**Theoretical Informatics and Applications** Will be set by the publisher
Informatique Théorique et Applications

, ,

# TREE INCLUSION PROBLEMS *

Patrick Cégielski[1], Irène Guessarian[2] and
Yuri Matiyasevich[3]

**Abstract**. Given two trees (a target $T$ and a pattern $P$) and a natural number $w$, *window embedded subtree problems* consist in deciding whether $P$ occurs as an embedded subtree of $T$ and/or finding the number of size (at most) $w$ windows of $T$ which contain pattern $P$ as an embedded subtree.

$P$ 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.

Our algorithms run in time $O(|T|2^{2^{|P|}})$ where $|T|$ (resp. $|P|$) is the size of $T$ (resp. $P$).

**1991 Mathematics Subject Classification.** 68Q25, 68W05.

For 60th birthday of Serge Grigorieff

## 1. Introduction

Given two trees, we study the following problems: can $P$ be obtained from $T$ by deleting nodes? if this holds, is $P$ contained in a reasonably small (i.e. of a small height) subtree of $T$? If $P$ is contained in a subtree of height $w$ (a "window") of $T$, how many times can this occur?

These problem generalize in a natural way the subsequence problems for words: we proved in [BCGM01], that the problem of counting the number of $w$-windows

*Keywords and phrases:* Subtree inclusion, algorithm.

* *Support from the Council for Grants of the President of the Russian Federation under grant NSh-8464.2006.1 is acknowledged by the third author.*

[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`

[3] Steklov Institute of Mathematics, Fontanka 27, St. Petersburg, 191023, Russia. e-mail: `yumat@pdmi.ras.ru`

of a text $t$ containing a pattern $p$ 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 $t$. The generalization to trees can be stated as follows: $P$ is an embedded subtree of $T$ if $P$ can be obtained by deleting some nodes from $T$ (if a node $v$ is deleted, the ingoing edge to $v$ (if it exists) is also deleted, and outgoing edges are replaced by edges going from the parent of $v$ (if it exists) to the children of $v$). $P$ is an embedded subtree of $T$ within a $w$-window if $P$ is an embedded subtree of $W$, and $W$ is a subtree of $T$ of height $w$.

We cannot hope to reduce (in a simple and succinct way) the problem of finding whether $P$ is an embedded subtree of $T$ within a $w$-window to the subsequence problem for words by encoding $T$ and $P$ by words $t$ and $p$ respectively, and then solving some subsequence problem for $t$ and $p$: it is known [KM95] that even the simpler problem of deciding whether $P$ is an embedded subtree of $T$ is NP-complete, hence the length of $t$ and/or $p$ would probably be exponential in the size of $T$ and/or $P$.

The problem of finding embedded occurrences of a pattern in a window of a tree is important in two areas extensively studied recently:

1. retrieving information from structured documents [KM95] such as dictionaries: via a pattern embedding the user can specify the structure and content of the parts of the document she/he is interested in.
2. discovering frequent substructures in semi-structured data; most semi-structured data are modeled by colored labelled trees (e.g. itemsets in relational databases, chemical compounds, XML documents), and mining such structures is naturally done via finding tree embeddings.

The paper is organized as follows: in section 2 we define notations and problems, in section 3 we describe a new algorithm to decide whether a pattern in embedded in a target (not interesting per se, but for the generalisations given in the following section), and section 4 presents our main contribution, namely, (i) determining (and counting) the $w$-windows containing a pattern as an embedded subtree, and (ii) determining (and counting) the embeddings of a pattern within a $w$-window of text.

## 2. Notations

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 $color : V \mapsto A$ 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 the *parent* of $v$ and $v$ is 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 descendants in $T$ are also its descendants in $T'$, then $T'$ is said to be a **bottom-up subtree** of $T$.

$T'$ is said to be a **proper** (bottom-up) subtree of $T$ if $T'$ is a (bottom-up) subtree of $T$ and $T' \neq T$.

Intuitively, a bottom-up subtree of $T$ can be obtained by taking a node $v$ of $T$ together with all of $v$'s descendants 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. $\tau$ 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. $\tau$ preserves the ancestor-descendant 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-descendant relationship of $T$.

