Context-Free Grammar Parsing by Message Passing

Article · November 1993

Source: CiteSeer

CITATIONS
11

READS
44

2 authors, including:

Randy Goebel
University of Alberta
167 PUBLICATIONS **1,638** CITATIONS

SEE PROFILE

Context-Free Grammar Parsing by Message Passing

Dekang Lin
Dept. of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada, R3T 2N2
lindek@cs.umanitoba.ca
(204) 474-8313

Randy Goebel
Dept. of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2H1
goebel@cs.ualberta.ca
(403) 492-2683

Abstract

A message passing algorithm for parsing context-free grammars (CFG) is presented. The grammar is represented by a network, whose nodes represent rules and non-terminals and whose links represent the structural relationships between the rules and non-terminals. The message passing algorithm computes a shared, packed parse forest for the input sentence.

The algorithm has the following characteristics:

- The worst case complexity of the algorithm is $O(|G|n^3)$, where n is the length of input sentence, $|G|$ is size of the grammar.
- The object-oriented nature of the algorithm makes it easier to extend the parser to handle more sophisticated grammar formalisms with CFG as the backbone.

Keywords: parsing, context-free grammar, message-passing algorithm, efficiency

1 Introduction

Context-free grammars (CFG) play an important role in natural language parsing. Many linguistic theories such as Lexical-Functional Grammar (LFG) [KB82] and Generalized Phrase Structure Grammar (GPSG) [GKPS85] use context-free grammars as the backbone; the parsing algorithms for these formalisms are usually extensions of context-free parsing algorithms such as chart parsers and Earley's parser [Ear70].

We present a message passing algorithm for parsing context-free grammars. The grammar is represented by a network, whose nodes represent rules and non-terminals and whose links represent the structural relationships between the rules and non-terminals. Given an input sentence, a shared, packed parse forest [Tom86] is computed by a message passing algorithm in the grammar network. The nodes in the network are also computing agents, which communicate by sending messages across the links in the network. The messages are intervals representing segments of the input sentence, and are sent from one node to another in the reverse direction of the links. When the message passing process finishes, a shared, packed parse forest can be retrieved from the network.

The worst case complexity of our algorithm is $O(|G|n^3)$, where n is the length of input sentence and $|G|$ is size of the grammar (the number of *occurrences* of non-terminal symbols in the set of grammar rules). Most natural language grammars tend to have a large number of rules but relatively small number of words in a sentence, so this is is a significant improvement over many previous CFG parsing algorithms. For example, Earley's parser and chart parsers have the complexity $O(|G|^2n^3)$ [BBR87, p.198]. Note further that the complexity measure is based on a serial simulation of parallel distributed message passing.

The next section introduces a network representation of context free grammars. The message passing algorithm is presented in Section 3. Section 4 discusses various properties of our algorithm. A description of an object-oriented implementation of the parser and its complexity analysis are presented in Sections 5 and 6. Section 7 deals with cyclic and ϵ-rules. Section 8 shows some experimental result. Finally, related work is discussed in Section 9.

2 Network Representation of Grammar

We define a context-free grammar as a 6-tuple $<N, O, T, s, P, L>$, where

- N is a finite non-empty set of non-terminals.
- O is the set of pre-terminals, a non-empty subset of N.
- T is the set of terminals, a finite non-empty set disjoint from N.
- s is the start symbol, an element of N.
- P is the set of production rules. The rules in P are in the form:
  $$R_i\text{: } m_i \longrightarrow d_{i_0} \ldots d_{i_{k_i}}$$
  where R_i is the name of the rule and $m_i, d_{i_0}, \ldots, d_{i_{k_i}}$ are non-terminals.
- L is the lexicon which consists of association pairs (w, p), where $w \in T$ is a terminal and $p \subset O$ names w's parts of speech.

This definition of context-free grammar is equivalent to its usual formulation such as in [HU69]. Figure 1 shows an example context-free grammar.

The grammar network representing a CFG $<N, O, T, s, P, L>$ is a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links, and is constructed as follows:

- There is a node in G representing each non-terminal and production rule in the grammar, i.e., $V = N \cup P$;
- The links E are between elements of N and rule names in P, as prescribed by the structure of each rule: for rule R_i: $m_i \longrightarrow d_{i_0} \ldots d_{i_{k_i}}$, G has a link from m_i to R_i and a link from R_i to d_{i_j} for each $j = 0, \ldots, k_i$. The link from R_i to d_{i_j} is labeled j.

