Zapiski Nauchnykh Seminarov POMI Will be set by the publisher
Notes of Mathematical Seminars of St.Petersburg Department of V.A.Steklov Institute of Mathematics

,

TREE INCLUSIONS IN WINDOWS AND SLICES

Patrick Cégielski[1] and Irène Guessarian[2]

Abstract. P is an embedded subtree of T if P can be obtained by deleting some nodes from T: if a node v is deleted, all edges adjacent to v are also deleted, and outgoing edges are replaced by edges going from the parent of v (if it exists) to the children of v. Deciding whether P is an embedded subtree of T is known to be NP-complete.

Given two trees (a target T and a pattern P) and a natural number w, we address two problems: 1. counting the number of windows of T having height exactly w and containing pattern P as an embedded subtree, and 2. counting the number of slices of T having height exactly w and containing pattern P as an embedded subtree.

1991 Mathematics Subject Classification. 68Q25, 68W05.

For 60th birthday of Yuri Matiyasevich

1. Introduction

Given two trees, we study the following problems: can P be obtained from T by deleting nodes? Is P contained in a reasonably small "window" of T? If P is contained in a subtree of height w (a "window") of T, how many times can this occur? What is a w-window is clear for a text: it is a factor of length w. There are at least two possibilities for trees: a w-window is either a tree (of height w) or a forest (a slice of the target consisting of the nodes of height from d up to and including $d + w$), see Figure 1. We attack both possibilities in this paper.

Given two trees (a target T and a pattern P) and a natural number w, we address two problems: 1. counting the number of w-windows of T which contain pattern P as an embedded subtree, and 2. counting the number of w-slices of T which contain pattern P as an embedded subtree. These problems generalize in a natural way the subsequence problems for words: we proved in [BCGM01], that the problem of counting the number of w-windows of a text t containing a pattern

Keywords and phrases: Subtree inclusion, algorithm.

[1] LACL, Université Paris 12, Route forestière Hurtault, F-77300 Fontainebleau, France, e-mail: cegielski@univ-paris12.fr

[2] LIAFA, UMR 7089 and Université Paris 6, 2 Place Jussieu, 75254 Paris Cedex 5, France; send correspondence to e-mail: ig@liafa.jussieu.fr

p as a subsequence (i.e. letters of p appear in the window in the same order as in p but are not necessarily consecutive and may be interleaved with other letters) can be solved in time $O(n)$ where n is the size of t.

In [CGM08] we solved related problems: deciding whether a pattern is an embedded subtree of a target within a w-window, counting the number of windows of height at most w of the target containing the pattern as an embedded subtree, and counting the number of occurrences of the pattern as an embedded subtree of the target within a window of height at most w. The algorithms in the present paper are of course inspired by the algorithms in [CGM08], to which Yu. Matiyasevich contributed in a fundamental way; the present paper is thus a natural complement to [CGM08].

The problem of finding embedded occurrences of a pattern in a window (or a slice) of a tree is important in two areas: 1. retrieving information from structured documents [KM95], and 2. discovering frequent substructures in semi-structured data such as XML documents; indeed counting the number of occurrences of substructures is a mandatory first step in discovering frequent substructures [CMNK05]. Unfortunately, even deciding whether P is an embedded subtree of T is known to be NP-complete [KM95]; however, our algorithms are linear in the size of the target and exponential in the size of the pattern, hence their complexity is fixed-parameter linear for fixed patterns.

The present paper consists of three more sections: after the definitions, one section will present the algorithm counting the number of w-windows of height exactly w where the pattern can be embedded, and the next one will present the algorithm counting the number of w-slices of height exactly w where the pattern can be embedded.

2. Definitions

Let A be an alphabet.

Definition 2.1. (i) A **tree** T on A is a finite connected acyclic graph $T = \langle V, E, color\rangle$, where V is the set of nodes, E is the set of edges, and $color\colon V \mapsto A$ is a coloring function: each node (or vertex) is colored by a letter of A.

(ii) A **rooted tree** $T = \langle V, E, color, r\rangle$ is a tree where a node r has been distinguished and is called the **root** of the tree.

