A bisection algorithm for grammar-based compression of ordered trees

Tatsuya Akutsu
Bioinformatics Center, Institute for Chemical Research, Kyoto University,
Uji, Kyoto 611-0011, Japan.
e-mail: takutsu@kuicr.kyoto-u.ac.jp

keywords: approximation algorithms, graph algorithms, tree grammars, data compression

1 Introduction

Among various approaches to data compression for text data, extensive studies have been done on grammar-based compression. Grammar-based compression is to find a small grammar generating a given string, and is useful not only for data compression but also for pattern extraction. From a theoretical viewpoint, it is known that finding the smallest context-free grammar (CFG) is NP-hard but it is approximated in polynomial time within a factor of $O(\log(n/m^*))$, where n is the size of an input data and m^* is the size of the smallest grammar [3, 6, 8].

It is reasonable to try to extend grammar-based compression for tree structured data. Indeed, various grammars and algorithms have been developed for that purpose [2, 7, 9]. However, to my knowledge, no algorithm has been known with a guaranteed approximation ratio. In this paper, we mainly consider rooted ordered trees. We define an *elementary ordered tree grammar* (EOTG) by extending CFG, and then present a polynomial time algorithm which approximates the smallest EOTG within a factor of $O(n^{5/6})$. We also show that the grammar and algorithm can be modified for rooted unordered trees of bounded degree.

2 Preliminaries

In this paper, we consider *rooted ordered trees* unless otherwise stated. For a tree T, $V(T)$, $E(T)$ and $r(T)$ denote a set of *nodes*, a set of *edges* and the *root* of T, respectively. The *size* of T is the number of nodes in T and is denoted by $|T|$. For a node v in a tree T, $T(v)$ denotes the subtree of T induced by v and its descendants. $T - T(v)$ and $T - T(v) \cup \{v\}$ denote

the subtrees of T induced by $V(T) - V(T(v))$ and by $V(T) - V(T(v)) \cup \{v\}$, respectively. The *depth* of a node v is the number of edges in the path from the root to v and is denoted by $d(v)$. $V_d(T)$ denotes the set of nodes of depth d in a tree T. For a node u and its children $v_1, \ldots, v_g$, $sub(u, v_{i_1}, \ldots, v_{i_h})$ denotes the subtree induced by u, $v_{i_1}, \ldots, v_{i_h}$ and the descendants of $v_{i_1}, \ldots, v_{i_h}$, where $(v_{i_1}, \ldots, v_{i_h})$ is a subsequence of $(v_1, \ldots, v_g)$. For a string s, $s[i]$ and $s[i, j]$ denote the ith letter of s and a substring between the ith and jth positions of s, respectively.

For simplicity, we treat each tree T as an edge labeled tree and let Σ be the set of edge labels. Node labeled trees can be transformed into edge labeled trees by assigning a label of a node (except the root) to the edge between the node and its parent. The depth-first search traversal of T (i.e., visiting children of each node according to their left-to-right order) gives an Euler tour beginning from the root and ending at the root where each edge $\{w, v\}$ is traversed twice in the opposite directions. Let $\Sigma' = \{a, \overline{a} | a \in \Sigma\}$, where $\overline{a} \notin \Sigma$. Let $(e_1, e_2, \ldots, e_{2n-2})$ be the sequence of directed edges in the Euler tour of T of size n. From this, we create the *Euler string* $es(T)$ of length $2n - 2$ over Σ'. Let $e = \{u, v\}$ be an edge in T, where u is the parent of v. Suppose that $e_i = (u, v)$ and $e_j = (v, u)$. It is to be noted that $i < j$ holds since e_is are ordered according to the Euler tour of T. Then, we define $i_1(e)$ and $i_2(e)$ by $i_1(e) = i$ and $i_2(e) = j$, respectively. We define $es(T)$ by letting $es(T)[i_1(e)] = \ell(e)$ and $es(T)[i_2(e)] = \overline{\ell(e)}$, where $\ell(e)$ is the label of e. It is known that T_1 is isomorphic to T_2 (including label information) if and only if $es(T_1) = es(T_2)$ [1].