Since each instance of non-terminal symbol in the grammar corresponds to a link in the grammar network, the complexity of constructing of the network is $O(|G| \log(|N| + |O| + |P|))$, where $|N| + |O| + |P|$ is the number of nodes in the network.

The network representing the grammar in Figure 1 is shown in Figure 2. There are two types of nodes in the network. The nodes representing non-terminals are called `NT` nodes and those representing phrase structure rules are known as `PSR` nodes. There are also two types of links in the network. A link from an `NT` node to a `PSR` node is called an `OrLink` and a link from a `PSR` node to an `NT` node is called an `AndLink`. The `AndLinks` are numbered $0 - n$ according to the ordering of the non-terminals in the corresponding production rule.

3 Parsing by Message Passing

We now present a message passing algorithm for parsing context-free languages. We use integers to represent the word boundaries in the input sentence. Segments of the sentence can then be represented by intervals. For example, consider the sentence Figure 3 the interval [1,4] then represents the segment *saw a man*. The complete sentence is represented by the interval [0,7].

The nodes in the network are computing agents which communicate with each other by passing messages across the links in the network. The messages are intervals representing segments of the input sentence, and are passed from one node to another in the reverse direction of the links. If a message [i, j] reaches an `NT` node (say NP), then this means that the segment [i, j] of the input sentence is an NP.

The nodes respond to messages according to their types. When an `NT` node receives a message, it first checks whether an identical message has been received before. If not, it sends the message further up the incoming links. For example, in Figure 2, when the node NP receives a message [0,1] from the node R_{np1}, it checks whether [0,1] has been received before. If not, the message [0,1] is sent to the nodes R_s, R_{vp2}, R_{np3} and R_{pp}.

A node has local memory which contains a set of items. An item is a pair (m, l), where m is an interval and l is a label of an outgoing link. When a `PSR` node receives a message m from link e, it creates an item (m, l_e), where l_e is the label for link e. If $l_e = 0$, then the item is saved into the local memory. The node combines the item (m, l_e) with other items in the local memory that are compatible with it. An item $(m', l_{e'})$ is compatible with (m, l_e) iff

1. there exists i, j, k such that $m' = [i, j]$ and $m = [j, k]$; and
2. $l_{e'} = l_e - 1$.

The combination of $(m', l_{e'})$ and (m, l_e) is $(m' \cup m, l_e)$, and is saved in local memory. If e is the last link, the message $m' \cup m$ is sent further. For instance, in Figure 2, when the node R_{np2} has received the message [3,4] from the node $*n$, an item

N={S, NP, VP, PP, $*n$, $*v$, $*p$, $*d$}
O={$*n$, $*v$, $*p$, $*d$}
T={I, saw, a, man, in, the, park}
s=S
P= R_s: S$\longrightarrow$NP VP; R_{np1}: NP$\longrightarrow*n$;
R_{np2}: NP$\longrightarrow*d\ *n$; R_{np3}: NP$\longrightarrow$NP PP;
R_{vp1}: VP$\longrightarrow$VP PP; R_{vp2}: VP$\longrightarrow*v$ NP;
R_{pp}: PP$\longrightarrow*p$ NP;
L= { (I, {$*n$}), (saw, {$*v$, $*n$}), (a, {$*d$}), (man, {$*n$}),
(in, {$*p$}), (the, {$*d$}), (park, {$*n$}) }

Figure 1: An example CFG grammar

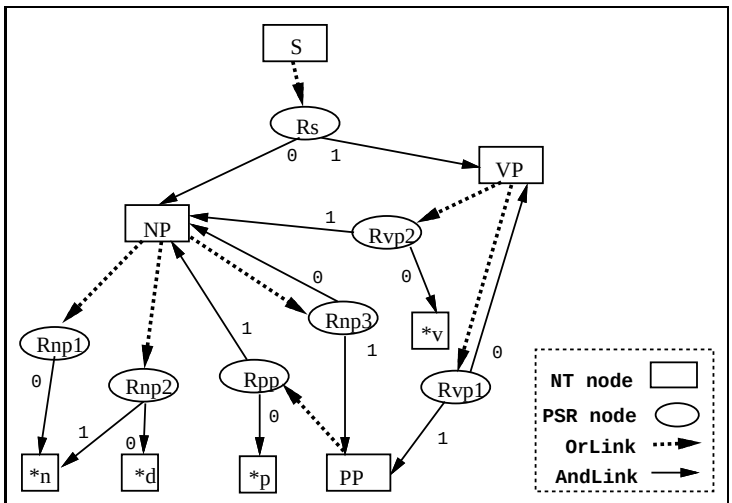

Figure 2: A network representation of grammar

([3,4], 1) is created. This item is combined with the item ([2,3], 0), which was created when the message [2,3] from the node $*d$ arrived. Their combination is ([2,4], 1) and a message [2,4] is sent to the the node NP.

