An $O(n^3)$ Agenda-Based Chart Parser for Arbitrary Probabilistic Context-Free Grammars

Dan Klein and **Christopher D. Manning**
Computer Science Department
Stanford University
Stanford, CA 94305-9040
{klein, manning}@cs.stanford.edu

Abstract

While $O(n^3)$ methods for parsing probabilistic context-free grammars (PCFGs) are well known, a tabular parsing framework for arbitrary PCFGs which allows for botton-up, top-down, and other parsing strategies, has not yet been provided. This paper presents such an algorithm, and shows its correctness and advantages over prior work. The paper finishes by bringing out the connections between the algorithm and work on hypergraphs, which permits us to extend the presented Viterbi (best parse) algorithm to an inside (total probability) algorithm.

1 Introduction and Related Work

Agenda-based active chart parsing (Kay, 1973; Kay, 1980; Pereira and Shieber, 1987) provides an elegant unification of the central ideas of tabular methods for context-free grammar parsing. Earley (1970)-style dotted items in the chart are specified to combine via "the fundamental rule" in an order-independent manner, such that the same basic algorithm supports top-down and bottom-up parsing, and the parser deals correctly with the difficult cases of left-recursive rules, empty elements, and unary rules, in a natural way.

However, while $O(n^3)$ methods for parsing probabilistic context-free grammars (PCFGs) are well known (Baker, 1979; Jelinek et al., 1992; Stolcke, 1995), an elegant general tabular parsing framework for PCFGs, corresponding to active chart parsing for CFGs, has not yet been provided. Producing a probabilistic version of an agenda-driven chart parser is not trivial. A central idea of such parsers is that the algorithm is correct and complete regardless of the order in which items on the agenda are processed. Achieving this is straightforward for categorical parsers, but problematic for probabilistic parsers, because, when an edge is first discovered, it may or may not be correctly scored. For example, consider extending an active edge VP→V.NP PP:[1,2] with an NP:[2,5] to form an edge VP→V NP.PP over [1,5]. In a categorical chart parser (CCP), we can assert the *existence* of this edge as soon as both component edges are found. Any NP will do; it need not be a best NP over that span. However, if we wish to score edges as we go along, there is a problem. In a Viterbi chart parser, if we later find a better way to form the NP, we will have to update not only the score of that NP, but also the score of any edge whose current score depends on that NP's score. This can potentially lead to an extremely inefficient upward propagation of scores every time a new traversal is explored. The agenda-based chart parser of Caraballo and Charniak (1998) (used for determining inside probabilities) suffers from exactly this problem: In Appendix A, they note that such updates "can be quite expensive in terms of CPU time", but merely suggest a method of thresholding which delays probability propagation until the amount of unpropagated probability mass has become significant, and suggest that this allows them to keep the performance of the parser "as $O(n^3)$ empirically." Goodman (1998) provides an insightful presentation unifying many categorical and probabilistic parsing algorithms in terms of the problem's semiring structure, but he merely notes (p. 172) the above problem, and on this basis puts probabilistic agenda-based chart parsers aside.

Most PCFG parsing work has used the bottom-up CKY algorithm (Kasami, 1965; Younger, 1967) with Chomsky Normal Form Grammars (Baker, 1979; Jelinek et al., 1992) or extended CKY parsers that work with n-ary branching grammars, but still not with empty constituents (Kupiec, 1991; Chappelier and Rajman, 1998). Such bottom-up parsers straightforwardly avoid the above problem, by always building all edges over shorter spans before building edges over longer spans which make use of them. However, such methods do not allow top-down grammar filtering, and often do not handle empty elements, cyclic unary productions, or n-ary rules. Stolcke (1995) presents a top-down parser for arbitrary PCFGs, which incorporates elements of the control strategies of Earley's (1970) parser and the Graham-Harrison-Russo parser (Graham et al., 1980). Stolcke provides a correct and efficient solution for parsing arbitrary PCFGs avoiding the problem of left-recursive predictions and unary rule completions through the use of precomputed matrices giving values for the closure of these operations. However, the add-ons for grammars with such rules make the resulting parser rather complex, and again we have a method only for a single parsing regimen, rather than a general tabular parsing framework.

In this paper we show how to do agenda-based active chart parsing of arbitrary PCFGs in $O(n^3)$ time, essentially by extending the key idea of Dijkstra's (1959) shortest path algorithm to chart parsing. The resulting algorithm handles arbitrary PCFGs and a range of parsing strategies in a simpler and more intuitive way than previously presented algorithms. For precisely the reason mentioned above, our parser will necessarily lose some of the agenda processing flexibility of a standard chart parser, however we preserve the flexibility with regard to selection of parsing introduction strategies (e.g. top-down, bottom-up, etc.).[1] We also describe how to extend this algorithm to calculate inside probabilities.

