Simulation relations as a means for pattern-matching in treebanks

Johanna Björklund[a,1,*], Lars-Daniel Öhman[b,1]

[a] *Dept. Comp. Sci, Umeå University, 901 87 Umeå Sweden*
[b] *Dept. Math. and Math. Stat., Umeå University, 901 87 Umeå Sweden*

Abstract

We consider the problem of finding the occurrences of a pattern tree t in a treebank represented as a directed acyclic graph g, and propose a two-tiered technique, consisting of a preprocessing routine and a search algorithm. We assume that whereas the treebank itself is large and more or less static, the pattern tree is small and frequently updated. To model varying abstraction levels in the data, we work with partially ordered alphabets and compute simulation relations rather than equivalence relations. For instance, vertices and edges are labelled with elements from a pair of preorders $\mathcal{P}_\Sigma$ and $\mathcal{P}_\Gamma$ instead of unstructured alphabets. Under the above assumptions, we obtain a search algorithm that runs in time $O(|\mathcal{P}_\Sigma| + \mathit{height}(t)\,|t|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$, where $(V/\mathcal{R}_g)$ is the number of equivalence classes in the coarsest simulation relation $\mathcal{R}_g$ on the vertex set V of g. The size of the treebank thus only affects the running time of the search algorithm indirectly, and this is due to the groundwork done by the preprocessing routine in time $O(|\mathcal{P}_\Sigma| + \mathit{height}(g)\,|g|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$.

Keywords: pattern matching, simulation, treebanks, complexity

1. Introduction

The last decade has seen immense progress in the fields of sensor and memory technology. The greater ease with which data can be collected and stored paves the way for statistical approaches to a wide range of application areas, including machine translation (Koehn, 2010), bioinformatics (Lee, 2010), and network analysis (Kolaczyk, 2009). However, the distillation of information from raw data requires algorithms and data structures that support indexing, analysis, and search. For these purposes, hierarchical approaches often prove particularly useful, as exemplified by standardized markup languages derived from XML that structure content as trees with auxiliary links. As demonstrated by Maneth et al. (2008), XML documents can be compactly represented as directed acyclic graphs (DAG).

*Principal corresponding author. Tel. +46 70 603 94 59, Fax. +46 90 786 99 95.
Email addresses: johanna.bjorklund@cs.umu.se (Johanna Björklund), lars-daniel.ohman@cs.umu.se (Lars-Daniel Öhman)

[1]Supported by Vinnova within the Vinnmer programme.

In this work, we study the pattern-matching problem for treebanks (i.e., sets of trees) and investigate how their inherent hierarchical structure can be used to facilitate searches. The scenario is that we have a large but more or less static treebank, consisting of e.g. XML formatted data or syntactically annotated text, that we frequently want to search for occurrences of relatively small, but varying, patterns. This search task is formalised as a pattern-matching problems for acyclic graphs, where the input is a set of trees T and an additional pattern tree t. The problem is: Does t occur as a pattern in T? In other words, is there an edge-preserving mapping that takes the vertices of t to those of a tree $t' \in T$? To save space, the large set T should be represented as a DAG g, whereas the comparatively small t may be given as an 'uncompressed' tree.

To solve this problem, we propose a two-step pattern-matching algorithm that consists in a preprocessing routine that is computationally demanding but only computed once, and a light-weight pattern-matching routine that is executed for every new search tree. The preprocessing consists in deriving the unique coarsest simulation relation on the node set V of the object treebank, through step-wise refinement of an approximating relation. The pattern-matching algorithm then retraces the steps taken by the preprocessing routine to decide to what equivalence class $[v]$ the root node v of a pattern tree belongs. The algorithm does not recompute the actual refinement steps, but follows instead a trail of 'breadcrumbs', i.e., a sequence of stored feature vectors. Once $[v]$ has been found, the occurrences of t in the treebank is quite simply the vertex set $[v] \setminus \{v\}$.

1.1. Related work

Pattern matching has been studied extensively for graphs in general, but also for restricted forms such as acyclic graphs and trees. When a graph is acyclic, we measure its height as the length of the longest path from a source vertex to a sink vertex. In our review of related work, g and t denote the object graph and pattern tree, respectively. We write V_x for the vertex set of the graph x and E_x for the edge set. The vertices are labelled with elements from a preorder $\mathcal{P}_\Sigma$ and the edges with elements from a preorder $\mathcal{P}_\Gamma$. With this notation, the complexity expression for our preprocessing routine is $O(|\mathcal{P}_\Sigma| + \mathit{height}(g)\,|g|\,|(V_g/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$, where $(V_g/\mathcal{R}_g)$ is the number of equivalence classes in coarsest simulation relation $\mathcal{R}_g$ on the vertex set of g. The search routine is closely guided by the preprocessing routine, and this shines through in its time complexity: $O(|\mathcal{P}_\Sigma| + \mathit{height}(t)\,|t|\,|(V_g/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$. It is worth noticing that the size of the treebank only affects the running time of the search routine indirectly through $\mathcal{R}_g$.