The input sentence is parsed from left to right. The parts of speech of the words in the sentence are first retrieved from the lexicon. For each word in the sentence, the interval representing that word is sent to the appropriate nodes. These messages activate a message passing process in the grammar network. When the process stops, i.e., no further messages are being sent, the message representing the next word is fed into the network. When the end of the sentence has been reached, the input sentence is successfully parsed iff the message $[0, n]$ has been received by the node representing the start symbol s, where n is the number of words in the input sentence. By tracing the origins of the message $[0, n]$, a shared, packed parse forest is obtained, from which all the parse trees of the input sentence can be retrieved.

To illustrate our parsing scheme, let's consider the example in Figure 3. Suppose the grammar network is the one in Figure 2. The parsing process is initiated by sending the message [0,1] to the node $*n$, which forwards the message to the nodes R_{np1} and R_{np2}. The node R_{np1} then sends [0,1] to the node NP, which in turn forwards the message to R_s, R_{pp}, R_{vp2} and R_{np3}. None of these nodes generate any new messages. To process the next word, the message [1,2] is sent to the nodes $*n$ and $*v$. A message passing process is initiated again. When that process stops, the message [2,3] is sent to the node $*d$; then [3,4] is sent to the node $*n$, and so on, until finally, [6,7] is sent to node $*n$. In Figure 4, the intervals beside each node are the messages that have

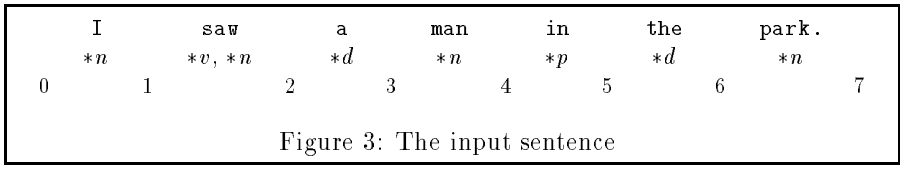

Figure 3: The input sentence

been sent by the node while parsing the sentence.

Figure 5 shows the trace of the message [0,7] at the node S, which constitutes a shared, packed parse forest for the input sentence.

4 Discussion

4.1 Lexical ambiguity

Words with multiple parts of speech can simply be dealt with by sending identical initial messages to each one of the nodes representing the part of speech of the word. In our example, the message [1,2] is sent to both $*n$ and $*v$, which are two possible parts of speech for the word *saw*.

4.2 Subtree sharing and local ambiguity packing

The idea of subtree sharing allows two or more parse trees with a common subtree to be represented only once. Subtree sharing is inherent in our parsing algorithm, because we represent parse trees by traces of the messages. For example, the message [2,4] (representing "a man") at the node NP is sent to all appropriate connected `PSR` nodes, including both R_{np3} and R_{vp2}. These latter copies will eventually be combined with other messages and to generate the message [0,7] at the start node S. The trace of the message [2,4] at node NP is thus shared.

A fragment of a sentence is locally ambiguous if the fragment can be reduced to a certain non-terminal in two or more ways. As has been observed (e.g., in [Tom86]), if a sentence has many local ambiguities, the total ambiguity would grow exponentially. In our parsing algorithm, the local ambiguities are locally contained because no identical messages are sent across a link more than once.

5 Object-oriented Implementation

The parser has been implemented in C++, an object-oriented extension of C. The object-oriented paradigm makes the relationship between nodes and links in the grammar network and their software counterpart more explicit. Communication via message passing is reflected in the message passing metaphor used in object-oriented languages.