**Definition 2.4.** A **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 $depth(r') + height(W)$.

For $w \in \mathbb{N}^*$, a $w$**-window** of $T$ is a window of $T$ of height at most $w$. $P$ 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 $T$.

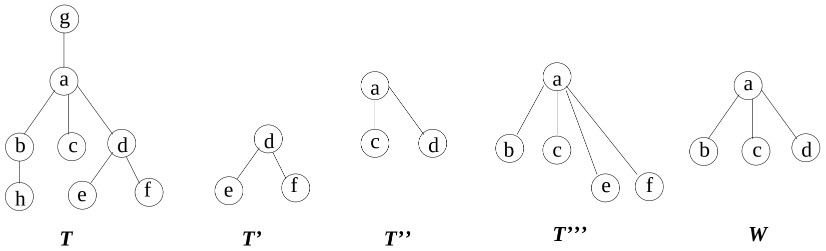


FIGURE 1. A tree $T$ with bottom-up subtree $T'$, subtree $T''$, embedded subtree $T'''$, and 1-window $W$.

**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. $W$ is a 1-window of $T$.*

THE PROBLEMS

- **Problem 1.** Given two trees, *target* tree $T$ and *pattern* tree $P$, to decide whether $P$ is an embedded subtree of $T$.
- **Problem 2.** Given two trees, *target* $T$ and *pattern* $P$, to decide whether $P$ is an embedded subtree of $T$ within a $w$-window. Subsidiarily, to count the number of windows of height at most $w$ of $T$ containing $P$ as an embedded subtree.
- **Problem 3.** Given two trees, *target* $T$ and *pattern* $P$, to count the number of occurrences of $P$ as an embedded subtree of $T$ within a $w$-window.

RELATED RESULTS

Different versions of the first problem have been considered. Papers [K92,KM95] shows that problem 1 is NP-complete, but can be solved in time $O(ptk2^{2k})$, where $p = |P|$ (resp. $t = |T|$) is the number of nodes (size) of $P$ (resp. $T$), **if** the out-degrees of the nodes of $P$ are bounded by $k$.

## 3. EMBEDDED SUBTREE SEARCH

We study Problem 1: given target tree $T$ and pattern tree $P$, to decide whether $P$ is an embedded subtree of $T$.

### 3.1. NOTATIONS

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 $P$: the bottom-up subtree rooted at node $v$ has the same label as node $v$. A bottom-up subtree of

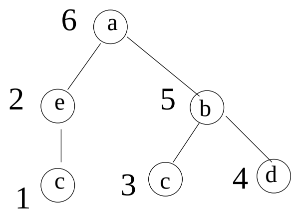


FIGURE 2. A pattern $P$ and a postorder labelling of its bottom-up subtrees.

$P$ 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.

**Definition 3.1.** A **forest** of $P$ is a set of bottom-up subtrees of $P$. A forest will be denoted by the set of labels of its roots. Forest $F$ is said to be a **max-forest** if all its trees are incomparable, i.e. there are no $t, t' \in F$ such that $t'$ is a proper bottom-up subtree of $t$.

We say that forest $F$ **dominates** forest $F'$ (different from $F$ itself) if every tree from $F'$ is a bottom-up subtree of some tree from $F$.

### 3.2. Idea

Let $T$ be a (big) tree (called the *target*) and let $P$ be a (small) tree (called the *pattern*). For each node $v$ of $T$ we will compute a *configuration*, which will be a set of max-forests of bottom-up subtrees of $P$.

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.

**Definition 3.2.** A **configuration** is a set $C = \{F_1, F_2, \ldots, F_k\}$, where each $F_i$ is a max-forest of $P$, and if $i \neq j$ then $F_i$ does not dominate $F_j$.

**Definition 3.3.** The union of two configurations $C = \{F_1, F_2, \ldots, F_k\}$ and $C' = \{F'_1, F'_2, \ldots, F'_{k'}\}$ is configuration $D$, denoted by $D = C \otimes C'$ and obtained as follows:

(1) let $D' = \left\{F_i \cup F'_{i'} | i \in \{1, ..., k\}, i' \in \{1, ..., k'\}\right\}$
(2) we pass from $D'$ to $D''$ by removing from each $F'_j = F_i \cup F'_{i'}$ which is not a max-forest all labels of subtrees of $P$ which are subtrees of a tree whose label is already present in $F'_j$
(3) we pass from $D''$ to $D$ by removing each $F''_j$ which is dominated by some $F''_i$.

