Property testing of regular tree languages[*]

Frédéric Magniez[1] and Michel de Rougemont[2]

[1] CNRS–LRI, UMR 8623 Université Paris–Sud, France, `magniez@lri.fr`
[2] LRI & Université Paris II, France `mdr@lri.fr`

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 regular trees. In the complete version of the paper, we show that the distance problem is NP-complete on ordered trees.

1 Introduction

Inspired by the notion of Self-Testing [3, 4], Property Testing has been initially defined and studied for graph properties [7]. It has been successfully extended for various classes of finite structures. Let $\mathbf{K}$ be a class of finite structures and a distance function dist, i.e. a function between structures of $\mathbf{K}$. An ε-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 ε-far from $\mathbf{K}_0$ with high probability. A class $\mathbf{K}_0$ is testable if for every sufficiently small ε there exists an ε-tester for $\mathbf{K}_0$ whose time complexity is in $O(f(\varepsilon))$, *i.e.* independent of n.

For the Hamming distance, regular languages and Σ_2-definable graphs are testable [2, 1]. Testers have also been generalized to the infinite regular languages [5]. In this paper we initiate the study of Property Testing with the Edit distance, when insertions and deletions of letters on words, of nodes and edges on trees, are the elementary operations. We specifically require an additional operation: the *move* of any entire subword or subtree in one step.

First (Section 3), we develop a new tester for regular languages on words that greatly simplifies the tester of [2] and improves its complexity by a $\log(1/\varepsilon)$ factor.

Then (Section 4), we initiate the study of Property Testing on trees. The testability of regular tree languages is a well known open problem [5] for the standard Edit distance. We solve this problem when moves are allowed, by proving the testability of regular ranked tree languages.

The Word Edit distance with moves decision problem and the standard Tree Edit distance decision problem are computable in polynomial time [6, 10]. In the complete version of the paper, we prove 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.

Finally (Section 5), we discuss the possibility of generalizing the testability to unranked trees. As a direct application, it would imply than one can decide in constant time if a large XML document follows a DTD or is far from it.

2 Preliminaries

2.1 Property Testing

Recall the notion of a (Property) tester [7] on a class $\mathbf{K}$ of finite structures for which a distance function between structures has been defined.

We say that two structures $U_n, V_m \in \mathbf{K}$, whose domains are respectively of size n and m, are ε-*close* if their distance is less than $\varepsilon \times \max(n, m)$. They are ε-*far* if they are not ε-close. In this paper, we consider this notion of closeness for words and trees since the representation of their structure is of linear size. For other classes, such as graphs, one may define the closeness relatively to the representation size (for e.g., εn^2 for graphs) instead of the domain size.

Definition 1. *Let $\varepsilon \geq 0$ be a real. An ε-tester for a class $\mathbf{K}_0 \subseteq \mathbf{K}$ is a random-ized algorithm A such that:*
(1) If $U \in \mathbf{K}_0$, A always accepts;
(2) If U is ε-far from $\mathbf{K}_0$, then $\Pr[A \text{ rejects}] \geq 2/3$.

The *query complexity* is the number of boolean queries to the structure U of $\mathbf{K}$. The *time complexity* is the usual time complexity where the complexity of a query is one and the time complexity of an arithmetic operation is also one.

A class $\mathbf{K}_0 \subseteq \mathbf{K}$ is *testable* if for every sufficiently small $\varepsilon > 0$, there exists an ε-tester whose time complexity depends only on ε.

2.2 Words

Let Σ be a finite alphabet of constant size and for the sake of simplicity, the reader might think that $\Sigma = \{0, 1\}$. We now consider the words on the alphabet Σ. Every word W is a finite structure $(N, [N], l : [N] \to \Sigma)$, where $[N]$ denote the set $\{1, \ldots, N\}$. 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. A word w is a *subword* of W if $w = W[i, \ldots, j]$, for some i, j.

An *elementary operation (on words)* is a deletion or an insertion of a letter, or a move: given a subword and a position, a *move* is a transformation of the word, where the subword has been removed of its current position and inserted in the given position. Notice we omit letter replacement operations since such an operation can be simulated using one deletion and one insertion.

The standard Edit distance only considers the operations without moves, and this new distance is essential for most of the arguments.

