

# Property testing of regular tree languages\*

Frédéric Magniez<sup>†</sup>

Michel de Rougemont<sup>‡</sup>

April 22, 2006

## Abstract

We consider the edit distance with moves on the class of words and the class of ordered trees. We first exhibit a simple tester for the class of regular languages on words and generalize it to the class of ranked and unranked regular trees. We also show that this distance problem is NP-complete on ordered trees.

## 1 Introduction

Inspired by the notion of self-testing [BK95, BLR93], property testing has been initially defined and studied for graph properties [GGR98]. It has been successfully extended for various classes of finite structures. Let  $\mathbf{K}$  be a class of finite structures and a distance function  $\text{dist}$ , i.e. a function between structures of  $\mathbf{K}$ . Two inputs are  $\varepsilon$ -far if their normalized distance is greater than  $\varepsilon$ . An  $\varepsilon$ -tester for a class  $\mathbf{K}_0 \subseteq \mathbf{K}$  is a randomized algorithm which takes a structure  $U_n$  of size  $n$  as input and decides if  $U_n \in \mathbf{K}_0$  or if  $U_n$  is  $\varepsilon$ -far from  $\mathbf{K}_0$  with high probability. A class  $\mathbf{K}_0 \subseteq \mathbf{K}$  is testable, if there is a randomized algorithm which is an  $\varepsilon$ -tester for  $\mathbf{K}_0$  for every sufficiently small  $\varepsilon$  given as input, and whose query and time complexities only depend on  $\varepsilon$ , i.e. independent of  $n$ .

For the Hamming distance, regular languages and  $\Sigma_2$ -definable graph properties are testable [AKNS00, AFKS00]. Testers have also been generalized to the infinite regular languages [CK02].

In this paper we initiate the study of property testing for words and trees under the edit distance with moves, where the elementary operations are not only the insertions and deletions of letters or nodes, but also the *moves* of any entire subword or subtree in one step.

First (Section 3), we develop a tester for regular languages on words that greatly simplifies the tester of [AKNS00] and improves its complexity by a  $\log(1/\varepsilon)$  factor, when we consider the edit distance with moves instead of the standard edit distance.

Then (Section 4), we initiate the study of property testing on trees. The testability of regular tree languages is an open problem [CK02] for the standard edit distance. We solve this problem when moves are allowed, by proving the testability of regular ranked tree languages.

Finally (Section 5), we generalize the testability to unranked trees. As a direct application, it implies that one can decide in constant time if a large XML document follows a DTD or is far from it.

The word edit distance with moves decision problem and the standard tree edit distance decision problem are computable in polynomial time [Cor03, Tai79]. We prove in (Section 6) that the tree edit distance with moves is NP-complete. It is then interesting to point out that this apparently more complex distance yields a tester for regular languages, whereas we do not know such a tester for the classical tree edit distance.

---

\*A preliminary version of this paper appeared in *Proceedings of 31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 932-944, Verlag, 2005.

Work supported by *ACI Sécurité Informatique: VERA* of the French Ministry of research.

<sup>†</sup>CNRS-LRI, UMR 8623 Université Paris-Sud, France

<sup>‡</sup>LRI & Université Paris II, France

## 2 Preliminaries

### 2.1 Property Testing

Recall the notion of a property tester [GGR98] on a class  $\mathbf{K}$  of finite structures for which a distance function between structures has been defined.

Let  $\varepsilon > 0$  be a real. We say that two structures  $U, U' \in \mathbf{K}$  are  $\varepsilon$ -close if their distance is at most  $\varepsilon \times M$ , where  $M$  is a normalization factor, that is the maximum of  $\text{dist}(V, V')$  when  $V$  and  $V'$  range over  $\mathbf{K}$  and have respectively same sizes than  $U$  and  $U'$ . They are  $\varepsilon$ -far if they are not  $\varepsilon$ -close. For words and trees,  $M$  is set to the maximal size of the respective structures, since this is always the order of  $M$ . (For dense graphs,  $M$  is the square of the maximal size of the respective structures.)

We now define the notion of  $\varepsilon$ -tester for a subclass of  $\mathbf{K}$ . Since our algorithms will be 1-sided error, we only defined testers in this way. Nonetheless, 2-sided error testers are also well investigated in various contexts.

**Definition 1.** Let  $\varepsilon \geq 0$  be a real. An  $\varepsilon$ -tester for a class  $\mathbf{K}_0 \subseteq \mathbf{K}$  is a randomized algorithm  $A$  such that:

- (1) If  $U \in \mathbf{K}_0$ ,  $A$  always accepts;
- (2) If  $U$  is  $\varepsilon$ -far from  $\mathbf{K}_0$ , then  $\Pr[A \text{ rejects}] \geq 2/3$ .

When the tester accesses the input structure  $U$  by queries (that are defined below depending on the class  $\mathbf{K}$ ), the *query complexity* is the number of queries to  $U$ . The *time complexity* is the usual definition, where we assume that the following operations are performed in constant time: arithmetic operations, a uniform random choice of an integer from any finite range not larger than the input size, and a query to the input.

A class  $\mathbf{K}_0 \subseteq \mathbf{K}$  is *testable*, if there is a randomized algorithm  $A$  such that for every sufficiently small  $\varepsilon > 0$  as input,  $A(\varepsilon)$  is an  $\varepsilon$ -tester of  $\mathbf{K}_0$  whose query and time complexities only depend on  $\varepsilon$ .

### 2.2 Words

Let  $\Sigma$  be a finite alphabet. We consider the words on the alphabet  $\Sigma$ . Every word  $W$  of size  $|W| = n$  is a finite structure  $(n, [n], l : [n] \rightarrow \Sigma)$ , where  $[n]$  denote the set  $\{1, \dots, n\}$ . The class  $\mathbf{K}$  is the set of all such structures. We will denote a subclass  $\mathbf{K}_0$  of  $\mathbf{K}$  as a subset  $L \subseteq \Sigma^*$ . In this context, a query  $i$  to some word  $W$  asks the letter  $W[i] = l(i)$ .

Let  $W$  be a word. Let  $W[i, j]$  be the word  $W[i] \dots W[j]$ . A word  $w$  is a *subword* of  $W$  if  $w = W[i, j]$ , for some  $i, j$ .

An *elementary operation (on words)* is a substitution, a deletion or an insertion of a letter, or a move: given  $(i, j, k)$  such that  $k \notin [i, j]$ , a *move* is a transformation of the word, where the subword  $W[i, \dots, j]$  has been removed from its current position and inserted in the position  $k$ . For instance if  $k > j$ , the resulting word is the concatenation  $W[1, i - 1]W[j + 1, k - 1]W[i, j]W[k, n]$ .

The *edit distance with moves* between two words  $W$  and  $W'$  is the minimum number of elementary operations necessary to reach  $W'$  from  $W$ , noted  $\text{dist}(W, W')$ . The distance between  $W$  and a language  $L$ , noted  $\text{dist}(W, L)$ , is the minimum  $\text{dist}(W, W')$  when  $W' \in L$ .

The standard edit distance only considers the operations without moves, and this new distance is essential for most of the arguments. From now, all our discussions and results will be for the edit distance with moves.

### 2.3 Trees

Fix a finite alphabet  $\Sigma$ . We consider ordered  $\Sigma$ -tree, *i.e.* trees with labels  $\sigma \in \Sigma$  on the nodes. A tree is *ranked* if the degree is bounded by a fixed constant, and *unranked* otherwise. We omit the term 'ordered', since all our trees will be ordered.

A *subtree*  $t$  of  $T$  is a tree induced by a subset of nodes  $V = \{v_1, \dots, v_m\}$  of  $T$  such that for every pair of nodes  $v_i, v_j$  the path between  $v_i$  and  $v_j$  in  $T$  is included in  $V$ . The leaves of  $T$  among  $\{v_1, \dots, v_m\}$  are leaves of  $t$ , while some nodes are leaves in  $t$  but not in  $T$  and called *\*-nodes* where their new label is  $*$  (that we suppose to be outside  $\Sigma$ ). By extension, a *subtree*  $t$  is a tree where some of the leaves are *\*-nodes*.

### 2.3.1 Ranked Trees

Let us first consider  $r$ -ranked trees for some fixed constant  $r$ . An  $r$ -ranked tree  $T$  of size  $|T| = n$  is a finite structure

$$(n, [n], \text{root}, l : [n] \rightarrow \Sigma, d : [n] \rightarrow [r], s : [n] \times [r] \rightarrow [n]),$$

where  $\text{root}$  is the distinguished element representing the root of  $T$ ,  $l$  is the label function,  $d$  is the degree function which gives the degree of any node, and  $s$  is the successor partial function which associates to every node  $v$  and any position  $i \in [d(v)]$  the  $i$ -th successor of  $v$ , that we will also name the successor of  $v$  at position  $i$ .

