Control Strategies for Parsing with Freer Word-Order Languages

Gerald Penn
Dept. of Computer Science
University of Toronto
Toronto M5S 3G4, Canada

Stefan Banjevic
Dept. of Mathematics
University of Toronto
Toronto M5S 2E4, Canada

Michael Demko
Dept. of Computer Science
University of Toronto
Toronto M5S 3G4, Canada

{gpenn,banjevic,mpademko}@cs.toronto.edu

Abstract

We provide two different methods for bounding search when parsing with freer word-order languages. Both of these can be thought of as exploiting alternative sources of constraints not commonly used in CFGs, in order to make up for the lack of more rigid word-order and the standard algorithms that use the assumption of rigid word-order implicitly. This work is preliminary in that it has not yet been evaluated on a large-scale grammar/corpus for a freer word-order language.

1 Introduction

This paper describes two contributions to the area of parsing over freer word-order (FWO) languages, i.e., languages that do not readily admit a semantically transparent context-free analysis, because of a looser connection between grammatical function assignment and linear constituent order than one finds in English. This is a particularly ripe area for constraint-based methods because such a large number of linguistic partial knowledge sources must be brought to bear on FWO parsing in order to restrict its search space to a size comparable to that of standard CFG-based parsing.

The first addresses the indexation of tabled substrings in generalized chart parsers for FWO languages. While chart parsing can famously be cast *as* deduction (Pereira and Warren, 1983), what chart parsing really *is* is an algebraic closure over the rules of a phrase structure grammar, which is most naturally expressed inside a constraint solver such as CHR (Morawietz, 2000). Ideally, we would like to use standard chart parsers for FWO languages, but because of the constituent ordering constraints that are implicit in the right-hand-sides (RHSs) of CFG rules, this is not possible without effectively converting a FWO grammar into a CFG by expanding its rule system exponentially into all possible RHS orders (Barton et al., 1987). FWO grammar rules generally cannot be used as they stand in a chart parser because tabled substrings record a non-terminal category C derived over a contiguous subspan of the input string from word i to word j. FWO languages have many phrasal categories that are not contiguous substrings.

Johnson (1985), Reape (1991) and others have suggested using bit vectors to index chart edges as an alternative to substring spans in the case of parsing over FWO languages, but that is really only half of the story. We still need a control strategy to tell us where we should be searching for some constituent at any point in a derivation. This paper provides such a control strategy, using this data structure, for doing search more effectively with a FWO grammar.

The second contribution addresses another source of constraints on the search space: the length of the input. While this number is not a constant across parses, it is constant within a single parse, and there are functions that can be precomputed for a fixed grammar which relate tight upper and lower bounds on the length of the input to both the height of a parse tree and other variables (defined below) whose values bound the recursion of the fixed phrase structure rule system. Iteratively computing and caching the values of these functions as needed allows us to invert them efficiently, and bound the depth of the search. This can be thought of as a partial substitute for the resource-bounded control that bottom-up parsing generally provides, Goal-directedness

Proceedings of the 3rd Workshop on Constraints and Language Processing (CSLP-06), pages 9–16,
Sydney, July 2006.

is maintained, because — with the use of constraint programming – it can still be used inside a top-down strategy. In principle, this could be worthwhile to compute for some CFGs as well, although the much larger search space covered by a naïve bottom-up parser in the case of FWO grammars (all possible subsequences, rather than all possible contiguous subsequences), makes it considerably more valuable in the present setting.

In the worst case, a binary-branching immediate dominance grammar (i.e., no linear precedence) could specify that every word belongs to the same category, W, and that phrases can be formed from every pair of words or phrases. A complete parsing chart in this case would have exponentially many edges, so nothing in this paper (or in the aforementioned work on bit vectors) actually improves the asymptotic complexity of the recognition task. Natural languages do not behave like this, however. In practice, one can expect more polymorphy in the part-of-speech/category system, more restrictions in the allowable combinations of words and phrases (specified in the immediate dominance components of a phrase structure rule system), and more restrictions in the allowable orders and discontinuities with which those argument categories can occur (specified in the linear precedence components of a phrase structure rule system).