Definition 2. *The distance between two words W and W' is the minimum num-ber of elementary operations necessary to reach W' from W, noted $\mathsf{dist}(W, W')$. The distance between W and a language L, noted $\mathsf{dist}(W, L)$, is the minimum $\mathsf{dist}(W, W')$ when $W' \in L$.*

2.3 Trees

Let T be an ordered Σ-tree, *i.e.* a tree with labels $\sigma \in \Sigma$ on the nodes. It 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.

Let us first consider r-ranked trees for some fixed constant r. An r-ranked tree T is a finite structure

$$(N, [N], root, l : [N] \to \Sigma, d : [N] \to [r], s : [N] \times [r] \to [N]),$$

where N is the size of T, $root$ 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.

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 of the node v and its i-th node successor in T, if $i \le d(v)$.

The classical Tree Edit distance [10] assumes basic insertions, deletions on a tree and modifications of labels (see Figure 1). A *node insertion* (u, σ) on an edge (v_1, v_2) replaces the edge (v_1, v_2) by the edge (v_1, u), set v_2 to be the only successor of u, and labels u by σ. A *node deletion* is the inverse of a node insertion. An *edge insertion* (v, u, σ, i) to a node v of T inserts the leaf u with label σ between the $(i-1)$-th and the i-th successor of v, provided that $d(v) < r$. The inverse operation is an *edge deletion*.

We will also allow some moves in T (see Figure 1). A *complete subtree t of T* takes a node of T as root and is the substructure restricted to a subset of nodes such that all leaves of t are also leaves of T. A *move (t, v, i)* of a complete subtree t to a node v moves in one step t between the $(i-1)$-th and the i-th successor of v, provided the degree of v allows it. An *elementary operation (on trees)* is one of the above operations. We define $\mathsf{dist}(T, T')$ and $\mathsf{dist}(T, L)$ as in Definition 2, for any trees T, T' and tree language L. For unranked trees, the above definitions
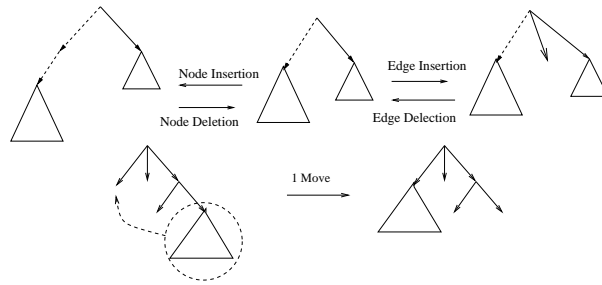

Fig. 1. Elementary operations on trees.

might be generalized by removing the degree condition and replacing $[r]$ by $[N]$.

Moreover the definition of a complete subtree is adapted so that t is a complete subtree of T if in addition it satisfies: every successors in t of every node v of t are subwords of the successors in T of v.

3 Testing regular languages

3.1 Basic definitions

Let $\mathcal{A}$ be a deterministic 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 $\mathcal{A}$ reaches 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 at most m that connects q_1 to q_2.*

Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connects states that are connected by a word of size 1, that is a letter. We assume without lost of generality that $G(\mathcal{A})$ is connected. Since we will only speak about strongly connected components, we omit the term 'strongly'. A connected component C of $G(\mathcal{A})$ is *truly connected* if there is a non empty path of $G(\mathcal{A})$ inside C. Observe that a nontruly connected component is necessarily a singleton. We will denote by $\widehat{G}(\mathcal{A})$ the graph of the connected components of $G(\mathcal{A})$.

Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connects states that are connected by a word of size 1, that is a letter. We assume without lost of generality that $G(\mathcal{A})$ is connected. Since we will only speak about strongly connected components, we omit the term 'strongly'. A connected component C of $G(\mathcal{A})$ is *truly connected* if there is a non empty path of $G(\mathcal{A})$ inside C. Observe that a nontruly connected component is necessarily a singleton. Let $\widehat{G}(\mathcal{A})$ denote the graph of the connected components of $G(\mathcal{A})$.

Definition 3. *Let $\Pi = (C_1, \ldots, C_k)$ be a path of $\widehat{G}(\mathcal{A})$. Then Π is* admissible *if C_1 (resp. C_k) contains an initial (resp. final) state.*

Definition 4.