Of course, we can apply existing grammar-based string compression algorithms to Euler strings in order to compress trees. Since our tree grammars can be transformed into CFGs as shown in the proof of Lemma 2, such an approach may yield better compression performances. However, in such a case, derived grammars do not necessarily correspond to tree grammars. As discussed in [7, 9], the purpose of grammar-based tree compression is not only to compress input trees but also to extract features (e.g., patterns) from input trees. Therefore, we need to obtain tree grammars from input trees.

3 Elementary Ordered Tree Grammar

We consider two types of trees: *tagged trees* and *non-tagged trees*. A non-tagged tree is a usual tree, where either a *terminal symbol* or a *nonterminal symbol* is attached as a label to each edge. A tagged tree is the same as a non-tagged tree except that exactly one leaf node is

tagged. An edge whose lower endpoint is a tagged node is called a *tagged edge*. The restriction of the number of tagged nodes is important. If multiple tagged nodes were allowed per tree, the resulting grammars would become more complicated and the time complexity of parsing would become much higher because we might examine much more numbers of combinations in parsing. Furthermore, it is unclear whether we can design a compression algorithm with a guaranteed approximation ratio because our proposed algorithm heavily depends on the fact that there exists at most one tagged node in a tree. We use a capital letter to denote a nonterminal symbol and a lower-case letter to denote a terminal symbol. We may identify an edge with its label, and a tree with its Euler string.

We consider the following two types of production rules for trees (see also Fig. 1)

(R1) Replace a non-tagged edge by a non-tagged tree T,

(R2) Replace a tagged edge by a tagged tree T_x,

where we start with a tree consisting of a non-tagged edge with the start symbol S.

For a tagged tree T_x, $es(T_x)$ also denotes the Euler string of T_x except that the tagged edge with label A in T_x is transformed into $Ax\overline{A}$, where x is the special symbol denoting the tag. We have an Euler string version of production rules as: **(R1')** $A\overline{A} \to es(T)$, **(R2')** $Ax\overline{A} \to es(T_x)$. The *size of a grammar* is defined as the total number of letters (in Euler strings) appearing in the right hand sides (RHSs) of rules excluding the tag symbol. That is, the size is the double of the number of edges in trees appearing in RHSs. An *EOTG* is defined by a 4-tuple $(\Sigma, \Gamma, S, \Delta)$ where Σ, Γ, $S \in \Gamma$ and Δ are a set of terminal symbols, a set of nonterminal symbols, the start symbol and a set of production rules, respectively. When we discuss compression algorithms, as in [3], we only consider EOTGs satisfying (i) each nonterminal appears in LHS of exactly one rule, and (ii) there exists an ordering of the nonterminals Γ such that each nonterminal precedes all nonterminals in its definition (i.e., the grammar is acyclic). Due to these properties, it is guaranteed that a grammar generates exactly one finite-size tree.

In this paper, we consider a special class of EOTG in which only the following types of production rules are allowed

(I) $A\overline{A} \to a\overline{a}$, **(I')** $Ax\overline{A} \to ax\overline{a}$, **(II)** $A\overline{A} \to BC\overline{CB}$, **(II')** $Ax\overline{A} \to BCx\overline{CB}$,
(III) $A\overline{A} \to B\overline{B}C\overline{C}$, **(IIIA)** $Ax\overline{A} \to B\overline{B}Cx\overline{C}$, **(IIIB)** $Ax\overline{A} \to Bx\overline{B}C\overline{C}$.

