TAL Recognition in $O(M(n^2))$ Time

Sanguthevar Rajasekaran
Dept. of CISE, Univ. of Florida
raj@cis.ufl.edu

Shibu Yooseph
Dept. of CIS, Univ. of Pennsylvania
yooseph@gradient.cis.upenn.edu

Abstract

We propose an $O(M(n^2))$ time algorithm for the recognition of Tree Adjoining Languages (TALs), where $n$ is the size of the input string and $M(k)$ is the time needed to multiply two $k \times k$ boolean matrices. Tree Adjoining Grammars (TAGs) are formalisms suitable for natural language processing and have received enormous attention in the past among not only natural language processing researchers but also algorithms designers. The first polynomial time algorithm for TAL parsing was proposed in 1986 and had a run time of $O(n^6)$. Quite recently, an $O(n^3 M(n))$ algorithm has been proposed. The algorithm presented in this paper improves the run time of the recent result using an entirely different approach.

1 Introduction

The Tree Adjoining Grammar (TAG) formalism was introduced by Joshi, Levy and Takahashi (1975). TAGs are tree generating systems, and are strictly more powerful than context-free grammars. They belong to the class of mildly context sensitive grammars (Joshi, et al., 1991). They have been found to be good grammatical systems for natural languages (Kroch, Joshi, 1985). The first polynomial time parsing algorithm for TALs was given by Vijayashanker and Joshi (1986), which had a run time of $O(n^6)$, for an input of size $n$. Their algorithm had a flavor similar to the Cocke-Younger-Kasami (CYK) algorithm for context-free grammars. An Earley-type parsing algorithm has been given by Schabes and Joshi (1988). An optimal linear time parallel parsing algorithm for TALs was given by Palis, Shende and Wei (1990). In a recent paper, Rajasekaran (1995) shows how TALs can be parsed in time $O(n^3M(n))$.

In this paper, we propose an $O(M(n^2))$ time recognition algorithm for TALs, where $M(k)$ is the time needed to multiply two $k \times k$ boolean matrices. The best known value for $M(k)$ is $O(k^{2.376})$ (Coppersmith, Winograd, 1990). Though our algorithm is similar in flavor to those of Graham, Harrison, & Ruzzo (1976), and Valiant (1975) (which were algorithms proposed for recognition of Context Free Languages (CFLs)), there are crucial differences. As such, the techniques of (Graham, et al., 1976) and (Valiant, 1975) do not seem to extend to TALs (Satta, 1993).

2 Tree Adjoining Grammars

A Tree Adjoining Grammar (TAG) consists of a quintuple $(N, \Sigma \cup \{\varepsilon\}, I, A, S)$, where

$N$ is a finite set of nonterminal symbols,
$\Sigma$ is a finite set of terminal symbols disjoint from $N$,
$\varepsilon$ is the empty terminal string not in $\Sigma$,
$I$ is a finite set of labelled initial trees,
$A$ is a finite set of auxiliary trees,
$S \in N$ is the distinguished start symbol.

The trees in $I \cup A$ are called elementary trees. All internal nodes of elementary trees are labelled with nonterminal symbols. Also, every initial tree is labelled at the root by the start symbol $S$ and has leaf nodes labelled with symbols from $\Sigma \cup \{\varepsilon\}$. An auxiliary tree has both its root and exactly one leaf (called the foot node ) labelled with the same non-terminal symbol. All other leaf nodes are labelled with symbols in $\Sigma \cup \{\varepsilon\}$, at least one of which has a label strictly in $\Sigma$. An example of a TAG is given in figure 1.