1. *Let C be a truly connected component of $G(\mathcal{A})$. A word w is C-simply feasible if it connects two states of C.*
2. *Let Π be a path of $\widehat{G}(\mathcal{A})$. A word w of is Π-feasible if it connects two states q_1 and q_2 along a path visiting only some of the connected components of Π.*

A word w is *(simply) Π-infeasible* if it is not (simply) Π-feasible.

A *cut* of a word W is an ordered partition of W in subwords. We will think on this partition as an ordered forest of words. Below we omit the term 'ordered'. A cut F is *Π-feasible* if every word of F is Π-feasible.

3.2 The tester

The tester takes random subwords of finite length of W and will test feasibility for finitely many Π, that is at most 2^m where m is the number of state of the automaton. The Robustness lemma will insure that if a word W is far, then with high probability a random subword of finite length will be infeasible.

Tester for regular language $(\mathcal{A}, \varepsilon, W)$:

If the size n of W is less than $15m^2/\varepsilon$
   then simply evaluate $\mathcal{A}$ on W and accept iff $\mathcal{A}$ accepts W.
Else do the following:
For $i = 1, \ldots, \log(5m^2/\varepsilon)$ {
   Compute $N_i = \Theta(\frac{2^{-i}m^3 \log(m^2/\varepsilon)}{\varepsilon})$.
   Choose N_i random subwords w_j^i of W of size 2^{i+1}, for $j = 1, \ldots, N_i$ }
For every admissible path Π of $\widehat{G}(\mathcal{A})$ {
   If all the w_j^i are Π-feasible then accept W (and stop) }
Reject W.

Theorem 1. *For every real $\varepsilon > 0$, every automaton $\mathcal{A}$ with m states, and every word W, the algorithm **Tester for regular language** $(\mathcal{A}, \varepsilon, W)$ is an ε-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 $O(2^m m^3 \log^2(m^2/\varepsilon)/\varepsilon)$.*

Proof. We can assume w.l.o.g. 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 Π-feasible for some admissible Π. Therefore every subword of W is Π-feasible for this path Π. Thus the tester accepts W with probability 1.

Suppose that $\mathsf{dist}(W, L) > \varepsilon n$ and fix an admissible path Π. Using the Robustness lemma (Lemma 1), we get that the probability to accept W for this Π is in $O(2^{-m})$. Since there is at most 2^m candidates Π, we can conclude, using the union bound, that the acceptance probability is upper bounded by $1/3$.   □

Corollary 1. *Regular properties of words are ε-testable.*

We now state the Robustness lemma. The notion of robustness was first defined in [9] and studied in [8]. In the rest of this section, we fix an automaton $\mathcal{A}$ and we call L its associated language.

Lemma 1 (Robustness). *Let $n \geq 15m^2/\varepsilon$, and let W be a word of size n such that $\mathsf{dist}(W, L) \geq \varepsilon n$. Then for every admissible path Π of $\widehat{G}(\mathcal{A})$, there exists an integer $1 \leq i \leq \log(5m^2/\varepsilon)$, such that the number of Π-infeasible subwords of size 2^{i+1} is at least $\frac{2^{i+1}}{90m^2 \log(5m^2/\varepsilon)} \times \varepsilon n$.*

The sketch of the proof of the Robustness lemma takes the following steps (see Figure 2):

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:
   (a) First, from a cut of minimal infeasible subwords, we construct a close feasible cut F.
   (b) Then the Merging lemma which shows that if a cut F is feasible, then it is close to L.
2. The Amplifying lemma shows that if there many infeasible words, then there are many short ones.
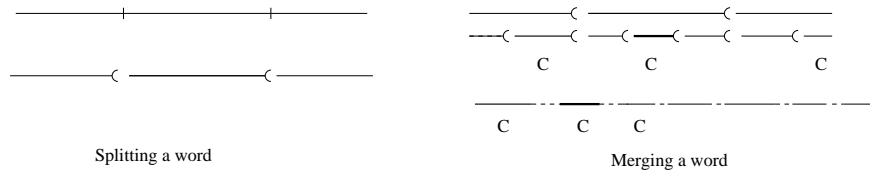

Splitting a word   Merging a word

Fig. 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 2).