In a rooted tree, edges are naturally directed (from the root to the leaves): all nodes have in-degree one except for the root which has in-degree zero; if (u, v) is an edge oriented from u to v, u is said to be the *parent* of v and v is is said to be a *child* of u; each node has exactly one parent, except for the root which has none; a node can have any finite number of children and childless nodes are called *leaves*. Nodes have a *depth*: the depth of the root is 0, and if the depth of node v is n, then all its children have depth $n+1$. Two nodes having the same parent are called *siblings*. The transitive closure of the parent (resp. child) relation is called the *ancestor* (resp. *descendent*) relation. The *height* of a tree is the maximum of the depths of its nodes.

In the case of trees, the notions similar to subword and subsequence for words exist and are called *subtree* and *embedded subtree*. Formally:

Definition 2.2. Let $T = \langle V, E, color, r\rangle$ and $T' = \langle V', E', color', r'\rangle$ be rooted trees, such that:

1. $V' \subseteq V$ and $E' \subseteq E$,
2. the restriction to V' of the ancestor relation of V coincides with the ancestor relation of V'
3. the coloring of V' is preserved in T', i.e. $\forall v' \in V' \quad color'(v') = color(v')$

Then T' is said to be a **subtree** of T.

Moreover, if for each node v' from T' all its descendents in T are also its descendents in T', then T' is said to be a **bottom-up subtree** of T.

Intuitively, a bottom-up subtree of T can be obtained by taking a node v of T together with all of v's descendents and corresponding edges; a subtree of T can be obtained by taking a bottom-up subtree of T and pruning some edges together with the subtree below the pruned edge. The bottom-up subtree of T rooted at node v will be denoted by $T[v]$.

Definition 2.3. Let $T = \langle V, E, color, r\rangle$ and $T' = \langle V', E', color', r'\rangle$ be rooted trees; an **embedding** from T' into T is an injective mapping $\tau\colon V' \hookrightarrow V$, such that:

1. for every $v' \in V'$, $color(\tau(v')) = color'(v')$, i.e. τ preserves colors.
2. v'_1 is an ancestor of v'_2 in T' iff $\tau(v'_1)$ is an ancestor of $\tau(v'_2)$ in T, i.e. τ preserves the ancestor-descendent relationship.

T' is said to be an **embedded subtree** of T if there exists an embedding from T' into T.

Intuitively, an embedded subtree of T is obtained by deleting some nodes from T and gluing together the remaining edges in a way preserving the ancestor-descendent relationship of T.

Definition 2.4. A w**-window** of $T = \langle V, E, color, r\rangle$ is a subtree $W = \langle V', E', color', r'\rangle$ of T such that V' contains all the descendants of r' from depth $depth(r')$ down to depth $w + depth(r')$.

P is an **embedded subtree of** T **within a** w**-window** if there is an embedding from P into T and moreover the image of P is contained in a w-window of T.

For $w \in \mathbb{N}^*$, a w**-window** of T has height **exactly** w.

Definition 2.5. Let T be a tree and w an positive integer. A w**-slice** of T is the forest consisting of all w'-windows, for $w' \leq w$, whose roots are at the same depth in T.

The slice of heigth w at depth k contains all the nodes from depth k to depth $k + w$ included.

Example 1. *In Figure 1, T', T'', T''' are respectively a bottom-up subtree, a subtree, and an embedded subtree of T. T''' is an embedded subtree of T within a 2-window. S is a 1-slice of T and W is a 1-window of T.*

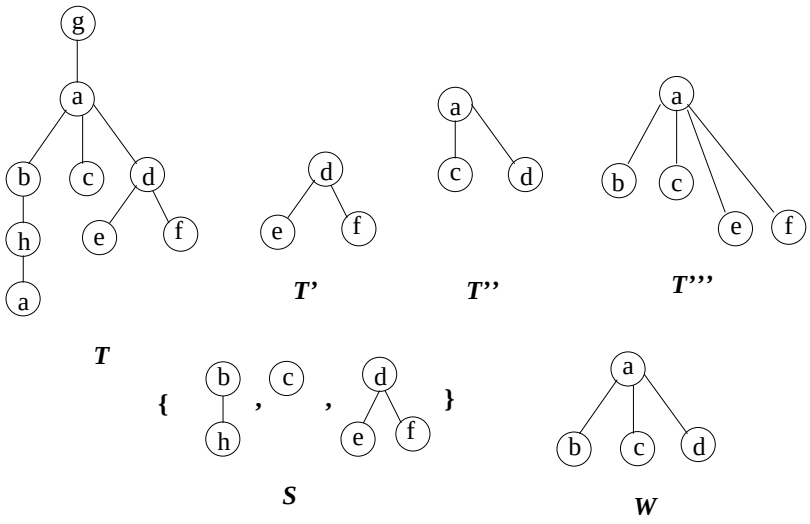

FIGURE 1. A tree T with bottom-up subtree T', subtree T'', embedded subtree T''', 1-slice S and 1-window W.