2 Viterbi Parsing Algorithm

Our algorithm has many of the same data structures of a standard active CCP. The fundamental data structure is the *chart*, which is composed of *nodes*, placed between words, and edges. *Edges* are labeled spans of nodes. There are two varieties of edges, active and passive. *Passive edges* are identified by a span and a label, such as NP:[2,5], and represent that there is some parse of that category over the span. *Active edges* are identified by a span, a label, and a grammar state, such as VP→V.NP PP:[1,2], and indicates that the grammar state is reachable over that span. In the case where grammar rules are encoded as lists, this state is simply an Earley-style dotted rule, and to reach it one must have been able to parse the sequence of categories which occur to the left of the dot. However, grammar rules can in general be encoded by any deterministic finite state automaton and so the label of the active edges is in general a DFSA state, with the list rules denoting particularly simple, linear DFSAs. The "fundamental rule" states that new edges are produced by combining active edges with compatible passive edges, advancing the active edge in the process to create a new edge. For example, the two edges described above can combine to form the active edge VP→V NP.PP:[1,5]. Edges themselves do not declare what active and passive edges combined to form that edge; rather this information is recorded in *traversals*, which are simply an (active edge, passive edge, result edge) triple.[2] As each edge can potentially be formed by many different traversals, this distinction between an edge itself and a traversal of an edge is a crucial one, although often lost in pedagogical presentations (e.g., Gazdar and Mellish (1989)).

The core cycle of a CCP is to process traversals into edges and to combine new edges with existing edges to

[1]Note that agenda selection strategies in an exhaustive chart parser merely re-order the work done, while differing introduction strategies can actually impact the total work done.

[2]The result edge is primarily to simplify proofs and pseudocode; it need not ever be stored in a traversal.

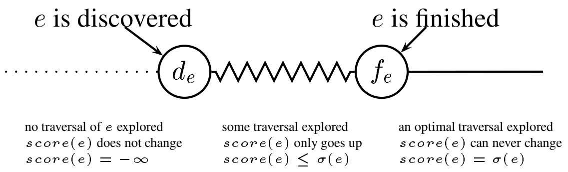

Figure 1: The Life Cycle of an Edge e

create new traversals. Edges which are not formed from other edges via traversals are *introduction edges*. *Passive introduction edges* are the words and are often all introduced during initialization. *Active introduction edges* are the initial states of rules and are introduced in accordance with the grammar strategy (top-down, bottom-up, etc.). To hold the traversals or edges which have not yet been processed, a CCP has a data structure called an *agenda*, which holds both traversals and edge introductions. Items from this agenda can be processed in any order whatsoever, even arbitrarily or randomly, without affecting the final chart contents.

In a probabilistic chart parser (PCP), the central data structures are augmented with scores. Grammar rules, which were previously encoded as symbolic DFSAs are weighted DFSAs with a weight for entering the initial state, a weight on each transition, and, for each accepting state, a weight on accepting in that state.

Edges are also scored, though with the exception of introductory edges whose scores are known *a priori*, edge scores are merely current best estimates. Each edge e is augmented with a field $score(e)$. This score, $score(e)$ (or $score(e,t)$ at a time t) is the best estimate to date of that edge's true (best) probability score, $\sigma(e)$. In our algorithm, this estimate will always be less than or equal to $\sigma(e)$, and will always in fact be equal for introductory edges.

The full algorithm is shown in pseudocode in figure 2. It is broadly similar to the standard categorical chart parsing algorithm. However, in order to solve the problem of entering edges into the chart before their correct score is known, we have a more articulated edge life cycle (shown in figure 1).[3] We distinguish edge *discovery* from edge *finishing*. A non-introductory edge is discovered the first time we explore a traversal which forms that edge (in exploreTraversal). An introductory edge is discovered whenever our parsing strategy indicates (during initialize or some other edge's finishEdge). Discovery is the point when we know that the edge *can* be parsed. An edge is finished when it is inserted into the chart and acted upon (in finishEdge). The primary significance of an edge's finishing time is that, as we will show, our algorithm maintains the property that when an edge is finished, it is correctly scored, i.e., $score(e) = \sigma(e)$.

A CCP stores all outstanding computation tasks in a single agenda, whether the tasks are unexplored traver-

[3]Note that the comments in the figure apply to non-introductory edges, though the timeline applies to all edges.

sals or uninserted introductory edges. We have two agendas and our typing is stronger. To store edges which have been discovered but not yet finished, we have a *finishing agenda*. To store traversals which have been generated but not explored, we have an *exploration agenda*.