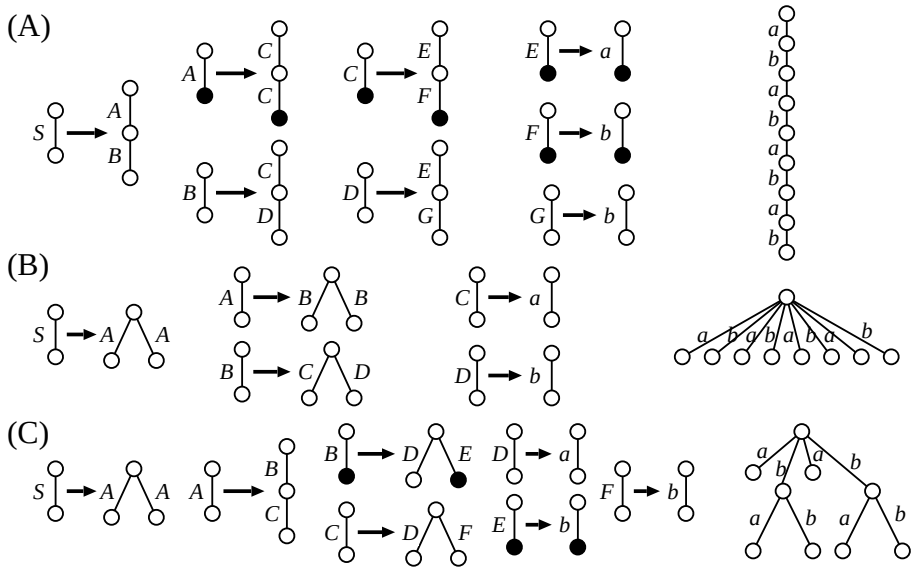

Figure 1: Examples of SEOTGs (left) and generated trees (right). Black nodes denote tagged nodes.

This restricted version of EOTG is referred as SEOTG (*Simple EOTG*). From examples (A) and (B) of Fig. 1, it is seen that EOTG is an extension of CFG for both vertical and horizontal directions. Though the number of rules of grammar (A) is greater than that for CFG (because of handling of tags), it is at most double.

Lemma 1 *Any EOTG of size m can be transformed into an SEOTG of size at most $3m$ that generates the same set of trees as EOTG does.*

Proof. We only show a recursive procedure to transform $A \to T_x$, where T_x is a tagged tree of size at least 3. Then, it is straight-forward to extend the procedure for all cases.

If there is only one child v of $r = r(T_x)$, we add $Ax\overline{A} \to BCx\overline{CB}$, where B and C are nonterminal edges for generating an edge (r, v) and a subtree rooted at v, respectively. Otherwise, suppose that there are g children $v_1, \ldots, v_g$ of r. If $T_x(v_1)$ is a tagged tree, we add $Ax\overline{A} \to Bx\overline{B}C\overline{C}$, otherwise we add $Ax\overline{A} \to B\overline{B}Cx\overline{C}$, where B and C correspond to $sub(r, v_1)$ and $sub(r, v_2, \ldots, v_g)$, respectively.

Suppose that T_x consists of m' edges. Then, the corresponding EOTG rule has size $2m'$. The number of production rules of the former type (including rules without tags) generated from T_x is bounded by m' because there exist at most m' edges in T_x labeled with nonterminal symbols. The number of production rules of the latter type is bounded by $m' - 1$ because

there exist at most m' edges in T_x, and each production rule partitions a relevant set of edges into two disjoint sets of edges[1]. Therefore, the total size of the resulting production rules for T_x is bounded by $2m' + 4(m' - 1) = 6m' - 4$, which is smaller than $3 \times 2m'$. By summing the sizes of all resulting rules, we have the lemma. □

4 Parsing Algorithm

Before discussing the compression algorithm, we show that parsing of a string for any EOTG (including ambiguous and cyclic cases) can be done in polynomial time using a dynamic programming (DP) algorithm. Based on Lemma 1, we only present an algorithm for SEOTG. For each nonterminal symbol A for rules of type (I'), (II'), (IIIA) and (IIIB) (resp. type (I), (II) and (III)), we construct a table $A[i,h,k,j]$ (resp. $A[i,j]$) where $i \leq h \leq k \leq j$. $A[i,h,k,j] = 1$ if $es(T)[i,h]$ and $es(T)[k,j]$ are derived from $Ax\overline{A}$, where the concatenation of $es(T)[i,h]$ and $es(T)[k,j]$ corresponds to a subtree, and $es(T)[h+1,k-1]$ corresponds to a subtree rooted at the lower endpoint of an edge corresponding to $es(T)[h]$ and $es(T)[k]$. Suppose that RHS of $Ax\overline{A}$ is of type (II'). Then, $A[i,h,k,j]$ can be computed by the following DP procedure

$$A[i,h,k,j] = \begin{cases} 1 & \text{if } (\exists g,f)(B[i,g,f,j] = 1 \text{ and } C[g+1,h,k,f-1] = 1), \\ 0 & \text{otherwise.} \end{cases}$$

For other type rules, $A[i,h,k,j]$ (or $A[i,j]$) can also be computed in a similar way. Since the size of $A[i,h,k,j]$s is $O(mn^4)$ and the time required per entry is $O(n^2)$ where m is the size of EOTG and $n = |T|$, the following theorem holds.

