An Efficient Algorithm to Induce Minimum Average Lookahead Grammars for Incremental LR Parsing

Dekai WU[1] **Yihai SHEN**
dekai@cs.ust.hk shenyh@cs.ust.hk

Human Language Technology Center
HKUST
Department of Computer Science
University of Science and Technology, Clear Water Bay, Hong Kong

Abstract

We define a new learning task, minimum average lookahead grammar induction, with strong potential implications for incremental parsing in NLP and cognitive models. Our thesis is that a suitable learning bias for grammar induction is to minimize the degree of lookahead required, on the underlying tenet that language evolution drove grammars to be efficiently parsable in incremental fashion. The input to the task is an unannotated corpus, plus a nondeterministic *constraining grammar* that serves as an abstract model of environmental constraints confirming or rejecting potential parses. The constraining grammar typically allows ambiguity and is itself poorly suited for an incremental parsing model, since it gives rise to a high degree of nondeterminism in parsing. The learning task, then, is to induce a deterministic $\mathrm{LR}(k)$ grammar under which it is possible to incrementally construct one of the correct parses for each sentence in the corpus, such that the average degree of lookahead needed to do so is minimized. This is a significantly more difficult optimization problem than merely compiling $\mathrm{LR}(k)$ grammars, since k is not specified in advance. Clearly, naïve approaches to this optimization can easily be computationally infeasible. However, by making combined use of GLR ancestor tables and incremental LR table construction methods, we obtain an $\mathrm{O}(n^3 + 2^m)$ greedy approximation algorithm for this task that is quite efficient in practice.

1 Introduction

Marcus' (1980) Determinism Hypothesis proposed that natural language can be parsed by a mechanism that operates "strictly deterministically" in that it does not simulate a nondeterministic machine. Although the structural details of the deterministic LR parsing model we employ in this paper diverge from those of Marcus, fundamentally we adhere to his constraints that (1) *all syntactic substructures created are permanent*, which prohibits simulating determinism by backtracking, (2) *all syntactic substructures created for a given input must be part of the output structure assigned to that input*, which prohibits memoizing intermediate results as in dynamic programming or beam search, and (3) *no temporary syntactic structures are encoded within the internal state of the machine*, which prohibits the moving of temporary structures into procedural codes.

A key issue is that, to give the Determinism Hypothesis teeth, it is necessary to limit the size of the decision window. Otherwise, it is always possible to circumvent the constraints simply by increasing the degree of lookahead or, equivalently, increasing the buffer size (which we might call the degree of "look-behind"); either way, increasing the decision window essentially delays decisions until enough disambiguating information is seen. In the limit, a decision window equal to the sentence length renders the claim of incremental parsing meaningless. Marcus simply postulated that a maximum buffer size of three was sufficient. In contrast, our approach permits greater flexibility and finer gradations, where the *average* degree of lookahead required can be *minimized* with the aim of assisting grammar induction.

Since Marcus' work, a significant body of work on incremental parsing has developed in the sentence processing community, but much of this work has actually suggested models with an *increased* amount of nondeterminism, often with probabilistic weights (e.g., Narayanan & Jurafsky (1998); Hale (2001)).

Meanwhile, in the way of formal methods, Tomita (1986) introduced Generalized LR parsing, which offers an interesting hybrid of nondeterministic dynamic programming surrounding LR parsing methods that were originally deterministic.

Additionally, methods for determinizing and

[1]The author would like to thank the Hong Kong Research Grants Council (RGC) for supporting this research in part through research grants RGC6083/99E, RGC6256/00E, and DAG03/04.EG09.

minimizing finite-state machines are well known (e.g., Mohri (2000), Béal & Carton (1968)). However, such methods (a) do not operate at the context-free level, (b) do not directly minimize lookahead, and (c) do not induce grammars under environmental constraints.

Unfortunately, there has still been relatively little work on automatic learning of grammars for deterministic parsers to date. Hermjakob & Mooney (1997) describe a semi-automatic procedure for learning a deterministic parser from a treebank, which requires the intervention of a human expert in the loop to determine appropriate derivation order, to resolve parsing conflicts between certain actions such as "merge" and "add-into", and to identify specific features for disambiguating actions. In our earlier work we described a deterministic parser with a fully automatically learned decision algorithm (Wong and Wu, 1999). But unlike our present work, the decision algorithms in both Hermjakob & Mooney (1997) and Wong & Wu (1999) are procedural; there is no explicit representation of the grammar that can be meaningfully inspected.