There is, to the best of our knowledge, no previous work on pattern matching for treebanks that considers structured vertex and edge alphabets. To relate our results to those of others before us, we focus temporarily on the special case where both preorders are identity relations and the alphabets themselves are fixed and do not appear in the complexity expressions. Under these restrictions, the complexity expression for the preprocessing routine is simplified to $O(\mathit{height}(g)\,|g|\,|(V_g/\mathcal{R}_g)|^2)$ and the corresponding complexity expression for the search routine becomes $O(\mathit{height}(t)\,|t|\,|(V_g/\mathcal{R}_g)|^2))$. Another obstacle for comparison is the appearance of $|(V_g/\mathcal{R}_g)|$ as a factor in the two expressions. Since we are working with acyclic graphs, the number of equivalence classes with respect to $\mathcal{R}_g$ is bounded from below by $\mathit{height}(g)$, and from above by $|g|$. Intuitively, the size of the simulation relation $\mathcal{R}_g$ is inversely proportional to the number of reoccurring substructures in the graph.

Wuu et al. (2000) also divides the computational work between a preprocessing step and a search routine, but in their scenario, it is the pattern tree that is fixed and the object tree that varies. The authors obtain a running time of $O(|t|\, h)$ for the preprocessing step, where h is the height of a certain tree constructed during preprocessing, is bounded from above by $|t|$, and a running time of $O(|g| \log |g|)$ for the actual matching. Whether their algorithm or ours is more efficient seems to depend on the amount of reoccurring structures in the data; we leave a closer comparison and perhaps also a fusing of the results as a subject for future work.

Simulation as a means for pattern-matching has also been investigated by Fan et al. (2010). The authors use a notion of bounded simulation rather different from ours, in which edges can be mapped onto paths bounded in length by some constant k. This leads to an algorithm that computes the largest occurrence in time

$$O\Big(|V_g|\,|E_g| + |V_t|\,|V_g|^2 + |V_g|\,|V_t|\Big) \quad .$$

Using a stack-based approach, Chen et al. (2005) obtain a pattern-matching algorithm for DAGs that executes in time $O(|g|\,|t|^2)$. Cheng et al. (2008) present and empirically evaluate a pattern-matching algorithm based on *reachability join*. Another empirical study is conducted by Yao and Zhang (2004) who consider tree pattern-matching algorithms within the context of XML Querying.

Wang et al. (2002) supply an algorithm for finding approximate patterns with respect to a certain edit-distance in undirected acyclic graphs. In the context of edit distances, pattern-matching for trees has also been treated under the name *tree inclusion* by Bille (2005). Kilpelinen and Mannila (1995) presented a decision algorithm for ordered trees that runs in time $O(|g|\,|t|)$. This bound was improved upon by Chen (1998) who gave an $O(|g|\,|\mathit{leaves}(t)|)$ algorithm, where $\mathit{leaves}(t)$ is the subset of V_t that lacks outgoing edges.

A closely related problem is that of subgraph isomorphism, which consists in deciding whether a given graph g has a subgraph g'. Since the subgraph isomorphism problem generalises both the maximum clique problem and the Hamilton path problem, it is NP-hard (Ullmann, 1976; Chen et al., 2008). However, polynomial time algorithms exists for restricted versions of the problem, such as those provided by Sundaram and Skiena (1995) for subgraphs with bounded flower number.

Regarding our theoretical framework, (bi-)simulation relations for graphs were first introduced by Milner (1982) as a formal means of investigating transition systems, and then extended to the weighted setting by Buchholz (2008) almost three decades later. An efficient algorithm for computing simulations relations on tree structures has been presented by Abdulla et al. (2008) and lifted to the weighted setting by Maletti (2009).

1.2. Outline

The disposition of this paper is as follows. Section 2 contains those concepts from algebra and automata theory on which our presentation rests. Section 3 establishes a theoretical framework and introduces the notion of approximated simulation. Section 4 outlines the preprocessing and search routines together with a discussion of their computational complexity. Section 5 first considers a special case of our framework and then turns to pattern matching for ordered trees. Section 6 contains concluding remarks and suggestions for future work.

2. Preliminaries

This section reviews useful notions and notations from the fields of algebra and tree automata. For an introduction to the latter field, see e.g. the survey by Comon et al. (1997) or the handbook by Droste et al. (2009). The relational modelling of graphs is inspired by a book on rewriting systems by Baader and Nipkow (1998).

