Tree-Adjoining Grammar Parsing and Boolean Matrix Multiplication

Giorgio Satta*†
Università di Venezia

The computational problem of parsing a sentence in a tree-adjoining language is investigated. An interesting relation is studied between this problem and the well-known computational problem of Boolean matrix multiplication: it is shown that any algorithm for the solution of the former problem can easily be converted into an algorithm for the solution of the latter problem. This result bears on at least two important computational issues. First, we realize that a straightforward method that improves the known upper bound for tree-adjoining grammar parsing is hard to find. Second, we understand which features of the tree-adjoining grammar parsing problem are responsible for the claimed difficulty.

1. Introduction

Among formalisms for the computation of syntactic description of natural language sentences, Tree-Adjoining Grammars (TAG) play a major role. The class of TAG's was first introduced in Joshi, Levy, and Takahashi (1975) and Joshi (1985); since then, formal and computational properties of this class have been extensively investigated, and the linguistic relevance of TAGs has been discussed in the literature as well. The reader who is interested in these topics is referred to some of the most recent works, for example Schabes (1990) and Frank (1992), and to the references therein.

Both in a theoretical vein and in view of possible natural language processing applications, the recognition and parsing problems for TAGs have been extensively studied and many algorithms have been proposed for their solution. On the basis of tabular techniques, the least time upper bound that has been attested is $O(|G||w|^6)$ for the random-access model of computation, $|G|$ being the size of the input grammar and $|w|$ the length of the input string. In recent years, improvement of such a worst-case running time has been a common goal for many researchers, but up to the present time the TAG parsing problem has strongly resisted all such attempts. Because of the record of all these efforts, the task of improving the above upper bound is actually regarded as a difficult one by many researchers.

In support of such a common feeling, in this paper we restate the TAG parsing problem as a search problem and relate it to the well-known computational problem of Boolean matrix multiplication. This is done in such a way that time upper bounds for TAG parsing can be transferred to time upper bounds for the latter problem. More precisely, we show that any algorithm for TAG parsing that improves the $O(|G||w|^6)$ time upper bound can be converted into an algorithm for Boolean matrix multiplication running in less than $O(m^3)$ time, m being the order of the input

* Università di Venezia, Scienze dell'Informazione, via Torino, 155, 30172 Mestre–Venezia, Italy. E-mail: satta@moo.dsi.unive.it.
† This research was done while the author was a post-doctoral fellow at the Institute for Research in Cognitive Science, University of Pennsylvania, 3401 Walnut Street, Philadelphia, PA 19104–6228, USA.

matrices. Crucially, Boolean matrix multiplication has been the object of investigation for many years: methods that are asymptotically faster than $O(m^3)$ are known, but the more considerable the improvement turned out to be, the more complex the involved computation was found to be. At the present time, the asymptotically fastest algorithms for Boolean matrix multiplication are considered to be only of theoretical interest, because the huge constants involved in the running time of these methods render prohibitive any practical application, given current computer hardware.

As a matter of fact, the design of practical algorithms for Boolean matrix multiplication that considerably improve the cubic time upper bound is regarded as a very difficult enterprise. A consequence of the results presented in this paper is that TAG parsing should also be considered as having the status of a problem that is "hard to improve," and there is enough evidence to think that methods for TAG parsing that are asymptotically faster than $O(|G||w|^6)$ are unlikely to be of any practical interest, i.e., will involve very complex computations.

The remaining part of this paper is organized as follows. The next section presents the definition of tree-adjoining grammar and introduces the two computational problems that are to be related. Section 3 establishes the main result. Section 4 draws on the computational consequences of such a result and reports some discussion. Finally, Section 5 concludes by indicating how similar results can be found for variants of the TAG parsing problem that have been recently discussed in the literature.

2. Preliminaries

This section introduces the Boolean matrix multiplication problem and the tree-adjoining grammar parsing problem, along with the definition of tree-adjoining grammar. The notation presented here will be used throughout the paper.

2.1 Boolean Matrix Multiplication

Let $\mathcal{B} = \{0, 1\}$ be the set of truth values. The logical symbols $\vee$, $\wedge$ are defined as usual. The set of Boolean square matrices $\mathbf{B}_m$, $m \geq 1$, is defined as the set of all $m \times m$ square matrices whose elements belong to $\mathcal{B}$. Given a matrix $A \in \mathbf{B}_m$, we say that A has **order** m; the element in the ith row and jth column of A is denoted by a_{ij}. In $\mathbf{B}_m$, the **product** of A and B, written $A \times B$, is a Boolean matrix C such that:

$$c_{ij} = \bigvee_{k=1}^{m} a_{ik} \wedge b_{kj}, \qquad 1 \leq i, j \leq m. \tag{1}$$

An instance of the **Boolean matrix multiplication problem** is therefore a pair $\langle A, B \rangle$ and the solution to such an instance consists of the matrix C such that $C = A \times B$. In what follows BMM will denote the set of all possible instances of the Boolean matrix multiplication problem.

2.2 Tree-Adjoining Grammars

The definition of TAG and the associated notion of derivation are briefly introduced in the following; the reader is also referred to the standard literature (see, for instance, Vijay-Shanker and Joshi [1985] or Joshi, Vijay-Shanker, and Weir [1991]).

A **tree-adjoining grammar** is a tree rewriting system denoted by a tuple $G = (V_N, V_T, S, I, A)$, where V_N and V_T are finite, disjoint sets of **nonterminal** and **terminal** symbols respectively, $S \in V_N$ is a distinguished symbol, and I and A are finite sets of **elementary** trees. Trees in I and A are called **initial** and **auxiliary** trees respectively and meet the following specifications. Internal (nonleaf) nodes in an elementary tree

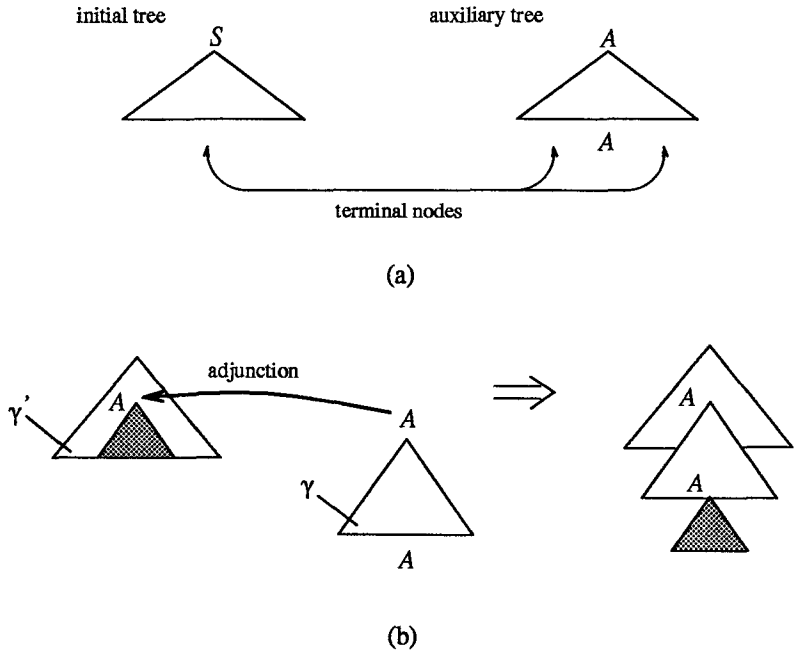

Figure 1
Definitions of (a) initial and auxiliary trees and (b) adjunction operation.

are labeled by symbols in V_N. An initial tree has a root labeled by S and leaf nodes labeled by symbols in $V_T \cup \{\varepsilon\}$. An auxiliary tree has leaf nodes labeled by symbols in $V_T \cup \{\varepsilon\}$ with the addition of one node, called the **foot** node, having the same nonterminal label as the root node (see Figure 1a). We define the size of G, written $|G|$, to be the total number of nodes in all the trees in the set $I \cup A$. In what follows we will also denote by TAG the class of all tree-adjoining grammars.

