SIAM J. COMPUT.
Vol. 1, No. 4, December 1972

A MINIMUM DISTANCE ERROR-CORRECTING PARSER FOR CONTEXT-FREE LANGUAGES*

ALFRED V. AHO† AND THOMAS G. PETERSON‡

Abstract. We assume three types of syntax errors can debase the sentences of a language generated by a context-free grammar: the replacement of a symbol by an incorrect symbol, the insertion of an extraneous symbol, or the deletion of a symbol. We present an algorithm that will parse any input string to completion finding the fewest possible number of errors. On a random access computer the algorithm requires time proportional to the cube of the length of the input.

Key words. Syntax error, error correction, parsing, context-free grammar, computational complexity

1. Introduction.

What should a compiler do when it discovers an error in the source program? Surprisingly, much of the literature published to date on compiler design has not adequately answered this question. Even in the area of parser design, relatively few papers have considered parsing algorithms that recover gracefully from syntax errors [3]–[8], [10]. Many published parsing algorithms call for a parser merely to halt and report error on encountering the first syntax error.

In this paper we present a parsing algorithm for context-free grammars that will parse any input string to completion finding the fewest possible number of syntax errors. On a random access computer the algorithm takes time proportional to the cube of the length of the input. Although this is the fastest known minimum distance error-correcting parsing algorithm for context-free grammars, the time required is probably excessive for most compiler applications. Nevertheless, we feel the algorithm can serve as the yardstick against which the error-detecting and correcting capabilities of other parsing algorithms can be evaluated.

2. Syntax errors.

Let L be a nonempty set of strings over some finite alphabet Σ. We assume that a string not in L is derived from some sentence in L by a sequence of error transformations. In this paper we assume the following three types of syntax errors are possible:

(i) the replacement of a symbol by another symbol,
(ii) the insertion of an extraneous symbol, and
(iii) the deletion of a symbol.

We shall describe these errors in terms of three transformations T_R, T_I and T_D, respectively, from Σ^* to the subsets of Σ^* defined as follows. For x and y in Σ^*:

(i) xby is in $T_R(xay)$ for all $a \neq b$ in Σ.
(ii) xay is in $T_I(xy)$ for all a in Σ.
(iii) xy is in $T_D(xay)$ for all a in Σ.

We write $x \vdash y$ if y is in $T_i(x)$ for some i in $\{R, I, D\}$. Note that $\vdash$ is symmetric.

If ρ is a binary relation, ρ^k will denote the composition of ρ with itself k times, ρ^0 the identity relation, and ρ^* the reflexive and transitive closure of ρ.

* Received by the editors June 6, 1972, and in revised form October 5, 1972.

† Bell Laboratories, Murray Hill, New Jersey 07974.

‡ Bell Laboratories, Whippany, New Jersey 07981.

We define a Hamming distance on strings in Σ^* by letting $d(x, y)$ be the smallest integer k for which $x \overset{k}{\vdash} y$.

Suppose L is a language over alphabet Σ. We define $E_k(L)$, the *set of strings in* Σ^* *with k errors*, as follows:

(i) $E_0(L) = L$.

(ii) $E_k(L) = \{x \text{ in } \Sigma^* |$ there exists a string w in $E_{k-1}(L)$ such that $w \vdash x$ and x is not in $E_{k-l}(L)$ for $1 \leqq l \leqq k\}$.

A string x is in $E_k(L)$ if and only if there is a sentence w in L such that w is distance k from x. Thus if $x \in E_k(L)$ and $x \vdash y$, then $y \in E_i(L)$ for some i such that $k - 1 \leqq i \leqq k + 1$.

Given a string x in $E_k(L)$ there is a sequence of intermediate strings $w_0, w_1, \cdots, w_k$ such that $w_i \in E_i(L)$ for $0 \leqq i \leqq k$ and $w_k = x$. A sequence of error transformations used to derive w_i from w_{i-1} for $1 \leqq i \leqq k$ will define a set of k errors in x.

For example, suppose $L = \{abc\}$. Then given the string $bbdc$, we can say the first b is a replacement error and the d is an insertion error because $abc \underset{T_R}{\vdash} bbc \underset{T_I}{\vdash} bbdc$.

Note that for an x in $E_k(L)$ there can be many different sequences of intermediate strings and transformations as above, so that in general the errors in x are not unique. If desired, uniqueness can be achieved by some lexicographic conventions regarding how the sequence of intermediate strings and error transformations are to be chosen.