Sets and numbers. We write $\mathbb{B}$ for the set of truth values $\{0, 1\}$ and the $\mathbb{N}$ for the natural numbers, or $\mathbb{N}^+$ if we wish to exclude 0 from $\mathbb{N}$. For $k \in \mathbb{N}$, we write $[k]$ for $\{1, \ldots, k\}$. As a special case, we have $[0] = \emptyset$.

An *alphabet* is a finite nonempty set. Let S be a set. The *strings over* S, written S^*, is the set of all finite sequences of elements of S, the *empty string* ε is the unique sequence $w \in S^*$ of length 0, and $S^+ = S^* \setminus \{\varepsilon\}$. The power set of S, i.e. the set of all subsets of S, is denoted by $pow(S)$.

Relations. Let S be a set and $\mathcal{R}, \mathcal{P} \subseteq S \times S$ relations. The *composition* of $\mathcal{R}$ and $\mathcal{P}$ is $\mathcal{R}\mathcal{P} = \{(s, s'') \mid \exists s' \in S \colon (s, s') \in \mathcal{R} \wedge (s', s'') \in \mathcal{P}\}$.

We write $\mathcal{R}^{-1}$ for the inverse of $\mathcal{R}$, and $\mathcal{R}^*$ for the transitive closure of $\mathcal{R}$. In other words, $\mathcal{R}^{-1} = \{(s', s) \mid (s, s') \in \mathcal{R}\}$ and $\mathcal{R}^*$ is the smallest superset of $\mathcal{R}$ such that $(s, s'), (s', s'') \in \mathcal{R}^*$ implies $(s, s'') \in \mathcal{R}^*$.

The relation $\mathcal{R}$ is:

- the *identity relation* I_S on S if $\mathcal{R} = \{(s, s) \mid s \in S\}$;
- *reflexive* if $I_S \subseteq \mathcal{R}$;
- *transitive* if $\mathcal{R}\mathcal{R}^* = \mathcal{R}$;
- *symmetric* if $(s, s') \in \mathcal{R}$ implies that $(s', s) \in \mathcal{R}$;
- *antisymmetric* if $(s, s') \in \mathcal{R}$ and $(s', s) \in \mathcal{R}$ implies that $s' = s$;
- an *equivalence relation* if it is reflexive, transitive and symmetric; and
- a *preorder* if it is reflexive and transitive.

We denote by $\mathcal{R}^\star$ the reflexive and transitive closure of $\mathcal{R}$, i.e., $\mathcal{R}^\star = (\mathcal{R} \cup \mathcal{R}^{-1})^*$. For $s \in S$, $\mathcal{R}(s)$ denotes the set $\{s' \mid (s, s') \in \mathcal{R}\}$. This operation is lifted to sets in the natural way, i.e., for $S' \subseteq S$, we have $\mathcal{R}(S') = \cup_{s \in S'} \mathcal{R}(s)$.

Partitions. Let S be a set, and let $\mathcal{R}$ and $\mathcal{P}$ be relations on S. We say that $\mathcal{R}$ is a *refinement* of $\mathcal{P}$ if $\mathcal{R} \subseteq \mathcal{P}$, and that $\mathcal{R}$ is a *proper* refinement of $\mathcal{P}$ if $\mathcal{R} \subset \mathcal{P}$. The *equivalence class* (or *block*) of an element $s \in S$ with respect to a preorder $\mathcal{R}$ is the set $[s]_\mathcal{R} = \{s' \mid (s, s'), (s', s) \in \mathcal{R}\}$. By definition, $[s]_\mathcal{R}$ and $[s']_\mathcal{R}$ are are either equal or disjoint, so $\mathcal{R}$ induces a partition $(S/\mathcal{R}) = \{[s]_\mathcal{R} \mid s \in S\}$ of S. If it is clear from the context which relation $\mathcal{R}$ is intended, we write $[s]$ for $[s]_\mathcal{R}$.

Orders. Let $\mathcal{R} \subseteq V \times V$ be a preorder. The *upset* of $v \in V$ with respect to $\mathcal{R}$ is simply $\mathcal{R}(v)$. An element $v \in V$ is *maximal* with respect to $\mathcal{R}$ if $\mathcal{R}(v) = [v]_\mathcal{R}$. For the remainder of this paper, Σ and Γ are fixed but arbitrary alphabets, and $\mathcal{P}_\Sigma$ and $\mathcal{P}_\Gamma$ are fixed but arbitrary preorders on Σ and Γ, respectively.

A *partial order* is an antisymmetric preorder. Given a preorder $\mathcal{R}$ on V, we denote by $\hat{\mathcal{R}}$ the partial order on $(V/\mathcal{R})$ given by $\hat{\mathcal{R}} = \{([v], [v']) \mid (v, v') \in \mathcal{R}\}$. If for any s, s' it holds that either (s, s') or $(s', s) \in \mathcal{R}$, then $\mathcal{R}$ is a *total order*.