In TAG, the notion of derivation is based on a composition operation called adjunction, defined in the following way. Let γ be an auxiliary tree having its root (and foot node) labeled by $A \in V_N$. Let also γ' be any tree containing a node η labeled by A, and let τ be the subtree of γ' rooted in η. The **adjunction** of γ into γ' at node η results in a tree specified as follows (see Figure 1b):

(i) the subtree τ is excised from γ';

(ii) the auxiliary tree γ replaces τ in γ', with the root of γ replacing the excised node η;

(iii) the subtree τ is attached to the resulting tree, with the foot node of γ replacing η in τ.

In TAG a **derivation** is the process of recursive composition of elementary trees using the adjunction operation; the resulting trees are called **derived** trees. Since adjunctions at different nodes can be performed in any order, we can adjoin derived trees into derived trees without affecting our arguments.

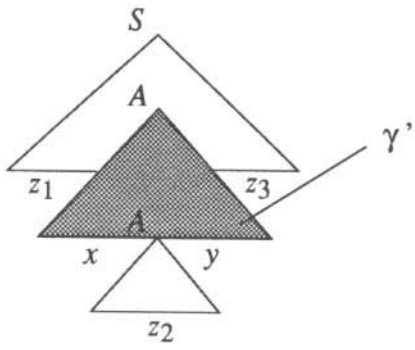

Figure 2
Parse tree γ' is included in a parse tree of string $w = z_1xz_2yz_3$ in $L(G)$. We say that the derivation of string pair $\langle x, y \rangle$ is a subderivation of a sentential derivation of w.

2.3 Tree-Adjoining Grammar Parsing

In order to introduce the definition of the TAG parsing problem on which our results are based, we define in the following the string language derived from a TAG and discuss the notion of parse forest along with the important issue of its representation. Given an alphabet V, we denote by V^* the set of all finite strings over V (null string ε included).

Although TAG is a class of tree rewriting systems, a derivation relation can be defined on strings in the following way. Let γ be an elementary tree and let γ' be a tree obtained from γ by means of zero or more adjunction operations. If the yield of γ' is a string $x \in V_T^*$, that is $\gamma \in I$, we say that γ **derives** x in G. If the yield of γ' is a string $xAy \in V_T^* V_N V_T^*$, that is $\gamma \in A$, we say that γ **derives** the pair $\langle x, y \rangle$ in G. In particular, the set of all strings in V_T^* that can be derived in G is denoted by $L(G)$. In this perspective then, an elementary or a derived tree is seen as a structural description of a string (a pair of strings) derived by the grammar; such a description is called a **parse tree**. The space of all parse trees associated with a given string by the grammar is called a **parse forest**.

We introduce now the notion of subderivation. Let w be a sentence in $L(G)$ and let $\gamma \in A$ derive the pair $\langle x, y \rangle$ in G, $x, y \in V_T^*$, with an associated parse tree γ'. If γ' is included in a parse tree of w, we have that $w = z_1xz_2yz_3$ for some $z_1, z_2, z_3 \in V_T^*$ (see Figure 2). Parse tree γ' represents the contribution of auxiliary tree γ to a derivation of w; we say therefore that the derivation of $\langle x, y \rangle$ from γ is a **subderivation** of a sentential derivation of w. As a consequence of the definition of parse forest, we have that all subderivations of the sentential derivations of w can be read off from the parse forest of w. Part of this information will be used to establish our result, as precisely stated in the next definition. We need some additional notation. Let $w = d_1 d_2 \cdots d_n$, $n > 0$, be a string over some alphabet; symbol ${}_pw_q$ denotes the substring $d_p d_{p+1} \cdots d_q$ for $1 \leq p \leq q \leq n$ and is undefined otherwise.

Definition 1
Let $G = (V_N, V_T, S, I, A)$ be a tree-adjoining grammar and $w \in V_T^*$ be an input string, $|w| = n$, $n > 0$. A **parse relation** $R_p \subseteq A \times \{1..n\}^4$ associated with the pair $\langle G, w \rangle$ is specified as follows. For every auxiliary tree γ in G and for natural numbers p, q, r and

s, $1 \leq p \leq q \leq r \leq s \leq n$, $R_p(\gamma, p, q, r, s)$ holds if and only if:

(i) the pair $\langle {}_pw_q, {}_rw_s \rangle$ can be derived by γ in G, and

(ii) the derivation in (i) is a subderivation of a sentential derivation of w in G.

The goal of a parsing algorithm for TAG is one of constructing a "suitable" representation for the parse forest of a given string, with respect to a given grammar. However, there is no common agreement in the literature on the requirements that such a representation should meet; therefore the issue of the representation of a parse forest deserves some discussion here.

There seems to be a trade-off between computational time and space in choosing among different representations of a parse forest. Note that, from an extreme perspective, the input itself can be considered as a highly compressed representation of the parse forest—one that needs a time-expensive process for parse tree retrieval.[1] More explicit representations offer the advantage of time-efficient retrieval of parse trees, at the cost of an increase in storage resources. In practice, most commonly used algorithms solve the parsing problem for TAGs by computing a superset of a parse relation (defined as above) and by representing it in such a way that its instances can be tested in constant time; such a condition is satisfied by the methods reported in Vijay-Shanker and Joshi (1985), Schabes and Joshi (1988), Palis, Schende, and Wei (1990), Schabes (1991), Lavelli and Satta (1991), Lang (1992), and Vijay-Shanker and Weir (1993). From such a representation, time-efficient computations can be used later to retrieve parse structures of the input string.

On the basis of the previous observation, we assume in the following that the solution of the parsing problem involves (at least) the computation of a representation for R_p such that its instances can be tested in constant time: we base our results on such an assumption. More precisely, an input instance of the **tree-adjoining grammar parsing problem** is defined to be any pair $\langle G, w \rangle$, and the unique solution of such an instance is provided by an explicit representation of relation (set) R_p associated with $\langle G, w \rangle$ as in Definition 1. In what follows, TGP will represent the set of all instances of the tree-adjoining grammar parsing problem.

3. Technical Part

In this section the Boolean matrix multiplication problem is related to the tree-adjoining grammar parsing problem, establishing the major result of this paper. A precise specification of the studied reduction is preceded by an informal discussion of the general idea underlying the construction.

3.1 The Basic Approach

Two maps $\mathcal{F}$ and $\mathcal{G}$ will be studied in this section. Map $\mathcal{F}$ establishes a correspondence between the set BMM and a proper subset of TGP containing, in some sense, its most difficult instances. Conversely, map $\mathcal{G}$ is defined on the set of solutions of all TGP problems in the image of $\mathcal{F}$, and gives values in the set of Boolean square matrices. Maps $\mathcal{F}$ and $\mathcal{G}$ are defined in such a way that, given any algorithm for the solution of the TGP problem, we can effectively construct an algorithm for the solution of the BMM problem using the commutative diagram shown in Figure 3.

1 I owe this observation to Bernard Lang (personal communication).

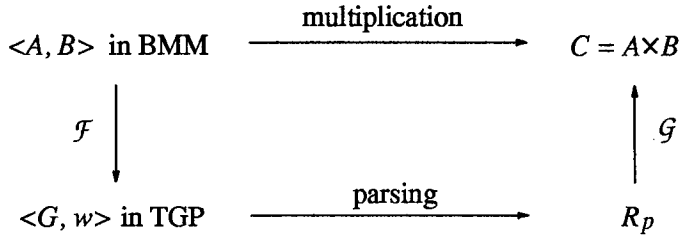

Figure 3
Maps $\mathcal{F}$ and $\mathcal{G}$ define a commutative diagram with respect to any algorithm for Boolean matrix multiplication and any parsing algorithm for tree-adjoining grammars.