3.3 Robustness of the tester

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

Proof. The proof is by contraposition and we understand feasible as Π-feasible. First we construct a cut $\mathcal{P}$ of W of size h' whose $h'-1$ first subwords are minimal infeasible and disjoint subwords. The last subword of $\mathcal{P}$ is either infeasible or feasible. And in this last case, the entire word W might feasible and $h' = 1$.

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

> Initially: $h' = 0$ and $i = j = 1$.
> While $(j \leq |W|)$ {
>   While (subword $W[i, \ldots, j]$ is Π-feasible and $j < |W|$) {increase j}
> $h' = h' + 1$, $w_{h'} = W[i, \ldots, j]$, $i = j + 1$, $j = i$.}

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 $i = 1, \ldots, h'$. When $w_{h'}$ is feasible then $w_{h'}' = w_{h'}$. By construction of w_i, the subwords w_i' are feasible. Let F be the cut of the $(w_i')_{1 \leq i \leq h'}$. Applying Lemma 3, we get that $\mathsf{dist}(F, L) \leq m + 2m^2 \times h'$. Because $\mathsf{dist}(W, F) \leq h'$, then $\mathsf{dist}(W, L) \leq m + 2m^2 \times h'$. But by assumption, $h' - 1 \leq \frac{h-3m^2}{2m^2}$, therefore $\mathsf{dist}(W, L) \leq h$. $\qquad\square$

Lemma 3 (Merging). *Let* $\Pi = (C_1, \ldots, C_k)$ *be an admissible path of* $\widehat{G}(\mathcal{A})$. *Let* F *be a* Π-*feasible cut of size* h'. *Then* $\mathsf{dist}(F, L) \leq m + 2m^2 h'$.

Proof. First, we split each subword of F in C-feasible subwords, for some $C \in \Pi$. Given a Π-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 keeps subwords that are C-feasible for some truly connected component C. Moreover, each initial subword of F splits in at most k subwords from which at most k letters are deleted, where k is less than m, where m is the number of state of the automaton. 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 Π 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 Π. Up to some renaming, we assume now that $(C_1', \ldots, C_l')$ are in the same order than $(C_1, \ldots, C_k)$, up to some repetitions.

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 world $W' = g_0.w_1.g_1 \ldots w_l.g_h$. By construction $W' \in L$. In this last step, we did at most $m \times (l+1)$ insertions.

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

Lemma 4 (Amplifying). *Let* Π *be a path of* $\widehat{G}(\mathcal{A})$. *Let* W *be a word of length* n *with at least* h' Π-*infeasible disjoint subwords. Then there exists an integer* $1 \leq i \leq \log(2n/h')$ *such that the number of* Π-*infeasible subwords of size* 2^{i+1} *is at least* $\frac{2^i(h'-4)}{6\log(2n/h')}$.

Proof. In this proof, we understand feasible as Π-feasible.

Let $w_1, \ldots, w'_h$ be some infeasible disjoint subwords of W. Let a be a positive integer. For every integer $i \geq 1$, let $s_i = |\{w_j : 2^{i-1} + 1 \leq |w_j| \leq 2^i\}|$. Since we have $|\{w_j : |w_j| > 2^a\}| \leq \frac{n}{2^a}$, we therefore get $\sum_{i=1}^{a} s_i \geq h' - \frac{n}{2^a}$.

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 $\frac{2^i}{3} \times \frac{h'-4}{2a}$. $\qquad\square$

Proof (of Lemma 1). From the Splitting lemma with $h = \varepsilon n$, the word W has more than $h' = \frac{2\varepsilon n}{5m^2}$ Π-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 Π-infeasible subwords of size 2^{i+1} is at least $\frac{2^i((2\varepsilon n/5m^2)-4)}{6\log(5m^2/\varepsilon)} \geq \frac{2^{i+1}}{90m^2\log(5m^2/\varepsilon)} \times \varepsilon n$. $\qquad\square$

4 Testing regular ranked tree languages

4.1 Basic definitions

A *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 σ, and $\delta : Q^{\leq r} \times \Sigma \to Q$ is the transition function.

