grammar $\mathcal{G}^m$ tells us how to generate a hypergraph G^m for a given input as in Chapter 2, with a root vertex $\bar{v}^m$ and leaf vertices $\underline{V}^m$.

The hypergraphs $G^0 \dots G^M$ must form a hierarchy in which G^{m-1} is a **relaxed projection** of G^m. A grammar G^{m-1} is a relaxed projection of G^m if it is a projection (as defined in Section 2.3.2) with projection function $\pi(\cdot)$ and approximate weights $w(e^\pi) \leq w(e)$ for all e that project to e^π. Given a target graph G^m and a projection function π, it is easy to construct a relaxed projection G^{m-1} by minimizing over edges collapsed by $\pi(\cdot)$:

$$w(e^\pi) = \min_{\pi^{-1}(e)} w(e^\pi)$$

where we denote the set of edges that project to e^π with $\pi^{-1}(e^\pi)$. Given a series of projection functions $\pi^1 \dots \pi^M$, we can construct a hierarchy of relaxed projections by projecting $\mathcal{G}^M$ to $\mathcal{G}^{M-1}$ using π^M, then $\mathcal{G}^{M-1}$ to $\mathcal{G}_{M-2}$ using π^{M-1}, and so on. Note that by construction, Viterbi inside and outside scores in a relaxed projection G^{m-1} are consistent heuristics for parses in G^m (Felzenszwalb and McAllester, 2007).

HA* differs from standard A* in two ways. First, A* only performs search for a single hypergraph, and all states in the search represent possible Viterbi inside scores for vertices in the target graph G^M. HA*, on the other hand, performs search in all levels in the hierarchy, with states representing possible scores for vertices for all graphs $G^0 \dots G^M$ in the hierarchy, though these states all co-exist on single, global priority queue. Second, while A* only computes inside scores for vertices it explores, HA* computes both inside and outside scores for vertices (except for vertices in G^M, where only inside scores are needed). Because Viterbi outside scores in a relaxed projection G^{m-1} form consistent heuristics for a graph G^m, HA* can compute both heuristics and final path costs using the same mechanism, without having to exhaustively compute inside and outside scores at any level of the hierarchy.

3.1.2 HA* for Parsing

Here, we describe the HA* algorithm in detail as it applies to parsing, in hopes of that concreteness will make the algorithm more readily understandable. We refer the reader to Appendix A and to Felzenszwalb and McAllester (2007) for a more general treatment.

For parsing, our grammars $\mathcal{G}^m$ take the form of a PCFG in Chomsky normal form[1] with rules of the form $r = A^m \rightarrow B^m\ C^m$, where A^m, B^m and C^m all represent non-terminal symbols in the $\mathcal{G}^m$. This vocabulary has a distinguished root symbol R^m. Each rule r has a non-negative[2] weight w_r (e.g. a negative log probability), and we wish to minimize the sum of the rule weights. Given an input sentence $S = s_0 \ldots s_{n-1}$ and a PCFG grammar, we construct a hypergraph in which each vertex represents a symbol A^m over a span[3] $[i, j]$ of the sentence, which we denote as (A^m, i, j). For each pair of vertices (B^m, i, l) and (C^m, l, j), the hypergraph contains an edge with head (A^m, i, j) for every rule $A^m \rightarrow B^m\ C^m$. There is one leaf vertex $\underline{v}^m = (s^m_i, i, i+1)$ for each word s^m_i in the input sentence, and the root vertex is $\bar{v}^m = (R^m, 0, n)$. We use the notation A^{m-1} to refer to $\pi^m(A^m)$.

As mentioned previously, HA* tracks states that compute both inside and outside scores for each vertex; we call these different types of states **inside states** and **outside states**. We denote inside and outside states as $\iota^m = I(X^m, i, j)$ and $o^m = O(X^m, i, j)$, respectively. For example, $I(\text{VP}, 0, 3)$ denotes a state that tracks the inside score of the vertex $(\text{VP}, 0, 3)$. When we need to denote the inside version of an outside edge, or the reverse, we write $o^m = \check{\iota}^m$, etc. We refer to the weight of inside and outside states as $w(\iota^m)$ and $w(o^m)$ on the chart.

Like A*, HA* can be formulated in terms of weighted deduction rules. As we described in Section 2.2.2, A* required one weighted deduction rule per hyperedge in the hypergraph, with antecedents corresponding to the tail of the hyperedge and conclusion corresponding to the head. Because HA* computes both inside and outside scores, and uses outside scores at one level as heuristics in the next, the rules are somewhat more complex. However, they can be compactly described in terms of **deduction rule schemata** which specify

[1]Chomsky normal form demands that all rules are binary. Unary rules and preterminal productions, which are present in most grammars, can be easily expressed as binary rules with an empty right child. Rules that have more than two children can be binarized.

[2]Non-negative weights are not actually a requirement for A* or HA*. We discuss this further in Appendix A.

[3]As is standard, we take the convention that the first index of a span is inclusive and the second is exclusive, so that the span $[0, 2]$ covers words s_0 and s_1.

Inside Deduction for A*

IN: $I(B,i,l) : w_A \quad I(C,l,j) : w_C \quad \xrightarrow{w_B+w_C+w_r+\hat{\alpha}(A,i,j)} \quad I(A,i,j) : w_B + w_C + w_r$

Table 3.1. Deduction rule schema for A* parsing. The items on the left of the $\rightarrow$ indicate what states must be present on the chart, and the item on the right is the state that may be added to the agenda. The weight of each edge appears after the colon. The rule r is $A \rightarrow B\ C$ with weight w_r.

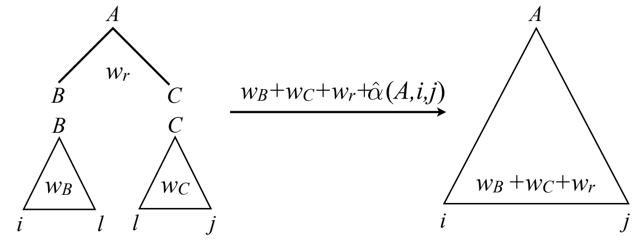

Figure 3.1. A graphical depiction of the deduction rule schema for A* (Table 3.1). Items to the left of the arrow indicate edges and rules that can be combined to produce the state to the right of the arrow. States are depicted as complete triangles. The value inside an state represents the weight of that state. Each new state is assigned the priority written above the arrow when added to the agenda.

general templates for instantiating concrete deduction rules. A* can be described by a single schema, shown in Table 3.1 and Figure 3.1. This schema is instantiated into a concrete deduction rule for each possible rule $r = A \rightarrow B\ C$ and indexes $0 \leq i < n$, $i < j \leq n$, and $i < l < j$.

The five deduction rule schemata that describe HA* for parsing are given in Table 3.2 and represented graphically in Figure 3.2. The general form of HA* is given in Table A.1 in the Appendix. The IN schema (a) is the familiar deduction rule from standard A*: we can combine two adjacent inside state using a binary rule to form a new inside state. The new twist is that because heuristics (scores of outside edges from the previous level) are also computed on the fly, they may not be ready yet. Therefore, we cannot carry out this deduction until the required outside state is present in the previous level's chart. That is, fine inside deductions wait for the relevant coarse outside states to be popped. While coarse outside states contribute to *priorities* of refined inside scores (as heuristic values), they do not actually affect the inside scores of states (again just like basic A*).

In standard A*, we begin with all terminal states on the agenda. However, in HA*, we cannot enqueue refined terminal states until their outside scores are ready. The IN-B schema specifies the base case for

Inside Deductions for Hierarchical A*

IN-B:	$O(s_i^{m-1}, i, i+1) : w$	$\xrightarrow{w}$	$I(s_i^m, i, i+1)$	$: 0$
IN:	$O(A^{m-1}, i, j) : w_A \;\; I(B^m, i, l) : w_B \;\; I(C^m, l, j) : w_C$	$\xrightarrow{w_B + w_C + w_r + w_A}$	$I(A^m, i, j)$	$: w_B + w_C + w_r$

Outside Deductions for Hierarchical A*

OUT-B:	$I(R^m, 0, n) : w$	$\xrightarrow{w}$	$O(R^m, 0, n)$	$: 0$
OUT-L:	$O(A^m, i, j) : w_A \;\; I(B^m, i, l) : w_B \;\; I(C^m, l, j) : w_C$	$\xrightarrow{w_A + w_C + w_r + w_B}$	$O(B^m, i, l)$	$: w_A + w_C + w_r$
OUT-R:	$O(A^m, i, j) : w_A \;\; I(B^m, i, l) : w_B \;\; I(C^m, l, j) : w_C$	$\xrightarrow{w_A + w_B + w_r + w_C}$	$O(C^m, l, j)$	$: w_A + w_B + w_r$

Table 3.2. Inside and outside deduction rule schemata for HA* parsing. The rule r is in all cases $A^m \to B^m \; C^m$.

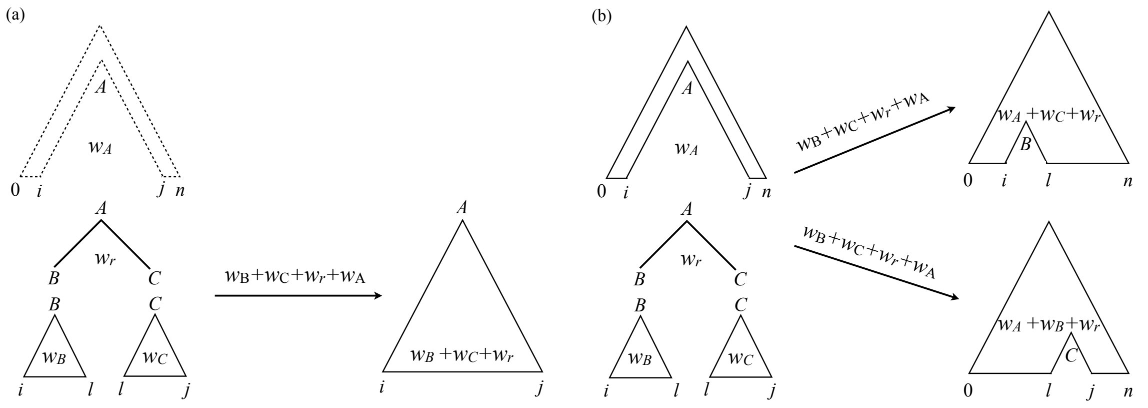

Figure 3.2. Non-base case deductions for HA* depicted graphically. (a) shows the IN schema and (b) shows the OUT-L and OUT-R schemata. Inside states are depicted as complete triangles, while outside states are depicted as chevrons. A state from a previous level in the hierarchy is denoted with dashed lines.

a inference at level m: we cannot begin until the outside score for the terminal symbol s_i is ready in the coarser level $m - 1$. The initial queue contains only the most abstract level's terminals, $I(s_i^0, i, i+1)$. The entire search terminates when the inside edge $I(R^m, 0, n)$, representing the root in the target hypergraph, is dequeued.

The deductions that assemble outside states are less familiar from the standard A* algorithm. These deductions take *larger* outside states and produce *smaller* sub-states by linking up with inside state, as shown in Figure 3.2(b). The OUT-B schema states that an outside pass for level m can be started if the inside score of the root symbol R^m for that level has been computed. The OUT-L and OUT-R rules are the deduction rules for building outside edges. OUT-L states that, given an outside state over the span $[i, j]$ and some inside state over $[i, l]$, we may construct an outside state over $[l, j]$.

As in standard A*, inside states are placed on the agenda with a priority equal to their path cost (inside score) and some estimate of their completion cost (outside score), now taken from the previous projection rather than a black box. Specifically, the priority function takes the form $p(\iota^m) = w(\iota^m) + w(\check{\iota}^{m-1})$, where $\check{\iota}^{m-1}$ is the outside version of ι one level previous in the hierarchy.

Outside states also have priorities that combine path costs with a completion estimate, except that the roles of inside and outside scores are reversed: the path cost for an outside state o^m is its (outside) score $w(o^m)$, while the completion cost is the inside score $w(\iota^m)$ of o's complementary inside state $\iota^m = \check{o^m}$. Therefore, $p(o^m) = w(o^m) + w(\check{o^m})$.

Note that inside states combine their inside scores with outside scores from a *previous level* (a lower bound), while outside states combine their outside scores with (exact) inside scores from *the same level*, which are already available. Felzenszwalb and McAllester (2007) show that these choices of priorities have the same guarantee as standard A*: whenever an inside or outside state comes off the queue, its path cost is optimal.

3.1.3 Theoretical Guarantees

HA* comes with correctness and monotonicity guarantees analogous to those of A*. We prove these guarantees in a way that slightly generalizes the proof given in (Felzenszwalb and McAllester, 2007) in Appendix A.2. The correctness guarantee states that when a state is removed for the agenda, $w(\iota^m) = \beta(\iota^m)$ for inside states and $w(o^m) = \alpha(o^m)$ for outside states. Together with the correctness guarantee, the monotonicity guarantee ensures that states are popped off the agenda in order of their **intrinsic priority** $\dot{p}(\cdot)$. The intrinsic priority for inside and outside states ι^m and o^m are given by

$$\dot{p}(\iota^m) = \beta(\iota^m) + \alpha(\check{\iota}^{m-1})$$

$$\dot{p}(o^m) = \alpha(o^m) + \beta(\check{o}^m)$$

In other words, HA* only pops of inside states ι^m for which

$$\beta(\iota^m) + \alpha(\breve{\iota}^{m-1}) \leq \beta(\iota_R^M)$$

and outside states o^m for which

$$\alpha(o^m) + \beta(\breve{o}^m) \leq \beta(\iota_R^M)$$

where $\beta(\iota_R^M)$ the Viterbi inside cost of the root in the target grammar.

3.2 Agenda-Driven Hierarchical Coarse-to-Fine Parsing

We would like to compare HA* and hierarchical CTF pruning. Unfortunately, these two algorithms are generally deployed in different architectures: CTF is most naturally implemented using a dynamic program like CKY, while best-first algorithms like A* are necessarily implemented with agenda-based parsers. To facilitate comparison, we would like to implement them in a common, agenda-driven architecture. An agenda-driven implementation of CTF allows us to put it on a level playing field with HA*, highlighting the effectiveness of the various parsing strategies and normalizing their implementations.

3.2.1 Hierarchical Coarse-to-Fine Pruning

First, we define (hierarchical) coarse-to-fine (CTF) pruning. In standard CTF, we exhaustively parse in each projection level, but skip edges whose projections in the previous level had sufficiently low scores. In particular, a state ι^m will be skipped entirely if its projection ι^{m-1} has a low max marginal $w(\breve{\iota}^{m-1}) + w(\iota^{m-1})$, that is, the score of the best tree containing ι^{m-1} was low compared to the score of the best overall root derivation ι_R^{m-1}. Formally, we prune all ι^m where $w(\iota^{m-1}) + w(\bar{\iota}^{m-1}) > w(\iota_R^{m-1}) + \tau$ for some threshold τ. Note that because this pruning can lead to search errors, we are not necessarily guaranteed that $w(\iota^m) = \beta(\iota^m)$ and $w(o^m) = \alpha(o^m)$, so the max-marginals computed by the algorithm are not necessarily the true max-marginals.

3.2.2 Pruning with Priority Functions

We want to implement CTF in an agenda-based setting, noting that the crucial property of CTF is not the CKY order of exploration, but the pruning of unlikely states, which can be equally well done in an agenda-based parser. In fact, it is possible to closely mimic dynamic programs like CKY using a best-first algorithm with a particular choice of priorities.

Note that we can replace the HA* priority function with an alternate priority function of our choosing. In doing so, we may lose the optimality guarantees of HA*, but we may also be able to achieve significant increases in performance. We do exactly this in order to put CTF pruning in an agenda-based framework.

The priority function we use to implement CTF in our agenda-based framework is:

$$\begin{aligned} p(\iota^m) &= w(\iota^m) \\ p(o^m) &= \begin{cases} \infty & w(o^m) + w(\check{o}^m) > w(\iota^m_R) + \tau^m \\ w(o^m) + w(\check{o}^m) & \text{otherwise} \end{cases} \end{aligned}$$

Here, $\tau^m \geq 0$ is a user-defined threshold for level m and $w(\iota^m_R)$ is the weight of the root at level m. These priorities lead to uniform-cost exploration for inside edges and completely suppress outside edges which would have been pruned in standard CTF. Note that, by the construction of the IN rule, pruning an outside edge also prunes all inside edges in the next level that depend on it; we therefore prune slightly earlier than in standard CTF. In any case, this priority function explores a similar *set* of states to CKY-based CTF,[4] but does not necessarily explore those states in the same order.