Both the BMM and the TGP problems are viewed here as search problems whose solutions are obtained by exploring a search space of elementary combinations. In the case of the BMM problem, the elementary combinations are the combinations of elements of the input matrices. If m is the order of these matrices, the solution of the problem requires the specification of $O(m^2)$ elements of the product matrix, where each element depends upon $O(m)$ elementary combinations (see relation (1)). Therefore the problem involves a search in a space of $O(m^3)$ different combinations. On the other hand, in the TGP problem the elementary combinations are taken to be single applications of the adjunction operation. In parsing a string w of length n according to a tree-adjoining grammar G, we have to construct a parse relation of size $O(|G|n^4)$ (see Definition 1), and there are $O(n^2)$ distinguishable combinations in which each element of the relation can be obtained. In the general case then, a number $O(|G|n^6)$ of distinguishable combinations are involved in the parsing problem, and we have to perform a search within an abstract space of this size.

In order to achieve our result, we then establish a size preserving correspondence between the two search spaces above. There is no way of representing matrices A and B within string w without blowing up the search space associated with the target parsing problem. Our choice will then be to represent the input matrices by means of grammar G, which fixes $|G|$ to a quantity $O(m^2)$. This forces the choice of n to a quantity $O(m^{\frac{1}{6}})$, obtaining therefore the desired relation $|G|n^6 = O(m^3)$.

The general idea underlying the construction is the following one. Observe that non-null elements a_{ik} and $b_{k'j}$ in the input matrices force element c_{ij} to value 1 in the product matrix if and only if $k = k'$. The check of such a condition can be transferred to the computation of an adjunction operation in the target parsing problem using the following encoding method. We fix a positive integer b to a (rounded) quantity $m^{\frac{1}{6}}$. Then we encode each index i of the input matrices by means of positive integers i_1, i_2, and i_3, such that i_1 is $O(b^4)$ and i_2, i_3 are $O(b)$. Condition $k = k'$ above is therefore reduced to the three tests $k_h = k'_h$, $1 \leq h \leq 3$, which can be performed independently. The test $k_1 = k'_1$ is precompiled into some auxiliary tree of G; the tests $k_2 = k'_2$ and $k_3 = k'_3$ are performed by the parser using the input string, as explained below.

Map $\mathcal{F}$ constructs a string w of distinguishable symbols by concatenating six "slices" $w^{(h)}$, $1 \leq h \leq 6$, each slice of length $O(b)$. Map $\mathcal{F}$ also encodes the input matrices A and B within the target grammar G; it does so by transforming each non-null element in the input matrices into an auxiliary tree of G in the following way. Non-null element a_{ik} is mapped into an auxiliary tree γ_1 having its root (and foot node)

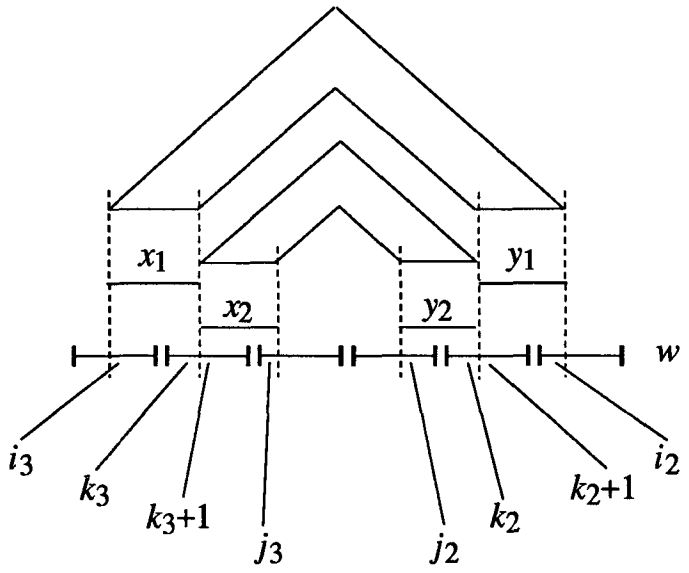

Figure 4
String w is composed of six slices, and auxiliary trees corresponding to non-null elements a_{ik} and $b_{k'j}$ derive string pairs $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ matching the slices of w as shown above; integers are used to indicate the position within a single slice of the boundary symbols in strings x_1, y_1, x_2, and y_2. The figure depicts the case $k = k'$, resulting in the exact nesting of the two derived trees.

labeled by a symbol including integers i_1 and k_1. Moreover, γ_1 will eventually derive a string pair $\langle x_1, y_1 \rangle$ with the following property. String x_1 is the smallest substring of w including the symbol in the i_3th position within slice $w^{(1)}$ and the symbol in the k_3th position within slice $w^{(2)}$. Furthermore, string y_1 is the smallest substring of w including the symbol in the (k_2+1)-th position within slice $w^{(5)}$ and the symbol in the i_2th position within slice $w^{(6)}$. This is schematically shown in Figure 4.

At the same time $\mathcal{F}$ maps non-null element $b_{k'j}$ into an auxiliary tree γ_2 having its root labeled by a symbol including integers k'_1 and j_1. Crucial to our construction, γ_2 will derive a pair of strings $\langle x_2, y_2 \rangle$ with the following property. String x_2 is the smallest substring of w including the symbol in the $(k'_3 + 1)$-th position within slice $w^{(2)}$ and the symbol in the j_3th position within slice $w^{(3)}$. Furthermore, string y_2 is the smallest substring of w including the symbol in the j_2th position within slice $w^{(4)}$ and the symbol in the k'_2th position within slice $w^{(5)}$. Let us call γ'_1 and γ'_2 the derived trees obtained from γ_1 and γ_2 as above. Observe that $k_2 = k'_2$ and $k_3 = k'_3$ if and only if the yields of γ'_1 and γ'_2 are exactly nested within w; see again Figure 4.

To complete the construction of G, map $\mathcal{F}$ provides an auxiliary tree γ_3 with the following property. Tree γ_3 can contribute to a sentential derivation of w in G if and only if γ'_1 and γ'_2 can be adjoined to it. This is in turn possible just in case integer k_1 in the root of γ_1 and integer k'_1 in the root of γ_2 coincide, as specified by the adjunction sites in γ_3, and the yields of γ'_1 and γ'_2 are exactly nested within w. It follows that, by deciding whether γ_3 contributes to a sentential derivation of w, the parser is able to perform the required test $k = k'$. Finally, index k in its coded form is discarded in the derivation process above, while indices i and j are preserved in such a way that map $\mathcal{G}$ can eventually recover non-null element c_{ij} by reading off the parse relation.

Table 1
Values of $f^{(b)}(i)$ for $b = 3$ and $1 \leq i \leq 15$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f_1^{(b)}(i)$	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2
$f_2^{(b)}(i)$	1	1	1	2	2	2	3	3	3	1	1	1	2	2	2
$f_3^{(b)}(i)$	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3

The next section presents a detailed specification of maps $\mathcal{F}$ and $\mathcal{G}$ and proves the above claimed properties. As we will see, the search space defined by the resulting instance of the TAG parsing problem includes the solution of the source matrix multiplication problem.

3.2 The Two Maps

The goal of this subsection is to establish a mapping between comparisons of matrix indices in Boolean matrix multiplication and instances of the adjunction operation in tree-adjoining grammar parsing. As already mentioned in the previous subsection, this result is achieved by encoding natural numbers using three positive integers. The encoding is then used to chop off matrix indices into smaller numbers that will be processed independently. This is explained in detail in the following.

For pairs of integers i and b, let $\mathrm{qn}(i,b)$ and $\mathrm{rm}(i,b)$ be the quotient and the remainder respectively of the integer division of i by b. We define $\mathrm{qn}_+(i,b) = \mathrm{qn}(i,b)+1$ whenever $\mathrm{rm}(i,b) \neq 0$ and $\mathrm{qn}_+(i,b) = \mathrm{qn}(i,b)$ otherwise; we also define $\mathrm{rm}_+(i,b) = \mathrm{rm}(i,b)$ whenever $\mathrm{rm}(i,b) \neq 0$, and $\mathrm{rm}_+(i,b) = b$ otherwise. Note that, for $i \geq 1$, $\mathrm{qn}_+(i,b) \geq 1$ and $1 \leq \mathrm{rm}_+(i,b) \leq b$.