A tree built from an operation involving two other trees is called a derived tree. The operation involved is called adjunction. Formally, adjunction is an operation which builds a new tree $\gamma$, from an auxiliary
tree $\beta$ and another tree $\alpha$ ($\alpha$ is any tree - initial, auxiliary or derived). Let $\alpha$ contain an internal node $m$ labelled $X$ and let $\beta$ be the auxiliary tree with root node also labelled $X$. The resulting tree $\gamma$, obtained by adjoining $\beta$ onto $\alpha$ at node $m$ is built as follows (figure 2):

1. The subtree of $\alpha$ rooted at $m$, call it $t$, is excised, leaving a copy of $m$ behind.
2. The auxiliary tree $\beta$ is attached at the copy of $m$ and its root node is identified with the copy of $m$.
3. The subtree $t$ is attached to the foot node of $\beta$ and the root node of $t$ (i.e. $m$) is identified with the foot node of $\beta$.

This definition can be extended to include adjunction constraints at nodes in a tree. The constraints include Selective, Null and Obligatory adjunction constraints. The algorithm we present here can be modified to include constraints.

For our purpose, we will assume that every internal node in an elementary tree has exactly 2 children.

Each node in a tree is represented by a tuple $<\text{tree, node index, label}>$. (For brevity, we will refer to a node with a single variable $m$ wherever there is no confusion.)

A good introduction to TAGs can be found in (Partee, et al., 1990).

3 Context Free recognition in $O(M(n))$ Time

The CFG $G = (N, \Sigma, P, A_1)$, where

- $N$ is a set of Nonterminals $\{A_1, A_2, ..., A_k\}$,
- $\Sigma$ is a finite set of terminals,
- $P$ is a finite set of productions,
- $A_1$ is the start symbol

is assumed to be in the Chomsky Normal Form. Valiant (1975) shows how the recognition problem can be reduced to the problem of finding Transitive Closure and how Transitive Closure can be reduced to Matrix Multiplication.

Given an input string $a_1a_2...a_n \in \Sigma^*$, the recursive algorithm makes use of an $(n+1) \times (n+1)$ upper triangular matrix $b$ defined by

$b_{i,i+1} = \{A_k \mid (A_k \rightarrow a_i) \in P\}$,
$b_{i,j} = \phi$, for $j \neq i + 1$

and proceeds to find the transitive closure $b^+$ of this matrix. (If $b^+$ is the transitive closure, then $A_k \in b^+_{i,j}$ is equivalent to $A_k \Rightarrow a_i...a_{i-1}$)

Instead of finding the transitive closure by the customary method based on recursively splitting into disjoint parts, a more complex procedure based on 'splitting with overlaps' is used. The extra cost involved in such a strategy can be made almost negligible. The algorithm is based on the following lemma

Lemma : Let $b$ be an $n \times n$ upper triangular matrix, and suppose that for any $r > n/2$, the transitive closure of the partitions $[1 \leq i, j \leq r]$ and $[n - r < i, j \leq n]$ are known. Then the closure of $b$ can be computed by

1. performing a single matrix multiplication, and
2. finding the closure of a $2(n-r) \times 2(n-r)$ upper triangular matrix of which the closure of the partitions $[1 \leq i, j \leq n-r]$ and $[n-r < i, j \leq 2(n-r)]$ are known.

Proof: See (Valiant, 1975) for details

The idea behind (Valiant, 1975) is based on visualizing $A_k \in b^+_{i,j}$ as spanning a tree rooted at the
Given an input string $a_1...a_n$ and 2 distinct symbol positions, $i$ and $j$, and a nonterminal $A_k$ such that $A_k \in b^+_{i',j'}$, where $i' \leq i, j' > j$, then there exists a non-terminal $A'_{k'}$ which is a descendent of $A_k$ in the tree rooted at $A_k$, such that $A'_{k'} \in b^+_{i'',j''}$, where $i'' \leq i, j'' > j$ and $A'_{k'}$ has two children $A_{k_1}$ and $A_{k_2}$ such that $A_{k_1} \in b^+_{i',s}$ and $A_{k_2} \in b^+_{s,j'}$ with $i < s < j$. $A'_{k'}$ can be thought of as a minimal node in this sense. (The descendent relation is both reflexive and transitive.)