Note that:

- In (2), we obtain a forest $F''_j$ such that all subtrees of $P$ belonging $F''_j$ are maximal in $F''_j$ (all subtrees of $F''_j$ are incomparable), i.e. $F''_j$ is a max-forest.

- The resulting $D = \{F_i | i = 1, \dots, l\}$ is a set of max-forests of $P$, such that if $i \neq j$ then $F_i$ does not dominate $F_j$, hence it is a configuration.

### 3.3. Algorithm

We traverse $T$ bottom-up (from leaves to root, or in postorder): to scan a node $v$ of $T$ we must first have scanned all its children. With each node $v$ of $T$ we will associate a configuration $C_v$ such that for every forest $F$ from $C_v$ all trees from $F$ can be simultaneously embedded into $T[v]$. This leads to the following algorithm. (See Figure 3**(1)** for an example.)

**Algorithm1**

Let $r :=$ the label of the root of $P$; //initialization
`FORALL` nodes $v$ of $T$ visited in bottom-up order `DO`

(1) `IF` node $v$ is a leaf of $T$, its configuration is the set of singletons $\{i\}$ where $i$ is the label of a leaf $v'$ of $P$ such that $color(v) = color(v') = a$.
//If no leaf of $P$ is colored $a$, the configuration of $v$ is the empty set.
`ENDIF`

(2) `IF` node $v$ is an internal node colored $a$, with children $v_1, \dots, v_n$, `THEN DO`
 (a) $\Delta := D := C_{v_1} \otimes C_{v_2} \otimes \cdots \otimes C_{v_n}$;
 (b) `FORALL` nodes $w$ of $P$ colored $a$ and labelled $j$ `DO` //$w$ has the same color as $v$
  Let $j_1, ..., j_p$ be the children (if they exists) of $j$ in $P$
  `IF` there is an $F_i \in D$ such that $\{j_1, ..., j_p\} \subseteq F_i$ // true if there are no children
  `THEN`
   `IF` $w = r$ `THEN DO`
    output "$P$ is an embedded subtree of $T$";
    `STOP ENDDO ENDIF`
   let $\Delta^*$ be the result of deleting $j_1, ..., j_p$ from all forests in $\Delta$;
   $\Delta := \Delta^* \cup \{\{j\}\}$
  `ENDIF`
  `ENDDO`
 (c) Remove from $\Delta$ all dominated forests and take the resulting configuration for $C_v$
`ENDDO`
`ENDIF`

`ENDDO`
output "$P$ isn't an embedded subtree of $T$"

**Comment:** when computing $\Delta^*$ we delete every occurrence of $j_1, ..., j_p$ in all forests because they are used only to obtain $j$; at a later stage (i) either we will choose $j$ but it already appears, or (ii) we will choose another subtree, and in that latter case we do not need $j_1, ..., j_p$.

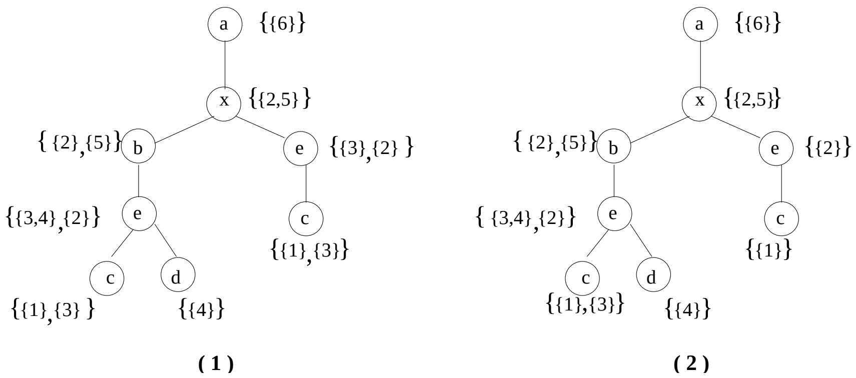


FIGURE 3. A target $T$ where the pattern of Figure 2 is embedded.

**Complexity:** the number of bottom-up subtrees of $P$ is bounded by $p$ where $p$ is the size of $P$, the number of forests is bounded by $2^p$, hence the number of configurations is at most $O(2^{2^p})$.