The class  $\mathbf{K}$  is the set of all such structures. We will denote a subclass  $\mathbf{K}_0$  of  $\mathbf{K}$  as a subset  $L$  of all  $r$ -ranked trees. In this context, a query  $(v, i)$  to some tree  $T$  asks the label and the degree  $d(v)$  of the node  $v$  and its  $i$ -th node successor in  $T$ , if  $i \leq d(v)$ .

Note that the structure of the tree itself is completely unknown. Only the size  $n$  and the rank  $r$  are given as an input. The access to the tree is only made by queries.

### 2.3.2 Unranked Trees

For unranked trees, there is no degree condition. An unranked tree is a structure:

$$(n, [n], \text{root}, l : [n] \rightarrow \Sigma, \text{suc} : [n] \rightarrow [n], \text{sib} : [n] \rightarrow [n]),$$

where  $n = |T|$  is the size of  $T$ ,  $\text{root}$  is the distinguished element representing the root of  $T$ ,  $l$  is the label function,  $\text{suc}$  is the successor partial function which associates with every node  $v$  the first successor of  $v$ , and  $\text{sib}$  is the sibling partial function which associates with every node  $v$  its next sibling, i.e. a node  $w$  such that if  $v$  is the  $i$ -th successor of  $u$ ,  $w$  is the  $(i + 1)$ -th successor of  $u$  if it exists.

The class  $\mathbf{K}$  is the set of all such structures. We will denote a subclass  $\mathbf{K}_0$  of  $\mathbf{K}$  as a subset  $L$  of all unranked trees. In this context, a query  $v$  to some tree  $T$  asks for the label of the node  $v$ , its successor and its sibling node in  $T$ . Notice that in the case of XML files, if we assume that  $T$  is given by its DOM (Document Object Model) structures, the model enables us to simulate our queries. On the other hand, the SAX model views the tree as a stream, and it is not as simple to access a random subtree.

Again, the structure of the tree is unknown. The access to the tree is made by queries given as input its size  $n$ .

### 2.3.3 Distance

The classical tree edit distance [Tai79] assumes basic insertions, deletions on a tree  $T$  and substitutions of labels (see Figure 1). A *substitution*  $(v, \sigma)$ , where  $\sigma \in \Sigma$ , changes the label of a node  $v$  into the label  $\sigma \in \Sigma$ . An *insertion*  $(u, \sigma, v, i, j)$  inserts a new node  $u$  in  $T$  with label  $\sigma$  which is the successor of  $v$  such that the successors of  $v$  between positions  $i$  and  $j$  are now the successors of  $u$ . A *deletion*  $u$  is the inverse of a node insertion. It suppresses the node  $u$  and sets the predecessor of all successors of  $u$  to the predecessor of  $u$ . If the position of  $u$  was  $i$ , then the positions of its successors are now  $i, i + 1, \dots$ . Note that a deletion on an  $r$ -ranked tree might gives a tree whose rank is greater than  $r$ . Therefore, we implicitly restrict ourselves only to deletions that gives a  $r$ -ranked tree when we are working with  $r$ -ranked trees.

We will also allow some moves in  $T$  (see Figure 1). A *move*  $(u, v, i)$  moves in one step  $u$  (and the corresponding subtree rooted at  $u$ ) as the  $i$ -th successor of  $v$ , shifting all the  $j$ -th successors of  $v$  for  $j \geq i$  by one. As a consequence, the new ancestor of  $u$  is now  $v$ . Moreover if  $u$  was the  $i'$ -th successor of  $v'$ , then all the successors of  $v'$  for  $j > i'$  are also shifted by one. Note that all moves are not necessarily valid. Let  $d$  be the number of successors of  $v$ . We require that  $i \leq d + 1$ , and in the case of  $r$ -ranked trees that  $d < r$ .

An *elementary operation (on trees)* is one of the above operations. The *tree edit distance with moves* between two trees  $T$  and  $T'$  is the minimum number of elementary operations necessary to reach  $T'$  from  $T$ , noted  $\text{dist}(T, T')$ . Similarly than words, the distance is extended to tree languages.

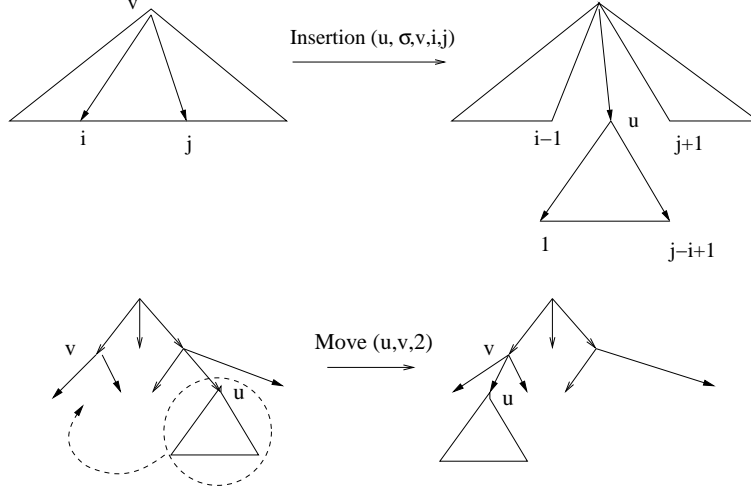


Figure 1: Elementary operations on trees.

### 2.3.4 Tree Automata

An (nondeterministic)  $r$ -ranked tree automaton is a 5-tuple  $\mathcal{A} = (Q, \Sigma, \delta, (I_\sigma)_{\sigma \in \Sigma}, F)$  where  $Q$  is the set of states,  $F \subseteq Q$  is the set of accepting states,  $I_\sigma \subseteq Q$  the set of initial states for  $\sigma$ , and  $\delta \subseteq (Q^{\leq r} \times \Sigma) \times Q$  is the transition relation.

An assignment  $\lambda$  for a  $r$ -ranked tree  $T$  determines states for its leaves such that if  $u$  is a leaf with label  $l(u)$ , then  $\lambda(u) \in I_{l(u)}$ . A run on a tree  $T$  extends  $\lambda$ , defined on the leaves, to all internal nodes: If  $u$  is a node with successors  $v_1, \dots, v_l$  where  $l \leq r$  in states  $\lambda(v_1), \dots, \lambda(v_l)$  then  $\lambda(u)$  satisfies  $\delta(\lambda(v_1), \dots, \lambda(v_l), l(u); \lambda(u)) \in \delta$ . A run *accepts* if the state of the root is in  $F$ . We assume that the transition relation is *complete*, meaning that the above extension can always continue until the root of  $T$ . In a formal way, for each sequence  $q_1, \dots, q_l, \sigma$ , there exists  $q \in Q$  such that  $(q_1, \dots, q_l, \sigma; q) \in \delta$ .

Then the tree language that *recognizes*  $\mathcal{A}$  is the set of trees that have an accepting run.

For unranked trees, the notion of tree automata is slightly different. Recall that unranked trees are trees where the degree of each node is unbounded. An *unranked tree automaton* generalizes the transition relation where the transition condition is encoded by a regular expression. Namely  $\delta$  is a finite subset of  $(Q \times \Sigma) \times \mathcal{R}(Q)$ , where  $\mathcal{R}(Q)$  is the set of regular expressions on  $Q$ . The notions of assignment and run can also be extended in the following way. A run extends an assignment on the leaves of a tree to all its nodes such that, if  $u$  is a node with successors  $v_1, \dots, v_l$  in states  $\lambda(v_1), \dots, \lambda(v_l)$ , then the word  $\lambda(v_1) \cdots \lambda(v_l)$  is recognized by some regular expression  $r$  such that  $\lambda(u)$  and  $r$  satisfy  $(\lambda(u), l(u); r) \in \delta$ . Then unranked tree regular languages are defined in a similar way.

## 3 Testing regular languages

### 3.1 Basic definitions

Let  $\mathcal{A}$  be a (nondeterministic) automaton on words with  $m$  states,  $m \geq 2$ , which recognizes a language  $L$ . We say that  $w$  *connects* the states  $q_1$  to the state  $q_2$  when starting from  $q_1$ , the automaton  $\mathcal{A}$  can reach  $q_2$  after reading word  $w$ . If  $w$  connects  $q_1$  to  $q_2$ , we also say that  $q_1$  *is connected* to  $q_2$ . This notion will be used for random subwords  $w$  of a fixed word  $W$ .

