An Efficient Enumeration Algorithm of Parses for Ambiguous Context-Free Languages

Nariyoshi YAMAI†, Tadashi SEKO†, Noboru KUBO†† and Toru KAWATA††

† Department of Information Engineering, Nara National College of Technology, Yamatokoriyama, Nara 639-11, Japan
†† Computer Systems Laboratories, Corporate Research and Development Group, SHARP Corporation, Tenri, Nara 632, Japan

Abstract

An efficient algorithm that enumerates parses of ambiguous context-free languages is described, and its time and space complexities are discussed.

When context-free parsers are used for natural language parsing, pattern recognition, and so forth, there may be a great number of parses for a sentence. One common strategy for efficient enumeration of parses is to assign an appropriate weight to each production, and to enumerate parses in the order of the total weight of all applied production. However, the existing algorithms taking this strategy can be applied only to the problems of limited areas such as regular languages; in the other areas only inefficient exhaustive searches are known.

In this paper, we first introduce a hierarchical graph suitable for enumeration. Using this graph, enumeration of parses in the order of acceptablity is equivalent to finding paths of this graph in the order of length. Then, we present an efficient enumeration algorithm with this graph, which can be applied to arbitrary context-free grammars. For enumeration of k parses in the order of the total weight of all applied productions, the time and space complexities of our algorithm are $O(n^3 + kn^2)$ and $O(n^3 + kn)$, respectively.

1 Introduction

Context-free parsers are commonly used for natural language parsing, pattern recognition, and so forth. In these applications, there may be a great number of parses (or derivations) for a sentence, only a few of which would be needed in later processes. Therefore, we look up only a few promising parses and do not make an inefficient exhaustive search of parses. In order to find a few promising parses efficiently, we often take a strategy that an appropriate weight is assigned to each production and parses are looked up in the order of the total weight of all applied productions. If the assigned weight is selected carefully to have strong correlation to whether a parse is accepted or not, looking up parses in the order of the total weight is equivalent to enumeration of parses in the order of acceptability. For example, in the punctuation problem of Japanese sentences, the number of the phrases of the sentence is known to be an excellent candidate for the weight of parses. However, the algorithms proposed so far that took this strategy are applied only to the problems of the limited areas such as regular languages, and they are not applied to general context-free languages.

In this paper, we present an efficient enumeration algorithm based on this strategy, which can be applied to general context-free grammars. We introduce a data structure suitable for enumeration of parses named

a *parse graph*, and present how to construct a parse graph in section 3. With a parse graph, a path between two special vertex, some of whose arcs are replaced iteratively by the path denoted by their labels, represents a right parse of the parsed sentence. Because the length of paths represents the total weight of all applied productions for parses, enumeration of parses in the order of the total weight of all applied productions is equivalent to finding paths on the parse graph in the order of length. In section 4, we show the outline of how to enumerate the parses of the ambiguous sentence in the order of their weight, using the parse graph. We also discuss the time and space complexities of the algorithm in that section.

2 Context-Free Parsing Algorithm

Several general context-free parsing algorithms have been proposed so far, namely Cocke-Younger-Kasami algorithm[2, 3], Earley's algorithm[4], Valiant's algorithm[5], Graham-Harrison-Ruzzo algorithm[6, 8], and so forth. The features of these algorithms are the following. Cocke-Younger-Kasami algorithm (CYK algorithm for short) is a kind of the bottom up parsing algorithms, and has $O(n^3)$ time complexity, where n is the length of the sentence. In this algorithm, the grammar is required to be written in Chomsky normal form. Earley's algorithm is a kind of the top down parsing algorithms, and has $O(n^3)$ time complexity. By contrast with CYK algorithm, no special production form is required in Earley's algorithm. Valiant's algorithm and Graham-Harrison-Ruzzo algorithm (GHR algorithm for short) are the modified versions of CYK algorithm and Earley's algorithm, respectively. Both of them use the technique of matrix multiplication in order to reduce the time complexity, The time complexity of Valiant's algorithm is $O(n^{2.81})$ and that of GHR algorithm is $O(n^3/\log n)$. However, in both algorithms, the overhead for matrix multiplication is so large that these algorithms don't seem suitable for the practical use.

In this paper, we adopt Earley's algorithm as the base of our algorithm because of the following two reasons:

(1) No special production form is required.