A *subtree t* of T takes a node v_1 of T as root and is the substructure restricted to nodes $\{v_1, \ldots, v_m\}$ where $v_2, \ldots, v_m$ are connected to v_1. The leaves of T among $\{v_1, \ldots, v_m\}$ are leaves of t, while some nodes are leaves in t but not in T and called $*$-nodes where the new label is $*$. By extension, a *subtree t* is a tree where some of the leaves are $*$-nodes.

An *assignment λ* for a subtree 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 λ on all the nodes of the subtree such that if u is a node with successors $v_1, \ldots, v_l$ where $l \leq r$ in states $\lambda(v_1), \ldots, \lambda(v_l)$ then $\lambda(u) = \delta(\lambda(v_1), ldots, \lambda(v_l), l(u))$. A run *accepts* if the state of the root is in F.

Two states q and q' are *connected* if there exists a finite subtree t of size at most r^m and a run λ such that one leaf of t is assigned the state q, and the root of t is assigned the state q'. Let $G(\mathcal{A})$ be the directed graph whose vertices are the states of $\mathcal{A}$ and edges connect states that are connected by a subtree of depth 1.

We assume without lost 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'. We consider a set Π of connected components of $G(\mathcal{A})$ and generalize the notions of *Π-feasibility* for subtrees.

Definition 5. *Let Π be a set of connected components of $G(\mathcal{A})$. Then Π is admissible if there is a pair (T_0, λ_0), the witness of Π, such that λ_0 is an assignment of the tree T_0 which visits every connected components Π, and no more.*

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

Definition 6. *Let Π be a set of connected components of $G(\mathcal{A})$. A path σ from a leaf to the root of a subtree t is Π-feasible if there exists a run which visits along σ only some connected components of Π. A subtree t is simply Π-feasible if there exists a path σ in T such that σ is Π-feasible. A subtree t is Π-feasible if there exists a run λ such that for all paths σ in t, σ is Π-feasible for λ.*

A subtree t is *(simply) Π-infeasible* if it is not (simply) Π-feasible.

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 $*$-node of one subtree and the root of the

other subtree. A *cut* of a tree T is a partial ordered partition of T in subtrees. We will think on this partition as an ordered forest of subtrees. A *forest* of subtrees is a partial ordered set of subtrees. Below we omit the term 'ordered'.

We naturally extend the Tree Edit distance (with moves) to forests, where the move operation can now either be applied to two subtrees of the forest or take one subtree and generate two new subtrees. Since the Tree Edit distance and the Tree Forest Edit distance are 4-equivalent (see Proposition 2), we do not distinguish them for the sake of simplicity. In other words, the Tree Forest Edit distance allows for some temporarily disconnection of complete subtrees.

Proposition 2. *If two trees T, T' have Tree Edit distance h then their Tree Forest Edit distance is in $[h/4, h]$.*

A forest of subtrees is Π-*feasible* if every subtree is Π-feasible.

4.2 The tester

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

Tester for regular ranked tree language $(\mathcal{A}, \varepsilon, T)$**:**

If the size n of T is in $O(r^{2m+1}/\varepsilon)$
   then simply evaluate $\mathcal{A}$ on T and accept iff $\mathcal{A}$ accepts T.
Else do the following:
Compute $N = \Theta(mr^{4m+3}/\varepsilon^2)$
For $i = 1, \ldots, 2r^{2m+1}/\varepsilon$ {
   Choose N random nodes v_j^i, for $j = 1, \ldots, N$.
   Query the i-subtree t_j^i of T from v_j^i, for $j = 1, \ldots, N$ }
For every admissible set Π of connected component of $G(\mathcal{A})$ {
   If all the t_j^i are Π-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 **Tester for regular ranked tree language** $(\mathcal{A}, \varepsilon, T)$ is an ε-tester for the language recognized by $\mathcal{A}$. Moreover, its query complexity is in $O(mr^{4m+3}2^{(r^{2m+1})/\varepsilon}/\varepsilon^2)$, and its time complexity in $O(2^m mr^{4m+3}2^{(r^{2m+1})/\varepsilon}/\varepsilon^2)$.*

Corollary 2. *Regular properties of trees are ε-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 follow the same arguments of Theorem 1 using the Robustness lemma for trees.

