Favor Short Dependencies: Parsing with Soft and Hard Constraints on Dependency Length

Jason Eisner and Noah A. Smith

Abstract In lexicalized phrase-structure or dependency parses, a word's modifiers tend to fall near it in the string. This fact can be exploited by parsers. We first show that a crude way to use dependency length as a parsing feature can substantially improve parsing speed and accuracy in English and Chinese, with more mixed results on German. We then show similar improvements by imposing *hard* bounds on dependency length and (additionally) modeling the resulting sequence of parse fragments. The approach with hard bounds, "vine grammar," accepts only a regular language, even though it happily retains a context-free parameterization and defines meaningful parse trees. We show how to parse this language in $O(n)$ time, using a novel chart parsing algorithm with a low grammar constant (rather than an impractically large finite-state recognizer with an exponential grammar constant).

1 Introduction

Many modern parsers identify the head word of each constituent they find. This makes it possible to identify the word-to-word dependencies implicit in a parse.[1] Some parsers, known as dependency parsers, even return these dependencies as their primary output. Why bother to identify dependencies? The typical reason is to model the fact that some word pairs are more likely than others to engage in a

Jason Eisner
Johns Hopkins University, Baltimore, MD, USA, e-mail: jason@cs.jhu.edu

Noah A. Smith
Carnegie Mellon University, Pittsburgh, PA, USA, e-mail: nasmith@cs.cmu.edu

[1] In a phrase-structure parse, if phrase X headed by word token x is a subconstituent of phrase Y headed by word token $y \neq x$, then x is said to depend on y. In a more powerful compositional formalism like LTAG or CCG, dependencies can be extracted from the derivation tree.

dependency relationship.[2] In this paper, we propose a different reason to identify dependencies in candidate parses: to evaluate not the dependency's word pair but its *length* (i.e., the *string distance* between the two words). Dependency lengths differ from typical parsing features in that they cannot be determined from tree-local information. Though lengths are not usually considered, we will see that bilexical dynamic-programming parsing algorithms can easily consider them as they build the parse.

Soft constraints. Like any other feature of trees, dependency lengths can be explicitly used as features in a probability model that chooses among trees. Such a model will tend to disfavor long dependencies (at least of some kinds), as these are empirically rare. In the first part of the paper, we show that such features improve a simple baseline dependency parser.

Hard constraints. If the bias against long dependencies is strengthened into a hard constraint that absolutely prohibits long dependencies, then the parser turns into a partial parser with only finite-state power. In the second part of the paper, we show how to perform chart parsing in asymptotic linear time with a low grammar constant. Such a partial parser does less work than a full parser in practice, and in many cases recovers a more precise set of dependencies (with little loss in recall).

2 Short Dependencies in Langugage

We assume that correct parses exhibit a "short-dependency preference": a word's dependents tend to be close to it in the string.[3] If the j^{th} word of a sentence depends on the i^{th} word, then $|i-j|$ tends to be small. This implies that neither i nor j is modified by complex phrases that fall between i and j. In terms of phrase structure, it implies that the *phrases* modifying word i from a given side tend to be (1) few in number, (2) ordered so that the longer phrases fall farther from i, and (3) internally structured so that the bulk of each phrase falls on the side of j away from i.

These principles have been blamed for several linguistic phenomena across languages, both by traditional linguists (Hawkins, 1994) and by computational linguists (Gildea and Temperley, 2007; Temperley, 2007). For example, (1) helps explain the "late closure" or "attach low" heuristic, whereby a modifier such as a PP is more likely to attach to the closest appropriate head (Frazier, 1979; Hobbs and Bear, 1990, for example). (2) helps account for heavy-shift: when an NP is long and complex, *take NP out*, *put NP on the table*, and *give NP to Mary* are likely to be rephrased

[2] It has recently been questioned whether these "bilexical" features actually contribute much to parsing performance (Klein and Manning, 2003b; Bikel, 2004), at least when one has only a million words of training data.

[3] In this paper, we consider only a crude notion of "closeness": the number of intervening words. Other distance measures could be substituted or added (following the literature on heavy-shift and sentence comprehension), including the phonological, morphological, syntactic, or referential (given/new) complexity of the intervening material (Gibson, 1998). In parsing, the most relevant previous work is due to Collins (1997), Klein and Manning (2003c), and McDonald et al (2005a), discussed in more detail in Sect. 7.

as *take out NP*, *put on the table NP*, and *give Mary NP*. (3) explains certain non-canonical word orders: in English, a noun's left modifier must become a right modifier if and only if it is right-heavy (*a taller politician* vs. *a politician taller than all her rivals*[4]), and a verb's left modifier may extrapose its right-heavy portion (*An aardvark walked in who had circumnavigated the globe*[5]).

Why should sentences prefer short dependencies? Such sentences may be easier for humans to produce and comprehend. Each word can quickly "discharge its responsibilities," emitting or finding all its dependents soon after it is uttered or heard; then it can be dropped from working memory (Church, 1980; Gibson, 1998). Such sentences also succumb nicely to disambiguation heuristics that *assume* short dependencies, such as low attachment. Thus, to improve comprehensibility, a speaker can make stylistic choices that shorten dependencies (e.g., heavy-shift), and a language can categorically prohibit some structures that lead to long dependencies (**a taller-than-all-her-rivals politician*; **the sentence that another sentence that had center-embedding was inside was incomprehensible*).

Such functionalist pressures are not all-powerful. For example, many languages use SOV basic word order where SVO (or OVS) would give shorter dependencies. However, where the data exhibit some short-dependency preference, computer parsers as well as human parsers can obtain speed and accuracy benefits by exploiting that fact.

3 Soft Constraints on Dependency Length

We now enhance simple baseline probabilistic parsers for English, Chinese, and German so that they consider dependency lengths. We confine ourselves (throughout the paper) to parsing part-of-speech (POS) tag sequences. This allows us to ignore data sparseness, out-of-vocabulary, smoothing, and pruning issues, but it means that our accuracy measures are not state-of-the-art. Our techniques could be straightforwardly adapted to (bi)lexicalized parsers on actual word sequences, though not necessarily with the same success.

---

[4] Whereas **a politician taller* and **a taller-than-all-her-rivals politician* are not allowed. The phenomenon is pervasive. Other examples: *a sleeping baby* vs. *a baby sleeping in a crib*; *a gun-toting honcho* vs. *a honcho toting a gun*; *recently seen friends* vs. *friends seen recently*.

[5] This actually splits the heavy left dependent *[an aardvark who ...]* into two non-adjacent pieces, moving the heavy second piece. By slightly stretching the *aardvark-who* dependency in this way, it greatly shortens *aardvark-walked*. The same is possible for heavy, non-final right dependents: *I met an aardvark yesterday who had circumnavigated the globe* again stretches *aardvark-who*, which greatly shortens *met-yesterday*. These examples illustrate (3) and (2) respectively. However, the resulting non-contiguous constituents lead to non-projective parses that are beyond the scope of this paper; see Sect. 8.

3.1 Grammar Formalism

Throughout this paper we will use split bilexical grammars, or SBGs (Eisner, 2000), a notationally simpler variant of split head-automaton grammars, or SHAGs (Eisner and Satta, 1999). The formalism is context-free and only allows projective parses (those that are free of crossing dependencies). We define here a probabilistic version,[6] which we use for the baseline models in our experiments. They are only baselines because the SBG generative process does *not* take note of dependency length.

An SBG is an tuple $\mathcal{G} = (\Sigma, \$, L, R)$. Σ is an alphabet of words. (In our experiments, we parse only POS tag sequences, so Σ is actually an alphabet of tags.) $\$ \notin \Sigma$ is a distinguished root symbol; let $\bar{\Sigma} = \Sigma \cup \{\$\}$. L and R are functions from $\bar{\Sigma}$ to probabilistic ε-free finite-state automata over Σ. Thus, for each $w \in \bar{\Sigma}$, the SBG specifies "left" and "right" probabilistic FSAs, L_w and R_w.

We use $\mathcal{L}_w(\mathcal{G}) : \bar{\Sigma}^* \to [0,1]$ to denote the probabilistic context-free language of phrases headed by w. $\mathcal{L}_w(\mathcal{G})$ is defined by the following simple top-down stochastic process for sampling from it:

1. Sample from the finite-state language $\mathcal{L}(L_w)$ a sequence $\lambda = w_{-1}w_{-2}\ldots w_{-\ell} \in \Sigma^*$ of left children, and from $\mathcal{L}(R_w)$ a sequence $\rho = w_1 w_2 \ldots w_r \in \Sigma^*$ of right children. Each sequence is found by a random walk on its probabilistic FSA. We say the children *depend* on w.
2. For each i from $-\ell$ to r with $i \neq 0$, recursively sample $\alpha_i \in \Sigma^*$ from the context-free language $\mathcal{L}_{w_i}(\mathcal{G})$. It is this step that indirectly determines dependency lengths.
3. Return $\alpha_{-\ell} \ldots \alpha_{-2}\alpha_{-1} w \alpha_1 \alpha_2 \ldots \alpha_r \in \bar{\Sigma}^*$, a concatenation of strings.

Notice that w's left children λ were generated in reverse order, so w_{-1} and w_1 are its closest children while $w_{-\ell}$ and w_r are the farthest.

Given an input sentence $\omega = w_1 w_2 \ldots w_n \in \Sigma^*$, a parser attempts to recover the highest-probability derivation by which $\$\omega$ could have been generated from $\mathcal{L}_{\$}(\mathcal{G})$. Thus, $\$ plays the role of w_0. A sample derivation is shown in Fig. 1a. Typically, $L_{\$}$ and $R_{\$}$ are defined so that $\$ must have no left children ($\ell = 0$) and at most one right child ($r \leq 1$), the latter serving as the conventional root of the parse.

3.2 Baseline Models

In the experiments reported here, we defined only very simple automata for L_w and R_w ($w \in \Sigma$). However, we tried three automaton types, of varying quality, so as to evaluate the benefit of adding length-sensitivity at three different levels of baseline performance.

[6] There is a straightforward generalization to *weighted* SBGs, which need not have a stochastic generative model.