Finally, we observe that there are also trainable stochastic shift-reduce parser models (Briscoe and Carroll, 1993), which are theoretically related to shift-reduce parsing, but operate in a highly nondeterministic fashion during parsing.

We believe the shortage of learning models for deterministic parsing is in no small part due to the difficulty of overcoming computational complexity barriers in the optimization problems this would involve. Many types of factors need to be optimized in learning, because deterministic parsing is much more sensitive to incorrect choice of structural features (e.g., categories, rules) than nondeterministic parsing that employ robustness mechanisms such as weighted charts.

Consequently, we suggest shifting attention to the development of new methods that directly address the problem of optimizing criteria associated with deterministic parsing, in computationally feasible ways. In particular, we aim in this paper to develop a method that efficiently searches for a parser under a minimum average lookahead cost function.

It should be emphasized that we view the role of a deterministic parser as one component in a larger model. A deterministic parsing stage can be expected to handle most input sentences, but not all. Other nondeterministic mechanisms will clearly be needed to handle a minority of cases, the most obvious being garden-path sentences.

In the sections that follow, we first formalize the learning problem. We then describe an efficient approximate algorithm for this task. The operation of this algorithm is illustrated with an example. Finally, we give an analysis of the complexity characteristics of the algorithm.

2 The minimum average lookahead (MAL) grammar problem

We formulate the learning task as follows:

Definition Given an unannotated corpus $S = \{S_1, \ldots, S_{|S|}\}$ plus a constraining grammar G_C, the *minimum average lookahead* grammar $G_{\text{MAL}}(S, G_C)$ is defined as

$$\arg\min_{G \subset G_C \wedge \forall i: \text{parse}_G(S_i) \neq \emptyset} \hat{k}\,(G)$$

where the *average lookahead* objective function $\hat{k}\,(G)$ is the average (over S) amount of lookahead that an LR parser for G needs in order to deterministically parse the sample S without any shift-reduce or reduce-reduce conflicts. If G is ambiguous in the sense that it generates more than one parse for any sentence in S, then $\hat{k}\,(G) = \infty$ since a conflict is unavoidable no matter how much lookahead is used.

In other words, G_{MAL} is the subset of rules of G_C that requires the smallest number of lookaheads on average so as to make parsing S using this subset of G deterministic.

Note that the *constraining grammar* G_C by nature is not deterministic. The constraining grammar serves merely as an abstract model of environmental constraints that confirm or reject potential parses. Since such feedback is typically quite permissive, the constraining grammar typically allows a great deal of ambiguity. This of course renders the constraining grammar itself poorly suited for an incremental parsing model, since it gives rise to a high degree of nondeterminism in parsing. In other words, we should not expect the constraining grammar alone to contain sufficient information to choose a deterministically parsable grammar.

For expository simplicity we assume all grammars are in standard context-free form in the discussion that follows, although numerous notational variations, generalizations, and restricted cases are certainly possible. We note also that, although the formalization is couched in terms of standard syntactic phrase structures, there is no reason why one could not employ categories and/or attributes on parse nodes representing *semantic* features. Doing so would permit the framework to accommodate some semantic information in minimizing lookahead for deterministic parsing, which would be more realistic from a cognitive modeling standpoint. (Of course, further extensions to integrate

more complex incremental semantic interpretation mechanisms into this framework could be explored as well.)

Finding the minimum average lookahead grammar is clearly a difficult optimization problem. To compute the value of $\hat{k}(G)$, one needs the LR table for that particular G, which is expensive to compute. Computing the LR table for all $G \subset G_C$ would be infeasible. It is a natural conjecture, in fact, that the problem of learning MAL grammars is NP-hard. We therefore seek an approximation algorithm with good performance, as discussed next.

3 An efficient approximate algorithm for learning incremental MAL parsers

We now describe an approximate method for efficiently learning a MAL grammar. During learning, the MAL grammar is represented simultaneously as both a set of standard production rules as well as an LR parsing table. Thus the learning algorithm outputs an explicit declarative rule set together with a corresponding compiled LR table.