These restrictions engender a system of constraints that, when considered as a whole, admit certain very useful, language-dependent strategies for resolving the (respectively, don't-care) nondeterministic choice points that a (resp., all-paths) parser must face, specifically: (1) which lexical categories to use (or, resp., in which order), given the input words, (2) which phrase structure rules to apply (resp., in which order), and (3) given a particular choice of phrase structure rule, in which order to search for the argument categories on its right-hand side (this one is don't-care nondeterministic even if the parser is looking for only the best/first parse). These heuristics are generally obtained either through the use of a parameter estimation method over a large amount of annotated data, or, in the case of a manually constructed grammar, simply through some implicit convention, such as the textual order in which the lexicon, rule system, or RHS categories are stated.[1]

This paper does not address how to find these heuristics. We assume that they exist, and instead address the problem of adapting a chart parser to their efficient use. To ignore this would involve conducting an enormous number of derivations, only to look in the chart at the end and discover that we have already derived the current bit-vector/category pair. In the case of standard CFG-based parsing, one generally avoids this by tabling so-called active edges, which record the subspaces on which a search has already been initiated. This works well because the only existentially quantified variables in the tabled entry are the interior nodes in the span which demarcate where one right-hand-side category ends and another adjacent one begins. To indicate that one is attempting to complete the rule, $S \rightarrow NP\ VP$, for example, one must only table the search from i to j for some k, such that NP is derivable from i to k and VP is derivable from k to j. Our first contribution can be thought of as a generalization of these active edges to the case of bit vectors.

2 FWO Parsing as Search within a Powerset Lattice

A standard chart-parser views constituents as extending over *spans*, contiguous intervals of a linear string. In FWO parsing, constituents partition the input into not necessarily contiguous subsequences, which can be thought of as bit vectors whose AND is 0 and whose OR is $2^n - 1$, given an initial n-length input string. For readability, and to avoid making an arbitrary choice as to whether the leftmost word should correspond to the most significant or least significant bit, we will refer to these constituents as subsets of $\{1 \ldots n\}$ rather than as n-length bit vectors. For simplicity and because of our heightened awareness of the importance of goal-directedness to FWO parsing (see the discussion in the previous section), we will only outline the strictly top-down variant of our strategy, although natural analogues do exist for the other orientations.

2.1 State

State is: $\langle N, \mathrm{CanBV}, \mathrm{ReqBV} \rangle$.
The returned result is: UsedBV **or failure.**

[1] In the case of the lexicon and rule system, there is a very long-standing tradition in logic programming of using this convention. To our knowledge, the first to apply it to the order of RHS categories, which only makes sense once one drops the implicit linear ordering implied by the RHSs of context-free grammar rules, was Daniels and Meurers (2002).

Following Penn and Haji-Abdolhosseini (2003), we can characterize a search state under these assumptions using one non-terminal, N, and two subsets/bit vectors, the *CanBV* and *ReqBV*.[2] CanBV is the set of all words that *can* be used to build an N, and ReqBV is the set of all words that *must* be used while building the N. CanBV always contains ReqBV, and what it additionally contains are optional words that may or may not be used. If search from this state is successful, i.e., N is found using ReqBV and nothing that is not in CanBV, then it returns a *UsedBV*, the subset of words that were actually used. We will assume here that our FWO grammars are not so free that one word can be used in the derivation of two or more sibling constituents, although there is clearly a generalization to this case.

2.2 Process

`Search`($\langle N, C, R\rangle$) can then be defined in the constraint solver as follows:

2.2.1 Initialization

A top-down parse of an n-length string begins with the state consisting of the distinguished category, S, of the grammar, and $\mathrm{CanBV} = \mathrm{ReqBV} = \{1 \ldots n\}$.

2.2.2 Active Edge Subsumption

The first step is to check the current state against states that have already been considered. For expository reasons, this will be presented below. Let us assume for now that this step always fails to produce a matching edge. We must then predict using the rules of the FWO grammar.

2.2.3 Initial Prediction

$\langle N, C, R\rangle \Longrightarrow \langle N_1, C, \phi\rangle$, **where:**

1. $N_0 \rightarrow N_1 \ldots N_k$,
2. $k > 1$, **and**
3. $N \sqcup N_0{\downarrow}$.