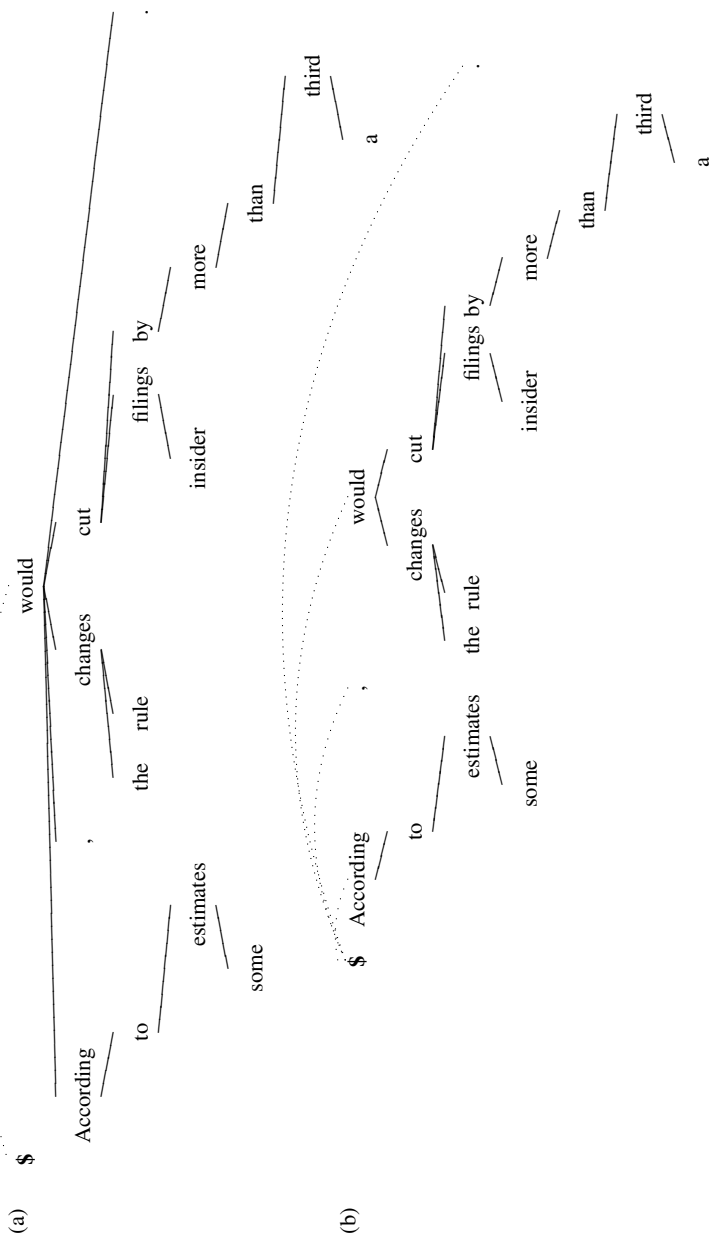


Fig. 1: (a) A dependency tree on words. (Our experiments use only POS tags.) (b) A partial parse for the same sentence retaining only tree dependencies of length $\leq k = 3$. The roots of the 4 resulting parse fragments are now connected only by their dotted-line "vine dependencies" on \$. Transforming (a) into (b) involves grafting subtrees rooted at "*According*", ",", and "." onto the vine.

In model A (the worst), each automaton has topology $\circledcirc\!\!\leftarrow\!\!\supset$, with a single state q_1, so token w's left dependents are conditionally independent of one another given w. In model C (the best), each automaton $\circledcirc\!\!\longrightarrow\!\!\circledcirc\!\!\leftarrow\!\!\supset$ has an extra state q_0 that allows the first (closest) dependent to be chosen differently from the rest. Model B is a compromise:[7] it is like model A, but each type $w \in \Sigma$ may have an elevated or reduced probability of having no dependents at all. This is accomplished by using automata $\circledcirc\!\!\longrightarrow\!\!\circledcirc\!\!\leftarrow\!\!\supset$ as in model C, which allows the stopping probabilities $p(\text{STOP} \mid q_0)$ and $p(\text{STOP} \mid q_1)$ to differ, but tying the conditional distributions $p(q_0 \xrightarrow{w} q_1 \mid q_0, \neg\text{STOP})$ and $p(q_1 \xrightarrow{w} q_1 \mid q_1, \neg\text{STOP})$.

Finally, throughout Sect. 3, $L_{\$}$ and $R_{\$}$ are restricted as above, so $R_{\$}$ gives a probability distribution over Σ only.

3.3 Length-Sensitive Models

None of the baseline models A–C *explicitly* model the distance between a head and child. We enhanced them by multiplying in some extra length-sensitive factors when computing a tree's probability. For each dependency, an extra factor $p(\Delta \mid \ldots)$ is multiplied in for the probability of the dependency's length $\Delta = |i - j|$, where i and j are the positions of the head and child in the *surface* string. In practice, this especially penalizes trees with long dependencies, because large values of Δ are empirically unlikely.

Note that this is a crude procedure. Each legal tree—whether its dependencies are long or short—has had its probability reduced by some extra factors $p(\Delta \mid \ldots) \leq 1$. Thus, the resulting model is *deficient* (does not sum to 1). (The remaining probability mass goes to impossible trees whose putative dependency lengths Δ are inconsistent with the tree structure.) One could develop non-deficient models (either log-linear or generative), but we will see that even the present crude approach helps.

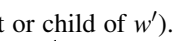
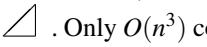
Again we tried three variants. In one version, this new probability $p(\Delta \mid \ldots)$ is conditioned only on the direction $d = \text{sign}(i - j)$ of the dependency. In another version, it is conditioned only on the POS tag h of the head. In a third version, it is conditioned on d, h, and the POS tag c of the child.

3.4 Parsing Algorithm

Fig. 2 gives a variant of Eisner and Satta's (1999) SHAG parsing algorithm, adapted to SBGs, which are easier to understand.[8] (We will modify this algorithm later in Sect. 5.) The algorithm obtains $O(n^3)$ runtime, despite the need to track the position of head words, by exploiting the conditional independence between a head's

[7] It is equivalent to the "dependency model with valence" of Klein and Manning (2004).

[8] The SHAG notation was designed to highlight the connection to *non*-split HAGs.

left children and its right children (given the head). It builds "half-constituents" denoted by  (a head word together with some modifying phrases on the right, i.e., $w\alpha_1 \dots \alpha_r$) and  (a head word together with some modifying phrases on the left, i.e., $\alpha_{-\ell} \dots \alpha_{-1} w$). A new dependency is introduced when  are combined to get the "trapezoid"  or  (a pair of linked head words with all the intervening phrases, i.e., $w\alpha_1 \dots \alpha_r \alpha'_{-\ell'} \dots \alpha'_{-1} w'$, where w is respectively the parent or child of w'). One can then combine  , or  . Only $O(n^3)$ combinations are possible in total when parsing a length-n sentence.

3.5 A Note on Word Senses

A remark is necessary about $:w$ and $:w'$ in Fig. 2, which represent *senses* of the words at positions h and h'. Like past algorithms for SBGs (Eisner, 2000), Fig. 2 is designed to be a bit more general and integrate sense disambiguation into parsing.[9] It formally runs on an input $\Omega = W_1 \dots W_n \subseteq \Sigma^*$, where each $W_i \subseteq \Sigma$ is a "confusion set" over possible values of the i^{th} word w_i. Thus Ω is a "confusion network." The algorithm recovers the highest-probability derivation that generates $\$\omega$ for *some* $\omega \in \Omega$ (i.e., $\omega = w_1 \dots w_n$ with $(\forall i) w_i \in W_i$).

[9] In the present paper, we adopt the simpler and slightly more flexible SBG formalism of Eisner (2000), which allows explicit word senses, but follow the asymptotically more efficient SHAG parsing algorithm of Eisner and Satta (1999), in order to save a factor of g in our runtimes. Thus Fig. 2 presents a version of the Eisner-Satta SHAG algorithm that has been converted to work with SBGs, exactly as sketched and motivated in footnote 6 of Eisner and Satta (1999).

This conversion preserves the *asymptotic* runtime of the Eisner-Satta algorithm. However, notice that the version in Fig. 2 does have a *practical* inefficiency, in that START-LEFT nondeterministically guesses each possible sense $w \in W_h$, and these g senses are pursued separately. This inefficiency can be repaired as follows. We should not need to commit to one of a word's g senses until we have seen all its left children (in order to match the behavior of the Eisner-Satta algorithm, which arrives at one of g "flip states" in the word's FSA only by accepting a sequence of children). Thus, the left triangles and left trapezoids of Fig. 2 should be simplified so that they do not carry a sense $:w$ at all, except in the case of the completed left triangle (marked F) that is produced by FINISH-LEFT. The FINISH-LEFT rule should nondeterministically choose a sense w of W_h according to the final state q, which reflects knowledge of W_h's sequence of left children.

For this strategy to work, the transitions in L_w (used by ATTACH-LEFT) clearly may not depend on the particular sense $w \in W_h$ but only on W_h. In other words, all $L_w : w \in W_h$ are really copies of a shared L_{W_h}, except that they may have different final states. This slightly inelegant restriction on the SBG involves no loss of generality, since the nondeterministic shared L_{W_h} is free to branch as soon as it likes onto paths that commit to the various senses w.

We remark without details that this modification to Fig. 2, which defers the choice of w for as long as possible, could be obtained mechanically as an instance of the speculation transformation of Eisner and Blatz (2007). Speculation could similarly be used to extend the trick to the lattice parsing of Sect. 3.7, where a left triangle would commit immediately to the initial state of its head arc but defer committing to the full head arc for as long as possible.

This extra level of generality is not needed for any of our experiments, but without it, SBG parsers would not be as flexible as SHAG parsers. We include it in this paper to broaden the applicability of both Fig. 2 and our extension of it in Sect. 5.

The "senses" can be used in an SBG to pass a finite amount of information between the left and right children of a word (Eisner, 2000). For example, to model the fronting of a direct object, an SBG might use a special sense of a verb, whose automata tend to generate both one more noun in the left child sequence λ and one fewer noun in the right child sequence ρ.

Senses can also be used to pass information between parents and children. Important uses are to encode lexical senses, or to enrich the dependency parse with constituent labels, dependency labels, or supertags (Bangalore and Joshi, 1999; Eisner, 2000). For example, the input token $W_i = \{bank_1/N/NP,\ bank_2/N/NP,\ bank_3/V/VP,\ bank_3/V/S\} \subset \Sigma$ allows four "senses" of bank, namely two nominal meanings, and two *syntactically different* versions of the verbal meaning, whose automata require them to expand into VP and S phrases respectively.

The cubic runtime is proportional to the number of ways of instantiating the inference rules in Fig. 2: $O(n^2(n+t')tg^2)$, where $n = |\Omega|$ is the input length, $g = \max_{i=1}^{n} |W_i|$ bounds the size of a confusion set, t bounds the number of states per automaton, and $t' \leq t$ bounds the number of automaton transitions from a state that emit the same word. For deterministic automata, $t' = 1$.

3.6 Probabilistic Parsing

It is easy to make the algorithm of Fig. 2 length-sensitive. When a new dependency is added by an ATTACH rule that combines  +  , the annotations on and suffice to determine the dependency's length $\Delta = |h - h'|$, direction $d = \text{sign}(h - h')$, head word w, and child word w'.