The algorithm works as follows. During initialization, all terminal passive edges (one per word in the sentence or lattice) are discovered, along with any initial active edges (for example, all ROOT→.* edges if we are using a top-down strategy and our goal category is ROOT). Note that all of these introductory edges are already correctly scored.[4]

The core loop of the algorithm is shown in figure 3. If there are any traversals to explore, a traversal t is removed from the exploration agenda and processed with exploreTraversal. Any removal order is allowed. In exploreTraversal, t's result edge e is checked against t (relaxEdge). If t forms e with a better score than previously known for e, e's score (and new best traversal) is updated. If e is an undiscovered non-introduction edge, then it becomes discovered.

If the exploration agenda is empty, the finishing agenda is checked. If it is non-empty, the edge with the best current score estimate is finished – removed and processed with finishEdge. This is the point at which the fundamental rule is applied (doFundamentalRule) and new active edges are introduced (in accordance with the active edge introduction strategy).[5]

3 Analysis

We outline the completeness of the algorithm: that it will discover and finish all edges and traversals which the grammar, goal, and words present allow. Then we argue correctness: that every edge e which is finished is, at its finishing time, labeled with the correct score. Finally, we give tight worst-case bounds on the time and memory usage of the algorithm.

3.1 Completeness

For space reasons, we simply sketch a proof of completeness. For the full proof, see (Self, in preparation). In order to argue completeness for a variety of word and rule introduction strategies, it is important to have a concrete notion of what such strategies are. Constraints on the word introduction strategy are only needed for correctness, and so we defer discussion until then. Let E be the set of edges, P the set of introductory passive edges (i.e., word edges), and A the set of introductory active edges (i.e., rule introductions).

[4] Other word introduction strategies are trivially possible, such as scanning the words incrementally in an outer loop from left-to-right. A sufficient constraint on scanning strategies is presented in section 3.

[5] In the application of the fundamental rule, an (active, passive) pair can potentially create two traversals. In categorical DFSA chart parsing, edges may be active, passive, or both. However, the passive and active versions of what would have been a single active/passive edge in a categorical parser will not in general have the same score, and so the algorithm introduces separate edges. The reason the scores will not generally be the same is discussed in section 3.

```
parse(Lattice sentence, Category goal)
  initialize(sentence, goal)
  while finishingAgenda is non-empty
    while explorationAgenda is non-empty
      get a traversal t from the explorationAgenda
      exploreTraversal(t)
    get a best edge e from the finishingAgenda
    finishEdge(e)

initialize(Lattice sentence, Category goal)
  create a new chart and new agendas
  for each word w:[start,end] in the sentence
    discoverEdge(w:[start,end])
  for each node x in the sentence
    if allow-empties
      discoverEdge(empty:[x,x])
  if top-down
    for each initial active edge a which can lead to the goal
      discoverEdge(a)

exploreTraversal(Traversal t)
  e = t.result
  if notYetDiscovered(e)
    discoverEdge(e)
  relaxEdge(e, t)

relaxEdge(Edge e, Traversal t)
  newScore = combineScores(t)
  if (newScore is better than e.score)
    update e.score
    update e.bestTraversal

discoverEdge(Edge e)
  add e to the finishingAgenda

finishEdge(Edge e)
  add e to the chart
  doFundamentalRule(e)
  if top-down
    doTopDownRule(e)
  if bottom-up
    doBottomUpRule(e)

doFundamentalRule(Edge e)
  if e is passive
    for all active edges a which end at e.start
      for active and/or passive result edges r
        create the traversal t = (a, e, r)
        add t to the explorationAgenda
  if e is active
    for all passive edges p which start at e.end
      for active and/or passive result edges r
        create the traversal t = (e, p, r)
        add t to the explorationAgenda

doTopDownRule(Edge e)
  if e is active
    for all transitions r that e can take
      for all introductory active edges a with LHS r.label
        if notDiscovered(a) then discoverEdge(a)

doBottomUpRule(Edge e)
  if e is passive
    for all introductory active edges a whose rules can begin
with e.label
      if notDiscovered(a) then discoverEdge(a)
```