### 3.4. Improvements

#### Improvement 1

Algorithm1 can be improved in practice by reducing the number of configurations. Let us say 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. This is trivial for the root of $T$. Passing from a node $v$ in $T$ to its child $w$ we just add to $c(v)$ each node $u$ in $P$ such that: 1) $u$ has the same color as $w$, and 2) the parent of $u$ is in $c(v)$.

The algorithm computing the set $c(w)$ of nodes of $P$ upward compatible with $w$ is as follows:

Let $r :=$ the label of the root of $P$; //initialization
`FORALL` nodes $w$ of $T$ visited top-down `DO`
 `IF` node $w$ is the root of $T$,

- `THEN` $c(w) = \begin{cases} \{r\} & \text{if } color(w) = color(r), \\ \emptyset & \text{otherwise.} \end{cases}$
- `ELSE DO` //$w$ has parent $v$ whose set of upward compatible nodes is $c(v)$
  $c(w) = c(v) \cup \{u | u \text{ is a node of P, and } parent(u) \in c(v), \text{ and } color(u) = color(w)\}$ `ENDDO`

 `ENDIF`
`ENDDO`

The complexity of this preprocessing is $O(tp)$. Then in step 1 of algorithm1 we can demand that $v'$ should be upward compatible with $v$. This could considerably reduce the number of configurations to deal with. For instance in Figure 3 the set of nodes upward compatible with the rightmost leaf of $T$ is $\{1, 2, 6\}$, which reduces by half the set of configurations on the rightmost path of $T$, see Figure 3**(2)**.

Improvement 2

The idea of upward compatibility can be further developed as follows. Suppose that $\tau : V' \to V$ is an embedding of $P = \langle V', E', \mathit{color}', r'\rangle$ into $T = \langle V, E, \mathit{color}, r\rangle$. We can consider an *inverse embedding* $\sigma$ which is a partial map from $V$ onto $\mathcal{P}(E') \cup V'$ defined as follows:

- if $v = \tau(v')$ then $\sigma(v) = v'$;
- if $v'_1$ is the parent of $v'_2$ in $P$, then for every internal node $v$ on the path from $\tau(v'_1)$ to $\tau(v'_2)$ the value of $\sigma(v)$ is equal to the edge between $v'_1$ and $v'_2$;
- for all other nodes of $T$ the value of $\sigma$ is left undefined.

For every $v$ from $T$ we can consider the set $S(v)$ of all possible values of $\sigma(v)$, for all possible embeddings $\tau$. It is easy to check that these sets satisfy the following conditions:

[A1] if $S(v)$ contains some $v'$ from $V'$ then $\mathit{color}(v) = \mathit{color}'(v')$;

[A2] if $S(v)$ contains some $v'$ from $V'$ then
  - if $v'$ has the parent $v'_1$ in $P$, then $v$ has the parent $v_1$ in $T$ and $S(v_1)$ contains either $v'_1$ or the the edge between $v'_1$ and $v'$;
  - for every child $v'_2$ of $v'$ in $P$, the node $v$ has a child $v_2$ such that the set $S(v_2)$ contains either $v'_2$ or the edge between $v'$ and $v'_2$

[A3] if $S(v)$ contains the edge between some $v'_2$ and its parent $v'_1$ in $P$ then
  - $v$ has the parent $v_1$ in $T$ and $S(v_1)$ contains either $v'_1$ or the edge between $v'_1$ and $v'_2$
  - $v$ has a child $v_2$ such that $S(v_2)$ contains either $v'_2$ or the edge between $v'_1$ and $v'_2$

We cannot easily calculate "true" sets $S(v)$ so instead of this we calculate (and dynamically maintain during the entire work of the algorithms) some sets $\tilde{S}(v)$ such that $S(v) \subseteq \tilde{S}(v)$. Initially, we put

$$\tilde{S}(v) := \{v' \mid v' \in V' \text{ and } \mathit{color}'(v') = \mathit{color}(v)\} \cup E' \tag{1}$$

and then diminish these sets as long as either [A2] or [A3] is violated.