3.1 Approximating assumptions

To overcome the obstacles mentioned in the previous section, we make the following approximations:

1. *Incremental approximation for MAL rule set computation*. We assume that the MAL grammar for a given corpus is approximately equal to the sequential union of the MAL grammar rules for each sentence in the corpus, where the set of MAL grammar rules for each sentence is determined relative to the set of all rules selected from preceding sentences in the corpus.

2. *Incremental approximation for LR state set computation*. We assume that the correct set of LR states for a given set of rules is approximately equal to that obtained by incrementally modifying the LR table and states from a slightly smaller subset of the rules. (Our approach exploits the fact that the correct set of states for LR (k) parsers is always independent of k.)

Combining these approximation assumptions enables us to utilize a sentence-by-sentence greedy approach to seeking a MAL grammar. Specifically, the algorithm iteratively computes a minimum average lookahead set of rules for each sentence in the training corpus, accumulating all rules found, while keeping the LR state set and table updated. The full algorithm is fairly complex; we discuss its key aspects here.

3.2 Structure of the iteration

As shown in Figure 1, **find_MAL_parser** accepts as input an unannotated corpus $S = \{S_1, \ldots, S_{|S|}\}$ plus a constraining grammar G_C , and outputs the LR table for a parser that can deterministically parse the entire training corpus using a minimum average lookahead.

The algorithm consists of an initialization step followed by an iterative loop. In the initialization step in lines 1–3, we create an empty LR table T, along with an empty set A of parsing action sequences defined as follows. A *parsing action sequence* $A(P)$ for a given parse P is the sequence of triples (state, input, action) that a shift-reduce parser follows in order to construct P. At any given point, T will hold the LR table computed from the MAL parse of all sentences already processed, and A will hold the corresponding parsing sequences for the MAL parses.

Entering the loop, we iteratively augment *T* by adding the states arising from the MAL parse F^* of each successive sentence in the training corpus and, in addition, cache the corresponding parsing action sequence $A(F^*)$ into the set A. This is done by first computing a chart for the sentence, in line 4, by parsing S_i under the constraining grammar G_C using the standard Earley (1970) procedure. We then call find_MAL_parse in line 5, to compute the parse that requires minimum average lookahead to resolve ambiguity. The items and states produced by the rules in $F^*$are added to the LR table *T* by calling incremental_update_LR in line 6, and the parsing action sequence of F^* is appended to A in line 7. Note that the indices of the original states in *T* are not changed and only items are added into them if need be so that *A* is not changed by adding items and states to *T*, and there might be new states introduced which are also indexed.

By definition, the true MAL grammar does not depend on the order of the sentences the learning algorithm inspects. However, find_MAL_parser processes the example sentences in order, and attempts to find the MAL grammar sentence by sentence. The order of the sentences impacts the grammar produced by the learning algorithm, so it is not guaranteed to find the true MAL grammar. However the approximation is well motivated particularly when we have large numbers of example sentences.

3.3 Incrementally updating the LR table

Given the structure of the loop, it can be seen that efficient learning of the set of MAL rules cannot be achieved without a component that can update the LR table incrementally as each rule is added into the

find_MAL_parser(S, G_C)

1. $T \leftarrow \emptyset$
2. $A \leftarrow \emptyset$
3. $i \leftarrow 0$
4. $C \leftarrow$ chart_parse(S_i, G_C)
5. $F^* \leftarrow$ find_MAL_parse(C, A, R)
6. $T \leftarrow$ incremental_update_LR(T, F^*)
7. append$(A, A(F^*))$
8. **if** $i < |S|$ **then** $i \leftarrow i + 1$; **goto** 4

Figure 1: Main algorithm find_MAL_parser.

current MAL grammar. Otherwise, every time a rule is found to be capable of reducing average lookahead and therefore is added into the MAL grammar, the LR table must be recomputed from scratch, which is sufficiently time consuming as to be infeasible when trying to learn a MAL grammar with a realistically large input corpus and/or constraining grammar.

