Finite-State Chart Constraints for Reduced Complexity Context-Free Parsing Pipelines

Brian Roark*
Oregon Health & Science University

Kristy Hollingshead**
University of Maryland

Nathan Bodenstab*
Oregon Health & Science University

We present methods for reducing the worst-case and typical-case complexity of a context-free parsing pipeline via hard constraints derived from finite-state pre-processing. We perform $O(n)$ predictions to determine if each word in the input sentence may begin or end a multi-word constituent in chart cells spanning two or more words, or allow single-word constituents in chart cells spanning the word itself. These pre-processing constraints prune the search space for any chart-based parsing algorithm and significantly decrease decoding time. In many cases cell population is reduced to zero, which we term chart cell "closing." We present methods for closing a sufficient number of chart cells to ensure provably quadratic or even linear worst-case complexity of context-free inference. In addition, we apply high precision constraints to achieve large typical-case speedups and combine both high precision and worst-case bound constraints to achieve superior performance on both short and long strings. These bounds on processing are achieved without reducing the parsing accuracy, and in some cases accuracy improves. We demonstrate that our method generalizes across multiple grammars and is complementary to other pruning techniques by presenting empirical results for both exact and approximate inference using the exhaustive CKY algorithm, the Charniak parser, and the Berkeley parser. We also report results parsing Chinese, where we achieve the best reported results for an individual model on the commonly reported data set.

1. Introduction

Although there have been great advances in the statistical modeling of hierarchical syntactic structure over the past 15 years, exact inference with such models remains very costly and most rich syntactic modeling approaches resort to heavy pruning, pipelining,

* Center for Spoken Language Understanding, Oregon Health & Science University, Beaverton, Oregon, 97006 USA. E-mails: `roarkbr@gmail.com, bodenstab@gmail.com`.

** Some of the work in this paper was done while Kristy Hollingshead was at OHSU. She is currently at the University of Maryland Institute for Advanced Computer Studies, College Park, Maryland, 20740 USA. E-mail: `hollingk@gmail.com`.

Submission received: 9 August 2011; revised submission received: 30 November 2011; accepted for publication: 4 January 2012.

or both. Pipeline systems make use of simpler models with more efficient inference to reduce the search space of the full model. For example, the well-known Ratnaparkhi (1999) parser used a part-of-speech (POS)-tagger and a finite-state noun phrase (NP) chunker as initial stages of a multi-stage Maximum Entropy parser. The Charniak (2000) parser uses a simple probalistic context-free grammar (PCFG) to sparsely populate a chart for a richer model, and Charniak and Johnson (2005) added a discriminatively trained reranker to the end of that pipeline.

Finite-state pre-processing for context-free parsing is very common as a means of reducing the amount of search required in the later stage. As mentioned earlier, the Ratnaparkhi pipeline used a finite-state POS-tagger and a finite-state NP-chunker to reduce the search space at the parsing stage, and achieved linear observed-time performance. Other recent examples of the utility of finite-state constraints for parsing pipelines include Glaysher and Moldovan (2006), Djordjevic, Curran, and Clark (2007), and Hollingshead and Roark (2007). Similar hard constraints have been applied for dependency parsing, as will be outlined in Section 2. Note that by making use of pre-processing constraints, such approaches are no longer performing full exact inference—these are approximate inference methods, as are the methods presented in this article.

Using finite-state chunkers early in a syntactic parsing pipeline has shown both an efficiency (Glaysher and Moldovan 2006) and an accuracy (Hollingshead and Roark 2007) benefit for parsing systems. Glaysher and Moldovan (2006) demonstrated an efficiency gain by explicitly disallowing constituents that cross chunk boundaries. Hollingshead and Roark (2007) demonstrated that high-precision constraints on early stages of the Charniak and Johnson (2005) pipeline (in the form of base phrase constraints derived either from a chunker or from later stages of an earlier iteration of the same pipeline) achieved significant accuracy improvements, by moving the pipeline search away from unlikely areas of the search space. All of these approaches (as with Ratnaparkhi earlier) achieve improvements by ruling out parts of the search space, and the gain can either be realized in efficiency (same accuracy, less time) and/or accuracy (same time, greater accuracy).

Rather than extracting constraints from taggers or chunkers built for different purposes, in this study we have trained prediction models to more directly reduce the number of entries stored in cells of a dynamic programming chart during parsing—even to the point of "closing" chart cells to all entries. We demonstrate results using three finite-state taggers that assign each word position in the sequence with a binary class label. The first tagger decides if the word can *begin* a constituent of span greater than one word; the second tagger decides if the word can *end* a constituent of span greater than one word; and the third tagger decides if a chart cell spanning a single word should contain phrase-level non-terminals, or only POS tags. Following the prediction of each word, chart cells spanning multiple words can be completely closed as follows: Given a chart cell (b, e) spanning words $w_b \ldots w_e$ where $b < e$, we can "close" cell (b, e) if the first tagger decides that w_b cannot be the first word of a multi-word constituent (MWC) or if the second tagger decides that w_e cannot be the last word in a MWC. Completely closing sufficient chart cells allows us to impose worst-case complexity bounds on the overall pipeline, a bound that none of the other previously mentioned methods for finite-state preprocessing can guarantee.

To complement closing multi-word constituent chart cells, our third tagger restricts the population of span-1 chart cells. We note that all span-1 chart cells must contain at least one POS tag and can therefore never be closed completely. Instead, our tagger restricts unary productions with POS tags on their right-hand side that span a single word. We term these single word constituents (SWCs). Disallowing SWCs alters span-1 cell population from potentially containing all non-terminals to just POS non-terminals. In practice, this decreases the number of entries in span-1 chart cells by 70% during exhaustive parsing, significantly reducing the number of allowable constituents in larger spans (Bodenstab, Hollingshead, and Roark 2011). Span-1 chart cells are also the most frequently queried cells in the Cocke-Younger-Kasami (CKY) algorithm. The search over possible midpoints will always include two cells spanning a single word—one as the first left child and one as the last right child. It is therefore beneficial to minimize the number of entries in these span-1 cells.

The pre-processing framework we have outlined is straightforward to incorporate into most existing context-free constituent parsers, a task we have already done for several state-of-the art parsers. In the following sections we formally define our approach to finite-state chart constraints and analyze the accuracy of each of the three taggers and their impact on parsing efficiency and accuracy when used to prune the search space of a constituent parser. We apply our methods to exhaustive CYK parsing with simple grammars, as well as to high-accuracy parsing approaches such as the Charniak and Johnson (2005) parsing pipeline and the Berkeley parser (Petrov and Klein 2007a, 2007b). Various methods for applying finite-state chart constraints are investigated, including methods that guarantee quadratic or linear complexity of the context-free parser.

2. Related Work

Hard constraints are ubiquitous within parsing pipelines. One of the most basic and standard techniques is the use of a POS-tag dictionary, whereby words are only allowed to be assigned one of a subset of the POS-tag vocabulary, often based on what has been observed in the training data. This will overly constrain polysemous word types that happen not to have been observed with one of their possible tags; yet the large efficiency gain of so restricting the tags is typically seen as outweighing the loss in coverage. POS-tag preprocessing has been used for both context-free constituent parsing (Ratnaparkhi 1999) and dependency parsing (McDonald et al. 2005). Richer tag sets can also be used to further constrain the parser, such as supertags (Bangalore and Joshi 1999), which contain information about how the word will syntactically integrate with other words in the sequence. Supertagging has been widely used to make parsing algorithms efficient, particularly those making use of context-sensitive grammars (Clark and Curran 2004).

By applying finite-state chart constraints to constituent parsing, the approaches pursued in this article constrain the possible shapes of unlabeled trees, eliminating from consideration trees with constituents over specific spans. There is thus some similarity with other tagging approaches (e.g., supertagging) that dictate how words combine with the rest of the sentence via specific syntactic structures. Supertagging is generally used to enumerate which sorts of structures are licensed, whereas the constraints in this article indicate unlabeled tree structures that are proscribed. Along the same lines, there is a very general similarity with coarse-to-fine search methods, such as those used in the Berkeley (Petrov and Klein 2007a) and Charniak (2000) parsers, and more general structured prediction cascades (Weiss, Sapp, and Taskar 2010; Weiss and Taskar 2010). Our approach also uses simpler models that reduce the search space for larger downstream models.

Dependency parsing involves constructing a graph of head/dependent relations, and many methods for constraining the space of possible dependency graphs have been investigated, such as requiring that each word have a single head or that the graph be acyclic. Nivre (2006) investigated the impact of such constraints on coverage and the number of candidate edges in the search space. Most interestingly, that paper found that constraining the degree of non-projectivity that is allowed can greatly reduce the number of arcs that must be considered during search, and, as long as some degree of non-projectivity is allowed, coverage is minimally impacted. Of course, the total absence of projectivity constraints allows for the use of spanning tree algorithms that can be quadratic complexity for certain classes of statistical models (McDonald et al. 2005), so the ultimate utility of such constraints varies depending on the model being used.

Other hard constraints have been applied to dependency parsing, including constraints on the maximum length of dependencies (Eisner and Smith 2005; Dreyer, Smith, and Smith 2006), which is known as **vine** parsing. Such vine parsers can be further constrained using taggers to determine the directionality and distance of each word's head in a way similar to our use of taggers (Sogaard and Kuhn 2009). More general arc filtering approaches, using a variety of features (including some inspired by previously published results of methods presented in this article) have been proposed to reduce the number of arcs considered for the dependency graph (Bergsma and Cherry 2010; Cherry and Bergsma 2011), resulting in large parsing speedups.

In a context-free constituent parsing pipeline, constraints on the final parse structure can be made in stages preceding the CYK algorithm. For example, base phrase chunking (Hollingshead and Roark 2007) involves identifying a span as a base phrase of some category, often NP. A base phrase constituent has no children other than pre-terminal POS-tags, which all have a single terminal child (i.e., there is no internal structure in the base phrase involving non-POS non-terminals). This has a number of implications for the context-free parser. First, there is no need to build internal structure within the identified base phrase constituent. Second, constituents that cross brackets with the base phrase cannot be part of the final tree structure. This second constraint on possible trees can be thought of as a constraint on chart cells, as pointed out in Glaysher and Moldovan (2006): No multi-word constituent can begin at a word falling within a base-phrase chunk, other than the first word of that chunk. Similarly, no multi-word constituent can end at a word falling within a base-phrase chunk, other than the last word of that chunk. These constraints rule out many possible structures that the full context-free parser would have otherwise considered.

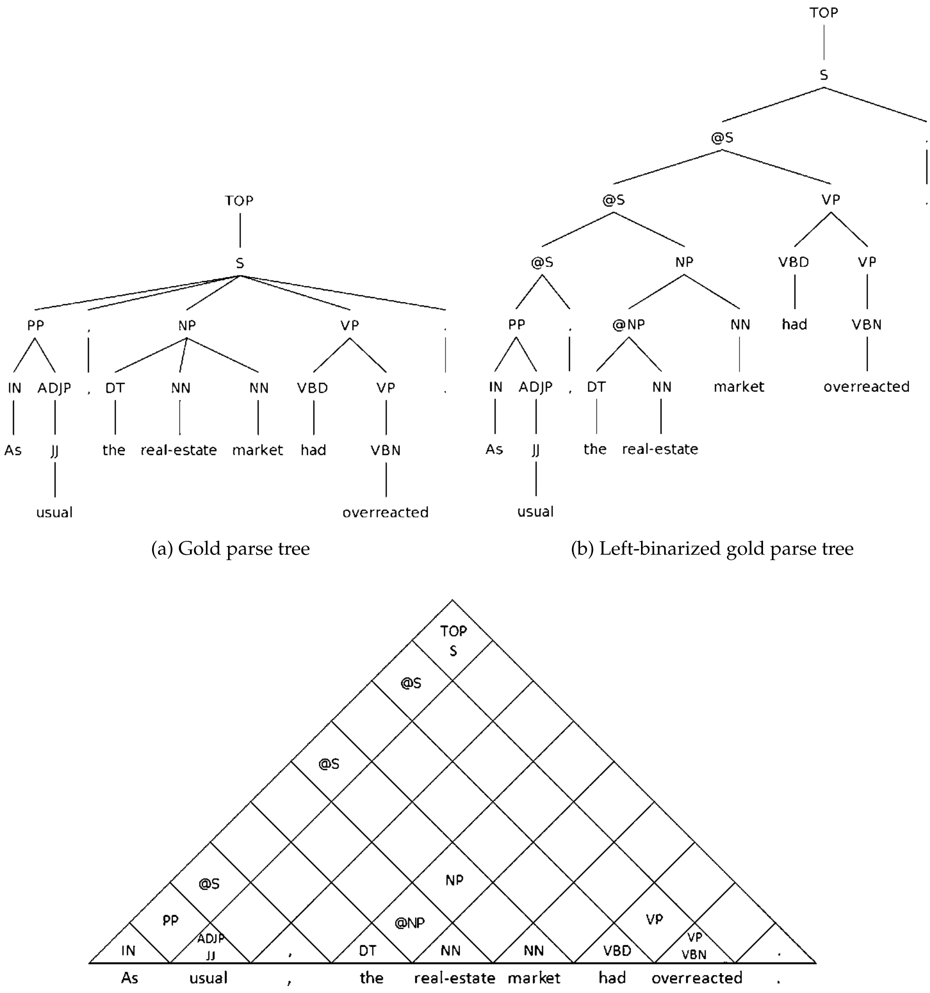
These begin and end constraints can be extracted from the output of the chunker, but the chunker is most often trained to optimize chunking accuracy, not parsing accuracy (or parsing precision). Further, these constraints can apply even for words that fall outside of typical chunks. For example, in English, verbs and prepositions tend to occur before their arguments, hence are often unlikely to end constituents, yet verbs and prepositions are rarely inside a typically defined base phrase. Instead of imposing parsing constraints from NLP pre-processing steps such as chunking, we propose that building specific prediction models to constrain the search space within the CYK chart will more directly optimize efficiency within a parsing pipeline.