3. Context-free grammars. A *context-free grammar* (grammar for short) is a 4-tuple $G = (N, \Sigma, P, S)$, where

(i) N and Σ are finite disjoint alphabets of *nonterminals* and *terminals*, respectively.

(ii) P is a finite set of productions of the form $A \to \alpha$, where A is in N and α is in $(N \cup \Sigma)^*$.

(iii) S is a distinguished symbol in N.

If $A \to \alpha$ is in P, we write $\beta A \gamma \underset{G}{\Rightarrow} \beta\alpha\gamma$ for all β and γ in $(N \cup \Sigma)^*$. If γ is in Σ^*, we also write $\beta A \gamma \underset{G,rm}{\Rightarrow} \beta\alpha\gamma$ to denote a right-most derivation. Similarly, if β is in Σ^*, we write $\beta A \gamma \underset{G,lm}{\Rightarrow} \beta\alpha\gamma$ to denote a left-most derivation. We shall drop the subscript G from $\Rightarrow$, $\underset{lm}{\Rightarrow}$, $\underset{rm}{\Rightarrow}$ whenever possible.

The language *generated by* G, denoted $L(G)$, is the set $\{w \text{ in } \Sigma^* | S \overset{*}{\Rightarrow} w\}$. If w is in $L(G)$, then there exists a sequence of strings in $(N \cup \Sigma)^*$, $\alpha_0, \alpha_1, \cdots, \alpha_n$, such that $\alpha_0 = S$, $\alpha_{i-1} \underset{rm}{\Rightarrow} \alpha_i$ for $1 \leqq i \leqq n$ and $\alpha_n = w$. Suppose production p_i in P is used to derive α_i from α_{i-1} for $1 \leqq i \leqq n$. The sequence of productions $\pi = p_1p_2 \cdots p_n$ is called a *parse of w according to G* and we shall write $S \overset{\pi}{\Rightarrow} w$ to denote the right-most derivation using this sequence of productions.

For the remainder of this paper we shall make the following assumptions about a grammar $G = (N, \Sigma, P, S)$:

(i) We assume $L(G) \neq \emptyset$.

(ii) We assume G contains no useless symbols. That is, for each X in $N \cup \Sigma$ there is a derivation of the form $S \overset{*}{\Rightarrow} wXy \overset{*}{\Rightarrow} wxy$, where w, x and y are in Σ^*.

The concept of one grammar covering another grammar will be useful. Let $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and $G_2 = (N_2, \Sigma_2, P_2, S_2)$ be two grammars such that $L(G_1) \subseteq L(G_2)$. We say G_2 *covers* G_1 if there is a homomorphism h from P_2 to P_1 such that if x is in $L(G_1)$ and π is a parse of w according to G_1, then there is a parse π' of w according to G_2 such that $h(\pi') = \pi$.

4. Error-correcting parser. Our problem can be stated as follows: Given a grammar $G = (N, \Sigma, P, S)$, we want an algorithm that takes as input any string x in Σ^* and produces as output a parse for some string w in $L(G)$ such that the distance between w and x is as small as possible. (If x is in $L(G)$, then clearly we will produce a parse for x.) An algorithm of this nature will be called an *error-correcting parser* for G.

We can design such an error-correcting parser for G as follows. First we add to G a set of error productions to obtain a covering grammar G' such that $L(G') = \Sigma^*$. Then we design a parser for G' that uses as few error productions as possible in parsing an input string x. The error productions used in a derivation of w according to G' will indicate the positions and types of the errors in w.

5. Error productions. The following algorithm will add error productions to the grammar $G = (N, \Sigma, P, S)$ such that the extended grammar covers G and generates Σ^*.

ALGORITHM 1. Let $G = (N, \Sigma, P, S)$ be a context-free grammar. From P construct a new set of productions P' as follows:

1. If $A \to \alpha_0 b_1 \alpha_1 b_2 \alpha_2 \cdots b_m \alpha_m$, $m \geqq 0$, is a production in P such that α_i is in N^* and b_i is in Σ, then add the production $A \to \alpha_0 E_{b_1} \alpha_1 E_{b_2} \alpha_2 \cdots E_{b_m} \alpha_m$ to P', where each E_{b_i} is a new nonterminal.