**Proposition 1.** *Let  $q_1$  be a state connected to  $q_2$ . Then there exists a word  $w$  of size less than  $m$  that connects  $q_1$  to  $q_2$ .*

Let  $G(\mathcal{A})$  be the transitive closure of the graph of the automaton:  $G(\mathcal{A})$  is a directed graph whose vertices are the states of  $\mathcal{A}$ , and whose edges connect states that are connected by any word (of size less than  $m$ ). We assume without loss of generality that  $G(\mathcal{A})$  is connected but not necessarily strongly connected. A strongly connected component of  $G(\mathcal{A})$  will be simply called a *connected component* of  $G(\mathcal{A})$ . A connected component of  $G(\mathcal{A})$  is *truly connected* if it is not a singleton. By definition, any truly connected component has a non empty path in  $G(\mathcal{A})$ . We will denote by  $\widehat{G}(\mathcal{A})$  the graph of the connected components of  $G(\mathcal{A})$ .

**Definition 2.** Let  $\Pi = (C_1, \dots, C_k)$  be a path of  $\widehat{G}(\mathcal{A})$ . Then  $\Pi$  is admissible if  $C_1$  contains an initial state, and  $C_k$  contains a final state.

**Definition 3.**

1. Let  $C$  be a truly connected component of  $G(\mathcal{A})$ . A word  $w$  is  $C$ -feasible if it connects two states of  $C$ .
2. Let  $\Pi$  be a path of  $\widehat{G}(\mathcal{A})$ . A word  $w$  is  $\Pi$ -feasible if it connects two states along a path visiting connected components along a subsequence of  $\Pi$ .

A word  $w$  is  $C$ -infeasible (respectively  $\Pi$ -infeasible) if it is not  $C$ -feasible (respectively  $\Pi$ -feasible).

A *cut* of a word  $W$  is an ordered partition of  $W$  in subwords, whose order is consistent with the one of  $W$ . Below we omit the term ‘ordered’. A cut is  $\Pi$ -feasible if the subwords defined by the cut are  $\Pi$ -feasible. Since a cut defines a word by the juxtaposition of the corresponding subwords according to its order, we naturally extend the distance on words to cuts.

### 3.2 The tester

The tester takes random subwords of finite length of  $W$  and tests their feasibility for every admissible paths  $\Pi$ , that is at most  $2^m$  where  $m$  is the number of states of the automaton. The Robustness lemma will insure that if a word  $W$  is far from  $L$ , then with high probability a random subword of finite length will be infeasible.

**Regular language tester**  $(\mathcal{A}, \varepsilon, W)$ :

If the size  $n$  of  $W$  is less than  $80m^2 \log(5m^2/\varepsilon)/\varepsilon$

  Read  $W$  and accept iff  $\mathcal{A}$  accepts  $W$

Else

  For  $i = 1, \dots, \log(5m^2/\varepsilon)$  {

    Compute  $N_i = \lceil (m+1) \times 20m^2 \log(m^2/\varepsilon)/(\varepsilon 2^i) \rceil$

    Choose  $N_i$  random subwords  $w_j^i$  of  $W$  of size  $2^{i+1}$ , for  $j = 1, \dots, N_i$  }

  For every admissible path  $\Pi$  of  $\widehat{G}(\mathcal{A})$  {

    If all the  $w_j^i$  are  $\Pi$ -feasible then accept  $W$  (and stop) }

  Reject  $W$ .

**Theorem 1.** For every real  $\varepsilon > 0$ , every automaton  $\mathcal{A}$  with  $m$  states, the algorithm **Regular language tester**  $(\mathcal{A}, \varepsilon, \cdot)$  is an  $\varepsilon$ -tester for the language recognized by  $\mathcal{A}$ . Moreover, its query complexity is in  $O(m^3 \log^2(m^2/\varepsilon)/\varepsilon)$ , and its time complexity in  $O(2^m m^4 \log^2(m^2/\varepsilon)/\varepsilon)$ .

**Corollary 1.** Regular properties of words are  $\varepsilon$ -testable.

Before proving Theorem 1, we need the following Robustness lemma which will be crucial for our proof. The notion of robustness was first defined in [RS96] and studied in [Rub99]. In the rest of this section, we fix an automaton  $\mathcal{A}$  and we call  $L$  its associated language.

**Lemma 1** (Robustness). *Let  $n \geq 80m^2 \log(5m^2/\varepsilon)/\varepsilon$ , and let  $W$  be a word of size  $n$  such that  $\text{dist}(W, L) \geq \varepsilon n$ . Then for every admissible path  $\Pi$  of  $\widehat{G}(\mathcal{A})$ , there exists an integer  $1 \leq i \leq \log(5m^2/\varepsilon)$ , such that the number of  $\Pi$ -infeasible subwords of size  $2^{i+1}$  is at least  $\frac{2^i}{20m^2 \log(5m^2/\varepsilon)} \times \varepsilon n$ .*

We delay the full proof of the Robustness lemma to the end of Section 3.3, and for now we only sketch its main steps:

1. The Splitting lemma shows that if the distance between  $W$  and  $L$  is large then there are many infeasible disjoint subwords. Its proof is by contraposition (see Figure 2):
  - (a) First, from a cut of minimal infeasible subwords, we construct a close feasible cut.
  - (b) Then the Merging lemma shows that if a cut is feasible, then it is close to  $L$ .

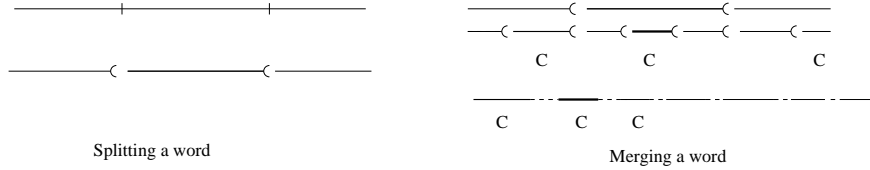


Figure 2: The correction (steps 1.a and 1.b) of a word with two infeasible subwords where  $C$  is some connected components (and  $h' = 3$  for the proof of Lemma 3).

2. The Amplifying lemma shows that if there are many infeasible words, then there are many short ones.

*Proof of Theorem 1.* We can assume without loss of generality that the size  $n$  of  $W$  is at least  $15m^2/\varepsilon$ , otherwise the proof of the correctness is obvious.

First, if  $W \in L$  then  $W$  is  $\Pi$ -feasible for some admissible  $\Pi$ . Therefore every subword of  $W$  is  $\Pi$ -feasible for this path  $\Pi$ . Thus the tester accepts  $W$  with probability 1.

Now suppose that  $\text{dist}(W, L) > \varepsilon n$ . Let us fix an admissible path  $\Pi$ . Using the Robustness lemma (Lemma 1), we get that the probability to accept  $W$  for this  $\Pi$  is at most  $2^{-m}/3$ . Now, since there are at most  $2^m$  candidates  $\Pi$ , we can conclude using the union bound, that the overall acceptance probability is upper bounded by  $1/3$ .

Last, for each integer  $i = 1, \dots, \log(5m^2/\varepsilon)$ , the tester queries  $N_i$  subwords of size  $2^{i+1}$ , that is  $O(m^3 \log(m^2\varepsilon)/\varepsilon)$  letters. Therefore the query complexity of the tester is clearly in  $O(m^3 \log^2(m^2/\varepsilon)/\varepsilon)$ . For the time complexity, we need to evaluate the complexity of deciding if a subword  $w$  is  $\Pi$ -feasible, for a fixed  $\Pi$ . This can be done in time  $m^2 \times |w|$ , using a  $m$ -dimensional vector of reachable states that we update while reading step by step the letters of  $w$ . Thus, the overall time complexity of the tester is in  $O(2^m m^4 \log^2(m^2/\varepsilon)/\varepsilon)$ .  $\square$

### 3.3 Robustness of the tester

**Lemma 2** (Merging). *Let  $\Pi = (C_1, \dots, C_k)$  be an admissible path of  $\widehat{G}(\mathcal{A})$ . Let  $F$  be a  $\Pi$ -feasible cut of size  $h'$ . Then  $\text{dist}(F, L) \leq m - 1 + (m^2 + 2m - 1) \times h'$ .*

*Proof.* First, we split each subword of  $F$  in  $C$ -feasible subwords, for some  $C \in \Pi$ . Given a  $\Pi$ -feasible subword  $w$  which connects  $p \in C_i$  to  $q \in C_j$ , we follow the automaton from  $p$  to  $q$  on  $w$ , and we delete each letter leading to a new connected component. Then the subword is cut along each deleted letter.

This technique only keeps subwords that are  $C$ -feasible for some truly connected component  $C$ . Moreover, each initial subword of  $F$  splits in at most  $m$  subwords with at most  $(m - 1)$  deletions. Let  $(w_i)_{1 \leq i \leq l}$  be the remaining subwords of  $F$ , where  $1 \leq l \leq m \times h'$ .