Lemma 5 (Robustness). *Let $n = \Omega(r^{2m+1}/\varepsilon)$, and let T be a r-ranked tree of size n such that $\mathsf{dist}(T, L) \geq \varepsilon n$. Then for every admissible set Π of connected components of $G(\mathcal{A})$, there exists an integer $1 \leq i \leq 2r^{2m+1}/\varepsilon$, such that the number of Π-infeasible i-subtrees is in $\Omega(\frac{1}{r^{4m+3}} \times \varepsilon^2 n)$.*

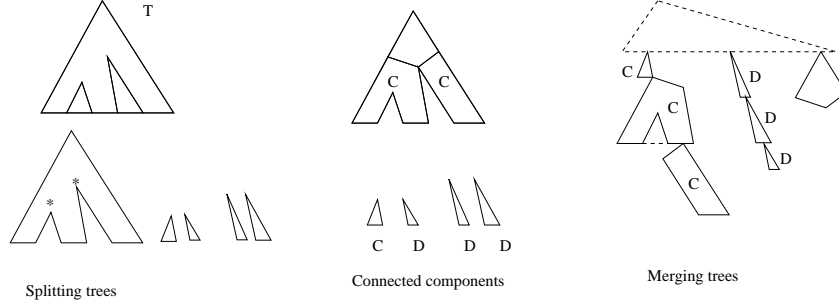The structure of the proof of the Robustness lemma is the same than the one of Lemma 1 (see Figure 3).

Splitting trees   Connected components   Merging trees

Fig. 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 6).

4.3 Robustness of the tester

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

Lemma 6 (Splitting). *Let Π be an admissible set of connected components of the graph $G(\mathcal{A})$. Let T be a tree such that $\mathsf{dist}(T, L) > h$. Then T has more than $\frac{1}{3r^{m+1}}(\frac{h}{r^m} - 1) - 1$ Π-infeasible subtrees.*

Proof. The proof is by contraposition and we understand feasible as Π-feasible. First we construct a cut $\mathcal{P}$ of T of size h' whose $h' - 1$ last subtrees are minimal infeasible and disjoint subtrees. It might be the case that the top subtree of $\mathcal{P}$ is Π-feasible. We visit T from the left to the right, and bottom-up. While visiting a node v, if the subtree below v is Π-infeasible, we add it in our cut and we consider v as a $*$-node in the remaining part of T. At the end of the procedure we get the desired cut $\mathcal{P} = (t_i)_{1 \leq i \leq h'}$, ordered as T and having at most h' $*$-node.

Now we explain how to get a tree $T' \in L$. Since t_i has a root of degree at most r, let $t_i^1, \ldots, t_i^r$ be the r subtrees from the root of t_i (some of them might be empty), for $i = 1, \ldots, h'$. By construction of t_i, the subtrees $t_i^1, \ldots, t_i^r$ are Π-feasible. When $t_{h'}$ is feasible then $t_{h'}^1 = t_{h'}$ and others $t_{h'}^j$ are empty. Let F be the forest $(t_i^1, \ldots, t_i^r)_{i=1,\ldots,h'}$ of size at most rh', in the same order than T. To get F from T, we use only rh' moves and h' edge deletions. Moreover F has at most h' $*$-nodes. Applying Lemma 7, we get that $\mathsf{dist}(F, L) \leq r^m(1 + h' + 2r^{m+1}h')$ and since $\mathsf{dist}(T, F) \leq (r+1)h'$, we conclude that $\mathsf{dist}(T, L) \leq r^m(1 + 3r^{m+1}h')$, majoring $(r + 1) + r^m(1 + 2r^{m+1})$ by $r^m \times 3r^{m+1}$. But by assumption, $h' - 1 \leq \frac{1}{3r^{m+1}}(\frac{h}{r^m} - 1) - 1$, therefore $\mathsf{dist}(T, L) \leq h$. $\qquad\square$

Lemma 7 (Merging). *Let Π be an admissible set of connected components of $G(\mathcal{A})$. Let F be a Π-feasible forest of size h'_1 with at most h'_2 $*$-nodes. Then $\mathsf{dist}(F, L) \leq r^m(1 + h'_2 + 2r^m h'_1)$.*