Definition 2
Let $b > 1$ be an integer. We associate with b a function $f^{(b)}$ defined on the set of positive natural numbers, specified as follows:

$$f^{(b)}(i) = \langle f_1^{(b)}(i), f_2^{(b)}(i), f_3^{(b)}(i) \rangle,$$

where

$$\begin{aligned} f_1^{(b)}(i) &= \mathrm{qn}_+(i, b^2), \\ f_2^{(b)}(i) &= \mathrm{qn}_+(\mathrm{rm}_+(i, b^2), b), \\ f_3^{(b)}(i) &= \mathrm{rm}_+(i, b). \end{aligned}$$

Table 1 shows some values of $f^{(b)}$ for the case $b = 3$.

Observe that, for $i \geq 1$, $f_3^{(b)}$ and $f_2^{(b)}$ give values in the range $\{1..b\}$, while $f_1^{(b)}$ can give any positive integer. It is not difficult to see that function $f^{(b)}$ establishes a one-to-one correspondence between the set $\mathbf{N}$ of positive natural numbers and the set $\mathbf{N} \times \{1..b\} \times \{1..b\}$. In an informal way, we will often refer to value $f_1^{(b)}(i)$ as the most significant digit corresponding to i, and to values $f_2^{(b)}(i)$ and $f_3^{(b)}(i)$ as the least significant digits corresponding to i. In the following, the superscript in $f^{(b)}$ will be omitted whenever b can be understood from the context.

We are now in a position to define in detail the maps $\mathcal{F}$ and $\mathcal{G}$ involved in the diagram of Figure 3 discussed in Section 3.1. As a first step, we study map $\mathcal{F}$ that takes as input an instance of BMM and returns an instance of TGP.

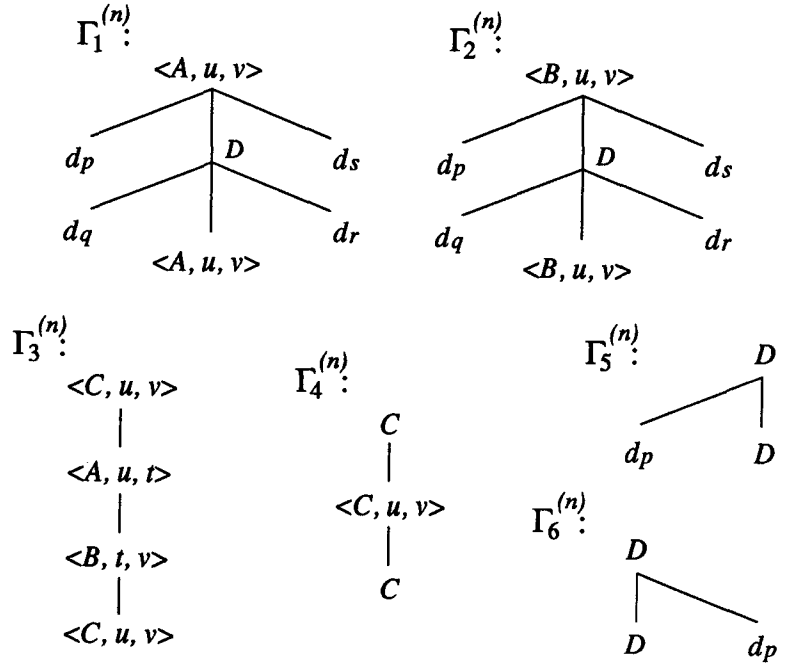

Figure 5
Definition of families of auxiliary trees $\Gamma_h^{(n)}$, $n > 1$ and $1 \leq h \leq 6$. Each tree in some class is specified by the values of the integer parameters corresponding to that class.

Let $n > 1$ be an integer. In the following we will refer to sets of terminal symbols

$$V_{\mathrm{T}}^{(n)} = \{d_p \mid 1 \leq p \leq 6(n+1)\},$$

and to sets of nonterminal symbols

$$V_{\mathrm{N}}^{(n)} = \{\langle A, u, v\rangle, \langle B, u, v\rangle, \langle C, u, v\rangle \mid 1 \leq u \leq v \leq n^4\} \cup \{C, D, S\}.$$

Based on these sets, Figure 5 defines families of auxiliary trees $\Gamma_h^{(n)}$, $1 \leq h \leq 6$. For example, an auxiliary tree $\gamma(p, q, r, s, u, v) \in \Gamma_1^{(n)}$ will be specified by providing actual values for the integer parameters p, q, r, s, u, and v, consistently with the definitions of sets $V_{\mathrm{T}}^{(n)}$ and $V_{\mathrm{N}}^{(n)}$. In the following we will also use the initial tree γ_s depicted in Figure 6.

The next definition introduces map $\mathcal{F}$, which is the core component of the proposed reduction. The definition is rather technical: it will be followed by a more intuitive example.

Definition 3
Let $\langle A, B\rangle$ be an instance of BMM, m the order of matrices A and B. Let also $n = \lfloor m^{\frac{1}{6}} \rfloor + 1$ and $\sigma = n + 1$. A map $\mathcal{F}$ is specified in such a way that $\mathcal{F}(\langle A, B\rangle) = \langle G, w\rangle$, where $G = (V_{\mathrm{N}}^{(n)}, V_{\mathrm{T}}^{(n)}, S, I, A^{(n)})$ and $w = d_1 d_2 \cdots d_{6\sigma}$. Set I contains the only initial tree γ_s; set

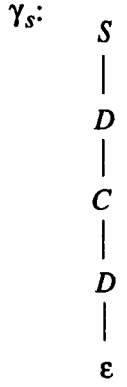

Figure 6
Definition of initial tree γ_s.

$A^{(n)}$ contains all and only the following elementary trees ($f_h^{(n)} = f_h$, $1 \leq h \leq 3$):

(i) for every $a_{ij} = 1$ in A, the auxiliary tree
$\gamma(p, q, r, s, u, v) \in \Gamma_1^{(n)}$
belongs to $A^{(n)}$, where
$p = f_3(i)$, $q = \sigma + f_3(j)$,
$r = 4\sigma + f_2(j) + 1$, $s = 5\sigma + f_2(i)$,
$u = f_1(i)$, $v = f_1(j)$;

(ii) for every $b_{ij} = 1$ in B, the auxiliary tree
$\gamma(p, q, r, s, u, v) \in \Gamma_2^{(n)}$
belongs to $A^{(n)}$, where
$p = \sigma + f_3(i) + 1$, $q = 2\sigma + f_3(j)$,
$r = 3\sigma + f_2(j)$, $s = 4\sigma + f_2(i)$,
$u = f_1(i)$, $v = f_1(j)$;

(iii) for every pair $\langle A, u, t\rangle, \langle B, t, v\rangle \in V_N^{(n)}$, the auxiliary trees
$\gamma(u, t, v) \in \Gamma_3^{(n)}$,
$\gamma(u, v) \in \Gamma_4^{(n)}$
belong to $A^{(n)}$;

(iv) for every $1 \leq p \leq 6\sigma$, the auxiliary trees
$\gamma(p) \in \Gamma_5^{(n)}$,
$\gamma(p) \in \Gamma_6^{(n)}$
belong to $A^{(n)}$.

In order to have a better understanding of map $\mathcal{F}$ and of the idea underlying grammar G and string w, we discuss in the following a simple example, adding more details to the informal discussion presented in Section 3.1.