2. For all a in Σ add to P' the productions
(a) $E_a \to a$,
(b) $E_a \to b$ for all b in Σ, $b \neq a$,
(c) $E_a \to Ha$,
(d) $I \to a$,
(e) $E_a \to e$, where e is the empty string.
Here H and I are new nonterminals.

3. Add to P' the productions
(a) $S' \to S$,
(b) $S' \to SH$,
(c) $H \to HI$,
(d) $H \to I$.
Here S' is a new start symbol.

Let $G = (N', \Sigma', P', S')$, where $N' = N \cup \{S', H, I\} \cup \{E_a | a \in \Sigma\}$. We shall call G' the *covering grammar* for G. □

In P' a production of the form $E_a \to b$, $E_a \to e$ or $I \to a$ is called a *terminal error production*. The production $E_a \to b$ introduces a replacement error. $E_a \to e$ introduces a deletion error. $I \to a$ introduces one insertion error. Since H can derive any nonempty string, the production $E_a \to Ha$ introduces a sequence of one or more insertion errors in front of a. To place one or more insertion errors at the end of a sentence we apply the production $S' \to SH$.

To see that G' covers G we note that by using the productions added to P' in step 1, we have $S \overset{*}{\underset{G}{\Rightarrow}} a_1 a_2 \cdots a_n$ if and only if $S \overset{*}{\underset{G'}{\Rightarrow}} E_{a_1} E_{a_2} \cdots E_{a_n}$. Using the productions $E_a \to a$, we have $E_{a_1}E_{a_2}\cdots E_{a_n} \overset{*}{\Rightarrow} a_1a_2\cdots a_n$. Thus for each w in $L(G)$, there is a parse of w according to G' from which the parse of w according to G can be recovered by means of a homomorphism.

Using the productions added to P' in steps 2 and 3, E_a can derive any terminal symbol $b \neq a$, or any terminal string ending in a, or the empty string. Moreover, $S' \to SH$ is also a production and H derives any nonempty terminal string. Thus, since $L(G)$ is presumed to be nonempty, $L(G') = \Sigma^*$.

The grammar G' is ambiguous. We will parse an input string according to G' but because of the following theorem we can try to use as few terminal error productions as possible.

THEOREM 1. *x is in $E_k(L(G))$ if and only if there is a derivation of x in G' using k terminal error productions and no derivation using fewer than k terminal error productions.*

Proof. Let h be the homomorphism from P' to P such that $h(p') = p$ if p' is a production added to P' in step 1 of Algorithm 1 and $h(p') = e$ otherwise. To prove the theorem we shall prove the following statement by induction on k.

(*) $S \underset{G}{\overset{\pi}{\Rightarrow}} w$, $x \in E_k(L(G))$ and $w \overset{k}{\vdash} x$ if and only if $S' \underset{G'}{\overset{\pi'}{\Rightarrow}} x$, $h(\pi') = \pi$, π' contains k terminal error productions and there is no π'' with fewer terminal error productions such that $S' \underset{G'}{\overset{\pi''}{\Rightarrow}} x$.

Basis: If $k = 0$, π' will contain only productions from step 1 and step 2(a). Thus (*) is trivially true.

Inductive step: Suppose $S \underset{G}{\overset{\pi}{\Rightarrow}} w$, $y \in E_{k+1}(L(G))$ and $w \overset{k+1}{\vdash} y$. Then there is an x in $E_k(L(G))$ such that $w \overset{k}{\vdash} x \vdash y$. From the inductive hypothesis, there is a derivation $S' \underset{G'}{\overset{\pi'}{\Rightarrow}} x$ such that $h(\pi') = \pi$ and π' contains k terminal error productions and there is no derivation of x using fewer error productions. If $x \vdash y$ by T_R such that the ith symbol in x, say a, is replaced by an erroneous symbol, say b, then we can modify the derivation $S' \underset{G'}{\overset{\pi'}{\Rightarrow}} x$ replacing the production of the form $E_a \to a$ that is used to derive the ith terminal symbol[1] in x by the derivation $E_a \Rightarrow b$. Thus, $S' \underset{G'}{\overset{\pi''}{\Rightarrow}} y$, where $h(\pi'') = \pi$ and π'' contains $k + 1$ terminal error productions. Similar arguments can be made if $x \vdash y$ using transformation T_I or T_D.