Theorem 1 *Whether or not a given tree T is generated from a given EOTG can be decided in $O(mn^6)$ time.*

5 Compression Algorithm

The compression algorithm is based on BISECTION [3, 5] and is denoted by TREE-BISECTION here. TREE-BISECTION recursively decomposes a given tree T_0 into smaller subtrees (see Fig. 2) until each subtree consists of an edge.

As a base case, suppose that the current tree T consists of an edge with label a. Then, we add the rule of $Ax\overline{A} \to ax\overline{a}$ if T is a tagged tree, and $A\overline{A} \to a\overline{a}$ otherwise.

[1]This property can be seen from the fact that every binary tree with m' leaves has $m' - 1$ internal nodes, where leaves correspond to edges in T_x and internal nodes correspond to production rules.

Next, suppose that T is a non-tagged tree of size greater than 2. Let r be the root of T. Let $u_1, \ldots, u_h$ be the children of a node u. Then, u_j is called the *heaviest* child (among $u_1, \ldots, u_h$) if $|T(u_j)|$ is largest. Let $(v_0, v_1, v_2, \ldots, v_g)$ be a *heavy chain* of T, which is constructed by following the heaviest children from the root $v_0 = r$. Let v_i be the first node such that $|T(v_i)| \leq \frac{1}{2}|T|$. Here we let $v = v_{i-1}$. We partition T into $T_1 = T - T(v) \cup \{v\}$ and $T_2 = T(v)$, where T_1 becomes a tree with tagged v, and $|T_1| \leq \frac{1}{2}|T| + 1$. Let $w_1, \ldots, w_h$ be the children of v. Then, $|T(w_l)| \leq \frac{1}{2}|T|$ holds for all w_l. Next, we find w_j that minimizes

$$| \ |sub(v, w_1, \ldots, w_j)| \ - \ |sub(v, w_{j+1}, \ldots, w_h)| \ |,$$

where the tie is broken arbitrarily. Then, we partition T_2 into $T_3 = sub(v, w_1, \ldots, w_j)$ and $T_4 = sub(v, w_{j+1}, \ldots, w_h)$. From $|T(w_l)| \leq \frac{1}{2}|T|$ for all w_l, we can see that $|T_3| \leq \frac{3}{4}|T| + 1$ and $|T_4| \leq \frac{3}{4}|T| + 1$ hold. The associated rules of SEOTG are created accordingly (see Fig. 2 (A)).

Finally, suppose that T is a tagged tree of size greater than 2. Let r be the root of T. Let $(v_0 = r, v_1, v_2, \ldots, v_g = x)$ be the path from the root to the tagged node x. Let v_i be the first node in the path such that $|T(v_i)| \leq \frac{1}{2}|T|$. Here we let $v = v_{i-1}$. As in the case of non-tagged tree, we partition T into $T_1 = T - T(v) \cup \{v\}$ and $T_2 = T(v)$. Let $w_1, \ldots, w_h$ be the children of v, and $w_j = v_i$. Then, we partition T_2 into $T_3 = sub(v, w_1, \ldots, w_j)$ and $T_4 = sub(v, w_{j+1}, \ldots, w_h)$ if $|sub(v, w_1, \ldots, w_j)| < |sub(v, w_j, \ldots, w_h)|$. Otherwise, we partition T_2 into $T_3 = sub(v, w_1, \ldots, w_{j-1})$ and $T_4 = sub(v, w_j, \ldots, w_h)$. Here, we assume without loss of generality that T_4 contains w_j. From $|T_4| \leq |T_3|$, we can see that $|T_4| \leq \frac{1}{2}|T|+1$ holds. Though the size of T_3 may be close to $|T|$, T_3 is a non-tagged tree and thus is decomposed into subtrees whose sizes are not greater than $\frac{3}{4}|T|$ in the next recursive step. The associated rules of SEOTG are created accordingly (see Fig. 2 (B)). It is to be noted that each of T_1, T_2, T_3 and T_4 contains at most one tagged node.