Now we explain how to move and glue the remaining subwords  $w_i$  in order to get a subword  $W' \in L$ . Let  $C'_i$  be a component of  $\Pi$  such that  $w_i$  is  $C'_i$ -feasible. Let  $p_i, q_i \in C'_i$  such that  $w_i$  connects  $p_i$  to  $q_i$ . Then, we

do  $(l-1)$  moves so that the components  $C'_i$  are in the order defined by  $\Pi$ . Up to some renaming, we assume now that  $(C'_1, \dots, C'_l)$  are in the same order than  $(C_1, \dots, C_k)$  with multiplicity.

We glue by induction. Let  $q_0$  be an initial state of  $C_1$ , and let  $p_{l+1}$  be an accepting state of  $C_k$ . For  $i = 0$  to  $i = l$  do the following. By Proposition 1, let  $g_i$  be a word of size at most  $m$  that connects  $q_i$  to  $p_{i+1}$ . By inserting  $g_i$  between  $w_i$  and  $w_{i+1}$ , we get the word  $W' = g_0 w_1 g_1 \dots w_l g_l$  such that  $W' \in L$ . In this last step, we did at most  $m \times (l+1)$  insertions.

The total number of elementary operations is less than  $(m-1) \times h' + (l-1) + m \times (l+1) \leq m-1 + (m^2 + 2m-1) \times h'$ , since  $l \leq m \times h'$  and  $m \geq 2$ .  $\square$

**Lemma 3** (Splitting). *Let  $\Pi$  be an admissible path of  $\widehat{G}(\mathcal{A})$ . Let  $W$  be a word such that  $\text{dist}(W, L) > h$ . Then there exists a cut of  $W$  that has more than  $\frac{h-3m^2}{2m^2}$   $\Pi$ -infeasible disjoint subwords.*

*Proof.* The proof is by contraposition. By assumption, any cut of  $W$  has at most  $\frac{h-m}{2m^2}$  infeasible subwords. We will show how to construct a cut of size  $h'$  with at least  $(h'-1)$  infeasible subwords such that  $\text{dist}(W, L) \leq m + 2m^2 \times h'$ , which together with our assumption gives  $\text{dist}(W, L) \leq h$ .

First we construct a cut  $\mathcal{P}$  of  $W$  of size  $h'$  whose  $(h'-1)$  first subwords are minimal  $\Pi$ -infeasible and disjoint subwords. The last subword of  $\mathcal{P}$  might be  $\Pi$ -feasible. In this case, the entire word  $W$  might also be  $\Pi$ -feasible and  $h' = 1$ .

We visit  $W$  from the left to the right and the construction of each  $\Pi$ -infeasible subword  $W[i, j]$  is done by induction on that walk.

Initially:  $h' = 0$ ,  $i = j = 1$ , and  $n = \text{size of } W$ .

While  $(j \leq n)$  {  
  While (subword  $W[i, j]$  is  $\Pi$ -feasible and  $j < n$ ) {increase  $j$ }  
   $h' = h' + 1$ ,  $w_{h'} = W[i, j]$ ,  
   $i = j + 1$ ,  $j = i$ .  
}

At the end of the procedure we get the desired partition  $\mathcal{P} = (w_i)_{1 \leq i \leq h'}$ .

Now we explain how to get a word  $W' \in L$ . Let  $w'_i$  be  $w_i$  without the last letter, for  $i = 1, \dots, h'$ . When  $w_{h'}$  is  $\Pi$ -feasible then  $w'_{h'} = w_{h'}$ . By construction of  $w_i$ , the subwords  $w'_i$  are  $\Pi$ -feasible. Let  $F$  be the cut of the  $(w'_i)_{1 \leq i \leq h'}$ . Applying Lemma 2, we get that  $\text{dist}(F, L) \leq m-1 + (m^2 + 2m-1) \times h'$ . Because  $\text{dist}(W, F) \leq h'$ , then  $\text{dist}(W, L) \leq m-1 + (m^2 + 2m) \times h' \leq m + 2m^2 \times h'$ , since  $m \geq 2$ .  $\square$

**Lemma 4** (Amplifying). *Let  $\Pi$  be a path of  $\widehat{G}(\mathcal{A})$ . Let  $W$  be a word of length  $n$  with at least  $h'$   $\Pi$ -infeasible disjoint subwords. Then there exists an integer  $1 \leq i \leq \log(2n/h')$  such that the number of  $\Pi$ -infeasible subwords of size  $2^{i+1}$  is at least  $\frac{2^i}{3} (\frac{h'}{2 \log(2n/h')} - 4)$ .*

*Proof.* In this proof, we understand feasible as  $\Pi$ -feasible.

Let  $w_1, \dots, w_{h'}$  be some infeasible disjoint subwords of  $W$ . For every integer  $i \geq 1$ , let  $s_i = |\{w_j : 2^{i-1} + 1 \leq |w_j| \leq 2^i\}|$ . Then  $\sum_{i \geq 1} s_i = h'$ . Let  $a$  be a positive integer. Then since  $|W| = n$ , we get that  $|\{w_j : |w_j| > 2^a\}| \leq \frac{n}{2^a}$ . Therefore  $\sum_{i=1}^a s_i \geq h' - \frac{n}{2^a}$ .

Take  $a = \log(2n/h')$ . Then  $\sum_{i=1}^a s_i \geq \frac{h'}{2}$ , thus there exists some  $1 \leq i \leq a$  such that  $s_i \geq \frac{h'}{2a}$ .

To lower bound the number of infeasible subwords of size  $2^{i+1}$ , we count the number of subwords of size  $2^{i+1}$  that contains a least one subword  $w_j$  whose size is in  $[2^{i-1} + 1, 2^i]$ . These subwords are also infeasible since they contain one of the infeasible subwords  $w_j$ . Note that since the subwords  $w_j$  are disjoint, each infeasible subword of length  $2^{i+1}$  contains at most 3 of the  $w_j$  of length greater than  $2^{i-1}$ . Moreover, each infeasible subword  $w_j$  of length at most  $2^i$  is included in at least  $2^i$  subwords of length  $2^{i+1}$  (except, maybe, the two first and the two last subwords). We then get that the number of infeasible subwords of size  $2^{i+1}$  is at least  $\frac{2^i}{3} \times (\frac{h'}{2a} - 4)$ .  $\square$

*Proof of Lemma 1.* From the Splitting lemma with  $h = \varepsilon n$ , the word  $W$  has more than  $h' = \frac{2\varepsilon n}{5m^2}$   $\Pi$ -infeasible disjoint subwords. Now, by the Amplifying lemma, there exists an integer  $1 \leq i \leq \log(5m^2/\varepsilon)$  such that

the number of  $\Pi$ -infeasible subwords of size  $2^{i+1}$  is at least  $\frac{2^i((2\varepsilon n/5m^2)-8\log(5m^2/\varepsilon))}{6\log(5m^2/\varepsilon)}$ . Using the hypothesis  $n \geq \frac{80m^2 \log(5m^2/\varepsilon)}{\varepsilon}$ , we get that  $(2\varepsilon n/5m^2) - 8\log(5m^2/\varepsilon) \geq (1.5\varepsilon n/5m^2)$ , and then we can lower bound the above by  $\frac{2^i \varepsilon n}{20\log(5m^2/\varepsilon)}$ .  $\square$

## 4 Testing regular ranked tree languages

### 4.1 Basic definitions

First, let us generalize the notion of assignment to subtrees. An *assignment*  $\lambda$  for a subtree  $t$  determines states for its leaves such that if  $u$  is a leaf with label  $l(u) \neq *$ , then  $\lambda(u) \in I_{l(u)}$ , otherwise  $\lambda(u)$  can be any state of  $Q$ . A run is defined for a subtree exactly in the same way as for trees. A *run* on a subtree  $t$  extends  $\lambda$ , defined on the leaves, to all internal nodes: If  $u$  is a node with successors  $v_1, \dots, v_l$  where  $l \leq r$  in states  $\lambda(v_1), \dots, \lambda(v_l)$  then  $\lambda(u)$  satisfies  $(\lambda(v_1), \dots, \lambda(v_l), l(u); \lambda(u)) \in \delta$ .

Two states  $q$  and  $q'$  are *connected* if there exists a finite tree  $t$  with exactly one  $*$ -leaf, and a run  $\lambda$  such that the  $*$ -leaf of  $t$  is assigned the state  $q$ , and the root of  $t$  is assigned the state  $q'$ . As in the case of words,  $t$  can be always chosen of size at most  $r^m$ .