(2) Earley's algorithm seems more suitable than Valiant's algorithm and GHR algorithm because the overhead of these two algorithms is quite large.

Let $G = (V_N, V_T, P, S)$ be a grammar, where V_N is the set of nonterminal symbols, V_T is the set of terminal symbols, P is the set of productions, and $S \in V_N$ is the start symbol. In Earley's algorithm, the *item lists* $I_0, I_1, \ldots, I_n, I_{n+1}$ are created, where n is the length of the parsed sentence. Each item list consists of several *items* $[A \rightarrow \alpha \cdot \beta\ (p), f]$, where $A \rightarrow \alpha\beta \in P$, p is the index number of the production, "$\cdot$" is the meta symbol that shows how much of the right side of the production has been recognized so far, and f is an integer which denotes the position in the input string at which we began to look for that instance of the production. The set of item lists $\{I_0, I_1, \ldots, I_n, I_{n+1}\}$ is called the *parse list*.

As for the time and space complexities for Earley's algorithm, the following are known[1].

(e-1) The time and space complexities for parsing a sentence by Earley's algorithm are $O(n^3)$ and $O(n^2)$, respectively, where n is the length of the parsed sentence.

(e-2) The time complexity for deriving a parse from the parse list is $O(n^2)$, where n is the length of the parsed sentence.

3 Parse Graphs

3.1 The features of parse graphs

The parse graph is a directed graph which consists of several connected components. Each connected component is called a *layer* of the parse graph. Each layer is an acyclic graph that has only one source, and it corresponds to either a nonterminal symbol or an integer. An layer corresponding to a nonterminal symbol has only one sink. With this graph, we can extract parses more efficiently than with a parse list of Earley's algorithm. As shown in the next section, a path between two special vertex, some of whose arcs are replaced iteratively by the path denoted by their labels, represents a right parse of the parsed sentence.

In the remainder of this paper, we use the following notations.

$L(f)$	The layer corresponding to an integer f.
$L(A)$	The layer corresponding to a nonterminal symbol A.
$L(v)$	The layer containing a vertex v.
$L(e)$	The layer containing an arc e.
$v_s(X)$	The source of the layer $L(X)$, where X is either an integer, a nonterminal symbol, a vertex, or an arc.
$v_t(A)$	The sink of the layer $L(A)$, where A is a nonterminal symbol. Note that the layer corresponding to a nonterminal symbol has only one sink.

In the parse graph, each arc has one of the following labels.

(1) An index number of the production p, which denotes the derivation by $A \rightarrow \alpha\ (p)$.

(2) A nonterminal symbol A, which denotes the derivation $A \stackrel{+}{\Rightarrow} \epsilon$.

(3) The index of a vertex v, which denotes the path from $v_s(v)$ to v.

When we describe the arc $e = (u, v)$ with its label of each kind, we use the notations $e(p)$, $e\langle A\rangle$, $e[v]$, or the alternative notations $(u, v; (p))$, $(u, v; \langle A\rangle)$, $(u, v; [v])$, respectively.

Instead of an item of the form $[A \rightarrow \alpha\cdot\beta\ (p), f]$ in Earley's algorithm, we use the triplet $[A \rightarrow \alpha\cdot\beta\ (p), f, v]$ as an item of our algorithm for constructing a parse graph, where v is the index of a vertex.

For example, we parse the sentence xx of the grammar shown in Figure 1. The parse list and the parse graph generated from this sentence are shown in Figure 2 and Figure 3, respectively.

In Figure 3, the label "(2)" of the arc from vertex #8 to vertex #9 indicates the derivation by $S \rightarrow SJ\ (2)$, the label "$\langle S\rangle$" of the arc from vertex #0 to vertex #1 indicates the derivation $S \stackrel{+}{\Rightarrow} \epsilon$, and the label "[7]" of the arc from vertex #2 to vertex #8 indicates the paths from vertex #0 to vertex #7.