Figure 2: Our Probabilistic Chart Parser

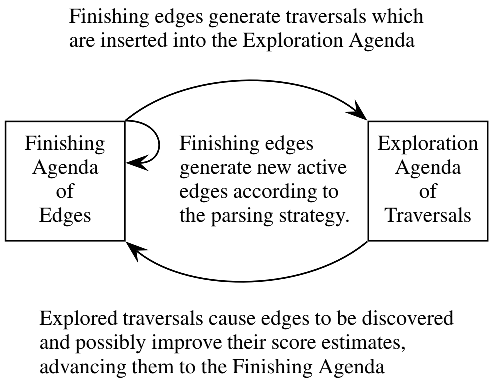

Figure 3: The Core Loop of the Parser

Definition 1 *An* edge-driven *rule introduction strategy is a mapping* $R: E \rightarrow 2^A$ *which takes an edge* e *to a set* I *of introductory active edges which are to be immediately discovered when* e *is finished. By* $I^{-1}(a)$ *we mean the set of edges* e *with* $a \in I(e)$.

The standard top-down and bottom-up strategies are both edge-driven.[6]

Theorem 1 *For any rule-driven active edge introduction strategy* R*, any DFSA grammar* G*, and any input lattice* L*, and goal edge* g*, there exists some agenda selection function* S *for which the sequence of edge insertions* I *made by a categorical chart parser and the sequence of edge finishings* F *made by our probabilistic chart parser are the same.*

Completeness means that the edges found (i.e., *finished*) by both parsers will be the same. This does *not* mean that the PCP will score them correctly, just that every edge which has a parse allowed by the grammar, words, and goal will be found.

The general idea behind this proof is to run the two parsers in parallel, showing by induction over corresponding points in the algorithms that $I = F$ so far and that for every edge in the PCP's finishing agenda, that edge is backed by some edge or traversal which can form it from the CCP's agenda. The selection function is the one which always makes the CCP process the agenda items which will cause the insertion of whichever edge the PCP will select from its finishing agenda at that point.

One important corollary of the full proof is that the parsability is constructive. Not only is every edge each parser discovers parsable, but there is some set of introductory edges and traversals processed by both which can be used to form a parse of that edge. We will need this corollary for our correctness proof.

[6] An example of a non-edge-driven strategy would be if we introduced an arbitrary undiscovered edge from A into an arbitrary zero-span whenever the finishing agenda was empty. It appears very difficult to state a criterion for non-edge-driven strategies which guarantees both their completeness and correctness.

Given this theorem, we inherit the completeness results from the literature on categorical chart parsing, including, in particular, the completeness of the top-down and bottom-up introduction strategies.

3.2 Correctness

We now show that any edge which is finished is correctly scored at finishing.

First, we need some notions about parse trees. A parse tree P is a binary tree of edge tokens[7] . A leaf in this tree is a token of either a word (if passive) or a rule introduction (if active). A non-leaf x is a token of a non-introductory edge and it has two children, a, and p which are tokens of an active edge and a passive edge, respectively, forming a token of some traversal (a, p, x) of that edge type. The reason we must make a type/token distinction is that a given edge may appear more than once in a parse tree. For example, consider empty words which may be used several times over the same zero-span, or an introductory active edge for a left-recursive rule. In this proof we use $type(x)$ to mean the edge type of a parse tree node x. We will say things like "active leaf" for "active introductory edge token" and "discovered node" for "node whose edge type has been discovered."

The basic idea is to avoid finishing incorrectly scored edges by always finishing the highest-scored edge available. This will cause us to work in an inside-outwards fashion when necessary to ensure that score propagation is never needed. The chief difficulties therefore occur when what might have been a high-scoring edge is unavailable for some reason.

The subtlest way this can occur is when an introductory edge is discovered too late. If this happens we may have already mistakenly finished some other edge, assigning it the best score that it could have had *without* that introductory edge's presence in the grammar or input. Because of this, we need tighter constraints on word and rule introduction strategies to prove correctness than those needed for completeness.

The condition on word introduction is simple.

Definition 2 *The* Word Introduction Condition *(or "no internal insertion"): If some edge* e *spanning a span* S *is being finished at time* f_e *then all words in* S *have been discovered at* f_e.

This is satisfied by any reasonable lattice scanning algorithm and any sentence scanning algorithm whatsoever. The only disallowed strategy is to insert words from a lattice into a span which has already had some spanning set of words discovered and has already been parsed. It should be fairly clear that this kind of internal-insertion strategy will lead to problems.

[7] Note that the binary tree corresponds to both the underlying n-ary parse and to an actual parse in a left-binarized version of the underlying n-ary grammar (modulo active introductions).

Now we supply some theoretical machinery for a condition on rule introductions.