As soon as we calculated configuration $C_v$ for some node $v$, we try to further trim $\tilde{S}(v)$ in the following way. The set $\tilde{S}(v)$ can be represented as the union $\tilde{S}(v) = \tilde{S}_{\mathrm{V}}(v) \cup \tilde{S}_{\mathrm{E}}(v)$ where $\tilde{S}_{\mathrm{V}}(v) = \tilde{S}(v) \cap V(P)$ and $\tilde{S}_{\mathrm{E}}(v) = \tilde{S}(v) \cap E(P)$. Now we can put

$$\tilde{S}(v) = \big(\tilde{S}_{\mathrm{V}}(v) \cap (\cup_{w \in F \in C_v} V(P[w]))\big) \cup \tilde{S}_{\mathrm{E}}(v). \tag{2}$$

(and then check conditions [A2] and [A3], of course).

In its turn, the calculation of $\tilde{S}(v)$ allows us to add on step (b) additional restriction on the choice of $j$, namely, we can demand that $j \in \tilde{S}(v)$.

Calculation of the $\tilde{S}(v)$'s can be done with only linear slow-down of the algorithm. This can be implemented as follows. Each of the two conditions in [A2] and the two conditions in [A3] can be expressed by a logical formula of the form

$$u' \in S(v) \Rightarrow u'_1 \in S(v_1) \vee \cdots \vee u'_k \in S(v_k) \tag{3}$$

where $v_1, \ldots, v_k$ are adjacent to $v$ in $T$ and $u'_1, \ldots, u'_k$ are either adjacent or incident to $u'$ in $P$. Initially, each node $v$ of $T$ writes down these formulas for each element $u'$ from the set (1). As soon as the right hand side of the implication (3) turns out to be empty (=`false`), the node removes $u'$ from $\tilde{S}(v)$ and then $v$ informs its parent (if it exists) and its children (if they exist) about this removal. Having got this information, the parent and the children delete corresponding disjunctive terms in their formulas. This process can propagate by a chain but since each time at least one disjunctive term is deleted, the total complexity is bounded by the total size of initial formulas which is at most $O(tp^2)$.

## 4. The Window Subsequence Algorithm

### 4.1. Problem 2

We study Problem 2: Given a target tree $T$ and a pattern tree $P$, to decide whether pattern $P$ is an embedded subtree of tree $T$ within a window of height $w$. We will solve an extended version of Problem 2, where we count the number of $w$-windows of $T$ where $P$ can be embedded.

The algorithm is somehow similar to Algorithm1, but the configurations contain not only the embedded bottom-up subtrees of $P$, but also the least possible depth of its image in $T$. We will thus store in configurations ordered pairs consisting of a bottom-up subtree $t$ of $P$ embedded in $T$, together with an integer $n$ representing the length of the longest root-to-leaf path (in $T$) of the current embedding of $t$. The number $n$ will be called a *depth-stamp*.

Moreover, for the extended version of Problem 2, a counter $N$ will count the number of $w$-windows containing $P$.

Configurations will be replaced by *s-configurations* where each occurrence of a subtree of $P$ will be augmented by the least possible value of the depth-stamp. We will modify accordingly the definition of union of configurations to keep track of the depth-stamps. 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.

**Definition 4.1.** A **stamped subtree** is an ordered pair $(t, n)$ where $t$ is a bottom-up subtree of $P$, and $n \in \mathbb{N}$ is called a 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 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 $F_j$.

**Example 4.2.** In a min-s-forest, only minimal stamped subtrees appear: for instance, considering the pattern of Figure 5 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 5, 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 4.3.** 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 $t$.

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.

**Example 4.4.** $\mathrm{red}(\ \{(1,1),(4,1),(4,2)\}\ ) = \{(1,1),(4,1)\}$ because $(4,1)$ has a depth-stamp less than $(4,2)$ hence the latter can be dropped;

$\mathrm{red}(\ \{(1,1),(4,1),(3,1)\}\ ) = \{(1,1),(4,1)\}$ because 3 is a subtree of 4 and they have the same depth-stamp hence $(3,1)$ can be dropped.

**Definition 4.5.** 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 $T''$.

**Definition 4.6.** The **union** of two s-configurations $C = \{F_1, F_2, \cdots, F_k\}$ and $C' = \{F'_1, F'_2, \cdots, F'_{k'}\}$ is s-configuration $D = C \otimes_{\mathrm{s}} C'$ obtained as follows:

1. let $D'' = \left\{\mathrm{red}(F_i \cup F'_{i'}) \mid i \in \{1, ..., k\}, i' \in \{1, ..., k'\}\right\}$

(2) we pass from $D''$ to $D$ by removing each $F''_j \in D''$ which is dominated by some $F''_i \in D''$.