Now suppose that there is a derivation $S' \underset{G'}{\overset{\rho}{\Rightarrow}} y$ such that $h(\rho) = \pi$ and ρ uses $l < k + 1$ terminal error productions. Clearly $l > 0$. Let the first terminal error production in ρ be $E_a \to b$. We could replace this production by $E_a \to a$ to obtain a parse ρ' such that $S' \underset{G'}{\overset{\rho'}{\Rightarrow}} x'$, $h(\rho') = \pi$ and ρ' contains $l - 1$ terminal error productions. But then by the inductive hypothesis x' would be in $E_{l-1}(L(G))$. However, this is impossible since $x' \vdash y$ and y is in $E_{k+1}(L(G))$. A similar argument prevails if the first terminal error production in ρ is $I \to a$ or $E_a \to e$.

The proof of the converse is straightforward.

6. Minimum distance parsing.

We shall now describe a parsing algorithm for G' that uses as few terminal error productions as possible. This algorithm is

[1] Note that if $x \in E_k(L(G))$, $y \in E_{k+1}(L(G))$ and $x \vdash y$ by a replacement error in position i of x, then the ith symbol in x cannot be a replacement error.

essentially Earley's [2] algorithm (without look ahead) with a provision added to keep count of the number of terminal error productions used. We employ the notation used in [1] to describe Earley's algorithm.

Informally our algorithm works as follows. Let G' be the covering grammar for G and let $x = a_1a_2 \cdots a_n$ be the input string to be parsed. We construct a sequence $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$ of lists of items. An item is an object of the form

$$[A \to \alpha \cdot \beta, i, k],$$

where

(i) $A \to \alpha\beta$ is a production in G';

(ii) $\cdot$ is a special metasymbol, indicating how much of the production is currently applicable to a parse;

(iii) i is an integer, $0 \leqq i \leqq n$, indicating the input position at which a derivation from α began.

(iv) k is a nonnegative integer indicating the number of terminal error productions used in the derivation from α.

Item $[A \to \alpha \cdot \beta, i, k]$ will be on list $\mathscr{I}_j$ if and only if for some γ in $(N \cup \Sigma)^*$,

$$(1) \qquad S' \overset{*}{\underset{G'}{\Rightarrow}} a_1a_2 \cdots a_iA\gamma,$$

$$(2) \qquad \alpha \overset{*}{\underset{G'}{\Rightarrow}} a_{i+1} \cdots a_j$$

such that derivation (2) uses k terminal error productions and there is no derivation of $a_{i+j} \cdots a_j$ from α using fewer terminal error productions.

Note that x is in $L(G)$ if and only if $[S' \to S \cdot, 0, 0]$ is in $\mathscr{I}_n$. From Theorem 1, x is in $E_k(L(G))$ if and only if an item of the form $[S' \to \alpha \cdot, 0, k]$ is in $\mathscr{I}_n$ and no item of the form $[S' \to \beta \cdot, 0, l]$ is in $\mathscr{I}_n$ for $l < k$.

The sequence $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$ will be called the sequence of *parse lists* for x. The following algorithm can be used to construct the parse lists for an input string.

ALGORITHM 2.

Input: The covering grammar $G' = (N', \Sigma', P', S')$ and an input string $x = a_1a_2 \cdots a_n$ in Σ^*.

Output: $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$, the parse lists for x.

Method: Initially, all lists are empty. Construct $\mathscr{I}_0$ as follows:

1. Add items $[S' \to \cdot S, 0, 0]$ and $[S' \to \cdot SH, 0, 0]$ to $\mathscr{I}_0$.

2. If $[A \to \alpha \cdot B\beta, 0, k]$ is in $\mathscr{I}_0$ and $B \to \gamma$ is a production in P', then add item $[B \to \cdot \gamma, 0, 0]$ to $\mathscr{I}_0$.

3. If $[A \to \alpha \cdot B\gamma, 0, k]$ and $[B \to \beta \cdot . 0, l]$ are in $\mathscr{I}_0$, then add $[A \to \alpha B \cdot \beta, 0, m]$ to $\mathscr{I}_0$, where $m = k + l + 1$ if $B \to \beta$ is a terminal error production and $m = k + l$ otherwise. Store with item $[A \to \alpha B \cdot \beta, 0, m]$ two pointers, the first to item $[A \to \alpha \cdot B\beta, 0, k]$, the second to item $[B \to \beta \cdot, 0, l]$. However, if $[A \to \alpha B \cdot \beta, 0, m']$ is already in $\mathscr{I}_0$ and $m' \leqq m$, then do not add $[A \to \alpha B \cdot \beta, 0, m]$ to $\mathscr{I}_0$. On the other hand, if $[A \to \alpha B \cdot \beta, 0, m'']$ is in $\mathscr{I}_0$ and $m'' > m$, then delete this item from $\mathscr{I}_0$.