Definition 3 *A possibly partial ordering $\prec_P$ of nodes (edge tokens) in a parse tree P* allows descent *if whenever a node x dominates a set of children C, for any $c \in C$, $c \preceq_P x$.*

Definition 4 *The* Rule Introduction Condition *(or "no rule blocking"): In any parse P of an edge e, there is some ordering $\prec_P$ of nodes which allows descent and such that for any active introduction edge a,* EITHER

(1) there is some edge x with a token x' in P which does not dominate any token a' of a and whose finishing will cause the introduction of a, that is $a \in I(x)$, and $x' \prec_P a'$, OR

(2) a must appear in any parse of e if e is passive and any edge which can extend from e if e is active.

This is wordy, but the key idea is that active introduction edges must "depend" on some other edge in the parse in such a way that if an active edge is undiscovered, we can track back to find another edge earlier in the parse which must also be undiscovered.

The last constraint we need is one on the weights of the DFSA rules. If a prefix of a rule is bad, its continuations must be as bad or worse. Otherwise, we may incorrectly delay extending a low scoring prefix.

Definition 5 *The* Grammar Weighting Condition *(or "no score gain"): No transition can improve the score of a trajectory through a DFSA rule, including starting state costs, transition costs, and accepting state costs.*

Now we are ready to state the theorem.

Theorem 2 *Given any DFSA grammar G and introduction strategies obeying the conditions above, for any input lattice L, any edge e which is finished by the algorithm at some time f_e has the property that $score(e, f_e) = \sigma(e)$.*

The proof is by contradiction. Take the first edge e which is selected from the finishing agenda and finished with an incorrect score estimate. Therefore at e's finishing time f_e, $score(e, f_e) < \sigma(e)$. Since e has a parse (by completeness), it has at least one (unfound) best parse. Choose one and call it BP. By virtue of being a best parse, $\sigma(BP) = \sigma(e)$. [8]

We claim that there is some edge x in BP which, at f_e, has been discovered, is correctly scored, i.e., $score(x, f_e) = \sigma(x)$, yet has not been finished. Assume such an x exists. Since x is discovered but not finished at f_e, it was in the finishing agenda with its current score just before e was chosen to be finished. But e was chosen from the finishing agenda, not x, so it must be that $score(x, f_e) \leq score(e, f_e)$.

On the other hand, since x is contained in BP, by "no score gain" it must be that $\sigma(e) = \sigma(BP) \leq \sigma(x)$. Thus, if we find such an edge x then $\sigma(e) \leq \sigma(x) = score(x, f_e) \leq score(e, f_e) < \sigma(e)$, a contradiction.

The rest of the proof involves showing the existence of such an x. Consider the nodes in BP. Since e is unfinished, there is a non-empty set of unfinished nodes, call it U. We want some $u \in U$ which both has no unfinished children and which is minimal by $\prec_P$. Clearly some elements are minimal since U is non-empty and finite. Call the set of minimal elements M. For any $u \in M$ which has an unfinished child, that child must also be minimal since $\prec_P$ allows descent. Therefore, removing all elements from M which have an unfinished child leaves a non-empty set. Choose any u from this set.

If u dominates two finished children (call them a and p), then since e is the first incorrectly finished edge, a and p's edge types had their correct scores at their finishing times. Because only relaxation can change score estimates and it can only raise them, their score estimates have not changed since then. Therefore, whenever the later of $type(a)$ and $type(p)$ was finished, the traversal $t = (type(a), type(p), type(u))$ was generated. And before anything else could have been finished, t was explored. Thus, $type(u)$ has been discovered and has been relaxed by t, say at time r_t. Therefore, at r_t, and therefore still at f_e, $score(type(u))$ can be no worse than its score in BP, which of course means its score has been correct since r_t. But recall that u is unfinished, so we are done.

If u dominates no finished nodes, then it is a leaf. If u is a word introduction, then by "no internal insertion" u has been discovered. Since words are correctly scored at discovery, we are done. If u is an rule introduction, then we must only show that u has been discovered, since rule introductions are correctly scored on discovery. To be sure of this, we must appeal to "no rule blocking." It is possible that u is a token of an edge which must appear in any parse of any passive edge equal to or extendable from e. If so, by condition (1) we are done. If not, then let x be the edge whose finishing will guarantee u's discovery. If x is unfinished, then some instance of x is $\prec_P u$ and unfinished, contradicting u's minimality. Thus we are done.

We have now proven the correctness of the algorithm for strategies meeting the given criteria. Both the traditional bottom-up and top-down strategies meet the "no rule blocking" criterion. We prove this for only the top-down strategy here.