As outlined in Penn and Haji-Abdolhosseini (2003), the predictive step from a state consisting of $\langle N, C, R\rangle$ using an immediate dominance rule, $N_0 \rightarrow N_1 \ldots N_k$, with $k > 1$ and no linear precedence constraints transits to a state $\langle N_1, C, \phi\rangle$ provided that N is compatible with N_0. In the case of a classical set of atomic non-terminals, compatibility should be interpreted as equality. In the case of Prolog terms, as in definite clause grammars, or typed feature structures, as in head-driven phrase structure grammar, compatibility can be interpreted as either unifiability or the asymmetric subsumption of N by N_0. Without loss of generality, we will assume unifiability here.

This initial predictive step says that there are, in general, no restrictions on which word must be consumed ($ReqBV = \phi$). Depending on the language chosen for expressing linear precedence restrictions, this set may be non-empty, and in fact, the definition of state used here may need to be generalized to something more complicated than a single set to express the required consumption constraints.

2.2.4 Subsequent Prediction

$\langle N, C, R\rangle \Longrightarrow \langle N_{j+1}, C_j, \phi\rangle$, **where:**

1. $N_0 \rightarrow N_1 \ldots N_k$,
2. $N \sqcup N_0{\downarrow}$,
3. $\langle N_1, C, \phi\rangle$ **succeeded with** U_1,
   $\vdots$
   $\langle N_j, C_{j-1}, \phi\rangle$ **succeeded with** U_j,
4. $k > 1$ and $1 \leq j < k - 1$, **and**
5. $C_j = C \cap \overline{U}_1 \cap \ldots \cap \overline{U}_j$.

Regardless of these generalizations, however, each subsequent predictive step, having recognized $N_1 \ldots N_j$, for $1 \leq j < k - 1$, computes the next CanBV C_j by removing the consumed words U_j from the previous CanBV C_{j-1}, and then transits to state $\langle N_{j+1}, C_j, \phi\rangle$. Removing the UsedBVs is the result of our assumption that no word can be used by two or more sibling constituents.

2.2.5 Completion

$\langle N, C, R\rangle \Longrightarrow \langle N_k, C_{k-1}, R_{k-1}\rangle$, **where:**

1. $N_0 \rightarrow N_1 \ldots N_k$,
2. $N \sqcup N_0{\downarrow}$,
3. $\langle N_1, C, \phi\rangle$ **succeeded with** U_1,
   $\vdots$
   $\langle N_{k-1}, C_{k-2}, \phi\rangle$ **succeeded with** U_{k-1},
4. $C_{k-1} = C \cap \overline{U}_1 \cap \ldots \cap \overline{U}_{k-1}$, **and**
5. $R_{k-1} = R \cap \overline{U}_1 \cap \ldots \cap \overline{U}_{k-1}$.

The completion step then involves recognizing the last RHS category (although this is no longer rightmost in terms of linear precedence). Here, the major difference from subsequent prediction is that there is now a potentially non-empty ReqBV. Only with the last RHS category are we actually in a position to enforce R from the source state.

If $\langle N_k, C_{k-1}, R_{k-1}\rangle$ succeeds with U_k, then $\langle N, C, R\rangle$ succeeds with $U_1 \cup \ldots \cup U_k$.

[2] Actually, Penn and Haji-Abdolhosseini (2003) use CanBV and *OptBV*, which can be defined as $\mathrm{CanBV} \cap \overline{\mathrm{ReqBV}}$.

2.3 Active Edge Subsumption Revisited

So far, this is very similar to the strategy outlined in Penn and Haji-Abdolhosseini (2003). If we were to add active edges in a manner similar to standard chart parsing, we would tabulate states like $\langle N_a, C_a, R_a \rangle$ and then compare them in step 2.2.2 to current states $\langle N, C, R \rangle$ by determining whether (classically) $N = N_a$, $C = C_a$, and $R = R_a$. This might catch some redundant search, but just as we can do better in the case of non-atomic categories by checking for subsumption ($N_a \sqsubseteq N$) or unifiability ($N \sqcup N_a{\downarrow}$), we can do better on C and R as well because these are sets that come with a natural notion of containment.

Figure 1 shows an example of how this containment can be used. Rather than comparing edges annotated with linear subspans, as in the case of CFG chart parsing, here we are comparing edges annotated with sublattices of the powerset lattice on n elements, each of which has a top element (its CanBV) and a bottom element (its ReqBV). Everything in between this top and bottom is a subset of words that has been (or will be) tried if that combination has been tabled as an active edge.