There are some exceptional cases: either T_1 is empty or v_{i-1} has only one child. In the former case, we directly decompose T into T_3 and T_4. In the latter case, we directly decompose T into T_1 and T_2. In each case, the properties on the size of trees and the number of tagged nodes are preserved.

If a generated subtree T is isomorphic to a previously generated subtree T', we assign the same nonterminal label to both edges corresponding to T and T', and do not recur for further decomposition of T.

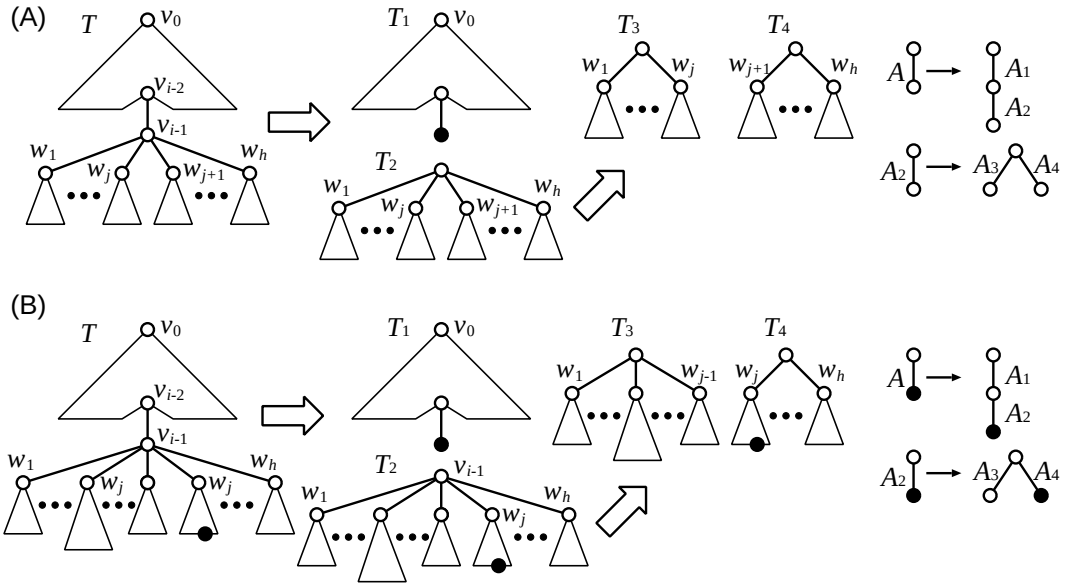

Figure 2: Illustration of TREE-BISECTION. (A) Case of non-tagged tree T. (B) Case of tagged-tree T. In each case, T_i is generated from A_i.

6 Analysis

If we consider trees of height 1 (i.e., the depth of each node is at most 1), EOTG corresponds to CFG and thus the lower bounds on the approximation ratio on compression in [3] holds for EOTG. In the same way, the lower bound for BISECTION (Theorem 5 in [3]) holds also for TREE-BISECTION.

Proposition 1 *The approximation ratio of* TREE-BISECTION *is* $\Omega(\sqrt{n}/\log n)$.

In order to analyze the upper bound of TREE-BISECTION, we first establish mk Lemma [3] for EOTG.

Lemma 2 *If a tree T is generated by an EOTG of size m, $es(T)$ contains at most $2mk$ distinct substrings of length k.*

Proof. We transform EOTG into CFG by splitting each rule (except the starting one) into a pair of rules in CFG by breaking LHS and RHS of each rule before and after x. For example, $Ax\overline{A} \rightarrow B\overline{B}Cx\overline{C}$ is split into $A \rightarrow B\overline{B}C$ and $\overline{A} \rightarrow \overline{C}$. Since we assume that a non-empty unique Euler string is generated by a given EOTG, the same string is generated by this CFG of size $2m$. Then, the proof of mk Lemma in [3] can be directly applied to this case. □

Using the above mentioned relationship between EOTG and CFG, the following proposition directly follows from Lemma 1 of [3].