Thus, given a string $a_1...a_n$ of length $n$, (say $r = 2/3$), the following steps are done:

1. Find the closure of the first 2/3, i.e. all nodes spanning trees which are within the first 2/3.

2. Find the closure of the last 2/3, i.e. all nodes spanning trees which are within the last 2/3.

3. Do a composition operation (i.e. matrix multiplication) on the nodes got as a result of Step 1 with nodes got as a result of Step 2.

4. Reduce problem size to $a_1...a_{n/3}a_{1+2n/3}...a_n$ and find closure of this input.

The point to note is that in step 3, we can get rid of the mid 1/3 and focus on the remaining problem size.

This approach does not work for TALs because of the presence of the adjunction operation.

Firstly, the data structure used, i.e. the 2-dimensional matrix with the given representation, is not sufficient as adjunction does not operate on contiguous strings. Suppose a node in a tree dominates a frontier which has the substring $a_ia_j$ to the left of the foot node and $a_ka_l$ to the right of the footnode. These substrings need not be a contiguous part of the input; in fact, when this tree is used for adjunction then a string is inserted between these two substrings. Thus in order to represent a node, we need to use a matrix of higher dimension, namely dimension 4, to characterize the substring that appears to the left of the footnode and the substring that appears to the right of the footnode.

Secondly, the observation we made about an entry $\in b^+$ is no longer quite true because of the presence of adjunction.

Thirdly, the technique of getting rid of the mid 1/3 and focusing on the reduced problem size alone, does not work as shown in figure 3: Suppose $\gamma$ is a derived tree in which $\exists$ a node $m$ on which adjunction was done by an auxiliary tree $\beta$. Even if we are able to identify the derived tree $\gamma_1$ rooted at $m$, we have to first identify $\beta$ before we can check for adjunction. $\beta$ need not be realised as a result of the composition operation involving the nodes from the first and last 2/3’s , (say $r = 2/3$). Thus, if we discard the mid 1/3, we will not be able to infer that the adjunction had indeed taken place at node $m$.

4 Notations

Before we introduce the algorithm, we state the notations that will be used.

We will be making use of a 4-dimensional matrix $A$ of size $(n+1) \times (n+1) \times (n+1) \times (n+1)$, where $n$ is the size of the input string.

(Vijayashanker, Joshi, 1986) Given a TAG $G$ and an input string $a_1a_2...a_n$, $n \geq 1$, the entries in $A$ will be nodes of the trees of $G$. We say, that a node $m$...
Node m has label X

Figure 3: Situation where we cannot infer the adjunction if we simply get rid of the mid 1/3

\(\eta, \text{node index, label} > \in A(i,j,k,l)\) iff m is a node in a derived tree \(\gamma\) and the subtree of \(\gamma\) rooted at m has a yield given by either \(a_{i+1}...a_jXa_{k+1}...a_l\) (where X is the footnode of \(\eta\), \(j < k\)) or \(a_{i+1}...a_l\) (when \(j = k\)).

If a node \(m \in A(i,j,k,l)\), we will refer to m as spanning a tree \((i,j,k,l)\).

When we refer to a node m being realised as a result of composition of two nodes \(m1\) and \(m2\), we mean that \(\exists\) an elementary tree in which m is the parent of \(m1\) and \(m2\).

A Grown Auxiliary Tree is defined to be either a tree resulting from an adjunction involving two auxiliary trees or a tree resulting from an adjunction involving an auxiliary tree and a grown auxiliary tree.

Given a node m spanning a tree \((i,j,k,l)\), we define the last operation to create this tree as follows:

- If the tree \((i,j,k,l)\) was created in a series of operations, which also involved an adjunction by an auxiliary tree (or a grown auxiliary tree) \((i,j_1,k_1,l)\) onto the node m, then we say that the last operation to create this tree is an adjunction operation; else the last operation to create the tree \((i,j,k,l)\) is a composition.