In this article, we focus on linear complexity finite-state methods for deriving constraints on the chart. Recent work has also examined methods for constraining each of the $O(N^2)$ chart cell independently (Bodenstab et al. 2011), permitting a finer-grained pruning (e.g., not just "open" or "closed" but an actual beam width prediction) and the use of features beyond the scope of our tagger. We discuss this and other extensions of the current methods in our concluding remarks.

3. Preliminaries

3.1 Dynamic Programming Chart

Dynamic programming for context-free inference generally makes use of a chart structure, as shown in Figure 1c for the left-binarized gold parse tree in Figure 1b. Each cell in the chart represents a collection of possible constituents covering a substring, which is identified by the indices of the first and last words of the substring. Let $w_1 \ldots w_n$ be a string of n words to be parsed. The cell identified with (b, e) will contain possible constituents spanning the substring $w_b \ldots w_e$, where the *span* of a constituent or a chart cell is defined as the number of words it covers, or $e - b + 1$. As an example, in this



(a) Gold parse tree

(b) Left-binarized gold parse tree

(c) Dynamic programming chart with binarized constituents

Figure 1
Gold parse tree (a), left-binarized representation of the same tree (b), and corresponding dynamic programming chart (c) for the sentence *As usual, the real-estate market had overreacted.* (sentence 1094 in WSJ Section 24). Each cell in the chart spans a unique substring of the input sentence. Non-terminals preceded with the symbol "@" are created through binarization (see Section 3.3).

article we will occasionally refer to span-1 chart cells, meaning all chart cells covering a single word.

Context-free inference using dynamic programming over a chart structure builds longer-span constituents by combining smaller span constituents, guided by rules in a context-free grammar. A context-free grammar $G = (V, T, S^{\dagger}, P)$ consists of: a set of non-terminal symbols V, including a special start symbol $S^{\dagger}$; a set of terminal symbols T; and a set of rule productions P of the form $A \rightarrow \beta$ for $A \in V$ and $\beta \in (V \cup T)^{*}$, i.e., a single non-terminal on the left-hand side of the rule production, and a sequence of 0 or more terminals or non-terminals on the right-hand side of the rule. If we have a rule production $A \rightarrow B\ C \in P$, a completed B entry in chart cell (b, m) and a completed C entry in chart cell $(m{+}1, e)$, we can place a completed A entry in chart cell (b, e). Such a chart cell entry is sometimes called an "edge" and can be represented by the tuple $(A \rightarrow B\ C, b, m, e)$.

Context-free inference has cubic complexity in the length of the string N, due to the $O(N^2)$ number of chart cells and $O(N)$ possible child configurations at each cell. As an example, a cell spanning $(b,\ e)$ must consider all possible configurations of two child constituents (child cells) that span a proper prefix $(b,\ m)$ and a proper suffix $(m{+}1,\ e)$ where $b \leq m < e$, leading to $O(N)$ possible midpoints.

3.2 The CYK Algorithm

Algorithm 1 contains pseudocode for the CYK algorithm (Kasami 1965; Younger 1967; Cocke and Schwartz 1970), where the context-free grammar G is assumed to be binarized. The function ρ maps each grammar production in P to a probability. Lines 1–3

Algorithm 1 Cyk
Pseudocode of the CYK algorithm using a binarized PCFG. Unary processing is simplified to allow only chains of length one (excluding lexical unary productions). Backpointer storage is omitted.

```
Input:
    $w_1 \dots w_n$: Input sentence
    $G$: Left-binarized PCFG
Output:
    $\alpha$: Viterbi-max score for all non-terminals over every span

Cyk($w_1 \dots w_n$, $G = (V, T, S^{\dagger}, P, \rho)$)
 1: for $s = 1$ to $n$ do                                        ▷ Span width: bottom-up traversal
 2:   for $b = 1$ to $n - s + 1$ do                              ▷ Begin word position
 3:     $e \leftarrow b{+}s{-}1$
 4:     for $A_i \in V$ do
 5:       if $s = 1$ then                                        ▷ Add lexical productions
 6:         $\alpha_i(b, e) \leftarrow \rho(A_i \rightarrow w_b)$
 7:       else
 8:         $\alpha_i(b, e) \leftarrow \max_{b \leq m < e} \left( \max_{j,k} \ \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b, m)\ \alpha_k(m + 1, e) \right)$
 9:     for $A_i \in V$ do                                       ▷ Add unary productions
10:       $\upsilon_i(b, e) \leftarrow \max \left( \alpha_i(b, e)\ ,\ \max_{j}\ \rho(A_i \rightarrow A_j)\ \alpha_j(b, e) \right)$
11:     $\alpha(b, e) \leftarrow \upsilon(b, e)$
12: return $\alpha$
```

iterate over all $O(N^2)$ chart cells in a bottom–up traversal. Line 6 initializes span-1 cells with all possible part-of-speech tags, and line 8 introduces the cubic complexity of the algorithm: maximization over all midpoints m, which is $O(N)$. The variable α stores the Viterbi-max score for each non-terminal $A_i \in V$ at each cell (b, e), representing the span's most probable derivation rooted in A_i. Backpointers indicating which argument(s) maximize each $\alpha_i(b, e)$ can be optionally recorded to efficiently extract the maximum likelihood solution at the end of inference, but we omit these for clarity; they are easily recoverable by storing the *argmax* for each *max* in lines 8 and 10.

We have also included pseudocode in Algorithm 1 to process unary productions in lines 9 and 10. Unary processing is a necessary step to recover the gold-standard trees of the Penn treebanks (Marcus, Marcinkiewicz, and Santorini 1993; Xue et al. 2005), but is often ignored in the presentation of the CYK algorithm. Because there is little discussion about unary processing in the literature, implementation details often differ from parser to parser. In Algorithm 1 we present a simplified version of unary processing that only allows unary chains of length 1 per span (excluding lexical productions). This approach is efficient and can be iterated as needed to represent the length of observed unary chain in the treebank. Note that line 10 uses the temporary variable υ to store the accumulated Viterbi-max scores α. This temporary variable is necessary due to the iterative nature in which we update α. If we were to write the result of line 10 directly to $\alpha_i(b, e)$, then the subsequent maximization of $\alpha_{i+1}(b, e)$ would use an unstable version of α, some of which would already be updated with unary productions, and some which would not.

3.3 Incomplete Edges: Chomsky Normal Form and Dotted-Rules

A key feature to the efficiency of the CYK algorithm is that all productions in the grammar G are assumed to have no more than two right-hand-side children. Rather than trying to combine an arbitrary number of smaller substrings (child cells), the CYK algorithm exploits shared structure between rules and only needs to consider pairwise combination. To conform to this requirement, *incomplete* edges are needed to represent that further combination is required to achieve a complete edge. This can either be performed in advance, for example, by transforming a grammar into Chomsky Normal Form resulting in "incomplete" non-terminals created by the transform, or incomplete edges can be represented through so-called dotted rules, as with the Earley (1970) algorithm, in which transformation is essentially performed on the fly. For example, if we have a rule production $A \to B\ C\ D \in P$, a completed B entry in chart cell (b, m_1) and a completed C entry in chart cell $(m_1{+}1, m_2)$, then we can place an *incomplete* edge $A \to B\ C \cdot D$ in chart cell (b, m_2). The dot signifies the division between what has already been combined (left of the dot), and what remains to be combined. Then, if we have an incomplete edge $A \to B\ C \cdot D$ in chart cell (b, m_2) and a complete D in cell $(m_2{+}1, e)$, we can place a completed A entry in chart cell (b, e).

Transforming a grammar into Chomsky Normal Form (CNF) is an off-line operation that converts rules with more than two children on the right-hand side into multiple binary rules. To do this, composite non-terminals are created during the transformation, which represent incomplete constituents (i.e., those edges that require further combination to be made complete).[1] For example, if we have a rule production $A \to B\ C\ D$ in

[1] In this section we assume that edges are extended from left-to-right, which requires a left-binarization of the grammar, but everything carries over straightforwardly to the right-binarized case.

the context-free grammar G, then a new composite non-terminal would be created (e.g., @A:BC) and two binary rules would replace the previous ternary rule: $A \rightarrow$ @A:BC D and @A:$BC \rightarrow B\ C$. The @A:BC non-terminal represents part of a rule expansion that needs to be combined with something else to produce a complete non-terminal from the original set of non-terminals.[2]

In addition to binarization, one frequent modification to the grammar is to create a Markov model of the dependencies on the right-hand side of the rule. One way to do this is to reduce the number of children categories annotated on our new composite non-terminals introduced by binarization. For example, if instead of @A:BC we assign the label @A:C to our new non-terminal—with the semantics "an incomplete A constituent with rightmost child C"—then the two rules that result from binarization are: $A \rightarrow$ @A:C D and @A:$C \rightarrow B\ C$. Probabilistically, the result of this is that the children non-terminals B and D are conditionally independent of each other given A and C. This approach will provide probability mass to combinations of children of the original category A that may not have been observed together, hence it should be seen as a form of smoothing. One can go further and remove all children categories from the new non-terminals (i.e., replacing @A:BC with just @A). This is the binarization pursued in the Berkeley parser, and is shown in Figure 1b.

In this article, we explicitly discuss unary productions of type $A \rightarrow B$ where B is a non-terminal, and include these productions in our grammar. These productions violate the definition of a CNF grammar, and therefore we will use the term "binarized grammar" for the remainder of the article to indicate a grammar in CNF with the addition of unary productions. We will assume that all of our grammars have been previously binarized and we define V' to be the set of non-terminals that are created through binarization, and denote edges where $A \in V'$ as **incomplete edges**. Note that categories $A \in V'$ are only used in binary productions, not unary productions, a consideration that will be used in our constrained parsing algorithm.

4. Finite-State Chart Constraints

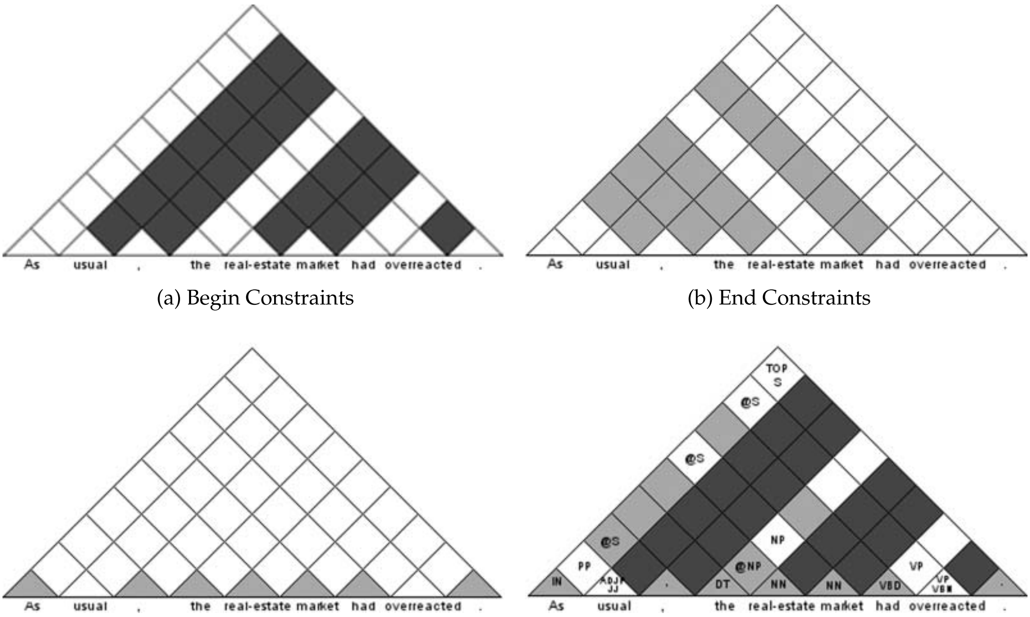
In this section, we will explicitly define our chart constraints, and present methods for using the constraints to constrain parsing. We begin with constraints on beginning or ending multi-word constituents, then move to constraining span-1 chart cells.

4.1 Constituent Begin and End Constraints

Our task is to learn which words (in the appropriate context) can begin (B) or end (E) multi-word constituents. We will treat this as a pre-processing step to parsing and use these constraints to either completely or partially close chart cells during execution of the CYK algorithm.

First, let us introduce notation. Given a set of labeled pairs (S, T) where S is a string of n words $w_1 \ldots w_n$ and T is the target constituent parse tree for S, we say that word $w_b \in B$ if there is a constituent spanning $w_b \ldots w_e$ for some $e > b$ and $w_b \in \overline{B}$ otherwise. Similarly, we say that word $w_e \in E$ if there is a constituent spanning $w_b \ldots w_e$ for some $b < e$ and $w_e \in \overline{E}$ otherwise. Recovery of these labels will be treated as two separate binary tagging tasks ($B/\overline{B}$ and $E/\overline{E}$).

[2] In this example, the symbol "@" indicates that the new non-terminal is incomplete, and the symbol ":" separates the original parent non-terminal from the children.


As usual , the real-estate market had overreacted .
As usual , the real-estate market had overreacted .
As usual , the real-estate market had overreacted .
TOP
S
@S
@S
@S
PP
NP
@NP
VP
IN
ADJP
JJ
.
DT
NN
NN
VBD
VP
VBN
.
As usual , the real-estate market had overreacted .

(a) Begin Constraints

(b) End Constraints

(c) Unary Constraints

(d) Joint Constraints

Figure 2
Begin (a), End (b), Unary (c), and the combination of the three (d) type of constraints for the dynamic programming chart used to parse *As usual, the real-estate market had overreacted* with a left-binarized grammar (see Figure 1). Black denotes "closed" cells, white cells are "open," and gray cells are only open to a restricted population (gray cells closed by $\overline{E}$ constraints only allow incomplete edges; gray cells closed by $\overline{U}$ constraints only allow POS tags).