So the additional cost of such a dependency, e.g. $p(\Delta \mid d, w, w')$, can be included as the weight of an extra antecedent to the rule, and so included in the weight of the resulting  or  .

To execute the inference rules in Fig. 2, common practice is to use a prioritized agenda (Eisner et al, 2005), at the price of an additional logarithmic factor in the runtime for maintaining the priority queue. In our experiments, derived items such as  ,  ,  , and  are prioritized by their Viterbi-inside probabilities. This is known as *uniform-cost search* or *shortest-hyperpath search* (Nederhof, 2003). We halt as soon as a full parse (the special *accept* item) pops from the agenda, since uniform-cost search (as a special case of the A* algorithm) guarantees this to be the maximum-probability parse. No other pruning is done.

With a prioritized agenda, a probability model that more sharply discriminates among parses will typically lead to a faster parser. (Low-probability constituents

languish at the back of the agenda and are never pursued.) We will see that the length-sensitive models do run faster for this reason.[10]

3.7 A Note on Lattice Parsing

A lattice is an acyclic FSA. Often parsing a weighted lattice is useful—for example, the output of a speech recognition or machine translation system. The parser extracts a good path through the lattice (i.e., one that explains the acoustic or source-language data) that also admits a good syntax tree (i.e., its string is likely under a generative syntactic language model, given by Sect. 3.1).

More generally, we may wish to parse an *arbitrary* FSA. For example, if we apply our inference rules using the inside semiring (Goodman, 1999), we obtain the total weight of all parses of all paths in the FSA. This provides a normalizing constant that is useful in learning, if the FSA is Σ^* or a "neighborhood" (contrast set) of distorted variants of the observed string (Smith and Eisner, 2005; Smith, 2006).

Thus, for completeness, we now present algorithms for the case where we are given an arbitrary FSA as input. A parse now consists of a choice of path through the given FSA together with an SBG dependency tree on the string accepted by that path. In the weighted case, the weight of the parse is the product of the respective FSA and SBG weights.

Sect. 3.5 already described a special case—the confusion network $\Omega = W_1 \ldots W_n$, which may be regarded as a particular unweighted lattice with n states and ng arcs. The confusion-network parsing algorithm of Fig. 2 can easily be generalized to parse an arbitrary weighted FSA, Ω:

- In general, the derivation tree of a triangle or trapezoid item now explains a path in Ω. The lower left corner of the item specifies the leftmost state or arc on that path, while the lower right corner specifies the rightmost state or arc. If the lower left corner specifies an arc, the weight of this leftmost arc is not included in the weight of the derivation tree (it will be added in by a later COMPLETE step).
- Each position h in Fig. 2 that is paired with a word w (i.e., $h:w$) now denotes an arc in Ω that is labeled with w. Similarly for $h':w'$.
- The special position 0, which is paired with the non-word **$**, denotes the initial state of Ω (rather than an arc).
- Each unpaired position i now denotes a state in Ω. In the ATTACH rules, $i-1$ should be modified to equal i. In END-VINE, the unpaired position n should be constrained by a new antecedent to be a final state of Ω, whose stopping weight is the weight of this antecedent.
- In the START-LEFT (respectively START-RIGHT) rule, where h in $h:w$ now denotes an arc, the unpaired h should be replaced by the start (respectively end) state of this arc.

[10] A better priority function that estimated outside costs would further improve performance (Caraballo and Charniak, 1998; Charniak et al, 1998; Klein and Manning, 2003a).

- In the START-LEFT rule, the antecedent $w \in W_h$ is replaced by a new antecedent requiring that h is an arc of Ω, labeled with w. The weight of this arc is the weight of the new antecedent.
- We add the following rule to handle arcs of Ω that are labeled with ε rather than with a word: TRAVERSE-ε: $\dfrac{\begin{array}{c} q \\ \diagdown \\ h\!:\!w \quad i \end{array} \qquad (i \xrightarrow{\varepsilon} j) \in \Omega}{\begin{array}{c} q \\ \diagdown \\ h\!:\!w \quad j \end{array}}$
- If Ω is cyclic, it is possible to obtain cyclic derivations (analogous to unary rule cycles in CFG parsing) in which an item is used to help derive itself. However, this is essentially unproblematic if this cyclic derivation always has worse probability than the original acyclic one, and hence does not improve the weight of the item (Goodman, 1999).

The resulting algorithm has runtime $O(m^2(n+t')t)$ for a lattice of n states and m arcs. In the confusion network case, where $m = ng$, this reduces to our earlier runtime of $O(n^2(n+t')tg^2)$ from Sect. 3.5.

The situation becomes somewhat trickier, however, when we wish to consider dependency lengths. For our soft constraints in Sect. 3.6, we needed to determine the length Δ of a new dependency that is added by an ATTACH rule. Unfortunately the distance $\Delta = |h - h'|$ is no longer well-defined now that h and h' denote arcs in an FSA rather than integer positions in a sentence. Different paths from h to h' might cover different numbers of words.

Before proceeding, let us generalize the notion of dependency length. Assume that each arc in the input FSA, Ω, comes equipped with a length. Recall that a parse specifies a finite path $h_1 h_2 \ldots h_n$ through Ω and a set of dependencies among the word tokens that label that path. If there is a leftward or rightward dependency between the word tokens that label arcs h_ℓ and h_r, where $\ell < r$, we define the length of this dependency to be the total length of the subpath $h_{\ell+1} h_{\ell+2} \ldots h_r$.[11]

When an ATTACH rule builds a trapezoid item, it adds a dependency. Our goal is to determine the length Δ of that dependency from the "width" of the trapezoid, so that the ATTACH rule can multiply in the appropriate penalty. The problem is that the width of an *item* is not well-defined: rather, each *derivation* (proof tree) of a triangle or trapezoid item has a possibly different width.

We define a derivation's width to be the total length of the subpath of Ω that is covered by the derivation, but excluding the leftmost arc of the subpath iff the item itself specifies that arc.[12] In other words, let i denote the item's leftmost state or

[11] It is an arbitrary decision for a dependency's length to include the length of its right word but not the length of its left word. We adopt that convention only for consistency with our earlier definition of dependency length, and to simplify the relationship between dependency length and derivation width. It might however be justified in terms of incremental parsing, since it encodes the wait time once the left word has been heard until the right word is fully available to link to it.

[12] This exclusion ensures that when we combine two such derivations using COMPLETE or ATTACH, then the consequent derivation's width is always the sum of its antecedent derivations'

the *end* state of its leftmost arc if specified, and j denote its rightmost state or the *start* state of its rightmost arc if specified. The derivation's width is the length of the subpath from i to j, plus the length of the rightmost arc if specified. Unfortunately, the item does not record this subpath, which differs by derivation; it only records i and j.

There are several possible attacks on the problem of defining the widths of triangle and trapezoid items:

Redefine length. One option is to ensure that all derivations of an item do have equal widths. This may be done by defining the arc lengths in Ω in a "consistent" way. Where Ω is an acoustic lattice derived from a speech signal, we can meaningfully associate a time $t(i)$ with each state $i \in \Omega$, and define the length of an arc to be the difference between its start and end times. Then *all* paths from state i to state j have the same length, namely $t(j) - t(i)$. In short, the problem goes away if we measure dependency length in acoustic milliseconds.

However, Ω is not necessarily an acoustic lattice. To measure a dependency's length by its string distance in words, as we have been doing thus far, we must define each arc's length to be be 1 or 0 according to whether it accepts a word or ε.[13] In this case, different paths from i to j do have different lengths.

Specialize the items. A second option is to augment each triangle or trapezoid item with a specific width Δ. In other words, we split an item that already specifies i and j into several more specific items, each of which allows only derivations of a particular width. The width of a consequent item can be determined easily by summing the widths of its antecedents.

Unfortunately, this exact method leads to more items and increased runtime (though only to the extent that there really are paths of many different lengths between i and j). In particular, it leads to infinitely many items if the input FSA Ω is cyclic.[14]

Use a lower bound based on shortest path. A third option is to *approximate* by using only a lower bound on dependency length. The ATTACH rule can consider the width of a trapezoid to be a lower bound on the widths of its derivations, specifically, the *shortest* path in Ω from i to j.[15] In other words, when evaluating a parse, we will define a dependency between arcs h_ℓ and h_r to be be "short" if these arcs are close on some path, though not necessarily on the path actually chosen by the parse.

Notice that one can improve this lower bound at the expense of greater runtime, by modifying Ω to use more states and less structure-sharing. A reasonable trick is

---

widths. Recall from the first bullet point above that the same exclusion was used when defining the *weight* of an item, and for the same reason.

[13] One could change the arc lengths to measure not in words but in one of the other measurement units from footnote 3.

[14] Although uniform-cost search will still terminate, provided that all cycles in Ω have positive cost. All sufficiently wide items will then have a cost worse than that of the best parse, so only finitely many items will pop from the priority queue.

[15] The shortest-path distances between all state pairs can be precomputed in $O(n^3 + m)$ time using the Floyd-Warshall algorithm. This preprocessing time is asymptotically dominated by the runtime of Fig. 2.

to intersect Ω with an FSA that accepts Σ^* and has the topology of a simple 4-cycle. This does not change the weighted language accepted by Ω, but it splits states of Ω so that two paths from i to j must accept the same number of words, modulo 4. For many practical lattices, this will often ensure that all short paths from i to j accept exactly the same number of words. It increases n by a factor of ≤ 4 and does not increase m at all.[16]

Partially specialize the items. By combining the previous two methods, we can keep the number of items finite. Fix a constant $k \geq 0$. Each item either records a specific width $\Delta \in [0,k]$, or else records that it has width $> k$. The former items conside only derivations of a particular width, while for the latter items we can use a lower bound.

Coarse to fine. There is another way to combine these methods. The lower-bounding method can be run as a "coarse pass," followed by the exact specialized-items method as a "fine pass." If shorter dependencies are always more likely than longer ones, then the Viterbi outside probabilities from the coarse pass are upper bounds on the Viterbi outside probabilities from the fine pass, and hence can be used as an admissible A* heuristic to prioritize derivations on the fine pass.