Nodes and links are implemented as objects. The class hierarchy of nodes in shown in Figure 6. `NT` is the class of nodes representing non-terminal symbols. `PSR` the classes of nodes representing PS-rules. `Node` is an abstraction of `NT` and `PSR`. Each class of objects has a set of methods (also called member functions), which provide the only public access to the objects. The methods of a class can either be inherited or redefined by its subclasses. The class `Node` has three methods: `Send, Receive,` and `IsNew.` The method `Send` is inherited by both subclasses. The method `Receive`, on the other hand, is redefined by the subclasses. The method `IsNew` returns true if no identical message has been received by the node. `Compatible, Combine` and `Complete` are methods of `PSR`. The functionality of the methods `Compatible` and `Combine` are as in Section 3. The method `Complete(item)` returns true if `item`=$([i,j],l)$ and l is the label of the last link of the node.

The class hierarchy for links is shown in Figure 7. The links from `NT` nodes to `PSR` nodes belong to the class `OrLink`. The method `Tail` returns the tail node of the link. The method `Label` returns an integer label for an `AndLink`.

The message passing algorithm is presented in pseudo code and is shown in Figure 8.

6 Complexity Analysis

We now analysize the time complexity of the algorithm in a uniform-cost RAM model. The size of the grammar $|G|$ is defined to be the number of *occurrences* of non-terminal symbols in production

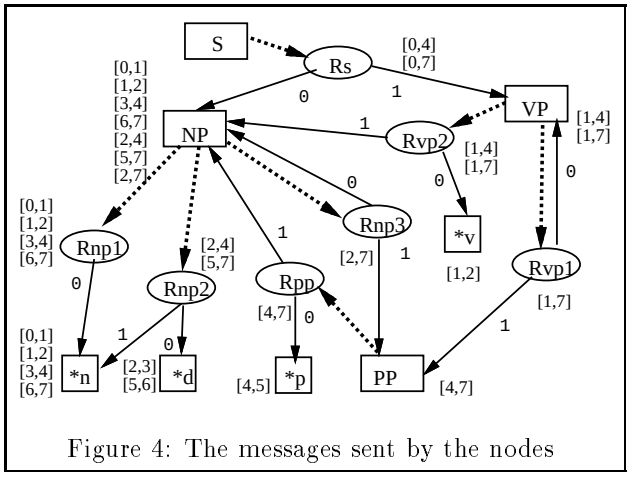

Figure 4: The messages sent by the nodes

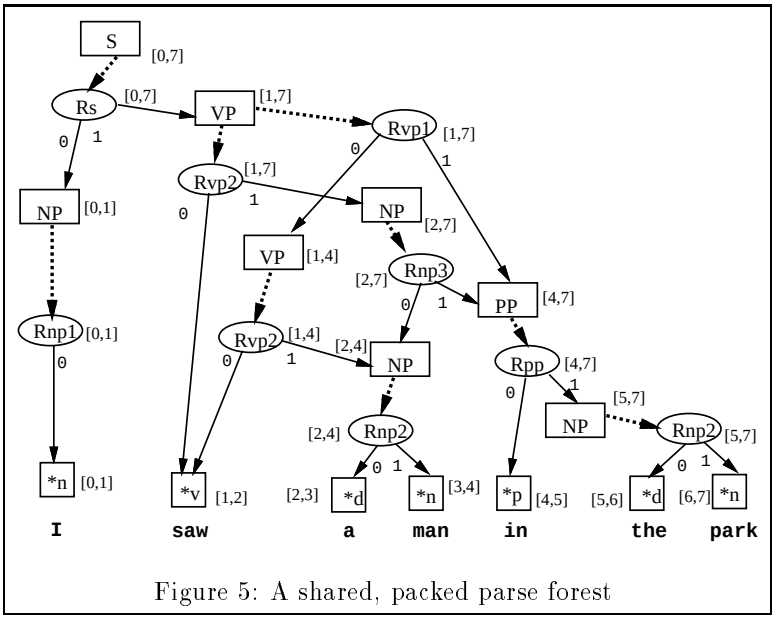

Figure 5: A shared, packed parse forest

rules. Since there is a one–one correspondence between the links in the grammar network and occurrences of the non-terminal symbols in the grammar, the number of links in the grammar network is equal to $|G|$. Since a message is an interval $[i, j]$, where $0 \leq i < j \leq n$, the total number of distinct messages is $O(n^2)$, where n is sentence length.