The concept of last operation is useful in modelling the steps required, in a bottom-up fashion, to create a tree.

5 Algorithm

Given that the set of initial and auxiliary trees can have leaf nodes labelled with \(\varepsilon\), we do some preprocessing on the TAG G to obtain an Association List (ASSOC LIST) for each node. ASSOC LIST \((m)\), where \(m\) is a node, will be useful in obtaining chains of nodes in elementary trees which have children labelled \(\varepsilon\).

Initialize ASSOC LIST \((m) = \phi, \forall m\), and then call procedure MAKELIST on each elementary tree, in a top down fashion starting with the root node.

Procedure MAKELIST \((m)\)

Begin

1. If m is a leaf then quit
2. If m has children \(m1\) and \(m2\) both yielding the empty string at their frontiers (i.e. m spans a subtree yielding \(\varepsilon\)) then
   \[\text{ASSOC LIST } (m1) = \text{ASSOC LIST } (m) \cup \{m\}\]
   \[\text{ASSOC LIST } (m2) = \text{ASSOC LIST } (m) \cup \{m\}\]
3. If m has children \(m1\) and \(m2\), with only \(m2\) yielding the empty string at its frontier, then
   \[\text{ASSOC LIST } (m1) = \text{ASSOC LIST } (m) \cup \{m\}\]
End

We initially fill \(A(i,i+1,i+1,i+1)\) with all nodes from \(S_{m1}, \forall m1\), where \(S_{m1} = \{m1\} \cup \text{ASSOC LIST } (m1), m1\) being a node with the same label as the input \(a_{i+1}\), for \(0 \leq i \leq n-1\). We also fill \(A(i,i,j,j), i \leq j\), with nodes from \(S_{m2}, \forall m2\), where
$S_m = \{m2\} \cup ASSOC\ LIST\ (m2)$, $m2$ being a foot node. All entries $A(i,i,i,i)$, $0 \leq i \leq n$, are filled with nodes from $S_m$, $\forall m3$, where $S_m = \{m3\} \cup ASSOC\ LIST\ (m3)$, $m3$ having label $\varepsilon$.

Following is the main procedure, Compute Nodes, which takes as input a sequence $r_1r_2...r_p$ of symbol positions (not necessarily contiguous). The procedure outputs all nodes spanning trees $(i,j,k,l)$, with $(i,j) \in \{r_1,r_2,...,r_p\}$ and $(j,k) \in \{r_1,r_1+1,...,r_p\}$. The procedure is initially called with the sequence $012...n$ corresponding to the input string $a_1...a_n$. The matrix $A$ is updated with every call to this procedure and it is updated with the nodes just realised and also with the nodes in the ASSOC\ LISTs of the nodes just realised.

Procedure Compute Nodes ($r_1r_2...r_p$)

**Begin**

1. If $p = 2$, then
   a. Compose all nodes $A(r_1,j,k,r_2)$ with all nodes $A(r_2,r_2,r_2)$, $r_1 \leq j \leq k \leq r_2$.
   Update $A$.
   b. Compose all nodes $A(r_1,j,k,r_2)$ with all nodes $A(r_1,r_1,r_1)$, $r_1 \leq j \leq k \leq r_2$.
   Update $A$.
   c. Check for adjunctions involving nodes realised from steps a and b. Update $A$.
   d. Return.

2. Compute Nodes ($r_1r_2...r_{2p/3}$).

3. Compute Nodes ($r_1r_2...r_{2p/3}$).

4. a. Compose nodes realised from step 2 with nodes realised from step 3.
   b. Update $A$.

5. a. Check for all possible adjunctions involving nodes realised as a result of step 4.
   b. Update $A$.

6. Compute Nodes ($r_1r_2...r_{2p/3}r_{1+2p/3}...r_p$)

**End**

**Steps 1a, 1b and 4a** can be carried out in the following manner:

Consider the composition of node $m1$ with node $m2$. For step 4a, there are two cases to take care of.