Aggregate over derivations. A final option is to approximate more tightly. Each triangle and each trapezoid can dynamically maintain an estimate $\bar{\Delta}$ of the *minimum* (or the *expected*) width of its derivations. Whenever an inference rule derives or rederives the item, it can update this $\bar{\Delta}$ estimate based on the current estimates $\bar{\Delta}$ at its antecedents.[17] When deriving a trapezoid, the ATTACH rule can estimate the dependency length that it needs as the total current $\bar{\Delta}$ of its antecedents.[18]

This may give a lower bound that is tighter than the one given earlier, since it attempts to use the shortest i-to-j subpath that is covered by some actual derivation of the item in question, rather than the shortest i-to-j subpath overall. Unfortunately, if we wish to ensure that it is a true lower bound (i.e., considers *all* relevant i-to-j subpaths), then we must incur the extra overhead of updating it when new derivations are found. Specifically, we must consider reducing the $\bar{\Delta}$ of an item whenever the $\bar{\Delta}$ of one of its antecedents is reduced. Since a trapezoid's $\bar{\Delta}$ in turn affects its weight, this may in turn force us to propagate increases or other updates to item weights.

[16] A related trick is to convert Ω to a trie (if it is acyclic). This makes the lower bound exact by ensuring that there are never multiple paths from i to j, but potentially increases the size of Ω exponentially.

[17] For the expected-width case, each item must maintain both $\sum_d p(d)\Delta(d)$ and $\sum_d p(d)$, where d ranges over derivations. These quantities can be updated easily, and their ratio is the expected width.

[18] This is more precise than using the $\bar{\Delta}$ of the consequent, which is muddied by other derivations that are irrelevant to this dependency length.

4 Experiments with Soft Constraints

We trained models A–C, using unsmoothed maximum likelihood estimation, on three treebanks: the Penn (English) Treebank (split in the standard way, §2–21 train/§23 test, or 950K/57K words), the Penn Chinese Treebank (80% train/10% test or 508K/55K words), and the German TIGER corpus (80%/10% or 539K/68K words).[19] Estimation was a simple matter of counting automaton events and normalizing counts into probabilities. For each model, we also trained the three length-sensitive versions described in Sect. 3.3.

The German corpus contains some non-projective trees, whose dependencies cross. None of our parsers can recover these non-projective dependencies, nor can our models produce them (but see Sect. 8). This fact was ignored when counting events for maximum likelihood estimation: in particular, we always trained L_w and R_w on the sequence of w's immediate children, even in non-projective trees.

Our results (Table 1) show that sharpening the probabilities with the most sophisticated distance factors $p(\Delta \mid d,h,c)$, consistently improved the *speed* of all parsers.[20] The change to the code is trivial. The only overhead is the cost of looking up and multiplying in the extra distance factors.

Accuracy also improved over the baseline models of English and Chinese, as well as the simpler baseline models of German. Again, the most sophisticated distance factors helped most, but even the simplest distance factor usually obtained most of the accuracy benefit.

German model C fell slightly in accuracy. The speedup here suggests that the probabilities were sharpened, but often in favor of the wrong parses. We did not analyze the errors on German; it may be relevant that 25% of the German sentences contained a non-projective dependency between non-punctuation tokens.

Studying the parser output for English, we found that the length-sensitive models preferred closer attachments, with 19.7% of tags having a nearer parent in the best parse under model C with $p(\Delta \mid d,h,c)$ than in the original model C, 77.7% having a parent at the same distance, and only 2.5% having a farther parent. The surviving long dependencies (at any length > 1) tended to be much more accurate, while the (now more numerous) length-1 dependencies were slightly less accurate than before.

We caution, however, that the length-sensitive models improved accuracy only in the aggregate. They corrected many erroneous attachments, but also introduced

[19] Heads were extracted for English using Michael Collins' rules and for Chinese using Fei Xia's rules (defaulting in both cases to right-most heads where the rules fail). German heads were extracted using the TIGER Java API; we discarded all resulting dependency structures that were cyclic or unconnected (6%).

[20] In all cases, we measure runtime abstractly by the number of items built and pushed on the agenda, where multiple ways of building the same item are counted multiple times. The items in question are  ,  ,  ,  , and in the case of Fig. 4, also  and  .) Note that if the agenda is a general priority queue, then popping an item takes logarithmic time, although pushing an item can be achieved in constant time using a Fibonacci-heap implementation.

	English (Penn Treebank)				Chinese (Chinese Treebank)				German (TIGER Corpus)			
	recall (%)		runtime	model	recall (%)		runtime	model	recall (%)		runtime	model
model	train	test	test	size	train	test	test	size	train	test	test	size
A (1 state)	62.0	62.2	93.6	1,878	50.7	49.3	146.7	782	70.9	72.0	53.4	1,598
+ $p(\Delta \mid d)$	70.1	70.6	97.0	2,032	59.0	58.0	161.9	1,037	72.3	73.0	53.2	1,763
+ $p(\Delta \mid h)$	70.5	71.0	94.7	3,091	60.5	59.1	148.3	1,759	73.1	74.0	48.3	2,575
+ $p(\Delta \mid d,h,c)$	72.8	73.1	70.4	16,305	62.2	60.6	106.7	7,828	75.0	75.1	31.6	12,325
B (2 states, tied arcs)	69.7	70.4	93.5	2,106	56.7	56.2	151.4	928	73.7	75.1	52.9	1,845
+ $p(\Delta \mid d)$	72.6	73.2	95.3	2,260	60.2	59.5	156.9	1,183	72.9	73.9	52.6	2,010
+ $p(\Delta \mid h)$	73.1	73.7	92.1	3,319	61.6	60.7	144.2	1,905	74.1	75.3	47.6	2,822
+ $p(\Delta \mid d,h,c)$	75.3	75.6	67.7	16,533	62.9	61.6	104.0	7,974	75.2	75.5	31.5	12,572
C (2 states)	72.7	73.1	90.3	3,233	61.8	61.0	148.3	1,314	75.6	76.9	48.5	2,638
+ $p(\Delta \mid d)$	73.9	74.5	91.7	3,387	61.5	60.6	154.7	1,569	74.3	75.0	48.9	2,803
+ $p(\Delta \mid h)$	74.3	75.0	88.6	4,446	63.1	61.9	141.9	2,291	75.2	76.3	44.3	3,615
+ $p(\Delta \mid d,h,c)$	75.3	75.5	66.6	17,660	63.4	61.8	103.4	8,360	75.1	75.2	31.0	13,365

Table 1: Dependency parsing of POS tag sequences with simple probabilistic split bilexical grammars. The models differ only in how they weight the same candidate parse trees. Length-sensitive models are larger but can improve dependency accuracy and speed. (**Recall** is measured as the fraction of non-punctuation tags whose correct parent (if not the **$** symbol) was recovered by the parser; it equals precision, unless the parser left some sentences unparsed (or incompletely parsed, as in Sect. 5), in which case precision is higher. **Runtime** is measured abstractly as the average number of items built (see footnote 20). **Model size** is measured as the number of nonzero parameters.)

new errors. We also caution that length sensitivity's most dramatic improvements to accuracy were on the worse baseline models, which had more room to improve. The better baseline models (B and C) were already able to indirectly capture some preference for short dependencies, by learning that some parts of speech were unlikely to have multiple left or multiple right dependents. Enhancing B and C therefore contributed less, and indeed may have had some harmful effect by over-penalizing some structures that were already appropriately penalized.[21] It remains to be seen, therefore, whether distance features would help state-of-the art parsers that are already much better than model C. Such parsers may already incorporate features that indirectly impose a good model of distance (see Sect. 7), though perhaps not as cheaply.

5 Hard Dependency-Length Constraints

We have seen how an explicit model of distance can improve the speed and accuracy of a simple probabilistic dependency parser. Another way to capitalize on the fact

[21] Owing to our deficient model. A log-linear or discriminative model would be trained to correct for overlapping penalties and would avoid this risk. Non-deficient generative models are also possible to design, along lines similar to footnote 22.

that most dependencies are local is to impose a *hard constraint* that simply forbids long dependencies.

The dependency trees that satisfy this constraint yield a regular string language.[22] The constraint prevents arbitrarily deep center-embedding, as well as arbitrarily many direct dependents on a given head, either of which would allow the non-regular language $\{a^n bc^n : 0 < n < \infty\}$. However, it *does* allow arbitrarily deep right- or left-branching structures.

5.1 Vine Grammars

The tighter the bound on dependency length, the fewer parse trees we allow and the faster we can find them using an algorithm similar to Fig. 2 (as we will see). If the bound is too tight to allow the correct parse of some sentence, we would still like to allow an accurate partial parse: a sequence of accurate parse fragments (Hindle, 1990; Abney, 1991; Appelt et al, 1993; Chen, 1995; Grefenstette, 1996). Furthermore, we would like to use the fact that some fragment sequences are presumably more likely than others.

Our partial parses will look like the one in Fig. 1b. where four subtrees rather than just one are dependent on \$. This is easy to arrange in the SBG formalism. We merely need to construct our SBG so that the automaton $R_{\$}$ is now permitted to generate multiple children—the roots of parse fragments.

This $R_{\$}$ is a probabilistic finite-state automaton that describes legal or likely root sequences in Σ^*. In our experiments in this section, we will train it to be a first-order (bigram) Markov model. (Thus we construct $R_{\$}$ in the usual way to have $|\Sigma|+1$ states, and train it on data like the other left and right automata. During generation, its state remembers the previously generated root, if any. Recall that we are working with POS tag sequences, so the roots, like all other words, are tags in Σ.)

The 4 subtrees in Fig. 1b appear as so many bunches of grapes hanging off a vine. We refer to the dotted dependencies upon \$ as *vine dependencies*, and the remaining, bilexical dependencies as *tree dependencies*.

One might informally use the term "vine grammar" (VG) for any generative formalism, intended for partial parsing, in which a parse is a constrained sequence of trees that cover the sentence. In general, a VG might use a two-part generative process: first generate a finite-state sequence of roots, then expand the roots according to some more powerful formalism. Conveniently, however, SBGs and other dependency grammars can integrate these two steps into a single formalism.

[22] One proof is to construct a strongly equivalent CFG without center-embedding (Nederhof, 2000). Each nonterminal has the form $\langle w, q, i, j\rangle$, where $w \in \Sigma$, q is a state of L_w or R_w, and $i, j \in \{0, 1, \ldots k-1, \geq k\}$. We leave the details as an exercise.

5.2 Feasible Parsing

Now, for both speed and accuracy, we will restrict the trees that may hang from the vine. We define a *feasible* parse under our SBG to be one in which all *tree* dependencies are short, i.e., their length never exceeds some hard bound k. The vine dependencies may have unbounded length, of course, as in Fig. 1b.

Sentences with feasible parses form a regular language. This would also be true under other definitions of feasibility: e.g., we could have limited the depth or width of each tree on the vine. However, that would have ruled out deeply right-branching trees, which are very common in language, and are also the traditional way to describe finite-state sublanguages within a context-free grammar. By contrast, our limitation on dependency length ensures regularity while still allowing (for any bound $k \geq 1$) arbitrarily wide and deep trees, such as $a \rightarrow b \rightarrow \ldots \rightarrow \mathit{root} \leftarrow \ldots \leftarrow y \leftarrow z$.