Our evaluation of the cost of each component of the algorithm is separated into the three parts specified in Figure 8. Each of the three components consists of several individual statements, labeled, respectively, s_i, t_i, and r_i for the send method for `Node` the receive method for `NT` and the receive method for the abstraction `PSR`. We assume that it takes unit time to combine two messages or to store a message into the local memory of a node. The right column in Figure 8 shows the complexity measure of each line of the algorithm. The values of s_i, t_i and r_i's are upper bounds for the number of times that line is executed in parsing a sentences of length n. The symbols $|G|$, $|G_O|$ and $|G_A|$ denote the total number of links, the number of `OrLinks`, and the number of `AndLinks` in the network, respectively. The complexity measures in Figure 8 are explained as follows:

s_1 is the number of times Node::Send is called. Since the calls to it are on lines 9, 14, 19, $s_1 = t_1 + r_3 + r_6$.

$s_2 = O(n^2)|G|$ because whenever s_2 is reached the

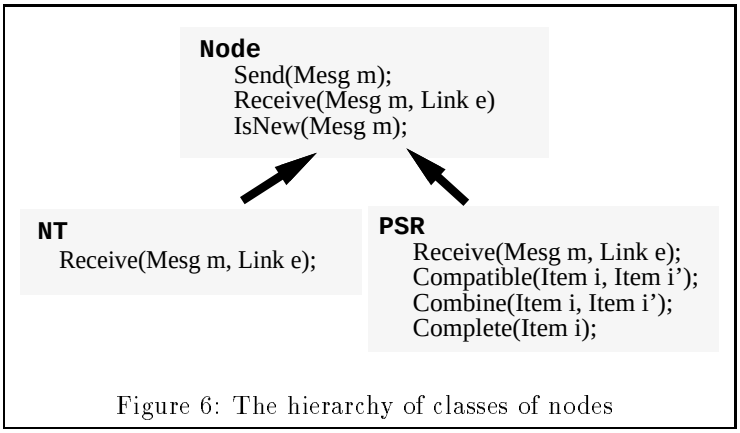

Figure 6: The hierarchy of classes of nodes

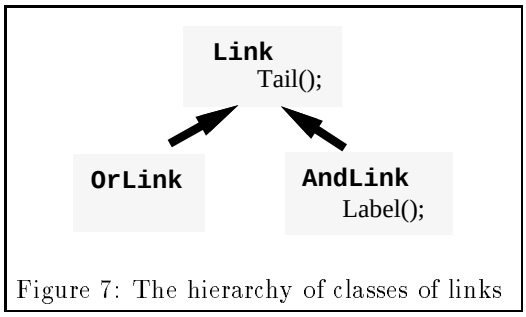

Figure 7: The hierarchy of classes of links

pair (m, e) is unique. The number of different m's is bounded by $O(n^2)$ and the number of different e's is $|G|$.

t_1 is the number of times NT::Recieve is called. Since each time NT::Receive is called, the pair (m,e) is unique and e mucst be an `OrLink`, $t_1 = O(n^2)|G_O|$.

r_1 is the number of times PSR::Receive is called. For the similar reason as t_1, $r_1 = O(n^2)|G_A|$.

$r_4 = O(n^3)|G_A|$ for the following reason: When a message $[i, j]$ arrives at a node from a link labeled l, an item $([i, j], l)$ is created. The items that are compatible with this item must be in the form $([k, i], l - 1)$, where k may be $0, 1, \ldots, i - 1$. Thus, $([i, j], l)$ is combined with at most i other items. Therefore $r_4 = n \times r_3 = O(n^3|G_A|)$.

The worst case complexity of the algorithm is

$$\sum s_i + \sum t_i + \sum r_i = O(|G|n^3).$$

7 Cyclic and ϵ rules

A set of production rules is cyclic if a symbol can be reduced to itself after non-zero number of reductions by these rules. A simple example of a cyclic rule set is:

$$A \longrightarrow B; \qquad B \longrightarrow C; \qquad C \longrightarrow A;$$

Cyclic rules do not present any difficulty for our parser, i.e., they do not cause infinite number of messages, because no identical messages are sent across a link more than once.

An ϵ-rule is a production rule whose right-hand-side is empty. For example: $A \longrightarrow \epsilon$ is an ϵ-rule and A is called a potentially empty non-terminal.

To handle ϵ-rules, we can expand a `PSR` node into k-1 `PSR` nodes, where k is the number of symbols on the right-hand-side of the production rule. Each of these nodes has no more than two outgoing links. For example, Figure 9.1, which represent the production rule A $\longrightarrow$ B C D E, can be expanded into Figure 9.2. The expanded network contains less than twice as many links as the original network.