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 $F''$.

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.

We now present Algorithm2 for problem 2. See Figure 4 for an illustration of algorithm2 with $w = 3$.

**Algorithm2**

Let $r :=$ the label of the root of $P$; $N := 0$; //initializations
`FORALL` nodes $v$ of $T$ visited in bottom-up order `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 $\mathit{color}(v') = \mathit{color}(v)$, //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`
  `FOR` $F_j \in C_{v_i}$ `DO` $F'_j := \{(l, d+1) | (l, d) \in F_j$ and $h + d + 1 \leq w$ where $h$ is the height of $l$ in $P$ $\}$ //subtree $(l, d+1)$ cannot contribute to any embedding in a $w$-window if $h + d + 1 > w$ `ENDDO`
  $C'_{v_i} := \left\{ F'_j \mid F_j \in C_{v_i} \right\}$ `ENDDO`
  (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 $a$
        $\Delta := \Delta \cup \left\{\{(j, 0)\}\right\}$ ;

    `ENDIF`
    `ENDDO`
  (d) Reduce the forests in $\Delta$ and remove from $\Delta$ all dominated forests

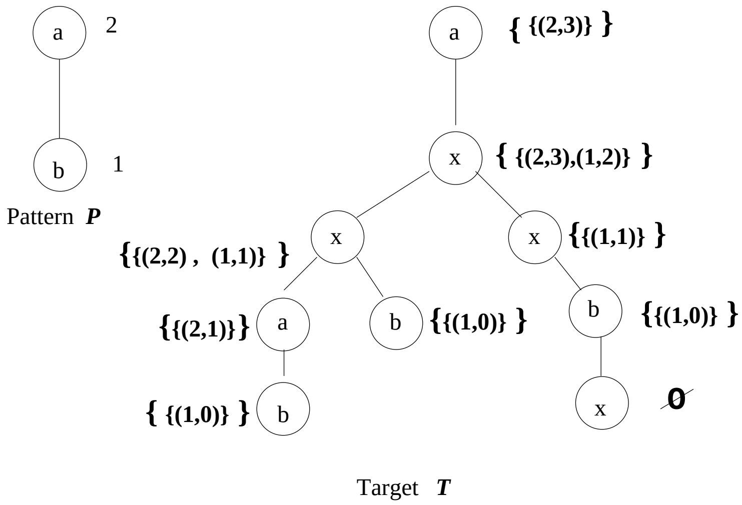


FIGURE 4. Pattern $P$ occurs in four 3-window (window of height 3 at most) in target $T$.

(e) Take the resulting configuration $\Delta$ for $C_v$
(f) `IF` there are $d$ and $F \in C_v$ such that $(r, d) \in F$
`THEN DO` let $d_0$ be the least possible value of such a $d$;
$N := N + 1 + (w - d_0)$ ;
output "$P$ is an embedded subtree of $T$ within $1+(w-d_r)$ $w$-windows 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 5: 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 $T$).

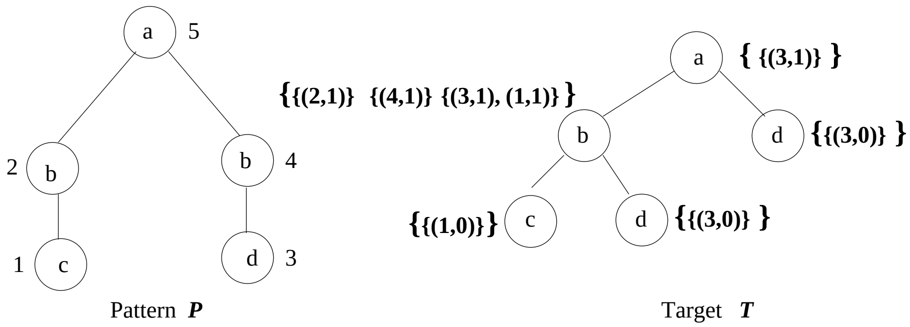


FIGURE 5. Pattern $P$ is not embedded in a 2-window of target $T$.