3. Counting w-windows

The problem: Given a (big) tree T (called the *target*) and a (small) tree P (called the *pattern*), we want to count the number of w-windows of T where P can be embedded.

We will consider only the case of pattern trees of height at least one: because patterns of height 0 consist of a single node, their search is trivial.

In a first step we will preprocess the pattern, then in a second step we will process the target.

First step: preprocessing the pattern Without loss of generality we may assume that the nodes of P are labelled: each node has a *unique* label from $\{1, \ldots, p\}$, where p is the number of nodes of P, see Figure 2. This yields a labelling of the bottom-up subtrees of P: bottom-up subtree rooted at node v has the same label as node v. A bottom-up subtree of P rooted at node v is represented either by the label of v or in the form $P[v]$: the bottom-up subtree rooted at node v having label j, will thus be denoted by j or $P[v]$ according to the context.

Second step: processing the target To solve our problem, we will adorn the nodes of T with *s-configurations* which will be sets of forests of stamped bottom-up subtrees of P. The intuitive meaning of stamp $n \in \mathbb{N}$ in stamped subtree (t, n) will be that subtree t of P can be embedded in a subtree of T of height n located below the current position. Intuitively, each forest of the configuration at node v represents a set of subtrees of P which can be embedded in $T[v]$ *simultaneously*, i.e. in such a way that the images of different trees wouldn't intersect.

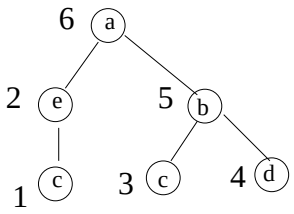

FIGURE 2. A pattern P and a postorder labelling of its bottom-up subtrees.

Definition 3.1. A **stamped subtree** is an ordered pair (t, n) where t is a bottom-up subtree of P, and n is a positive integer in $\mathbb{N}$, called the *depth-stamp*.

A set F of stamped trees is called an **s-forest**, and it is said to be a **min-s-forest** if the following two conditions hold

- there are no (t, n) and $(t, n') \in F$ such that $n' < n$, i.e. all its subtrees occur at the least possible depth in T,
- there are no (t, n) and $(t', n') \in F$ such that $n' \geq n$ and t' is a proper[1] bottom-up subtree of t.

An s-forest F is said to **dominate** s-forest F' if for every (t', n') from F' there is a (t, n) from F such that

- t' is a bottom-up subtree of t, and
- $n' \geq n$.

An **s-configuration** is a set $C = \{F_1, F_2, \ldots, F_k\}$, where each F_i is a min-s-forest, and if $i \neq j$ then F_i does not dominate F_j.

In a min-s-forest, only minimal stamped subtrees appear: for instance, considering the pattern of Figure 3 and identifying a subtree of P with the label of its root, {(1,1),(3,0),(4,1)} is a min-s-forest, while neither {(1,1),(4,1),(4,2)} nor {(1,1),(4,1),(3,1)} are min-s-forests. Forest F dominates forest F' if, intuitively, all the possibilities of embeddings contained in F' are subsumed by those contained in F. For example, considering again the pattern of Figure 3, forest {(2,1),(4,1)} dominates forests {(2,1),(3,1)} and {(2,1),(4,2)}, but forest {(2,1),(4,1)} does not dominate forest {(2,0),(4,2)}.

Definition 3.2. If an s-forest F is not a min-s-forest, we can associate with F a **reduced** forest, the min-s-forest $D = \mathrm{red}(F)$ obtained by the following algorithm:

(1) remove all stamped subtrees $(t, n) \in F$ such that there is a $(t, n') \in F$ with $n' < n$: then all subtrees of P belonging to F are affected with the minimal possible depth-stamp.
(2) remove all stamped subtrees $(t', n') \in F$ such that there is $(t, n) \in F$ such that $n' \geq n$ and t' is a proper bottom-up subtree of t.