3.2 An algorithm for constructing a parse graph

Our algorithm for constructing a parse graph is based on Earley's algorithm. In Earley's algorithm, one of three operations is performed on each item, depending on its form, to add more items to the item lists. In our algorithm, these operations not only add more items to item lists but also add new vertices and arcs to the parse graph, shown as follows.

$$\begin{array}{ll} S \rightarrow \epsilon & (1) \\ S \rightarrow SJ & (2) \\ J \rightarrow F & (3) \\ J \rightarrow I & (4) \\ F \rightarrow x & (5) \\ I \rightarrow x & (6) \end{array}$$

Figure 1: An ambiguous context-free grammar

I_0 :

Item	Production	Origin	Vertex
$[\ S' \rightarrow \cdot S\$$	(0),	0,	$0\ ]$
$[\ S \rightarrow \cdot$	(1),	0,	$0\ ]$
$[\ S \rightarrow \cdot SJ$	(0),	0,	$0\ ]$
$[\ S' \rightarrow S \cdot \$$	(0),	0,	$1\ ]$
$[\ S \rightarrow S \cdot J$	(2),	0,	$2\ ]$
$[\ J \rightarrow \cdot F$	(3),	0,	$0\ ]$
$[\ J \rightarrow \cdot I$	(4),	0,	$0\ ]$
$[\ F \rightarrow \cdot x$	(5),	0,	$0\ ]$
$[\ I \rightarrow \cdot x$	(6),	0,	$0\ ]$

I_1 :

Item	Production	Origin	Vertex
$[\ F \rightarrow x \cdot$	(5),	0,	$0\ ]$
$[\ I \rightarrow x \cdot$	(6),	0,	$0\ ]$
$[\ J \rightarrow F \cdot$	(3),	0,	$4\ ]$
$[\ J \rightarrow I \cdot$	(4),	0,	$6\ ]$
$[\ S \rightarrow SJ \cdot$	(2),	0,	$8\ ]$
$[\ S' \rightarrow S \cdot \$$	(0),	0,	$10\ ]$
$[\ S \rightarrow S \cdot J$	(2),	0,	$11\ ]$
$[\ J \rightarrow \cdot F$	(3),	1,	$12\ ]$
$[\ J \rightarrow \cdot I$	(4),	1,	$12\ ]$
$[\ F \rightarrow \cdot x$	(5),	1,	$12\ ]$
$[\ I \rightarrow \cdot x$	(6),	1,	$12\ ]$

I_2 :

Item	Production	Origin	Vertex
$[\ F \rightarrow x \cdot$	(5),	1,	$12\ ]$
$[\ I \rightarrow x \cdot$	(6),	1,	$12\ ]$
$[\ J \rightarrow F \cdot$	(3),	1,	$14\ ]$
$[\ J \rightarrow I \cdot$	(4),	1,	$16\ ]$
$[\ S \rightarrow SJ \cdot$	(2),	0,	$18\ ]$
$[\ S' \rightarrow S \cdot \$$	(0),	0,	$20\ ]$
$[\ S \rightarrow S \cdot J$	(2),	0,	$21\ ]$
$[\ J \rightarrow \cdot F$	(3),	2,	$22\ ]$
$[\ J \rightarrow \cdot I$	(4),	2,	$22\ ]$
$[\ F \rightarrow \cdot x$	(5),	2,	$22\ ]$
$[\ I \rightarrow \cdot x$	(6),	2,	$22\ ]$

I_3 :