Theorem 3 *The top-down rule introduction strategy meets the "no rule blocking" criterion.*

Assign to each node e in P the number of nodes in P which would be completed before e in a top-down

[8] We define the score function σ for a traversal t of an edge e to mean the score of the highest scoring parse of e which contains an instance of t at its root, and we define it for a parse P to be the score of that specific parse.

stack parse of P. Since no node is completed before its children in such a parse, this allows descent. In a top-down parse, for every introduction node there is a c-commanding active edge node to the left of it which will introduce it, and therefore (1) holds unless the active edge introduction in question is the very leftmost one of P. Call this leftmost active node a. Since P is a parse of some edge e with label L, a also has label L. Thus, if any parse R whatsoever of e is found, its leftmost active leaf is a token of some active edge b with label L. But then, whenever b was discovered, so was a, since the top-down introduction strategy always introduces all rules of the same label at once.

3.3 Asymptotic Bounds and Performance

We briefly motivate and state the complexity bounds. Let n be the number of nodes in the input lattice, C the number of categories in the grammar, and S the number of states in the grammar. $C \leq S$ since each state is contained in a rule for some category. The maximum number of edges E is $(C + S)n^2$, and the maximum number of traversals T is $2SCn^3$. Time is dominated by the work per traversal, which can be made amortized $O(1)$ (with a Fibonacci heap-backed priority queue), so the total time is $O(T) = O(SCn^3)$. For memory, there are several $O(E)$ data structures holding edges. The concern is the exploration agenda which holds traversals. But everything on this agenda at any one time resulted from a single call to doFundamentalRule, and so its size is $O(E)$. Therefore, the total memory is $O(E) = O(Sn^2)$.

Much work in probabilistic parsing has concentrated on "best first" or beam parsing (Collins, 1997; Caraballo and Charniak, 1998). There are obvious good reasons for exploring such methods, but such work has often partially been justified on the grounds of the impossibility of doing exhaustive parsing for large probabilistic grammars (e.g, Caraballo and Charniak (1998, 292) suggest that for a covering grammar derived from the Penn Treebank that "it was impractical to parse sentences to exhaustion using our existing hardware"). It is thus perhaps of interest to note that, using a trie DFSA rule representation, our parser can do exhaustive parsing with a Penn Treebank WSJ covering grammar of sentences of the usually considered lengths (up to 100 words) in reasonable time on a Pentium III workstation with 1 Gb of memory – even with a Java implementation.

4 Parsing and Hypergraphs

One view of parsing is as a process of logical deduction (Pereira and Warren, 1983; Shieber et al., 1995). There is also a well-known, deep connection between logic, in particular satisfiability, and directed hypergraphs (Berge, 1973; Gallo et al., 1993). In this section, we draw the third line in the triangle to connect parsing with directed hypergraph algorithms. The advantage of this view is that, while there is no clear way to generalize the logical view of parsing to probabilistic parsing, directed hypergraphs have an immediate and well-studied notion of weighting. We use this connection to describe the preceding algorithm in hypergraph-theoretic terms and to outline a more complex algorithm for computing inside probabilities within a cubic time bound.

First we give some preliminary definitions about directed hypergraphs. For a more detailed treatment, see for example (Berge, 1973).

Definition 6 *A* directed hypergraph $\mathbf{G}$ *is a pair* (V, E) *where* V *is a set of nodes and* E *is a set of* directed hyperarcs. *A directed hyperarc is a pair* (T, H) *where the tail* T *and head* H *are disjoint subsets of* V.

Definition 7 *A* B-(hyper)graph *is a directed hypergraph where the hyperarcs are* B-arcs. *A B-arc is a hyperarc for which* H *is a singleton set.*

It is easy to see the construction which will provide the link between hypergraphs and deduction. Nodes p will correspond to propositions, and directed hyperarcs $\{t_1 \dots t_m\} \rightarrow \{h_1 \dots h_n\}$ will correspond to a rule $t_1 \wedge \dots t_m \rightarrow h_1 \vee \dots h_n$. In the case of B-arcs, the corresponding rules will be Horn clauses.

Definition 8 *A* simple path $p = s \rightsquigarrow t$ *is a sequence* $(s = v_0, e_1, v_1, \dots e_n, v_n)$ *of alternating nodes and hyperarcs where*

(1) each hyperarc is distinct
(2) $\forall i \in \{1, \dots, n\}, v_i \in \mathit{head}(e_i)$
(3) $\forall i \in \{0, \dots, n-1\}, v_i \in \mathit{tail}(e_{i+1})$

Definition 9 *A* B-path P *in a B-graph* G *from a node* s *to a node* t *is a minimal subgraph* $(V_P, E_P) < G$ *in which:*

$s, t \in V_P$
$\forall v \in V_P - \{s\}, \exists p = s \rightsquigarrow t$, *$p$ a simple path in P*

We can now state the hypergraphical view of parsing. We view the edges and traversals in our grammar as a B-graph $\mathbf{G}$. Edges are the nodes in $\mathbf{G}$, while traversals are B-arcs with the active and passive edges in the tail and the result edge as the head. We call this the *induced binarized B-graph* of a grammar G.[9] This B-graph corresponds to the satisfiability graph of the Horn clause form of the binarized grammar.