3.3 Experiments

3.3.1 Evaluation

Our focus is parsing speed. Thus, we would ideally evaluate our algorithms in terms of CPU time. However, this measure is problematic: CPU time is influenced by a variety of factors, including the architecture

[4]The set of states explored by uniform-cost search can be smaller than exhaustive search, and so even with no pruning ($\tau^m = \infty$), our agenda-driven search explores a potentially smaller set of states than CKY.

of the hardware, low-level implementation details, and other running processes, all of which are hard to normalize.

It is common to evaluate best-first parsers in terms of states *removed* off the agenda. This measure is used by Charniak et al. (1998) and Klein and Manning (2003c). However, when states from grammars of varying size are processed on the same agenda, the number of successor states per edge popped changes depending on what grammar the state was constructed from. In particular, states in more refined grammars are more expensive than states in coarser grammars. Thus, our basic unit of measurement will be states *pushed* onto the agenda. We found in our experiments that this was well correlated with CPU time.

3.3.2 State-Split Grammars

We first experimented with the grammars described in Petrov et al. (2006). Starting with an X-Bar grammar, they iteratively refine each symbol in the grammar by adding latent substates via a split-merge procedure. This training procedure creates a natural hierarchy of grammars, and is thus ideal for our purposes. We used the Berkeley Parser[5] to train such grammars on sections 2-21 of the Penn Treebank (Marcus et al., 1993). We ran 6 split-merge cycles, producing a total of 7 grammars. These grammars range in size from 98 symbols and 8773 rules in the unsplit X-Bar grammar to 1139 symbols and 973696 rules in the 6-split grammar. We then parsed all sentences of length $\leq$ 30 of section 23 of the Treebank with these grammars. Our "target grammar" was in all cases the largest (most split) grammar. Our parsing objective was to find the Viterbi derivation (i.e. fully refined structure) in this grammar. Note that this differs from the objective used by Petrov and Klein (2007), who use a variational approximation to the most probable parse.

A* versus HA*

We first compare HA* with standard A*. In A* as presented by Klein and Manning (2003c), an auxiliary grammar can be used, but we are restricted to only one and we must compute inside and outside estimates for that grammar exhaustively. For our single auxiliary grammar, we chose the 3-split grammar; we found that this grammar provided the best overall speed.

[5]http://berkeleyparser.googlecode.com

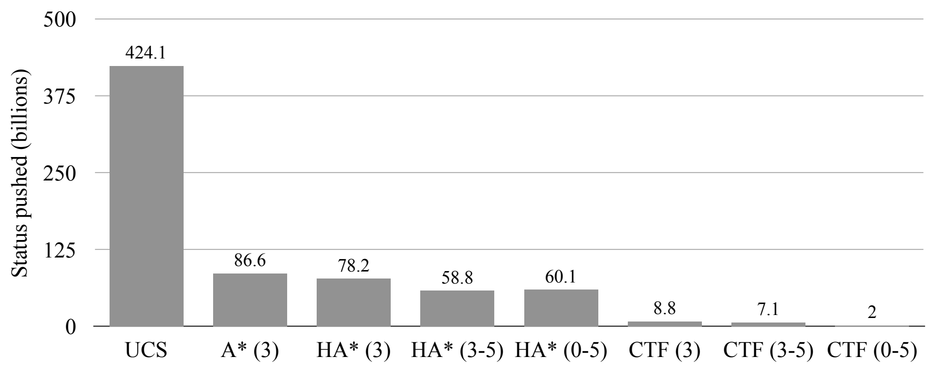

Figure 3.3. Efficiency of several hierarchical parsing algorithms, across the test set. UCS and all A* variants are optimal and thus make no search errors. The CTF variants all make search errors on about 2% of sentences. The number in parenthesis on the axis labels indicates which auxiliary grammars were used for computing heuristics and/or pruning. For example, HA* (3-5) means that HA* was run with a hierarchy in which the 3-, 4-, and 5-split grammars were the auxiliary grammars.

For HA*, we can include as many or as few auxiliary grammars from the hierarchy as desired. Ideally, we would find that each auxiliary grammar increases performance. To check this, we performed experiments with all 6 auxiliary grammars (0-5 split); the largest 3 grammars (3-5 split); and only the 3-split grammar.

Figure 3.3 shows the results of these experiments. As a baseline, we also compare with uniform cost search (UCS) (A* with $h = 0$). A* provides a speed-up of about a factor of 5 over this UCS baseline. Interestingly, HA* using only the 3-split grammar is faster than A* by about 10% despite using the same grammars. This is because, unlike A*, HA* need not exhaustively parse the 3-split grammar before beginning to search in the target grammar.

When we add the 4- and 5-split grammars to HA*, it increases performance by another 25%. However, we can also see an important failure case of HA*: using all 6 auxiliary grammars actually *decreases* performance compared to using only 3-5. This is because HA* requires that auxiliary grammars are all relaxed projections of the target grammar. Since the weights of the rules in the smaller grammars are the minimum of a large set of rules in the target grammar, these grammars have costs that are so cheap that all states in those grammars will be processed long before much progress is made in the refined, more expensive levels. The time spent parsing in the smaller grammars is thus entirely wasted. This is in sharp contrast to hierarchical CTF (see below) where adding levels is always beneficial.

To quantify the effect of optimistically cheap costs in the coarsest projections, we can look at the degree to which the outside costs in auxiliary grammars underestimate the true outside cost in the target grammar

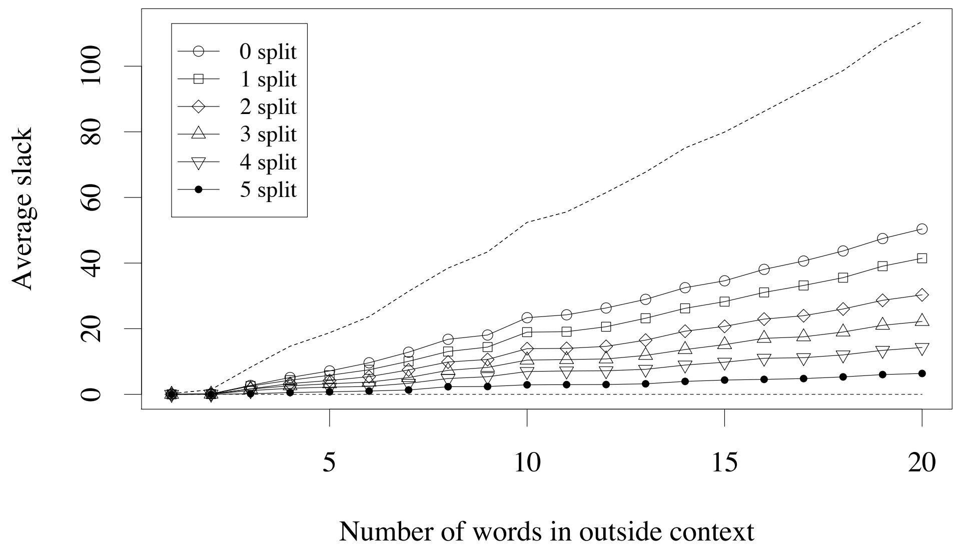

Figure 3.4. Average slack (difference between estimated outside cost and true outside cost) at each level of abstraction as a function of the size of the outside context. The average is over states in the Viterbi tree. The lower and upper dashed lines represent the slack of the exact and uniformly zero heuristics.

(the “slack”). In Figure 3.4, we plot the average slack as a function of outside context size (number of unincorporated words) for each of the auxiliary grammars. The slack for large outside contexts gets very large for the smaller, coarser grammars. In Figure 3.5, we plot the number of states pushed when bounding with each auxiliary grammar individually, against the average slack in that grammar. This plot shows that greater slack leads to more work, reflecting the theoretical property of A* that the work done can be exponential in the slack of the heuristic.

HA* versus CTF

In this section, we compare HA* to CTF, again using the grammars of Petrov et al. (2006). It is important to note, however, that we do not use the same grammars when parsing with these two algorithms. While we use the same *projections* to coarsen the target grammar, the scores in the CTF case need not be lower

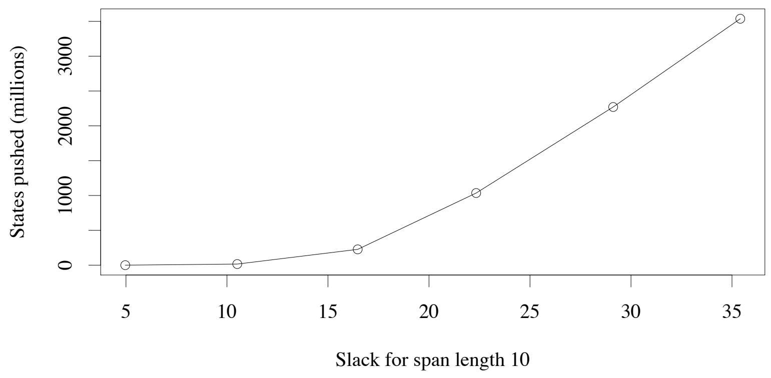

Figure 3.5. States pushed as a function of the average slack for spans of length 10 when parsing with each auxiliary grammar individually.

bounds. Instead, we follow Petrov and Klein (2007) in taking coarse grammar weights which make the induced distribution over trees as close as possible to the target in KL-divergence. These grammars represent not a minimum projection, but more of an average.[6]

The performance of CTF as compared to HA* is shown in Figure 3.3. CTF represents a significant speed up over HA*. The key advantage of CTF, as shown here, is that, where the work saved by using coarser projections falls off for HA*, the work saved with CTF increases with the addition of highly coarse grammars. Adding the 0- through 2-split grammars to CTF was responsible for a factor of 8 speed-up with no additional search errors.

Another important property of CTF is that it scales far better with sentence length than does HA*. Figure 3.6 shows a plot of states pushed against sentence length. This is not surprising in light of the increase in slack that comes with parsing longer sentences. The more words in an outside context, the more slack there will generally be in the outside estimate, which triggers the time explosion.

Since we prune based on thresholds τ^m in CTF, we can explore the relationship between the number of search errors made and the speed of the parser. While it is possible to tune thresholds for each grammar individually, we use a single threshold for simplicity. In Figure 3.7, we plot the performance of CTF using all 6 auxiliary grammars for various values of τ. For a moderate number of search errors ($< 5\%$), CTF

[6]We tried using these average projections as heuristics in HA*, but doing so violates consistency, causes many search errors, and does not substantially speed up the search.

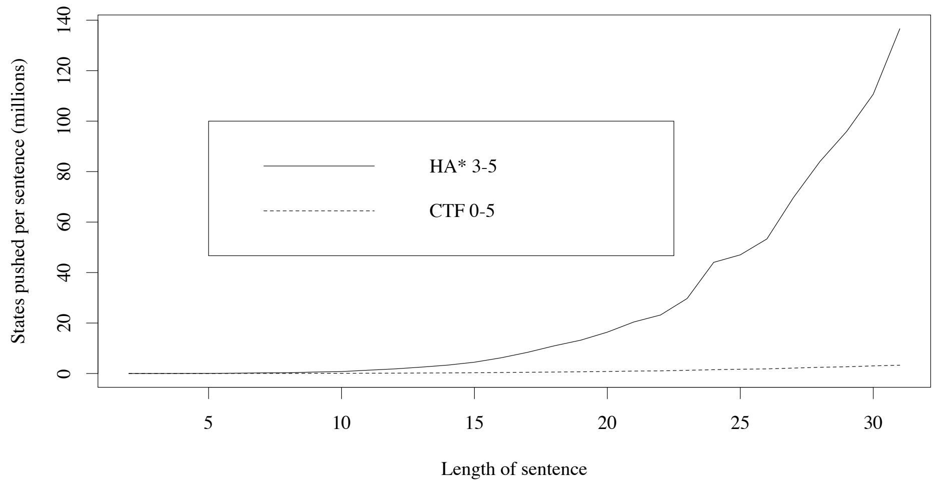

Figure 3.6. States pushed as function of sentence length for HA* 3-5 and CTF 0-5.

parses more than 10 times faster than HA* and nearly 100 times faster than UCS. However, below a certain tolerance for search errors ($< 1\%$) on these grammars, HA* is the faster option.[7]

3.3.3 Lexicalized Parsing

We also experimented with the lexicalized parsing model described in Klein and Manning (2003a). This lexicalized parsing model is constructed as the product of a dependency model and the unlexicalized PCFG model in Klein and Manning (2003b). We constructed these grammars using the Stanford Parser.[8] The PCFG has 19054 symbols 36078 rules. The combined (sentence-specific) grammar has n times as many symbols and $2n^2$ times as many rules, where n is the length of an input sentence. This model was trained on sections 2-20 of the Penn Treebank and tested on section 21.

For these lexicalized grammars, we did not perform experiments with UCS or more than one level of HA*. We used only the single PCFG projection used in Klein and Manning (2003a). This grammar differs from the state split grammars in that it factors into two separate projections, a dependency projection and a PCFG. Klein and Manning (2003a) show that one can use the sum of outside scores computed in these two

[7]In Petrov and Klein (2007), fewer search errors are reported; this difference is because their search objective is more closely aligned to the CTF pruning criterion.

[8]http://nlp.stanford.edu/software/

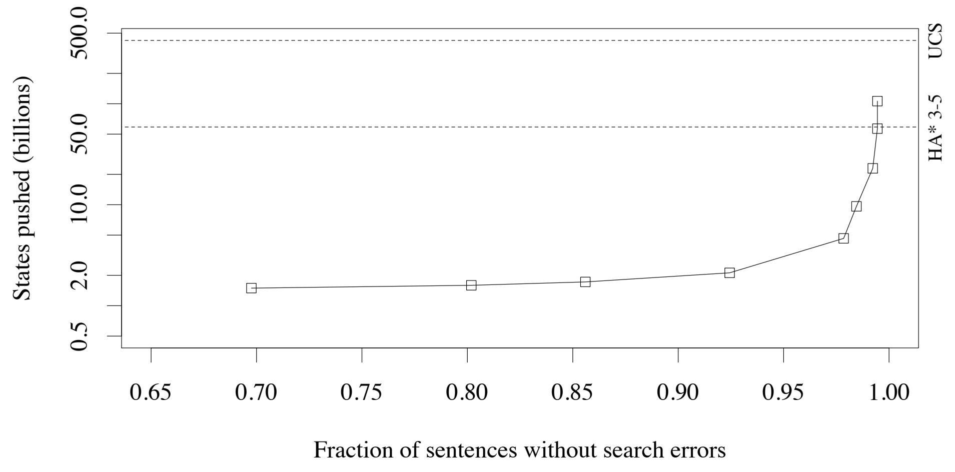

Figure 3.7. Performance of CTF as a function of search errors for state split grammars. The dashed lines represent the time taken by UCS and HA* which make no search errors. As search accuracy increases, the time taken by CTF increases until it eventually becomes slower than HA*. The y-axis is a log scale.

projections as a consistent heuristic in the combined lexicalized grammar. The generalization of HA* to the factored case is straightforward but not effective. We therefore treated the dependency projection as a black box and used only the PCFG projection inside the HA* framework. When computing A* outside estimates in the combined space, we use the sum of the two projections' outside scores as our completion costs. This is the same procedure as Klein and Manning (2003a). For CTF, we carry out a uniform cost search in the combined space where we have pruned items based on their max-marginals in the PCFG model only.

In Figure 3.8, we examine the speed/accuracy trade off for the lexicalized grammar. The trend here is the reverse of the result for the state split grammars: HA* is always faster than posterior pruning, even for thresholds which produce many search errors. This is because the heuristic used in this model is actually an extraordinarily tight bound – on average, the slack even for spans of length 1 was less than 1% of the overall model cost. Thus, for this particular model, we see that applying informed but optimal search procedures can produce significant speed ups over even approximate search schemes.