Our goal is to find the *best feasible* parse (if any). (In our scenario, one will typically exist—at worse, just a vine of tiny single-word trees.) Rather than transform the grammar as in footnote 22, our strategy is to modify the parser so that it only considers feasible parses. The interesting problem is to achieve linear-time parsing with a grammar constant that is as small as for ordinary parsing.

One could enforce this restriction by modifying either the grammar $\mathcal{G}$ or the parser. In this paper, we leave the grammar alone, but restrict the parser so that it is only permitted to find *short* within-tree dependencies. Other parses may be permitted by the vine grammar but are not found by our parser.

We also correspondingly modify the training data so that we only train on feasible parses. That is, we break any long dependencies and thereby fragment each training parse (a single tree) into a vine of one or more restricted trees. When we break a child-to-parent dependency, we reattach the child to \$.[23] This process, *grafting*, is illustrated in Fig. 1. Although this new parse may score less than 100% recall of the original dependencies, it is the best feasible parse, so we would like to train the parser to find it.[24] By training on the modified data, we learn more appropriate statistics for both $R_{\$}$ and the other automata. If we trained on the original trees, we would inaptly learn that $R_{\$}$ always generates a single root rather than a certain kind of sequence of roots.

For evaluation, we score tree dependencies in our feasible parses against the tree dependencies in the *unmodified* gold standard parses, which are not necessarily feasible. We also show oracle performance.

---

[23] Any dependency *covering* the child must also be broken to preserve projectivity. This case arises later; see footnote 34.

[24] Although our projective parser will still not be able to find it if it is non-projective (possible in German). Arguably we should have defined a more aggressive grafting procedure that produced projective parses, but we did not. See Sect. 8 for discussion of non-projective vine grammar parsing, which would always be able to recover the best feasible parse.

5.2.1 Approach #1: FSA Parsing

Since we are now dealing with a regular or rational string language, it is possible in principle to construct a weighted finite-state automaton (FSA) and use it to search for the best feasible parse. The idea is to find the highest-weighted path that accepts the input string $\omega = w_1 w_2 \ldots w_n$. Using the Viterbi algorithm, this takes time $O(n)$.

The trouble is that this linear runtime hides a constant factor, which depends on the size of the relevant part of the FSA and may be enormous for any correct FSA.[25] Consider an example from Fig 1b. After nondeterministically reading $w_1 \ldots w_{11} =$ *According... insider* along the *correct* path, the FSA state must record (at least) that *insider* has no parent yet and that $R_{\$}$, R_{would}, and R_{cut} are in particular states that may still accept more children. Else the FSA cannot know whether to accept the continuation $w_{12} \ldots w_n =$ *filings by more than a third* .

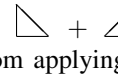
In general, after parsing a prefix $w_1 \ldots w_j$, the FSA state must somehow record information about all incompletely linked words in the past. It must record the sequence of past words w_i ($i \leq j$) that still need a parent or child in the future; if w_i still needs a child, it must also record the state of R_{w_i}.

Our restriction to dependency length $\leq k$ is what allows us to build a weighted *finite*-state automaton (as opposed to some kind of pushdown automaton with an unbounded number of configurations). We need only build the *finitely* many states in which the incompletely linked words are limited to at most $w_0 = \$$ and the k most recent words, $w_{j-k+1} \ldots w_j$. Other states cannot extend into a feasible parse, and can be pruned.

However, this still allows the FSA to be in $O(2^k t^{k+1})$ different states after nondeterministically reading $w_1 \ldots w_j$. Then the runtime of the Viterbi algorithm, though linear in n, is exponential in k.

5.2.2 Approach #2: Ordinary Chart Parsing

A much better idea for most purposes is to use a chart parser. This allows the usual dynamic programming techniques for reusing computation. (The FSA in the previous section failed to exploit many such opportunities: exponentially many states would have proceeded redundantly by building the same $w_{j+1} w_{j+2} w_{j+3}$ constituent.)

It is simple to restrict our algorithm of Fig. 2 to find only feasible parses. It is the ATTACH rules  that add dependencies: simply use a side condition to block them from applying unless $|h - h'| \leq k$ (short tree dependency) or $h = 0$ (vine dependency). This ensures that all  and will have width $\leq k$ or have their left edge at 0. One might now incorrectly expect runtime linear in n. Unfortunately, the number of possible ATTACH combinations, which add new dependencies, is reduced from $O(n^3)$ to $O(nk^2)$, because i and h' are now restricted

[25] The full runtime is $O(nE)$, where E is the number of FSA edges, or for a tighter estimate, the number of FSA edges that can be traversed by reading ω.

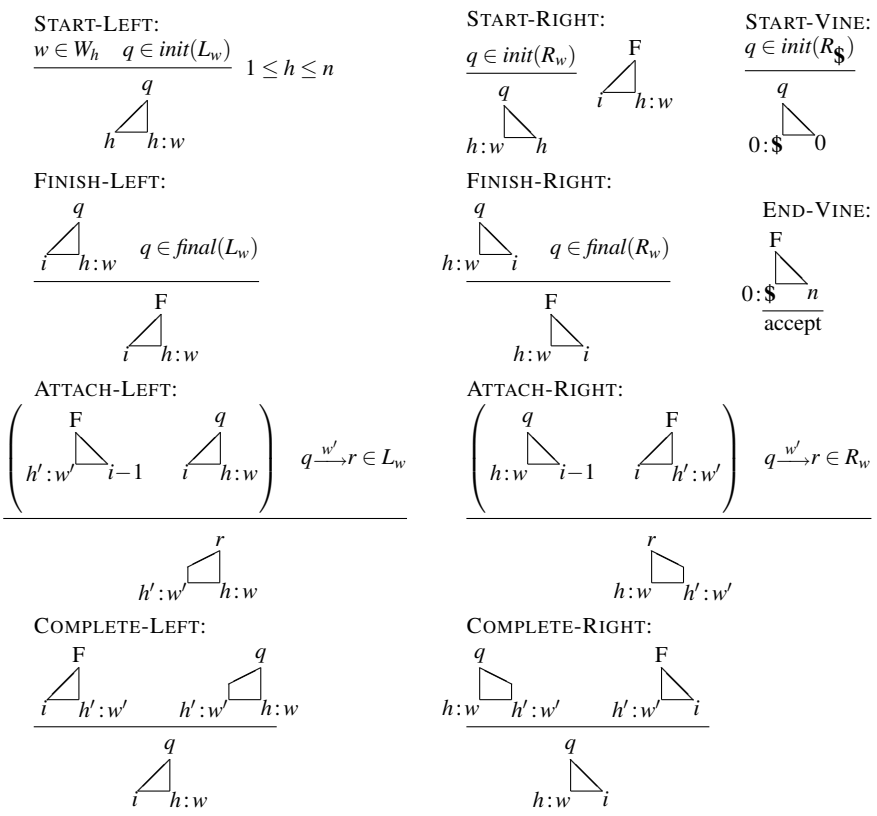


Fig. 2: An algorithm that parses $W_1 \ldots W_n$ in cubic time $O(n^2(n+t')tg^2)$. Adapted with improvements from (Eisner and Satta, 1999, Fig. 3); see footnote 9 for a further practical speedup that delays the disambiguation of word senses. The algorithm is specified as a collection of deductive inference rules. Once one has derived all antecedent items above the horizontal line and any side conditions to the right of the line, one may derive the consequent item below the line. The parentheses in the ATTACH rules indicate the deduction of an intermediate item that "forgets" i. Weighted agenda-based deduction is handled in the usual way (Goodman, 1999): i.e., the weight of a consequent item is the product of the weights of its antecedents (not including side conditions), maximized (or summed) over all ways of deriving that consequent. The probabilities governing the automaton L_w, namely $p(\text{start at } q)$, $p(q \xrightarrow{w'} r \mid q)$, and $p(\text{stop} \mid q)$, respectively give the weights of the axiomatic items $q \in init(L_w)$, $q \xrightarrow{w'} r \in L_w$, and $q \in final(L_w)$; similarly for R_w. The weight of the axiomatic item $w \in W_h$ is 1, but could be modified to define a penalty (not mentioned in Sect. 3.5) for generating w rather than some other element of W_h.

to a narrow range given h. Unfortunately, the half-constituents $\triangle$ and $\triangle$ may still be arbitrarily wide, thanks to arbitrary right- and left-branching: a feasible vine parse may be a sequence of wide trees $\triangle$. Thus there are $O(n^2k)$ possible

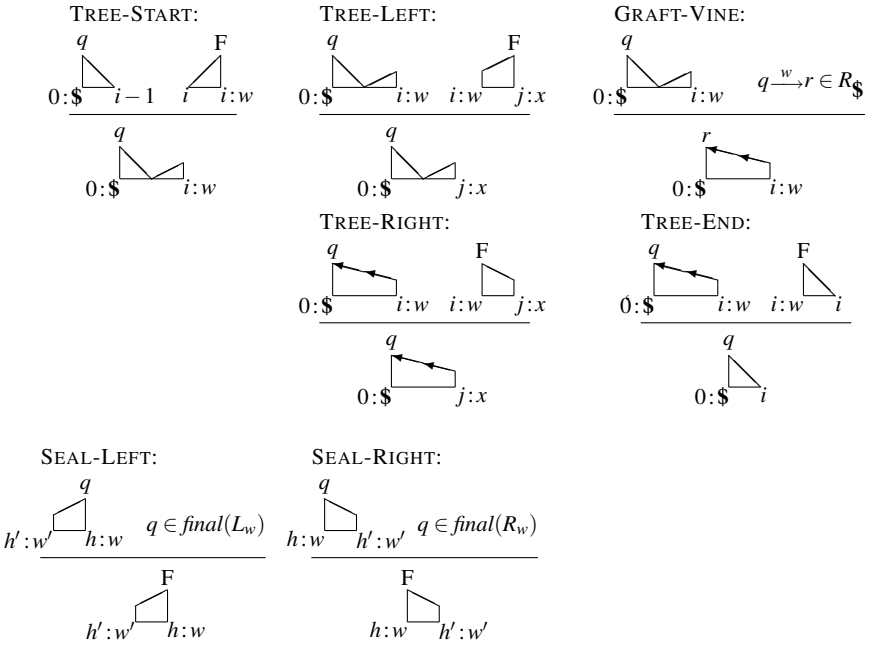


Fig. 3: Extension to the algorithm in Fig. 2. If the ATTACH rules (Fig. 2) are restricted to apply only when $|h-h'| \leq k$, and the COMPLETE rules (Fig. 2) only when $|h-i| < k$, then the additional rules above will assemble the resulting fragments into a vine parse. In this case, ATTACH-RIGHT should also be restricted to $h > 0$, to prevent duplicate derivations (spurious ambiguity). The runtime is $O(nk(k+t')tg^2)$, dominated by the ATTACH rules from Fig. 2; the additional rules above require only $O(nktg^2 + ngtt')$ additional time.