[1] t' is said to be a proper (bottom-up) subtree of t if (i) t' is a (bottom-up) subtree of t, and (ii) $t' \neq t$.

Notice that we remove stamped subtrees having the larger stamp n because only the subtrees having the minimal possible n will give us the best possible embeddings.

Definition 3.3. A subtree T' of T is said to be a **minimal subtree** of T containing P iff P is an embedded subtree of T', but there is no proper subtree T'' of T' such that P is an embedded subtree of T''.

For instance, considering Figure 3 again, the subtree of T rooted at b is a minimal subtree containing subtree 2 of the pattern, but is not a minimal subtree containing subtree 1 of the pattern.

Definition 3.4. The **union** of two s-configurations $C = \{F_1, F_2, \ldots, F_k\}$ and $C' = \{F'_1, F'_2, \ldots, F'_{k'}\}$ is s-configuration $D = C \otimes_{\mathrm{s}} C'$ obtained as follows:

(1) let $D = \left\{\mathrm{red}(F_i \cup F'_{i'}) \mid i \in \{1, ..., k\}, i' \in \{1, ..., k'\}\right\} = \left\{F''_1, F''_2, \ldots, F''_{k''}\right\}$
(2) if F''_i is dominated by F''_j remove F''_i from D.

Intuitively, (1) ensures that each forest of $C \otimes_{\mathrm{s}} C'$ is a min-s-forest, namely we keep only of the "best" (i.e. minimal) possible stamped subtrees, and (2) ensures that we keep only non redundant min-s-forests, i.e. if all information from forest F' is already present in forest F, we discard forest F'.

Idea of the algorithm It is easy to see [BCGM01, K92] that a window of height w of T contains P as an embedded subtree iff it contains a minimal subtree of T containing P; therefore, it is enough to count the number of w-windows of T containing a minimal subtree containing P.

For each node v of T we will compute an *s-configuration*, which will be a set of min-s-forests of stamped bottom-up subtrees of P. The idea of the algorithm is to increment the number of w-windows each time a stamped tree (P, d) with $d \leq w$ is found in one of the forests of the configuration. However we must be careful not to count windows of height less than w which might occur near the leaves of T: to this end, we will first tag each node v of T with an integer indicating whether the height $T(v)$ is at least w. To this end it suffices to let, for each node v of T, $tag(v) := \min(w, height(T[v]))$.

Algorithm1
INPUT.- Two trees, T and P, and an integer w.
OUTPUT.- The number N of w-windows of T containing P as embedded subtree.

Let $r :=$ the label of the root of P; $N := 0$; *//initialization*
`FORALL` nodes of T visited bottom-up `DO`

(1) `IF` node v is leaf of T, `THEN` the configuration of v is the set of singletons $\{(i, 0)\}$ where i is the label of a leaf v' of P such that $color(v') = color(v)$, *//if no leaf of P has the same color as v the configuration of v is the empty set.*
(2) `IF` node v is an internal node colored a, with children $v_1, \ldots, v_n$, whose respective configurations are C_{v_i}, $i = 1, \ldots, n$, `THEN DO`
   (a) `FOR` $i = 1, \ldots, n$, `DO`

$C'_{v_i} := \{\{(l, d+1) | (l, d) \in F_j$ and $h + d + 1 \leq w$ where h is the depth of l in P $\} \mid F_j \in C_{v_i}\}$//*subtree* $(l, d+1)$ *cannot contribute to any embedding in a* w*-window if* $h + d + 1 > w$ ENDDO

(b) $\Delta := D := C'_{v_1} \otimes_s C'_{v_2} \otimes_s \cdots \otimes_s C'_{v_n}$;

(c) FORALL nodes w of P colored a and with label j DO //w *has same color as* v

- IF node w is not a leaf of P,
  //w *is an internal node of* P *labelled* j
- THEN Let $j_1, ..., j_p$ be the children of j in P,
  FOR $\{(j_1, d_1), ..., (j_p, d_p)\} \subseteq F_i \in D$
  $\Delta := \Delta \cup \{\{(j, \max\{d_1, ..., d_p\})\}\}$;
- ELSE //w *is a leaf labelled* j *and colored* a
  $\Delta := \Delta \cup \{\{(j, 0)\}\}$;