Proposition 2 *The smallest EOTG that generates a tree of size n has size $\Omega(\log n)$.*

Here, we consider the tree $\mathcal{T}$ representing a recursive process of TREE-BISECTION, where each subtree constructed in TREE-BISECTION is associated with a distinct node in $\mathcal{T}$ in the following way. T_0 corresponds to the root of $\mathcal{T}$. If T is decomposed into T_1 and T_2 in TREE-BISECTION, the node corresponding to T has two children corresponding to T_1 and T_2. For each node p of $\mathcal{T}$, $e(p)$ denotes the number of edges in the corresponding tree.

Lemma 3 $\sum_{p \in V_d(\mathcal{T})} e(p) \leq |E(T_0)| = n - 1$ *holds for any d.*

Proof. TREE-BISECTION recursively decomposes a tree into edge disjoint subtrees. Since the sum is taken over edge disjoint subtrees that are obtained from T_0, the lemma holds. □

Lemma 4 *The depth of recursive calls of* TREE-BISECTION *is $O(\log n)$.*

Proof. Consider any downward path $(p_1, p_2, \ldots, p_5)$ in $\mathcal{T}$. Let T^i denote a tree associated with p_i. Then, we can see that $|T^5| \leq \frac{3}{4}|T^1|$ always holds. Therefore, the length of a path from the root of $\mathcal{T}$ to any leaf is $5(\log_{(4/3)} n + 1)$. □

From this lemma, it is seen that TREE-BISECTION works in polynomial time. Now, we show our main result.

Theorem 2 TREE-BISECTION *computes in polynomial time an SEOTG of size $O(m^* n^{5/6})$ for a given rooted ordered tree T, where m^* is the size of the smallest EOTG for T, and $n = |T|$.*

Proof. As in [3], it is enough to bound the number of non-isomorphic subtrees generated by TREE-BISECTION because a production rule of size at most 4 is generated per subtree.

First, we count the number of subtrees generated by TREE-BISECTION whose sizes are greater than n^α, where α is a constant to be determined later. From Lemma 3, the number of such subtrees generated by recursive calls at depth d is $(n-1)/n^\alpha < n^{1-\alpha}$. Since the maximum depth is $O(\log n)$ from Lemma 4, the number of subtrees whose sizes are greater than n^α is $O(n^{1-\alpha} \log n)$.

Next, we count the number of non-isomorphic subtrees of size at most n^α. Recall that the Euler string of any tagged tree has a form of $s_1 x s_2$, where each s_1 and s_2 is a substring of $es(T_0)$. Therefore, the number of non-isomorphic subtrees of size k is bounded by

$$2m^*(2k-2) \ + \ \sum_{k_1=1}^{(2k-2)-1} (2m^* k_1)(2m^*((2k-2)-k_1)) \leq c_1 (m^*)^2 k^3$$

from Lemma 2 since the length of the Euler string of a subtree of size k is $2(k-1)$, where c_1 is some constant. Then, the number of non-isomorphic subtrees of size at most n^α is $\sum_{k=1}^{n^\alpha} c_1 (m^*)^2 k^3 \leq c_2 \cdot (m^*)^2 \cdot n^{4\alpha}$.

By summing up these two numbers, the total number of non-isomorphic subtrees generated by TREE-BISECTION is $O((m^*)^2 \cdot n^{4\alpha} + n^{1-\alpha} \log n)$. Letting $\alpha = 1/6$ and assuming that m^* is $O(n^{(1/6)})$, we can see that the total number of non-isomorphic subtrees is

$$O(m^* \cdot n^{(1/6)} \cdot n^{(4/6)} + n^{(5/6)} \log n) \ = \ O(m^* \cdot n^{(5/6)} + n^{(5/6)} \log n).$$

Since $m^* \cdot n^{(5/6)} \geq n$ holds for $m^* \geq n^{1/6}$ and m^* is $\Omega(\log n)$, the number of non-isomorphic subtrees generated by TREE-BISECTION is $O(m^* n^{5/6})$. □

TREE-BISECTION can be modified for compression of *rooted unordered trees of bounded degree* (i.e., rooted unordered trees in which the number of children of each node is bounded by a constant H). In this case, the definitions of grammars remain the same except that we do not distinguish the orders of children. For example, we do not distinguish type (IIIA) rules from type (IIIB) rules. Let EUTG (elementary unordered tree grammar) and SEUTG (simple elementary unordered tree grammar) be the resulting grammars corresponding to EOTG and SEOTG, respectively.