COMPLETE combinations, not to mention $O(n^2)$ ATTACH-RIGHT combinations for which $h = 0$. So the runtime remains quadratic. We now fix this problem.

5.2.3 Approach #3: Specialized Chart Parsing

How, then, do we get linear runtime *and* a reasonable grammar constant? We give two ways to achieve runtime of $O(nk^2)$.

First, we observe without details that we can easily achieve this by starting instead with the algorithm of Eisner (2000),[26] rather than Eisner and Satta (1999), and again refusing to add long tree dependencies. That algorithm effectively concatenates only trapezoids, not triangles (i.e., half-constituents). Each is spanned by a single dependency and so has width $\leq k$. The vine dependencies do lead to wide trapezoids, but these are constrained to start at 0, where **$** is. So the algorithm tries

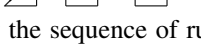
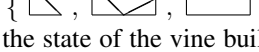
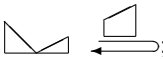
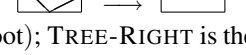
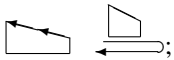
[26] With a small change that when two items are combined, the *right* item (rather than the left) must be simple (in the terms of Eisner (2000)).

at most $O(nk^2)$ trapezoid combinations of the form ${}_h\square_i + {}_i\square_j$ (like the ATTACH combinations above) and $O(nk)$ combinations of the form ${}_0\square_i + {}_i\square_j$, where $i-h \leq k$, $j-i \leq k$. The precise runtime is $O(nk(k+t')tg^3)$, in terms of the parameters of Sect. 3.5. In the unrestricted case where $k = n$, we recover exactly the algorithm of Eisner (2000) and its runtime.

We now propose a hybrid linear-time algorithm that further improves asymptotic runtime to $O(nk(k+t')tg^2)$, saving a factor of g in the grammar constant. While we will still build trapezoids as in Eisner (2000), the factor-of-g savings will come from building the internal structure of a trapezoid from both ends inward rather than from left to right. In the unrestricted case where $k = n$, this improved *runtime* exactly matches that of Eisner and Satta (1999) and Fig. 2 (as given in Sect. 3.5)—although the new *algorithm* itself is a hybrid of Eisner and Satta (1999) and Eisner (2000), since we already saw in Sect. 5.2.2 that simply restricting Eisner and Satta (1999) would not give linear runtime.

We observe that since within-tree dependencies must have length $\leq k$, they can all be captured within Eisner-Satta trapezoids of width $\leq k$. So our vine grammar parse  can be assembled by simply *concatenating* a sequence of the form ()* of these narrow trapezoids interspersed with width-0 triangles. As this is a *regular* sequence, we can assemble it in linear time from left to right (rather than in the order of Eisner and Satta (1999)), multiplying the items' probabilities together. Whenever we start adding the right half  of a tree along the vine, we have discovered that tree's root, so we multiply in the probability of a $\$ \leftarrow$ root vine dependency.

Formally, our hybrid parsing algorithm restricts the original rules of Fig. 2 to build only trapezoids of width $\leq k$ and triangles of width $< k$.[27] The additional inference rules in Fig. 3 then assemble the final VG parse from left to right as just described.

Specifically, the sequence ()* (all with F at the apex) is attached from left to right by the sequence of rules TREE-START TREE-LEFT* GRAFT-VINE TREE-RIGHT* TREE-END in Fig. 3. It is helpful to regard these rules as carrying out transitions in a small FSA whose state set is {  } (all with 0 : $ at the left edge; these pictorially represent the state of the vine built so far). TREE-START is the arc  ; TREE-LEFT is the self-loop  ; GRAFT-VINE is the ε-transition from  that is weighted by the vine dependency probability $p(\$ \leftarrow \text{root})$; TREE-RIGHT is the self-loop  ; finally, TREE-END is the transition  that loops back to the start state to accept the next fragment tree.

[27] For the experiments of Sect. 6.1, where k varied by type, we restricted these rules as tightly as possible given h and h'.

To understand the SEAL-LEFT rule in Fig. 3, notice that if a left trapezoid headed by $h:w$ has already received all its left children, there are two ways that it can be combined with its stopping weight from L_w. Following Fig. 2, we can use COMPLETE-LEFT to turn it into a left triangle for the last time, after which FINISH-LEFT will incorporate its stopping weight, changing its apex state to F. (Word w then combines with its parent using ATTACH-RIGHT or COMPLETE-LEFT, according to whether the parent is to the left or right of h.) However, these rules may not be permitted if their outputs are too wide. The alternative is to incorporate the left trapezoid directly into the vine parse using TREE-LEFT from Fig. 3. In this case, SEAL-LEFT must be used to incorporate the stopping weight, since we have bypassed FINISH-LEFT. (Word w then combines with its parent using GRAFT-VINE or another instance of TREE-LEFT, according to whether the parent is to the left of h (i.e., \$) or the right of h.) SEAL-RIGHT behaves similarly.

5.2.4 Lattice Parsing

Again, for completeness, we explain how to extend the final linear-time algorithm of Sect. 5.2.3 to parse a lattice or other input FSA, Ω.

We modify Fig. 2 to handle lattice parsing, exactly as in Sect. 3.7, and modify Fig. 3 similarly. Now, much as in Sect. 5.2.3, we restrict the ATTACH and COMPLETE rules in the modified Fig. 2 to apply only when the total width of the two triangle or trapezoid antecedents is $\leq k$. We then assemble the resulting fragments using the modified Fig. 3, as before.

But how do we define the width of a triangle or trapezoid? The simplest approach is to specialize these items as proposed in Sect. 3.7. Each such item records a explicit width $\Delta \in [0,k]$. Because of our hard constraints, we never need to build specialize items that are wider than that.[28]

The runtime of this algorithm depends on properties of Ω and its arc lengths. Let n be the number of states and m be the number of arcs. Define M' to be an upper bound, for all states $i \in \Omega$, on the size of the set $\{(j,\Delta) : \Omega$ contains a path of the form $i \ldots j$ having length $\Delta \leq k\}$. Define M similarly except that now j ranges over Ω's arcs rather than its states; note that $M \in [M', M'm]$. An upper bound on the runtime is then $O(mM(M'+t')t)$. In the confusion network case, with n states, $m = ng$ arcs, $M' = O(k)$, $M = O(kg)$, this reduces to our earlier runtime of $O(nk(k+t')tg^2)$ from Sect. 5.2.3.

An alternative approach is at least as efficient, both asymptotically and practically. To avoid specializing the triangle or trapezoid items to record explicit widths,

[28] We do not specialize the vine items, i.e., items whose left boundary is 0:\$. Vine items can have unbounded width $\Delta > k$, but it is unnecessary for them to record this width because it never comes into play.

we define their widths using the shortest-path-based lower bounds from Sect. 3.7.[29] We are willing to combine two such items iff the sum of their widths is $\leq k$.[30]

The resulting search may consider some infeasible parses. In other words, it is possible that the search will return a parse that contains some dependencies that cover string distance $> k$, if this infeasible parse happens to score better than any of the feasible parses.[31] However, this is unproblematic if our goal is simply to prune infeasible parses in the interest of speed. If our pruning is incomplete and we can still maintain linear-time parsing, so much the better. That is, we are not required to consider infeasible parses (since we suppose that they will usually be suboptimal), but neither are we forbidden to consider them or allow them to win on the merits.

If we really wish to consider only feasible parses, it may still be efficient to run the lower-bounding method first, as part of an A* algorithm (cf. Sect. 3.7). Since the lower-bounding method considers too many derivations, it produces optimistic probability estimates relative to the feasible-only parser that specializes the items. Thus, run the lower-bounding parser first. If this returns an infeasible parse, then compute its Viterbi-outside probabilities, and use them as an admissible A* heuristic when reparsing with the feasible-only parser.

6 Experiments with Hard Constraints

Our experiments used the asymptotically fast hybrid parsing algorithm of Sect. 5.2.3. We used the same left and right automata as in model C, the best-performing model from Sect. 3.2. However, we now define $R_{\$}$ to be a first-order (bigram) Markov model (Sect. 5.1). We trained and tested on the same headed treebanks as before (Sect. 4), except that we modified the *training* trees to make them feasible (Sect. 5.2).

Results with hard constraints are shown in Figure 4, showing both the precision/recall tradeoff (upper left) and the speed/accuracy tradeoff (other graphs), for $k \in \{1,2,...,10,15,20\}$. Dots correspond to different values of k. Tighter bounds k typically improve precision at the expense of recall, with the result that on English and Chinese, $k = 7$ (for example) actually achieves better F-measure accuracy than the $k = \infty$ unbounded parser (shown with $+$), not merely greater speed.

---

[29] As in footnote 15, we may precompute the shortest-path distances between all state pairs, but here we only need to do this for the mM pairs whose distances are $\leq k$. Using a simple agenda-based relaxation algorithm that derives all such pairs together with their shortest-path distances, this takes time $O(mMb)$, where $b \leq M'$ is an upper bound on a state's number of outgoing transitions of length $\leq k$. This preprocessing time is asymptotically dominated by the runtime of the main algorithm.

[30] This test is more efficient to implement in a chart parser than requiring the width of the consequent to be $\leq k$. It rules out more combinations, since with lower-bound widths, a consequent of width $\leq k$ could be produced from two antecedents of total width $> k$. (The shortest path connecting its endpoints may not pass through the midpoint where the antecedents are joined.)

[31] For instance, suppose the best derivation of an item of width 3 happens to cover a subpath in Ω of length 5. The item will nonetheless permitted to combine with an adjacent item of width $k-3$, perhaps resulting in the best parse overall, with a dependency of length $k+2$.