ENDIF
ENDDO

(d) Reduce the forests in Δ and remove from Δ all dominated forests

(e) Take the resulting configuration Δ for C_v

(f) IF $tag(v) = w$ AND there is $F \in C_v$ such that $(r, d) \in F$
THEN DO $N := N + 1$;
output "P is an embedded subtree of T within a window of size exactly w at node v".
ENDDO
ENDIF

ENDDO
ENDIF

ENDDO

Notice that in step (c) of our algorithm, the loop
FOR $\{(j_1, d_1), ..., (j_p, d_p)\} \subseteq F_i \in D$
$\Delta := \Delta \cup \{\{(j, \max\{d_1, ..., d_p\})\}\}$;
ensures that, for each "good" subset of each F_i, we add in the configuration a forest consisting of a **single subtree** P' of P: the choice of subtree P' excludes all other subtrees of P possible at that stage (because node v of T can be used to match only one subtree of P with root having the same label as v). Consider P, T as in Figure 3: the FOR loop in step (c) rightly prevents us from saying that P is embedded in T (by excluding the simultaneous embedding of subtrees 2 and 4 of P in the subtree with root colored b of T).

See Figures 3 and 4 for illustrations of algorithm1 with $w = 2$. The tags are omitted for clarity in the Figures.

It is easy to extend Algorithm1 for searching simultaneously a set $P_1, \ldots, P_f$ of patterns: to each node v of T we attach an f-tuple of configurations $C_v^1, \ldots, C_v^f$, one configuration for each pattern (the space complexity will be multiplied by f). At each step of Algorithm1, the f configurations are treated in parallel. Let $r_1, \ldots, r_f$ be the labels of the roots of $P_1, \ldots, P_f$; it then suffices to change step

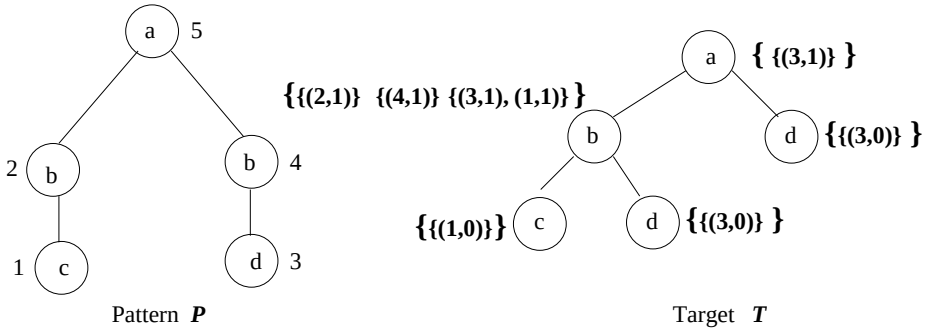

FIGURE 3. Pattern P is not embedded in a 2-window of target T.

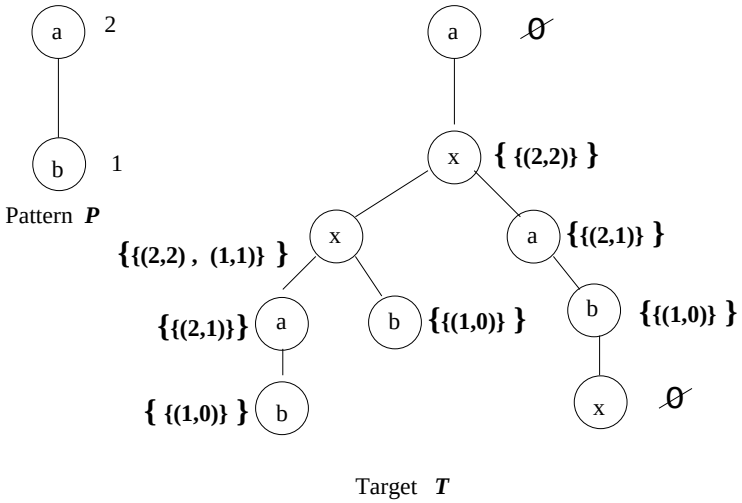

FIGURE 4. Pattern P occurs in three 2-windows and two 2-slices in target T.

(f) of Algorithm1 by substituting "for $i = 1, \ldots, f$, there is $F^i \in C_v^i$ such that $(r_i, d_i) \in F^i$" for "there is $F \in C_v$ such that $(r, d) \in F$".