The **incremental_update_LR** function incrementally updates the LR table in an efficient fashion that avoids recomputing the entire table. The inputs to incremental_update_LR are a pre-existing LR table T, and a set of new rules R to be added. This algorithm is derived from the incremental LR parser generator algorithm *ilalr* and is relatively complex; see Horspool (1988) for details. Historically, work on incremental parser generators first concentrated on LL(1) parsers. Fischer (1980) was first to describe a method for incrementally updating an LR(1) table. Heering *et al.*(1990) use the principle of lazy evaluation to attack the same problem. Our design of incremental_update_LR is more closely related to *ilalr* for the following reasons:

- *ilalr* has the property that when updating the LR table to contain the newly added rules, it does not change the index of each already existing state. This is important for our task as the slightest change in the states might affect significantly the parsing sequences of the sentences that have already been processed.
- Although worst case complexity for *ilalr* is exponential in the number of rules in the grammar, empirically it is quite efficient in practical use. Heuristics are used to improve the speed of the algorithm, and as we do not need to compute lookahead sets, the speed of the algorithm can be further improved.

compute_average_lookahead(r, A)

1. $h \leftarrow$ lookahead(r, A)
2. **if** $\exists v_1 = (i, s, a_1, k_1, r_1, d_1)$
   - **then** $k = k_1 = (m_1, l_1)$
   - **else** $k = (0, \infty)$
3. // note $v' = (i', s', a', k', r', d')$ and $k' = (m', l')$
4. $l'' = \frac{m'l'+h}{m'+1}$
5. **if** $l'' < l$
   - **then** $l = l''$
   - **else** $m = m' + 1$

Figure 2: Algorithm compute_average_lookahead.

The method is approximate, and may yield slight, acceptable deviations from the optimal table. *ilalr* is not an exact LR table generator in the sense that it may create states that should not exist and may miss some states that should exist. The algorithm is based on the assumption that *most* states in the original LR table occur with the same kernel items in the updated LR table. Empirically, the assumption is valid as the proportion of superfluous states is typically only in the 0.1% to 1.3% range.

3.4 Finding minimum average lookahead parses

The function **find_MAL_parse** selects the full parse $F*$ of a given sentence that requires the least *average lookahead* $\hat{k}\,(A(\,F\,))$ to resolve any shift-reduce or reduce-reduce conflicts with a set A of parsing action sequences, such that $F*$ is a subset of a chart C. The inputs to find_MAL_parse, more specifically, are a chart C containing all the partial parses in the input sentence, and the set A containing the parsing action sequences of the MAL parse of all sentences processed so far. The algorithm operates by constructing a graph-structured stack of the same form as in GLR parsing (Tomita, 1986)(Tomita and Ng, 1991) while simultaneously computing the minimum average lookahead. Note that Tomita's original method for constructing the graph-structured stack has exponential time complexity $O\left(n^{\rho+1}\right)$, in which n and ρ are the length of the sentence and the length of the longest rhs of any rule in the grammar. As a result, Tomita's algorithm achieves $O\left(n^3\right)$ for grammars in Chomsky normal form but is potentially exponential when productions are of unrestricted length, which in practice

is the case with most parsing problems. We follow Kipps (1991) in modifying Tomita's algorithm to allow it to run in time proportional to n^3 for grammars with productions of arbitrary length. The most time consuming part in Tomita's algorithm is when reduce actions are executed in which the ancestors of the current node have to be found incurring time complexity n^ρ. To avoid this we employ an ancestor table to keep the ancestors of each node in the GLR forest which is updated dynamically as the GLR forest is being constructed. This modification brings down the time complexity of reduce actions to n^2 in the worst case, and allows the function **build_GLR_forest** to construct the graph-structured stack in $O\left(n^3\right)$. Aside from constructing the graph-structured stack, we compute the average lookahead for each LR state transition taken during the construction. Whenever there is a shift or reduce action in the algorithm, a new vertex for the graph-structured stack is generated, and the function compute_average_lookahead is called to ascertain the average lookahead of the new vertex. Finally, reconstruct_MAL_parse is called to recover the full parse $F*$ for the MAL parsing action sequence.

Figure 2 shows the **compute_average_lookahead** function, which estimates the average lookahead of a vertex v generated by an LR state transition r. To facilitate computations involving average lookahead, we use a 6-tuple (i, s, a, k, r, d) instead of the more common triple form (i, s, a) to represent each vertex in the graph-structured stack, where:

- i: The index of the right side of the coverage of the vertex. The vertices with the same right side i will be kept in U_i.
- s: The state in which the vertex is in.
- a: The ancestor table of the vertex.
- k: The average lookahead information, in the form of a pair (m, l) where l is the minimum average lookahead of all paths leading from the root to this vertex and m is the number of state transitions in that MAL path.
- r: The parsing action that generates the vertex along the path that needs minimum average lookahead. *r* is a triple (d_1, d_2, f) denoting applying the action f on the vertex d_1 to generate the vertex d_2.
- d: The unique index of the vertex.