Item	Production	Origin	Vertex
$[\ S' \rightarrow S\$ \cdot$	(0),	0,	$20\ ]$

Figure 2: A parse list for the sentence xx of the grammar in Figure 1

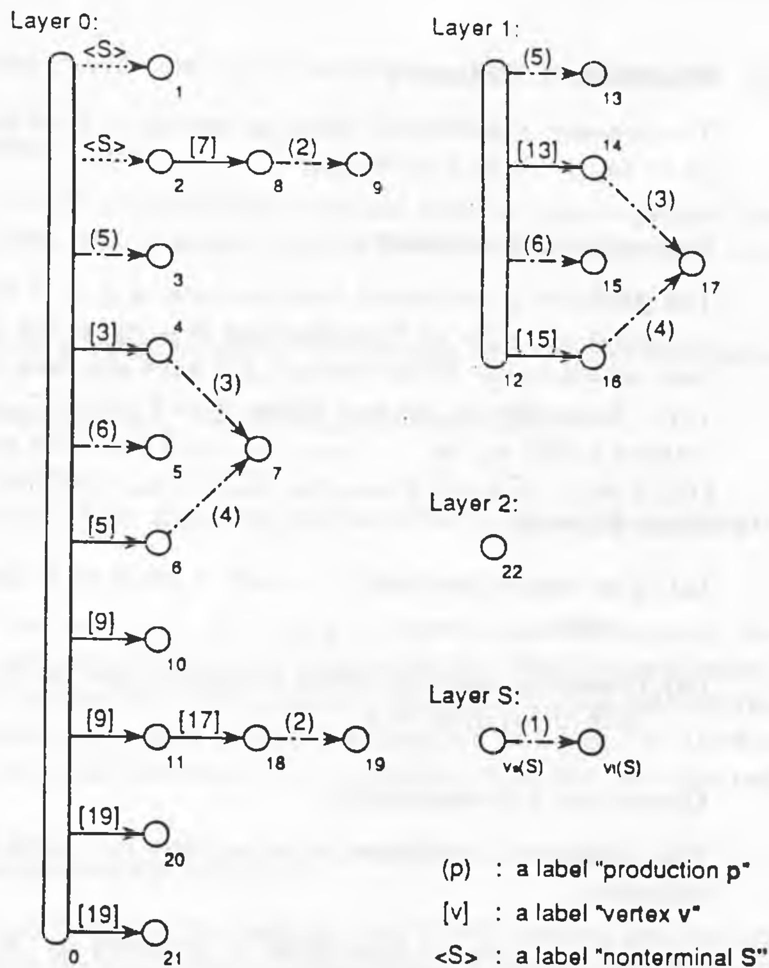

Figure 3: A parse graph for the sentence xx of the grammar in Figure 1

Operation 1 (scanner)

The *scanner* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot a_{j+1}\beta\ (p), f, v]$. It puts the item $[A \rightarrow \alpha a_{j+1} \cdot \beta\ (p), f, v]$ to I_{j+1}.

Operation 2 (predictor)

The *predictor* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot B\beta\ (p), f, v]$. It adds items $[B \rightarrow \cdot\gamma_k\ (p_k), j, v_s(j)]$ for all B-productions $B \rightarrow \gamma_k\ (p_k)$ to I_j, except in the case where these items have already been added to I_j. If the vertex $v_s(j)$ have not been created yet, the predictor creates $v_s(j)$ to the layer $L(j)$. Especially, in the case where $B \Rightarrow C_1C_2\cdots C_m \overset{*}{\Rightarrow} \epsilon$ and $C_1C_2\cdots C_m \in V_N^*$, the predictor adds the vertices $v_s(B)$, v_1, v_2, ..., v_{m-1}, v_m, $v_t(B)$, and the arcs $(v_s(B), v_1; \langle C_1\rangle)$, $(v_1, v_2; \langle C_2\rangle)$, $(v_2, v_3; \langle C_3\rangle)$, ..., $(v_{m-2}, v_{m-1}; \langle C_{m-1}\rangle)$, $(v_{m-1}, v_m; \langle C_m\rangle)$, $(v_m, v_t(B); (p))$ to $L(B)$ if they are not in $L(B)$, and performs one of the following:

(a) If an item of the form $[A \rightarrow \alpha B \cdot \beta\ (p), f, w]$ is already in I_j, then add the arc $(v, w; \langle B\rangle)$ to the parse graph.

(b) Otherwise, add the vertex w and the arc $(v, w; \langle B\rangle)$ to the parse graph, and add the item $[A \rightarrow \alpha B \cdot \beta\ (p), f, w]$ to I_j.

Operation 3 (completer)

The *completer* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot\ (p), f, v]$. It performs one of the following:

(a) If $f = j$, then the item would be processed by the predictor. Therefore, the completer does nothing.

(b) If $f \neq j$, and there exists an item of the form $[A \rightarrow \beta \cdot\ (q), f, u]$ ($p \neq q$, $u \neq v$) in I_j, and the arc $(u, w; (q))$ in the parse graph, then add the arc $(v, w; (p))$ to the parse graph.

(c) Otherwise, add a new vertex x and a new arc $(v, x; (p))$ to the parse graph. Furthermore, for all items of the form $[B_k \rightarrow \gamma_k \cdot A\delta_k\ (p_k), f_k, u_k]$ in I_f, perform one of the following:

  (c-1) If there exists an item of the form $[B_k \rightarrow \gamma_k A \cdot \delta_k\ (p_k), f_k, v_k]$ in I_j where $u_k \neq v_k$, then add a new arc $(u_k, v_k; [x])$ to the parse graph.