Let  $G(\mathcal{A})$  be the transitive closure of the graph of the tree automaton:  $G(\mathcal{A})$  is a directed graph whose vertices are the states of  $\mathcal{A}$ , and whose edges connect states that are connected by any subtree (of size at most  $r^m$ ).

We assume without loss of generality that  $G(\mathcal{A})$  is connected. We define  $\widehat{G}(\mathcal{A})$  and the notion of *truly connected* as in Section 3.1, and we omit the term ‘strongly’. Let  $\mathcal{C}(\mathcal{A})$  be the set of connected components of  $G(\mathcal{A})$ . We generalize the notions of  $\Pi$ -feasibility for subtrees where  $\Pi$  is a subset of  $\mathcal{C}(\mathcal{A})$ .

**Definition 4.** Let  $\Pi \subseteq \mathcal{C}(\mathcal{A})$ . Then  $\Pi$  is *admissible* if there is a pair  $(T_0, \lambda_0)$ , the witness of  $\Pi$ , such that  $\lambda_0$  is an assignment of the tree  $T_0$  which visits every connected components  $\Pi$ , and no more.

Observe that  $T_0$  can be always chosen such that its size is at most  $r^m$ .

**Definition 5.** Let  $\Pi \subseteq \mathcal{C}(\mathcal{A})$ .

1. A path  $\sigma$  from a leaf to the root of a subtree  $t$  is  $\Pi$ -feasible if there exists a run which visits along  $\sigma$  connected components along a subset of  $\Pi$ .
2. A subtree  $t$  is *simply*  $\Pi$ -feasible if there exists a  $\Pi$ -feasible path  $\sigma$  in  $T$ .
3. A subtree  $t$  is  $\Pi$ -feasible if there exists a run  $\lambda$  such that every path  $\sigma$  of  $t$  is  $\Pi$ -feasible for the same run  $\lambda$ .

A subtree  $t$  is (*simply*)  $\Pi$ -*infeasible* if it is not (*simply*)  $\Pi$ -feasible.

We now define two related notions of partitions of a tree into subtrees, namely *cut* and *forest*. Intuitively, the cut will be constructed such that its subtrees are infeasible. These subtrees will be in fact minimal infeasible, leading to a forest of maximal feasible subtrees.

We say that two subtrees of a tree  $T$  are *disjoint* if they are node-disjoint except in one node that might be both a  $*$ -leaf of one subtree and the root of the other subtree. When we want to precise that two subtrees have no node in common, we explicitly say that they are *node-disjoint*.

A *cut* of a tree  $T$  is a subset of nodes of  $T$ , where  $T$  will be cut. A cut of  $T$  defines inductively a partially ordered partition of  $T$  into disjoint subtrees, whose order is defined by  $T$ :

We visit  $T$  bottom-up. While visiting a node  $v$ , if  $v$  is in the cut, then the subtree of  $T$  rooted in  $v$  becomes a new subtree of the partition that we suppress from  $T$ , keeping  $v$  as a  $*$ -node in the remaining part of  $T$ .

Depending on the context, we will consider a cut of  $T$  either as a subset of nodes, or as such a partially ordered partition of subtrees.

Those subtrees of the cut will be defined as minimal infeasible subtrees of  $T$ . By removing the root of those subtrees, they will become maximal feasible subtrees. This operation on the cut naturally defines another partition of  $T$  which is node-disjoint, and that we call the *forest* of the cut. Each subtree  $t$  of the cut whose root is also a  $*$ -node in another subtree of the cut, is replaced by the subtrees of its root (at most  $r$ ). A forest is therefore a partially ordered partition of  $T$  into node-disjoint subtrees whose order is defined by  $T$ .

By definition, we only consider forests that correspond to some implicit tree  $T$ . Therefore we naturally extend our distance on trees to such forests. An elementary operation on a forest modifies implicitly the corresponding tree.

The size of a forest of subtrees is the number of its subtrees, and a forest is  $\Pi$ -feasible if all its subtrees are  $\Pi$ -feasible.

## 4.2 The tester

A  $k$ -subtree of  $T$  from  $v$  is a subtree of  $T$  with  $v$  as a root and containing every nodes at depth at most  $k$  below  $v$ . The tester generates random  $k$ -subtrees from random nodes  $v$ . More precisely, the tester is going to select  $\Theta(\frac{mr}{\varepsilon^2})$  random  $i$ -subtrees, for  $i = 1, \dots, r^{\Theta(m)}/\varepsilon$ , and check if they are all  $\Pi$ -feasible, for some admissible  $\Pi$ .

### Regular ranked tree language tester $(\mathcal{A}, \varepsilon, T)$ :

If the size  $n$  of  $T$  is less than  $10r^{2m+1}/\varepsilon$   
 Read  $T$  and accept iff  $\mathcal{A}$  accepts  $T$   
 Else  
 Compute  $N = \lceil (m+1) \times 64r^{4m+3}/\varepsilon^2 \rceil$   
 For  $i = 1, \dots, 16r^{2m+1}/\varepsilon$  {  
   Choose  $N$  random nodes  $v_j^i$ , for  $j = 1, \dots, N$ .  
   Query the  $i$ -subtree  $t_j^i$  of  $T$  from  $v_j^i$ , for  $j = 1, \dots, N$  }  
 For every admissible subset  $\Pi \subseteq \mathcal{C}(\mathcal{A})$ . {  
   If all the  $t_j^i$  are  $\Pi$ -feasible then accept  $T$  (and stop) }  
 Reject  $T$ .

**Theorem 2.** *For every real  $\varepsilon > 0$ , every  $r$ -ranked tree automaton  $\mathcal{A}$  with  $m$  states, and every  $r$ -ranked tree  $T$ , the algorithm **Regular ranked tree language tester**  $(\mathcal{A}, \varepsilon, T)$  is an  $\varepsilon$ -tester for the language recognized by  $\mathcal{A}$ . Moreover, its query and time complexities are in  $r^{O(r^{2m+1}/\varepsilon)}$ .*

**Corollary 2.** *Regular properties of ranked trees are testable.*

In the rest of this section, we fix an  $r$ -ranked automaton  $\mathcal{A}$  and we call  $L$  its associated language. The proof of Theorem 2 will follow the same arguments as Theorem 1 using the Robustness lemma for trees.

**Lemma 5** (Robustness). *Let  $n \geq 10r^{2m+1}/\varepsilon$ , and let  $T$  be a  $r$ -ranked tree of size  $n$  such that  $\text{dist}(T, L) \geq \varepsilon n$ . Then for every admissible subset  $\Pi \subseteq \mathcal{C}(\mathcal{A})$ , there exists an integer  $1 \leq i \leq 2r^{2m+1}/\varepsilon$ , such that the number of  $\Pi$ -infeasible  $i$ -subtrees is at least  $\frac{1}{64r^{4m+3}} \times \varepsilon^2 n$ .*

We delay the proof of the above lemma to the end of Section 4.3. Its structure is the same as the one of Lemma 1:

1. The Splitting lemma shows that if the distance between  $T$  and  $L$  is large then there are many infeasible disjoint subtrees. Its proof is by contraposition (see Figure 3):

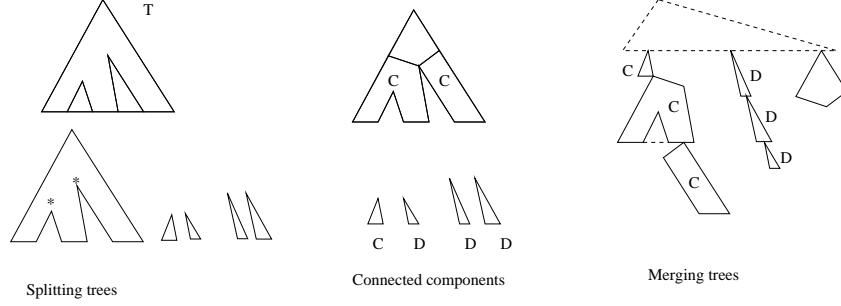


Figure 3: The correction of a tree with two infeasible subtrees where we mention  $C$  and  $D$  as some connected components (and  $h' = 3$  for the proof of Lemma 7).

- (a) First, from a cut of minimal infeasible subtrees, we construct a close feasible forest.
- (b) Then the Merging lemma shows that if a forest is feasible, then it is close to  $L$ .

2. The Amplifying lemma shows that if there are many infeasible subtrees, then there are many short ones.

*Proof of Theorem 2.* We can assume without loss of generality that the size  $n$  of  $T$  is large enough.

First, if  $T \in L$  then the tester will accept for some admissible set  $\Pi$  with probability 1.