Let us define a Boolean matrix by specifying only its non-null elements. Assume then that an input instance $\langle A, B\rangle$ of the BMM problem consists of two matrices of

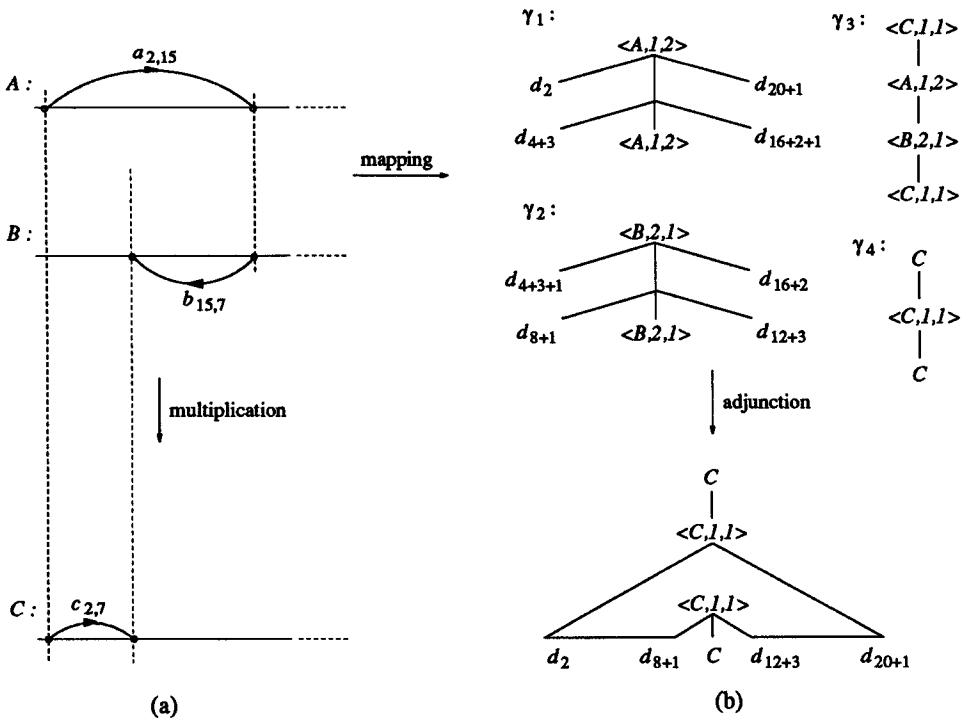

Figure 7
Part (a) shows how non-null elements $a_{2,15}$ and $b_{15,7}$ in matrices A and B combine together, forcing element $c_{2,7}$ in matrix C to value 1; each array element is represented as an arc in a directed graph. Correspondingly, trees γ_1, γ_2, γ_3 and γ_4 in (b) are introduced in G by map $\mathcal{F}$. These trees can be composed, using adjunction, with trees in $\Gamma_5^{(3)}$ and $\Gamma_6^{(3)}$, in such a way that a derived tree is obtained that matches string w and encodes the indices of $c_{2,7}$.

order $m = 64$, specified as

$$A = \{a_{2,15}\}, \qquad B = \{b_{15,7}\}.$$

The multiplication of matrices A and B results in matrix C consisting of the only non-null element $c_{2,7}$. As already mentioned in Section 3.1, the multiplication process can be seen as a test for equality performed on the second index of $a_{2,15}$ and the first index of $b_{15,7}$; in the following these indices will be called "intermediate" indices. Element $c_{2,7}$ in the product matrix is forced to value 1 if this test succeeds, and the intermediate indices are discarded in the process. See Figure 7a for a schematic representation of such an operation.

In performing an adjunction operation, two requirements must be satisfied. First, the nonterminal label at the adjunction site must match the nonterminal label at the root (and foot node) of the adjoined tree; and second, adjunction must compose trees in such a way that the derived string is compatible with w. In the proposed reduction, each test for equality performed on some pair of intermediate indices by matrix multiplication is transferred to an adjunction operation by map $\mathcal{F}$, using as targets the two requirements just described. This is exemplified in the following.

According to Definition 3, we find $n = 3$ and $\sigma = 4$. Map $\mathcal{F}$ then constructs a string $w = d_1 d_2 \cdots d_{24}$, which can be thought of as composed of six slices $w^{(h)} = {}_{4(h-1)+1}w_{4h}$, $1 \leq h \leq 6$. Each element in a single slice will be used as a placeholder to record information about matrix indices. Furthermore, map $\mathcal{F}$ exploits function $f^{(3)}$ (see Table 1) in order to map each non-null element of the input matrices to an auxiliary tree in $\Gamma_1^{(3)}$ or $\Gamma_2^{(3)}$ (steps (i) and (ii) in Definition 3). More specifically, each index of a non-null element is converted into three digits: the most significative one is encoded as part of the nonterminal symbols, and the two least significative digits are encoded by the terminal symbols in the target tree. Trees γ_1 and γ_2 obtained in this way from non-null elements $a_{2,15}$ and $b_{15,7}$ respectively have been depicted in Figure 7b. Two additional trees $\gamma_3 \in \Gamma_3^{(3)}$ and $\gamma_4 \in \Gamma_4^{(3)}$ have been reported in the figure, that are also added to G by $\mathcal{F}$ (step (iii) in Definition 3).

Crucial to our construction, the test on the intermediate indices of elements $a_{2,15}$ and $b_{15,7}$ has been reduced to three independent tests involving smaller integers. More precisely, the equality test on the most significative digits obtained from the intermediate indices has been transferred to the requirement on the matching of the nonterminal labels of the nodes involved in the adjunction. In fact, γ_1 and γ_2 can be adjoined into γ_3 just in case such a test is satisfied. At the same time, the equality test on the least significative digits obtained from the intermediate indices has been transferred to the requirement on the matching of the derived string with w. In fact, after the adjunction of γ_1 and γ_2 into γ_3 takes place, no terminal symbol can intervene between the internal boundaries of the yield of γ_1 and the external boundaries of the yield of γ_2 in slices $w^{(2)}$ and $w^{(5)}$ (see again Figure 7b). Then γ_3 can participate in a sentential derivation of w just in case all three equality tests above are simultaneously satisfied.

The choice of the order of $|w|$ has been dictated by general considerations on the size of the search spaces associated with the two problems at hand, as already discussed in Section 3.1. As a note, we observe that slices $w^{(2)}$ and $w^{(5)}$ are used in the above construction to pair together least significative digits obtained from intermediate indices. The fact that these indices have range in $\{1..n\}$ forces the choice of the length of these slices to $\sigma = n+1$; the example in Figure 7 actually uses the $(n+1)$-th symbol of $w^{(2)}$. For uniformity, this value has then been extended to all other slices, fixing $|w|$ to 6σ.

In Lemma 1 below we will state in a more precise way the above arguments, and we will also show how derivations of the kind outlined above are the only derivations in G than can match string w, proving therefore the correctness of the reduction. To complete the diagram of Figure 3, we now turn to the specification of map $\mathcal{G}$.

Definition 4
Let $\langle G, w \rangle$ be an instance of TGP in the image of map $\mathcal{F}$, and let m, n, and σ be as in Definition 3. Let also R_p be the parse relation that solves instance $\langle G, w \rangle$. A map $\mathcal{G}$ is specified in such a way that $\mathcal{G}(R_p) = C$, C a Boolean matrix of order m, and element c_{ij} is non-null if and only if $R_p(\gamma, p, q, r, s)$ holds for an auxiliary tree $\gamma(u, v) \in \Gamma_4^{(n)}$, where ($f_h^{(n)} = f_h$, $1 \leq h \leq 3$)

$$\begin{array}{llll} p = & f_3(i), & q = & 2\sigma + f_3(j), \\ r = & 3\sigma + f_2(j), & s = & 5\sigma + f_2(i), \\ u = & f_1(i), & v = & f_1(j). \end{array}$$

In the above definition, function $f^{(n)}$ is used to retrieve the indices of non-null elements