  (c-2) Otherwise, add a new vertex v_k and a new arc $(u_k, v_k; [x])$ to the parse graph, and add a new item $[B_k \rightarrow \gamma_k A \cdot \delta_k\ (p_k), f_k, v_k]$ to I_j.

We describe our algorithm for constructing a parse graph as follows:

Algorithm 1. An algorithm for constructing a parse graph

A context-free grammar $G = (V_N, V_T, P, S)$ and a sentence $a_1a_2\cdots a_n$ are given.

[**step 1**] Add the meta symbol "\$" to the tail of the sentence. Add the production $S' \rightarrow S\$\ (0)$ to P. Create the parse graph consisting of $v_s(0)$. Create the item list I_0 consisting of $[S' \rightarrow \cdot S\$\ (0), v_s(0)]$.

[**step 2**] Create the item lists $I_0, I_1, \ldots, I_{n+1}$ in order, by performing the following operations from $k = 1$ to $k = n$.

(1) Perform the predictor or the completer to add items to the item list I_k, until no more items can be added to I_k.

(2) Then, perform the scanner to add items to I_{k+1}.

[step 3] If I_{n+1} has an item of the form $[S' \rightarrow S\$ \cdot \ (0), v]$, then it means that the parser accepts the sentence, and the algorithm terminates. Otherwise, it means that the parser rejects the sentence, and the algorithm terminates.

Note that this algorithm is the same as Earley's algorithm except the portion for constructing a parse graph.

As for the time and space complexities of this algorithm, the following theorem holds.

Theorem 1. The time and space complexities of our algorithm are both $O(n^3)$, where n is the length of the sentence.

(proof) Consider the number of items in the item lists. According to three operations, namely the scanner, the predictor and the completer, each item list does not have items such that their first and second components are the same. Therefore, each item list has $O(n)$ items, because the number of the kinds of the first component is constant, and that of the second component is not more than $n + 2$. Hence, the number of items of the parse list is $O(n^2)$, because the parse list consists of $n + 2$ item lists. Consider the time and space complexities of the operations per item.

(1) As for the scanner, the time and space complexities are both $O(1)$.

(2) As for the predictor, at most $O(|P|)$ items are added to the item list, and $O(|P|)$ vertices and arcs are added to the parse graph, where $|P|$ denotes the number of the productions. Therefore, the time and space complexities are both $O(|P|) = O(1)$.

(3) As for the completer, if the second component of the performed item is f, the completer scans all items in I_f, adds at most $O(n)$ items to the item list, and adds at most $O(n)$ vertices and arcs to the parse graph. Therefore, the time and space complexities are both $O(n)$.

Consequently, the time and space complexities of the operations per item is $O(n)$. Therefore, the time and space complexities of the parse graph construction algorithm are both $O(n^3)$. □

Compared with (e–1) in section 2, the time complexity for constructing a parse graph is the same as Earley's algorithm, but the space complexity is worse because the number of arcs in a parse graph is $O(n^3)$.

4 Enumeration of Parses

4.1 Extracting parses

In order to extract parses from a parse graph, we introduce a *traversal paths* of a parse graph. The notation $\pi(u, v)$ represents traversal paths from u to v.

A *traversal path* from a vertex u to a vertex v is defined as follows provided that $L(u) = L(v)$.

(1) A null sequence is defined as a traversal path if $u = v$.

(2) The sequence of the arcs $e_1e_2\cdots e_n$, where $e_i = (u_i, v_i)$, is defined as a traversal path if $u = u_1, v_1 = u_2, v_2 = u_3, \ldots, v_{n-1} = u_n, v_n = v$.

(3) Let $e_1e_2\cdots e_n$ be a traversal path from u to v, in which an arc e_i is labeled with a nonterminal symbol A. The sequence of the arc $e_1\cdots e_{i-1}\pi(v_s(A), v_t(A))e_{i+1}\cdots e_n$ in which $e_i\langle A\rangle$ is replaced by a traversal path $\pi(v_s(A), v_t(A))$ is defined as a traversal path.