4. Repeat steps 2 and 3 until no new items can be added to $\mathscr{I}_0$.

Suppose that $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_{j-1}$ have been constructed. We construct $\mathscr{I}_j$, $1 \leqq j \leqq n$, as follows:

5. For each item $[A \to \alpha \cdot a\beta, i, k]$ in $\mathscr{I}_{j-1}$ such that $a = a_j$, add $[A \to \alpha a \cdot \beta, i, k]$ to $\mathscr{I}_j$. Along with this item add a pointer to item $[A \to \alpha \cdot a\beta, i, k]$ in $\mathscr{I}_{j-1}$.

6. If $[B \to \gamma \cdot, i, k]$ is in $\mathscr{I}_j$ and $[A \to \alpha \cdot B\beta, h, l]$ is in $\mathscr{I}_i$, then add item $[A \to \alpha B \cdot \beta, h, m]$ to $\mathscr{I}_j$, where $m = k + l + 1$ if $B \to \gamma$ is a terminal error production and $m = k + l$ otherwise. Include with item $[A \to \alpha B \cdot \beta, h, m]$ two pointers, the first to item $[A \to \alpha \cdot B\beta, h, l]$ in $\mathscr{I}_i$ and the second to $[B \to \gamma \cdot, i, k]$ in $\mathscr{I}_j$.

However, if $[A \to \alpha B \cdot \beta, h, m']$ is already in $\mathscr{I}_j$ for some $m' \leqq m$, then do not add $[A \to \alpha B \cdot \beta, h, m]$ to $\mathscr{I}_j$. Likewise, if $[A \to \alpha B \cdot \beta, h, m'']$ is in $\mathscr{I}_j$ for some $m'' > m$, then delete this item from $\mathscr{I}_j$.

7. If $[A \to \alpha \cdot B\beta, i, k]$ is in $\mathscr{I}_j$ and $B \to \gamma$ is in P', then add $[B \to \cdot \gamma, j, 0]$ to $\mathscr{I}_j$.

8. Repeat steps 6 and 7 until no new items can be added to $\mathscr{I}_j$.

In this manner construct the sequence of parse lists $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$.

This algorithm is essentially Earley's algorithm with one additional field in each item to keep track of the number of terminal error productions used. We are only interested in derivations using as few terminal error productions as possible. The pointers stored with the items will be used to reconstruct a parse from the sequence of parse lists. The following lemmas describe the behavior of Algorithm 2.

Let $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$ be the parse lists for $x = a_1 a_2 \cdots a_n$ constructed by Algorithm 2.

LEMMA 1. *If* $[A \to \alpha \cdot \beta, i, k]$ *is in* $\mathscr{I}_j$, *then*

(i) $S' \overset{*}{\Rightarrow} a_1 \cdots a_i A \gamma$ *for some* γ, *and*

(ii) $\alpha \overset{*}{\Rightarrow} a_{i+1} \cdots a_j$ *using* k *terminal error productions.*

Proof. The proof is a straightforward induction on the order in which items are added to the parse lists. □

LEMMA 2. *If*

(i) $S' \underset{lm}{\overset{*}{\Rightarrow}} a_1 \cdots a_h A \beta' \underset{lm}{\Rightarrow} a_1 \cdots a_h \alpha \beta \beta'$,

(ii) $\alpha \overset{*}{\Rightarrow} a_{h+1} \cdots a_i$ *using* k *terminal error productions,*

(iii) $\beta \overset{*}{\Rightarrow} a_{i+1} \cdots a_j$ *using* l *terminal error productions, and*

(iv) *there is no derivation of the form* $\alpha\beta \overset{*}{\Rightarrow} a_{h+1} \cdots a_j$ *using fewer than* $k + l$ *terminal error productions,*

then $[A \to \alpha \cdot \beta, h, k]$ *is in* $\mathscr{I}_i$ *and* $[A \to \alpha\beta \cdot, h, k + l]$ *is in* $\mathscr{I}_j$.