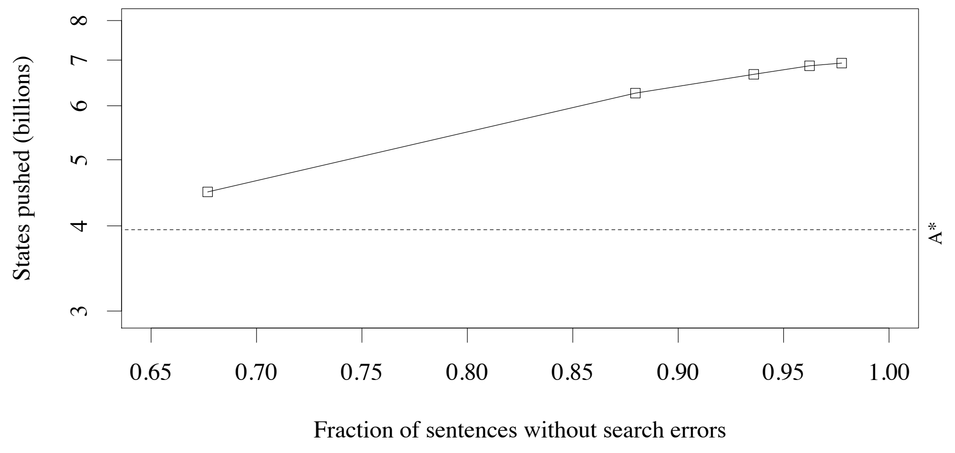

Figure 3.8. Performance of CTF for lexicalized parsing as a function of search errors. The dashed line represents the time taken by A*, which makes no search errors. The y-axis is a log scale.

Chapter 4

Hierarchical Search with Bridge Scores

The fundamental insight of the HA* algorithm is that outside scores from a previous level can be used as completion costs for inside states, and inside costs from the same level can be used as completion costs for outside states. The key property of these completion costs is that they form consistent and admissible heuristic costs, they are not the only quantities that satisfy this requirement.

In this section, we discuss a modification of HA* that can compute **bridge** outside scores, which are bounds on true outside costs that are tighter than completely coarse outside scores used by HA*. These bridge outside scores mix inside and outside costs from finer grammars with inside costs from coarser grammars. Because the bridge costs represent tighter estimates of the true outside costs, we expect them to reduce the work of computing inside costs in finer grammars. At the same time, because bridge costs mix computation from coarser and finer levels of the hierarchy, they are more expensive to compute than purely coarse outside costs. Whether the work saved by using tighter estimates outweighs the extra computation needed to compute them is an empirical question.

Experimentally, we find that the use of bridge outside costs substantially outperforms the HA* algorithm when the coarsest levels of the hierarchy are very loose approximations of the target grammar. For hierarchies with tighter estimates, we show that BHA* obtains comparable performance to HA*. In other words, BHA* is more robust to poorly constructed hierarchies.

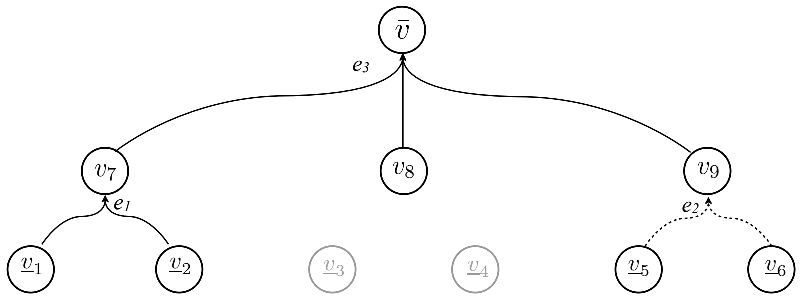

Figure 4.1. A pictorial representation of the bridge outside score for the vertex v_8. The outside path $\ddot{\rho}(v_8) = \{e_1, e_2, e_3\}$ has bridge outside score $\tilde{w}(\rho(v_8)) = w(e_1) + w(\pi(e_2)) + w(e_3)$. The hyperedge e_2 (shown with dotted lines) is scored with its coarse weight because it is a right cousin of v_8.

4.1 Bridge Scores

The outside scores computed by HA* are useful for prioritizing computation in more refined grammars. As an alternative, we propose a novel **bridge outside cost** $\tilde{\alpha}(v^m)$. Intuitively, this cost represents the cost of the best path to the root where rules "above" and "left" of a vertex v come from edges at level m, and rules "below" and "right" of v^m come from level $m-1$; see Figure 4.1 for a graphical depiction. More formally, given an outside path $\ddot{\rho}(v^m)$, let the **right cousins** $rc(\ddot{\rho}(v))$ of v^m in $\ddot{\rho}(v^m)$ be the set of all edges e^m such that $h(e^m)$ is neither a descendant nor ancestor of v^m and $h(e^m)$ lies to the right of v^m in $\ddot{\rho}(v^m)$. We define the **bridge outside weight** $\tilde{w}(\ddot{\rho}(v^m))$ as

$$\tilde{w}(\ddot{\rho}(v^m)) = \sum_{e^m \in rc(\ddot{\rho}(v^m))} w(e^{m-1}) + \sum_{e^m \in \ddot{\rho}(v^m) - rc(\ddot{\rho}(v^m))} w(e^m)$$

and the **bridge Viterbi outside cost** as $\tilde{\alpha}(v^m)$ as

$$\tilde{\alpha}(v^m) = \min_{\ddot{\rho}(v^m)} \tilde{w}(\ddot{\rho}(v^m))$$

.

Like ordinary outside costs, bridge outside costs form consistent and admissible estimates of the true

Name	Rule					
IN-BASE			$\tilde{O}(s_i^m, i, i+1) : w$	$\xrightarrow{w}$	$I(s_i^m, i, i+1)$	$: 0$
IN	$\tilde{O}(A^m, i, j) : w_A$	$I(B^m, i, l) : w_B$	$I(C^m, l, j) : w_C$	$\xrightarrow{w_B+w_C+w_r+w_A}$	$I(A^m, i, j)$	$: w_B + w_C + w_r$
OUT-BASE			$I(R^m, 0, n) : w$	$\xrightarrow{w}$	$\tilde{O}(R^m, 0, n)$	$: 0$
OUT-L	$\tilde{O}(A^m, i, j) : w_A$	$I(B^{m-1}, i, l) : w_B$	$I(C^{m-1}, l, j) : w_C$	$\xrightarrow{w_A+w_C+w_r+w_B}$	$\tilde{O}(B^m, i, l)$	$: w_A + w_C + w_r$
OUT-R	$\tilde{O}(A^m, i, j) : w_A$	$I(B^m, i, l) : w_B$	$I(C^{m-1}, l, j) : w_C$	$\xrightarrow{w_A+w_B+w_r+w_C}$	$\tilde{O}(C^m, l, j)$	$: w_A + w_B + w_r$

Table 4.1. Deduction rules for BHA* parsing. The rule r is in all cases $A^m \to B^m\ C^m$.

Viterbi outside score $\alpha(v)$ of a vertex v. Because bridge costs mix rules from the finer and coarser grammar, bridge costs are at least as good an estimate of the true outside score as entirely coarse outside costs, and will in general be much tighter. That is, we have

$$\alpha(v^{m-1}) \leq \tilde{\alpha}(v^m) \leq \alpha(v^m)$$

In particular, note that the bridge costs become better approximations farther right in the sentence, and the bridge cost of the last word in the sentence is equal to the Viterbi outside cost of that word.

To compute bridge outside costs in the special case of parsing, we introduce bridge outside items $\tilde{o} = \tilde{O}(A^m, i, j)$(b). The deduction rule schemata which build both inside items and bridge outside items are shown in Table 4.1 and graphically in Figure 4.2. The rules are very similar to those which define HA*, but there are two important differences. First, inside states wait for bridge outside items *at the same level*, while outside states wait for inside items from the previous level. Second, the left and right outside deductions are no longer symmetric – bridge outside states can extended to the left given two coarse inside states, but can only be extended to the right given an exact inside state on the left and coarse inside state on the right.

4.2 Theoretical Guarantees

These deduction rules come with guarantees analogous to those of HA*. The monotonicity guarantee ensures that inside and (bridge) outside items are processed in order of:

$$\hat{p}(\iota^m) = \beta(\iota^m) + \tilde{\alpha}(\check{\iota}^m)$$

$$\hat{p}(\tilde{o}^m) = \tilde{\alpha}(\tilde{o}^m) + \beta(\check{\tilde{o}}^{m-1})$$

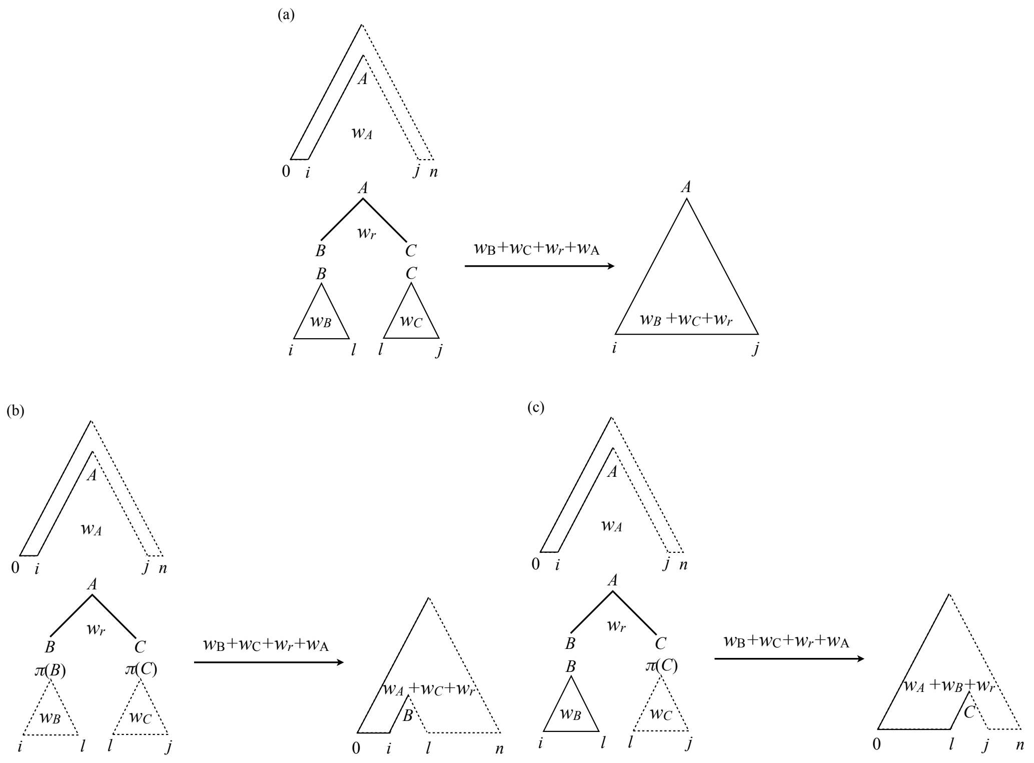

Figure 4.2. Non-base case deductions for BHA* depicted graphically. (a) shows the IN schema, (b) shows OUT-L, and (c) shows OUT-R. Inside states are depicted as complete triangles, while outside states are depicted as chevrons. A state from a previous level in the hierarchy is denoted with dashed lines.

The correctness guarantee ensures that when an item is removed from the agenda, its weight will be equal to $\beta(\iota^m)$ for inside items and $\tilde{\alpha}(\tilde{o}^m)$ for bridge items. The efficiency guarantee remains the same as HA*, though because the intrinsic priorities are different, the set of items processed will be different from those processed by HA*.

A proof of these guarantee is provided in Appendix A.3. The proof for BHA* follows the proof for HA* with minor modifications. The key property of HA* needed for these proofs is that coarse outside costs form consistent and admissible heuristics for inside states, and exact inside costs form consistent and admissible heuristics for outside states. BHA* also has this property, with bridge outside costs forming admissible and consistent heuristics for inside states, and coarse inside costs forming admissible and consistent heuristics for outside states.

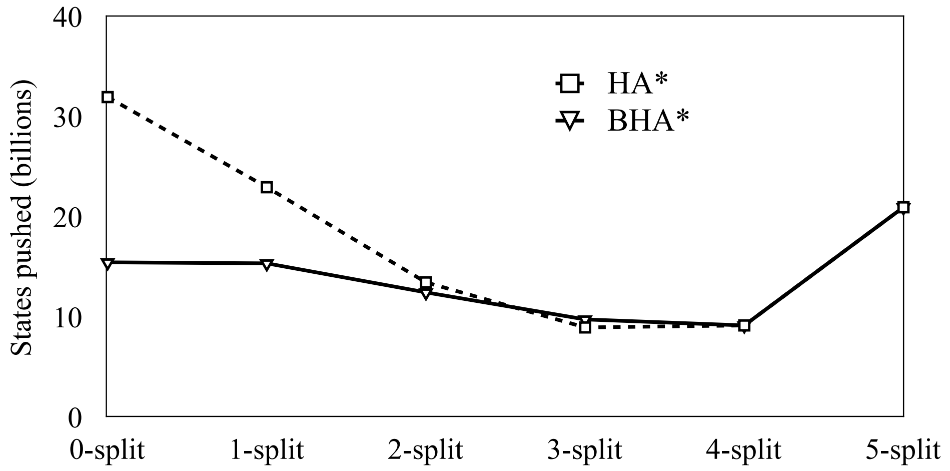

Figure 4.3. Performance of HA* and BHA* as a function of increasing refinement of the coarse grammar. Performance is measured in terms of number of states pushed onto the agenda, so lower is faster.

4.3 Experiments

The performance of BHA* is determined by the efficiency guarantee given in the previous section. However, we cannot determine in advance whether BHA* will be faster than HA*. In fact, BHA* has the potential to be slower – BHA* builds both inside and bridge outside states under the target grammar, where HA* only builds inside states. It is an empirical, grammar- and hierarchy-dependent question whether the increased tightness of the outside estimates outweighs the additional cost needed to compute them. We demonstrate empirically in this section that for hierarchies with very loosely approximating coarse grammars, BHA* can outperform HA*, while for hierarchies with good approximations, performance of the two algorithms is comparable.

We performed experiments with the grammars of Petrov et al. (2006). The training procedure for these grammars produces a hierarchy of increasingly refined grammars through state-splitting, so a natural projection function $\pi^m(\cdot)$ is given. We used the Berkeley Parser[1] to learn such grammars from Sections 2-21 of the Penn Treebank (Marcus et al., 1993). We trained with 6 split-merge cycles, producing 7 grammars. We tested these grammars on 300 sentences of length ≤ 25 of Section 23 of the Treebank. Our "target grammar" was in all cases the most split grammar.

[1]http://berkeleyparser.googlecode.com

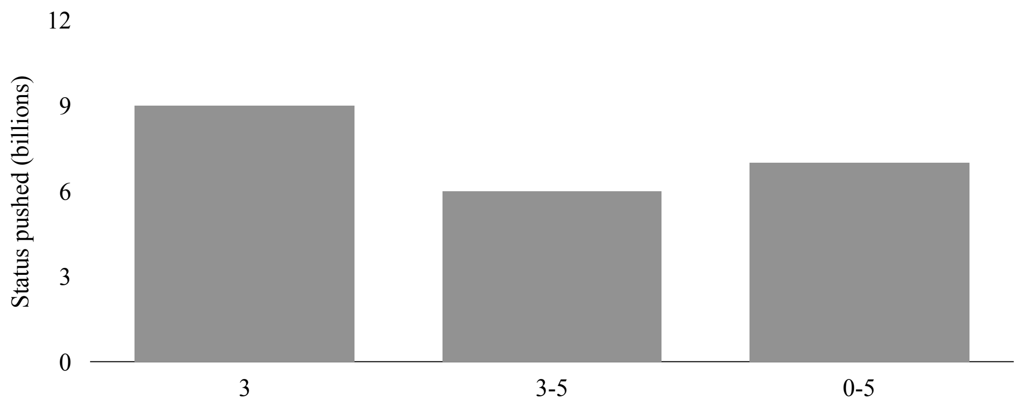

Figure 4.4. Performance of BHA* on hierarchies of varying size, measured in terms of number of states pushed onto the agenda. Along the x-axis, we show which coarse grammars were used in the hierarchy. For example, 3-5 indicates the 3-,4-, and 5-split grammars were used as auxiliary grammars.