Figure 1 assumes that $n = 6$, and that we have tabled an active edge (dashed lines) with $C_a = \{1, 2, 4, 5, 6\}$, and $R_a = \{1, 2\}$. Now suppose later that we decide to search for the same category in $C = \{1, 2, 3, 4, 5, 6\}$, $R = \{1, 2\}$ (dotted lines). Here, $C \neq C_a$, so an equality-based comparison would fail, but a better strategy would be to reallocate the one extra bit in C (3) to R, and then search $C' = \{1, 2, 3, 4, 5, 6\}$, $R' = \{1, 2, 3\}$ (solid lines). As shown in Figure 1, this solid region fills in all and only the region left unsearched by the active edge.

This is actually just one of five possible cases that can arise during the comparison. The complete algorithm is given in Figure 2. This algorithm works as a filter, which either blocks the current state from further exploration, allows it to be further explored, or breaks it into several other states that can be concurrently explored. Step 1(a) deals with category unifiability. If the current category, N, is unifiable with the tabled active category, N_a, then 1(a) breaks N into more specific pieces that are either incompatible with N_a or subsumed by N_a. By the time we get to 1(b), we know we are dealing with a piece that is subsumed by N_a. O stands for "optional," CanBV bits that are not required.

`Check`($\langle N, C, R \rangle$):

- For each active edge, a, with $\langle N_a, C_a, R_a \rangle$,
  1. If $N \sqcup N_a{\downarrow}$, then:
     (a) For each minimal category N' such that $N \sqsubseteq N'$ and $N' \sqcup N_a{\uparrow}$, concurrently:
     – Let $N := N'$, and continue [to next active edge].
     (b) Let $N := N \sqcup N_a$, $O := C \cap \overline{R}$ and $O_a := C_a \cap \overline{R_a}$.
     (c) If $C_a \cap \overline{O_a} \cap \overline{C} \neq \phi$, then continue [to next active edge].
     (d) If $C \cap \overline{O} \cap \overline{C_a} \neq \phi$, then continue [to next active edge].
     (e) If $(Z :=) O \cap \overline{C_a} \neq \phi$, then:
     i. Let $O := O \cap \overline{Z}$,
     ii. Concurrently:
     A. continue [to next active edge], and
     B. (1) Let $C := C \cap \overline{Z}$,
     (2) goto (1) [to reconsider this active edge].
     (f) If $(Z :=) C_a \cap \overline{O_a} \cap O \neq \phi$, then:
     i. Let $O := O \cap \overline{Z}$, $C := C \cap \overline{Z}$,
     ii. continue [to next active edge].
     (g) Fail — this state is subsumed by an active edge.
  2. else continue [to next active edge].

Figure 2: Active edge checking algorithm.

Only one of 1(g) or the bodies of 1(c), 1(d), 1(e) or 1(f) is ever executed in a single pass through the loop. These are the five cases that can arise during subset/bit vector comparison, and they must be tried in the order given. Viewing the current state's CanBV and ReqBV as a modification of the active edge's, the first four cases correspond to: the removal of required words (1(c)), the addition of required words (1(d)), the addition of optional (non-required) words (1(e)), and the reallocation of required words to optional words (1(f)). Unless one of these four cases has happened, the current sublattice has already been searched in its entirety (1(g)).

2.4 Linear Precedence Constraints

The elaboration above has assumed the absence of any linear precedence constraints. This is the