For compression of unordered trees, we modify TREE-BISECTION as follows.

- In partition of a non-tagged tree T, we partition T_2 into $T_{2+i} = sub(v, w_i)$ $(i = 1, \ldots, h)$.
- In partition of a tagged tree T, we also partition T_2 into $T_{2+i} = sub(v, w_i)$ $(i = 1, \ldots, h)$.
- We replace the subtree isomorphism test for ordered trees with one for unordered trees.

Let UNORDERED-TREE-BISECTION denote the resulting algorithm. It is to be noted that UNORDERED-TREE-BISECTION does not necessarily output an SEUTG, instead it may output an EUTG because production rules of size $2H$ may be generated. Since T_2 is uniquely determined from T and is uniquely decomposed into T_{2+i}s, the decomposition of an input

tree is independent of the ordering of children. That is, the same unordered grammar is always obtained if isomorphic trees are given.

Theorem 3 UNORDERED-TREE-BISECTION *computes in polynomial time an EUTG of size* $O(m^* n^{5/6})$ *for a given rooted unordered tree* T *of bounded degree, where* m^* *is the size of the smallest EUTG for* T*, and* $n = |T|$*.*

Proof. It is straight-forward to verify that Proposition 2, Lemma 3 and Lemma 4 hold for UNORDERED-TREE-BISECTION. Since it is known that isomorphism of unordered trees can be tested in linear time [4], the algorithm works in polynomial time.

Next, we consider the approximation ratio. Though we do not distinguish the orders of children in grammars or input trees, we can assume that an input tree is generated by a minimum size EOTG and then the orders of children are ignored. Therefore, the number of non-isomorphic unordered subtrees of size k is bounded by $c_1(m^*)^2 k^3$ as in the proof of Theorem 2 and thus the total number of non-isomorphic subtrees produced by the algorithm is $O(m^* \cdot n^{(5/6)} + n^{(5/6)} \log n)$, which is $O(m^* \cdot n^{(5/6)})$ because m^* is $\Omega(\log n)$. Since the size of each generated production rule is bounded by $2H$ and H is assumed to be a constant, the size of the resulting grammar remains $O(m^* \cdot n^{(5/6)})$. □

Acknowledgement

The author would like to thank Morihiro Hayashida and Yang Zhao for helpful discussions.

References

[1] T. Akutsu, A relation between edit distance for ordered trees and edit distance for Euler strings, Information Processing Letters 100 (2006) 105–109.

[2] G. Busatto, M. Lohrey, S. Maneth, Efficient memory representation of XML document trees, Information Systems 33 (2008) 456–474.

[3] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, A. Shelat, The smallest grammar problem, IEEE Transactions on Information Theory 51 (2005) 2554–2576.

[4] Y. Dinitz, A. Itai, M. Rodeh, On an algorithm of Zemlyachenko for subtree isomorphism, Information Processing Letters 70 (1999) 141–146.

[5] J. C. Kieffer, E-H. Yang, Grammar-based codes: A new class of universal lossless source codes, IEEE Transactions on Information Theory 46 (2000) 737–754.

[6] W. Rytter, Application of Lempel-Ziv factorization to the approximation of grammar-based compression, Theoretical Computer Science 302 (2003) 211–222.

[7] S. Murakami, K. Doi, A. Yamamoto, Finding frequent patterns from compressed tree-structured data, Proc. 11th Int. Conf. Discovery Science (2008) 284–295.

[8] H. Sakamoto, S. Maruyama, T. Kida, S. Shimozono, A space-saving approximation algorithm for grammar-based compression, IEICE Transactions on Information and Systems 92-D (2009) 158–165.

[9] K. Yamagata, T. Uchida, T. Shoudai, Y. Nakamura, An effective grammar-based compression algorithm for tree structured data, Proc. 13th Int. Conf. Inductive Logic Programming (2003) 383–400.