In our first experiment, we construct 2-level hierarchies consisting of one coarse grammar and the target grammar. By varying the coarse grammar from the 0-split (X-bar) through 5-split grammars, we can investigate the performance of each algorithm as a function of the coarseness of the coarse grammar. We follow Pauls and Klein (2009b) in using the number of items pushed as a machine- and implementation-independent measure of speed. In Figure 4.3, we show the performance of HA* and BHA* as a function of the total number of items pushed onto the agenda. We see that for very coarse approximating grammars, BHA* substantially outperforms HA*, but for more refined approximating grammars the performance is comparable, with HA* slightly outperforming BHA* on the 3-split grammar.

Finally, we verify that BHA* can benefit from multi-level hierarchies as HA* can. We constructed two multi-level hierarchies: a 4-level hierarchy consisting of the 3-,4-,5-, and 6- split grammars, and 7-level hierarchy consisting of all grammars. In Figure 4.4, we show the performance of BHA* on these multi-level hierarchies, as well as the best 2-level hierarchy from the previous experiment. Our results echo the results of Pauls and Klein (2009b): although the addition of the reasonably refined 4- and 5-split grammars produces modest performance gains, the addition of coarser grammars can actually hurt overall performance.

Chapter 5

K-Best List Extraction

Many situations require a search algorithm to return the k-best paths rather than only the 1-best. Uses for k-best lists include minimum Bayes risk decoding (Goodman, 1998; Kumar and Byrne, 2004), discriminative reranking (Collins, 2000; Charniak and Johnson, 2005), and discriminative training (Och, 2003; McClosky et al., 2006). The most efficient known algorithm for k-best parsing (Jiménez and Marzal, 2000; Huang and Chiang, 2005) performs an initial bottom-up dynamic programming pass before extracting the k-best parses. In that algorithm, the initial pass is, by far, the bottleneck (Huang and Chiang, 2005).

In this chapter, we describe an extension of A* parsing which integrates k-best search with an A*-based exploration of the 1-best chart. A* parsing can avoid significant amounts of computation by guiding 1-best search with heuristic estimates of parse completion costs, and has been applied successfully in several domains (Klein and Manning, 2002, 2003c; Haghighi et al., 2007). Our algorithm extends the speed-ups achieved in the 1-best case to the k-best case and is optimal under the same conditions as a standard A* algorithm. The amount of work done in the k-best phase is no more than the amount of work done by the algorithm of Huang and Chiang (2005). Our algorithm is also equivalent to standard A* parsing (up to ties) if it is terminated after the 1-best derivation is found. Finally, our algorithm can be written down in terms of deduction rules, and thus falls into the well-understood view of parsing as weighted deduction (Shieber et al., 1995; Goodman, 1998; Nederhof, 2003). In fact, our algorithm can be understood as a special case of HA*. In addition to presenting the algorithm, we show experiments in which we extract k-best lists for three different kinds of grammars: the lexicalized grammars of Klein and Manning (2003a), the state-split grammars of Petrov et al. (2006), and the tree transducer grammars of Galley et al. (2006). We demonstrate that optimal k-best lists can be extracted significantly faster using our algorithm than with previous methods.

5.1 A Naive k-Best A* Algorithm

Due to the optimal substructure property of hypergraphs, efficient 1-best search algorithms search over a space of states representing optimal paths to vertices, rather than the space of paths. This is the essence of 1-best dynamic programming – although most vertices can be reached by many paths, a state representing each vertex will be popped exactly once during parsing, with a score and backpointers representing the 1-best path to that vertex.

However, k-best lists necessarily involve searching over suboptimal paths, so any k-best search algorithm must at some point consider suboptimal paths to vertices. One way to compute k-best paths is therefore to abandon optimal substructure and dynamic programming entirely, and to search over space of paths. That is, we can run A* search over states that represent inside paths $\rho(v)$ to a vertex v. In this expanded search space, each distinct path has its own state, derivable only in one way.

If we continue to search long enough, we will pop multiple complete paths to the root. The monotonicity guarantee of A* ensures that the first k complete paths that come off the agenda will be the k-best paths. Because we are using A*, we can include a heuristic to speed up search. Because of the context freedom of the grammar, any consistent heuristic for 1-best A* is also consistent for this search over paths.

We refer to this approach as NAIVE. While correct, NAIVE is massively inefficient. In comparison with A* parsing over G, where there are $O(n^2)$ inside states, the size of the derivation space is exponential in the length of the input.

This naive algorithm is, of course, not novel, either in general approach or specific computation. Early k-best parsers functioned by abandoning dynamic programming and performing beam search on derivations (Ratnaparkhi, 1999; Collins, 2000). Huang (2005) proposes an extension of Knuth's algorithm (Knuth,

Inside Vertex Deductions (Used in A* and KA*)

$$\textbf{IN:}\quad I(B,i,l):w_1 \qquad I(C,l,j):w_2 \qquad \xrightarrow{w_1+w_2+w_r+\hat{\alpha}(A,i,j)} \qquad I(A,i,j) \quad : \quad w_1+w_2+w_r$$

Table 5.1. The deduction schema (IN) for building inside edge items, using a supplied heuristic. This schema is sufficient on its own for 1-best A*, and it is used in KA*. Here, r is the rule $A \rightarrow B\ C$.

Inside Derivation Deductions (Used in NAIVE)

$$\textbf{DERIV:}\quad D(T_B,i,l):w_B \qquad D(T_C,l,j):w_C \qquad \xrightarrow{w_B+w_C+w_r+\hat{\alpha}(A,i,j)} \qquad D\left(\begin{array}{c}\text{A}\\ \diagup\diagdown\\ \text{T}_B\ \ \text{T}_C\end{array},i,j\right) \quad : \quad w_1+w_2+w_r$$

Table 5.2. The deduction schema for building derivations, using a supplied heuristic. T_B and T_C denote full tree structures rooted at symbols B and C. This schema is the same as the IN deduction schema, but operates on the space of fully specified inside derivations rather than dynamic programming edges. This schema forms the NAIVE k-best algorithm.

Outside Edge Deductions (Used in KA*)

OUT-B:	$I(R,0,n):w$			$\xrightarrow{w}$	$O(R,0,n)$	:	0
OUT-L:	$O(A,i,j):w_A$	$I(B,i,l):w_B$	$I(C,l,j):w_C$	$\xrightarrow{w_A+w_C+w_r+w_B}$	$O(B,i,l)$	:	$w_A+w_C+w_r$
OUT-R:	$O(A,i,j):w_A$	$I(B,i,l):w_B$	$I(C,l,j):w_C$	$\xrightarrow{w_A+w_B+w_r+w_C}$	$O(C,l,j)$	:	$w_A+w_B+w_r$

Table 5.3. The deduction schemata for building ouside edge items. The first schema is a base case that constructs an outside item for the goal $(G,0,n)$ from the inside item $I(G,0,n)$. The second two schemata build outside items in a top-down fashion. Note that for outside items, the completion cost is the weight of an inside item rather than a value computed by a heuristic.

Delayed Inside Derivation Deductions (Used in KA*)

$$\textbf{DERIV:}\quad O(A,i,j):w_A \quad D(T_B,i,l):w_B \quad D(T_C,l,j):w_C \quad \xrightarrow{w_B+w_C+w_r+w_A} \quad D\left(\begin{array}{c}\text{A}\\ \diagup\diagdown\\ \text{T}_B\ \ \text{T}_C\end{array},i,j\right) \ : \ w_B+w_C+w_r$$

Table 5.4. The deduction schema for building derivations, using exact outside scores computed using OUT deductions. The dependency on the outside item $O(A,i,j)$ delays building derivation items until exact Viterbi outside scores have been computed. This is the final search space for the KA* algorithm.

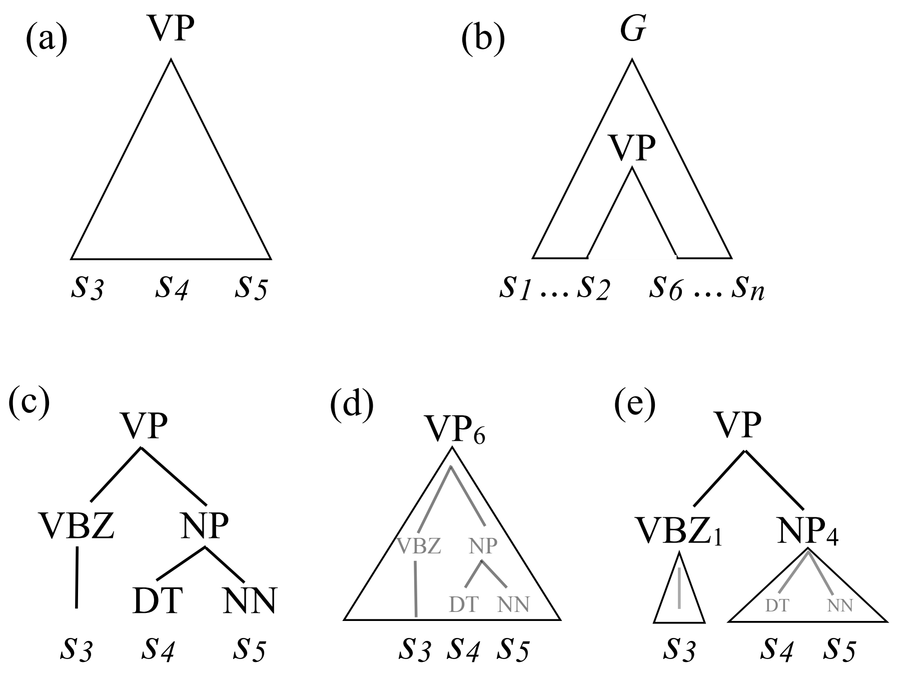

Figure 5.1. Representations of the different types of items used in parsing. (a) An inside edge item: $I(\text{VP}, 2, 5)$. (b) An outside edge item: $O(\text{VP}, 2, 5)$. (c) An inside derivation item: $D(T_{\text{VP}}, 2, 5)$ for a tree T_{VP}. (d) A ranked derivation item: $K(\text{VP}, 2, 5, 6)$. (e) A modified inside derivation item (with backpointers to ranked items): $D(\text{VP}, 2, 5, 3, \text{VP} \rightarrow \text{VBZ NP}, 1, 4)$.

1977) to produce k-best lists by searching in the space of derivations, which is essentially this algorithm. While Huang (2005) makes no explicit mention of a heuristic, it would be easy to incorporate one into their formulation.

5.1.1 NAIVE Search for Parsing

Although NAIVE is an intractable algorithm, it is a very useful tool for understanding our fool algorithm. For concreteness, we describe it in detail here. As in Chapter 3, we will specialize the algorithm to parsing for added clarity.

NAIVE searches over states that we call *derivation states*, and are of the form $d = D(T_A, i, j)$, specifying an entire parse tree T_A rooted at symbol A and spanning $s_{i+1} \dots s_j$ (see Figure 5.1(c)). Derivation states are combined using the DERIV schema of Table 5.2. The goals in this space, representing root parses, are any derivation states rooted at symbol R that span the entire input.

We refer to the weight of a derivation state d as $\delta(d)$, analogous to $\beta(\iota)$ for (1-best) inside states.[1] By the monotonicity property of A*, we know that NAIVE will pop all derivation states d with

$$\delta(d) + \hat{\alpha}(d) \leq \delta(d_{R_k})$$

where d_{R_k} is derivation state corresponding to the kth-best root parse. Even for reasonable heuristics, this number can be very large; see Section 5.4 for empirical results.

5.2 A New k-Best A* Parser

While NAIVE suffers severe performance degradation for loose heuristics, we make the observation that if $\hat{\alpha}(\cdot)$ is the *perfect* heuristic ($\hat{\alpha}(d) = \alpha(d)$), then the search is actually very efficient. In particular, the search will only pop derivation states satisfying

$$\delta(d) + \alpha(\check{d}) \leq \delta(d_{R_k})$$

where $\check{d}$ is the outside state corresponding to the derivation state d. In fact, the set of derivation items d satisfying this inequality is exactly the set which appear in the k-best complete paths (as always, modulo ties). NAIVE would therefore be quite efficient if we could obtain exact Viterbi outside scores $\alpha(\cdot)$.

One option is to compute outside scores with exhaustive dynamic programming over the original grammar. In a certain sense, described in greater detail below, this precomputation of exact heuristics is equivalent to the k-best extraction algorithm of Huang and Chiang (2005). However, this exhaustive 1-best work is precisely what we want to use A* to avoid.

Our algorithm solves this problem by integrating three searches into a single agenda-driven process. First, an A* search in the space of inside states with an (imperfect) external heuristic $\hat{\alpha}(\cdot)$ finds exact inside scores. Second, exact outside scores are computed from inside and outside states. Finally, these exact

[1]The new symbol emphasizes that δ scores a specific path rather than a minimum over a set of paths. Note that unlike standard A*, because each state d represents a unique path, it is easy to see that the weight of the state $w(d)$ is always the same as the weight of the derivation $\delta(d)$ that d represents.

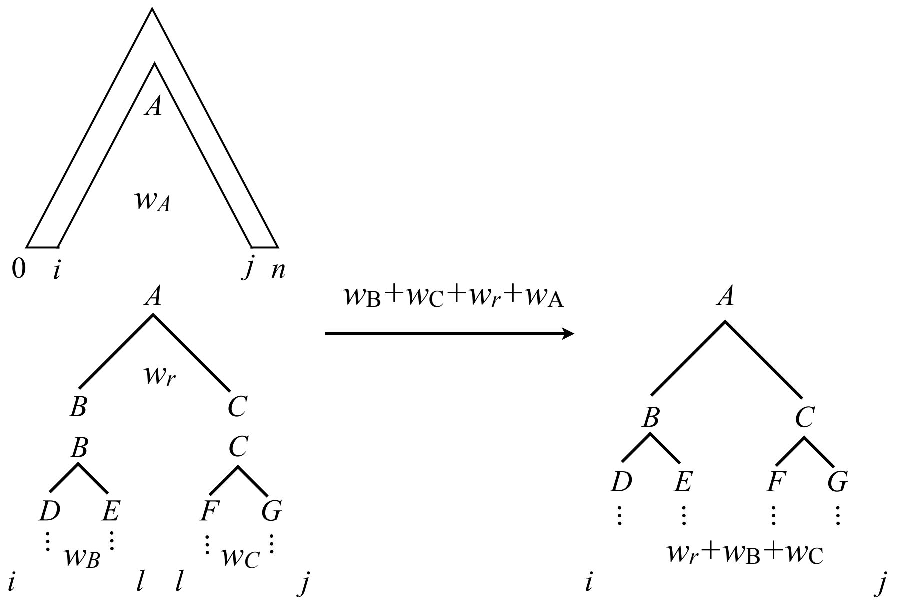
A
w_A
0
i
j
n
A
w_r
B
C
B
C
D
E
F
G
w_B
w_C
i
l
l
j
$w_{\mathrm{B}}+w_{\mathrm{C}}+w_r+w_{\mathrm{A}}$
A
B
C
D
E
F
G
$w_r+w_{\mathrm{B}}+w_{\mathrm{C}}$
i
j

Figure 5.2. The delayed DERIV deduction rule schema.

outside scores guide the search over derivation states. It can be useful to imagine these three operations as operating in phases, but they are all interleaved, progressing in order of their various priorities.

These computations can be performed by the HA* algorithm of Chapter 3. In particular, we note that the space of inside states is a relaxed projection of the space of derivation states. Therefore, we can run HA* on a two-level hierarchy in which the finest level is the derivation space and the coarsest level is inside state space; we provide more details in Section 5.2.2. In fact, because our algorithm is a special case of HA*, we need not limit ourselves to this two-level hierarchy – further relaxed projections of the inside state space can be used to further speed the search.

5.2.1 Deduction Rules

Although our algorithm is a special case of HA*, we describe it here as a stand-alone algorithm for clarity. Table 5.1 and Table 5.3 shows the IN and OUT deduction schemata for building inside and outside items, familiar from HA*. In Table 5.4 and Figure 5.2, we show a modified DERIV deduction schema that delays the construction of a derivation state until its corresponding outside state has been popped.