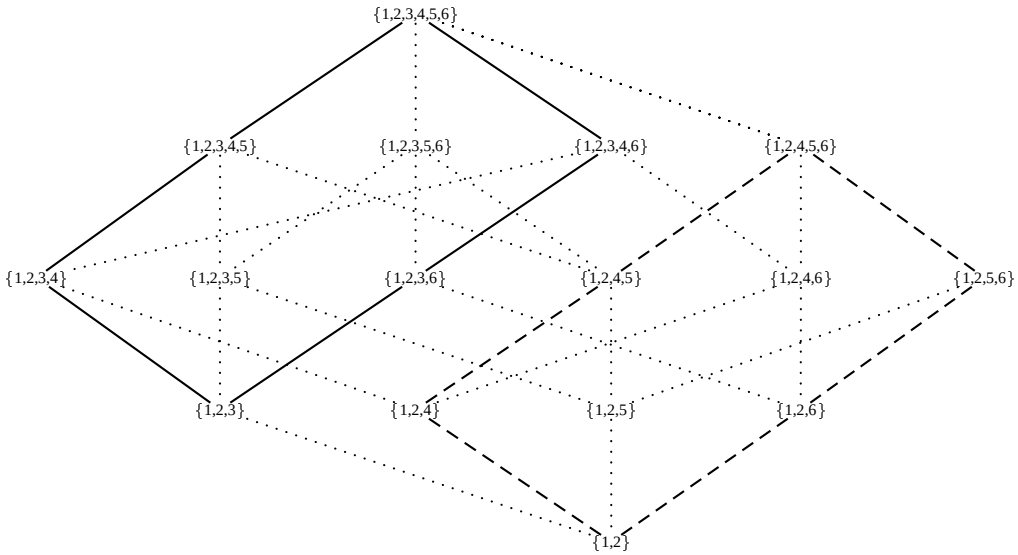

Figure 1: A powerset lattice representation of active edge checking with CanBV and ReqBV.

worst case, from a complexity perspective. The propagation rules of section 2.2 can remain unchanged in a concurrent constraint-based framework in which other linear precedence constraints observe the resulting algebraic closure and fail when violated, but it is possible to integrate these into the propagators for efficiency. In either case, the active edge subsumption procedure remains unchanged.

For lack of space, we do not consider the characterization of linear precedence constraints in terms of CanBV and ReqBV further here.

3 Category Graphs and Iteratively Computed Yields

Whereas in the last section we trivialized linear precedence, the constraints of this section simply do not use them. Given a FWO grammar, G, with immediate dominance rules, R, over a set of non-terminals, N, we define the *category graph* of G to be the smallest directed bipartite graph, $C(G) = \langle V, E \rangle$, such that:

- $V = N \cup R \cup \{\mathrm{Lex}, \mathrm{Empty}\}$,
- $(X, r) \in E$ if non-terminal X appears on the RHS of rule r,
- $(r, X) \in E$ if the LHS non-terminal of r is X,
- $(\mathrm{Lex}, r) \in E$ if there is a terminal on the RHS of rule r, and
- $(\mathrm{Empty}, r) \in E$ if r is an empty production rule.

We will call the vertices of $C(G)$ either *category nodes* or *rule nodes*. Lex and Empty are considered category nodes. The category graph of the grammar in Figure 3, for example, is shown in

S $\rightarrow$ VP NP	$\mathrm{VP}_1 \rightarrow$ V NP
$\mathrm{NP}_1 \rightarrow$ N' S	$\mathrm{VP}_2 \rightarrow$ V
$\mathrm{NP}_2 \rightarrow$ N'	N $\rightarrow \{$boy, girl$\}$
$\mathrm{N'}_1 \rightarrow$ N Det	Det $\rightarrow \{$a, the, this$\}$
$\mathrm{N'}_2 \rightarrow$ N	V $\rightarrow \{$sees, calls$\}$

Figure 3: A sample CFG-like grammar.

Figure 4. By convention, we draw category nodes with circles, and rule nodes with boxes, and we label rule nodes by the LHS categories of the rules they correspond to plus an index. For brevity, we will assume a normal form for our grammars here, in which the RHS of every rule is either a string of non-terminals or a single terminal.

Category graphs are a minor variation of the "grammar graphs" of Moencke and Wilhelm (1982), but we will use them for a very different purpose. For brevity, we will consider only atomic non-terminals in the remainder of this section. Category graphs can be constructed for partially ordered sets of non-terminals, but in this case, they can only be used to approximate the values of the functions that they exactly compute in the atomic case.

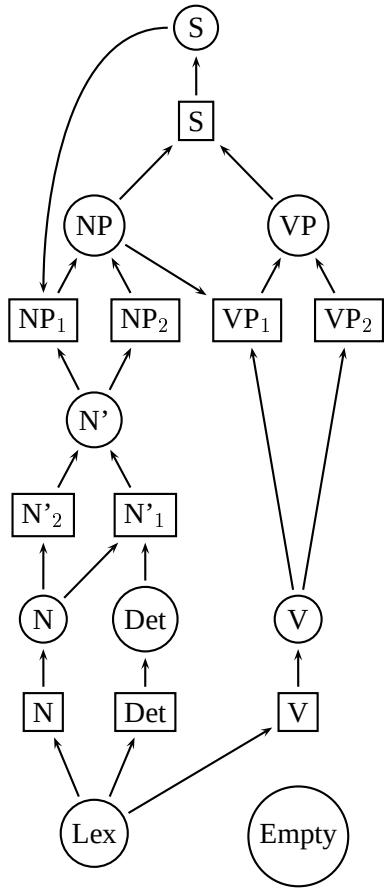

Figure 4: The category graph for the grammar in Figure 3.