The inputs to compute_average_lookahead are an LR state transition $r = (d', d, f)$ taken in the construction of the graph-structured stack where d' and d are the index of vertices and *f* is an action, and the set A containing the parsing action sequences of the MAL parse of all sentences processed so far. Let $v = (i, s, a, k, r, d)$ be the new vertex with index d, and let $v' = (i', s', a', k', r', d')$ be the vertex with index d'. The function proceeds by first computing the lookahead needed to resolve conflicts between r and A. Next we check whether v is a packed node and initialize k in v; if not, k is initialized to $(0, 0)$, and otherwise it is copied from the packed node. We then compute the average lookahead needed to go from v' to v and check whether it provides a more economical way to resolve conflicts. The average lookahead of a vertex v generated by applying an action f on vertex v' can be computed from k' of v' and the lookahead needed to generate v from v'. v can be generated by applying different actions on different vertices and k keeps the one that needs minimum average lookahead and f keeps that action.

Finally, the **reconstruct_MAL_parse** function is called after construction of the entire graph-structured stack is complete in order to recover the full minimum average lookahead parse tree. We assume the grammar has only one start symbol and rebuild the parse tree from the state that is labelled with the start symbol.

Table 1: Example constraining grammar.

(1)	S → NP VP
(2)	VP → v NP
(3)	VP → v PP
(4)	VP → v
(5)	VP → v p
(6)	VP → v det
(7)	PP → p NP
(8)	NP → NP PP
(9)	NP → n
(10)	NP → det n
(11)	VP → VP n

4 An example

We now walk through a simplified example so as to fix ideas and illustrate the operation of the algorithm. Table 1 shows a simple constraining grammar G_C which we will use for this example.

Now consider the small training corpus:

1. I did.
2. He went to Africa.
3. I bought a ticket.

Table 2: LR state transitions and lookaheads for sentence 1.

[$_S$ [$_{NP}$ I_n] [$_{VP}$ did_v]]	$\hat{k}$
(0, 1, sh, 0) (1, 2, re9, 0) (2, 4, sh, 0)	
(4, 5, re4, 0) (5, 3, re1, 0) (3, acc)	0*

To begin with, find_MAL_parser considers sentence 1. In this particular case, chart_parse(S_1, G_C) finds only one valid parse. The GLR forest is built, giving the LR state transitions and parsing actions shown in Table 2, where each tuple (d', d, f, k) gives the state prior to the action, the state resulting from the action, the action, and the average lookahead. Here compute_average_lookahead determines that the average lookahead $\hat{k}$ is 0. From this parse tree, incremental_update_LR accepts rules (1), (4), and (9) and updates the previously empty LR table T.

Next, find_MAL_parser considers sentence 2. Here, chart_parse(S_1, G_C) finds two possible parses, leading to the LR state transitions and parsing actions shown in Table 3. This time, the average lookahead calculation is sensitive to the what was already entered into the LR table T during the previous step of processing sentence 1. For example, in the first parse, the fourth transition (4, 6, sh, 1) requires a lookahead of 1 in order to avoid a shift-reduce conflict with (4, 5, re4, 0) from sentence 1. The sixth transition (1, 9, re9, 2) requires a lookahead of 2. It turns out that the first parse has an average lookahead of 0.20,while the second parse has an average lookahead of 0.33. We thus prefer the first parse tree, calling incremental_update_LR to further update the LR table T using rules (3) and (7).

Finally, find_MAL parser considers sentence 3. Again, chart_parse(S_1, G_C) finds two possible parses, leading this time to the LR state transitions and parsing actions shown in Table 4. Various lookaheads are again needed to avoid conflicts with the existing rules in T. The first parse has an average lookahead of 0.22, and is selected in preference to the second parse which has an average lookahead of 0.33. From the first parse tree, incremental_update_LR accepts rules (2) and (10) to again update the LR table T.

Thus the final output MAL grammar, requiring a lookahead of 1, is shown in Table 5.

Table 3: LR state transitions and lookaheads for sentence 2.