Now suppose that  $\text{dist}(T, L) > \varepsilon n$ . The argument is similar to the one given in the proof of Theorem 1. Using the Robustness lemma for trees (Lemma 5), we get that for each admissible set  $\Pi$ , the probability to accept  $T$  for this  $\Pi$  is at most  $2^{-m}/3$ . Now, since there are at most  $2^m$  candidates  $\Pi$ , we can then conclude using the union bound, that the overall acceptance probability is upper bounded by  $1/3$ .

For the complexity, we observe that the bottleneck is due to the sampling of the  $i$ -subtrees that might be of size  $2^i$ .  $\square$

### 4.3 Robustness of the tester

In this section, all the trees we consider are  $r$ -ranked trees.

**Lemma 6** (Merging). *Let  $\Pi \subseteq \mathcal{C}(\mathcal{A})$  be admissible. Let  $F$  be the forest of a cut of a tree  $T$ . Assume that  $F$  is a  $\Pi$ -feasible forest of size  $h'_1$  with at most  $h'_2$   $*$ -nodes. Then  $\text{dist}(T, L) \leq r^m(2m + 2h'_2 + 3r^m h'_1)$ .*

*Proof.* First, we construct a subforest  $F'$  by splitting each subtree  $t$  of  $F$  into simply  $C$ -feasible subtrees, for some  $C$  of  $\Pi$ .

In more details, fix such a  $t \in F$ , and let  $\lambda$  be a run of  $t$  such that all paths of  $t$  are  $\Pi$ -feasible. Let  $C$  be the connected component of the root of  $t$ , and let  $\sigma$  be the longest path from the root such that  $\lambda(\sigma) \subseteq C$ , i.e. all nodes of  $\sigma$  are in a state of  $C$ . Denote by  $v$  the last node of  $\sigma$ . Then we cut  $t$  just before leaving  $C$ , that is between  $v$  and its successors. This leads to one simply  $C$ -feasible subtree from the root of  $t$  where the label of  $v$  is now  $*$ , and  $r$  new  $\Pi$ -feasible subtrees from the successors of  $v$ . We iterate the argument for the  $r$  subtrees using the restrictions of the same run  $\lambda$ . Therefore, the paths that we will consider in the  $r$  subtrees will start with a component  $D \neq C$  such that  $C$  is connected to  $D$  in  $\widehat{G}(\mathcal{A})$ . This guarantees a depth recursion of size at most  $m$ .

Therefore, at the end of the process each subtree of  $F$  generates at most  $1 + r + \dots + r^{m-1} \leq r^m$  subtrees in  $F'$ , that are  $C$ -feasible, for some  $C$  of  $\Pi$ , and at most  $r^m$  new  $*$ -nodes. Therefore, the size of the resulting forest  $F'$  is at most  $h''_1 = r^m \times h'_1$  and the number of  $*$ -nodes is at most  $h''_2 = (h'_2 + r^m \times h'_1)$ .

Now we only consider subtrees that are simply  $C$ -feasible for some truly connected component  $C$  of  $\Pi$ , and delete the other ones, necessarily of size 1, using at most  $h''_1$  deletions. Let  $(t_i)_{1 \leq i \leq k}$  be the remaining subtrees of  $F'$ , where  $1 \leq k \leq h''_1$ .

We now explain how to move and glue the remaining subtrees  $t_i$  in order to get a tree  $T' \in L$ . Let  $(T_0, \lambda_0)$  be a witness of  $\Pi$  such that  $T_0$  is of size at most  $r^m$ . First we reduce the size of the forest to at most  $m$  subtrees. Let  $C_{t_i}$  be a connected component of  $\Pi$  such that  $t_i$  is simply  $C_{t_i}$ -feasible. We first move and glue linearly each subtrees  $t_i$  with the same  $C_{t_i} = C$ . At each  $*$ -node, a tree of size at most  $r^m$  is also inserted so that the resulting subtree is simply  $C$ -feasible and without any  $*$ -nodes. The total number of moves or insertions of this step is at most  $2r^m \times h_2''$

Last, our final forest consists in inserting  $T_0$  and gluing the remaining subtrees to  $T_0$  in order to get a tree  $T' \in L$ , using at most  $m$  moves, and  $r^m \times m$  insertions.

The total number of elementary operations is less than

$$\begin{aligned} & h_1'' + 2r^m \times h_2'' + (m + r^m \times m) \\ = & r^m \times h_1' + 2r^m(h_2' + r^m \times h_1') + (m + r^m \times m) \\ \leq & r^m(2m + 2h_2' + 3r^m \times h_1') \end{aligned}$$

□

**Lemma 7** (Splitting). *Let  $\Pi \subseteq \mathcal{C}(\mathcal{A})$  be admissible. Let  $T$  be a tree such that  $\text{dist}(T, L) > h$ . Then  $T$  has more than  $\frac{1}{4r^{m+1}}(\frac{h}{r^m} - m) - 1$   $\Pi$ -infeasible disjoint subtrees.*

*Proof.* The proof is by contraposition. By assumption, any cut of  $T$  has at most  $\frac{1}{4r^{m+1}}(\frac{h}{r^m} - m) - 1$  infeasible subtrees. We will show how to construct a cut of size  $h'$  with at least  $(h' - 1)$  infeasible subwords such that  $\text{dist}(T, L) \leq r^m(m + 4r^{m+1}h')$ , which together with our assumption gives  $\text{dist}(T, L) \leq h$ .

First we construct a cut  $\mathcal{P}$  of  $T$  of size  $h'$  whose  $(h' - 1)$  subtrees are minimal  $\Pi$ -infeasible and disjoint subtrees. It might be the case that the top subtree of  $\mathcal{P}$  is  $\Pi$ -feasible.

We visit  $T$  bottom-up. While visiting a node  $v$ , if the subtree below  $v$  is  $\Pi$ -infeasible, we add it in our cut  $\mathcal{P}$ , we suppress the subtree rooted at  $v$  from  $T$ , and we consider  $v$  as a  $*$ -node in the remaining part of  $T$ .

At the end of the procedure we get the desired cut, which is in terms of subtrees  $\mathcal{P} = (t_i)_{1 \leq i \leq h'}$  and have at most  $h'$   $*$ -nodes.

Now we explain how to get a tree  $T' \in L$ . Let  $F$  be the forest of our cut. Every  $t_i$  has a root of degree at most  $r$ . Let  $t_i^1, \dots, t_i^r$  be the  $r$  subtrees from the root of  $t_i$  (some of them might be empty), for  $i = 1, \dots, h'$ . The forest  $F$  is therefore  $(t_i^1, \dots, t_i^r)_{i=1, \dots, h'}$  and has size at most  $rh'$ , in the same order than  $T$ . By construction  $F$  is  $\Pi$ -feasible and has at most  $h'$   $*$ -nodes. Applying Lemma 6, we get that  $\text{dist}(T, L) \leq r^m(2m + 2h' + 3r^m(rh'))$ , which is upper bounded by  $\text{dist}(T, L) \leq r^m(m + 4r^{m+1}h')$ . □

**Lemma 8** (Amplifying). *Let  $\Pi \subseteq \mathcal{C}(\mathcal{A})$  be admissible. Let  $T$  be a tree of size  $n$  with at least  $h'$   $\Pi$ -infeasible disjoint subtrees. Then there exists an integer  $1 \leq i \leq 2n/h'$  such that the number of  $\Pi$ -infeasible  $i$ -subtrees is at least  $\frac{1}{r} \times \frac{h'}{4n} \times h'$ .*

*Proof.* In this proof, we understand feasible as  $\Pi$ -feasible and we follow the structure of the proof of Lemma 4.

Let  $t_1, \dots, t_{h'}$  be some infeasible disjoint subtrees of  $T$ . Let  $a$  be a positive integer. For every integer  $i \geq 1$ , let  $s_i = |\{t_j : \text{depth}(t_j) = i\}|$ . Since the root of a subtree may be shared with the leaf of another subtree as a  $*$ -node, we have  $|\{t_j : \text{depth}(t_j) > a\}| \leq \frac{n}{a+1}$ , and therefore  $\sum_{i=1}^a s_i \geq h' - \frac{n}{a}$ .

Take  $a = \frac{2n}{h'}$ . Then  $\sum_{i=1}^a s_i \geq \frac{h'}{2}$ , thus there exists some  $1 \leq i \leq a$  such that  $s_i \geq \frac{h'}{2a}$ .

To lower bound the number of infeasible  $i$ -subtrees, we count the number of  $i$ -subtrees that contain at least one subtree  $t_j$  of depth  $i$ . These subtrees are also infeasible since they contain one of the infeasible subtrees  $t_j$ . Note that since the subtrees  $t_j$  are disjoint, each infeasible  $i$ -subtree contains at most  $r$  of the  $t_j$  of depth  $i$ . Moreover, each infeasible subtree  $t_j$  of depth  $i$  is included in at least one infeasible  $i$ -subtree.