Although it may be obvious that we can rule out multi-word constituents with particular begin and end positions, there may be *incomplete* structures within the parser that should not be ruled out by these same constraints. Hence the notion of "closing" a chart cell is slightly more complicated than it may initially seem (which accounts for our use of quotes around the term). Consider the chart representation in Figure 2 with the following constraints, where $\overline{B}$ is the set of words disallowed from beginning a multi-word constituent and $\overline{E}$ is the set of words disallowed from ending a multi-word constituent[3]

$\overline{B}$: {*"usual"*, *","*, *"real-estate"*, *"market"*, *"overreacted"*}
$\overline{E}$: {*","*, *"the"*, *"real-estate"*, *"had"*}

Given our constraints, suppose that w_b is in class $\overline{B}$ and w_e is in class $\overline{E}$, for $b < e$. We can "close" all cells (b, m_1) such that $m_1 > b$ and all cells (m_2, e) such that $m_2 < e$, based on the fact that multi-word constituents cannot begin with word w_b and cannot end with w_e. In Figure 2 this is depicted by the black and gray diagonals through the chart, "closing" those chart cells.

[3] In this example we list the actual words from the sentence for clarity with Figure 2, but in practice we classify word positions as a specific word may occur multiple times in the same sentence with potentially different B or E labels.

If a chart cell (b, e) has been "closed" due to begin or end constraints then it is clear that complete edges should not be permitted in the cell since these represent precisely the multi-word constituents that are being ruled out. But what about incomplete edges that are introduced through grammar binarization or dotted rule parsing? To the extent that an incomplete edge can be extended to a valid complete edge, it must be allowed. There are two cases where this is possible. If $w_b \in \overline{B}$, then under the assumption that incomplete edges are extended from left-to-right (see footnote 1), the incomplete edge should be discarded, because any completed edges that could result from extending that incomplete edge would have the same begin position. Stated another way, if $w_b \in \overline{B}$ for chart cell (b, e) then all chart cells (b, i) for $i > b$ must also be closed. For example, in Figure 2, the cell associated with the two-word substring *real-estate market* can be closed to both complete and incomplete edges, since *real-estate* $\in \overline{B}$, and any complete edge built from entries in that cell would also have to start with the same word and hence would be discarded. Thus, the whole diagonal is closed. If, however, $w_b \in B$ and $w_e \in \overline{E}$, such as the cell associated with the two-word substring *the real-estate* in Figure 2, a complete edge—achieved by extending the incomplete edge—may end at w_i for $i > e$, and cell (b, i) may be open (*the real-estate market*), hence the incomplete edge should be allowed in cell (b, e).

In Section 4.4 we discuss limitations on how such incomplete edges arise in closed cells, which has consequences for the worst-case complexity under certain conditions.

4.2 Unary Constraints

In addition to begin and end constraints, we also introduce unary constraints in span-1 cells. Although we cannot close span-1 cells entirely because each of these cells must contain at least one POS tag, we can reduce the population of these cells by restricting the type of constituents they contain. We define a **single-word constituent** (SWC) as any unary production $A \rightarrow B$ in the grammar such that B is a non-terminal (not a lexicon entry) and the production spans a single word. The productions ADJP → JJ and VP → VBN in Figure 1a are examples of SWCs. Note that TOP → S and JJ →*usual* in Figure 1a are also unary productions, but by definition they are not SWC unary productions. We train a distinct tagger, as is done for B and E constraints, to label each word position as either in U or $\overline{U}$, indicating that the word position may or may not be extended by a SWC, respectively.

Because the search over possible grammar extension from two child cells in the CYK algorithm is analogous to a database JOIN operation, the efficiency of this cross-product hinges on the population of the two child cells that are intersected. We focus on constraining the population of span-1 chart cells for three reasons. First, the begin/end constituent constraints only affect chart cells spanning more than one word and leave span-1 chart cells completely unpruned. By pruning entries in these span-1 cells, we complement multi-word constituent pruning so that all chart cells are now candidates for finite-state tagging constraints. Second, span-1 chart cells are the most frequently queried cells in the CYK algorithm. The search over possible midpoints will always include two cells spanning a single word—one as the first left child and one as the last right child. It is therefore important that the number of entries in these cells be minimized to make bottom–up CYK processing efficient. Finally, as we will show in Section 5, only 11.2% of words in the WSJ treebank are labeled with SWC productions. With oracle unary constraints, the possibility of constraining nearly 90% of span-1 chart cells has promising efficiency benefits to downstream processing.

To reiterate, unary constraints in span-1 chart cells never close the cell completely because each span-1 cell must contain at least one POS non-terminal. Instead, if $w_i \in \overline{U}$ then we simply do not add phrase-level non-terminals to chart cell (i, i). Similar to chart cells spanning multiple words that cannot be closed completely, these span-1 chart cells partially restrict the population of the cell, which we will empirically show to reduce processing time over unconstrained CYK parsing. These partially closed chart cells are represented as gray in Figure 2.

4.3 The Constrained CYK Algorithm

Our discussion of cell closing in Section 4.1 highlighted different conditions for closing cells for complete and incomplete edges. We will refer to specific conditions in the pseudocode, which we enumerate here. The constraints determine three possible conditions for cell (b, e) spanning multiple words:

1. $w_b \in \overline{B}$: cell is closed to *all* constituents, both complete and incomplete
2. $w_b \in B$ and $w_e \in \overline{E}$: cell is closed to complete constituents
3. $w_b \in B$ and $w_e \in E$: cell is open to all constituents

and two possible conditions for cell (i, i) spanning a single word:

4. $w_i \in \overline{U}$: cell is closed to unary phrase-level constituents
5. $w_i \in U$: cell is open to all constituents

We will refer to a chart cell (b, e) that matches the above condition as "case 1 cells," "case 2 cells," and so on, throughout the remainder of this article. Figure 2 represents these five cases pictorially, where case 1 cells are black, case 2 and 4 cells are gray, and case 3 and 5 cells are white.

Algorithm 2 contains pseudocode of our modified CYK algorithm that takes into account $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints. Line 4 of Algorithm 2 is the first modification from the standard CYK processing of Algorithm 1: We now consider only words in set B to begin multi-word constituents. Chart cells excluded from this criteria fall into case 1 and require no work. Line 5 determines if chart cell (b, e) is of case 2 (partially open) or case 3 (completely open). If $w_e \in \overline{E}$, then we skip to lines 16–17 and only incomplete edges are permitted. Note that there is only one possible midpoint for case 2 chart cells, which results in constant-time work (see proof in Section 4.4) and unary productions are not considered because all entries in the cell are incomplete constituents, which only participate in binary productions. Otherwise, if $w_e \in E$ on Line 5, then the cell is open to all constituents and processing occurs as in the standard CYK algorithm (lines 6–10 of Algorithm 2 are identical to lines 4–8 of Algorithm 1). Finally, unary productions are added in lines 12–14, but restricted to multi-word spanning chart cells, or span-1 cells where $w_b \in U$.

4.4 Proofs to Bound Worst-Case Complexity

All of the proofs in this section rely on constraints imposed by $\overline{B}$ and $\overline{E}$. Pruning provided by the $\overline{U}$ constraints reduces decoding time in practice, but does not provide additional reductions in complexity. Our proofs of complexity bounds will rest on the number of cells that fall in cases 1–3 outlined in the Section 4.3, and the amount

Algorithm 2 ConstrainedCyk
Pseudocode of a modified CYK algorithm, with constituent begin/end and unary constraints. Unary processing is simplified to allow only chains of length one (excluding lexical unary productions). Backpointer storage is omitted.

```
Input:
    $w_1 \ldots w_n$: Input sentence
    $G$: Left-binarized PCFG
    $V'$: Set of binarized non-terminals from $V$
    $B, E, U$: Begin, End, and Unary chart constraints
Output:
    $\alpha$: Viterbi-max scores for all non-terminals over every span

ConstrainedCyk($w_1 \ldots w_n$, $G = (V, T, S^{\dagger}, P, \rho), V', B, E, U$)
for $s = 1$ to $n$ do                                         ▷ Span width: bottom-up traversal
    for $b = 1$ to $n-s+1$ do                                 ▷ Begin word position
        $e \leftarrow b+s-1$
        if $w_b \in B$ or $s = 1$ then                        ▷ Case 1 cells excluded
            if $w_e \in E$ or $s = 1$ then
                for $A_i \in V$ do
                    if $s = 1$ then                           ▷ Add lexical productions
                        $\alpha_i(b,b) \leftarrow \rho(A_i \rightarrow w_b)$
                    else                                      ▷ Case 3: cell open
                        $\alpha_i(b,e) \leftarrow \max_{b \leq m < e} \left( \max_{j,k} \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b,m)\ \alpha_k(m+1,e) \right)$
                if $s > 1$ or $w_b \in U$ then                ▷ Case 5 for span-1 cells
                    for $A_i \in V$ do                        ▷ Add unary productions
                        $\upsilon_i(b,e) \leftarrow \max \left( \alpha_i(b,e)\ ,\ \max_j \rho(A_i \rightarrow A_j)\ \alpha_j(b,e) \right)$
                    $\alpha(b,e) \leftarrow \upsilon(b,e)$
            else                                              ▷ Case 2: closed to complete constituents
                for $A_i \in V'$ do                           ▷ Only consider binarized non-terminals
                    $\alpha_i(b,e) \leftarrow \max_{j,k} \rho(A_i \rightarrow A_j\ A_k)\ \alpha_j(b,e-1)\ \alpha_k(e,e)$
return $\alpha$
```

of work in each case. The amount of work for each case is related to how the Cyk and ConstrainedCyk algorithms performs their search. Each cell (b,e) in the chart spans the substring $w_b \ldots w_e$, and building non-terminal categories in that cell involves combining non-terminal categories (via rules in the context-free grammar) found in cells spanning adjacent substrings $w_b \ldots w_m$ and $w_{m+1} \ldots w_e$. The substring $w_b \ldots w_e$ can be as large as the entire sentence, requiring a search over $O(N)$ possible *midpoint* words w_m. This accounts for the linear amount of work in these case 3, open, cells.

More formally, we can define an upper bound on the work required in a chart cell as $\mathcal{W} = |P| \cdot |M|$ where $|P|$ is the number grammar productions bounded above by the constant size of the grammar, and $|M|$ is the number of midpoints considered. We denote the upper bound on the amount of work done in case 1 cells by $\mathcal{W}_1$ and the total number of case 1 cells considered by $\mathcal{C}_1$ (similarly for case 2 and 3). With this decomposition, the complexity of the ConstrainedCyk algorithm can be written as $O(|\mathcal{C}_1|\mathcal{W}_1 + |\mathcal{C}_2|\mathcal{W}_2 + |\mathcal{C}_3|\mathcal{W}_3)$. Because there is no work for case 1 cells ($\mathcal{W}_1 = 0$), these can be ignored. In fact, we can reduce the complexity even further.

Theorem 1
$\mathcal{W}_2$ is constant and ConstrainedCyk has complexity $O(|\mathcal{C}_2| + |\mathcal{C}_3|\mathcal{W}_3)$.

Proof
Without loss of generality, we assume a left-binarized grammar. The upper bound on required work for cell (b, e) is defined as $\mathcal{W} = |P| \cdot |M|$ where $|P|$ is the number of grammar productions bounded above by the constant size of the grammar, and $|M|$ is the number of midpoints considered. For some (b, e) where $1 \leq b < e \leq n$, assume $w_b \in B$ and $w_e \in \overline{E}$, that is, $(b, e) \in \mathcal{C}_2$. The possible child cells of (b, e) are (b, m) and $(m + 1, e)$ where $b \leq m < e$. Because we assume a left-binarized grammar, the production's right child must be a complete non-terminal and $(m + 1, e) \in \mathcal{C}_3$ must hold. But $w_e \in \overline{E}$ so $(m + 1, e)$ cannot be in $\mathcal{C}_3$ unless $m = e - 1$ (span 1 cell). Therefore, the only valid midpoint is $m = e - 1$ and $\mathcal{W}_2 = |P|{\cdot}1$ is constant, hence $O(|\mathcal{C}_2|\mathcal{W}_2 + |\mathcal{C}_3|\mathcal{W}_3) = O(|\mathcal{C}_2| + |\mathcal{C}_3|\mathcal{W}_3)$. ■

In summary, we find that case 2 chart cells (gray in Figure 2) are surprisingly cheap to process because the right child cell must also be in case 2, resulting in only one possible midpoint to consider. Next, we show that bounding the number of open cells and applying Theorem 1 leads to quadratically bounded parsing.

Theorem 2
If $|\mathcal{C}_3| < \lambda N$ for some constant λ then ConstrainedCyk has complexity $O(N^2)$.

Proof
The total number of cells in the chart for a string of length N is $N(N + 1)/2$, therefore $|\mathcal{C}_2| < N^2$. Further, the number of midpoints for any cell is less than N, hence $\mathcal{W}_3 < N$. If we bound $|\mathcal{C}_3| < \lambda N$, then it follows directly from Theorem 1 that ConstrainedCyk has complexity $O(N^2 + \lambda N \cdot N) = O(N^2)$. ■

We have just proved in Theorem 2 that we can reduce the complexity of constituent parsing from $O(N^3)$ to $O(N^2)$ by restricting the number of open cells in the chart via constraints $\overline{B}$ and $\overline{E}$. Next, we will show that this bound can be reduced even further to $O(N)$ when the number of words in B is held constant. We call this approach "linear constraints." There are two things to note when considering these constraints. First, the proof relies only on the size of set B, leaving $\overline{E}$ potentially empty and limiting the efficiency gains on short sentence compared to high-precision and quadratically bounded constraints. Second, limiting the size of B to a constant value is a restrictive criterion. When applied to a long sentence, this will over-constrain the search space and impose a branching bias on possible trees. In Section 7.2 we show that if the branching bias from linear constraints matches the branching bias of the treebank, then accuracy is not severely affected and linear constraints can provide a useful bound on long sentences that often dominate computational resources.