[$_S$ [$_{NP}$ He_n] [$_{VP}$ $went_v$ [$_{PP}$ to_p [$_{NP}$ $Africa_n$]]]]	$\hat{k}$
(0, 1, sh, 0) (1, 2, re9, 0) (2, 4, sh, 0)	
(4, 6, sh, 1) (6, 1, sh, 0) (1, 9, re9, 1)	
(9, 7, re7, 0) (7, 5, re3, 0) (5, 3, re1, 0)	
(3, acc)	**2/10**
[$_S$ [$_{NP}$ He_n] [$_{VP}$ [$_{VP}$ $went_v$ to_p] $Africa_n$]]	
(0, 1, sh, 0) (1, 2, re9, 0) (2, 4, sh, 0)	
(4, 6, sh, 1) (6, 5, re5, 0) (5, 8, sh, 1)	
(8, 5, re11, 0) (5, 3, re1, 1) (3, acc)	3/9

Table 4: LR state transitions and lookaheads for sentence 3.

[$_S$ [$_{NP}$ I_n] [$_{VP}$ $bought_v$ [$_{NP}$ a_{det} $ticket_n$]]]	$\hat{k}$
(0, 1, sh, 0) (1, 2, re9, 1) (2, 4, sh, 0)	
(4, 8, sh, 1) (8, 11, sh, 0) (11, 12, re10, 0)	
(12, 5, re12, 0) (5, 3, re1, 0) (3, acc)	**2/9**
[$_S$ [$_{NP}$ I_n] [$_{VP}$ [$_{VP}$ $bought_v$ a_{det}] $ticket_n$]]	
(0, 1, sh, 0) (1, 2, re9, 1) (2, 4, sh, 0)	
(4, 8, sh, 1) (8, 5, re6, 0) (5, 10, sh, 1)	
(10, 5, re11, 0) (5, 3, re1, 0) (3, acc)	3/9

Table 5: Output MAL grammar.

(1)	S → NP VP
(2)	VP → v NP
(3)	VP → v PP
(4)	VP → v
(7)	PP → p NP
(9)	NP → n
(10)	NP → det n

5 Complexity analysis

5.1 Time complexity

Since the algorithm executes each of its five main steps once for each sentence in the corpus, the time complexity of the algorithm is upper bounded by the sum of the time complexities of those five steps. Suppose n is the maximum length of any sentence in the corpus, and m is the number of rules in the grammar. Then for each of the five steps:

1. **chart_parse** is $O\left(n^3\right)$.
2. **build_GLR_forest** is $O\left(n^3\right)$. As discussed previously, the use of an ancestor table allows

the graph-structured stack to be built in $O(n^3)$ in the worst case.

3. **compute_average_lookahead** is $O\left(n^2\right)$. As the number of lookaheads needed by each parsing action is computed by comparing the parsing action with the MAL parsing action sequences for all previous sentences, the time complexity of this function depends on the maximum length of any sentence that has already been processed, which is bounded by n. The dynamic programming method used to locate the most economical parse in terms of average lookahead, described above, can be seen to be quadratic in n.
4. **reconstruct_MAL_parse** is $O\left(n^2\right)$. This is bounded by the number of LR state transitions in each full parse of the sentence, which is is $O\left(n^2\right)$. Note, however, that Tanaka *et al.* (1992) propose an enhancement that can reconstruct the parse trees in time linear to n; this is a direction for future improvement of our algorithm.
5. **incremental_update_LR** is $O\left(2^m\right)$. As with *ilalr*, theoretically the worst time complexity is exponential in the number of rules in the existing grammar. However, various heuristics can be employed to make the algorithm quicker, and in practical experiments the algorithm is quite fast and precise in producing LR tables, particularly since m is very small relative to $|S|$.

The time complexity of the algorithm for each sentence is thus $O\left(n^3\right) + O\left(n^3\right) + O\left(n^2\right) + O\left(n^2\right) + O\left(2^m\right)$ which is $O(n^3 + 2^m)$. Given $|S|$ sentences in the corpus, the total training time is $O\left(\left(n^3 + 2^m\right) \cdot |S\,|\right)$.