Proof. The proof is an induction on the sum of the lengths of derivations (i), (ii) and (iii). □

From Lemmas 1 and 2 we can conclude that x is in $E_k(L(G))$ for some $k \geqq 0$ if and only if $\mathscr{I}_n$ contains an item of the form $[S' \to \alpha \cdot, 0, k]$ and no item of the form $[S' \to \beta \cdot, 0, l]$, where $l < k$.

From the parse lists we can extract a parse for x that uses the fewest number of terminal error productions by means of the following algorithm.

ALGORITHM 3.

Input: $\mathscr{I}_0, \mathscr{I}_1, \cdots, \mathscr{I}_n$, the parse lists for $x = a_1 \cdots a_n$.

Output: A parse π for x according to G' such that π contains as few terminal error productions as possible.

Method: In $\mathscr{I}_n$ choose an item of the form $[S' \to \alpha \cdot, 0, k]$, where k is as small as possible. Let π initially be the empty string. Then execute the routine *parse*($[S' \to \alpha \cdot, 0, k], \mathscr{I}_n$) where *parse*($[A \to \alpha \cdot \beta, i, l], \mathscr{I}_j$) is defined as follows:

1. If $\beta = e$, then let π be the previous value of π followed by production $A \to \alpha$. Otherwise, π is unchanged.

2. (a) If $\alpha = \alpha' a$, execute *parse*$([A \to \alpha' \cdot a\beta, i, l], \mathscr{I}_{j-1})$, where $[A \to \alpha' \cdot a\beta, i, l]$ is the item in $\mathscr{I}_{j-1}$ to which item $[A \to \alpha' a \cdot \beta, i, l]$ on $\mathscr{I}_j$ has a pointer. Return.

(b) If $\alpha = \alpha' B$, then execute *parse*$([B \to \gamma \cdot, h, m], \mathscr{I}_j)$ followed by *parse*$([A \to \alpha' \cdot B\beta, i, k], \mathscr{I}_h)$, where $[A \to \alpha' \cdot B\beta, i, k]$ on $\mathscr{I}_h$ and $[B \to \gamma \cdot, h, m]$ on $\mathscr{I}_j$ are the two items pointed to by item $[A \to \alpha \cdot \beta, i, l]$ on $\mathscr{I}_j$. Return.

(c) If $\alpha = e$, return. □

Algorithm 3 traces out a right-most derivation of x using the pointers stored with the items to guide the derivation. After executing *parse*$([S' \to \alpha \cdot, 0, k], \mathscr{I}_n)$, π will be a sequence of productions in a right-most derivation of x from S' in G' using k terminal error productions. We can obtain a parse of a string w in $L(G)$ such that $w \overset{k}{\vdash} x$ by applying the homomorphism h in the proof of Theorem 1 to π.

Now let us examine the time complexity of finding a minimum distance parse for an input string $x = a_1 \cdots a_n$. We shall use the notation $g(n)$ is $O(f(n))$ to mean there is a constant c such that $g(n) \leqq cf(n)$ for all $n \geqq 1$.

LEMMA 3. *Algorithm 2 can be implemented to run in time $O(n^3)$ on a random access computer, where n is the length of the input string to be parsed.*

Proof. Steps 1–4 of Algorithm 2 compute $\mathscr{I}_0$. Since $\mathscr{I}_0$ contains a fixed number of items, these steps can be executed in constant time.

Let us now examine the amount of time required to compute $\mathscr{I}_j$. We first note that each list of items $\mathscr{I}_i$, $0 \leqq i < j$, contains at most ci items for some constant c, because for each h, $0 \leqq h \leqq i$, there is at most one item of the form $[A \to \alpha \cdot \beta, h, k]$ on $\mathscr{I}_i$. Thus, step 5 of Algorithm 2 can be executed in $O(j - 1)$ time.