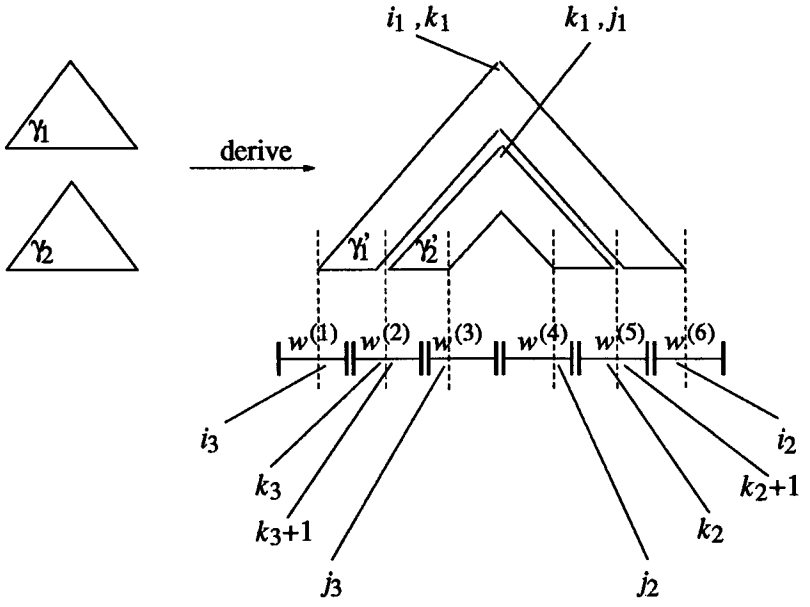

Figure 8
Non-null elements a_{ik} and b_{kj} are mapped into auxiliary trees γ_1 and γ_2 in G, and trees γ_1' and γ_2' can successively be obtained compatibly with string w. As a convention, symbols δ_h, $\delta \in \{i,k,j\}$ and $1 \leq h \leq 3$, denote integers $f_h(\delta)$, which indicate either positions within each single slice or components of nonterminal symbols labeling tree nodes.

of matrix C. In this case also, the most significative digits associated with the retrieved indices are encoded within the nonterminal symbols of the auxiliary tree $\gamma(u,v)$, while the two least significative digits are encoded by the position of the yield boundaries of the string derived from $\gamma(u,v)$ consistently with the input string w. To conclude our previous example, we see that if we apply the relations in Definition 4 to the derived tree at the bottom of Figure 7b, we get indices 2 and 7 of the only non-null element in C.

The following result shows that any algorithm for the solution of a generic instance of TGP can be converted into an algorithm for the solution of the BMM problem, via the computation of maps $\mathcal{F}$ and $\mathcal{G}$. This concludes the present section.

Lemma 1
Let $\langle A,B\rangle$ be an instance of BMM and let $\langle G,w\rangle = \mathcal{F}(\langle A,B\rangle)$. Let also R_p be the parse relation that solves $\langle G,w\rangle$. Then we have

$$A \times B = \mathcal{G}(R_p).$$

Proof
Assume that m is the order of the matrices A and B, n is the natural number associated with m as in Definition 3, and $\sigma = n+1$. Let $C = A \times B$ and $C' = \mathcal{G}(R_p)$.

To prove $c_{ij} = 1$ implies $c'_{ij} = 1$, we go through a sentential derivation of w in G and then apply the definition of $\mathcal{G}$. If $c_{ij} = 1$, then there exists k, $1 \leq k \leq m$, such that $a_{ik} = b_{kj} = 1$. Let γ_1 and γ_2 be the unique auxiliary trees in G associated by map $\mathcal{F}$

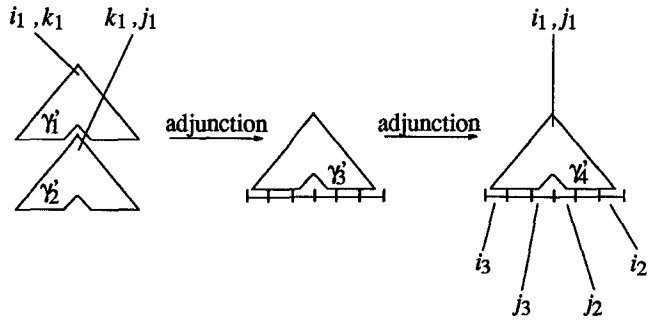

Figure 9
Tree γ_4' is derived from trees γ_1', γ_2' and auxiliary trees in G. We use the same conventions as in Figure 8.

with a_{ik} and b_{kj} respectively (steps (i) and (ii) in Definition 3). Tree γ_1 has root (and foot node) labeled by nonterminal $\langle A, f_1(i), f_1(k)\rangle$; furthermore, the terminal symbols in the yield of γ_1 are (from left to right) $d_{f_3(i)}$, $d_{\sigma+f_3(k)}$, $d_{4\sigma+f_2(k)+1}$ and $d_{5\sigma+f_2(i)}$. The only pair of substrings of w that γ_1 can derive, by means of zero or more adjunctions of trees in $\Gamma_5^{(n)}$ and $\Gamma_6^{(n)}$, is

$$\langle {}_{f_3(i)}w_{\sigma+f_3(k)}, {}_{4\sigma+f_2(k)+1}w_{5\sigma+f_2(i)}\rangle.$$

Call γ_1' a parse tree associated with such a derivation (see Figure 8). In a similar way, auxiliary tree γ_2 has root labeled by nonterminal $\langle B, f_1(k), f_1(j)\rangle$ and derives pair

$$\langle {}_{\sigma+f_3(k)+1}w_{2\sigma+f_3(j)}, {}_{3\sigma+f_2(j)}w_{4\sigma+f_2(k)}\rangle$$

of substrings of w. Call γ_2' a parse tree associated with the derivation (see again Figure 8).

According to step (iii) in Definition 3, grammar G also includes auxiliary trees $\gamma_3 = \gamma(f_1(i), f_1(k), f_1(j)) \in \Gamma_3^{(n)}$ and $\gamma_4 = \gamma(f_1(i), f_1(j)) \in \Gamma_4^{(n)}$. Note that the yields of trees γ_1' and γ_2' are exactly nested within w; moreover, the root (and the foot) nodes of γ_1 and γ_2 have been preserved in the derivation. Therefore γ_1' and γ_2' can be adjoined into γ_3 and the resulting tree γ_3' can in turn be adjoined into γ_4. In this way, γ_4 derives the pair of substrings of w

$$\langle {}_{f_3(i)}w_{2\sigma+f_3(j)}, {}_{3\sigma+f_2(j)}w_{5\sigma+f_2(i)}\rangle.$$

Call γ_4' the resulting derived tree (see Figure 9). Since derived tree γ_4' can be adjoined into γ_s in G and a tree can be eventually derived for the input string w, we have

$$R_p(\gamma_4, f_3(i), 2\sigma + f_3(j), 3\sigma + f_2(j), 5\sigma + f_2(i)),$$

and from the definition of $\mathcal{G}$ we get $c'_{ij} = 1$.

Assuming $c'_{ij} = 1$, we now prove $c_{ij} = 1$; this is done by arguing that the only sentential derivations for w that are allowed by G are those of the kind outlined above. From the definition of $\mathcal{G}$ we have that

$$R_p(\gamma_4, f_3(i), 2\sigma + f_3(j), 3\sigma + f_2(j), 5\sigma + f_2(i))$$

holds for the auxiliary tree $\gamma_4 = \gamma(f_1(i), f_1(j)) \in \Gamma_4^{(n)}$. Equivalently, there exists at least one derivation from γ_4 of strings

$$\langle x, y \rangle = \langle_{f_3(i)} w_{2\sigma+f_3(j)}, {}_{3\sigma+f_2(j)} w_{5\sigma+f_2(i)} \rangle \tag{2}$$

that participates in a sentential derivation of w. Fix such a derivation.

We first observe that, in order to derive any terminal symbol from γ_4, auxiliary trees in $\Gamma_1^{(n)}$, $\Gamma_2^{(n)}$ and $\Gamma_3^{(n)}$ must be used. Any tree in $\Gamma_1^{(n)}$ can only derive symbols in slices $w^{(h)}$, $h \in \{1,2,5,6\}$, and any tree in $\Gamma_2^{(n)}$ can only derive symbols in slices $w^{(h)}$, $h \in \{2,3,4,5\}$. Therefore at least one tree in $\Gamma_1^{(n)}$ and at least one tree in $\Gamma_2^{(n)}$ must be used in the derivation of $\langle x, y \rangle$, since $\langle x, y \rangle$ includes terminal symbols from every slice of w. Furthermore, if more than one tree in $\Gamma_1^{(n)}$ is used in a derivation in G, the resulting string cannot match w. The same argument applies to trees in $\Gamma_2^{(n)}$. We must then conclude that exactly one tree in $\Gamma_1^{(n)}$, one tree in $\Gamma_2^{(n)}$, and one tree in $\Gamma_3^{(n)}$ have been used in the derivation of $\langle x, y \rangle$ from γ_4. Call the above trees $\gamma_1 = \gamma(p, k_3, k_2+1, s, u, k_1) \in \Gamma_1^{(n)}$, $\gamma_2 = \gamma(k'_3+1, q, r, k'_2, k'_1, v) \in \Gamma_2^{(n)}$, and $\gamma_3 = \gamma(u', t, v') \in \Gamma_3^{(n)}$.