Graphs. A *(vertex- and edge-labelled) graph* is a tuple $g = (V, E, s, l)$ where V is a set of *vertices* and E is a binary relation on V representing *edges*. The mappings $s : V \mapsto \Sigma$ and $l : E \mapsto \Gamma$ are the *vertex- and edge-labellings* of g, respectively. To avoid always stating the entire tuple, we denote by V_g and E_g the sets V and E of g, respectively. Furthermore, we use $v \xrightarrow{\gamma} v'$ as a shorthand for $(v, v') \in E \wedge l((v, v')) = \gamma$.

Let $g = (V, E, s, l)$ be a graph and $V' \subseteq V$. The *size* of g is $|g| = |V| + |E|$. The *subgraph* of g *induced* by V' is the graph $g[V'] = (V', E \cap (V' \times V'), s, l)$, with the mappings s and l properly restricted.

A *path* through g is a sequence $v_1, \ldots, v_k$ of vertices such that $(v_i, v_{i+1}) \in E$ for every $i \in [k-1]$. The path is a *cycle* if $v_1 = v_k$. The *length* of a path $v_1, \ldots, v_k$ is $k - 1$. The vertex $v' \in V$ is *reachable* from $v \in V$ or, in other terminology, there is a path from v to v' if $v' \in E^*(v)$. The $\mathit{height}(g)$ of g is the length of the longest path through g.

The graph g is *acyclic* if $I_V \cap E^* = \emptyset$ and *connected* if $V \times V \subseteq E^\star$. Let g be an acyclic graph. The *subgraph* g *attached at* $U \subseteq V_g$, which we write as $g|_U$ is $g[E_g^*(U)]$. Informally, the subgraph attached at U consists of everything 'downstream' from U. Using the notion of height, we stratify the vertices of g by letting

$$V_g^i = \{v \in V_g \mid \mathit{height}(g|_{\{v\}}) \leq i\}$$

for every $i \in [\mathit{height}(g)]$, where we may drop the g from V_g if it is clear from the context.

Trees. The graph $g = (V, E, s, l)$ is a *tree* if

- $(V, E \cup E^{-1})$ is acyclic, and
- there is a unique vertex v such that $V = E^*(v)$. If such a v exists, then it is referred to as the *root* of t and denoted by $\mathit{root}(t)$.

From this definition, it follows immediately that every tree is connected and acyclic. The *leaves* of a tree t, written $\mathit{leaves}(t)$, is the set $\{v \in V' \mid E(v) = \emptyset\}$.

A *subtree* of t rooted at $v \in V_t$, henceforth denoted by $t|_v$, is a subgraph of t attached at $\{v\}$. A subtree is *direct* if it is rooted at a vertex in $E(\mathit{root}(t))$. We denote the set of all subtrees of t by $\mathit{subtrees}(t)$.

We write a tree $t = (V, E, s, l)$ with direct subtrees $t_1, \ldots, t_k$ as as $\sigma[\gamma_1 \colon t_1, \ldots, \gamma_k \colon t_k]$, where $\sigma = s(\mathit{root}(t))$ and $\gamma_i = l((\mathit{root}(t), \mathit{root}(t_i)))$ for every $i \in [k]$. If $k = 0$, we omit the brackets and write t simply as σ.

3. Theoretical framework

We address the following informally stated pattern-matching problem for trees.

> Given a comprehensive treebank and a sequence of pattern trees, at what positions in the treebank do the pattern trees occur?

We shall assume that the treebank is represented by a DAG g for the sake of compactness, but that the pattern trees, being comparatively small and given in a one-by-one fashion, are stored as regular trees. In our formalisation, we take the statement that a pattern tree t occurs in g to mean that there is a mapping that identifies t with a connected subgraph t' of g. When there are vertex- or edge-labellings involved, these are to be

respected by the projection. To avoid unnecessary loss of generality, we shall not require the labellings of t and t' to be identical, but rather that the labels assigned to the vertices and edges of t are consistently 'smaller' than those assigned to the vertices and edges of t'. An intuitive interpretation is that g can in some sense accommodate for t.

Definition 1 (Occurrence). A tree $t = (V_t, E_t, s_t, l_t)$ *occurs* in a directed acyclic graph $g = (V_g, E_g, s_g, l_g)$ at vertex $v \in V_g$ if there is a function $\varphi\colon V_t \mapsto V_g$ such that:

- $\varphi(root(t)) = v$;
- for every $u \in V_t$, $(s_t(u), s_g(\varphi(u))) \in \mathcal{P}_\Sigma$;
- for every $e = (v_1, v_2) \in E_t$, $e' = (\varphi(v_1), \varphi(v_2)) \in E_g$ and $(l_t(e), l_g(e')) \in \mathcal{P}_\Gamma$.