**Case 1**

If node $m1$ in a derived tree is the ancestor of the foot node, and node $m2$ is its right sibling, such that $m1 \in A(i,j,k,l)$ and $m2 \in A(l,r,r,s)$, then their parent, say node $m$ should belong to $A(i,j,k,s)$. This composition of $m1$ with $m2$ can be reduced to a boolean matrix multiplication in the following way: (We use a technique similar to the one used in (Rajasekaran, 1995)) Construct two boolean matrices $B1$, of size $((n+1)^2p/3) \times (p/3)$ and $B2$, of size $(p/3) \times (p/3)$.

- $B1(ijk,l) = 1$ iff $m1 \in A(i,j,k,l)$ and $i \in \{r_1,...,r_{p/3}\}$ and $l \in \{r_{1+p/3},...,r_{2p/3}\}$
  $= 0$ otherwise

- Note that in $B1$ $0 \leq j \leq k \leq n$.

- $B2(l,s) = 1$ iff $m2 \in A(l,r,r,s)$ and $l \in \{r_{1+p/3},...,r_{2p/3}\}$ and $s \in \{r_{1+2p/3},...,r_p\}$
  $= 0$ otherwise

Clearly the dot product of the $ijk^{th}$ row of $B1$ with the $s^{th}$ column of $B2$ is a 1 iff $m \in A(i,j,k,s)$. Thus, update $A(i,j,k,s)$ with $\{m\} \cup ASSOC\ LIST\ (m)$.

**Case 2**

If node $m2$ in a derived tree is the ancestor of the foot node, and node $m1$ is its left sibling, such that $m1 \in A(i,j,j,l)$ and $m2 \in A(l,p,q,r)$, then their parent, say node $m$ should belong to $A(i,p,q,s)$. This can also be handled similar to the manner described for case 1. Update $A(i,p,q,s)$ with $\{m\} \cup ASSOC\ LIST\ (m)$.

Notice that Case 1 also covers step 1a and Case 2 also covers step 1b.

**Step 5a** and **Step 1c** can be carried out in the following manner:

We know that if a node $m \in A(i,j,k,l)$, and the root $m1$ of an auxiliary tree $A(r,i,l,s)$, then adjoining the tree $\eta$, rooted at $m1$, onto the node $m$, results in the node $m$ spanning a tree $(r,j,k,s)$, i.e. $m \in A(r,j,k,s)$.

We can essentially use the previous technique of reducing to boolean matrix multiplication. Construct two matrices $C1$ and $C2$ of sizes $(p^3/9) \times (n+1)^3$ and $(n+1)^2 \times (n+1)^2$, respectively, as follows:

$$C1(i,j,k) = 1$$ if $\exists m1$, root of an auxiliary tree $A(i,j,k,l)$, with same label as $m$ and $C1(i,j,k) = 0$ otherwise

Note that in $C1$ $i \in \{r_1,...,r_{p/3}\}$, $l \in \{r_{1+2p/3},...,r_p\}$, and $0 \leq j \leq k \leq n$.

$$C2(q,t,rs) = 1$$ if $m \in A(q,r,s,t)$
  $= 0$ otherwise
Note that in C2 $0 \leq q < r \leq s < t \leq n$.
Clearly the dot product of the $i$th row of C1 with the $rs$th column of C2 is a 1 iff $m \in A(i,r,s,l)$. Thus, update $A(i,r,s,l)$ with $\{m\} \cup \text{ASSOC LIST}(m)$.

The input string $a_1a_2...a_n$ is in the language generated by the TAG G iff $\exists$ a node labelled $S$ in some $A(0,j,j,n)$, $0 \leq j \leq n$.