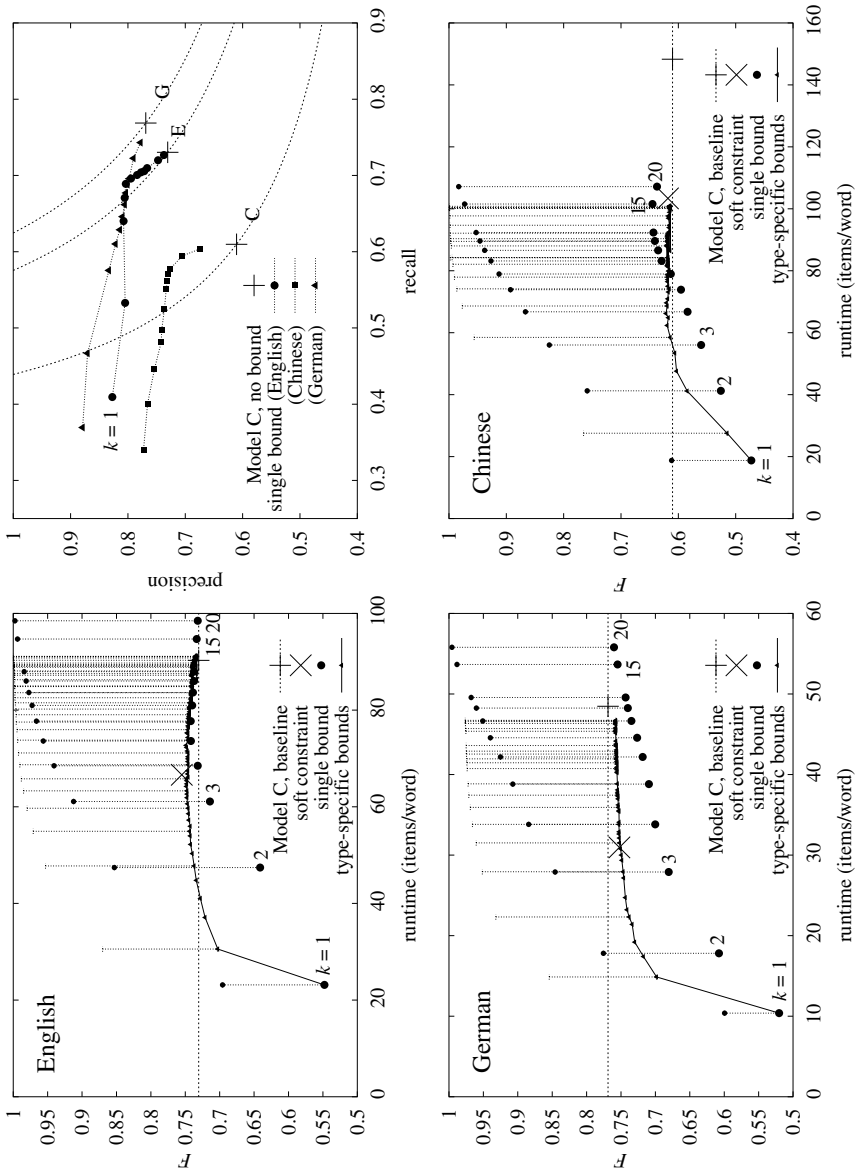


Fig. 4: ***(upper left)*** **Trading recall for precision:** Imposing bounds can improve precision at the expense of recall, for English and Chinese. German performance suffers more. Bounds shown are $k = \{1,2,...,10,15,20\}$. The dotted lines show constant F-measure of the unbounded model. ***(remaining graphs)*** **Trading accuracy for speed by varying the set of feasible parses:** The baseline (no length bound) is shown as $+$. Tighter bounds always improve speed, except for the most lax bounds, for which vine construction overhead incurs a slowdown. Type-specific bounds (Sect. 6.1) tend to maintain good F-measure at higher speeds than the single-bound approach. The vertical bars connect each experiment to its "oracle" accuracy (i.e., the F-measure if we had recovered the best feasible parse, as constructed from the gold-standard parse by grafting: see Sect. 5.2). The "soft constraint" point marked with $\times$ shows the $p(\Delta \mid d,h,c)$-augmented model of Sect. 3.

We observed separately that changing $R_{\$}$ from a bigram to a unigram model significantly hurt accuracy. This shows that it is in fact useful to empirically model likely *sequences* of parse fragments, as our vine grammar does.

Note that we continue to report runtime in terms of items built (see footnote 20). The absolute runtimes are not comparable across parsers because our prototype implementations of the different kinds of parser (baseline, soft constraints, single-bound, and the type-specific bounds in the next section) are known to suffer from different inefficiencies. However, to give a general idea, 60-word English sentences parsed in around 300ms with no bounds, but at around 200ms with either a distance model $p(\Delta \mid d,h,c)$ or a generous hard bound of $k = 10$.

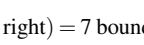
6.1 Finer-Grained Hard Constraints

The dependency length bound k need not be a single value. Substantially better accuracy can be retained if each dependency type—each $(h,c,d) =$ (head tag, child tag, direction) tuple—has its own bound $k(h,c,d)$.[32] We call these *type-specific* bounds: they create a many-dimensional space of possible parsers. We measured speed and accuracy along a sensible path through this space, gradually tightening the bounds using the following process:

1. Initialize each bound $k(h,c,d)$ to the maximum distance observed in training (or 1 for unseen triples).[33]
2. Greedily choose a bound $k(h,c,d)$ such that, if its value is decremented and trees that violate the new bound are accordingly broken, the *fewest* dependencies will be broken.[34]
3. Decrement the bound $k(h,c,d)$ and modify the training data to respect the bound by breaking dependencies that violate the bound and "grafting" the loose portion onto the vine. Retrain the parser on the training data.
4. If all bounds are not equal to 1, go to step 2.

The performance of every 200th model along the trajectory of this search is plotted in Fig. 4. The graph shows that type-specific bounds can speed up the parser to a given level with less loss in accuracy.

---

[32] Note that $k(h,c,\text{right}) = 7$ bounds the width of  . For a finer-grained approach, we could instead separately bound the widths of  and  , say by $k_r(h,c,\text{right}) = 4$ and $k_l(h,c,\text{right}) = 2$.

[33] In the case of the German TIGER corpus, which contains non-projective dependencies, we first make the training trees into projective vines by raising all non-projective child nodes to become heads on the vine.

[34] Not counting dependencies that must be broken indirectly in order to maintain projectivity. (If word 4 depends on word 7 which depends on word 2, and the $4 \rightarrow 7$ dependency is broken, making 4 a root, then we must also break the $2 \rightarrow 7$ dependency.)

7 Related Work

An earlier version of this chapter was originally published as Eisner and Smith (2005). Since then, it has become more common to consider soft dependency-length features in dependency parsing. Indeed, at the same time as our 2005 paper, McDonald et al (2005a) used length features within a discriminatively trained model (versus our deficient generative model that redundantly generates dependency lengths). Furthermore, they considered not only approximately how many words intervened between a child and its parent, but also the POS tags of these words. These length and length-like features were very helpful, and variants were used in a subsequent extension by Hall (2007). Turian and Melamed's history-based parser (Turian and Melamed, 2006) also considered various kinds of length features when making its decisions.

There is also relevant work on soft length constraints that predates ours, and which like our present paper uses generative models. Klein and Manning (2003c) conditioned child generation at a given position on whether the position was adjacent to the parent, and they conditioned stopping on whether the position was 0, 1, 2–5, 6–10, or more than 11 words away from the parent, which is essentially a length feature. Even earlier, Collins (1997) used three binary features of the intervening material as conditioning context for generating a child: did the intervening material contain (a) any word tokens at all, (b) any verbs, (c) any commas or colons? Note that (b) is effective because it measures the length of a dependency in terms of the number of alternative attachment sites that the dependent skipped over, a notion that was generalized by the intervening POS features of McDonald et al (2005a), mentioned above.

Some parsers do not evaluate directly whether a parse respects the short-dependency preference, but they do have other features that address some of the phenomena in Sect. 2. For example, Charniak and Johnson's reranker for phrase-structure parses (Charniak and Johnson, 2005) has "Heavy" features that can learn to favor late placement of *large constituents* in English, e.g., for heavy-shift. However, these other features make rather different distinctions and generalizations than ours do. It would be interesting to compare their empirical benefit.

We have subsequently applied our own soft constraint model to *unsupervised* parsing. By imposing a bias against long dependencies during unsupervised learning, we obtained substantial improvements in accuracy over plain Expectation-Maximization and other previous methods. Further improvements were obtained by gradually relaxing ("annealing") this bias as learning proceeded (Smith and Eisner, 2006; Smith, 2006).

As for hard constraints (Sect. 5), our limitation on dependency length can be regarded as approximating a context-free language by a subset that is a regular language. Our "vines" then let us concatenate several strings in this subset, which typically yields a superset of the original context-free language.

Subset and superset approximations of (weighted) CFLs by (weighted) regular languages, usually by preventing center-embedding, have been widely explored;

Nederhof (2000) gives a thorough review. Our approach limits *all* dependency lengths (not just center-embedding).[35] Further, we derive weights from a modified treebank rather than by approximating the true weights. And though representing a regular language by a finite-state automaton (FSA) is useful for other purposes, we argued that the FSA in this case can be large, and that recognition and parsing are much more efficient with a modified version of a context-free chart parsing algorithm.

Bertsch and Nederhof (1999) gave a linear-time recognition algorithm for the recognition of the regular closure of deterministic context-free languages. Our result is slightly related, since a vine grammar is the Kleene closure of a different kind of restricted CFL (not deterministic, but restricted in its dependency length, hence regular).

Empirically, the algorithms described above were applied in Dreyer et al (2006) to the construction of more interesting dependency parsing models. While the performance of those models was not competitive, that paper presents further evidence that hard bounds on dependency length need not harm the parser's precision.

8 Future Work

The simple POS-sequence models we used as an experimental baseline are certainly not among the best parsers available today. They were chosen to illustrate how modeling and exploiting distance in syntax can affect various performance measures. Our approach may be helpful for other parsing situations as well.

First, we hope that our results will generalize to more expressively weighted grammars, such as log-linear models that can include head-child distance alongside and in conjunction with other rich features.

Second, fast approximate parsing may play a role in more accurate parsing. It might be used to rapidly compute approximate outside-probability estimates to prioritize best-first search (Caraballo and Charniak, 1998, for example). It might also be used to speed up the early iterations of training a weighted parsing model, which for modern training methods tends to require repeated parsing (either for the best parse, as in Taskar et al (2004), or all parses, as in Miyao and Tsujii (2002)). Note that our algorithms also admit *inside-outside* variants (Goodman, 1999), allowing iterative estimation methods for log-linear models such as Miyao and Tsujii (2002).

Third, it would be useful to investigate algorithmic techniques and empirical benefits for limiting dependency length in more powerful grammar formalisms. Our runtime reduction from $O(n^3) \rightarrow O(nk^2)$ for a length-k bound applies only to a "split" bilexical grammar.[36] More expressive grammar formalisms include lexical-

[35] Of course, this still allows right-branching or left-branching to unbounded depth.