The function φ is a *certificate* that t occurs in g at v. The *occurrences of t in g* is the vertex set

$$occur_g(t) = \{v \in V_g \mid t \text{ occurs at } v\} \ .$$

The central task of our pattern-matching problem can now be formalised in terms of Definition 1:

Problem 2. *Given a DAG g and a tree t, find $occur_g(t)$.*

Since we wish to compute the set $occur_g(t)$ for a fixed DAG g but varying pattern trees t, we propose a two-tiered solution that consists of (i) a preprocessing routine that derives an index structure from g, and (ii) a search routine that takes advantage of the index structure to find the occurrences of some particular pattern tree. The index structure that we have in mind is the *coarsest simulation preorder* on g, that is, the most inclusive of a particular class of relations on V_g, which we will now define.

Intuitively, a vertex $v' \in V_g$ *simulates* a vertex $v \in V_g$ if every sequence of steps along directed edges that can be taken when starting from v, can also be taken when starting from v'. As Lemma 12 will show, if a vertex v' simulates a vertex v, then every tree that occurs at v also occurs at v'.

Definition 3 (Simulation). Let $g = (V, E, s, l)$ be a graph. The binary relation $\mathcal{R}$ on V is a *simulation* on g if $(v, v') \in \mathcal{R}$ implies that:

1. $(s(v), s(v')) \in \mathcal{P}_\Sigma$; and
2. for every $u \in V$, $\gamma \in \Gamma$ such that $v \xrightarrow{\gamma} u$; there are $u' \in \mathcal{R}(u)$, $\gamma' \in \mathcal{P}_\Gamma(\gamma)$ such that $v' \xrightarrow{\gamma'} u'$.

Recall that a relation $\mathcal{R}$ is said to be *coarser* than a relation $\mathcal{R}'$ if $\mathcal{R}' \subseteq \mathcal{R}$.

Lemma 4. *For every graph g, there is a unique coarsest simulation $\mathcal{R}_g$ on g.*

Proof. Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be distinct simulations on $g = (V, E, s, l)$. We show that the relation $\mathcal{R} = (\mathcal{R}_1 \cup \mathcal{R}_2)^*$, which is strictly coarser than $\mathcal{R}_1$ and $\mathcal{R}_2$, is a simulation on g. The proof has its starting point in the simple observation that whenever

$(v, v') \in \mathcal{R} = (\mathcal{R}_1 \cup \mathcal{R}_2)^*$ there are vertices $v_1, \ldots, v_k \in V$ such that $v = v_1$, $v' = v_k$, and $(v_i, v_{i+1}) \in \mathcal{R}_1 \cup \mathcal{R}_2$ for every $i \in [k-1]$.

Let us first verify that Condition 1 of Definition 3 is met. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are simulations, $(s(v_i), s(v_{i+1})) \in \mathcal{P}_\Sigma$ for every $i \in [k-1]$. Since $\mathcal{P}_\Sigma$ is a preorder it is transitive, so also $(s(v), s(v')) \in \mathcal{P}_\Sigma$.

To verify Condition 2, assume that $v_1 \xrightarrow{\gamma} u_1$ and that v, v' and $v_1, \ldots, v_k$ are as above. By induction on the length of the sequence $v_1, \ldots, v_k$ and the fact that $\mathcal{R}_1$ and $\mathcal{R}_2$ are simulations it follows that for every $i \in [k-1]$, there are $u_{i+1} \in \mathcal{R}_1(u_i) \cup \mathcal{R}_2(u_i), \gamma_{i+1} \in \mathcal{P}_\Gamma(\gamma_i)$ such that $v_{i+1} \xrightarrow{\gamma_{i+1}} u_{i+1}$. Since both $\mathcal{R}$ and $\mathcal{P}_\Gamma$ are preorders and thus transitive, we have $(u_1, u_k) \in \mathcal{R}$ and $(\gamma_1, \gamma_k) \in \mathcal{P}_\Gamma$. Thus Condition 2 is satisfied, and thus $\mathcal{R}$ is a simulation.

Since g is finite, there is a finite number of simulations, and thus, by applying the above construction, we see that there exists a unique coarsest simulation. □

The coarsest simulation on g (which shall henceforth be denoted by $\mathcal{R}_g$) can be computed through an iterative refinement of the preorder induced by $\mathcal{P}_\Sigma$ on V_g, as detailed in Definition 5. When these approximation steps eventually reach a fix-point, the sought simulation has been obtained.