5.2 Space complexity

As with time complexity, an upper bound on the space complexity can be obtained from the five main steps:

1. **chart_parse** is $O\left(n^3\right)$.
2. **build_GLR_forest** is $O\left(n^2\right)$. The space complexity of both Tomita's original algorithm and the modified algorithm is n^2.
3. **compute_average_lookahead** is $O\left(n^2\right)$. The space usage of compute_average_lookahead directly corresponds to the dynamic programming structure, like the time complexity.
4. **reconstruct_MAL_parse** is $O\left(n\right)$. This is bounded by the number of vertices in the graph-structured stack, which is is $O\left(n\right)$.
5. **incremental_update_LR** is $O\left(2^m\right)$. As with time complexity, although the worst time complexity is exponential in the number of rules in the existing grammar, in practice this is not the major bottleneck.

The space complexity of the algorithm is thus $O\left(n^3\right) + O\left(n^2\right) + O\left(n^2\right) + O\left(n\right) + O\left(2^m\right)$ which is again $O(n^3 + 2^m)$.

6 Conclusion

We have defined a new grammar learning task based on the concept of a *minimum average lookahead* (MAL) objective criterion. This approach provides an alternative direction for modeling of incremental parsing: it emphatically avoids *increasing* the amount of nondeterminism in the parsing models, as has been done in across a wide range of recent models, including probabilized dynamic programming parsers as well as GLR approaches. In contrast, the objective here is to learn completely deterministic parsers from unannotated corpora, with loose environmental guidance from nondeterministic *constraining grammars*.

Within this context, we have presented a greedy algorithm for the difficult task of learning approximately MAL grammars for deterministic incremental LR(k) parsers, with a time complexity of $O\left(\left(n^3 + 2^m\right) \cdot |S\,|\right)$ and a space complexity of $O(n^3 + 2^m)$. This algorithm is efficient in practice, and thus enables a broad range of applications where degree of lookahead serves as a grammar induction bias.

Numerous future directions are suggested by this model. One obvious line of work involves experiments varying the types of corpora as well as the numerous parameters within the MAL grammar learning algorithm, to test predictions against various modeling criteria. More efficient algorithms and heuristics could help further increase the applicability of the model. In addition, the accuracy of the model could be strengthened by reducing sensitivity to some of the approximating assumptions.

References

Marie-Pierre Beal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theoretical Computer Science*, 289(1), 1968.

Ted Briscoe and John Carroll. Generalised probabilistic LR parsing for unification-based grammars. *Computational Linguistics*, 19(1):25–60, 1993.

Jay Earley. An efficient context-free parsing algo-

rithm. *Communications of the Association for Computing Machinery*, 13(2), 1970.

G. Fischer. Incremental LR(1) parser construction as an aid to syntactical extensibility. Technical report, Department of Computer Science, University of Dortmund, Federal Republic of Germany, 1980. PhD Dissertation, Tech. Report 102.

John Hale. A probabilistic Earley parser as a psycholinguistic model. In *NAACL-2001: Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

Jan Heering, Paul Klint, and Jan Rekers. Incremental generation of parsers. *IEEE Transactions on Software Engineering*, 16(12):1344–1351, 1990.

Ulf Hermjakob and Raymond J. Mooney. Learning parse and translation decisions from examples with rich context. In *ACL/EACL'97: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 482–489, 1997.

R. Nigel Horspool. Incremental generation of LR parsers. Technical report, University of Victoria, Victoria, B.C., Canada, 1988. Report DCS-79-IR.

James R. Kipps. GLR parsing in time O(n^3). In M. Tomita, editor, *Generalized LR Parsing*, pages 43–60. Kluwer, Boston, 1991.

Mitchell P. Marcus. *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, MA, 1980.

Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1–2):177–201, 2000.

Srini Narayanan and Daniel Jurafsky. Bayesian models of human sentence processing. In *Proceedings of CogSci-98*, 1998.

Hozumi Tanaka, K.G. Suresh, and Koiti Yamada. A family of generalized LR parsing algorithms using ancestors table. Technical report, Department of Computer Science, Tokyo Institute of Technology, Tokyo, Japan, 1992. TR92-0019.

Masaru Tomita and See-Kiong Ng. The Generalized LR parsing algorithm. In Masaru Tomita, editor, *Generalized LR Parsing*, pages 1–16. Kluwer, Boston, 1991.

Masaru Tomita. *Efficient Parsing for Natural Language*. Kluwer, Boston, 1986.

Aboy Wong and Dekai Wu. Learning a lightweight robust deterministic parser. In *EUROSPEECH'99: Sixth European Conference on Speech Communication and Technology*, Budapest, Sep 1999.