Complexity: the number of bottom-up subtrees of P is bounded by p where p is the size (number of nodes) of P; the number of subtrees in a min-s-forest is bounded by p (because of the condition that there are no (t, n) and $(t, n') \in F$ such that $n' < n$, each bottom-up subtree of P can occur at most once in a forest). Let $h(P)$ be the height of P. The number of min-s-forests is bounded by $K = (w - h(P) + 2)^p$: indeed if n is the depth-stamp of subtree $P[v]$ whose root is at depth $h(P) - j$ in P, then $j \leq n \leq w - (h(P) - j)$, hence n can take at most $w - h(P) + 1$ values (this is a coarse bound because the depth-stamps are submitted to additional restrictions); if we add an additional stamp-value -1, with

$(t, -1)$ meaning that subtree t is not embeddable hence is not in the forest, we obtain the bound $K = (w - h(P) + 2)^p$ for the number of min-s-forests. Assume that the maximum out-degree of the nodes of T is M, then the complexity of the various stages in step 2. of algorithm1 can be decomposed as follows:

(a) MK
(b) $M(K^2 + Kp^2) + K^2p^2$ (we perform M times a $\otimes$ between two configurations, each one consisting of at most K forests, which takes MK^2 steps, we reduce each time the forests, which takes MKp^2 steps, and we can remove only once at the end dominated forests in K^2p^2 steps)
(c) p^2K
(d) K^4 (Δ consists of at most K forests, and for each pair of forests, we can check in time K^2 whether one of them dominates the other)
(f) Kp

Hence, assuming that $M \leq K^2$, the execution time of step (d) will dominate the execution time of algorithm1, whose complexity will be $O(|T|(2(w-h(P)+1))^{4p})$. The bound is thus linear in the size of T and exponential in the size of P. Hence our counting problem is fixed-parameter tractable [FG06], and even better, fixed-parameter linear.

A drastic reduction of this complexity is unlikely because the simpler problem of deciding whether P can be embedded in T has been proved to be NP-complete [K92, KM95]. However, Algorithm1 itself can be improved in practice by reducing the number of configurations. The idea (due to Y. Matiyasevich) is as follows: node v from the target and node v' from the pattern are **upward compatible**, if the path from v' to the root of P can be embedded into the path from v to the root of T. We can preprocess target T in order to precompute for each node v in T the set $c(v)$ of all nodes of P which are upward compatible with it and then demand in step 1. that v' should be upward compatible with v. This could considerably reduce the number of configurations to deal with.

4. Counting w-slices

The problem: Given a target tree T and a pattern tree P, to count the number of w-slices of T where P can be embedded. We will again consider only the case of pattern trees of height at least one (the seach for patterns of height 0 being trivial).

Idea.- The idea of the algorithm is a variant of the previous algorithm but we now traverse the target bottom–up and level-wise: first the nodes (necessarily leaves) of depth n (the height of the target), then the nodes of depth $n-1$ and so on. We use two integer variables: a counter N and a potentiometer po, both initialized to zero. Intuitively, the value of the potentiometer gives the number of w-slices above the current one where we are sure that the pattern will be embedded. When we meet an occurrence of the pattern (r, d) (as in the previous algorithm), we update the potentiometer to $\max(po, w - d)$ because we are sure the $w - d$ following slices will contain an occurrence of the pattern. After treatment of every node of a given

depth h, if $n-h > w$ (to be sure the height of the slice is w) and the potentiometer is not zero, we increment the counter and decrement the potentiometer. When we arrive to the root, N contains the number of w-slices in which an occurrence of the pattern occurs.