Definition 5 (Approximated simulation). Let $g = (V, E, s, l)$ be a directed graph. The 0-*approximated simulation* on g is the relation

$$\mathcal{R}_0 = \{(v, v') \in V \times V \mid (s(v), s(v')) \in \mathcal{P}_\Sigma\} \ .$$

For every $i \in \mathbb{N}^+$, the i-approximated simulation on g is

$$\begin{aligned} \mathcal{R}_i \quad = \quad \{(v, v') \in \mathcal{R}_{i-1} \mid \exists u \in V, \gamma \in \Gamma : v \xrightarrow{\gamma} u \Rightarrow \\ \exists u' \in \mathcal{R}_{i-1}(u), \gamma' \in \mathcal{P}_\Gamma(\gamma) : v' \xrightarrow{\gamma'} u'\} \ . \end{aligned}$$

Note that an approximated simulation is not in general a simulation, but as we shall see, the approximation sequence converges to the coarsest simulation $\mathcal{R}_g$ on g.

Lemma 6 (Refinement). *For every $i \in \mathbb{N}$,*

1. *$\mathcal{R}_i$ is a preorder;*
2. *$\mathcal{R}_i \subseteq \{(v, v') \in V \times V \mid (s(v), s(v')) \in \mathcal{P}_\Sigma\}$; and*
3. *$\mathcal{R}_{i+1} \subseteq \mathcal{R}_i$.*

Proof. Let us first show that $\mathcal{R}_i$ is a preorder. It follows directly from Definition 5 that $\mathcal{R}_i$ is reflexive for every $i \in \mathbb{N}$. For $i = 0$, the transitivity of $\mathcal{R}_i$ follows from the fact that $\mathcal{P}_\Sigma$ is a preorder. For $i \geq 1$, suppose that $(v, v'), (v', v'') \in \mathcal{R}_i$, $u \in V$, $\gamma \in \Gamma$, and $v \xrightarrow{\gamma} u$. By Definition 5, there exists a $u' \in \mathcal{R}_{i-1}(v')$ and a $\gamma' \in \mathcal{P}_\Gamma(\gamma)$ such that $v' \xrightarrow{\gamma} u'$. Since $(v', v') \in \mathcal{R}_i$ there is also a $u'' \in \mathcal{R}_{i-1}(u')$ and a $\gamma'' \in \mathcal{P}_\Gamma(\gamma')$ such that $v'' \xrightarrow{\gamma} u''$. Since $\mathcal{P}_\Gamma$ is a preorder and $\mathcal{R}_{i-1}$ transitive by the induction hypothesis, $(u, u'') \in \mathcal{R}_{i-1}$ and $(\gamma, \gamma'') \in \mathcal{P}_\Gamma$ so $(v, v'') \in \mathcal{R}_i$, establishing $\mathcal{R}_i$ as a preorder. The two inclusions are immediate from Definition 5. □

The proof of convergence is informed by a treatise on the relational coarsest partition problem by Paige and Tarjan (1987). For the sake of completeness, we state here an adapted version of a key lemma and the correctness proof.

Lemma 7 (Paige and Tarjan (1987), Lemma 2). *For every $i \in \mathbb{N}$, $\mathcal{R}_g \subseteq \mathcal{R}_i$.*

If we compare Definition 3 and Definition 5, we see that if the relation $\mathcal{R}_{i+1}$ coincides with $\mathcal{R}_i$ then it is a simulation. By Lemma 6, we always have $\mathcal{R}_{i+1} \subseteq \mathcal{R}_i$. Furthermore, for every $i \in \mathbb{N}$ it is clear that $\{(v, v) \mid v \in V\} \subseteq \mathcal{R}_i$ so convergence is guaranteed after at most $|V|^2$ approximation steps. From Lemma 7 it is then a short step to Lemma 8.

Lemma 8 (Paige and Tarjan (1987), Theorem 2). *There is a $k \in \mathbb{N}$ such that for every $i \geq k$, $\mathcal{R}_i$ is a simulation, and $\mathcal{R}_{i+1} = \mathcal{R}_i = \mathcal{R}_g$.*

We are primarily interested in acyclic graphs, and for this class the approximation sequence converges rapidly. We recall that V_g^i is the set of vertices of g from which all leaves will be reached in i steps or less.

Lemma 9 (Acyclic case). *Let g be an acyclic graph. For every $i \leq \mathit{height}(g)$,*

$$\mathcal{R}_i \cap (V_g^i \times V_g) = \mathcal{R}_g \cap (V_g^i \times V_g) \ .$$

Proof. For every $(v, v') \in (V_g^0 \times V_g)$,

$$(v, v') \in \mathcal{R}_0 \iff (s(v), s(v')) \in \mathcal{P}_\Sigma \iff (v, v') \in \mathcal{R}_g \ .$$

This establishes a base for an inductive proof over the structure of g.

For the induction, let $(v, v') \in \mathcal{R}_g \cap (V_g^i \times V_g)$. From Lemma 6 we know that $\mathcal{R}_g \subseteq \mathcal{R}_i$, so $\mathcal{R}_g \cap (V_g^i \times V_g) \subseteq \mathcal{R}_i \cap (V_g^i \times V_g)$.