We first prove limits on the total number of open chart cells (case 3) and work required in these cells, which directly leads to the proof bounding the complexity of the algorithm.

Lemma 1
If $|B| \leq \lambda$ for some λ, then $\mathcal{W}_3 \leq \lambda|P|$.

Proof
It has been defined that $\mathcal{W}_1$ is zero and shown in Theorem 1 that $\mathcal{W}_2$ is constant. For chart cell $(b, e) \in \mathcal{C}_3$ there are $e - b$ possible midpoints to consider. For some midpoint m where $b \leq m < e$, if $w_m \notin B$ then $(m, e) \in \mathcal{C}_1$ and contains no complete or incomplete constituent entries. Therefore midpoint m is not a valid midpoint for cell (b, e). Because $|B| \leq \lambda$, there are at most λ midpoints to consider and the work required in (b, e) is bounded above by the constant number of productions in the grammar ($|P|$) and λ midpoints, thus $\mathcal{W}_3 \leq \lambda|P|$. ■

Lemma 2
If $|B| \leq \lambda$ for some λ, then $|\mathcal{C}_2| + |\mathcal{C}_3| \leq \lambda N$.

Proof
For a string of length N and some $w_b \in B$, there are at most N substrings $w_b \ldots w_e$ to consider, hence $O(N)$ chart cells for each word $w_b \in B$. Because $|B| \leq \lambda$, then there are at most λN cells $(b, e) \notin \mathcal{C}_1$ and $|\mathcal{C}_2| + |\mathcal{C}_3| \leq \lambda N$. ■

Theorem 3
If $|B| \leq \lambda$ for some λ then ConstrainedCyk has complexity $O(N)$.

Proof
From Theorem 1 we know that ConstrainedCyk has complexity $O(|\mathcal{C}_2| + |\mathcal{C}_3|\mathcal{W}_3)$. Given that $|B| \leq \lambda$, it follows from Lemmas 1 and 2 that ConstrainedCyk has complexity $O(\lambda^2 N|P|) = O(N)$. ■

In this section, we have been discussing worst-case complexity bounds, but there is a strong expectation for large typical-case complexity savings that aren't explicitly accounted for here. To provide some intuition as to why this might be, consider again the chart structure in Figure 2d. The black cells in the chart represent the cells that have been closed when $w_j \in \overline{B}$ (case 1 cells). Because there is no work required for these cells, the total amount of work required to parse the sentence is reduced. The quadratic bound does not include any potential reduction of work for the remaining open cells, however. Thus the amount of work to parse the sentence will be less than the worst-case quadratic bound because of this reduction in processing. In Section 7.2 we compute the empirical complexity of each constraint method and compare that with its theoretical bound.

5. Tagging Chart Constraints

To better understand the proposed tagging tasks and their likely utility, we will first look at the distribution of classes and our ability to automatically assign them correctly.

Note that we do not consider the first word w_1 and the last word w_N during the begin-constituent and end-constituent prediction tasks because they are unambiguous in terms of whether they begin or end constituents of span greater than one. The first word w_1 must begin a constituent spanning the whole string, and the last word w_N must end that same constituent. The first word w_1 cannot end a constituent of length greater than 1; similarly, the last word w_N cannot begin a constituent of length greater than 1. We therefore omit $\overline{B}$ and $\overline{E}$ at these two word positions from prediction, leading to $N - 2$ begin-constituent and $N - 2$ end-constituent ambiguous predictions for a string of length N.

Table 1 displays word statistics from the training sections of the Penn English treebank (Marcus, Marcinkiewicz, and Santorini 1993) and the Penn Chinese treebank (Xue et al. 2005), and the positive and negative class label frequency of all words in the data.

From the nearly 1 million words in the English corpus, just over 870,000 are neither the first nor the last word in the string, therefore possible members of the sets $\overline{B}$ or $\overline{E}$ (i.e., neither beginning a multi-word constituent ($\overline{B}$) nor ending a multi-word constituent ($\overline{E}$)). Of these 870,000 words, over half (50.5%) do not begin multi-word constituents, and nearly three quarters (74.3%) do not end multi-word constituents. The skewed distribution of E to $\overline{E}$ reflects the right-branching structure of English. Finally, almost 90% of words are not labeled with a single-word constituent, demonstrating the infrequency at which these productions occur in the English treebank.

The Chinese treebank is approximately half of the size of the English treebank in terms of the number of sentences and word count. Again, we see a bias towards right-branching trees ($|B| > |E|$), but the skew of the distributions is much smaller than it is for English. The most significant difference between the two corpora is the percentage of single-word constituents in Chinese compared with English. Due to the annotation guidelines of the Chinese treebank, nearly 40% of words contain single-word constituent unary productions. Because these occur with such high frequency, we can assume that unary constraints may have a smaller impact on efficiency for Chinese than for English, because an accurate tagger will constrain fewer cells. We still have the expectation, though, that accurate tagging of SWC productions will increase parsing accuracy for both English and Chinese.

To automatically predict the class of each word position, we train a binary predictor from supervised data for each language/word-class pair, tuning performance on the respective development sets (WSJ Section 24 for English and Penn Chinese Treebank articles 301-325 for Chinese). Word classes are extracted from the treebank trees by observing constituents beginning or ending at each word position, or by observing single word constituents. We use the tagger from Hollingshead, Fisher, and Roark (2005) to train six linear models (three for English, three for Chinese) with the averaged perceptron algorithm (Collins 2002).

Table 2 summarizes the features implemented in our tagger for $\overline{B}$, $\overline{E}$, and $\overline{U}$ identification. In the table, the φ features are instantiated as POS-tags (provided by a separately trained log-linear POS-tagger) and τ features are instantiated as constituent tags ($\overline{B}$, $\overline{E}$, and $\overline{U}$ class labels). The feature set used in the tagger includes the n-grams of surrounding words, the n-grams of surrounding POS-tags, and the constituent tags of

Table 1
Statistics on extracted word classes for English (Sections 2–21 of the Penn WSJ treebank) and Chinese (articles 1–270 and 400–1151 of the Penn Chinese treebank).

	Corpus totals		**Begin class**		**End class**		**Unary class**	
	Strings	Words	B	$\overline{B}$	E	$\overline{E}$	U	$\overline{U}$
English								
Count	39,832	950,028	430,841	439,558	223,544	646,855	105,973	844,055
Percent			49.5	50.5	25.7	74.3	11.2	88.8
Chinese								
Count	18,086	493,708	188,612	269,000	165,591	292,021	196,732	296,976
Percent			41.2	58.8	36.2	63.8	39.9	60.1

Table 2
Tagger features for $\overline{B}$, $\overline{E}$, and $\overline{U}$.

	LEX	ORTHO	POS
τ_i	τ_i, w_i	$\tau_i, w_i[0]$	τ_i, φ_i
τ_{i-1}, τ_i	τ_i, w_{i-1}	$\tau_i, w_i[0..1]$	τ_i, φ_{i-1}
τ_{i-2}, τ_i	τ_i, w_{i+1}	$\tau_i, w_i[0..2]$	$\tau_i, \varphi_{i-1}, \varphi_i$
$\tau_{i-2}, \tau_{i-1}, \tau_i$	τ_i, w_{i-2}	$\tau_i, w_i[0..3]$	τ_i, φ_{i+1}
	τ_i, w_{i+2}	$\tau_i, w_i[n]$	$\tau_i, \varphi_i, \varphi_{i+1}$
	τ_i, w_{i-1}, w_i	$\tau_i, w_i[n\text{-}1..n]$	$\tau_i, \varphi_{i-1}, \varphi_i, \varphi_{i+1}$
	τ_i, w_i, w_{i+1}	$\tau_i, w_i[n\text{-}2..n]$	τ_i, φ_{i-2}
		$\tau_i, w_i[n\text{-}3..n]$	$\tau_i, \varphi_{i-2}, \varphi_{i-1}$
		$\tau_i, w_i \subseteq$ Digit	$\tau_i, \varphi_{i-2}, \varphi_{i-1}, \varphi_i$
		$\tau_i, w_i \subseteq$ UpperCase	τ_i, φ_{i+2}
		$\tau_i, w_i \subseteq$ Hyphen	$\tau_i, \varphi_{i+1}, \varphi_{i+2}$
			$\tau_i, \varphi_i, \varphi_{i+1}, \varphi_{i+2}$

All lexical (LEX), orthographic (ORTHO), and part-of-speech (POS) features are duplicated to also occur with τ_{i-1}; e.g., $\{\tau_{i-1}, \tau_i, w_i\}$ as a LEX feature.

the preceding words. The *n*-gram features are represented by the words within a three-word window of the current word. The tag features are represented as unigram, bigram, and trigram tags (i.e., constituent tags from the current and two previous words). These features are based on the feature set implemented by Sha and Pereira (2003) for NP chunking. Additional orthographical features are used for unknown and rare words (words that occur fewer than five times in the training data), such as the prefixes and suffixes of the word (up to the first and last four characters of the word), and the presence of a hyphen, a digit, or a capitalized letter, following the features implemented by Ratnaparkhi (1999). Note that the orthographic feature templates, including the prefix (e.g., $w_i[0..1]$) and suffix (e.g., $w_i[n\text{-}2..n]$) templates, are only activated for unknown and rare words. When applying our tagging model to Chinese data, all feature functions were left in the model as-is, and not tailored to the specifics of the language.

We ran various tagging experiments on the development set and report accuracy results in Table 3 for all three predictions tasks, using Viterbi decoding. We trained

Table 3
Tagging accuracy on the respective development sets (WSJ Section 24 for English and Penn Chinese Treebank articles 301–325 for Chinese) for binary classes $\overline{B}$, $\overline{E}$, and $\overline{U}$, for various Markov orders.

Tagging Task	**Markov order**		
	0	**1**	**2**
English			
$\overline{B}$ (no multi-word constituent begin)	96.7	96.9	96.9
$\overline{E}$ (no multi-word constituent end)	97.3	97.3	97.3
$\overline{U}$ (no span-1 unary constituent)	98.3	98.3	98.3
Chinese			
$\overline{B}$ (no multi-word constituent begin)	94.8	95.4	95.2
$\overline{E}$ (no multi-word constituent end)	96.2	96.4	96.6
$\overline{U}$ (no span-1 unary constituent)	95.9	96.2	96.3

models with Markov order-0 (constituent tags predicted for each word independently), order-1 (features with constituent tag pairs), and order-2 (features with constituent tag triples). In general, tagging accuracy for English is higher than for Chinese, especially for the $\overline{U}$ and $\overline{B}$ tasks. Given the consistent improvement from Markov order-0 to Markov order-1 (particularly on the Chinese data), and for the sake of consistency, we have chosen to perform Markov order-1 prediction for all results in the remainder of this article.

6. Experimental Set-up

In the sections that follow, we present empirical trials to examine the behavior of chart constraints under a variety of conditions. First, we detail the data, evaluation, and parsers used in these experiments.

6.1 Data Sets and Evaluation

For English, all stochastic grammars are induced from the Penn WSJ Treebank (Marcus, Marcinkiewicz, and Santorini 1993). Sections 2–21 of the treebank are used as training, Section 00 as held-out (for determining stopping criteria during training and some parameter tuning), Section 24 as development, and Section 23 as test set. For Chinese, we use the Penn Chinese Treebank (Xue et al. 2005). Articles 1–270 and 400–1151 are used for training, articles 301–325 for both held-out and development, and articles 271–300 for testing. Supervised class labels are extracted from the non-binarized treebank trees for B, E, and U (as well as their complements).

All results report F-measure labeled bracketing accuracy (harmonic mean of labeled precision and labeled recall) for all sentences in the data set (Black et al. 1991), and timing is reported using an Intel 3.00GHz processor with 6MB of cache and 16GB of memory. Timing results include both the pre-processing time to tag the chart constraints as well as the subsequent context-free inference, but tagging time is relatively negligible as it takes less than three seconds to tag the entire development corpus.

6.2 Tagging Methods and Closing Chart Cells

We have three separate tagging tasks, each with two possible tags for every word w_i in the input string: (1) B or $\overline{B}$; (2) E or $\overline{E}$; and (3) U or $\overline{U}$. Our taggers are as described in Section 5.