For satisfiability graphs, one also needs to add special nodes for *true* and *false* and arcs to represent truth assignments. Similarly, we will want to add a special node s for a source and arcs $(s, w : [x, y])$ for any word spanning $[x,y]$ in our input lattice. We also must add arcs to each introductory active edge. Once we have done this, we call the result the induced binarized B-graph of the grammar G and input lattice L.

We now equate B-reachability with parse existence.

[9]Note that one can straightforwardly view n-ary grammar rules as B-arcs with tails of size n, but we restrict our attention to the view which most closely parallels the parsing done in this paper.

Definition 10 *A node t in a B-graph is B-reachable from a source s iff there exists a B-path from s to t.*

Theorem 4 *In an induced binarized B-graph of a grammar G and a lattice L, a node e is B-reachable from s iff a parse of the edge e exists. Furthermore, each parse of e corresponds to a particular B-path.*[10]

We do not prove this here; the statement itself carries the important observation. Note that a CCP can be seen as doing a reachability search over this graph. The key difference is that in chart parsing we do not have the entire graph to begin with, but rather we construct it on the fly. Just as reachability can be done breadth-first, depth-first, forward, or backwards, categorical chart parsing can be done in any direction whatsoever – provided the introduction rules which dynamically create the graph do so properly.

B-reachability algorithms generally run in time linear in the size of the graph (Gallo et al., 1993). Note that the size of the graph is $O(V+E) = O(E+T) = O(SCn^3)$, exactly the bound for CCPs.

At this point we have closed the triangle. Now we develop the connection between Viterbi parsing and weighted hypergraph algorithms.

Definition 11 *The* weight *of a B-arc A is a function $w\colon A \to \mathbb{R}$ which assigns a weight to each arc.*

Definition 12 *The* cost *of a B-path P is the sum (or product) of the weights of all arcs on the path.*

Note that if the weight of a traversal B-arc is the transition cost of that traversal and if the weight of the (s, e) edge introduction arcs is the score of the introduced edge, then the cost of a path is the score of the (minimal) corresponding parse. Thus, we can view best-parse algorithms as shortest-path algorithms. In exactly this sense, the algorithm presented above is closely related to the B-graph extension of Dijkstra's algorithm (Gallo et al., 1993). It should be no surprise that our proof of correctness is, at the core, parallel to the standard Dijkstra's algorithm correctness proofs. It should also be no surprise that the main reason our proofs of correctness were more challenging is the "just in time" nature of edge introductions.

4.1 Inside Probabilities

One can push the correspondence further. Calculating inside probabilities is essentially the problem of summing the costs of all B-paths. There are several issues with this problem which did not arise before.

First, dynamic programming solutions to the sum-paths problem either do not deal with cycles (meaning we would have to restrict or transform our grammar to make sure there weren't any in the hypergraph) or do not run in linear time (which is undesirable).

Second, with sum-paths the subtle non-reversibility of the parse to path mapping becomes an issue. While many parses can correspond to a single B-path, there is at most one best parse corresponding to a given B-path. The other, non-minimal, parses don't matter for categorical parsing because they do not create any new edges, and they don't matter for best parsing because they never improve an edge's score. However, for inside probabilities, they are important to count.

Due to space constraints, we merely sketch an algorithm for computing inside probabilities for arbitrary PCFGs. The core issue is that any edge x which can be used in a parse of an edge e can contribute to the score of e. If there are no empties or unary transitions, it is easy enough to put a condition on when an x can contribute to e's sum: if $span(x) \prec span(e)$. In that case, one could organize the computation around span sizes, finishing all edges of a given span size before moving on to the next, accumulating traversals as they are explored. This is exactly the CKY algorithm for inside probabilities put into our framework. It also corresponds to the obvious linear time path-sum dynamic program for acyclic B-graphs.

Once we allow an arbitrary grammar, the B-graph develops cycles. For a normal graph with cycles, one can model the path-sum flow and use matrix math to solve in closed form the infinite sums caused by cyclic dependencies (Stolcke, 1995). This is not a linear time operation, and is in fact cubic in the number of graph nodes as it involves a matrix inversion. However, in our B-graph the cycles are not all across the graph, but rather exist only among edges with identical spans, in regions which are in fact isomorphic to normal graphs. Furthermore, even in these regions we need not do a matrix closure over the entire same-span region, but rather only its strongly connected components (SCCs). The SCCs themselves form an acyclic B-graph for which we can use a linear algorithm.