6 Complexity
Steps 1a, 1b and 4a can be computed in $O(n^2M(p))$.
Steps 5a and 1c can be computed in $O((n^2/p)^2M(p^2))$.
If $T(p)$ is the time taken by the procedure Compute Nodes, for an input of size $p$, then $T(p) = 3T(2p/3) + O(n^2M(p)) + O((n^2/p)^2M(p^2))$ where $n$ is the initial size of the input string.
Solving the recurrence relation, we get $T(n) = O(M(n^2))$.

7 Proof of Correctness
We will show the proof of correctness of the algorithm by induction on the length of the sequence of symbol positions.
But first, we make an observation, given any two symbol positions $(r_x, r_t)$, $r_t > r_x + 1$, and a node $m$ spanning a tree $(i,j,k,l)$ such that $i \leq r_x$ and $l \geq r_t$ with $j$ and $k$ in any of the possible combinations as shown in figure 4.

$\exists$ a node $m$ which is a descendant of the node $m$ in the tree $(i,j,k,l)$ and which either $\in \text{ASSOC LIST}(m_1)$ or is the same as $m_1$, with $m_1$ having one of the two properties mentioned below:

1. $m_1$ spans a tree $(i_1,j_1,k_1,l_1)$ such that the last operation to create this tree was a composition operation involving two nodes $m_2$ and $m_3$ with $m_2$ spanning $(i_1,j_2,k_2,l_2)$ and $m_3$ spanning $(l_2,j_3,k_3,l_3)$, (with $r_x < l_2 < r_t$, $(i_1 \leq r_x)$, $(r_1 \leq l_1)$ and either $(j_2 = k_2, j_2 = j_1, k_3 = k_1)$ or $(j_2 = j_1, k_2 = k_1, j_3 = k_3)$)
2. $m_1$ spans a tree $(i_1,j_1,k_1,l_1)$ such that the last operation to create this tree was an adjunction by an auxiliary tree (or a grown auxiliary tree) $(i_1,j_2,k_2,l_1)$, rooted at node $m_2$, onto the node $m_1$ spanning the tree $(j_2,j_1,k_1,l_2)$ such that node $m_2$ has either the property mentioned in (1) or belongs to the ASSOC LIST of a node which has the property mentioned in (1). (The labels of $m_1$ and $m_2$ being the same)

Any node satisfying the above observation will be called a minimal node w.r.t. the symbol positions $(r_x,r_t)$.
The minimal nodes can be identified in the following manner. If the node $m$ spans $(i,j,k,l)$ such that the last operation to create this tree is a composition of the form in figure 5a, then $m \cup \text{ASSOC LIST}(m)$ is minimal. Else, if it is as shown in figure 5b, we can concentrate on the tree spanned by node $m_1$ and repeat the process. But, if the last operation to create $(i,j,k,l)$ was an adjunction as shown in figure 5c, we can concentrate on the tree $(i_1,j,k,l_1)$ initially spanned by node $m$. If the only adjunction was by an auxiliary tree, on node $m$ spanning tree $(i_1,j,k,l_1)$ as shown in figure 5d, then the set of minimal nodes will include both $m$ and the root $m_1$ of the auxiliary tree and the nodes in their respective ASSOC LISTS. But if the adjunction was by a grown auxiliary tree as shown in figure 5e, then the minimal nodes include the roots of $\beta_1,\beta_2,...,\beta_8,\gamma$ and the node $m$.

Given a sequence $< r_1, r_2, ..., r_p >$, we call $(r_q, r_{q+1})$ a gap, iff $r_{q+1} \neq r_q + 1$. Identifying minimal nodes w.r.t. every new gap created, will serve our purpose in determining all the nodes spanning trees $(i,j,k,l)$, with $(i,l) \in \{r_1,r_2,...,r_p\}$.