We then find which nodes in the network are potentially null. An `NT` node is potentially null if it has an `OrLink` to a pontentially null node. A `PSR` node is potentially null if all is outgoing `AndLinks`

	"Send" Method for `Node`	complexity
1	`Node::Send`(Message m) {	
2	if (not `IsNew`(m)) then return;	$s_1 = t_1 + r_3 + r_6$
3	for each incoming link e do {	
4	t = e.`Tail`();	$s_2 = O(n^2)\|G\|$
5	t.`Receive`(m, e);	$s_3 = s_2$
6	}	
7	}	

	"Receive" Method for `NT`	complexity
8	`NT::Receive`(Mesg m, Link e) {	
9	if (`IsNew`(m)) `Send`(m);	$t_1 = O(n^2)\|G_O\|$
10	}	

	"Receive" Method for `PSR`	complexity
11	`PSR::Receive`(Mesg m, Link e) {	
12	i = Item(m, e.Label());	$r_1 = O(n^2)\|G_A\|$
13	save i in local storage;	$r_2 = r_1$
14	if (`Complete`(i)) then `Send`(m);	$r_3 = r_2$
15	for all i' in local store such that `Compatible`(i', i){	
16	item = `Combine`(i', i);	$r_4 = n \times r_3$
17	save item in local memory;	$r_5 = r_4$
18	if (`IsNew`(item) and `Complete`(item)) then	
19	`Send`(mesg(item));	$r_6 = r_4$
20	}	
21	}	

Figure 8: A message passing algorithm

are points to pontentially null nodes. Since this can be done by traversing the links in the network no more than once, the time to find the potentially null nodes is $O(|G|)$.

In the expanded network, whenever a `PSR` node receive a message via an `AndLink`, if the other `AndLink` points to a potentially null node and the node has not received a identically message before, a copy of the message is forworded to the parent of the node. Since the expension is linear and the complexity of the parsing algorithm does not change with respect to the size of the network, CFG grammars with ϵ-rules can be parsed within $O(|G|n^3)$.

8 Experimental results

The implementation of the parsing algorithm has been tested on a SPARCstation/SLC. The test grammar is from [Tom86, Grammar III, pp.172–6], which contains about 220 phrase structure rules. The 40 test sentences are also from [Tom86, Appendix G, pp.185-9]. A summary of the experimental results is shown in Table 1. The sentences in [Tom86] are referred to by their Sentence No.'s. The Length refers to the number of words in the sentence. The Parsing Time is measured in seconds.

9 Related Work

The algorithm presented here is similar to Cocke-Kasami-Younger algorithm [You67], an improved version of it [GHR80], and chart parsing, in that dynamic programming technique is employed to combine smaller parse trees into larger ones. The the equivalent of matrix or chart in these algorithms are distributed over the nodes that represent rules and symbols in our algorithm. The advantage of the object-oriented nature of our algorithm is that nodes may be heterogeneous. That is, they may impose different sets constraints on message combination and progation. For example, [Lin93] presented a parser based on the CFG parsing algorithm pre-