This ensures that derivation states are only explored once their perfect heuristics (outside scores) have been computed.

We call the algorithm which executes these rules KA*. In words, it functions as follows: we initialize the agenda with the terminal states $I(\mathrm{s}_i, i, i+1)$ and $D(\mathrm{s}_i, i, i+1)$ for $i = 0 \dots n-1$. We compute inside scores in standard A* fashion using the IN deduction schema, using any heuristic we might provide to 1-best A*. Once the inside state $I(R, 0, n)$ is found, we automatically begin to compute outside scores via the OUT deduction schema. Once $O(\mathrm{s}_i, i, i+1)$ is found, we can begin to also search in the space of derivation items, using the perfect heuristics given by the just-computed outside scores. Note, however, that all computation is done with a single agenda, so the processing of all three types of items is interleaved, with the k-best search possibly terminating without a full inside computation. As with NAIVE, the algorithm terminates when a k-th root derivation is dequeued.

5.2.2 Correctness

We prove the correctness of this algorithm by reduction to the to HA*; details can be found in Appendix A.4. The fact that KA* can be reduced to an instance of HA* has advantages in addition to the fact that the proof is simple. First, the algorithm is naturally online in the sense that it can be stopped at any k without advance specification, a fact which follows from the monotonicity guarantee of HA*. Second, the algorithm can be easily exploit of hierarchy of relaxed projections because it is already an instance of HA*.

In terms of efficiency, we can characterize the amount of work done using the monotonicity guarantee. Let d_{R_k} be the kth-best derivation item for the goal edge g. Our algorithm processes all derivation states d, outside states o, and inside states ι satisfying

$$\begin{aligned}
\delta(d) + \alpha(\check{d}) &\leq \delta(d_{R_k}) \\
\beta(\check{o}) + \alpha(o) &\leq \delta(d_{R_k}) \\
\beta(\iota) + \hat{\alpha}(\iota) &\leq \delta(d_{R_k})
\end{aligned}$$

We have already argued that the set of derivation items satisfying the first inequality is the set of subtrees that appear in the optimal k-best parses, modulo ties. Similarly, it can be shown that the second inequality is satisfied only for states that appear in the optimal k-best parses. The last inequality characterizes the amount of work done in the bottom-up pass. We compare this to 1-best A*, which pops all inside states i satisfying

$$\beta(\iota) + \hat{\alpha}(\iota) \leq \beta(\iota_R) = \delta(d_{R_1})$$

Thus, the "extra" inside states popped in the bottom-up pass during k-best parsing as compared to 1-best parsing are those items i satisfying

$$\delta(d_{R_1}) \leq \beta(\iota) + \hat{\alpha}(\iota) \leq \delta(d_{R_k})$$

The question of how many states satisfy these inequalities is empirical; we show in our experiments that it is small for reasonable heuristics. At worst, the bottom-up phase pops all inside items and reduces to exhaustive dynamic programming.

5.2.3 Lazy Successor Functions

The monotonicity guarantee ensures that we will only dequeue derivation fragments of top parses. However, we will *enqueue* all combinations of such items, which is wasteful. By exploiting a local ordering amongst derivations, we can be more conservative about combination and gain the advantages of a lazy successor function (Huang and Chiang, 2005).

To do so, we represent inside derivations not by explicitly specifying entire trees, but rather by using **ranked backpointers**. In this representation, inside derivations are represented in two ways, shown in Figure 5.1(d) and (e). The first way (d) simply adds a rank a to an edge, giving a tuple (A, i, j, a). The corresponding item is the *ranked derivation item* $K(A, i, j, a)$, which represents the ath-best derivation of A over (i, j). The second representation (e) is a backpointer of the form (A, i, j, l, r, a, b), specifying the derivation formed by combining the ath-best derivation of (B, i, l) and the bth-best derivation of (C, l, j) using rule $r = A \to B\ C$. The corresponding items $D(A, i, j, l, r, a, b)$ form of our inside derivation items.

The modified deduction schemata for the NAIVE algorithm over these representations are shown in Table 5.5. The BUILD schema produces new inside derivation items from ranked derivation items, while the RANK schema assigns each derivation item a rank; together they function like DERIV. We can find the

k-best list by searching until $K(G, 0, n, k)$ is removed from the agenda. The k-best derivations can then be extracted by following the backpointers for $K(G, 0, n, 1)$ $\dots$ $K(G, 0, n, k)$. The KA* algorithm can be modified in the same way, shown in Table 5.6.

The actual laziness is provided by additionally delaying the combination of ranked items. When an item $K(B, i, l, u)$ is popped off the queue, a naive implementation would loop over items $K(C, l, j, v)$ for all v, C, and j (and similarly for left combinations). Fortunately, little looping is actually necessary: there is a partial ordering of derivation items, namely, that $D(A, i, j, l, r, a, b)$ will have a lower computed priority than $D(A, i, j, l, r, a - 1, b)$ and $D(A, i, j, l, r, a, b - 1)$ (Jiménez and Marzal, 2000). So, we can wait until one of the latter two is built before "triggering" the construction of the former. This triggering is similar to the "lazy frontier" used by Huang and Chiang (2005). All of our experiments use this lazy representation.

5.3 Discussion

5.3.1 Mixing Dynamic Programming and KA*

One appealing aspect of the HA* algorithm that KA* inherits is its indifference to how antecedents on the chart are computing when applying deduction rules. As long as all antecedents on the chart have their correct scores, HA* (and hence KA*) will operate correctly. This means that KA* can be "warm-started" by, for example, pre-computing inside scores exhaustively by dynamic programming, before initiating an agenda-based search over outside states and derivation states.

Note that this pre-computation of inside scores might compute scores for states that HA* would not consider, and hence do some extra work in terms of states considered. In fact, if a good heuristic $\hat{\alpha}(\cdot)$ is available, the work saved by running the agenda-based search can be quite significant. However, for heuristics that offer only loose bounds on the true outside costs, the work saved might be small. Furthermore, it is often the case that low-level optimization is much easier in dynamic programs, so that dynamic programming can actually be faster in terms of CPU time, despite doing more work overall. In such cases, pre-computing inside scores may be faster overall.

5.3.2 Related Work

While formulated very differently, one limiting case of our algorithm relates closely to the EXH algorithm of Huang and Chiang (2005). In particular, if all inside items are processed before any derivation items (or are computed using dynamic programming as discussed above), the subsequent number of derivation items and outside items popped by KA* is nearly identical to the number popped by EXH in our experiments – both algorithms have the same ordering bounds on which derivation items are popped. The conceptual difference between the algorithms in this limited case is that EXH places k-best items on local priority queues per edge, while KA* makes use of one global queue. Thus, in addition to providing a method for speeding up k-best extraction with A*, our algorithm also provides an alternate form of Huang and Chiang (2005)'s k-best extraction that can be phrased in a weighted deduction system.

5.4 Experiments

5.4.1 State-Split Grammars

We performed our first experiments with the grammars of Petrov et al. (2006). The training procedure for these grammars produces a hierarchy of increasingly refined grammars through state-splitting. We followed Pauls and Klein (2009b) in computing heuristics for the most refined grammar from outside scores for less-split grammars.

We used the Berkeley Parser[2] to learn such grammars from Sections 2-21 of the Penn Treebank (Marcus et al., 1993). We trained with 6 split-merge cycles, producing 7 grammars. We tested these grammars on 100 sentences of length at most 30 of Section 23 of the Treebank. Our "target grammar" was in all cases the most split grammar.

Heuristics computed from projections to successively smaller grammars in the hierarchy form successively looser bounds on the outside scores. This allows us to examine the performance as a function of the

[2]http://berkeleyparser.googlecode.com

Ranked Inside Derivation Deductions (Lazy Version of NAIVE)

BUILD:	$K(B,i,l,a) : w_B$	$K(C,l,j,b) : w_C$	$\xrightarrow{w_B+w_C+w_r+\hat{\alpha}(A,i,j)}$	$D(A,i,j,l,r,a,b)$	:	$w_B + w_C + w_r$
RANK:	$D_1(A,i,j,\cdot) : w_1 \ldots$	$D_k(A,i,j,\cdot) : w_k$	$\xrightarrow{\min_m w_m+\hat{\alpha}(A,i,j)}$	$K(A,i,j,k)$	:	$\min_m w_m$

Table 5.5. The schemata for simultaneously building and ranking derivations, using a supplied heuristic, for the lazier form of the NAIVE algorithm. BUILD builds larger derivations from smaller ones. RANK numbers derivations for each vertex. Note that RANK requires distinct D_i, so a rank k RANK rule will first apply (optimally) as soon as the kth-best inside derivation item for a given vertex is removed from the queue. However, it will also still formally apply (suboptimally) for all derivation items dequeued after the kth. In practice, the RANK schema need not be implemented explicitly – one can simply assign a rank to each inside derivation item when it is removed from the agenda, and directly add the appropriate ranked inside item to the chart.

Delayed Ranked Inside Derivation Deductions (Lazy Version of KA*)

BUILD:	$O(A,i,j) : w_A$	$K(B,i,l,a) : w_B$	$K(C,l,j,b) : w_C$	$\xrightarrow{w_B+w_C+w_r+w_A}$	$D(A,i,j,l,r,a,b)$	:	$w_B + w_C + w_r$
RANK:	$D_1(A,i,j,\cdot) : w_1 \ldots$	$D_k(A,i,j,\cdot) : w_k$	$O(A,i,j) : w_A$	$\xrightarrow{\min_m w_m+w_A}$	$K(A,i,j,k)$	:	$\min_m w_m$

Table 5.6. The deduction schemata for building and ranking derivations, using exact outside scores computed from OUT deductions, used for the lazier form of the KA* algorithm.

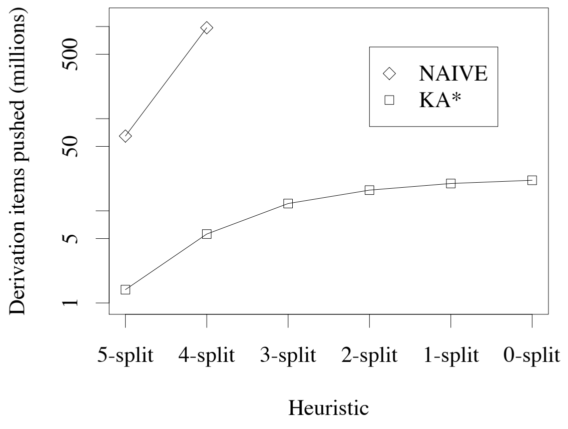

Figure 5.3. Number of derivation items enqueued as a function of heuristic. Heuristics are shown in decreasing order of tightness. The y-axis is on a log-scale.

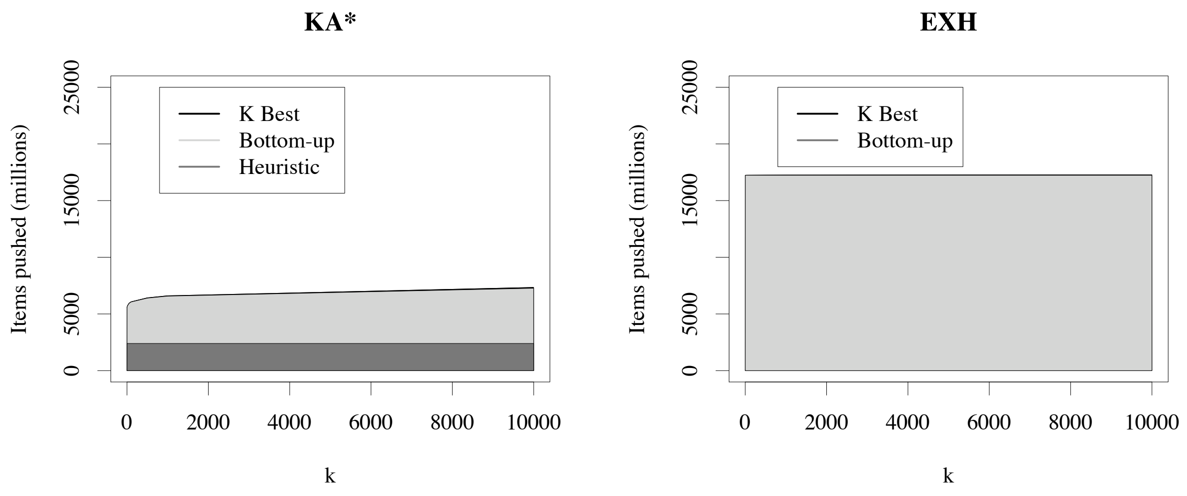

Figure 5.4. The cost of k-best extraction as a function of k for state-split grammars, for both KA* and EXH. The amount of time spent in the k-best phase is negligible compared to the cost of the bottom-up phase in both cases.

tightness of the heuristic. We first compared our algorithm KA* against the NAIVE algorithm. We extracted 1000-best lists using each algorithm, with heuristics computed using each of the 6 smaller grammars.

In Figure 5.3, we evaluate only the k-best extraction phase by plotting the number of derivation items and outside items added to the agenda as a function of the heuristic used, for increasingly loose heuristics. We follow earlier work (Pauls and Klein, 2009b) in using number of states pushed as the primary, hardware-invariant metric for evaluating performance of our algorithms.[3] While KA* scales roughly linearly with the looseness of the heuristic, NAIVE degrades very quickly as the heuristics get worse. For heuristics given by grammars weaker than the 4-split grammar, NAIVE ran out of memory.

Since the bottom-up pass of k-best parsing is the bottleneck, we also examine the time spent in the 1-best phase of k-best parsing. As a baseline, we compared KA* to the approach of Huang and Chiang (2005), which we will call EXH (see below for more explanation) since it requires exhaustive parsing in the bottom-up pass. We performed the exhaustive parsing needed for EXH in our agenda-based parser to facilitate comparison. For KA*, we included the cost of computing the heuristic, which was done by running our agenda-based parser exhaustively on a smaller grammar to compute outside items; we chose the 3-split grammar for the heuristic since it gives the best overall tradeoff of heuristic and bottom-up parsing time.

[3]We found that states pushed was generally well correlated with parsing time.

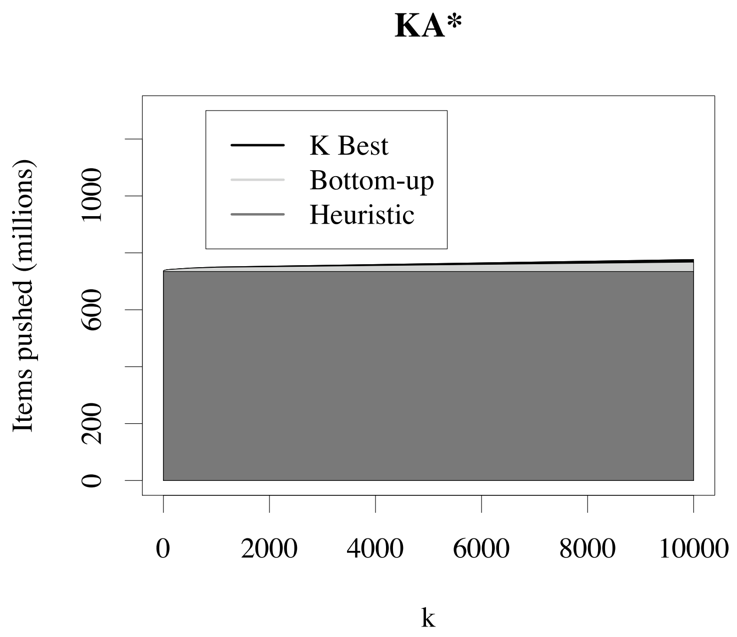

Figure 5.5. The performance of KA* for lexicalized grammars. The performance is dominated by the computation of the heuristic, so that both the bottom-up phase and the k-best phase are barely visible.

We separated the items enqueued into items enqueued while computing the heuristic (not strictly part of the algorithm), inside items (“bottom-up”), and derivation and outside items (together “k-best”). The results are shown in Figure 5.4. The cost of k-best extraction is clearly dwarfed by the the 1-best computation in both cases. However, KA* is significantly faster over the bottom-up computations, even when the cost of computing the heuristic is included.