Theorem: Given an increasing sequence $< r_1, r_2, ..., r_p >$ of symbol positions and given

a. $\forall$ gaps $(r_q, r_{q+1})$, all nodes spanning trees $(i,j,k,l)$ with $r_q < i < j \leq k \leq l < r_{q+1}$
b. $\forall$ gaps $(r_q, r_{q+1})$, all nodes spanning trees $(i,j,k,l)$ such that either $r_q < i < r_{q+1}$ or $r_q < l < r_{q+1}$
c. $\forall$ gaps $(r_q, r_{q+1})$, all the minimal nodes for the gap such that these nodes span trees $(i,j,k,l)$ with $(i,l) \in \{r_1,r_2,...,r_p\}$ and $i \leq l$

in addition to the initialization information, the algorithm computes all the nodes spanning trees $(i,j,k,l)$ with $(i,l) \in \{r_1,r_2,...,r_p\}$ and $i \leq j \leq k \leq l$.

Proof:
Base Cases:
For length = 1, it is trivial as this information is already known as a result of initialization.
For length = 2, there are two cases to consider:

1. \( r_2 = r_1 + 1 \), in which case a composition involving nodes from \( A(r_1, r_1, r_1, r_1) \) with nodes from \( A(r_1, r_2, r_2, r_2) \) and a composition involving nodes from \( A(r_1, r_2, r_2, r_2) \) with nodes from \( A(r_2, r_2, r_2, r_2) \), followed by a check for adjunction involving nodes realised from the previous two compositions, will be sufficient. Note that since there is only one symbol from the input (namely, \( a_{r_2} \)), and because an auxiliary tree has at least one label from \( \Sigma \), thus, checking for one adjunction is sufficient as there can be at most one adjunction.

2. \( r_2 \neq r_1 + 1 \), implies that \((r_1, r_2)\) is a gap. Thus, in addition to the information given as per the theorem, a composition involving nodes from \( A(r_1, j, k, r_2) \) with nodes from \( A(r_2, r_2, r_2, r_2) \) and a composition involving nodes from \( A(r_1, r_1, r_1, r_1) \) with nodes from \( A(r_1, j, k, r_2) \), \((r_1 \leq j \leq k \leq r_2)\), followed by an adjunction involving nodes realised as a result of the previous two compositions will be sufficient as the only adjunction to take care of involves the adjunction of some auxiliary tree onto a node \( m \) which yields \( \varepsilon \), and \( m \in A(r_1, r_1, r_1, r_1) \) or \( m \in A(r_2, r_2, r_2, r_2) \).

**Induction hypothesis**: \( \forall \) increasing sequence \( < r_1, r_2, \ldots, r_q > \) of symbol positions of length \( \leq p \) (i.e. \( q \leq p \)), the algorithm, given the information as required by the theorem, computes all nodes spanning trees \((i, j, k, l)\) such that \((i, l) \in \{ r_1, r_2, \ldots, r_q \} \) and \( i \leq j \leq k \leq l \). **Induction**: Given an increasing sequence \( < r_1, r_2, \ldots, r_p, r_{p+1} > \) of symbol positions together with the information required as per parts a, b, c of the theorem, the algorithm proceeds as follows:

1. By the induction hypothesis, the algorithm correctly computes all nodes spanning trees \((i, j, k, l)\) within the first \( 2/3 \), i.e., \( \{i, l\} \in \{ r_1, r_2, \ldots, r_{2(p+1)/3} \} \) and \( i \leq l \). By the hypothesis, it also computes all nodes \((i', j', k', l')\) within the last \( 2/3 \), i.e., \( \{i', l'\} \in \{ r_{1+(p+1)/3}, \ldots, r_{p+1} \} \) and \( i' \leq l' \).