4.2. **Problem 3.**

Given two trees, *target* $T$ and *pattern* $P$, we want to count the number of occurrences of $P$ as an embedded subtree of $T$ within a window of height $w$.

In order to solve problem 3, configurations will now be replaced by *ms-configurations* which are multisets of multiforests (i.e. multisets of stamped subtrees of $P$). We must also modify the definition of union of configurations to keep track of the multiplicities. Note that we now neither reduce the multiforests nor remove dominated multiforests.

**Definition 4.7.** An **ms-configuration** is a multiset $C = \{F_1, F_2, \dots, F_k\}$, where each $F_i$ is a multiset of bottom-up stamped subtrees of $P$.

**Definition 4.8.** The **union** of two ms-configurations $C = \{F_1, F_2, \dots, F_k\}$ and $C' = \{F'_1, F'_2, \dots, F'_{k'}\}$ is ms-configuration $D = C \otimes_{\mathrm{ms}} C' = \big\{F_i \cup F'_{i'} \mid i \in \{1, ..., k\}\,,\ i' \in \{1, ..., k'\}\big\}$.

Algorithm3 below will count the number of embeddings of $P$ as an embedded subtree of $T$ within a $w$-window. The reader is invited to note that

(1) we now must count all embedded occurrences of $P$ within a $w$-window and not only the minimal ones.
(2) we consider $P$ as a *colored labelled tree*, that is each node has a (unique) *label* from $\{1, \dots, p\}$, and a (not necessarily unique) *color* from the alphabet: different nodes can have the same color but not the same label. For instance in Figure 6, $P$ has 2 occurrences in a 2-window of $T$.

Figure 7 illustrates algorithm3 on the same $P,\ T$ as in Figure 4, with $w = 2$. Figure 8 illustrates algorithm3 with thread-like trees $P,\ T$ with $w = 3$.

**Algorithm3**

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

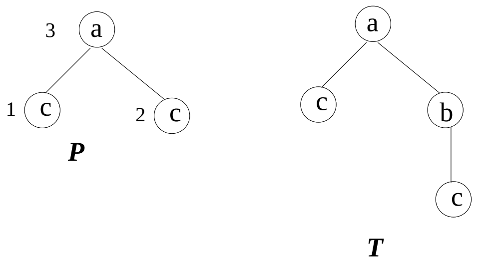


FIGURE 6. Pattern $P$ has 2 occurrences in a window of height 2 of $T$.

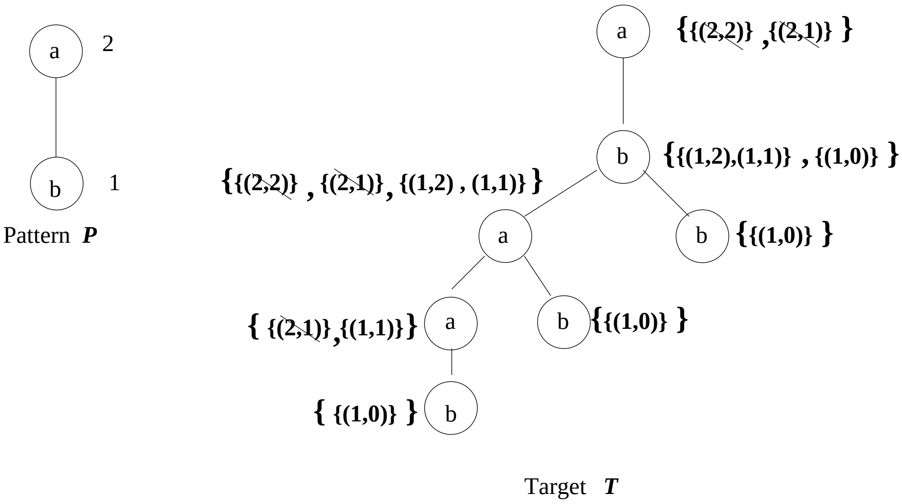


FIGURE 7. Pattern $P$ has 5 occurrences (which are crossed with a backslash) in a 2-window (window of height at most 2) of target $T$.

(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 $color(v') = color(v)$, //the configuration of $v$ is the empty set if no leaf of $P$ has the same color as $v$.