5.4.2 Lexicalized Parsing

We also experimented with the lexicalized parsing model described in Klein and Manning (2003a). This model is constructed as the product of a dependency model and the unlexicalized PCFG model in Klein and Manning (2003b). We constructed these grammars using the Stanford Parser.[4] The model was trained on Sections 2-20 of the Penn Treebank and tested on 100 sentences of Section 21 of length at most 30 words.

For this grammar, Klein and Manning (2003a) showed that a very accurate heuristic can be constructed

[4]http://nlp.stanford.edu/software/

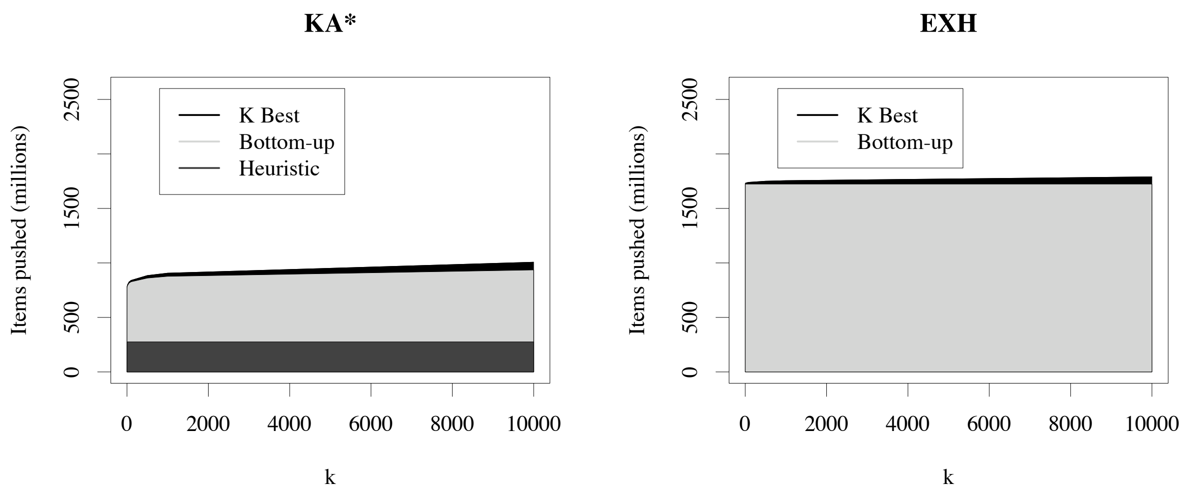

Figure 5.6. The cost of k-best list extraction as a function of k for tree transducer grammars, for both KA* and EXH.

by taking the sum of outside scores computed with the dependency model and the PCFG model individually. We report performance as a function of k for KA* in Figure 5.5. Both NAIVE and EXH are impractical on these grammars due to memory limitations. For KA*, computing the heuristic is the bottleneck, after which bottom-up parsing and k-best extraction are very fast.

5.4.3 Tree Transducer Grammars

Syntactic machine translation (Galley et al., 2004) uses tree transducer grammars to translate sentences. Transducer rules are synchronous context-free productions that have both a source and a target side. We examine the cost of k-best parsing in the source side of such grammars with KA*, which can be a first step in translation.

We extracted a grammar from 220 million words of Arabic-English bitext using the approach of Galley et al. (2006), extracting rules with at most 3 non-terminals. These rules are highly lexicalized. About 300K rules are applicable for a typical 30-word sentence; we filter the rest. We tested on 100 sentences of length at most 40 from the NIST05 Arabic-English test set.

We used a simple but effective heuristic for these grammars, similar to the FILTER heuristic suggested in Klein and Manning (2003c). We projected the source projection to a smaller grammar by collapsing all non-terminal symbols to X, and also collapsing pre-terminals into related clusters. For example, we collapsed

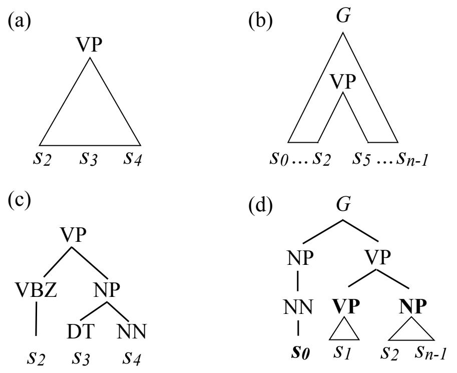

Figure 5.7. Representations of the different types of states used in parsing. (a) An inside edge item $I(\text{VP}, 2, 5)$. (b) An outside edge item $O(\text{VP}, 2, 5)$. (c) An inside derivation item: $D(T^{\text{VP}}, 2, 5)$. (d) An outside derivation item: $Q(T^{R}_{\text{VP}}, 2, 3, \{(\text{NP}, 3, n)\}$. The vertices in boldface are frontier vertices.

the tags NN, NNS, NNP, and NNPS to N. This projection reduced the number of grammar symbols from 149 to 36. Using it as a heuristic for the full grammar suppressed $\sim 60\%$ of the total items (Figure 5.6).

5.5 Top-Down k-Best A*

Although KA* is just at least as efficient as the EXH algorithm of Jiménez and Marzal (2000) and Huang and Chiang (2005) (and much more efficient with an accurate heuristic), it has some extra conceptual baggage. EXH only performs two passes – an exhaustive dynamic programming inside pass, followed by a lazy, top-down k-best extraction pass – while KA* computes has both an inside and outside 1-best pass, followed by a bottom-up k-best pass.

To simplify KA* further, in this section, we describe TKA*, a top-down variant of KA* that, like EXH, performs only an inside pass before extracting k-best lists top-down, but maintains the same optimality and efficiency guarantees as KA*. Eliminating the outside pass makes KA* simpler both in implementation and description.

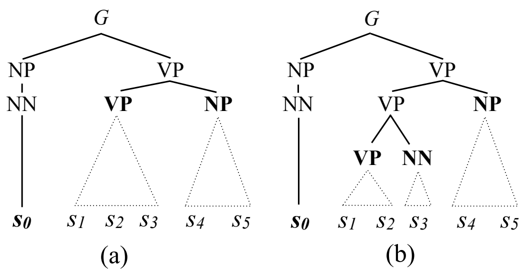

Figure 5.8. Top-down expansion of an outside derivation item. (a) An outside derivation item before expansion at $(\text{VP}, 2, 5)$. (b) The result of expanding the item in (a) using the rule VP→ VB NN. Frontier vertices are marked in boldface.

TKA*

IN:		$I(B,i,l) : w_B$	$I(C,l,j) : w_C$	$\xrightarrow{w_B+w_C+w_r+\hat{\alpha}(A,i,j)}$	$I(A,i,j) : w_B + w_C + w_r$
OUT-D:	$Q(T_A^R, i, j, \mathcal{F}) : w_A$	$I(B,i,l) : w_B$	$I(C,l,j) : w_C$	$\xrightarrow{w_A+w_r+w_B+w_C+\beta(\mathcal{F})}$	$Q(T_B^R, i, l, \mathcal{F}_C) : w_A + w_r$

Table 5.7. The deduction rules used in TKA*. Here, r is the rule $A \rightarrow B\ C$. IN is the standard inside deduction from A*. In OUT-D, the tree T_B^R is the tree T_A^R extended at (A, i, j) with rule r, $\mathcal{F}_C$ is the list $\mathcal{F}$ with (C, l, j) prepended, and $\beta(\mathcal{F})$ is $\sum_{v \in \mathcal{F}} \beta(v)$. Whenever the left child $I(B, i, l)$ of an application of OUT-D represents a terminal, the next vertex is removed from $\mathcal{F}$ and is used as the new point of expansion.

5.5.1 TKA*

KA* efficiently explores the space of derivation states because it waits for the exact Viterbi outside costs before building each derivation state. However, these outside costs and associated deduction states are only auxiliary quantities used to guide the exploration of inside derivations: they allow KA* to prioritize currently constructed inside derivation items (i.e., constructed derivations of the root) by their optimal completion cost. Outside costs are thus only necessary because we construct partial derivations bottom-up; if we constructed partial derivations in a top-down fashion, all we would need to compute optimal completion costs are Viterbi inside scores, and we could forget the outside pass.

TKA* does exactly that. Inside states are constructed in the same way as KA*, but once the inside state $I(R, 0, n)$ has been discovered, TKA* begins building partial derivations from the root outwards. We replace the derivation states of KA* with **outside derivation states**, which represent trees rooted at the root and expanding downwards; we will call derivation states for KA* "inside derivation states" to avoid ambiguity. These outside derivation states bottom out in a list of vertices called the **frontier** vertices. See Figure 5.7(d) for a graphical representation. When a frontier vertex represents a single word in the input, i.e. is of the form $(\mathrm{s}_0^i, i, i+1)$, we say that vertex is **complete**. An outside derivation can be expanded by applying a rule to one of its incomplete frontier vertices; see Figure 5.8. In the same way that inside derivation states wait on exact outside scores before being built, outside derivation states wait on the inside states of all frontier vertices before they can be constructed.

Although building derivations top-down obviates the need for a 1-best outside pass, it raises a new issue. When building derivations bottom-up, the only way to expand a particular inside derivation is to combine it with another inside derivation to build a bigger tree. In contrast, an outside derivation can be expanded anywhere along its frontier. Naively building derivations top-down would lead to an exponential number of expansion choices.

We solve this issue by always expanding the left-most incomplete frontier vertex of an outside derivation state. We show the deduction rule OUT-D which performs this deduction in Figure 5.7. We denote an outside derivation state as $Q(T_A^R, i, j, \mathcal{F})$, where T_A^R is a tree rooted at the root with left-most incomplete vertex (A, i, j), and $\mathcal{F}$ is the list of incomplete frontier vertices excluding (A, i, j). Whenever the application of this rule "completes" the left-most vertex, the next vertex is removed from $\mathcal{F}$ and used as the new point of expansion. Once all frontier vertices are complete, the item represents a correctly scored derivation of the goal, explored in a pre-order traversal.

5.5.2 Implementation Details

Building derivations bottom-up is convenient from an indexing point of view: since larger derivations are built from smaller ones, it is not necessary to construct the larger derivation from scratch. Instead, one can simply construct a new tree whose children point to the old trees, saving both memory and CPU time.

In order keep the same efficiency when building trees top-down, a slightly different data structure is necessary. We represent top-down derivations as a lazy list of expansions. The top node T_G^R is an empty list, and whenever we expand an outside derivation item $Q(T_A^R, i, j, \mathcal{F})$ with a rule $r = A \rightarrow B\ C$ and split point l, the resulting derivation T_B^G is a new list item with (r, l) as the head data, and T_A^R as its tail. The tree can be reconstructed later by recursively reconstructing the parent, and adding the edges (B, i, l) and (C, k, l) as children of (A, i, j).

5.5.3 Correctness

We provide a brief proof of correctness of the more general form of TKA* in Appendix A.5.

5.5.4 Performance

We have argued that TKA* is simpler than TKA*, but we do not expect it to do any more or less work than KA*, modulo grammar specific optimizations. Therefore, in this section, we simply verify that the additional work of extracting k-best lists with TKA* is negligible compared to the time spent building 1-best inside edges.

We examined the time spent building 100-best for the same experimental setup as Pauls and Klein (2009a).[5] On 100 sentences, our implementation of TKA* constructed 3.46 billion items, of which about 2% were outside derivation items. Our implementation of KA* constructed 3.41 billion edges, of which about 0.1% were outside edge items or inside derivation items. In other words, the cost of k-best extraction is dwarfed by the the 1-best inside edge computation in both cases. The reason for the slight performance advantage of KA* is that our implementation of KA* uses lazy optimizations discussed in Pauls and Klein (2009a), and while such optimizations could easily be incorporated in TKA*, we have not yet done so in our implementation.

[5]This setup used 3- and 6-round state-split grammar from Petrov et al. (2006), the former used to compute a heuristic for the latter, tested on sentences of length up to 25.

Chapter 6

Conclusions

This thesis has discussed efficient optimal search techniques for problems that can be formulated as search in a hypergraph. We have provided a novel description and proof of correctness for the Hierarchical A* algorithm of Felzenszwalb and McAllester (2007). We have empirically compared this algorithm to well-known non-optimal search techniques, and found that non-optimal search techniques are much faster unless a very accurate heuristic can be obtained for a particular problem (Section 3.3.3) or the user has a very low tolerance for search errors (Section 3.3.2). We also described a novel variant of Hierarchical A* that is more robust to poor choices of grammar hierarchies.

We have also described two novel A*-based algorithms that can extract k-best lists rather than just the 1-best. We have shown that this algorithm can lead to significant speed ups over existing state-of-the-art k-best extraction algorithms given an appropriate heuristic.

Bibliography

Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

Eugene Charniak, Sharon Goldwater, and Mark Johnson. Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 127–133. Morgan Kaufmann, 1998.

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. Multilevel coarse-to-fine pcfg parsing. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 168–175, Morristown, NJ, USA, 2006. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1220835.1220857.

Michael Collins. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, 2000.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269271, 1959.

P. Felzenszwalb and D. McAllester. The generalized A* architecture. *Journal of Artificial Intelligence Research*, 2007.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-ACL)*, 2004.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio

Thayer. Scalable inference and training of context-rich syntactic translation models. In *The Annual Conference of the Association for Computational Linguistics (ACL)*, 2006.

Giorgio Gallo, Giustino Longo, Sang Nguyen, and Stefano Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3), 1993.

Joshua Goodman. *Parsing Inside-Out*. PhD thesis, Harvard University, 1998.

Aria Haghighi, John DeNero, and Dan Klein. Approximate factoring for A* search. In *Proceedings of HLT-NAACL*, 2007.

P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100107, 1968.

Liang Huang. Unpublished manuscript. `http://www.cis.upenn.edu/~lhuang3/knuth.pdf`, 2005.

Liang Huang. *Forest-Based Algorithms in Natural Language Processing*. PhD thesis, Harvard University, 1998.

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, pages 53–64, 2005.

Víctor M. Jiménez and Andrés Marzal. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, pages 183–192, London, UK, 2000. Springer-Verlag. ISBN 3-540-67946-4.

Dan Klein and Chris Manning. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the penn treebank. In *Proceedings of the Association for Computational Linguistics*, 2001a.

Dan Klein and Chris Manning. Fast exact inference with a factored model for natural language processing. In *Proceedings of NIPS*, 2002.

Dan Klein and Chris Manning. Factored A* search for models over sequences and trees. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003a.

Dan Klein and Chris Manning. Accurate unlexicalized parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2003b.

Dan Klein and Christopher D. Manning. A* parsing: Fast exact Viterbi parse selection. In *Proceedings of the Human Language Technology Conference and the North American Association for Computational Linguistics (HLT-NAACL)*, pages 119–126, 2003c.

Dan Klein and Christopher D. Manning. Parsing and hypergraphs. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, pages 123–134, 2001b.

Donald Knuth. A generalization of Dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical phrase-based translation. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, 2003.

Shankar Kumar and William Byrne. Minimum bayes-risk decoding for statistical machine translation. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2004.

Bruce Lowerre. *The Harpy Speech Recognition System*. PhD thesis, Carnegie Mellon University, 1976.

M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. In *Computational Linguistics*, 1993.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 152–159, 2006.

Mark-Jan Nederhof. Weighted deductive parsing and Knuth's algorithm. *Computationl Linguistics*, 29(1): 135–143, 2003. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/089120103321337467.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics (ACL)*, pages 160–167, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1075096.1075117.

Adam Pauls and Dan Klein. Hierarchical A* parsing with bridge outside scores. In *Proceedings of the Association for Computational Linguistics (Short Paper Track)*, 2010a.

Adam Pauls and Dan Klein. K-best A* parsing. In *Proceedings of the Association for Computational Linguistics*, 2009a.

Adam Pauls and Dan Klein. Hierarchical search for parsing. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2009b.

Adam Pauls and Dan Klein. Top-down k-best A* parsing. In *Proceedings of the Association for Computational Linguistics (Short Paper Track)*, 2010b.

Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, 2007.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proccedings of the Association for Computational Linguistics (ACL)*, 2006.

Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. In *Machine Learning*, volume 34, pages 151–5175, 1999.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

Noah A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May 2011.

Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–404, 1997.

Appendix A

Proof of Correctness of Hierarchical A*

In this appendix, we present proofs of correctness for HA*, BHA*, KA*, and TKA*. Rather than prove the correctness of HA* directly, as was done by Felzenszwalb and McAllester (2007), we show that HA* can be reformulated as a special case of Knuth's algorithm (Knuth, 1977). This simpler and more elegant proof allows us to generalize HA* to allow the use of negative weights and an initial consistent heuristic. The correctness of BHA* can be proven with slight modifications to this proof, and the correctness of KA* and TKA* can be proven by reduction to HA* and BHA*, respectively.

We start by reviewing Knuth's algorithm in Section A.1, and arguing that weights in Knuth's algorithm need not be real numbers, but can be any generalized to any set with a total ordering. Then, we show in Section A.2 that HA* is an instance of Knuth's algorithm in which the weights w are pairs (β, α) ordered by $\beta + \alpha$, where β is an inside score and α is an outside score. The proofs for BHA*, KA*, and TKA* follow in Sections A.3, A.4, and A.5.

A.1 Knuth's Algorithm

Knuth (1977) describes an algorithm for finding the best path in a hypergraph. Although Knuth's original paper was formulated somewhat differently, Knuth's algorithm can be thought of as operating on the weighted deduction rules described in Section 2.2.2. In particular, it operates on weighted deduction rules

ψ for which the priority function $p_\psi(\cdot)$ and the cost function $g_\psi(\cdot)$ are the same, so that each rule ψ has the form

$$\phi_1 : w_1, \ldots, \phi_n : w_n \xrightarrow{g_\psi(w_1,\ldots,w_n)} \phi_0 : g_\psi(w_1, \ldots, w_n)$$

Knuth (1977) assumes that the functions $g_\psi(\cdot)$ are **superior** for all deduction rules. A function $g_\psi(w_1, \ldots, w_n)$ is superior if

$$w'_i \geq w_i \text{ implies } g_\psi(w_1, \ldots, w'_i, \ldots, w_n) \geq g_\psi(w_1, \ldots, w_i, \ldots, w_n)$$

and

$$g_\psi(w_1, \ldots, w_n) \geq \max(w_1, \ldots, w_n)$$

In words, a function is superior if it is monotone non-decreasing in each argument and upper bounds all arguments.

Knuth (1977) shows that for superior functions, execution of the set of deduction rules will always remove each state from the agenda with its minimum possible weight. In other words, Knuth's algorithm establishes the correctness guarantee of Section 2.2.2.

A.1.1 Monotonicity Guarantee for Knuth's Algorithm

Because in Knuth's algorithm, the priority of a state is always the same as its weight, a simple proof also establishes the monotonicity guarantee of Section 2.2.2.

Theorem 1. *During execution of any set of weighted deduction rules with $p_\psi(\cdot) \equiv g_\psi(\cdot)$ and $g_\psi(\cdot)$ superior for all deduction rules ψ, if state ϕ' is removed from the agenda with weight w' after some other state ϕ with weight w, then $w' \geq w$. In other words, states are removed from the agenda in decreasing order of their weight.*

Proof. The proof is by contradiction. Suppose that the monotonicity guarantee is violated, i.e. that some state ϕ' is removed from the agenda with weight w' after some other state ϕ with weight w where $w' < w$. Without loss of generality, suppose ϕ' is the first state for which the monotonicity guarantee is violated, so that the guarantee holds for all states already removed from the agenda. If Let ψ and ψ' be the deduction rules that derived ϕ and ϕ' with their current weights, and let $\phi'_1 \dots \phi'_n$ and $\phi_1 \dots \phi_n$ be their antecedents and $w'_1 \dots w'_n$ and $w_1 \dots w_n$ be their weights.[1] Finally, let $\phi'_{i'}$ and ϕ_i be the last states of $\phi'_1 \dots \phi'_n$ and $\phi_1 \dots \phi_n$ removed from the agenda, respectively, so that as soon as one is removed from the agenda, the rule ψ' or ψ can be applied. Note that by assumption, the monotonicity guarantee holds for the antecedents, so it must be that $w'_i \geq \max(w'_1, \dots, w'_n)$ and $w_i \geq \max(w_1, \dots, w_n)$.

There are two cases. First suppose, $\phi'_{i'}$ was removed from the agenda before ϕ_i. But then ϕ' must have been enqueued before ϕ because ϕ' can be enqueued as soon as $\phi'_{i'}$ is dequeued. Because $w' < w$, and the priority of a state is the same as its weight, ϕ' must have been removed from the agenda before ϕ because it was enqueued first and has a cheaper priority, producing a contradiction.

Now suppose $\phi'_{i'}$ was removed from the agenda after ϕ_i. If $w'_i > w$, then $\phi'_{i'}$ will also be removed after ϕ. But because $g_{\psi'}(\cdot)$ is superior, $w' \geq w'_i > w$ because $w' = g_{\psi'}(w'_1, \dots, w'_i, \dots, w'_n) \geq \max(w'_1, \dots, w'_i, \dots, w'_n) \geq w'_i > w$, which contradicts $w' < w$. If, on the other hand, $w'_i \leq w$, then $\phi'_{i'}$ will be removed from the agenda before ϕ. Thus, ϕ' is placed on the agenda with priority w' while ϕ is still on the agenda with priority w. Since $w' < w$, ϕ' must be removed before ϕ, producing a contradiction. $\square$

A.1.2 Knuth's Algorithm with Generalized Weights

Although Knuth's algorithm assumes that states all weights are real, non-negative numbers, the proof generalizes without modification to weights of any type as long as a total ordering $\succeq$ can be defined on the weights. Then, a function $g_\psi(\cdot)$ is superior if

$$w'_i \succeq w_i \text{ implies } g_\psi(w_1, \dots, w'_i, \dots, w_n) \succeq g_\psi(w_1, \dots, w_i, \dots, w_n)$$

and

$$g_\psi(w_1, \dots, w_n) \succeq \max_{\succeq}(w_1, \dots, w_n)$$

In particular, negative weights are permitted, as long as $g_\psi(\cdot)$ is superior. We will find this generalization useful in our later proofs.

[1]To account for initial states, we can imagine that each initial state ϕ is produced by a unary deduction rule ψ with antecedent ϕ_0 having weight $-\infty$.

Deductions for HA* on General Hypergraphs

IN-B:	$O(\underline{v}^{m-1}) : \alpha$	$\xrightarrow{\alpha}$	$I(\underline{v}^m)$	$: 0$
IN:	$O(v_0^{m-1}) : \alpha \;\; I(v_1^m) : \beta_1 \;\ldots\; I(v_n^m) : \beta_n$	$\xrightarrow{w(e)+\alpha+\sum_{i=1}^n \beta_i}$	$I(v_0)$	$: w(e) + \sum_{i=1}^n \beta_i$
OUT-B:	$I(\bar{v}^m) : \beta$	$\xrightarrow{\beta}$	$O(\bar{v}^m)$	$: 0$
OUT:	$O(v_0^m) : \alpha \;\; I(v_1^m) : \beta_1 \;\ldots\; I(v_i^m) : \beta_i \;\ldots\; I(v_n^m) : \beta_n$	$\xrightarrow{w(e)+\alpha+\sum_{i=1}^n \beta_i}$	$O(v_i^m)$	$: \alpha + w(e) + \sum_{i' \neq i} \beta_{i'}$

Table A.1. Deduction rules for HA* on general hypergraphs. Note that because of the way the chart is initialized, the weight of a state $O(v^{-1})$ is given by the outside heuristic $\hat{\alpha}(v^0)$. The IN and OUT schemata both refer to an edge $e = v_1^m \ldots v_n^m \xrightarrow{w(e)} v^m$.

Reformulated Deductions for HA*

IN-B:	$O(\underline{v}^m) : (0, \alpha)$	$\rightarrow$	$I(\underline{v}^m)$	$: (0, \alpha)$
IN:	$O(v^{m-1}) : (\beta_h, \alpha_h) \;\; I(v_1^m) : (\beta_1, \alpha_1) \;\ldots\; I(v_n^m) : (\beta_n, \alpha_n)$	$\rightarrow$	$I(v^m)$	$: (w(e) + \sum_{i=1}^n \beta_i, \alpha_h)$
OUT-B:	$I(\bar{v}^m) : (\beta, 0)$	$\rightarrow$	$O(\bar{v}^m)$	$: (\beta, 0)$
OUT:	$O(v^m) : (\beta_h, \alpha_h) \;\; I(v_1^m) : (\beta_1, \alpha_1) \;\ldots\; I(v_n^m) : (\beta_n, \alpha_n)$	$\rightarrow$	$O(v_i^m)$	$: (\beta_i, \alpha_h + w(e) + \sum_{i' \neq i} \beta_{i'})$

Table A.2. Reformulation of the HA* deductions (Table A.1). These rules are equivalent to HA*, but in a form that can be cast as an instance of Knuth's algorithm. We omit the priority above the arrow because in this formulation, the priority of a rule is the same as its weight ($p_\psi(\cdot) \equiv g_\psi(\cdot)$). Note that because of the way the chart is initialized, the weight of a state $O(v^{-1})$ is given by $(-\infty, \hat{\alpha}(v^0))$. The IN and OUT schemata both refer to an edge $e = v_1^m \ldots v_n^m \xrightarrow{w(e)} v^m$.

A.2 Proof of Correctness for HA*

In the preceding section, we established that the correctness and monotonicity guarantees hold for weighted deduction rules in the case where $g_\psi(\cdot)$ is superior and $p_\psi(\cdot) = g_\psi(\cdot)$. In this section, we will show that HA* can be reformulated as an instance of Knuth's algorithm, and hence inherits these guarantees.

The deduction rule schemata for HA* on general hypergraphs are shown in Table A.1, and our reformulation is shown in Table A.2. In the reformulation, each state has a (generalized) weight $w = (\beta, \alpha)$ with $w \succeq w'$ iff $\beta + \alpha \geq \beta' + \alpha'$. For inside states $I(v^m)$ at the mth level of the hierarchy, β will be used to compute Viterbi inside costs at level m, while α will be used to compute Viterbi outside costs at level $m-1$. For outside states $O(v^m)$, β and α will be used to compute Viterbi inside and outside costs at the mth level.

Unlike the version of HA* presented in Felzenszwalb and McAllester (2007), our reformulation also assumes the presence of a consistent heuristic $\hat{\alpha}(v^0)$ defined for all 0th-level vertices v^0. The heuristic $\hat{\alpha}(\cdot)$ is necessary to properly deal with negative weights. In the special calse that all weights are non-negative, as Felzenszwalb and McAllester (2007) assume, then the trivial heuristic $\hat{\alpha}(\cdot) \equiv 0$ is consistent and no explicit heuristic is needed. Our reformulation is initialized by placing inside states $I(\underline{v}^0)$ with weight $w = (0, \hat{\alpha}(\underline{v}^0))$ on the chart for all leaf vertices $\underline{v}^0$ at the 0th level and outside states $O(v^{-1})$ with weight $w = (-\infty, \hat{\alpha}(v^0))$ for all v^0 in G^0.

We build up to the full proof with some lemmas. We start by proving in Lemma 1 that the inside rule schemata (IN and IN-B) produce the correct weights, assuming the correctness of the outside rule schemata (OUT and OUT-B). In Lemma 2, we show that the outside schemata are correct, if we assume the correctness of the inside schemata. In Theorem 2, we resolve the mutual recursion of these two lemmas with an appropriate base case and provide a summary of our claims.

Lemma 1. *During execution of the deduction rules in Table A.2, suppose that the weight of any outside state $O(v^m)$ removed from the agenda is $w = (\beta(v^m), \alpha(v^m))$. Then, we have that:*

(i) The weight of an inside state $I(v^m)$ on the agenda is $(w(\rho(v^m)), \alpha(v^{m-1}))$ for some path $\rho(v^m)$ in G^m to a vertex v^m.

(ii) The weight of an inside state $I(v^m)$ when it is removed from the agenda is $w = (\beta(v^m), \alpha(v^{m-1}))$.

Proof. We prove (i) by induction: we assume that (i) holds for antecedent states of a rule, and show that it holds for the conclusion. As a base case, consider an inside state $I(\underline{v}^m)$ for a leaf vertex $\underline{v}^m$. This state was placed on the agenda according to the IN-B rule, so its weight is $(0, \alpha(v^{m-1}))$. The only path to a leaf vertex is the empty path, which has weight 0, so the lemma holds.

Now consider an inside state $I(v^m)$ on the agenda where v^m is non-leaf vertex. This state was placed on the agenda by the IN schema for some edge e^m. We assume by induction that the lemma holds for all vertices in $v_i^m \in T(e^m)$, and show that the lemma holds for $h(e^m) = v^m$. The weight w_i^m for each inside state $I(v_i^m)$ is $(w(\rho(v_i^m)), \alpha(v_i^{m-1}))$ by the inductive assumption, and so the weight of $I(v^m)$ according to the IN schema is

$$\begin{aligned}\big(\beta_1 + \cdots + \beta_n + w(e^m), \alpha(v^{m-1})\big) &= \big(w(\rho(v_1^m)) + \cdots + w(\rho(v_n^m)) + w(e^m), \alpha(v^{m-1})\big) \\ &= \big(w(\rho(v^m)), \alpha(v^{m-1})\big)\end{aligned}$$

where $\rho(v^m) = \{e^m\} \cup \rho(v_1^m) \cup \cdots \cup \rho(v_n^m)$,which proves (i). Because $\hat{\alpha}(\cdot)$ is a consistent heuristic for the 0th level, and $\alpha(v^{m-1})$ is a consistent heuristic all subsequent levels, we know that

$$w(\rho(v^m)) + \alpha(v^{m-1}) = w(\rho(v_1^m)) + \cdots + w(\rho(v_n^m)) + w(e^m) + \alpha(v^{m-1}) \geq w(\rho(v_i^m)) + \alpha(v_i^{m-1})$$

for $i = 1 \ldots n$, and so $w \succeq w'$ where w is the weight of the conclusion and w' is the weight of any antecedent inside state. We also know

$$w(\rho(v^m)) + \alpha(v^{m-1}) \geq \beta(v^m) + \alpha(v^{m-1}) \geq \beta(v^{m-1}) + \alpha(v^{m-1})$$

so the weight of conclusion is also greater than or equal to the weight of the outside antecedent state. Hence, $g_\psi(\cdot)$ is superior for the IN deduction schema.[2]

To prove (ii), we note that by the correctness of Knuth's algorithm, we can conclude that the weight $w = (\beta, \alpha)$ for a state $I(v)$ that is minimized. Since $\alpha = \alpha(v^{m-1})$ is the same for all rules that proposed $I(v)$, β must be minimized. By (i), $\beta = w(\rho(v))$ for some path $\rho(v)$, and since the minimum over paths $\rho(v)$ is $\beta(v)$, we must have $\beta = \beta(v)$, which proves (ii).[3] $\square$

We now prove the correctness of the outside deduction rules, assuming that inside states have weight $w = (\beta(v^m), \alpha(v^{m-1}))$ for all v^m at any level.

Lemma 2. *During execution of the deduction rules in Table A.2, suppose that the weight of an inside state $I(v^m)$ for any v^m is $w = (\beta(v^m), \alpha(v^{m-1})$. Then, we have that:*

(i) The weight of an outside state $O(v^m)$ on the agenda is $w = (\beta(v^m), w(\ddot{\rho}(v^m))$ for some outside path $\ddot{\rho}(v^m)$ in G^m.

(ii) The weight of an outside state $O(v^m)$ when it is removed from the agenda is $w = (\beta(v^m), \alpha(v^m))$.

[2]Superiority also requires that $g_\psi(\cdot)$ be non-decreasing in each argument. Because $g_\psi(\cdot)$ is in all cases the sum of β of inside states and α of outside states, it is clearly non-decreasing in those values. Moreover, α for an inside state is the same for all rules that propose that state, and β is the same for all proposals of an outside state. Therefore, $w' \succeq w$ with $w' = (\beta', \alpha')$ and $w = (\beta, \alpha)$ implies $\beta' \geq \beta$ for inside states and $\alpha' \geq \alpha$ for outside states, which in turn implies that $g_\psi(\ldots, w', \ldots) \succeq g_\psi(\ldots, w, \ldots)$.