2. The composition step involving the nodes from the first and last \( 2/3 \) of the sequence \( < r_1, r_2, \ldots, r_p, r_{p+1} > \), followed by the adjunction step captures all nodes \( m \) such that either:
   a. \( m \) spans a tree \((i, j, k, l)\) such that the last operation to create this tree was a composition operation on two nodes \( m_1 \) and \( m_2 \) with \( m_1 \) spanning \((i, j', k', l')\) and \( m_2 \) spanning \((l', j'', k'', l')\). (with \( i \in \{ r_1, r_2, \ldots, r_{(p+1)/3} \} \), \( l' \in \{ r_{1+(p+1)/3}, \ldots, r_{2(p+1)/3} \} \) and \( l \in \{ r_{1+2(p+1)/3}, \ldots, r_{p+1} \} \), and either \((i' = k'\), \( j'' = j, k'' = k)\) or \((j' = j, k' = k, j'' = k')\).
   b. \( m \) spans a tree \((i, j, k, l)\) such that the last operation to create this tree was an adjunc-
Note that, in addition to the nodes \( m \) captured from \( a \) or \( b \), we will also be realising nodes \( \in ASSOC\ LIST \) ( \( m \)).

The nodes captured as a result of 2 are the minimal nodes with respect to the gap \( (r_{(p+1)/3}, r_{1+2(p+1)/3}) \) with the additional property that the trees \((i,j,k,l)\) they span are such that \( i \in \{r_1,r_2,...,r_{(p+1)/3}\} \) and \( l \in \{r_{1+2(p+1)/3},...,r_{p+1}\} \).

Before we can apply the hypothesis on the sequence \( <r_1,r_2,...,r_{(p+1)/3},r_{1+2(p+1)/3},...,r_{p+1}>\), we have to make sure that the conditions in parts \( a,b,c \) of the theorem are met for the new gap \( (r_{(p+1)/3}, r_{1+2(p+1)/3}) \). It is easy to see that conditions for parts \( a \) and \( b \) are met for this gap. We have also seen that as a result of step 2, all the minimal nodes w.r.t the gap \( (r_{(p+1)/3}, r_{1+2(p+1)/3}) \), with the desired property as required in part \( c \) have been computed. Thus applying the hypothesis on the sequence \( <r_1,r_2,...,r_{(p+1)/3},r_{1+2(p+1)/3},...,r_{p+1}>\), the algorithm in the end correctly computes all the nodes spanning trees \((i,j,k,l)\) with \( \{i,l\} \in \{r_1,r_2,...,r_{p+1}\} \) and \( i \leq j \leq k \leq l \).

8 Implementation

The TAL recognizer given in this paper was implemented in Scheme on a SPARC station-10/30. Theoretical results in this paper and those in (Rajasekaran, 1995) clearly demonstrate that asymptotically fast algorithms can be obtained for TAL parsing with the help of matrix multiplication al-

![Figure 5: Identifying minimal nodes](image-url)
The above implementation results suggest that even in practice better parsing algorithms can be obtained through the use of matrix multiplication techniques.

9 Conclusions

In this paper we have presented an \(O(M(n^2))\) time algorithm for parsing TALs, \(n\) being the length of the input string. We have also demonstrated with our implementation work that matrix multiplication techniques can help us obtain efficient parsing algorithms.

Acknowledgements

This research was supported in part by an NSF Research Initiation Award CCR-92-09260 and an ARO grant DAAL03-89-C-0031.

References


G. Satta, Tree Adjoining Grammar Parsing and Boolean Matrix Multiplication, to be presented in the 31st Meeting of the Association for Computational Linguistics, 1993.

G. Satta, Personal Communication, September 1993.


<table>
<thead>
<tr>
<th>String</th>
<th>Answer</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>Yes</td>
<td>3.1</td>
</tr>
<tr>
<td>aabbcc</td>
<td>Yes</td>
<td>6.1</td>
</tr>
<tr>
<td>aacabc</td>
<td>No</td>
<td>8.0</td>
</tr>
<tr>
<td>abacabac</td>
<td>No</td>
<td>11.7</td>
</tr>
<tr>
<td>aaabbbcccc</td>
<td>Yes</td>
<td>11.4</td>
</tr>
</tbody>
</table>