As a second step, we observe that γ_3 can be adjoined into γ_4 only if $u' = f_1(i)$ and $v' = f_1(j)$ and γ_3 can host γ_1 and γ_2 just in case $u' = u$, $v' = v$, and $k_1 = k'_1 = t$. We also observe that, after these adjunctions take place, the leftmost terminal symbol in the yield of γ_4 will be the leftmost terminal symbol in the yield of γ_1, that is d_p. From relation (2) we then conclude that $p = f_3(i)$. Similarly, we can argue that $q = 2\sigma + f_3(j)$, $r = 3\sigma + f_2(j)$ and $s = 5\sigma + f_2(i)$. Finally, adjunction of γ_1 and γ_2 into γ_3 can match w just in case $k_3 = k'_3$ and $k_2 = k'_2$.

From the relations inferred above, we conclude that we can rewrite γ_1 as $\gamma(f_3(i), k_3, k_2+1, 5\sigma + f_2(i), f_1(i), k_1) \in \Gamma_1^{(n)}$ and γ_2 as $\gamma(k_3+1, 2\sigma + f_3(j), 3\sigma + f_2(j), k_2, k_1, f_1(j)) \in \Gamma_2^{(n)}$. Since f is one-to-one and $k_2, k_3 \in \{1..n\}$, there exists k such that $f(k) = \langle k_1, k_2, k_3 \rangle$. From steps (i) and (ii) in Definition 3, we then have that a_{ik} and b_{kj} are non-null and then $c_{ij} = 1$. □

4. Computational Consequences

The results presented in the previous section are developed here under a computational perspective. Some interesting computational consequences will then be drawn for the tree-adjoining grammar parsing problem. The following analysis assumes the random-access machine as the model of computation.

4.1 Transferring of Time Upper Bounds

We show in the following how time upper bounds for the TGP problem can be transferred to time upper bounds for the BMM problem using the commutative diagram studied in the previous section.

Let $\langle A, B \rangle$ be an instance of BMM and let $\langle G, w \rangle = \mathcal{F}(\langle A, B \rangle)$; m and n are specified as in Definition 3. Observe that, since $n^6 > m$, function $f^{(n)}$ maps set $\{1..m\}$ into product set $\{1..n^4\} \times \{1..n\} \times \{1..n\}$, in other words we have $1 \leq f_1(i) \leq n^4$ and $1 \leq f_2(i), f_3(i) \leq n$

for $1 \leq i \leq m$. From the definition of $\mathcal{F}$, we see that G contains $O(m^2)$ auxiliary trees from each of the classes $\Gamma_1^{(n)}$, $\Gamma_2^{(n)}$ and $\Gamma_3^{(n)}$. This determines the size of G and we have $|\langle G, w\rangle| = O(m^2)$, since $|w| = O(n)$. Each auxiliary tree introduced in G at steps (i) and (ii) of Definition 3 requires the computation of a constant number of instances of function $f^{(n)}$ on some integer i, $1 \leq i \leq m$. Such a computation can be carried out in an amount of time $O(\log^2(m))$ using standard algorithms for integer division. Summing up, the entire computation of $\mathcal{F}$ on an instance $\langle A, B\rangle$ takes time $O(m^2 \log^2(m))$.

Let R_p be the parse relation that solves $\langle G, w\rangle = \mathcal{F}(\langle A, B\rangle)$. From Definition 1 and the above observations we have that $|R_p| = O(m^2 n^4)$, that is $|R_p| = O(m^{2+\frac{2}{3}})$. We can compute $C = \mathcal{G}(R_p)$ in the following way. For every element c_{ij} we compute $f^{(n)}(i)$ and $f^{(n)}(j)$ and then check R_p according to Definition 4. (Recall also our assumption that an instance of R_p can be tested in constant time.) Again we find that the entire computation takes an amount of time $O(m^2 \log^2(m))$. We observe that the computation of $\mathcal{F}$ and $\mathcal{G}$ takes an amount of time (asymptotically) very close to the one needed to store $\langle A, B\rangle$ or C.

As a consequence of the above discussion and of Lemma 1, we have that any time upper bound for the TGP problem can be transferred to an upper bound for the BMM problem, down to the time needed for the computation of transformations $\mathcal{F}$ and $\mathcal{G}$. The following statement gives an example.

Theorem 1
Let A_P be an algorithm for the solution of the TGP problem having running time $O(|G|^p |w|^q)$. Then any instance of BMM can be solved in time $O(\max\{m^{2p+\frac{q}{6}}, m^2 \log^2(m)\})$, where m is the order of the input matrices.

Proof
From Lemma 1 and from the previous discussion we have that two Boolean matrices of order m can be multiplied in time $O(|G|^p |w|^q + m^2 \log^2(m))$, where $|G| = O(m^2)$ and $|w| = O(m^{\frac{1}{6}})$. □

Observe that, according to our definition, the TGP problem has a trivial time lower bound $O(|R_p|)$, since this is the amount of time needed in the worst case to store a representation for R_p that can be accessed in constant time. In practice this means that the upper bound transfer stated by the above result is effective down to $O(m^{2+\frac{2}{3}})$.

4.2 Time Upper Bounds for TGP

In previous sections we have related the complexity of tree-adjoining grammar parsing to the complexity of Boolean matrix multiplication. Here we speculate on the consequences of the presented result.

As a computational problem, Boolean matrix multiplication has been an object of investigation for many years. Researchers have tried to improve the well-known $O(m^3)$ time upper bound, m the order of the input matrices, and methods were found that work asymptotically faster than the standard cubic time algorithm. Strassen's divide and conquer algorithm that runs in time $O(m^{2.81})$ (see for instance Cormen, Leiserson, and Rivest [1990]) has been the first one in the series, and the best time upper bound known to date is approximately $O(m^{2.376})$, as reported in Coppersmith and Winograd (1990). It is worth noting here that the closer researchers have come to the $O(m^2)$ trivial time lower bound, the more complex the computation involved in these methods has become. In fact, if Strassen's algorithm outperforms the $O(m^3)$ standard algorithm only

for input matrices of order greater than 45 or so (see again Cormen, Leiserson, and Rivest [1990]), recently discovered methods that are asymptotically faster are definitely prohibitive, given current computer hardware. At present, no straightforward method is known for Boolean matrix multiplication that considerably improves the cubic upper bound and that can be used in practical cases. Also, there is enough evidence that, if such a method exists, its discovery should be a very difficult enterprise.

Let us now turn to the TAG parsing problem. Many algorithms have been proposed for its solution and an $O(|G||I \cup A||w|^6)$ time upper bound has been given in the literature; see for instance Schabes (1990). We remark here that the dependency on the grammar size can be further improved using techniques similar to the one proposed in Graham, Harrison, and Ruzzo (1980) for the context-free grammar recognition/parsing problem: this results in an $O(|G||w|^6)$ time upper bound for the general case. Theorem 1 can be used to transfer this upper bound to an upper bound for Boolean matrix multiplication, finding the already mentioned $O(m^3)$ result.

More interestingly, Theorem 1 implies that any method for the solution of the tree-adjoining grammar parsing problem having running time $O(|G||w|^5)$ will give us a method for Boolean matrix multiplication having running time $O(m^{2.8\bar{3}})$. Likewise, any $O(|G||w|^4)$ time method for the former problem will result in an $O(m^{2.\bar{6}})$ time method for Boolean matrix multiplication. Even if the involved constants hidden in the studied construction are large, the resulting methods will still be competitive with known methods for Boolean matrix multiplication that improve the cubic time upper bound. We conclude then that the TAG parsing problem should also be considered as having the status of a problem that is "difficult" to improve, and we have enough evidence to think that methods for TAG parsing that are asymptotically faster than $O(|G||w|^6)$ are unlikely to be practical, i.e., will involve rather complex computations.