(2) `IF` node $v$ is an internal node colored $a$, with children $v_1, \dots, v_n$, whose respective configurations are $C_{v_i}$, $i = 1, \dots, n$, `THEN DO`
   (a) `FOR` $i = 1, \dots, n$, `DO`
   `FOR` $F_j \in C_{v_i}$ `DO` $F'_j := \{(l, d+1) | (l,d) \in F_j$ and $h + d + 1 \leq w$ where $h$ is the height of $l$ in $P$ $\}$
   `ENDDO`
   $C'_{v_i} := \left\{ F'_j \mid F_j \in C_{v_i} \right\}$ `ENDDO`
   (b) $\Delta := D := C'_{v_1} \otimes_{\mathrm{ms}} C'_{v_2} \otimes_{\mathrm{ms}} \cdots \otimes_{\mathrm{ms}} C'_{v_n}$ ;
   (c) `FORALL` nodes $w$ of $P$ colored $a$ and labelled $j$ `DO` //$w$ has same color as $v$
   - `IF` node $w$ is a not a leaf

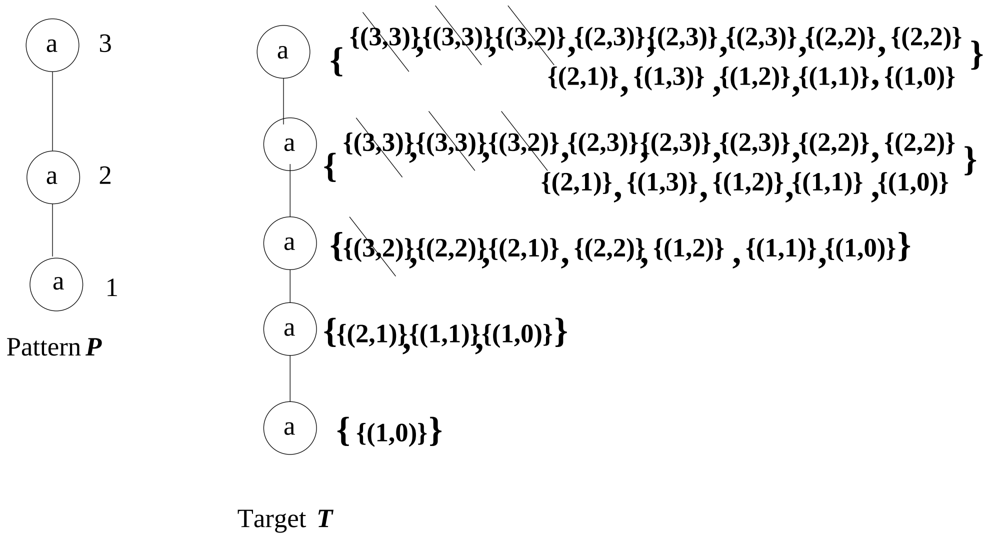


FIGURE 8. Pattern $P$ has 7 occurrences (crossed with a backslash) in a 3-window of target $T$.

$//w$ is an internal node labelled $j$

- `THEN` Let $j_1, ..., j_p$ be the children of $j$ in $P$ ; `DO`
  `FORALL` occurrences $\{(j_1, d_1), ..., (j_p, d_p)\} \subseteq F_i \in D$

  $\Delta := \Delta \cup \big\{\{(j, \max\{d_1, ..., d_p\})\}\big\}$ ;
  `ENDDO`
- `ELSE` $\Delta := \Delta \cup \big\{\{(j, 0)\}\big\}$ ; $//w$ is a leaf labelled $j$

`ENDIF`
`ENDDO`

(d) `FORALL` $(r, d_r)$ such that $(r, d_r) \in F_j \in \Delta$ `DO`
output "$P$ is an embedded subtree of $T$ within a $w$-window at node $v$".
$N := N + 1$ ;
$F_j := F_j \setminus \{(r, d_r)\}$ ; `ENDDO`

(e) $C_v := \Delta$

`ENDDO`
`ENDIF`

`ENDDO`

## 5. Conclusion

In the present paper we answered some problems about tree inclusions, namely 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 $w$-window.

There are many other interesting problems concerning tree inclusions, for instance, counting the number of windows of height exactly $w$ of the target containing the pattern as an embedded subtree, or counting in a slice of the target tree.

## 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.

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

[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.