Proof. First, we split each subtree t of F in simply C-feasible subtrees, for some C of Π. Fix such a $t \in F$. Let λ be a run of t such that all paths of t are Π-feasible. Fix a path σ of t and let C be the connected component of the root of t. We follow σ top-down until we leave C after a node v. Then we cut t just before leaving C, that is between v and its successors using r edge deletions and r moves. This leads to one simply C-feasible subtree from the root of t where the label of v is now $*$, and r new Π-feasible subtrees from the successors of v. We iterate the argument for the last r subtrees using the restrictions of the same run λ, so that the next paths of the last r subtrees will start with the next connected component. At the end of the process, at most $1 + r + \ldots + r^{m-1} \leq r^m$ C-feasible subtrees are generated from t using r^m edge deletions and r^m moves.

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

We now explain how to move and glue the remaining subtrees w_i in order to get a tree $T' \in L$. Let C_i be a connected component of Π such that t_i is simply C_i-feasible. We first move and glue linearly each subtrees t_i with the same C_i. At each $*$-node, a tree of size r^m is also inserted so that the resulting subtree is simply C_i-feasible and without any $*$-nodes. Then the remaining subtrees are connected to T_0 in order to get a tree $T' \in L$. We have done $(k-1)$ moves and $r^m \times (k+1) + r^m \times h'_2$ insertions and the total number of operations is less than: $(2r^m \times h'_1) + (r^m \times h'_1) + (r^m \times h'_1 + r^m \times (r^m \times h'_1 + 1) + r^m \times h'_2)$ which is less than $r^m + 2r^{2m} \times h'_1 + r^m \times h'_2$. $\qquad\square$

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

Proof. In this proof, we understand feasible as Π-feasible and we follow the structure of the proof of Lemma 4.

Let $t_1, \ldots, t_{h'}$ be some infeasible disjoint subtrees of T. Let a be a positive integer. For every integer $i \geq 1$, let $s_i = |\{t_j : \mathrm{depth}(t_j) = i\}|$. Since the root of a subtree may be shared a with the leaf of another subtree as a $*$-node, we have $|\{t_j : \mathrm{depth}(t_j) > a\}| \leq \frac{n}{a+1}$, 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 contains a 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-subtrees 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 $\frac{1}{r} \times \frac{h'}{2a}$. $\qquad\square$

5 Extension to unranked trees

An *unranked tree automaton* generalizes the transition function to $\delta : Q \times \Sigma \rightarrow 2^{Q^*}$ such that $\delta(q, a)$ is a regular language on Q. A run λ is generalized such that if u is a node with successors $v_1, ..., v_l$ 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 $\lambda(u) = q$.

We consider two approaches to generalize the **Tester for regular ranked tree language** to unranked regular trees. In a direct approach we are able to prove a Splitting lemma and a Merging lemma for any unranked tree automaton. The remaining main obstacle is the existence of an efficient random generator of subtrees for a corresponding Amplifying lemma.

Another possible approach consists to encode unranked trees T by binary trees $e(T)$ using a classical encoding to construct a binary automaton that accepts the encoded unranked trees accepted by the unranked automaton, and to apply the tester on binary trees. In case of XML files, assume they are given by their DOM (Document Object Model) structures. We can efficiently generate any random k-subtrees on the encoded tree from the DOM and simulate efficiently the **Tester for regular ranked tree language** on the encoded tree. There is a remaining obstacle consisting in lower bounding the distance of two encoded trees $\mathsf{dist}(e(T), e(T'))$ by $\mathsf{dist}(T, T')$. Even if it is clear that $\mathsf{dist}(e(T), e(T')) \leq 2\mathsf{dist}(T, T')$, the opposite inequality is rather technical.

References

1. N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20:451–476, 2000.
2. N. Alon, M. Krivelich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6), 2000.
3. M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.
4. 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.
5. H. Chockler and O. Kupferman. ω-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.
6. G. Cormode. *Sequence Distance Embeddings*. PhD thesis, University of Warwick, 2003.
7. 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.
8. R. Rubinfeld. On the robustness of functional equations. *SIAM Journal on Computing*, 28(6):1972–1997, 1999.
9. R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):23–32, 1996.
10. K. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 26:422–433, 1979.