5. Remarks and Conclusion

Polynomial time reductions between decision/search problems are commonly used in providing hardness results for complexity classes not known to be included in P (P is the class of all languages decidable in deterministic polynomial time). We have studied here a polynomial time reduction between Boolean matrix multiplication and TAG parsing, two problems already known to be in P. However, the choice of the mapping allows one to transfer upper bounds from the first problem to the other. In this way TAG parsing inherits from Boolean matrix multiplication the reputation of being a problem tough to improve. We comment in the following on the significance of this result.

As already discussed, the notion of the parse forest is an informal one, and there is no common agreement on which specifications such a structure should meet. The obtained results are based on the assumption that a parsing algorithm for TAG should be able to provide a representation for a parse forest such that instances of the parse relation R_p in Definition 1 can be retrieved in constant time. Whatever the specifications of the output parse forest structure will be, it seems quite reasonable to require that an explicit representation of relation R_p can be extracted from the output in linear time with respect to the size of the output itself, therefore without affecting the overall running time of the method. This requirement is satisfied by all TAG parsers that have been presented to date in the literature.

As a second point, the studied construction provides an interesting insight into the structure of the TAG parsing problem. We see for instance that the major source of

complexity derives from cases of properly nested adjunction operations. Such cases are responsible for a bounded amount of nondeterminism in the computation: to detect how a string divides into subparts according to the adjunction of a derived tree into another, we have to consider many possibilities in general, as much as we do to detect a non-null element within a product Boolean matrix. A closer look at the studied construction reveals also that the parsing problem for linear TAG does not seem easier than the general case, since $\mathcal{F}$ maps instances of BMM to instances of TGP restricted to such a class (a linear TAG is a TAG whose elementary trees allow adjunction only into nodes along a single spine). This contrasts with the related case of context-free grammar parsing, where the restriction of the problem to linear grammars can be solved in time $O(|G||w|^2)$ but no method is known for the general case working with this bound. As expected from our result, the techniques that are used for linear context-free grammar parsing cannot be easily generalized to improve the parsing problem for linear TAGs with respect to the general case.

Finally, we want to discuss here an interesting extension of the studied result. The TAG parsing problem can be generalized to cases in which the input is a lattice representation of a string of terminal symbols along with a partially specified parse relation associated with it. This has many applications for ill-formed input and error-correcting parsing. The TAG lattice parsing problem can still be solved in $O(|G||w|^6)$ time: the general parsing method provided in Lang (1992) can be used to this purpose, and already known tabular methods for TAG parsing can be easily adapted as well.

Without giving the technical details of the argument, we sketch here how Boolean matrix multiplication can be related to TAG lattice parsing. For order m matrices, one can use an encoding function $f_l^{(n)}$, where $n = \lfloor m^{1/2} \rfloor + 1$, mapping set $\{1..m\}$ into product set $\{1..n\} \times \{1..n\}$. This allows a direct encoding of any instance $\langle A, B \rangle$ of the BMM problem into a word lattice w_l consisting of $6(n+1)$ nodes and $O(m^2)$ arcs, where some arcs involve four nodes and represent a derived tree corresponding to a non-null element in either A or B. Then we can use a grammar G in the target instance of the TAG lattice problem that is defined independently of $\langle A, B \rangle$ and therefore has constant size. (Such a grammar can be obtained from families $\Gamma_3^{(n)}$ and $\Gamma_4^{(n)}$ defined in Section 3 by deleting the integer components in each nonterminal symbol.) The construction obtained in this way relates therefore the BMM problem to the fixed grammar parsing problem, and provides a result even stronger than the one presented in Theorem 1. We have in fact that any algorithm for TAG lattice parsing having running time $O(|G|^p|w|^q)$ can be converted into an algorithm for Boolean matrix multiplication running in time $O(\max\{m^{\frac{q}{2}}, m^2 \log^2(m)\})$, independently of p. As an example, $O(|G|^p|w|^4)$ for TAG lattice parsing becomes $O(m^2 \log^2(m))$ for matrix multiplication, for any p. Since many tabular methods for TAG parsing can be easily extended to TAG lattice parsing, this means that the chances of getting an $O(|G|^p|w|^4)$ time upper bound for the TAG parsing problem itself by means of these techniques are really small.

Acknowledgments

I am indebted to Yves Schabes who suggested to me the original idea of relating a standard computational problem to the tree-adjoining grammar parsing problem. I want to thank Bernard Lang, Owen Rambow, and Yves Schabes, who have provided, directly or indirectly, important suggestions for the development of the ideas in this paper. Comments from two anonymous referees have also been very helpful in improving the exposition of the results reported in this paper. Finally, I am grateful to Aravind Joshi for his support in this research. None of these people is responsible for any error in this work. This research was partially funded by the following grants: ARO grant DAAL

03-89-C-0031, DARPA grant N00014-90-J-1863, NSF grant IRI 90-16592, and Ben Franklin grant 91S.3078C-1.

References

Coppersmith, D., and Winograd, S. (1990). "Matrix multiplication via arithmetic progression." *Journal of Symbolic Computation*, 9(3), 251–280. Special Issue on Computational Algebraic Complexity.

Cormen, T. H.; Leiserson, C. E.; and Rivest, R. L. (1990). *Introduction to Algorithms*. The MIT Press.

Frank, R. (1992). *Syntactic locality and tree adjoining grammar: grammatical, acquisition and processing perspectives*. Doctoral dissertation, University of Pennsylvania.

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

Joshi, A. K. (1985). "How much context-sensitivity is necessary for characterizing structural descriptions—tree adjoining grammars. In *Natural Language Processing—Theoretical, Computational and Psychological Perspectives*, edited by D. Dowty, L. Karttunen, and A. Zwicky, 206–250. Cambridge University Press. Originally presented in a Workshop on Natural Language Parsing at Ohio State University, Columbus, Ohio, May 1983.

Joshi, A. K.; Levy, L. S.; and Takahashi, M. (1975). "Tree adjunct grammars." *Journal of Computer System and Science*, 10(1), 136–163.

Joshi, A.; Vijay-Shanker, K.; and Weir, D. (1991). "The convergence of mildly context-sensitive grammatical formalisms." In *Foundational Issues in Natural Language Processing*, edited by S. Shieber and T. Wasow, 31–82. MIT Press.

Lang, B. (1992). "Recognition can be harder than parsing." Abstract submitted to the Second TAG Workshop, June 1992.

Lavelli, A., and Satta, G. (1991). "Bidirectional parsing of lexicalized tree adjoining grammars." In *Proceedings, Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Berlin, 1991, 27–32.

Palis, M. A.; Shende, S.; and Wei, D. S. L. (1990). "An optimal linear-time parallel parser for tree-adjoining languages." *SIAM Journal on Computing*, 19(1), 1–31.

Schabes, Y. (1990). *Mathematical and computational aspects of lexicalized grammars*. Doctoral dissertation, University of Pennsylvania. Available as technical report (MS-CIS-90-48, LINC LAB179) from the Department of Computer Science.

Schabes, Y. (1991). "The valid prefix property and left to right parsing of tree-adjoining grammar." In *Proceedings, Second International Workshop on Parsing Technologies*, Cancun, Mexico, February 1991, 21–30.

Schabes, Y., and Joshi, A. K. (1988). "An Earley-type parsing algorithm for tree adjoining grammars." In *Proceedings, 26th Meeting of the Association for Computational Linguistics*, Buffalo, June 1988, 258–269.

Vijay-Shanker, K., and Joshi, A. K. (1985). "Some computational properties of tree adjoining grammars." In *Proceedings, 23rd Meeting of the Association for Computational Linguistics*, Chicago, July 1985, 82–93.

Vijay-Shanker, K., and Weir, D. J. (1993). "The use of shared forests in TAG parsing." In *Proceedings, Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht, 1993, 384–393.