(4) Let $e_1e_2\cdots e_n$ be a traversal path from u to v, in which an arc e_i is labeled with the index of a vertex v. The sequence of the arc $e_1\cdots e_{i-1}\pi(v_s(v), v)e_{i+1}\cdots e_n$ in which $e_i[v]$ is replaced by a traversal path $\pi(v_s(v), v)$ is defined as a traversal path.

Especially, the traversal path that has only the arcs labeled with the index of the production is called a *proper traversal path*. The notation $\pi^*(u, v)$ represents proper traversal paths from u to v. This notation is also used to represent the sequence of the labels of the proper traversal paths.

As for the relationship between proper traversal paths and parses, the following theorem holds.

Theorem 2. If there exist two items $[A \rightarrow \alpha \cdot \beta\gamma\ (p), f, u] \in I_j$, $[A \rightarrow \alpha\beta \cdot \gamma\ (p), f, v] \in I_k$, where $\alpha, \beta, \gamma \in V^*$, the sequence of the labels $\pi^*(u, v)$ is the reverse order of the sequence of the production numbers used for the rightmost derivation $\beta \overset{*}{\underset{rm}{\Rightarrow}} a_{j+1}\cdots a_k$.

(proof) It is easy to prove this theorem by induction on the length of the derivation sequence. □

Let $v_t(0)$ be the third component v of the item $[S' \rightarrow S\$\cdot\ (0), 0, v] \in I_{n+1}$. According to theorem 2, the sequences of the labels $\pi^*(v_s(0), v_t(0))$ represent the right parses of the parsed sentence. An example of a proper traversal path of the parse graph in Figure 3 is shown in Figure 4, where $v_s(0)$ is vertex #0 and $v_t(0)$ is vertex #20.

A right parse can be extracted from the parse graph by searching a proper traversal path from $v_t(0)$ toward $v_s(0)$. This extraction can be done without backtracking, because each layer has only one source. Therefore, the following theorem holds.

Theorem 3. If the given grammar is cycle-free, the time complexity for extracting a parse is $O(n)$, where n is the length of the sentence.

(proof) If the grammar is cycle-free, the length of the parse is $O(n)$. Therefore, the time complexity is $O(n)$. □

Compared with (e–2) in section 2, the time complexity for extracting a parse is better than Earley's algorithm.

4.2 An algorithm for parse enumeration

Using a parse graph, enumeration of the parses in the order of the total weight is equivalent to enumeration of the proper traversal paths from $v_s(0)$ to $v_t(0)$ in the order of the length. While many researchers have developed the algorithms for finding the k shortest paths[9, 10, 11, 12, 13], we apply one of them developed by Katoh, Ibaraki and Mine[10] to the parse graph recursively. Because of the lack of the space, we explain