Suppose now that we have $(v, v') \in \mathcal{R}_i \cap (V_g^i \times V_g)$. From Lemma 6 it follows that $(v, v') \in \mathcal{R}_{i-1}$, and from Definition 5 it follows that for all $u \in V_g$, $\gamma \in \Gamma$, such that $v \xrightarrow{\gamma} u$, there are $u' \in \mathcal{R}_{i-1}$, $\gamma' \in \mathcal{P}_\Gamma(\gamma)$ such that $v' \xrightarrow{\gamma'} u'$. Since g is acyclic, $u \in V_g^{i-1}$. Due to the induction hypothesis, we thus always have that $(u, u') \in \mathcal{R}_g$. Being the coarsest simulation on g, $\mathcal{R}_g$ must therefore contain (v, v'), so $(v, v') \in \mathcal{R}_g \cap (V_g^i \times V_g)$, and thus $\mathcal{R}_i \cap (V_g^i \times V_g) \subseteq \mathcal{R}_g \cap (V_g^i \times V_g)$. □

Note that by the definition of V_g^i, for every $i \geq \mathit{height}(g)$ it holds that $\mathcal{R}_i \subseteq (V_g^i \times V_g)$. We therefore have the following corollary.

Corollary 10 (Acyclic bound). *If g is acyclic and $i \geq \mathit{height}(g)$, then $\mathcal{R}_i = \mathcal{R}_g$.*

From an inductive argument, quite similar to the proof of Lemma 9, it follows that a vertex located higher up in g cannot be simulation-dominated by a vertex located at a lower level. More formally:

Propostition 11 (Stratified simulation). *Let g be an acyclic graph with height h and let $\mathcal{R}$ be a simulation on g. Then,*

$$\mathcal{R} \subseteq \bigcup_{i,j \in [h],\, i \leq j} V_g^i \times V_g^j \ .$$

Proposition 11 means in particular that the root of a tree cannot be simulated by any other node in the tree, so if $\mathcal{R}$ is a simulation on the tree t, then $\mathcal{R}(root(t)) = \{root(t)\}$.

We are now ready to relate the notions of occurence and approximated simulation.

Lemma 12 (Upward closed). *Let g be a DAG, t a tree, and $\mathcal{R}_j$ a j-approximated simulation on the disjoint union $g \uplus t$ for some $j \geq height(t)$. Then*

$$occur_{g \uplus t}(t) = \mathcal{R}_j(root(t)) \ .$$

Proof. It is clear $t = \sigma[\gamma_1 \colon t_1, \ldots, \gamma_k \colon t_k]$ occurs in $g \uplus t = (V_{g \uplus t}, E_{g \uplus t}, s_{g \uplus t}, l_{g \uplus t})$ at $root(t)$. We prove by induction on the height of t, that t also occurs in $g \uplus t$ at $v \in V_{g \uplus t}$ if and only if $(root(t), v) \in \mathcal{R}_j$. The following line of argument can be used both for the base case, when $height(t)$ and k are zero, and for the inductive step, when $height(t)$ and k are greater than zero.

$$
\begin{aligned}
& (root(t), v) \in \mathcal{R}_j \\
\iff \quad & \text{by Definition 5} \\
& (root(t), v) \in \mathcal{R}_{j-1} \wedge \\
& \forall i \in \{1, \ldots, k\} : \exists v_i \in \mathcal{R}_{j-1}(root(t_i)), \gamma_i' \in \mathcal{P}_\Gamma(\gamma_i) : v \xrightarrow{\gamma_i'} v_i \\
\iff \quad & \text{by Lemma 6 and the induction hypothesis} \\
& (\sigma, s_{g \uplus t}(v)) \in \mathcal{P}_\Sigma \wedge \\
& \forall i \in \{1, \ldots, k\} : \exists v_i \in occur_{g \uplus t}(t_i), \gamma_i' \in \mathcal{P}_\Gamma(\gamma_i) : v \xrightarrow{\gamma_i'} v_i \\
\iff \quad & \text{by Definition 1} \\
& v \in occur_{g \uplus t}(t)
\end{aligned}
$$

□

Combining Lemma 9, Lemma 12, and Proposition 11, we finally arrive at Theorem 13.

Theorem 13. *Let g be a DAG, t a tree with root v_t, and $\mathcal{R}_j$ a j-approximated simulation on $g \uplus t$ for some $j \geq height(t)$. Then,*

$$occur_g(t) \ = \ \mathcal{R}_j(v_t) \setminus V_t \ = \ \mathcal{R}_{g \uplus t}(v_t) \setminus \{v_t\} \ .$$

4. Algorithm

We propose a two-tiered approach to pattern matching that consists of a preprocessing routine and a search routine, henceforth referred to as Preprocess and Search, respectively. The input to Preprocess is a corpus of trees, represented as a DAG g. The routine computes and stores a sequence of approximated simulations $\mathcal{R}_0, \mathcal{R}_1, \ldots$ (see Definition 5) until the unique coarsest simulation $\mathcal{R}_g$ on V_g is found.