Our algorithm uses these ideas. In addition to edges and traversals, we will have an SCC object called an edge cluster. Edges will still have scores which will be their total sum probability estimate to date (always an underestimate or correct). Clusters will have not scores but tier values (analogous to span sizes for CKY but which also order SCCs inside same-span regions). The finishing agenda will hold clusters and finishing a cluster will happen when all the acyclic inputs to the edges in the cluster are known. Finishing will involve a matrix closure over the cluster at which point all of the edges in the cluster will be finished.

The difficultly is then online tier numbering, SCC detection, and the matrix inversions. Of these, the matrix inversion costs are the largest potential asymptotic slowdown, totaling up to $O(S^3n^2)$ time.

The runtime of the algorithm is $O(SCn^3 + S^3n^2)$ and the space requirements are $O(S^2n^2)$. At first sight, one

[10]The actual path-parse correspondence is not one-to-one in general because of cyclic same-span constructions, but categorical parsing is concerned with parsability of edges, not individual parse representation.

might wonder how this can possibly be superior to simply transforming the grammar once and doing CKY parsing. S^2 and S^3 would be very large for a huge grammar, and one may well be doing the same work over and over again, in essence doing the same grammar transform for every span.

However, there are reasons to prefer this method. First, all causes of cycles (left-recursion, empties, and unaries), are treated identically and naturally.

Second, for a huge grammar, this method could be much faster. The S^3 term comes from the matrix inversions which are actually cubic only in the size of the actual SCCs.[11] For large grammars, the total may be small compared to n^3 and may not appreciably slow the parser down. Furthermore, if those sizes are much much less than S, the small inversions could be cheaper, even done n^2 times, than a single naive S^3 transform of the entire grammar. Note that this win is not biggest for, say, standard lexicalized grammars, where the interaction between the lexical items and categories is easy to factor and it is easy enough to preprocess the CFG backbone, but rather in complex grammars where the interactions between conditioning environments may be subtle and difficult to compute in advance. In this case, our algorithm will straightforwardly and dynamically do only the necessary SCC analysis and inversions.

[11] E.g., without unaries the max size of an SCC is C not S.

References

James K. Baker. 1979. Trainable grammars for speech recognition. In D. H. Klatt and J. J. Wolf, editors, *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, pp 547–550.

C. Berge. 1973. *Graphs and Hypergraphs*. North-Holland, Amsterdam.

Sharon A. Caraballo and Eugene Charniak. 1998. New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, 24:275–298.

J.-C. Chappelier and M. Rajman. 1998. A generalized CYK algorithm for parsing stochastic CFG. In *First Workshop on Tabulation in Parsing and Deduction (TAPD98)*, pp 133–137, Paris.

Michael John Collins. 1997. Three generative, lexicalised models for statistical parsing. In *ACL 35/EACL 8*, pp 16–23.

E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

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

G. Gallo, G. Longo, S. Pallottino, and Sang Nguyen. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201.

Gerald Gazdar and Chris Mellish. 1989. *Natural Language Processing in Prolog*. Addison-Wesley.

Joshua Goodman. 1998. *Parsing inside-out*. Ph.D. thesis, Harvard University.

Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, July.

F. Jelinek, J. D. Lafferty, and R. L. Mercer. 1992. Basic methods of probabilistic context free grammars. In P. Laface and R. De Mori, editors, *Speech Recognition and Understanding: Recent Advances, Trends, and Applications*, volume 75 of *Series F: Computer and Systems Sciences*. Springer Verlag.

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

Martin Kay. 1973. The MIND system. In Randall Rustin, editor, *Natural Language Processing*, pages 155–188. Algorithmics Press, New York.

Martin Kay. 1980. Algorithm schemata and data structures in syntactic processing. Technical Report CSL-80-12, Xerox PARC, Palo Alto, CA, October.

Julian Kupiec. 1991. A trellis-based algorithm for estimating the parameters of a hidden stochastic context-free grammar. In *Proceedings of the Speech and Natural Language Workshop*, pages 241–246. DARPA.

Fernando Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis*, volume 10. CSLI Publications, Stanford, CA.

Fernando C.N. Pereira and David H.D. Warren. 1983. Parsing as deduction. In *ACL 21*, pp 137–144.

Self. in preparation. XXX. MS.

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

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21:165–202.

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