Within a pipeline system that leverages hard constraints, one may want to choose a tagger operating point that favors precision of constraints over recall to avoid over-constraining the downstream parser. We have two methods for trading recall for precision that will be detailed later in this section, both relying on calculating the cumulative score S_i for each of the binary tags at each word position w_i. That is, (using B as the example tag):

$$S_i(B \mid w_1 \ldots w_n) = \log \sum_{\tau_1 \ldots \tau_n} \delta(\tau_i, B)\, e^{\Phi(w_1 \ldots w_n, \tau_1 \ldots \tau_n) \cdot \mathbf{w}} \quad (1)$$

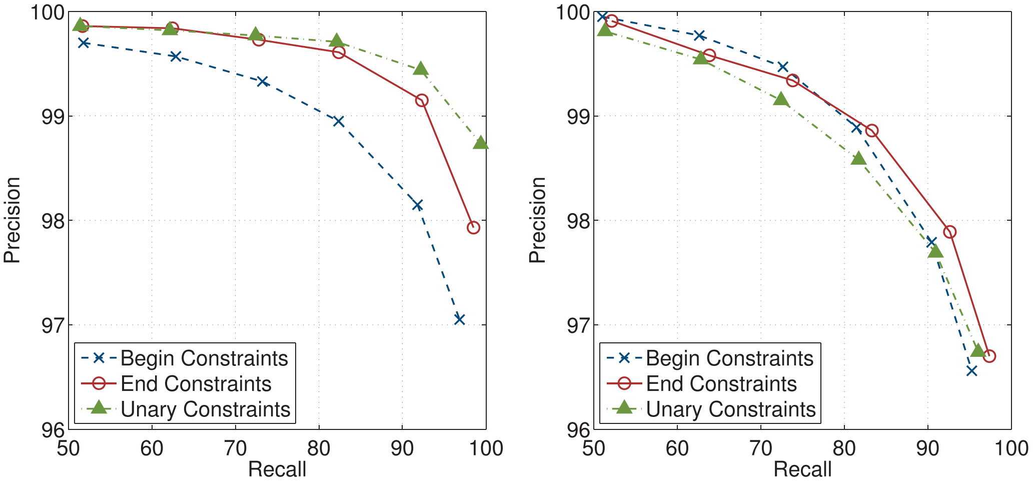
where $\sum$ sums over all possible tag sequences for sentence $w_1 \ldots w_n$; $\delta(\tau_i, B) = 1$ if $\tau_i = B$ and 0 otherwise; $\Phi(w_1 \ldots w_n, \tau_1 \ldots \tau_n)$ maps the word string and particular tag string to a d-dimensional (global) feature vector; and $\mathbf{w}$ is the d-dimensional parameter vector estimated by the averaged perceptron algorithm. Note that this cumulative score over all tag sequences that have B in position i can be calculated efficiently using the forward–backward algorithm. We can compare the cumulative scores $S_i(B)$ and $S_i(\overline{B})$ to decide how to tag word w_i, and define the cumulative score ratio (CSR) as follows:

$$\mathrm{CSR}(w_i) = S_i(\overline{B}) - S_i(B) \tag{2}$$

If we want to tag $\overline{B}$ with high precision, and thus avoid over-constraining the parser, we can change our decision criterion to produce fewer such tags. We present two different selection criteria in the next two subsections.

To show the effect of precision-oriented decision criteria, Figure 3 shows the precision/recall tradeoff at various operating points, using the global high-precision method detailed herein, for all three tags on both the English and the Chinese development sets. As expected, we see that the English $\overline{B}$ curve is significantly lower than the English $\overline{E}$ and $\overline{U}$ curves. This is due to the near-uniform prior on $\overline{B}$ in the data ($\overline{E}$ and $\overline{U}$ are much higher-frequency classes). Still, we can achieve 99% precision for $\overline{B}$ with recall above 70%. We do much better with $\overline{E}$ and $\overline{U}$ and see that when precision is 99% for these two tags, recall does not drop below 90%. For the Chinese tagging task, $\overline{B}$, $\overline{E}$, and $\overline{U}$ all have similar performance as we trade precision for recall. Here, as with the English $\overline{B}$ tag, we achieve 99% precision with recall still above 70%.

We can see from these results that our finite-state tagging approach yields very high accuracy on these tasks, as well as the ability to provide high precision (above 99%) operating points with a tolerable loss in recall. In what follows, we present two approaches to adjusting precision: first by adjusting the overall precision and recall of the tagger directly, as shown here; and second, by adjusting the precision and recall of tagging results on a per-sentence basis. We then discuss how complexity-bounded constraints are implemented in our experiments. In Section 7.1 we discuss empirical results showing how adjusting the tagger in these ways affects parsing performance.



(a) English

(b) Chinese

Figure 3
Tagger precision/recall tradeoff of $\overline{B}$, $\overline{E}$, and $\overline{U}$ on the development set for English (a) and Chinese (b).

6.2.1 Global High Precision. Global high precision constraints (GHP) are implemented using the cumulative score ratios directly to determine if $w_b \in B$ or $\overline{B}$.[4] Given the cumulative score ratio as defined herein, we define the set $\overline{B}$ under global high precision constraints as:

$$\mathrm{GHP}_\lambda(w_i \ldots w_N) = \{w_i \in \overline{B} : \mathrm{CSR}(w_i) > \lambda\} \tag{3}$$

The default decision boundary $\lambda = 0$ provides high tagging accuracy, but as mentioned in Section 6.2 we may want to increase the precision of the tagger so that the subsequent parser is not over constrained. To do this, we increase the threshold parameter λ towards positive infinity, potentially moving some words in the corpus from $\overline{B}$ to B. The same procedure is applied to E and U tags.

6.2.2 Sentence-Level High Precision. Global high precision constraints, as defined here, affect the precision of the tagger over the entire corpus, but may or may not affect individual sentences. Because parsing operates on a sentence-by-sentence basis, it may be beneficial to modify the precision of the tagger based on the current sentence being processed. We call this approach sentence-level high precision (HP).[5]

To increase tagging precision at the sentence level, we compute the cumulative scores S_i and the cumulative scores ratio (CSR) at all word positions as described in this article. We next tag all word positions with a $\mathrm{CSR}(w_i)$ score less than zero with B, and rank the remaining word positions according to their CSR score. The lowest-ranking λ fraction of this sorted list are tagged as $\overline{B}$, and the rest default to B. When $\lambda = 0$, zero percent of words are tagged with $\overline{B}$ (i.e., no constraints are applied); and when $\lambda = 1$, 100 percent of the words with $\mathrm{CSR}(w_i) > 0$ are tagged with $\overline{B}$. This ensures that the high-precision threshold is adapted to each sentence, even if its absolute CSR scores are not similar to others in the corpus.

6.2.3 Quadratic Bounds. We impose quadratic bounds on context-free parsing by restricting the number of open cells (case 3 in Section 4.3) to be less than or equal to λN for some tuning parameter λ and sentence length N. This only involves the sets B and E, as unary constraints are restricted to span-1 cells. Given gold parse trees t^* and a function $T(B, E)$ to generate the set of all valid parse trees satisfying constraints B and E, the optimal quadratically bounded constraints would be:

$$\mathrm{Quad}_\lambda(w_i \ldots w_N) = \operatorname*{argmax}_{B,E}\big(\max_{t \in T(B,E)} F_1(t^*, t) \text{ s.t. } |\mathcal{C}_3| \leq \lambda N\big) \tag{4}$$

that is, the set of constraints that provides the optimal F-score of the best remaining candidate in the chart while still imposing the required bound on the number of $\mathcal{C}_3$ cells. This equation is an instance of combinatorial optimization and solving it exactly is NP-complete. Furthermore, we do not have access to gold parse trees t^* during parsing and must rely on the posterior scores of the tagger.

We instead use a greedy approach to approximate Equation (4). To accomplish this, at every sentence we first sort both the B and E CSR scores for each word position

[4] In our experiments these scores are in the range $\pm 10^3$.

[5] As a historical note, we referred to this approach as simply "high precision constraints" in Roark and Hollingshead (2008) and Roark and Hollingshead (2009).

into a single list. Next, we assume all word positions are in $\overline{B}$ and $\overline{E}$, then starting from the top of the sorted list (highest posterior probability for set inclusion), we continue to add word positions to their respective open set and compute the number of open cells with the given constraints while $|\mathcal{C}_3| < \lambda N$. By doing this, we guarantee that only a linear number of case 3 cells are open in the chart, which leads to quadratic worst-case parsing complexity.

6.2.4 Linear Bounds. Imposing $O(N)$ complexity bounds requires constraining the size of the set B such that $|B| \leq \lambda$ for some constant λ. As with quadratic complexity bounds, we wish to find the optimal set B to fulfill these requirements:

$$\text{Linear}_\lambda(w_i \ldots w_N) = \underset{B}{\text{argmax}}(\max_{t \in T(B)} F_1(t^*, t) \text{ such that } |B| \leq \lambda) \tag{5}$$

We again resort to a greedy method to determine the set B, as this optimization problem is still NP-complete and gold trees are not available. B is constructed by sorting all word positions by their CSR scores for B, and then adding only the highest-scoring λ entries to the inclusion set. All other word positions are closed and in the set $\overline{B}$.

Because this method does not consider the set E to impose limits on processing, it will be shown in Section 7.2 that $O(N)$ complexity bounding is not as effective as a stand-alone method when compared to the quadratic complexity or high precision constraints presented here.

6.3 Parsers

We will present results constraining several different parsers. We first present results for exhaustive parsing using both the CYK and the CONSTRAINEDCYK algorithms. We use a basic CYK exhaustive parser, termed the BUBS parser,[6] to parse with a simple PCFG model that uses non-terminal node-labels as provided by the Penn Treebank, after removing empty nodes, node indices, and function tags. The results presented here replicate and extend the results presented in Roark and Hollingshead (2009), using a different CYK parser.[7] The BUBS parser is an open-source high-efficiency parser that is grammar agnostic and can be run in either exhaustive mode or with various approximate inference options (detailed more fully in Section 8.1). It has been shown to parse exhaustively with very competitive or superior efficiency compared with other highly optimized CYK parsers (Dunlop, Bodenstab, and Roark 2011). In contrast to the results in Roark and Hollingshead (2009), here we present results with both left- and right-binarized PCFGs induced using a Markov order-2 transform, as detailed in Section 3.3, and also present results for parsing Chinese.

We will then present results applying finite-state chart constraints to state-of-the-art parsers, and evaluate the additional efficiency gains these constraints provide, even when these parsers are already heavily pruned. To simplify the presentation of these

[6] `http://code.google.com/p/bubs-parser.`

[7] Note that there are several differences between the two parsers, including the way in which grammars are induced, leading to different baseline accuracies and parsing times. Most notably, the parser from Roark and Hollingshead (2009) relied upon POS-tagger output, and collapsed unary productions in a way that effectively led to parent annotation in certain unary chain constructions. The current results do not exploit those additional annotations, hence the baseline F-measure accuracy is a couple of percentage points lower.

results, we concentrate in Section 7.1 on parsing with the simple Markov order-2 grammars, and then move to higher accuracy parsers in Section 8.

7. CYK Parsing with Finite-State Chart Constraints

7.1 High-Precision Constraints

As mentioned in Section 6.2, we can adjust the severity of pruning to favor precision over recall. Figure 4a shows parse time versus F-measure labeled parse accuracy on the English development set for the baseline (unconstrained) exact-inference CYK parser, and for various parameterizations of global high precision (GHP), sentence-level high precision (HP), and unary constraints. Note that we see accuracy increasing over the baseline in Figure 4a with the imposition of these constraints at some operating points. This is not surprising, though, as the finite-state tagger makes use of lexical

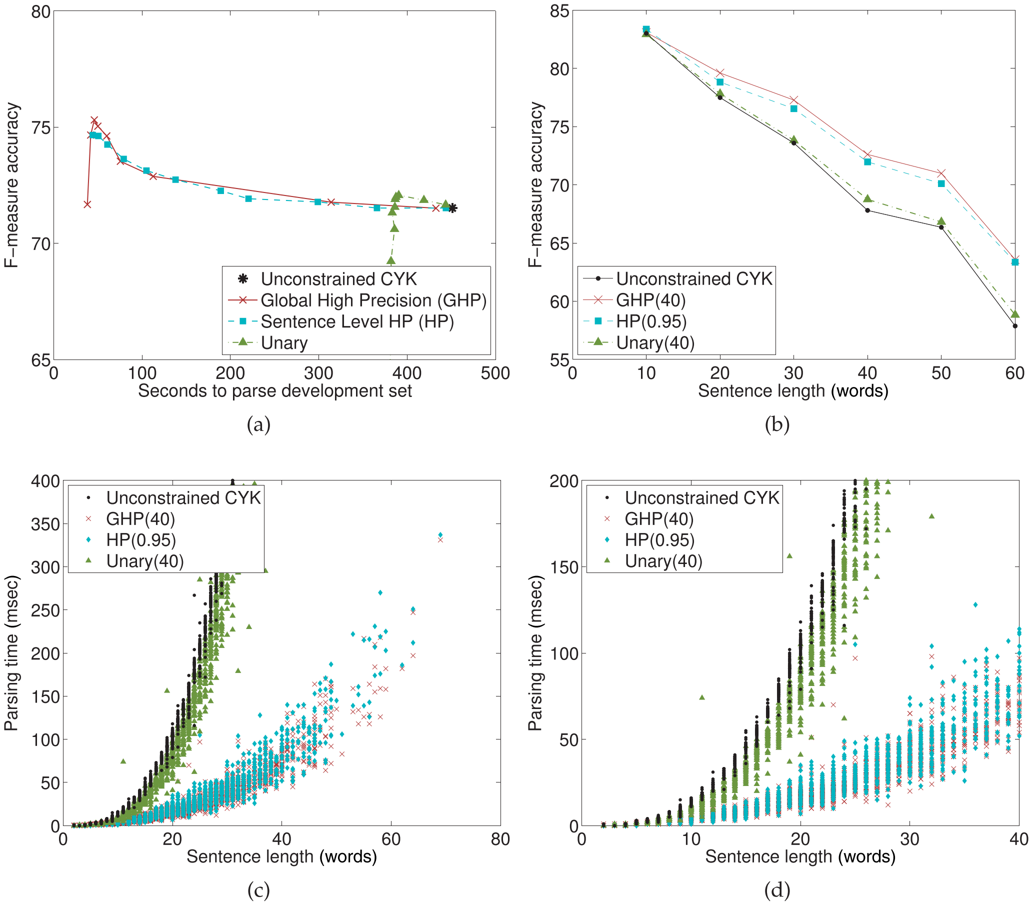


Figure 4
English development set results (WSJ Section 24) applying global high precision (GHP), sentence-level high precision (HP), and unary constraints with the CONSTRAINEDCYK algorithm. We sweep over multiple values of λ in (a) and plot results in (b), (c), and (d) with the optimal value for each constraint found in (a). F-measure accuracy in (b) is computed over binned sentence lengths. Figure (d) plots the same data as (c), but zoomed in.

information that the simple PCFG does not, hence there is complementary information added that improves the model. The best operating points—fast parsing and relatively high accuracy—are achieved for GHP constraints at $\lambda = 40$, for sentence-level HP at $\lambda = 0.95$, and for unary constraints at $\lambda = 40$. These high-precision parameterizations achieve roughly an order of magnitude speed-up and between 0.2 absolute (for unary constraints) and 3.7 absolute (for high-precision constraints) F-measure improvement over the baseline unconstrained parser.