Algorithm2

INPUT.- Two trees, T and P, and an integer w.
OUTPUT.- The number N of w-slices of T containing P as embedded subtree.
//initializations
Let $r :=$ the label of the root of P;
Let $n :=$ the height of T;
Let $N := 0$;
Let $po := 0$;
// Treatment
`FOR` $h := n$ `TO` 0
`DO`
`FORALL` nodes of T of depth h in T
`DO`
1. `IF` node v is a leaf of T `THEN`
the configuration of v is the set of singletons $\{(i,0)\}$
where i is the label of a leaf v' of P such that
a is the color of both v and v'
//if no leaf of P has the same color as v
//the configuration of v is the empty set.
2. `IF` node v is an internal node colored a,
with children $v_1, \ldots, v_n$
whose respective configurations are C_{v_i}, $i = 1, \ldots, n$ `THEN`
`DO`
(a) `FOR` $i = 1, \ldots, n$
`DO`
$C'_{v_i} := \big\{\{(l, d+1) \mid (l,d) \in F_j$ and $h + d + 1 \leq w$
where h is the depth of l in P $\} \mid F_j \in C_{v_i}\big\}$
`ENDDO`
// subtree $(l, d+1)$ cannot contribute to any
// embedding in a w-window if $h + d + 1 > w$
(b) $\Delta := D := C'_{v_1} \otimes_{\mathrm{s}} C'_{v_2} \otimes_{\mathrm{s}} \cdots \otimes_{\mathrm{s}} C'_{v_n}$;
(c) `FORALL` nodes w of P colored a and with label j
`DO` *// w has same color as v*
`IF` node w is not a leaf of P
//w is an internal node of P labelled j
`THEN` Let $j_1, ..., j_p$ be the children of j in P
`FOR` $\{(j_1, d_1), ..., (j_p, d_p)\} \subseteq F_i \in D$
$\Delta := \Delta \cup \{\{(j, \max\{d_1, ..., d_p\})\}\}$;
`ELSE` *//w is a leaf labelled j and colored a*

```
                    Δ := Δ ∪ {{(j, 0)}} ;
                ENDIF
                ENDDO
            (d) Reduce the forests in Δ
                and remove from Δ all dominated forests
            (e) Take the resulting configuration Δ for C_v
            (f) FORALL F ∈ C_v such that (r, d) ∈ F
                DO
                po := max(po, w − d) ;
                output "P is an embedded subtree of T
                      within a slice of size exactly w at node v".
                F := F \ {(r, d)}
                ENDDO
            ENDDO
        ENDIF
  ENDDO
IF n − h > w AND po ≠ 0 THEN
  DO
  N := N + 1;
  po := po − 1;
  ENDDO
ENDIF
ENDDO
```

See Figure 4 for an illustration of algorithm2 with $w = 2$. The potentiometers are omitted for clarity in Figure 4. As in Algorithm1, it is easy to extend algorithm2 to the simultaneous search of several patterns.

The complexity of algorithm2 is similar to the complexity of algorithm1.

5. Conclusion

Given two trees (a target T and a pattern P) and a natural number w, we gave algorithms solving the following two problems: 1. counting the number of height exactly w windows of T which contain pattern P as an embedded subtree, and 2. counting the number of height exactly w slices of T which contain pattern P as an embedded subtree.

For target tress with bounded branching width, our algorithms are linear in the size of the target and exponential in the size of the pattern, i.e. their complexity is both fixed-parameter linear [FG06], and CONSTANT-DELAY$_{lin}$ [DG06]. Of course, our algorithms also apply in the case of target trees with unbounded branching width, but in that latter case we conjecture they are not in CONSTANT-DELAY$_{lin}$.

Acknowledgment: We are grateful to Arnaud Durand for his reading, discussions and helpful comments.

References

[BCGM01] L. Boasson, P. Cegielski, I. Guessarian, Yu. Matiyasevich, Window Accumulated Subsequence Matching is linear, Annals of Pure and Applied Logic Vol. 113 (2001), pp. 59-80.

[CGM08] P. Cegielski, I. Guessarian, Yu. Matiyasevich, Tree inclusion problems, submitted (2007).

[CMNK05] Y. Chi, R. Muntz, S. Nijssen, J. Kok, Frequent subtree mining – an overview, Fundamenta Informaticae, Volume 66 (2005), pp. 161–198.

[DG06] A. Durand, E.Grandjean, First-order queries on structures of bounded degree are computable with constant delay, To appear in ACM Transactions on Computational Logic, 2006.

[FG06] J. Flum, M. Grohe, Parameterized Complexity Theory, Springer-Verlag, Berlin (2006).

[K92] P. Kilpelainen, Tree matching problems with applications to structured text databases, PhD Thesis, Helsinki (1992), `http://thesis.helsinki.fi/julkaisut/mat/tieto/vk/kilpelainen/`

[KM95] P. Kilpelainen, H. Mannila, Ordered and Unordered Tree Inclusion, SIAM Journal on Computing , Volume 24 ,B Issue 2 B (April 1995), pp. 340 - 356.

Communicated by (The editor will be set by the publisher).
today.