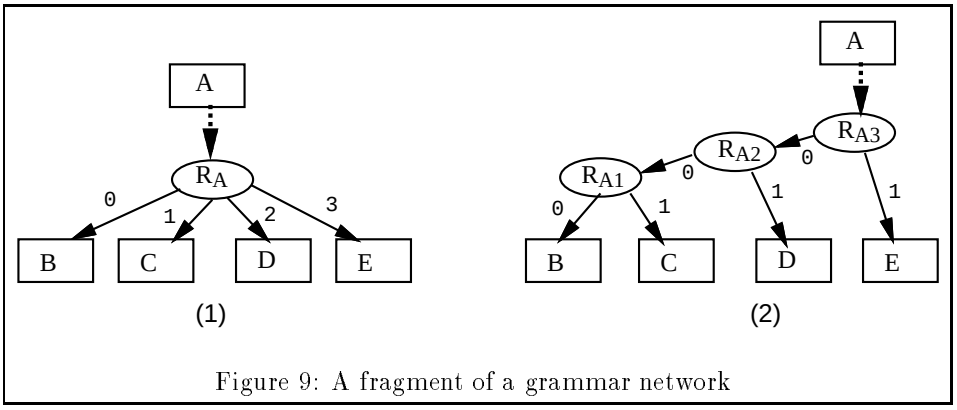

Figure 9: A fragment of a grammar network

Table 1: Experimental Results

Sentence No.	Length	Parse Time	Sentence No.	Length	Parse Time	Sentence No.	Length	Parse Time	Sentence No.	Length	Parse Time
1	19	0.213	11	18	0.176	21	19	0.167	31	23	0.205
2	11	0.064	12	19	0.166	22	14	0.059	32	30	0.248
3	26	0.281	13	19	0.139	23	13	0.085	33	28	0.255
4	8	0.037	14	12	0.076	24	7	0.030	34	26	0.278
5	12	0.064	15	11	0.058	25	7	0.034	35	23	0.165
6	13	0.071	16	14	0.135	26	9	0.048	36	1	0.001
7	11	0.079	17	13	0.112	27	13	0.133	37	2	0.001
8	7	0.034	18	17	0.136	28	34	0.279	38	4	0.013
9	15	0.072	19	27	0.147	29	8	0.050	39	5	0.027
10	22	0.147	20	13	0.082	30	28	0.248	40	5	0.043

sented here, which implements the Government and Binding Theory [Cho81, vW86]. The messages are augemented with attribute values and the principles in GB theory are implemented as constraints on the attribute values during the message passing process.

Yonezawa and Ohsawa [YO88] presented a parallel object-oriented parser for context-free languages. A brief comparison is shown in Table 2.

It can be seen that our algorithm involves smaller networks, simpler messages, and has greater capability. Yonezawa and Ohsawa claimed that the time it takes their algorithm to find the first parse tree is $O(nh)$, where h is the height of the parse tree. However, their analysis implicitly assumed that each node has the capability of a non-deterministic Turing machine, i.e., they are always able to pick the right messages to send. Moreover, their parse trees are generated one by one, in a pipeline manner. Since there can be exponential number of parses, merely enumerating the results may take exponential time.

10 Conclusion

We have presented a network representation of context-free grammars and an all-path parsing method which uses message passing in the corresponding grammar network. The result is a shared, packed parse forest for the input sentence. The worst case complexity of the algorithm is $O(|G|n^3)$. The object-oriented nature of the algorithm make extensible to handle more sophisticated grammar formalisms.

Acknowledgement

The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Re-

Table 2: A comparison with Yonezawa and Ohsawa

	Lin and Goebel	Yonezawa and Ohsawa
Node	a non-terminal symbol	an occurrence of a terminal or non-terminal symbol
Message	interval	partial parse tree
Multi-parts-of-speech	Yes	Yes
Cyclic rules	Yes	No
ϵ-rule	Yes	No
Subtree sharing	Yes	No
Local ambiguity packing	Yes	No
Complexity	$O(\|G\|n^3)$	exponential

search Council, and the participation of PRECARN Associates Inc. Thanks are due to Fred Popowich for helpful comments on an earlier draft. We are very grateful to anonymous reviewers for their detailed and valuable comments.

References

[BBR87] G. Edward Barton, Jr., Robert C. Berwick, and Eric Sven Ristad. *Computational Complexity and Natural Language*. Computational Models of Cognition and Perception. The MIT Press, Cambridge, Massachusetts, 1987.

[Cho81] Norm Chomsky. *Lectures on Government and Binding*. Foris Publications, Cinnaminson, USA, 1981.

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *ACM Communications*, 13:94–102, 1970.

[GHR80] S. L. Graham, M. A. Harrison, and W. L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, July 1980.

[GKPS85] Gerald Gazdar, Ewan Klein, Geoffery Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Basil Blackwell Publisher Ltd, Oxford, UK, 1985.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Publishing Company, 1969.

[KB82] R. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In J Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, 1982.

[Lin93] Dekang Lin. Principle-based parsing without overgeneration. (submitted to ACL–93), 1993.

[Tom86] Masaru Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Norwell, Massachusetts, 1986.

[vW86] Henk van Riemsdijk and Edwin Williams. *Introduction to the Theory of Grammar*. Current Studies in Linguistics. The MIT Press, Cambridge, Massachusetts, 1986.

[YO88] Akinori Yonezawa and Ichiro Ohsawa. Object-oriented parallel parsing for context-free grammars. In *Proceedings of COLING-88*, 1988.

[You67] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.