In order to analyze how these constraints affect accuracy with respect to sentence length, we turn to Figure 4b. In this plot, F-measure accuracy is computed over binned sentence lengths in increments of ten. All sentences of length greater than 60 are included in the final bin. As one might expect, accuracy declines with sentence length for all models because the number of possible trees is exponential in the sentence length and errors in one portion of the tree can adversely affect prediction of other constituents in nearby structures. We see that all constraints provide accuracy gains over the baseline at all sentence lengths, but point out that the gains by GHP B and E constraints are larger for longer sentences. As stated earlier, this is due to the model-correcting behavior of cell constraints: Lexical features are leveraged to prune trees that the PCFG may favor but are not syntactically correct with respect to the entire sentence. Because longer sentences are poorly modeled by the PCFG, cell constraints play a larger role in restricting the space of possible trees considered and correcting modeling errors.

We can get a different perspective of how these constraints affect parsing time by considering the scatter plots in Figures 4c and 4d, which plot each sentence according to its length and parsing time at four operating points: baseline (unconstrained); global high precision at $\lambda = 40$; sentence-level high precision at $\lambda = 0.95$; and unary at $\lambda = 40$. Figure 4c shows data points with up to 80 words and 400 msec of parsing time. Figure 4d zooms in to under 200 msec and up to 40 words. It can be seen in each graph that unconstrained CYK parsing quickly leaves the plotted area via a steep cubic curve (least-squares fit is $N^{2.9}$). Unary constraints operate at the same cubic complexity as the baseline, but with a constant factor decrease in parsing time (least-squares fit is also $N^{2.9}$). The plots for GHP and HP show dramatic decreases in typical-case runtime compared with the baseline. We again run a least-squares fit to the data and find that both GHP and HP constraints follow a $N^{2.5}$ trajectory.

The empirical complexity and run-time performance of GHP and HP are nearly identical relative to all other chart constraints. But looking a little closer, we see in Figures 4c and 4d that that GHP constraints are slightly faster than HP for some sentences, and accumulated over the entire development set, this leads to both higher accuracy (75.1 vs. 74.6 F-measure) and faster parsing time (46 vs. 50 seconds).[8] This leads to the conclusion that when optimizing parameters for high-precision constraints, we can better tune the overall pipeline by choosing an operating point based on the corpus-level tagger performance as opposed to tuning for sentence-specific precision goals. Consequently, other than Table 4, we only apply GHP constraints on test set results in the remainder of this section.

Although Figure 4a displays the constrained F-measure parse accuracy as a function of parsing time, one may also be interested in how tagger precision directly affects parse accuracy. To answer this question, we apply high precision constraints to sets B and E in isolation and plot results in Figure 5. Note that when only constraints on E are applied, no chart cells can be completely closed and parsing time does not significantly decrease.

[8] See Table 4 for additional performance comparisons.

Likewise, when we apply constraints on B, a chart cell is either open to all constituents (case 3) or closed to all constituents (case 1). When constraining either B or E, we are predicting the structure of the final parse tree, which (as can be seen in Figure 5) has large effects on parse F-measure.

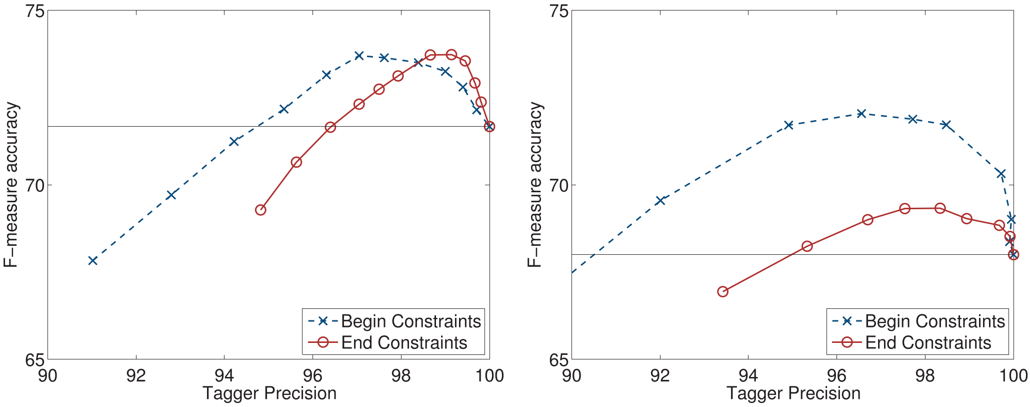
We point out in Figure 5 that to achieve optimal parsing F-measure, tagger precision must be above 97% for both English and Chinese. For both languages, B constraints are more forgiving and do not adversely affect parsing F-measure as quickly as E constraints. These results may lead one to tune two separate high-precision parameters—one to constrain B and one to constrain E. We ran such experiments but found no significant gains compared to tying these parameters together.

7.2 Complexity-Bounded Constraints

Figure 6a plots F-measure accuracy versus time to parse the entire development set for two complexity bounded constraints: $O(N^2)$ and $O(N)$. We also include the previously plotted global high-precision constraints for comparison. The large asterisk in the plot depicts the baseline accuracy and efficiency of standard CYK parsing without constraints. We sweep over various parameterizations for each method, from very lightly constrained to very heavily constrained. The complexity-bounded constraints are not combined with the high-precision constraints for this plot (but are later in the article).

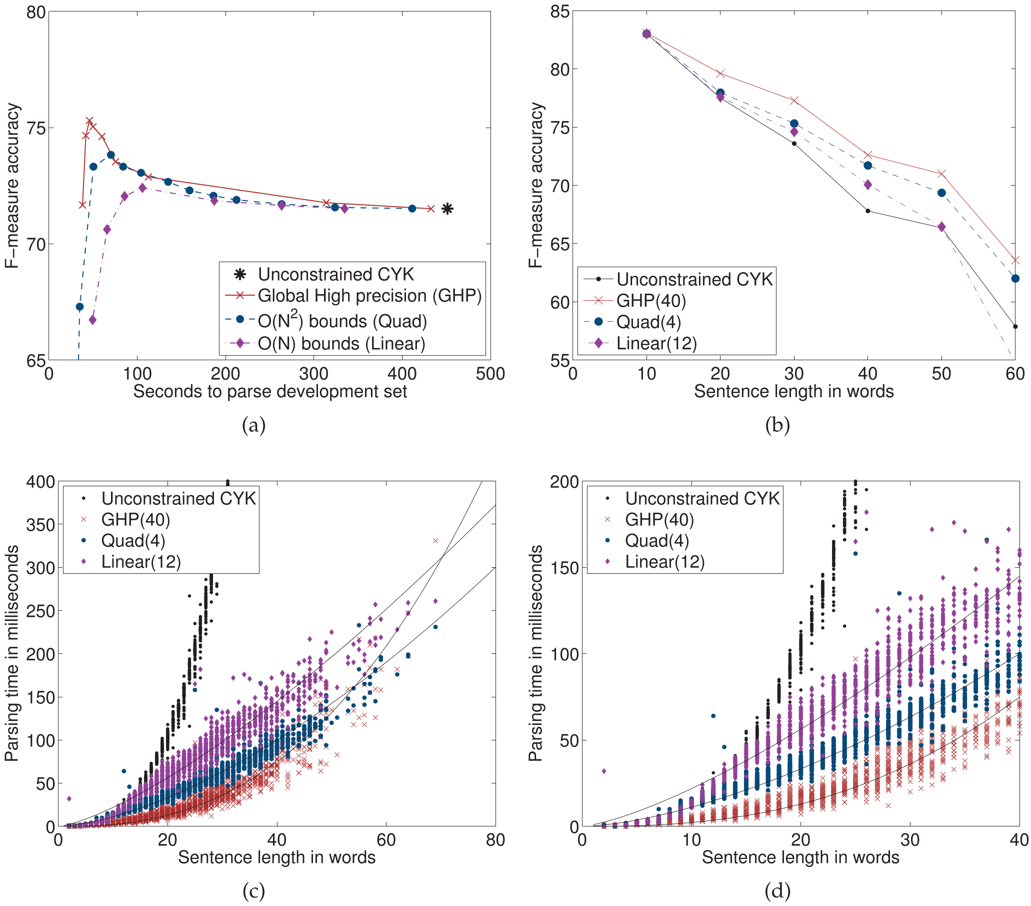
As can be seen in Figure 6a, the linear-bounded method does not, as applied, achieve a favorable accuracy/efficiency tradeoff curve compared with the quadratic bound or high-precision constraints. This is not surprising, given that no words are excluded from the set E for this method, hence far fewer constraints overall are applied with the linear-bounded constraints.

In addition, we see in Figure 6b that unlike high precision and quadratic constraints, the linear method hurts accuracy on longer sentences over the baseline unconstrained algorithm, notably for sentences greater than 50 words. We attribute this to the fact that for longer sentences, say length 65, only $\lambda = 16$ words are allowed in B for linear



(a) English (b) Chinese

Figure 5
The effects of tagger precision on parsing F-measure. B and E constraints are applied in isolation; no other constraints are used during parsing. Results on the English development set in (a) and Chinese in (b). Baseline unconstrained F-measure accuracy is indicated with the horizontal black line.



(a)

(b)

(c)

(d)

Figure 6
English development set results (WSJ Section 24), applying complexity-bounding constraints with the ConstrainedCyk algorithm. We sweep over multiple values of λ in (a) and plot results in (b), (c), and (d) with the optimal value for each constraint found in (a). F-measure accuracy in (b) is computed over binned sentence lengths. Figure (d) plots the same data as (c), but zoomed in.

constraints, which severely limits the search space for these sentences to the point of pruning gold constituents and decreasing overall accuracy.

Next we turn to the scatter plots of Figures 6c and 6d. Fitting an exponential curve to the data for each constraint via least-squares, we find that global high-precision constraints follow a $N^{2.5}$ trajectory, quadratic at $N^{1.6}$, and linear at $N^{1.4}$. It is interesting that quadratic constraints actually perform at sub-quadratic run-time complexity. This is because the quadric complexity proof in Section 4.4 assumes that the linear number of open cells each process $O(N)$ midpoints. But in practice, many midpoints are not considered due to the heavily constrained chart, decreasing the average-case runtime of ConstrainedCyk with quadratically bounded constraints.

Also interesting is that linear constraints perform worse than $O(N)$ at $N^{1.4}$. We attribute this to the nature of the data set. When parsing with linear constraints, we see that for short sentences parsing complexity is actually cubic. Because we allow a constant number of word positions in the open set B, sentences with length less than λ are completely unconstrained and all chart cells remain open. In Figure 6d it looks as if parsing becomes linear after the sentence length notably exceeds λ, around $N = 20$. Assuming our corpus contained a disproportionate number of long sentences, empirical complexity of linear constraints should approach $O(N)$, but due to the large number of short sentences, we see an empirical complexity greater than $O(N)$.

Three important points can be taken away from Figure 6. First, although we can provably constrain parsing to linear speeds, in practice this method is inferior to both quadratically bounded constraints and high-precision constraints. Second, high-precision constraints are more accurate (see Figure 6b) and more efficient (see Figure 6d) for shorter strings than the quadratically bound constraints; yet with longer strings the quadratic constraints better control parsing time than the high-precision constraints (see Figure 6c). Finally, at the "crossover" point, where quadratic constraints start becoming more efficient than high-precision constraints (roughly 50–60 words, see Figure 4c), there is a larger variance in the parsing time with high-precision constraints versus those with quadratic bounds. This illustrates the difference between the two methods of selecting constraints: High-precision constraints can provide very strong typical-case gains, but there is no guarantee of worst-case performance. In this way, the high-precision constraints are similar to other tagging-derived constraints like POS-tags or chunks.

7.3 Combining Constraints

Depending on the length of the string, the complexity-bound constraints may close more or fewer chart cells than the high-precision constraints—more for long strings, fewer for short strings. We can achieve worst-case bounds along with superior typical-case speed-ups by combining both methods. This is accomplished by taking the union of high-precision $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints with their respective complexity-bounded sets.

When combining complexity-bound constraints with high-precision constraints, we first chose operating parameters for each complexity-bounded method at the point where efficiency is greatest and accuracy has yet to decline. These operating points can be seen as the "knee" of the curves in Figure 6a. For the quadratic complexity method, we set $\lambda = 4$, limiting the number of open cells to $4N$. For the linear complexity method, we set $\lambda = 12$, limiting the number of word positions in B to a maximum of 12 members.

Table 4 displays F-measure accuracy and parsing time (in seconds) for many individual and combined constraints on the development set: unconstrained CYK parsing; unary constraints; global high-precision (GHP) and sentence-level high-precision (HP) constraints; and $O(N^2)$ and $O(N)$ complexity-bounded constraints (Quad and Linear, respectively). We present all parsing results in Table 4 using both a left- and right-binarized Markov order-2 grammar so that the effects of grammar binarization on finite-state constraints can be evaluated. Pre-processing the grammar with a right or left binarization alters the nature and distribution of child non-terminals for grammar productions. Because $\overline{B}$ and $\overline{E}$ constraints prune the chart differently depending on the grammar binarization, we suspect that one method may outperform the other due to the branching bias of the language being parsed.