Restricting search to unexplored sublattices helps us with recursion in a grammar in that it stops redundant search, but in some cases, recursion can be additionally bounded (above and below) not because it is redundant but because it cannot possibly yield a string as short or long as the current input string. Inputs are unbounded in size across parses, but within a single parse, the input is fixed to a constant size. Category graphs can be used to calculate bounds as a function of this size. We will refer below to the length of an input string below a particular non-terminal in a parse tree as the *yield* of that non-terminal instance. The *height* of a non-terminal instance in a parse tree is 1 if it is pre-terminal, and 1 plus the maximum height of any of its daughter non-terminals otherwise. Non-terminal categories can have a range of possible yields and heights.

3.1 Parse Tree Height

Given a non-terminal, X, let $X^{max}(h)$ be the maximum yield that a non-terminal instance of X at height h in any parse tree can produce, given the fixed grammar G. Likewise, let $X^{min}(h)$ be the minimum yield that such an instance must produce. Also, as an abuse of functional notation, let:

$$X^{max}(\leq h) = \max_{0 \leq j \leq h} X^{max}(j)$$
$$X^{min}(\leq h) = \min_{0 \leq j \leq h} X^{min}(j)$$

Now, using these, we can come back and define $X^{max}(h)$ and $X^{min}(h)$:

$$\mathrm{Lex}^{max}(h) =$$
$$\mathrm{Lex}^{min}(h) = \begin{cases} 1 & h = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$
$$\mathrm{Empty}^{max}(h) =$$
$$\mathrm{Empty}^{min}(h) = \begin{cases} 0 & h = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and for all other category nodes, X:

$$\mathrm{X}^{max}(1) =$$
$$\mathrm{X}^{min}(1) = \begin{cases} 0 & X \rightarrow \epsilon \in R \\ 1 & X \rightarrow t \in R \\ \text{undefined} & \text{otherwise} \end{cases}$$

and for $h > 1$:

$$X^{max}(h) = \max_{X \rightarrow X_1 \ldots X_k \in R} \left(\max_{1 \leq i \leq k} X_i^{max}(h-1) + \sum_{j=1, j \neq i}^{k} X_j^{max}(\leq h-1) \right)$$
$$X^{min}(h) = \min_{X \rightarrow X_1 \ldots X_k \in R} \left(\min_{1 \leq i \leq k} X_i^{min}(h-1) + \sum_{j=1, j \neq i}^{k} X_j^{min}(\leq h-1) \right).$$

For example, in Figure 3, there is only one rule with S as a LHS category, so:

$$\mathrm{S}^{max}(h) = \max \begin{cases} \mathrm{NP}^{max}(h-1) + \mathrm{VP}^{max}(\leq h-1) \\ \mathrm{NP}^{max}(\leq h-1) + \mathrm{VP}^{max}(h-1) \end{cases}$$
$$\mathrm{S}^{min}(h) = \min \begin{cases} \mathrm{NP}^{min}(h-1) + \mathrm{VP}^{min}(\leq h-1) \\ \mathrm{NP}^{min}(\leq h-1) + \mathrm{VP}^{min}(h-1). \end{cases}$$

These functions compute yields as a function of height. We know the yield, however, and want bounds on height. Given a grammar in which the non-pre-terminal rules have a constant branching factor, we also know that $X^{max}(h)$ and $X^{min}(h)$, are monotonically non-decreasing in h, where they are defined. This means that we can iteratively compute $X^{max}(h)$, for all non-terminals X, and all values h out to the first h' that produces a value strictly greater than the current yield (the length of the given input). Similarly, we can compute $X^{min}(h)$, for all non-terminals X, and

all values h out to the first h'' that is equal to or greater than the current yield. The height of the resulting parse tree, h, can then be bounded as $h' - 1 \leq h \leq h''$. These iterative computations can be cached and reused across different inputs. In general, in the absence of a constant branching factor, we still have a finite maximum branching factor, from which an upper bound on any potential decrease in $X^{max}(h)$ and $X^{min}(h)$ can be determined.