We then get that the number of infeasible  $i$ -subtrees is at least  $\frac{1}{r} \times \frac{h'}{2a}$ . □

*Proof of Lemma 5.* From the Splitting lemma with  $h = \varepsilon n$  and since  $n$  is large enough, the tree  $T$  has a number of  $\Pi$ -infeasible disjoint subtrees greater than  $h' = \frac{\varepsilon n}{8r^{2m+1}}$ . Now, by the Amplifying lemma, there exists an integer  $1 \leq i \leq 16r^{2m+1}/\varepsilon$  such that the number of  $\Pi$ -infeasible  $i$ -subtrees is at least  $\frac{1}{64r^{4m+3}} \times \varepsilon^2 n$ .  $\square$

## 5 Extension to unranked trees

A run  $\lambda$  is generalized such that if  $u$  is a node with successors  $v_1, \dots, v_l$  in states  $\lambda(v_1), \dots, \lambda(v_l)$  and there is a  $q$  such that  $\lambda(v_1), \dots, \lambda(v_l) \in \delta(q, l(u))$ , then  $\lambda(u) = q$ .

In order to generalize **Regular ranked tree language tester** to unranked regular trees we can either directly construct a new tester, or encode an unranked tree as a binary tree and use **Regular ranked tree language tester** on this encoding. We study here the second approach.

Every unranked tree  $T$  can be coded as a binary tree  $e(T)$  but many encodings are possible. Consider the Rabin encoding where each node  $v$  of the unranked tree is a node  $v$  in the binary encoding, the left successor of  $v$  in the binary tree is its first successor in the unranked tree, the right successor of  $v$  in the binary tree is its first sibling in the unranked tree. New nodes with labels  $\perp$  are added to complete the binary tree when there are no successor or no sibling in the unranked tree (See Figure 4).

It is a classical observation that a language  $L$  of unranked trees is regular [Tha67] iff the language  $L'$  of its encoding is regular. In the case of XML files, the labels (or tags) are the states and we associate with each label  $q$  a regular expression  $r_q$ . A node labelled  $q$  is *parsed correctly* if the labels of its successors belong to the regular expression  $r_q$ . The DTD gives the sequence of rules  $q : r(q)$  but it is assumed to be 1-unambiguous [BKW98]. In a general situation, we associate with a state  $q$  and symbol  $a$  a regular expression  $r_{a,q}$ .

If we code unranked trees as binary trees, the trees have a degree 2, but an additional label  $\perp$  is used for certain leaves. If we remove the leaves labelled with  $\perp$ , we consider trees with degree at most 2, where nodes may only have a left or a right successor. An *extended 2-ranked tree* is a structure:

$$(N, [N], root, l : [N] \rightarrow \Sigma, d : [N] \rightarrow [2], s : [N] \times [2] \rightarrow [N]),$$

as before, i.e.  $l$  is the label function,  $d$  is the degree function and  $s$  is the successor partial function which defines two successors. Nodes may have a left and a right successor, or a left successor, or a right successor. The root has only a left successor. We then have a natural one-to-one correspondence between every unranked tree and extended 2-ranked tree (See Figure 4).

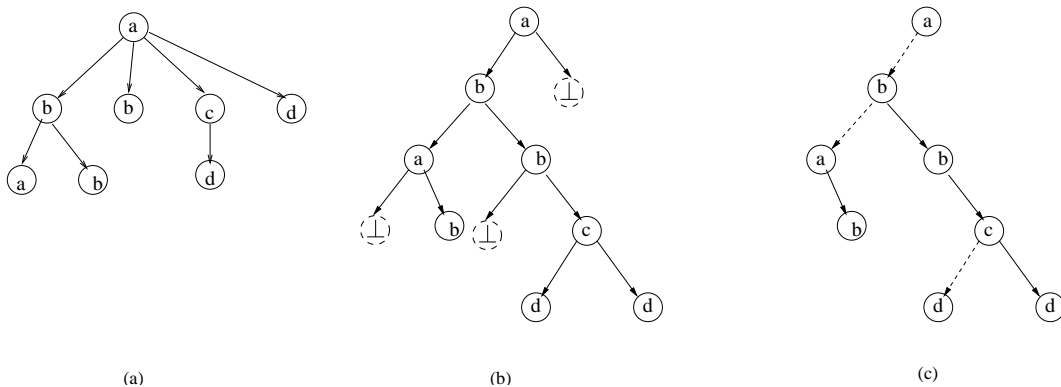


Figure 4: Encoding of an unranked tree in (a), as a binary tree with  $\perp$  in (b), and as an extended 2-ranked tree in (c).

Generalize automata and edit distance with moves to extended 2-ranked trees. Then our tester on 2-ranked trees can be directly generalized to extended 2-ranked trees. The distance between two unranked

trees is of the same order as the distance on their encodings as extended 2-ranked trees: we just check that every elementary operation on unranked trees (resp. extended 2-ranked trees) translates in finitely many operations on extended 2-ranked trees (resp. unranked trees).

**Proposition 2.** *For every unranked trees  $T$  and  $T'$ ,*

$$\text{dist}(e(T), e(T'))/3 \leq \text{dist}(T, T') \leq 3 \times \text{dist}(e(T), e(T')).$$

*Proof.* As previously said, we only need to decompose any elementary transformation from one encoding to the other one. An insertion (respectively a deletion) on  $T$  is simulated by 1 insertion (respectively 1 deletion) and 1 move on  $e(T)$ . A move on  $T$  is simulated by 1 insertion, 1 move and 1 deletion on  $e(T)$ .

Conversely, an insertion (respectively a deletion) on  $e(T)$  corresponds to 1 insertion (respectively 1 deletion) on  $T$ . A move on  $e(T)$  corresponds to 1 insertion, 1 move and 1 deletion on  $T$ .  $\square$

Moreover one can sample any  $k$ -subtree of  $e(T)$ , for any  $k$ , since one can compute the  $k$ -subtree of  $e(T)$  from any node  $v$  by using  $2^k$  queries *suc* and *sib* to  $T$ , starting from node  $v$ . Therefore our tester can be extended to unranked trees and we obtain:

**Corollary 3.** *Regular properties of unranked trees are testable.*

## 6 The tree edit distance with moves problem on ordered trees

The *tree distance problem* takes two ordered trees  $T_1, T_2$  and an integer  $p$  as input, and decides if  $\text{dist}(T_1, T_2) \leq p$ . For the case of the tree edit distance without moves, the problem is known to be NP-complete on unordered trees and P-computable on ordered trees [Tai79, AG97]. Let EDM be the tree edit distance problem for ordered trees and the edit distance with moves. We will show that EDM is NP-complete for both unranked trees and ranked trees. One might notice that this result remains true for unordered trees when the edit distance with moves is generalized to those trees, but this is not the purpose of this paper.

Recall that the one-dimensional perfect bin-packing problem, BP, is strongly NP-complete, i.e. BP is still NP-complete when instances are unary encoded [GJ79] (since it is 3-Partition with less constraints). The input of BP is a set  $X = \{x_1, \dots, x_n\}$  of positive integers and an integer  $B$ , the capacity. Then the input  $X, B$  satisfies BP if there exists a partition of  $X$  into  $X_1, \dots, X_k$  that satisfies the *BP condition*:  $\sum_{x_i \in X_j} x_i = B$ , for every  $j = 1, \dots, k$ . Note that  $k$  is uniquely determined. If  $X, B$  satisfies BP, then necessarily  $B$  divides  $\sum_{i=1}^n x_i$  and  $k$  satisfies  $\sum_{i=1}^n x_i = k \times B$ .

We start as a warm up to prove that EDM is NP-complete on unranked trees. Then we will generalize to binary trees. The proof will consist in a reduction of BP to EDM, using an encoding of an instance of BP into subtrees with many branches. Formally, a *branch* of length  $l$  is a linear tree with  $l$  nodes, i.e. such that every node has only one successor, excepting the leaf.

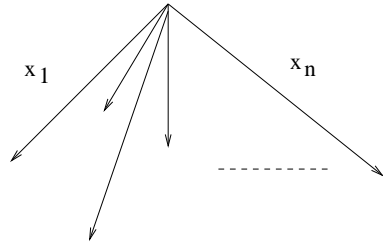
**Proposition 3.** *EDM is NP-complete on unranked trees.*

*Proof.* Consider a reduction from BP to EDM, similar to the one introduced by [SS06] in the case of words. Given  $X, B$  such that  $\sum_{i=1}^n x_i = k \times B$ , for some integer  $k$ , we construct (see Figure 5)  $T_1$  as an unranked ordered tree of size  $k \times B + 1$  with  $n$  branches connected to the root: the  $i$ -th branch has length  $x_i$ . The tree  $T_2$  is a tree of size  $k \times B + 1$  with  $k$  branches of length  $B$  connected to the root. Last, the distance parameter is  $p = n - k$ .