We find three general trends in Table 4. First, the efficiency benefits of combining constraints are relatively small. We suspect this is because the data set contains mostly shorter sentences. Global high-precision constraints outperform the complexity bounded constraints on sentences of length 10 to 50, which makes up the majority of the development set. It is not until we parse longer sentences that the trends start to differ and exhibit characteristics of the complexity bounds. Thus by combining high-

Table 4
English development set results (WSJ Section 24) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars under various individual and combined constraints.

	Constraints	F_1	Precision	Recall	Seconds	Speed-up
Right-binarized grammar	None (baseline CYK)	71.5	74.5	68.8	451	
	GHP(40)	75.3	78.6	72.3	46	9.8x
	HP(0.95)	74.6	77.8	71.7	50	8.9x
	Quad(4)	73.8	77.0	70.9	70	6.4x
	Linear(12)	72.4	75.4	69.6	106	4.3x
	Unary(40)	72.0	75.4	68.9	386	1.2x
	HP(0.95) + Quad(4)	74.6	77.8	71.7	48	9.5x
	HP(0.95) + Linear(12)	74.4	77.6	71.5	48	9.5x
	GHP(40) + Quad(4)	75.3	78.5	72.3	45	10.0x
	GHP(40) + Linear(12)	75.1	78.4	72.1	44	10.2x
	GHP(40) + Unary(40)	75.7	79.4	72.4	34	13.4x
	GHP(40) + Unary(40) + Quad(4)	75.8	79.5	72.4	33	13.9x
Left-binarized grammar	None (baseline CYK)	71.7	74.5	69.1	774	
	GHP(40)	75.4	78.5	72.5	66	11.2x
	HP(0.95)	74.9	77.9	72.1	75	10.3x
	Quad(4)	74.0	77.0	71.2	99	7.8x
	Linear(24)	71.7	74.5	69.1	448	1.7x
	Unary(40)	71.9	75.2	69.0	607	1.3x
	HP(0.95) + Quad(4)	74.9	78.0	72.1	69	11.2x
	HP(0.95) + Linear(24)	74.7	77.8	71.9	71	11.0x
	GHP(40) + Quad(4)	75.4	78.5	72.5	62	12.6x
	GHP(40) + Linear(24)	75.2	78.4	72.3	62	12.6x
	GHP(40) + Unary(40)	75.7	79.3	72.5	37	20.9x
	GHP(40) + Unary(40) + Quad(4)	75.7	79.3	72.5	37	20.9x

precision and complexity constraints, we attain the typical-case efficiency benefits of high-precision constraints with worst-case complexity bounds.

Second, we see that the efficiency gain combining unary and high-precision constraints is more than additive. Unary constraints alone for the right-binarized grammar decreased parsing time by 1.3x, but in conjunction with high-precision constraints, parsing time is decreased from 66 seconds to 37 seconds, an additional 1.8x speed-up. We suspect that this additional gain comes from cache efficiencies (due to the heavily pruned nature of the chart, the population of commonly-queried span-1 chart cells have a higher likelihood of remaining in high-speed cache, decreasing overall parsing time), but we leave empirical verification of this theory to future work.

The third trend we see in Table 4 is that there are significant efficiency differences when parsing with a right- or left-binarized grammar. The difference in baseline performance has been previously studied (Song, Ding, and Lin 2008; Dunlop, Bodenstab, and Roark 2010), and our results confirm that a right-binarized grammar is superior for parsing the WSJ treebank due to the right-branching bias of parse trees in this corpus. Furthermore, linear constraints were far less effective with a left-binarized grammar,

Table 5
English test set results (WSJ Section 23) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars.

	Constraints	F_1	Precision	Recall	Seconds	Speed-up
Right	None (baseline CYK)	71.7	74.6	69.0	628	
	Unary(40)	72.1	75.5	69.0	525	1.2x
	GHP(40)	75.8	79.0	72.8	75	8.4x
	GHP(40) + Unary(40)	76.1	79.8	72.7	57	11.0x
Left	None (baseline CYK)	72.0	74.8	69.4	1,063	
	Unary(40)	72.4	75.8	69.4	910	1.2x
	GHP(40)	76.1	79.3	73.2	106	10.1x
	GHP(40) + Unary(40)	76.4	80.0	73.1	60	17.9x

Table 6
Chinese test set results (PCTB Sections 271–300) for the ConstrainedCyk algorithm with both left- and right-binarized Markov order-2 grammars.

	Constraints	F_1	Precision	Recall	Seconds	Speed-up
Right	None (baseline CYK)	60.5	64.6	56.9	157	
	Unary(20)	62.3	67.6	57.8	141	1.1x
	GHP(20)	66.9	71.4	63.0	17	9.1x
	GHP(20) + Unary(20)	68.9	74.4	64.1	15	10.2x
Left	None (baseline CYK)	60.4	64.5	56.9	269	
	Unary(20)	62.1	67.2	57.8	234	1.2x
	GHP(20)	66.0	70.5	62.1	33	8.2x
	GHP(20) + Unary(20)	68.0	73.5	63.2	23	11.7x

requiring a tuning parameter of $\lambda = 24$ such that accuracy was not adversely affected (compare with $\lambda = 12$ for right-binarized results). This is also caused by the branching bias inherent in the treebank. For example, consider a left-binarized grammar and linear constraints; in the extreme case where $\lambda = 0$, only w_1 will be in the open set B, forcing all constituents in the final tree to start at w_1. This results in a completely left-branching tree. With a right-binarized grammar, only the last word position will be in E, resulting in a completely right-branching tree. Thus, an overly constrained linear-bounded parse will favor one branching direction over the other. Because the treebank is biased towards right-branching trees, a right-binarized grammar is more favorable when linear constraints are applied.

Finally, we note that after all constraints have been applied, the accuracy and efficiency differences between the two binarization strategies nearly disappear.

To validate the selected operating points on unseen data, we present results on the test sets for English in Table 5 and Chinese in Table 6.[9] We parse individually with global high-precision and unary constraints, then present the combined result as this gave the best performance on the development set. In all cases, we see efficiency improvements greater than 10-fold and accuracy improvements of 4.4 absolute F-measure for English,

[9] We used the development set (articles 301–325 of the Penn Chinese treebank) to tune chart constraint parameters for Chinese, exactly as was done for English.

and 8.4 absolute F-measure for Chinese. Note that these efficiency improvements are achieved with no additional techniques for speeding up search; modulo the cell closing mechanism, the CYK parsing is exhaustive—it explores all possible category combinations from all open child cells. Techniques such as coarse-to-fine, A^* parsing, or beam-search are all orthogonal to the current approach, and could be applied in conjunction to achieve additional speedups.

In the next section, we investigate the use of chart constraints with a number of high-accuracy parsers, and empirically evaluate the combination of our chart constraint methods with popular heuristic search methods for parsing. These parsers include the Charniak parser, the Berkeley parser, and an in-house beam-search parser.

8. High-Accuracy Parsing with Finite-State Chart Constraints

In this section we evaluate the additive efficiency gains provided by finite-state constraints to state-of-the-art parsers. We apply $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints to three parsers, all of which already prune the search space to make decoding time with large grammars more practical. Each high-accuracy parser that we evaluate also prunes the search space in a different way—agenda-based search, coarse-to-fine pruning, and beam-search. Applying finite-state constraints to these three distinct parsers demonstrates how our constraints interact with other well-known pruning algorithms. In what follows we describe the pruning inherent in the three parsers and how we apply finite-state constraints within each framework.

8.1 Parsers

8.1.1 Charniak Parser. The Charniak (2000) parser is a multi-stage, agenda-driven parser that can be constrained by pruning edges before they are placed on the agenda. The first stage of the Charniak parser uses an agenda and a simple PCFG to build a sparse chart, which is used to limit the search in later stages with the full model. We focus on this first stage, because it is here that we will be constraining the parser. The edges on the agenda and in the chart are dotted rules, as described in Section 3.3. When edges are created, they are pushed onto the agenda. Edges that are popped from the agenda are placed in the chart, and then combined with other chart entries to create new edges that are pushed onto the agenda. Once a complete edge spanning the whole string is placed in the chart, at least one full solution must exist. Instead of terminating the initial chart population at this point, a technique called "over-parsing" is used that continues adding edges to the chart (and agenda) until a parameterized number of additional edges have been added. A small over-parsing value will heavily constrain the search space of the later stages within the pipeline, and a large value will often increase accuracy at the expense of efficiency. Upon reaching the desired number of edges, the next stage of the pipeline receives the chart as input and any edges remaining on the agenda are discarded.

We constrain the first stage of the Charniak parser by restricting agenda edges. When an edge is created for cell (b, e), where $b < e$, it is not placed on the agenda if either of the following two conditions hold: 1) $w_b \in \overline{B}$; or 2) the edge is complete and $w_e \in \overline{E}$. With these constraints, a large number of edges that would have previously been considered in the first stage of this pipeline will now be ignored. This allows us to either reduce the amount of over-parsing, which will increase efficiency, or maintain the over-parsing threshold and expand the search space in more promising directions according to the chart constraints. In this article, we have chosen to do the latter. Note that speed-ups are still observed, presumably due to the parser finding a complete edge spanning the whole sentence more quickly, thus leading to slight reductions in total edges added to the chart.

8.1.2 Berkeley Parser. The Berkeley parser (Petrov and Klein 2007a) is a multi-level coarse-to-fine parser that operates over a set of coarse-to-fine grammars, $G_1 \ldots G_t$. At each grammar level, the inside and outside constituent probabilities are computed with coarse grammar G_i and used to prune the subsequent search within G_{i+1}. This continues until the sentence is parsed with the target grammar G_t. The initial grammar G_1 is a Markov order-0 grammar, and the target grammar G_t is a latent-variable grammar induced through multiple iterations of splitting and merging non-terminals to maximize the likelihood of the training data (Petrov and Klein 2007b). This explicit target grammar is very large, consisting of 4.3 million productions, 2.4 million of which are lexical productions. We refer to G_t as the Berkeley grammar.

We apply chart constraints to the Berkeley parser during the initial inside pass with grammar G_1. Inside scores are computed via the ConstrainedCyk algorithm of Algorithm 2 modulo the fact that the standard inside/outside sum over scores is used in lines 10, 13, and 17 instead of the Viterbi-max. It is unnecessary to constrain either the subsequent outside pass or the search with the following grammars $G_2 \ldots G_t$, as the coarse-to-fine algorithm only considers constituents remaining in the chart from the previous coarse grammar. Reducing the population of chart cells during the initial G_1 inside pass consequently prunes the search space for all levels of the coarse-to-fine search. We also note that even though there are $t = 6$ grammar levels in our experiments with the Berkeley parser, exhaustive parsing with G_1 consumes nearly 50% of the total parse time (Petrov, personal communication). Applying chart constraints during this initial pass is where we see the largest efficiency gain—much more than when chart constraints supplement coarse-to-fine pruning in subsequent passes.

8.1.3 BUBS Parser. The bottom–up beam-search parser (BUBS) is a variation of the CYK algorithm where, at each chart cell, all possible edges are sorted by a Figure-of-Merit (FOM) and only the k-best edges are retained (Bodenstab et al. 2011). We follow this setup and use the Boundary FOM (Caraballo and Charniak 1997), but do not apply beam-width prediction in these experiments as chart constraints and beam-width prediction prune the search space in similar ways (see Bodenstab et al. [2011] for a comparison of the two methods). The BUBS parser is grammar agnostic, so to achieve high accuracy we parse with the Berkeley latent variable grammar (G_t described in the previous subsection), yet only require a single pass over the chart. The BUBS parser performs Viterbi decoding and does not marginalize over the latent variables or compute the max-rule solution as is done in the Berkeley parser. This leads to a lower F-measure score in the final results even though both parsers use the same grammar.

In this article, we apply finite-state constraints to the BUBS parser in a fashion almost identical to the ConstrainedCyk algorithm. Because the BUBS parser is a beam-search parser, the difference is that instead of retaining the max score for all non-terminals A_i at each chart cell, we only retain the max score for the k-best non-terminals. Otherwise, $\overline{B}$, $\overline{E}$, and $\overline{U}$ constraints are used to prune the search space in the same way.

8.2 High-Accuracy Parsing Results

Figure 7 displays accuracy and efficiency results of applying three independent constraints to the three parsers: high precision, quadratically bounded, and unary constraints. We sweep over possible tuning parameters from unconstrained (baseline

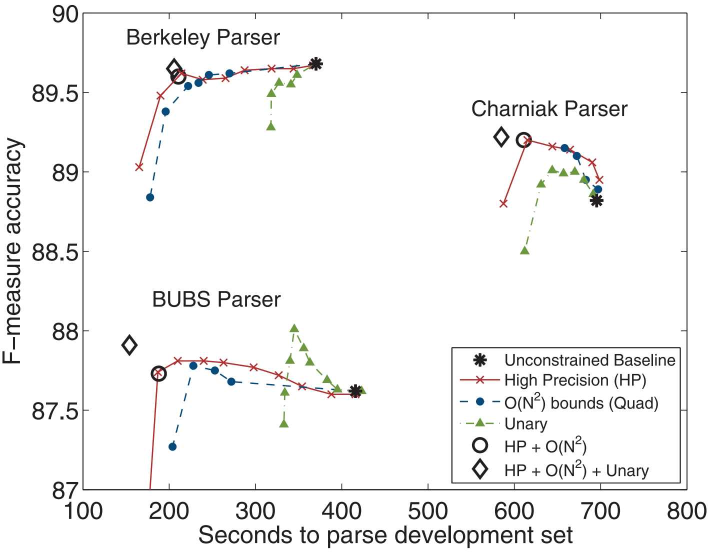


Figure 7
English accuracy and efficiency results of applying high precision, quadratic, and unary constraints at multiple values of λ to the Charniak, Berkeley, and BUBS parsers, all of which already heavily prune the search space.

asterisk) to overly constrained such that accuracy is adversely affected. We also plot the optimal combination of high precision and quadratic constraints (diamond) and the combination of all three constraints (circle) for each parser which was computed via a grid search over the joint parameter space.