This provides an interval constraint. Because there may be heights for which $X^{max}(h)$ and $X^{min}(h)$ is not defined, one could, with small enough intervals, additionally define a finite domain constraint that excludes these.

These recursive definitions are well-founded when there is at least one finite string derivable by every non-terminal in the grammar. The X^{min} functions converge in the presence of unit production cycles in $C(G)$; the X^{max} functions can also converge in this case. Convergence restricts our ability to constrain search with yields.

A proper empirical test of the efficacy of these constraints requires large-scale phrase structure grammars with weakened word-order constraints, which are very difficult to come by. On the other hand, our preliminary experiments with simple top-down parsing on the Penn Treebank II suggest that even in the case of classical context-free grammars, yield constraints can improve the efficiency of parsing. The latency of constraint enforcement has proven to be a real issue in this case (weaker bounds that are faster to enforce can produce better results), but the fact that yield constraints produce any benefit whatsoever with CFGs is very promising, since the search space is so much smaller than in the FWO case, and edge indexing is so much easier.

3.2 Cycle Variables

The heights of non-terminals from whose category nodes the cycles of $C(G)$ are not path-accessible can easily be bounded. Using the above height-dependent yield equations, the heights of the other non-terminals can also be bounded, because any input string fixes the yield to a finite value, and thus the height to a finite range (in the absence of converging X^{min} sequences). But we can do better. We can condition these bounds not only upon height but upon the individual rules used. We could even make them depend upon sequences of rules, or on vertical chains of non-terminals within trees. If $C(G)$ contains cycles, however, there are infinitely many such chains (although finitely many of any given length), but trips around cycles themselves can also be counted.

Let us formally specify that a *cycle* refers to a unique path from some category node to itself, such that every node along the path except the last is unique. Note that because $C(G)$ is bipartite, paths alternate between category nodes and rule nodes.

Now we can enumerate the distinct cycles of any category graph. In Figure 4, there are two, both passing through NP and S, with one passing through VP in addition. Note that cycles, even though they are unique, may share nodes as these two do. For each cycle, we will arbitrarily choose an *index node* for it, and call the unique edge along the cycle leading into that node its *index link*. It will be convenient to choose the distinguished non-terminal, S, as the index node when it appears in a cycle, and in other cases, to choose a node with a minimal path-distance to S in the category graph.

For each cycle, we will also assign it a unique *cycle variable* (written n, m etc.). The domain of this variable is the natural numbers and it counts the number of times in a parse that we traverse this cycle as we search top-down for a tree. When an index link is traversed, the corresponding cycle variable must be incremented.

For each category node X in $C(G)$, we can define the maximum and minimum yield as before, but now instead of height being the only independent parameter, we also make these functions depend on the cycle variables of all of the cycles that pass through X. If X has no cycles passing through it, then its only parameter is still h. We can also easily extend the definition of these functions to rule nodes.

Rather than provide the general definitions here, we simply give some of the equations for Figure 4,

for shortage of space:

$$\mathrm{S}^{max}(h,n,m) = \mathrm{S}^{max}(h,\bar{n},\bar{m})$$
$$\mathrm{S}^{max}(h,n,\bar{m}) = \mathrm{S}^{max}(h,\bar{n},\bar{m})$$
$$\mathrm{S}^{max}(h,\bar{n},\bar{m}) =$$
$$\max_{\substack{i+j=n,\\ k+l=m}} \begin{cases} \mathrm{NP}^{max}(h-1,\bar{i},k) \\ \quad +\mathrm{VP}^{max}(\leq h-1,j,\bar{l}) \\ \mathrm{NP}^{max}(\leq h-1,\bar{i},k) \\ \quad +\mathrm{VP}^{max}(h-1,j,\bar{l}) \end{cases}$$
$$\mathrm{NP}^{max}(h,\bar{n},\bar{m}) = \max \begin{cases} \mathrm{NP}_1^{max}(h,\bar{n},\bar{m}) \\ \mathrm{NP}_2^{max}(h,n,m) \end{cases}$$
$$\mathrm{NP}_1^{max}(h,\bar{n},\bar{m}) =$$
$$\max \begin{cases} \mathrm{N'}^{max}(h-1) \\ \quad +\mathrm{S}^{max}(\leq h-1,\overline{n-1},\overline{m-1}) \\ \mathrm{N'}^{max}(\leq h-1) \\ \quad +\mathrm{S}^{max}(h-1,\overline{n-1},\overline{m-1}) \end{cases}$$
$$\mathrm{NP}_1^{max}(h,n,\bar{m}) =$$
$$\max \begin{cases} \mathrm{N'}^{max}(h-1) \\ \quad +\mathrm{S}^{max}(\leq h-1,n,\overline{m-1}) \\ \mathrm{N'}^{max}(\leq h-1) \\ \quad +\mathrm{S}^{max}(h-1,n,\overline{m-1}) \end{cases}$$
$$\mathrm{NP}_1^{max}(h,\bar{n},m) =$$
$$\max \begin{cases} \mathrm{N'}^{max}(h-1) \\ \quad +\mathrm{S}^{max}(\leq h-1,\overline{n-1},m) \\ \mathrm{N'}^{max}(\leq h-1) \\ \quad +\mathrm{S}^{max}(h-1,\overline{n-1},m) \end{cases}$$
$$\mathrm{NP}_2^{max}(h,n,m) = \begin{cases} \mathrm{N'}^{max}(h-1) & n=m=0 \\ \text{undefined} & o.w. \end{cases}$$
$$\mathrm{VP}_1^{max}(h,n,\bar{m}) =$$
$$\max \begin{cases} \mathrm{V}^{max}(h-1) \\ \quad +\mathrm{NP}^{max}(\leq h-1,n,\overline{m-1}) \\ \mathrm{V}^{max}(\leq h-1) \\ \quad +\mathrm{NP}^{max}(h-1,n,\overline{m-1}) \end{cases}$$

We think of functions in which overscores are written over some parameters as entirely different functions that have witnessed partial traversals through the cycles corresponding to the overscored parameters, beginning at the respective index nodes of those cycles.

Cycle variables are a local measure of nonterminal instances in that they do not depend on the absolute height of the tree — only on a fixed range of nodes above and below them in the tree. These makes them more suitable for the iterative computation of yields that we are interested in. Because X^{max} and X^{min} are now multivariate functions in general, we must tabulate an entire table out to some bound in each dimension, from which we obtain an entire frontier of acceptable values for the height and each cycle variable. Again, these can be posed either as interval constraints or finite domain constraints.

In the case of grammars over atomic categories, using a single cycle variable for every distinct cycle is generally not an option. The grammar induced from the local trees of the 35-sentence section wsj_0105 of the Penn Treebank II, for example, has 49 non-terminals and 258 rules, with 153,026 cycles. Grouping together cycles that differ only in their rule nodes, we are left with 204 groupings, and in fact, they pass through only 12 category nodes. Yet the category node with the largest number of incident cycles (NP) would still require 163 cycle (grouping) variables — too many to iteratively compute these functions efficiently. Naturally, it would be possible to conflate more cycles to obtain cruder but more efficient bounds.

References

G. E. Barton, R. C. Berwick, and E. S. Ristad. 1987. *Computational Complexity and Natural Language*. MIT Press.

M. Daniels and W. D. Meurers. 2002. Improving the efficiency of parsing with discontinuous constituents. In *7th International Workshop on Natural Language Understanding and Logic Programming (NLULP)*.

M. Johnson. 1985. Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pages 127–132.

U. Moencke and R. Wilhelm. 1982. Iterative algorithms on grammar graphs. In H. J. Schneider and H. Goettler, editors, *Proceedings of the 8th Conference on Graphtheoretic Concepts in Computer Science (WG 82)*, pages 177–194. Carl Hanser Verlag.

F. Morawietz. 2000. Chart parsing and constraint programming. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-00)*, volume 1, pages 551–557.

G. Penn and M. Haji-Abdolhosseini. 2003. Topological parsing. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 283–290.

F. C. N. Pereira and D. H. D. Warren. 1983. Parsing as deduction. In *Proceedings of 21st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 137–144.

M. Reape. 1991. Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In M. Reape, editor, *Word Order in Germanic and Parsing*, pages 41–70. Centre for Cognitive Science, University of Edinburgh.