[36] The obvious reduction for unsplit head automaton grammars, say, is only $O(n^4) \rightarrow O(n^3k)$, following Eisner and Satta (1999). Alternatively, one can convert the unsplit HAG to a split one that preserves the set of feasible (length $\leq k$) parses, but then g becomes prohibitively large in the worst case.

ized CFG, CCG, and TAG (see footnote 1). Furthermore, various kinds of *synchronous* grammars (Shieber and Schabes, 1990; Wu, 1997) have seen a resurgence in statistical machine translation since the work of Chiang (2005). Their high runtime complexity might be reduced by limiting monolingual dependency length (Schafer and Yarowsky, 2003).

One tool in deriving further algorithms of this sort is to apply general-purpose transformations (Sikkel, 1997; Eisner and Blatz, 2007) to logical algorithm specifications such as the inference rules shown in Figs. 2 and 3. For example, Eisner and Blatz (2007) showed how to derive (a variant of) the $O(n^3)$ algorithm of Fig. 2 by transforming a naive $O(n^5)$ algorithm. Further transformations might be able to continue by deriving Fig. 3, and these transformations might generalize to other grammar formalisms.

Fourth, it would be useful to try limiting the dependency length in *non-projective* parsing, specifically for the tractable "edge-factored" case where $t = 1$ and $g = 1$ (as in our "model A" experiments). Here we can easily show a runtime reduction from $O(n^2) \rightarrow O(kn \log n)$ for a length-k bound. The $O(n^2)$ result for edge-factored non-projective parsing is due to McDonald et al (2005b), who directly applied a directed minimum spanning tree algorithm of Tarjan (1977) to the dense directed graph of all $O(n^2)$ possible dependency edges. Our "vine grammar" restriction would simply strip this graph down to a sparser graph of only $m = O(kn)$ possible edges (namely, the edges of length $\leq k$ together with the edges from \$). Another algorithm also in Tarjan (1977) can then find the desired non-projective tree in only $O(m \log n)$ time ($= O(kn \log n)$). It remains an empirical question whether this would lead to a desirable speed-accuracy tradeoff for non-projective dependency parsing.

Fifth, an obvious application of our algorithms is for linear-time, on-the-fly parsing or language modeling of long streams of tokens. Even though sentence boundaries can be accurately identified on the fly in newspaper text (Reynar and Ratnaparkhi, 1997), this is harder in informal genres and in speech, particularly given the lack of punctuation (Liu et al, 2005). Thus, one might want the syntactic model to help determine the segmentation. This is what a vine grammar does, permitting unboundedly long parsed fragments (which in practice would typically be the top-level sentences) as long as they do not contain long dependencies. For parsing such streams, our $O(nk^2)$ algorithm can be easily adapted to do *incremental* chart parsing in this situation, in linear time and space, perhaps using a k that is fairly generous (but still $\ll n$). For syntactic language modeling!syntactic, an inside-algorithm version can be modified without too much difficulty so that it finds the probability of a given prefix string (or lattice state) under a vine grammar, summing over just the feasible prefix parses.[37] This modest vine approximation to Stolcke's exact PCFG syntactic language model (Stolcke, 1995) could make it more practical by speeding it up from cubic to linear time, as an alternative to switching to the history-based

[37] Note that the vine grammar as we have presented it is a deficient model, since unless we reparameterize it to consider dependency lengths, it also allocates some probability to infeasible parses that are not included in this sum. However, the short-dependency preference suggests that these infeasible parses should not usually contribute much to the total probability that we seek.

models and approximate multistack decoders of subsequent work on syntactic language modeling (Chelba and Jelinek, 2000, *et seq.)*.

9 Conclusion

We have described a novel reason for identifying headword-to-headword dependencies while parsing: to consider their length. We have demonstrated that simple bilexical parsers of English, Chinese, and German can exploit a "short-dependency preference" to improve parsing runtime and dependency precision and the expense of recall. Notably, *soft* constraints on dependency length can improve both speed and accuracy, and *hard* constraints allow improved precision and speed with some loss in recall (on English and Chinese, remarkably little loss). Further, for the hard constraint "length $\leq k$," we have given an $O(nk^2)$ partial parsing algorithm for split bilexical grammars; the grammar constant is no worse than for state-of-the-art $O(n^3)$ algorithms. This algorithm strings together the partial trees' roots along a "vine." We also noted a non-projective variant that runs in time $O(kn \log n)$.

Our approach might be adapted to richer parsing formalisms, including synchronous ones, and should be helpful as an approximation to full parsing when fast, high-precision recovery of syntactic information is needed, or when the input stream is very long.

Acknowledgements This work was supported by NSF ITR grant IIS-0313193 to the first author and a fellowship from the Fannie and John Hertz Foundation to the second author. The views expressed are not necessarily endorsed by the sponsors. The authors thank Mark Johnson, Eugene Charniak, Charles Schafer, Keith Hall, and John Hale for helpful discussion and Elliott Drábek and Markus Dreyer for insights on (respectively) Chinese and German parsing. They also thank an anonymous reviewer for suggesting the German experiments.

References

Abney SP (1991) Parsing by chunks. In: Principle-Based Parsing: Computation and Psycholinguistics, Kluwer

Appelt DE, Hobbs JR, Bear J, Israel D, Tyson M (1993) FASTUS: A finite-state processor for information extraction from real-world text. In: Proceedings of IJCAI

Bangalore S, Joshi AK (1999) Supertagging: An approach to almost parsing. Computational Linguistics 25(2):237–265

Bertsch E, Nederhof MJ (1999) Regular closure of deterministic languages. SIAM Journal on Computing 29(1):81–102

Bikel D (2004) A distributional analysis of a lexicalized statistical parsing model. In: Proceedings of EMNLP

Caraballo SA, Charniak E (1998) New figures of merit for best-first probabilistic chart parsing. Computational Linguistics 24(2):275–98

Charniak E, Johnson M (2005) Coarse-to-fine n-best parsing and maxent discriminative reranking. In: Proceedings of ACL

Charniak E, Goldwater S, Johnson M (1998) Edge-based best-first chart parsing. In: Proc. of VLC

Chelba C, Jelinek F (2000) Structured language modeling. Computer Speech and Language 14:283–332

Chen S (1995) Bayesian grammar induction for language modeling. In: Proceedings of ACL

Chiang D (2005) A hierarchical phrase-based model for statistical machine translation. In: Proceedings of ACL

Church KW (1980) On memory limitations in natural language processing. Master's thesis, MIT

Collins M (1997) Three generative, lexicalised models for statistical parsing. In: Proceedings of ACL

Dreyer M, Smith DA, Smith NA (2006) Vine parsing and minimum risk reranking for speed and precision. In: Proceedings of CoNLL

Eisner J (2000) Bilexical grammars and their cubic-time parsing algorithms. In: Advances in Probabilistic and Other Parsing Technologies, Kluwer

Eisner J, Blatz J (2007) Program transformations for optimization of parsing algorithms and other weighted logic programs. In: Proceedings of FG

Eisner J, Satta G (1999) Efficient parsing for bilexical CFGs and head automaton grammars. In: Proceedings of ACL

Eisner J, Smith NA (2005) Parsing with soft and hard constraints on dependency length. In: Proceedings of IWPT

Eisner J, Goldlust E, Smith NA (2005) Compiling Comp Ling: Practical weighted dynamic programming and the Dyna language. In: Proceedings of HLT-EMNLP

Frazier L (1979) On comprehending sentences: Syntactic parsing strategies. PhD thesis, University of Massachusetts

Gibson E (1998) Linguistic complexity: Locality of syntactic dependencies. Cognition 68:1–76

Gildea D, Temperley D (2007) Optimizing grammars for minimum dependency length. In: Proceedings of ACL

Goodman J (1999) Semiring parsing. Computational Linguistics 25(4):573–605

Grefenstette G (1996) Light parsing as finite-state filtering. In: Proceedings of the Workshop on Extended Finite-State Models of Language

Hall K (2007) *k*-best spanning tree parsing. In: Proceedings of ACL

Hawkins J (1994) A Performance Theory of Order and Constituency. Cambridge University Press

Hindle D (1990) Noun classification from predicate-argument structure. In: Proceedings of ACL

Hobbs JR, Bear J (1990) Two principles of parse preference. In: Proceedings of COLING

Klein D, Manning CD (2003a) A* parsing: Fast exact viterbi parse selection. In: Proc. of HLT-NAACL

Klein D, Manning CD (2003b) Accurate unlexicalized parsing. In: Proceedings of ACL

Klein D, Manning CD (2003c) Fast exact inference with a factored model for natural language parsing. In: Advances in NIPS 15

Klein D, Manning CD (2004) Corpus-based induction of syntactic structure: Models of dependency and constituency. In: Proceedings of ACL

Liu Y, Stolcke A, Shriberg E, Harper M (2005) Using conditional random fields for sentence boundary detection in speech. In: Proceedings of ACL

McDonald R, Crammer K, Pereira F (2005a) Online large-margin training of dependency parsers. In: Proceedings of ACL

McDonald R, Pereira F, Ribarov K, Hajič J (2005b) Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of HLT-EMNLP

Miyao Y, Tsujii J (2002) Maximum entropy estimation for feature forests. In: Proceedings of HLT

Nederhof MJ (2000) Practical experiments with regular approximation of context-free languages. CL 26(1):17–44

Nederhof MJ (2003) Weighted deductive parsing and Knuth's algorithm. Computational Linguistics 29(1):135–143

Reynar JC, Ratnaparkhi A (1997) A maximum entropy approach to identifying sentence boundaries. In: Proceedings of ANLP

Schafer C, Yarowsky D (2003) A two-level syntax-based approach to Arabic-English statistical machine translation. In: Proceedings of the Workshop on MT for Semitic Languages

Shieber S, Schabes Y (1990) Synchronous tree adjoining grammars. In: Proceedings of COLING

Sikkel K (1997) Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms. Texts in Theoretical Computer Science, Springer-Verlag

Smith NA (2006) Novel estimation methods for unsupervised discovery of latent structure in natural language text. PhD thesis, Johns Hopkins University

Smith NA, Eisner J (2005) Contrastive estimation: Training log-linear models on unlabeled data. In: Proceedings of ACL, pp 354–362

Smith NA, Eisner J (2006) Annealing structural bias in multilingual weighted grammar induction. In: Proceedings of COLING-ACL

Stolcke A (1995) An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. Computational Linguistics 21(2):165–201

Tarjan RE (1977) Finding optimum branchings. Networks 7(1):25–35

Taskar B, Klein D, Collins M, Koller D, Manning C (2004) Max-margin parsing. In: Proceedings of EMNLP

Temperley D (2007) Minimization of dependency length in written English. Cognition 105:300–333

Turian J, Melamed ID (2006) Advances in discriminative parsing. In: Proceedings of COLING-ACL

Wu D (1997) Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. Computational Linguistics 23(3):377–404

Index