There are many interesting patterns in Figure 7. First, all three constraints independently improve the accuracy and efficiency of all three parsers, with the exception of accuracy in the Berkeley parser. This is a powerful result considering each of these parsers is simultaneously performing various alternative forms of pruning, which were (presumably) tuned for optimal accuracy and efficiency on this same data set. We also note that the efficiency gains from all three constraints are not identical. In particular, high precision and quadratic constraints outperforms unary constraints in isolation. But this should be expected as unary constraints only partially close $O(n)$ chart cells whereas both high precision and quadratic constraints affect $O(n^2)$ chart cells. Nevertheless, looking at the optimal point combining all three constraints, we see that adding unary constraints to begin/end constraints does provide additional gains (in both accuracy and efficiency) for the BUBS and Charniak parsers.

The Berkeley parser appears to benefit from $\overline{B}$ and $\overline{E}$ constraints, but sees almost no gain from unary constraints. The reason for this has to do with the implementation details of combining (joining) two child cells within the inner loop of the CYK algorithm (line 8 in Algorithm 1). In bottom–up CYK parsing, to extend derivations of adjacent substrings into new constituents spanning the combined string, one can either iterate over all binary productions in the grammar and test if the new derivation is valid (we call this “grammar loop”), or one can take the cross-product of active entries in the cells spanning the substrings and poll the grammar for possible derivations (we call this “cross-product”). With the cross-product approach, fewer active entries in either child cell leads to fewer grammar access operations. Thus, pruning constituents in smaller-span cells directly affects the overall efficiency of parsing. On the other hand, with the grammar loop method there is a constant number of grammar access operations (i.e., the number of grammar productions) and the number of active entries in each child cell has little impact on efficiency. Therefore, with the grammar loop implementation of the CYK algorithm, pruning techniques such as unary constraints will have very little impact on the final run-time efficiency of the parser unless the list of grammar productions is modified given the constraints. For example, alternate iterators over grammar productions could be created such that they only consider a subset of all possible productions. If the left child is a span-1 chart cell in $\overline{U}$, then only grammar productions with a POS tag as the left child need to be considered. Looping over this smaller list, opposed to all possible grammar productions, can reduce the overall runtime. The Berkeley parser contains the grammar-loop implementation of the CYK algorithm. Although all grammar productions are iterated over at each cell, the parser maintains meta-information indicating where non-terminals have been placed in the chart, allowing it to quickly skip over a subset of the productions that are incompatible with state of the chart. This optimization improves efficiency, but does not take full advantage of restricted cell population imposed by unary constraints, and thus does not benefit as greatly when compared to the BUBS or Charniak parsers.

The second trend we see in Figure 7 is that accuracy actually improves with additional constraints. This is expected with the Charniak parser as we keep the amount of search space fixed in hopes of pursuing more accurate global paths, but both the Berkeley and BUBS parsers are simply eliminating search paths they would have otherwise considered. Although it is unusual that pruning leads to higher accuracy during search, it is not wholly unexpected here as our finite-state tagger makes use of lexical relationships that the PCFG does not (i.e., lexical relationships based on the linear string rather than over the syntactic structure). By leveraging this new information to constrain the search space, we are indirectly improving the quality of the model. We also suspect that the Berkeley parser sees less of an accuracy improvement than the BUBS parsers because the coarse-to-fine pruning within the Berkeley parser is more "globally informed" than the beam-search within the BUBS parser. By leveraging the coarse-grained inside/outside distribution of trees over the input sentence, the Berkeley parser can more intelligently prune the search space with respect to the target grammar and may not benefit from the additional information inherent in the finite-state tagging model.

The third observation we point out in Figure 7 is that we see no additional gains from combining high-precision constraints with quadratic complexity constraints. With all three parsers, high-precision constraints are empirically superior to quadratic constraints, even though high-precision constraints come with no guarantee on worst-case complexity reduction. It is our hypothesis that the additional pruning provided by quadratic constraints for exhaustive CYK parsing is already removed by the internal pruning of each of the three high-accuracy parsers. We therefore report testing results using only high-precision and unary constraints for these high-accuracy parsers.

We apply models tuned on the development set to unseen English test data (WSJ Section 23) in Table 7, and Chinese test data (PCTB articles 271–300) in Table 8. For English, we see similar trends as we did on the development set results: Decoding time is nearly halved when chart constraints are applied to these already heavily constrained parsers, without any loss in accuracy. We also see independent gains from both unary and high-precision constraints, and additive efficiency gains when combined.

Applying chart constraints to Chinese parsing in Table 8 gives substantially larger accuracy and efficiency gains than English for both the BUBS and Berkeley parser. In particular, the accuracy of the BUBS parser increases by 2.3 points absolute ($p = 0.0002$),

Table 7
English test set results (WSJ Section 23) applying sentence-level high precision and unary constraints to three parsers with parameter settings tuned on development data.

Parser	F_1	Precision	Recall	Seconds	Speed-up
BUBS (2010)	88.4	88.5	88.3	586	
+ Unary(100)	88.5	88.7	88.3	486	1.2x
+ HP(0.9)	88.7	88.9	88.6	349	1.7x
+ HP(0.9) + Unary(100)	88.7	89.0	88.4	283	2.1x
Charniak (2000)	89.7	89.7	89.6	1,116	
+ Unary(100)	89.8	89.8	89.7	900	1.2x
+ HP(0.8)	89.8	90.0	89.6	716	1.6x
+ HP(0.8) + Unary(100)	89.7	90.0	89.5	679	1.6x
Berkeley (2007)	90.2	90.3	90.0	564	
+ Unary(125)	90.1	90.3	89.9	495	1.1x
+ HP(0.7)	90.2	90.4	90.0	320	1.8x
+ HP(0.7) + Unary(125)	90.2	90.4	89.9	289	2.0x

Table 8
Chinese test set results (PCTB articles 271–300) applying sentence-level high-precision and unary constraints to two parsers with parameter settings tuned on development data.

Parser	F_1	Precision	Recall	Seconds	Speed-up
BUBS (2010)	79.5	79.5	79.1	169	
+ Unary(50)	80.7	82.1	79.4	153	1.1x
+ HP(0.8)	81.1	81.5	80.7	75	2.3x
+ HP(0.8) + Unary(50)	81.8	83.1	80.5	44	3.8x
Berkeley (2007)	83.9	84.5	83.3	141	
+ Unary(50)	84.5	85.9	83.0	125	1.1x
+ HP(0.7)	84.5	85.1	83.8	64	2.2x
+ HP(0.7) + Unary(50)	84.7	86.1	83.4	57	2.5x

and the Berkeley parser increases by 0.8 points absolute to 84.7 ($p = 0.0119$), the highest accuracy we are aware of for an individual model on this data set.[10,11] These increases relative to English may be surprising as chart constraint tagging accuracy for Chinese is worse than English (see Table 3). We attribute this large gain to the lower baseline accuracy of parsing with the Chinese treebank, allowing our method to contribute additional syntactic constraints that were otherwise unmodeled by the PCFG.

9. Conclusion and Future Work

We have presented finite-state pre-processing methods to constrain context-free parsing that reduce both the worst-case complexity and overall run time. Four unique

[10] Significance was tested using stratified shuffling.

[11] Zhang et al. (2009) report an F-measure of 85.5 with a *k*-best combination of parsers, and Burkett, Blitzer, and Klein (2010) report an F-measure of 86.0 by leveraging parallel English data for training, but our model is trained from the Chinese treebank alone and is integrated into the Berkeley parser, making it very efficient.

bounding methods were presented, with high-precision constraints showing superior performance in empirical trials. Applying these constraints to context-free parsing increased efficiency by over 20 times for exhaustive CYK parsing, and nearly doubled the speed of the Charniak and Berkeley parsers—both of which have been previously tuned for optimal accuracy/efficiency performance. We have shown that our method generalizes across multiple grammars and languages, and that constraining both multi-word chart cells and single-word chart cells produces additive efficiency gains.

In future work we plan to investigate additional methods to make context-free parsing more efficient. In particular, we believe the dynamic programming chart could be constrained even further if we take into account the population and structure of chart cells at a finer level. For example, the *B* and *E* constraints we have presented here do not take into account the height of the constituent they are predicting. Instead, they simply open or close the entire diagonal in the chart. Refining this structured prediction to bins, as is done in Zhang et al. (2010), or classifying chart cells directly, as is done in Bodenstab et al. (2011), has shown promise, but we believe that further research in this area would yield addition efficiency gains. In particular, those two papers do not differentiate between non-terminals when opening or closing cells, and we hypothesize that learning separate finite-state classifiers for automatically derived clusters of non-terminals may increase performance.

Acknowledgments

Portions of this paper have appeared in conference papers: Roark and Hollingshead (2008), Roark and Hollingshead (2009), and Bodenstab, Hollingshead, and Roark (2011). We thank three anonymous reviewers for their insightful comments and suggestions. Also thanks to Aaron Dunlop for being so swell. This research was supported in part by NSF grants IIS-0447214 and IIS-0811745, and DARPA grant HR0011-09-1-0041. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF or DARPA.

References

Bangalore, Srinivas and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics*, 25:237–265.

Bergsma, Shane and Colin Cherry. 2010. Fast and accurate arc filtering for dependency parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 53–61, Beijing.

Black, E., S. Abney, S. Flickenger, C. Gdaniec, C. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. 1991. Procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 306–311, Pacific Grove, CA.

Bodenstab, Nathan, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 440–449, Portland, OR.

Bodenstab, Nathan, Kristy Hollingshead, and Brian Roark. 2011. Unary constraints for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 676–681, Portland, OR.

Burkett, David, John Blitzer, and Dan Klein. 2010. Joint parsing and alignment with weakly synchronized grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 127–135, Los Angeles, CA.

Caraballo, Sharon A. and Eugene Charniak. 1997. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24:275–298.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–139, Seattle, WA.

Charniak, Eugene and Mark Johnson. 2005. Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180, Sydney.

Cherry, Colin and Shane Bergsma. 2011. Joint training of dependency parsing filters through latent support vector machines. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 200–205, Toulouse.

Clark, Stephen and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of COLING*, pages 282–288, Geneva.

Cocke, John and Jacob T. Schwartz. 1970. Programming languages and their compilers: Preliminary notes. Courant Institute of Mathematical Sciences, New York University.

Collins, Michael. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, Philadelphia, PA.

Djordjevic, Bojan, James R. Curran, and Stephen Clark. 2007. Improving the efficiency of a wide-coverage CCG parser. In *Proceedings of the 10th International Conference on Parsing Technologies*, IWPT '07, pages 39–47, Prague.

Dreyer, Markus, David A. Smith, and Noah A. Smith. 2006. Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 201–205, New York, NY.

Dunlop, Aaron, Nathan Bodenstab, and Brian Roark. 2010. Reducing the grammar constant: an analysis of CYK parsing efficiency. Technical report CSLU-2010-02, Oregon Health & Science University, Beaverton, OR.

Dunlop, Aaron, Nathan Bodenstab, and Brian Roark. 2011. Efficient matrix-encoded grammars and low latency parallelization strategies for CYK. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT)*, pages 163–174, Dublin.

Earley, Jay. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 6(8):451–455.

Eisner, Jason and Noah A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT)*, pages 30–41, Vancouver.

Glaysher, Elliot and Dan Moldovan. 2006. Speeding up full syntactic parsing by leveraging partial parsing decisions. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 295–300, Sydney.

Hollingshead, Kristy, Seeger Fisher, and Brian Roark. 2005. Comparing and combining finite-state and context-free parsers. In *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 787–794, Vancouver.

Hollingshead, Kristy and Brian Roark. 2007. Pipeline iteration. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 952–959, Prague.

Kasami, Tadao. 1965. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.

Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19:313–330.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530, Vancouver.

Nivre, Joakim. 2006. Constraints on non-projective dependency parsing. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 73–80, Trento.

Petrov, Slav and Dan Klein. 2007a. Improved inference for unlexicalized parsing. In *Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 404–411, Rochester, NY.

Petrov, Slav and Dan Klein. 2007b. Learning and inference for hierarchically split PCFGs. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, pages 1663–1666, Vancouver.

Ratnaparkhi, Adwait. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175.

Roark, Brian and Kristy Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING)*, pages 745–752, Manchester.

Roark, Brian and Kristy Hollingshead. 2009. Linear complexity context-free parsing pipelines via chart constraints. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 647–655, Boulder, CO.

Sha, Fei and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL*, pages 134–141, Edmonton.

Sogaard, Anders and Jonas Kuhn. 2009. Using a maximum entropy-based tagger to improve a very fast vine parser. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 206–209, Paris.

Song, Xinying, Shilin Ding, and Chin-Yew Lin. 2008. Better binarization for the CKY parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 167–176, Honolulu, HI.

Weiss, David, Benjamin Sapp, and Ben Taskar. 2010. Sidestepping intractable inference with structured ensemble cascades. In *Proceedings of NIPS*, pages 2415–2423, Vancouver.

Weiss, David and Benjamin Taskar. 2010. Structured prediction cascades. *Journal of Machine Learning Research - Proceedings Track*, 9:916–923.

Xue, Naiwen, Fei Xia, Fu-dong Chiou, and Marta Palmer. 2005. The Penn Chinese treebank: Phrase structure annotation of a large corpus. *Natural Language Engingeering*, 11:207–238.

Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208.

Zhang, Hui, Min Zhang, Chew Lim Tan, and Haizhou Li. 2009. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3*, EMNLP '09, pages 1552–1560, Singapore.

Zhang, Yue, Byung Gyu Ahn, Stephen Clark, Curt Van Wyk, James R. Curran, and Laura Rimell. 2010. Chart pruning for fast lexicalised-grammar parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1472–1479, Beijing.