[3]We might worry that the deductive program only proposes a subset of possible paths $\rho(v)$ to a given vertex, and therefore the minimum over all such path withs is not $\beta(v)$. However, it is easy to verify that the IN deduction rules can propose all possible paths to v.

Proof. The proof of (i) is by induction as in Lemma 1, though the direction of the inductive assumption is reversed: we assume that the lemma holds for a vertex $v = h(e)$, and show that it holds for all vertices $v' \in T(e)$.

As a base case, consider an outside state $O(\bar{v}^m)$ proposed by the OUT-B schema. The weight of this state is the weight of $I(\bar{v}^m)$, which by assumption is $w = (\beta(\bar{v}^m), \alpha(\bar{v}^{m-1})) = (\beta(\bar{v}^m), 0)$. The only outside path from the root is the empty path with weight 0, so (i) holds. Now, consider an outside state $O(v_i^m)$ proposed by the OUT rule for edge e^m with $h(e^m) = v^m$ and $v_i^m \in T(e^m)$, and assume that (i) holds for $O(v^m)$. Then the weight of $O(v_i^m)$ is

$$\begin{aligned}&\big(\beta(v_i^m), \beta(v_1^m) + \cdots + \beta(v_{i-1}^m) + \beta(v_{i+1}^m) + \cdots + \beta(v_n^m) + w(e^m) + w(\ddot{\rho}(v^m))\big)\\ = &\,(\beta(v_i^m), w(\ddot{\rho}(v_i^m)))\end{aligned}$$

where $w(\ddot{\rho}(v_i^m))$ is the weight of an outside path for v_i^m in which v_i^m is the child of edge e^m and each sibling of v_i^m has its optimal inside path. Therefore, (i) holds for the conclusion. It should be clear that $(\beta(v_i^m), w(\ddot{\rho}(v_i^m))) \succeq (\beta(v^m), w(\ddot{\rho}(v^m)))$ because

$$\beta(v_i^m) + w(\ddot{\rho}(v_i^m)) = \beta(v_1^m) + \cdots + \beta(v_n^m) + w(e^m) + w(\ddot{\rho}(v^m)) \geq \beta(v^m) + w(\ddot{\rho}(v^m))$$

$\beta(v^m)$. Therefore, $g_\psi(\cdot)$ is superior for OUT, and by the correctness of Knuth's algorithm, the weight of a state $O(v^m)$ when removed from the agenda must be a minimum. Because β is the same for all rules which produce $O(v^m)$, if the weight is a minimum, then α must be a minimum, so $\alpha = \alpha(v^m)$. This proves (ii) for the OUT schema. □

We are now ready to state the correctness claims about our reformulation of HA*.

Theorem 2. *Given:*

(a) A hierarchy of hypergraphs $G^0 \dots G^M$ in which G^{m-1} is a relaxed projection of G^m for each $m = 1 \dots M$

(b) A consistent heuristic $\hat{\alpha}(v^0)$ defined for all $v^0 \in G^0$

(c) A chart initialized with states $I(\underline{v}^0)$ with weight $w = (0, \hat{\alpha}(\underline{v}^0))$ for all $\underline{v}^0$ and states $O(v^{-1})$ with weight $w = (-\infty, \hat{\alpha}(v^0))$ for all v^0 in G^0.

(d) Deduction rules from the schemata in Table A.2 for all edges $e^m \in G^m$ for $m = 0 \dots M$.

Then:

(i) The weight of an inside state $I(v^m)$ when it is removed from the agenda is $w = (\beta(v^m), \alpha(v^{m-1}))$.

(ii) The weight of an outside state $O(v^m)$ when it is removed from the agenda is $w = (\beta(v^m), \alpha(v^m))$

(iii) When a state with weight $w = (\beta, \alpha)$ is removed from the agenda, then only states with weight $w' = (\beta', \alpha')$ with $\beta' + \alpha' \leq \beta + \alpha$ have already been removed from the agenda.

Proof. We already proved (i) in Lemma 1 and (ii) in Lemma 2. Although these proofs are mutually recursive, the inside deduction schemata at level 0 act as a base case because those schemata do not have outside states as antecedents in the 0th level of the hierarchy, relying instead on the heuristic $\hat{\alpha}(\cdot)$. (iii) follows immediately from the monotonicity guarantee of Knuth's algorithm. $\square$

Together, (i) and (ii) establish the correctness guarantee of Section 3.1.3, while (iii) establishes the efficiency guarantee.

A.3 Proof of Correctness for BHA*

The proof of correctness for BHA* is analogous to that for HA*: we can transform it into an equivalent formulation and prove correctness by appealing to Knuth's algorithm. The only difference from the proof for HA* is that we need to verify that bridge outside scores $\tilde{\alpha}(v)$ ensure superiority of the inside deduction schemata, and mixed coarse and fine inside scores ensure superiority of the outside deduction schemata.

To generalize BHA* to negative weights, we will require a heuristic as we did for HA*. However, in this case, we will need an **inside heuristic** $\hat{\beta}(v^m)$ which lower bounds the Viterbi inside cost of a vertex v^m. An inside heuristic is consistent for a graph G if

$$\hat{\beta}(v) \leq w(e) + \sum_{v' \in T(e)} \hat{\beta}(v')$$

Deductions for BHA* on General Hypergraphs

IN-B:	$\tilde{O}(\underline{v}^m) : \alpha$	$\xrightarrow{\alpha}$	$I(\underline{v}^m)$	$: 0$
IN:	$\tilde{O}(v_0^{m-1}) : \alpha \;\; I(v_1^m) : \beta_1 \;\ldots\; I(v_n^m) : \beta_n$	$\xrightarrow{\alpha + w(e) + \sum_{i=1}^n \beta_i}$	$I(v_0)$	$: w(e) + \sum_{i=1}^n \beta_i$
OUT-B:	$I(\bar{v}^m) : \beta$	$\xrightarrow{\beta}$	$\tilde{O}(\bar{v}^m)$	$: 0$
OUT:	$\tilde{O}(v_0^m) : \alpha \;\; I(v_1^{m-\mathbf{1}_{i \le 1}}) : \beta_1 \;\ldots\; I(v_n^{m-\mathbf{1}_{i \le n}}) : \beta_n$	$\xrightarrow{\alpha + w(e) + \sum_{i'=1}^n \beta_{i'}}$	$\tilde{O}(v_i^m)$	$: \alpha + w(e) + \sum_{i' \ne i} \beta_{i'}$

Table A.3. Deduction rule schemata for BHA* on general hypergraphs. Note that because of the way the chart is initialized, the weight for a state $I(v^{-1})$ is $\hat{\beta}(v^0)$. The IN and OUT schemata both refer to an edge $e = v_1^m \ldots v_n^m \xrightarrow{w(e)} v^m$.

Reformulated Deductions for BHA* on General Hypergraphs

IN-B:	$\tilde{O}(\underline{v}^m) : (0, \alpha)$	$\rightarrow$	$I(\underline{v}^m)$	$:(0, \alpha)$
IN:	$\tilde{O}(v^m) : (\beta_h, \alpha_h) \;\; I(v_1^m) : (\beta_1, \alpha_1) \;\ldots\; I(v_n^m) : (\beta_n, \alpha_n)$	$\rightarrow$	$I(v^m)$	$:(w(e^m) + \sum_{i=1}^n \beta_i, \alpha_h)$
OUT-B:	$I(\bar{v}^m) : (\beta, 0)$	$\rightarrow$	$\tilde{O}(\bar{v}^m)$	$:(\beta, 0)$
OUT:	$\tilde{O}(v^m) : (\beta_h, \alpha_h) \;\; I(v_1^{m-\mathbf{1}_{i \le 1}}) : (\beta_1, \alpha_1) \;\ldots\; I(v_n^{m-\mathbf{1}_{i \le n}}) : (\beta_n, \alpha_n)$	$\rightarrow$	$\tilde{O}(v_i^m)$	$:(\beta_c, \alpha_h + w(e^m) + \sum_{i' \ne i} \beta_{i'})$

Table A.4. Reformulated deduction rule schemata for BHA* that allow it to be cast as an instance of Knuth's algorithm. We omit the priority above the arrow because in this formulation, the priority of a rule is the same as its weight ($p_\psi(\cdot) \equiv g_\psi(\cdot)$). Note that because of the way the chart is initialized, the weight for a state $I(v^{-1})$ is given by $(\hat{\beta}(v^0), -\infty)$. The IN and OUT schemata both refer to an edge $e = v_1^m \ldots v_n^m \xrightarrow{w(e)} v^m$.

for all vertices v and edges e with $h(e) = v$. A consistent inside heuristic is also admissible, meaning $\hat{\beta}(v) \le \beta(v)$ for all v. It is easy to see that inside scores from a relaxed projection G' of G form consistent inside heuristics for G.

We show the general form of BHA* in Table A.3 and the modified version for proof purposes in Table A.4. The chart is initialized with inside states $I(v^{-1})$ having weight $w = (\hat{\beta}(v^0), -\infty)$ for all $v^0 \in G^0$, and the outside state $O(\bar{v}^0)$ having weight $w = (\hat{\beta}(v^0), 0)$.

To prove the correctness of BHA*, we first prove Lemmas 3 and 4 which are analogous to Lemmas 1 and 2.

Lemma 3. *During execution of the deduction rules in Table A.4, suppose that the weight of any outside state $\tilde{O}(v^m)$ removed from the agenda is $w = (\beta(v^m), \tilde{\alpha}(v^m))$. Then, we have that:*

(i) The weight of an inside state $I(v^m)$ on the agenda is $(w(\rho(v^m)), \tilde{\alpha}(v^m))$ for some path $\rho(v^m)$ to a vertex v^m.

(ii) The weight of an inside state $I(v^m)$ when it is removed from the agenda is $w = (\beta(v^m), \tilde{\alpha}(v^m))$.

Proof. We can prove this lemma using the same argument we used for Lemma 1 if we can establish that bridge Viterbi outside costs $\tilde{\alpha}(v^m)$ are consistent outside heuristics for level m.

We know that bridge outside costs satisfy

$$\tilde{\alpha}(v_i^m) \leq w(e^m) + \tilde{\alpha}(v_i^m) + \sum_{i'=1}^{i'-1} w(\rho(v_{i'}^m)) + \sum_{i'=i+1}^{n} w(\rho(v_{i'}^{m-1}))$$

for all $e^m = v_1^m \; \dots v_n^m \rightarrow v^m$ and inside paths $\rho(v_{i'}^m)$ and $\rho(v_{i'}^{m-1})$.

Then for any inside path $\rho(v_i^m)$,

$$\begin{aligned} w(\rho(v_i^m)) + \tilde{\alpha}(v_i^m) &\leq w(\rho(v_i^m)) + w(e^m) + \tilde{\alpha}(v^m) + \sum_{i'=1}^{i'-1} w(\rho(v_{i'}^m)) + \sum_{i'=i+1}^{n} w(\rho(v_{i'}^{m-1})) \\ &= w(e^m) + \tilde{\alpha}(v^m) + \left(\sum_{i'=1}^{i'-1} w(\rho(v_{i'}^m)) + w(\rho(v_i^m)) + \sum_{i'=i+1}^{n} w(\rho(v_{i'}^{m-1})) \right) \\ &\leq w(e^m) + \tilde{\alpha}(v^m) + \sum_{i'=1}^{n} w(\rho(v_{i'}^m)) \end{aligned}$$

so $\tilde{\alpha}(\cdot)$ is a consistent outside heuristic for level m. □

We now proof that the outside deductions are correct assuming the inside deductions work as desired.

Lemma 4. *During execution of the deduction rules in Table A.4, suppose that the weight of an inside state $I(v^m)$ for any v^m with $w = (\beta(v^m), \tilde{\alpha}(v^m))$. Then, we have that:*

(i) The weight of a bridge outside state $\tilde{O}(v^m)$ on the agenda is $w = (\beta(v^{m-1}), \tilde{w}(\ddot{\rho}(v^m))$ for some outside path $\ddot{\rho}(v^m)$.

(ii) The weight of an outside state $O(v^m)$ when it is removed from the agenda is $w = (\beta(v^{m-1}), \tilde{\alpha}(v^m))$.

Proof. The proof of (i) is nearly identical to the proof of Lemma 2(i). The base case is unchanged, and the weight of a bridge outside state $\tilde{O}(v_i^m)$ proposed by the OUT rule for edge e^m with $h(e^m) = v^m$ and $v_i^m \in T(e^m)$, is

$$\begin{aligned}&(\beta(v_i^{m-1}), \beta(v_1^m) + \cdots + \beta(v_{i-1}^m) + \beta(v_{i+1}^{m-1}) + \cdots + \beta(v_n^{m-1}) + w(e^m) + \tilde{w}(\ddot{\rho}(v^m)))\\ =&(\beta(v_i^{m-1}), \tilde{w}(\ddot{\rho}(v_i^m)))\end{aligned}$$

where $\tilde{w}(\ddot{\rho}(v_i^m))$ is the bridge weight of an outside path for v_i^m in which v_i^m is a child of edge e^m, each left sibling of v_i^m has its optimal inside path, and each right sibling of v_i^m has a bound on its optimal coarse inside path (or a bound on its optimal inside path given by the inside heuristic $\hat{\beta}(\cdot)$). Therefore, (i) holds for the conclusion.

The proof of (ii) is also the same as in Lemma 2(ii) if we can show the superiority of $g_\psi(\cdot)$, i.e. that $(\beta(v_i^{m-1}), \tilde{w}(\ddot{\rho}(v_i^m))) \succeq (\beta(v^{m-1}), \tilde{w}(\ddot{\rho}(v^m)))$. This holds because

$$\begin{aligned}\beta(v_1^m) + \cdots + \beta(v_{i-1}^m) + &\beta(v_i^{m-1}) + \cdots + \beta(v_n^{m-1}) + w(e^m) + \tilde{w}(\ddot{\rho}(v^m))\\ &\geq \beta(v_1^{m-1}) + \cdots + \beta(v_n^{m-1}) + w(e^m) + \tilde{w}(\ddot{\rho}(v^m))\\ &\geq \beta(v^{m-1}) + \tilde{w}(\ddot{\rho}(v^m))\end{aligned}$$

where the last inequality holds for $m > 0$ because $\beta(v^{m-1})$ is a consistent inside heuristic for level $m-1$, and also holds for level $m = 0$ because $\beta(v^{-1}) = \hat{\beta}(v^0)$, which is a consistent inside heuristic for level 0. $\square$

A.4 Proof of Correctness for KA*

We prove the correctness of this algorithm by a reduction to HA*. We construct an instance of HA* as follows: let G^0 be the hypergraph generated by the input hypergraph $\mathcal{G}$, and let G^1, the target hypergraph of our HA* instance, be a hypergraph in which there is one vertex v^1 for all inside paths $\rho(v)$ to all vertices $v \in G^0$ and each edge e^1 has the form

$$\rho(v_1)\ldots\rho(v_n) \xrightarrow{w} \rho(v)$$

for an edge $e = v_1 \ldots v_n \xrightarrow{w} v$ from G^0, and where $\rho(v) = \rho(v_1) \cup \cdots \cup \rho(v_n) \cup \{e\}$. By construction, G^0 is a relaxed projection of G^1 with projection function $\pi(\rho(v)) = v$, so the guarantees of HA* carry over to KA*. Since each vertex in G^1 corresponds to an inside path in G^0, KA* considers all possible paths to the root. By the monotonicity guarantee, KA* will pop off complete paths to the root in ascending order of weight.

A.5 Proof of Correctness for TKA*

We prove the correctness of this algorithm by a reduction to BHA*, using the same hierarchy of grammars G^0 and G^1 from the previous section. We can think of the outside derivation states $Q(\cdot)$ of TKA* as computing bridge outside scores for G^1. In particular, like the deduction rules that compute bridge outside costs for BHA*, the deduction rules that build outside derivation states in TKA* mix "coarse" states to the right of the current expansion point (inside vertex states from G^0) and "fine" states to the left of the current expansion point (complete inside derivations).