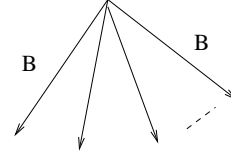
Since the out-degree of the root of  $T_1$  is  $n$  and the one of  $T_2$  is  $k$ , at least  $n - k$  branches require one operation in order to modify  $T_1$  to  $T_2$ . Therefore  $\text{dist}(T_1, T_2) \geq n - k$ .

If the input  $X, B$  satisfies BP, there exists a partition  $X_1, \dots, X_k$  which satisfies the BP condition. Each  $X_j$  defines  $|X_j| - 1$  moves of branches of  $T_1$  to one of its branches, in order to get one branch of  $T_2$  of size  $B$ . Therefore,  $n - k$  moves of some branches of  $T_1$  gives  $k$  branches of size  $B$ , so that  $\text{dist}(T_1, T_2) \leq n - k$ .

If the input  $X, B$  does not satisfy BP, then for any partition  $X_1, \dots, X_k$ , the BP condition is not satisfied. Let us show that distance  $\text{dist}(T_1, T_2) > n - k$ . First, we show by contradiction that performing only moves



(a)  $T_1$  made of  $n$  branches each of length  $x_i$



(b)  $T_2$  made of  $k$  branches of length  $B$

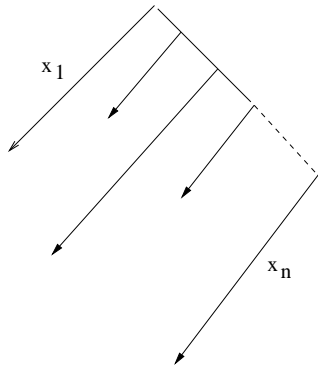
Figure 5: Unranked Trees  $T_1$  and  $T_2$ .

requires more than  $n - k$  moves. Assume that a sequence of  $n - k$  moves can modify  $T_1$  to  $T_2$ . Then the moves are exactly the ones of  $n - k$  branches, which define a partition that satisfies the BP condition. This is a contradiction from our assumption that  $X, B$  does not satisfy BP.

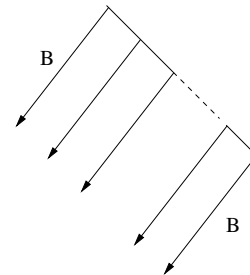
Second, consider a sequence of elementary operations (insertions, deletions or moves) that modify  $T_1$  to  $T_2$ . We want to prove that the length of the sequence is greater than  $n - k$ . Since  $T_1$  and  $T_2$  have the same size, the number of insertions equals the number of deletions. Let  $m$  be the number of moves in the considered sequence. If  $m > n - k$  then the sequence is long enough. If  $m = n - k$ , we are in the first case. Assume that  $m < n - k$ . Since the out-degree of  $T_2$  is  $k$ , at least  $n - k - m > 1$  branches were not deleted by the moves. This required at least  $n - k - m$  deletions. Therefore the number of insertions is also at least  $n - k - m$ . Thus the length of the sequence is at least  $2(n - k - m) + m > n - k$  when  $m < n - k$ , and we can conclude that  $\text{dist}(T_1, T_2) > n - k$ .  $\square$

**Theorem 3.** *EDM is NP-complete on 2-ranked trees.*

*Proof.* Consider a similar reduction from BP to EDM on binary trees. Given  $X, B$  such that  $\sum_{i=1}^n x_i = k \times B$ , for some integer  $k$ , we construct (see Figure 6) a tree  $T_1$  of size  $k \times B + n$  and a tree  $T_2$  of size  $k \times B + k$ . For  $T_1$  we start with a single branch of length  $n$ . Then we add bottom-up a left branch of length  $x_i$  to the  $i$ -th node of the initial branch. The tree  $T_2$  is similarly constructed starting with a branch of length  $k$  and adding  $k$  left branches of length  $B$ . Last, the distance parameter is  $p = 2(n - k)$ . Note that  $T_1$  and  $T_2$  look like the binary encoding of their analogous for the unranked case.



(a) Binary tree  $T_1$  made of  $n$  branches each of length  $x_i$



(b) Binary tree  $T_2$  made of  $k$  branches of length  $B$

Figure 6: Binary Trees  $T_1$  and  $T_2$ .

Since  $T_1$  has  $n - k$  more nodes than  $T_2$ , one needs to delete at least  $n - k$  nodes from  $T_1$ . Note that the

out-degree of all these additional nodes is 2. In order to delete each of these nodes, one need to perform at least one operation on one of its ancestor. Such an operation has to decrease the out-degree of the ancestor to 1. Last, a single operation can decrease the out-degree of only one node. Therefore, we have  $\text{dist}(T_1, T_2) \geq 2(n - k)$ .

If the input  $X, B$  satisfies BP, we proceed similarly as for the unranked case. There exists a partition  $X_1, \dots, X_k$  which satisfies the BP condition, and therefore defines  $n - k$  moves of some branches of  $T_1$  into the remaining  $k$  branches. We then need to remove the  $(n - k)$  nodes where the moves originated, i.e. the roots of the moved branches, so that  $\text{EDM}(T_1, T_2) \leq 2(n - k)$ .

If the input  $X, B$  does not satisfy BP, then for any partition  $X_1, \dots, X_k$ , the BP condition is not satisfied. First, assume that there is a sequence of transformations from  $T_1$  to  $T_2$  that consists in  $n - k$  moves and  $n - k$  deletions. Then necessarily, the moves are used to decrease the degree of the deleted 2-degree nodes. Moreover, since they must not increase any out-degree, they have to move some left branches to the leaves of other left branches. This means that if we get  $T_2$  in this way, the moves define a partition that satisfies the BP condition, which is a contradiction.

Now we want to prove that the length of any sequence that modify  $T_1$  to  $T_2$  is greater than  $2(n - k)$ . As in the unranked case, the number of deletions equals the number of insertions plus  $n - k$ , that is the size of  $T_1$  minus the size of  $T_2$ . Let  $m$  be the number of moves. If  $m > n - k$  then the sequence is long enough. If  $m = n - k$ , there is no insertion and we are in the first case. Assume that  $m < n - k$ , then there is  $n - k - m$  branches that need to be deleted, which requires at least  $n - k - m$  insertions. Therefore the length of the sequence is  $m + (n - k - m) + (n - k - m + n - k) > 2(n - k)$  when  $m < n - k$ , and we conclude that  $\text{dist}(T_1, T_2) > 2(n - k)$ .  $\square$

## 7 Conclusion

We proved that ranked and unranked regular tree languages have testers for the tree edit distance with moves. We first gave a tester simpler than the [AKNS00] tester for regular languages when the edit distance with moves replaces the standard edit distance. Then we extended the construction to ranked trees. As a corollary, unranked trees are also testable because we can use an encoding into extended 2-ranked trees where the distances are close. Although the distance problem between two ordered trees is P computable for the classical tree edit distance, it is NP-complete for the tree edit distance with moves.

## Acknowledgment

We thank Dana Shapira for sending us a preliminary version of [SS06], and the anonymous referees for their constructive remarks that improved the presentation of the paper, and for having pointed out a mistake in the original proof of Proposition 3.

## References

- [AFKS00] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.
- [AG97] A. Apostolico and Z. Galil. *Pattern Matching Algorithms*, chapter 14: Approximate Tree Pattern Matching. Oxford University Press, 1997.
- [AKNS00] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6), 2000.
- [BK95] M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

- [BKW98] A. Bruggemann-Klein. and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142:182–206, 1998.
- [BLR93] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.
- [CK02] H. Chockler and O. Kupferman.  $\omega$ -regular languages are testable with a constant number of queries. In *Proceedings of the 6th Workshop on Randomization and Approximation Techniques in Computer Science*, pages 26–38, 2002. LNCS volume 2483.
- [Cor03] G. Cormode. *Sequence Distance Embeddings*. PhD thesis, University of Warwick, 2003.
- [GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
- [GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. Bell Laboratories Murry Hill, NJ, 1979.
- [RS96] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):23–32, 1996.
- [Rub99] R. Rubinfeld. On the robustness of functional equations. *SIAM Journal on Computing*, 28(6):1972–1997, 1999.
- [SS06] D. Shapira and J. Storer. Edit distance with move operations. *Journal of Algorithms*, 2006. To appear.
- [Tai79] K. C. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26:422–433, 1979.
- [Tha67] J. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1(4):317–322, 1967.