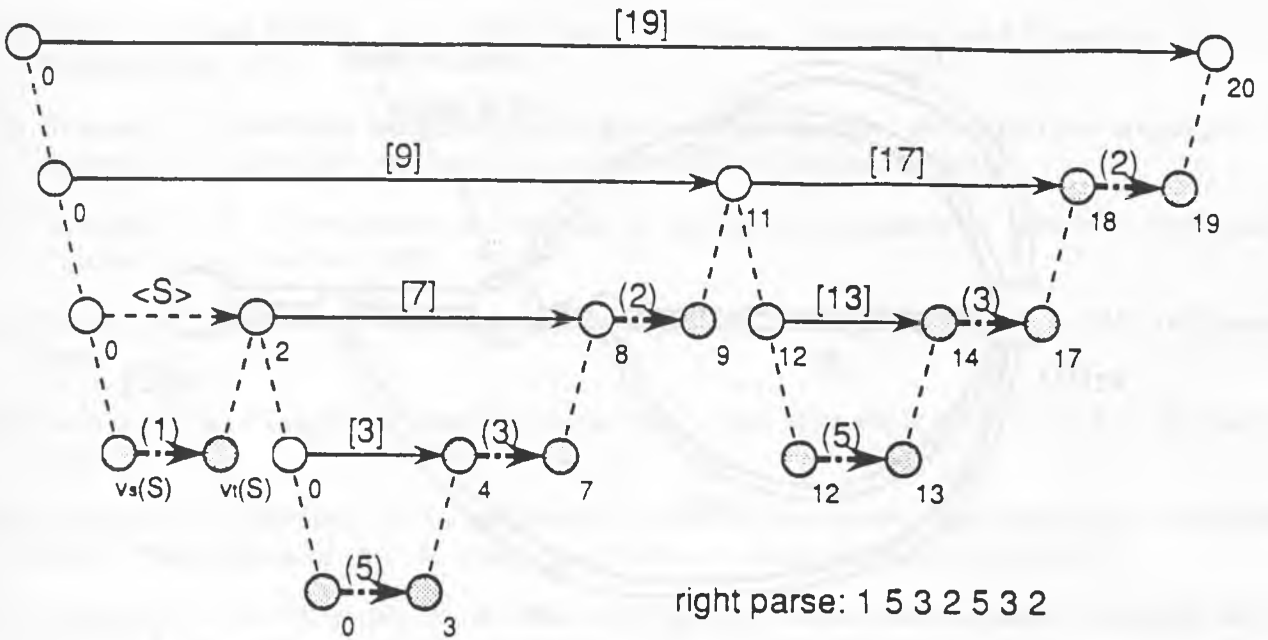

Figure 4: A proper traversal path from vertex #0 to vertex #20

only the outline of the algorithm. The details of the algorithm are described in [14]. In the following discussion, the k-th shortest traversal path from $v_s(0)$ to $v_t(0)$ is referred to as π^k.

First of all, derive the *shortest path tree* for $v_s(0)$, denoted as $T(v_s(0))$, which consists of the arcs of the shortest paths from $v_s(0)$ to all other vertices. The shortest path tree can easily be derived in the algorithm for constructing a parse graph. π^1 can be extracted from $T(v_s(0))$. π^2 consists of the path of $T(v_s(0))$ from $v_s(0)$ to a vertex u, the arc (u, v) where v is one of the vertices on π^1, and the subpath of π^1 from v to $v_t(0)$. Therefore, the number of the candidates of π^2 is the same as the sum of the in-degree of all vertices on the shortest path. As for the parse graph, the length of the shortest path and the in-degree of a vertex are both $O(n)$[14], and hence we can derive π^2 in $O(n^2)$. In order to derive π^3, all paths from $v_s(0)$ to $v_t(0)$ except π^1 and π^2 are divided into three sets as follows (see Figure 5):

(1) The set of paths that join the subpath common to π^1 and π^2. The shortest path in this set is referred to as π_a.

(2) The set of paths that join π^1, and contain the subpath common to π^1 and π^2 as their final subpath. The shortest path in this set is referred to as π_b.

(4) The set of paths that join π^2, and contain the subpath common to π^1 and π^2 as their final subpath. The shortest path in this set is referred to as π_c.

π_a, π_b, and π_c can be derived in the same manner as deriving π^2 in $O(n^2)$, respectively. π^3 is the shortest one of π_a, π_b, and π_c, and the rest of these paths are stored in another set as the candidates of π^4. $\pi^4, \pi^5, \ldots$ are derived by repeating the similar calculation. Therefore, the time and space complexities of the enumeration of the k shortest paths are $O(n^3 + kn^2)$ and $O(n^2 + kn)$, respectively.

In the above discussion, the k shortest paths are derived. However, we can also derive the k longest paths in the same manner.