We shall now show that the repeated application of steps 6 and 7 can be implemented in $O(j^2)$ time in the following manner. We can construct a directed graph D in which each node of D is labeled by an item on $\mathscr{I}_j$ and an edge is drawn from a node labeled I to a node labeled I' if item I on $\mathscr{I}_j$ can cause item I' to appear on $\mathscr{I}_j$ because of step 6 or 7. For example, if item $[B \to \gamma \cdot i, \ ]$ is on $\mathscr{I}_j$ and list $\mathscr{I}_i$ contains item $[A \to \alpha \cdot B\beta, h, k]$, then by step 6 we must add item $[A \to \alpha B \cdot \beta, h, \ ]$ to $\mathscr{I}_j$. Thus we would have a node I labeled $[B \to \gamma \cdot, i, \ ]$ in D and an edge from this node to a node I' labeled $[A \to \alpha B \cdot \beta, h, \ ]$. We would also label the edge from I to I' with k, the error count of item $[A \to \alpha \cdot B\beta, h, k]$ on $\mathscr{I}_i$. We shall use k subsequently to help determine the error count for item $[A \to \alpha B \cdot \beta, h, \ ]$. When we first construct this graph, however, we shall initially leave the error counts in all items in $\mathscr{I}_j$ empty except for items of the form $[A \to \alpha a_j \cdot \beta, i, k]$, those whose error count is zero, and those of the form $[A \to \alpha \cdot, j, k]$. Thus the graph D will contain $O(j)$ nodes and $O(j^2)$ edges, and can be constructed in time $O(j^2)$.

To determine the items that will finally be on $\mathscr{I}_j$ we must now determine the correct values for the empty error counts. We can find these values in time $O(j^2)$ as follows. First we isolate the strongly connected components of D. This can be done in time proportional to the number of edges in D. (See [9], for example.) If we treat the strongly connected components of D as single nodes, we have essentially reduced D to a directed acyclic graph D'. We can then evaluate the minimum error counts for the nodes of D by examining the nodes of D in such an order that all direct predecessors of a node N in D' are examined before N is examined.

If N represents a strongly connected component of D, then we need to traverse the edges in N only a fixed number of times to percolate the minimum error counts throughout the nodes of N. This follows from the fact that consideration of an item I cannot cause its own error count to subsequently decrease.

In this fashion we can determine the minimum error count for all items labeling the nodes of D in time proportional to the number of edges in D.

In conclusion, Algorithm 2 can be implemented in $O(n^3)$ time because each list of items can be created in $O(n^2)$ time. $\square$

Algorithm 3 can be implemented in $O(n)$ time. Also the parse for the word in $L(G)$ can be created from the parse according to G' in $O(n)$ time. Thus we have a minimum distance error-correcting parser that operates in time $O(n^3)$.

7. Concluding remarks.

We can extend this parsing method to include other types of errors. For example, certain transposition errors can be generated by error productions and we could include these productions with the other error productions.

If we do not want to generate all of Σ^* with our covering grammar, we could restrict the error productions so that they would only generate the syntax errors that are most likely to occur. Wirth [10] has considered a scheme of this nature in conjunction with precedence parsing.

In compiler applications it is desirable to use a fast parsing algorithm such as LL or LR parsing. On encountering an error we could invoke our error-correcting parser. Other methods for error recovery and correction in LR parsing are discussed in [4]–[8].

Finally, it is interesting to ask how a programming language can be designed so as to maximize the distance between correct programs.

REFERENCES

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

[2] J. Earley, *An efficient context-free parsing algorithm*, Comm. ACM, 13 (1970), pp. 94–102.

[3] E. T. Irons, *An error-correcting parse algorithm*, Ibid., 6 (1963), pp. 669–673.

[4] L. R. James, *A syntax directed error recovery method*, Tech. Rep. CSRG-13, Computer Systems Research Group, University of Toronto, Toronto, 1972.

[5] R. P. Leinius, *Error detection and recovery for syntax directed compiler systems*, Doctoral thesis, University of Wisconsin, Madison, 1970.

[6] J. P. Levy, *Automatic correction of syntax errors in programming languages*, Tech. Rep. 71–116, Dept. of Computer Science, Cornell University, Ithaca, N.Y., 1971.

[7] G. T. McGruther, *An approach to automating syntax error detection, recovery, and correction for LR(k) grammars*, M.S. thesis, Naval Postgraduate School, Monterey, Calif., 1972.

[8] T. G. Peterson, *Syntax error detection, correction and recovery in parsers*, Doctoral thesis, Stevens Institute of Technology, Hoboken, N.J., 1972.

[9] R. Tarjan, *Depth-first search and linear graph algorithms*, this Journal, 1 (1972), pp. 146–160.

[10] N. Wirth, *PL360, a programming language for the 360 computers*, J. Assoc. Comput. Mach., 15 (1968), pp. 37–74.