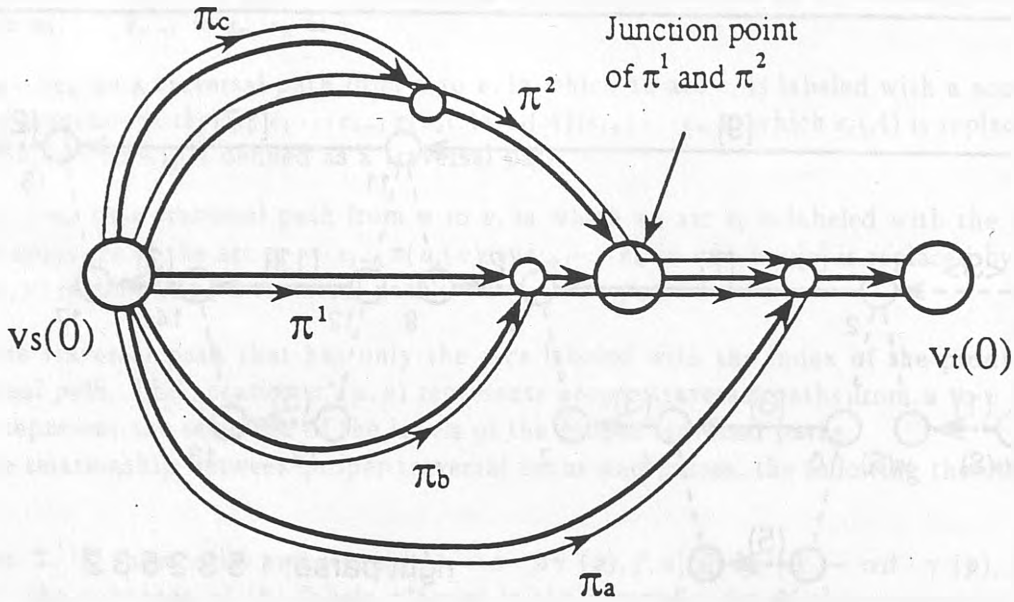

Figure 5: The relation among π^1, π^2, and π^3

Table 1: The time and space complexities of our algorithms (n:the length of the sentence)

Complexity	Construction of parse graph	Enumeration of k parses
Time	$O(n^3)$	$O(n^3 + kn^2)$
Space	$O(n^3)$	$O(n^2 + kn)$

We summarize the time and space complexities of our algorithms in Table 1.

5 Conclusion

In this paper, we have presented an algorithm for the enumeration of the parses in the order of the acceptability. This algorithm can be applied to the general context-free languages. In order to enumerate parses efficiently, we have introduced a data structure suitable for the enumeration called the parse graph. Using a parse graph, we can enumerate k parses in the order of acceptability efficiently in $O(n^3 + kn^2)$.

Acknowledgement

We appreciate Prof. Isao Shirakawa and Prof. Hideo Miyahara of Osaka University for their helpful support. The first author thanks Prof. Toshiro Araki, Dr. Hiroshi Deguchi, Dr. Shinji Shimojo of Osaka University, and Mr. Toshiyuki Masui of SHARP Corporation for their helpful suggestions and comments on this paper.

References

[1] Aho, A. V. and Ullman, J. D., *The Theory of Parsing, Translation, and Compiling, Vol.1:Parsing*, Prentice-Hall, 1972.

[2] Kasami, T., "An efficient recognition and syntax analysis algorithm: for context-free languages", *Science Report*, AF CRL-65-758, Air Force Cambridge Research Laboratory, 1965.

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

[4] Earley, J., "An efficient context-free parsing algorithm", *Communication of A.C.M.*, 13-2, pp.94–102, 1970.

[5] Valiant, L. G., "General context-free recognition in less than cubic time", *J.C.S.S.*, 10, pp.308–315, 1975.

[6] Graham, S. L., Harrison, M. A., and Ruzzo, W. L., "On line context-free recognition in less than cubic time", *Proc. 8th Annu. A.C.M. Symp. on Theory of Computing*, pp.112–120, 1976.

[7] Graham, S. L., and Harrison, M. A., "Parsing of general context-free languages", *Advances in Computers*, 14, Academic Press, pp.415–462, 1976.

[8] Graham, S. L., and Harrison, M. A., "An improved context-free recognizer", *A.C.M. Trans. on Programming Languages and Systems*, 2-3, pp.415–462, 1980.

[9] Yen, J. Y., "Finding the K shortest loopless paths in a network", *Management Science*, 17, pp.712–716, 1971.

[10] Katoh, N., Ibaraki, T., and Mine, H., "An efficient algorithm for K shortest simple paths", *Networks*, 12, pp.411–427, 1982.

[11] Fox, B. L., "Data structures and computer science techniques in operations research", *Operations Research*, 26, pp.686–717, 1978.

[12] Denardo, E. V., and Fox, B. L., "Shortest-route methods: 1. reaching, pruning and buckets", *Operations Research*, 27, pp.161–186, 1979.

[13] Lawer, E. L., "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem", *Management Science*, 18, pp.401–405, 1972.

[14] Yamai, N., "A study for parsing of ambiguous languages using hierarchical graph representation of all derivations", *Master Thesis of Osaka University*, 1986 (in Japanese).