The input to the algorithm Search is a pattern tree t together with the sequence of relations $\mathcal{R}_0, \mathcal{R}_1, \ldots, \mathcal{R}_g$ computed by Preprocess; the output is the set of vertices

$occur_g(t)$ where t occurs in g. The algorithm SEARCH begins by verifying that the height h of t does not exceed that of g — if it does, the routine simply returns the empty set and exits. SEARCH then computes the h-approximated simulation $\mathcal{R}'_h$ on the graph $g \uplus t$, based on the relations $\mathcal{R}_0, \ldots, \mathcal{R}_h$ and returns $\mathcal{R}'_h(root(t)) \setminus \{root(t)\}$.

The correctness of this approach follows from Theorem 13, so what remains is to find the time complexities of the two routines. For this purpose, we fix an acyclic graph $g = (V, E, s, l)$. In our computations, we represent a preorder $\mathcal{R}$ on a set V by the Hasse diagram, as introduced by Vogt (1985), for the partial order $\hat{\mathcal{R}}$, together with an enumeration of the elements of $[v]_{\mathcal{R}}$ for every $[v]_{\mathcal{R}} \in (V/\mathcal{R})$. The size $|\mathcal{P}|$ of $\mathcal{P}$ is thus taken to be the combined size of these two data structures.

To get a uniform upper bound we state and prove Lemma 14.

Lemma 14. *For every $i \in \mathbb{N}$, $|\hat{\mathcal{R}}_i| \leq |(V/\mathcal{R}_g)|^2$.*

PROOF (SKETCH). The edges of a Hasse diagram H with vertex set V_H is a subset of $V_H \times V_H$, so H has no more than $|V_H|^2$ edges. By Lemma 8, $\mathcal{R}_g \subseteq \mathcal{R}_i$, so it follows that for every $i \in \mathbb{N}$, $|(V/\mathcal{R}_i)| \leq |(V/\mathcal{R}_g)|$, so $|\hat{\mathcal{R}}_i| \leq |(V/\mathcal{R}_g)|^2$. □

Let us first establish the cost of computing approximated simulations.

Lemma 15 (Approximation refinement). *The computational cost of obtaining and refining an approximation is as follows.*

1. *The relation $\mathcal{R}_0$ is computed in $O(|V| + |\mathcal{P}_\Sigma|)$ computation steps.*
2. *For every $i \in \mathbb{N}$, $\mathcal{R}_{i+1}$ is derived from $\mathcal{R}_i$ in time $O(|g|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$.*

PROOF. A representation of $\mathcal{R}_0$ can be obtained from a representation of $\mathcal{P}_\Sigma$ by adding to every vertex $[\sigma]$ in the Hasse diagram for $\hat{\mathcal{P}}_\Sigma$, a reference to every vertex in the set $\{s^{-1}(\sigma') \mid \sigma' \in [\sigma]\}$. The time required for the initiation step is thus $O(|V| + |\mathcal{P}_\Sigma|)$.

The relation $\mathcal{R}_{i+1}$ is derived from $\mathcal{R}_i$ as follows:

1. For every $[u] \in (V/\mathcal{R}_i)$, the vertices in the upset $\mathcal{R}_i(u)$ of u are marked with '$[u]$'. This can be done in time $O((|\hat{\mathcal{R}}_i| + |V|)\,|(V/\mathcal{R}_i)|)$ if the Hasse diagram of $\hat{\mathcal{R}}_i$ is traversed bottom-up.

2. Another bottom-up traversal is made, this time of the graph g. The annotations that were made in Step 1 are now used to establish the characteristic function $char : V \times (V/\mathcal{R}_i) \times (\Gamma/\mathcal{P}_\Gamma) \mapsto \mathbb{B}$ defined by

$$char(v, [u], [\gamma]) \quad = \quad \begin{cases} 1 & \text{if } \exists \gamma' \in [\gamma], \exists u' \in \mathcal{R}_i(u), : v \xrightarrow{\gamma'} u' \text{ , and} \\ 0 & \text{otherwise.} \end{cases}$$

   The cost of this step is $O(|g|\,|(V/\mathcal{R}_i)|\,|\mathcal{P}_\Gamma|)$.

3. Finally, we refine $\mathcal{R}_i$ into $\mathcal{R}_{i-1}$ by modifying its Hasse diagram to account for the new evidence in *char*. The cost for this step is

$$O\Big(|(V/\mathcal{R}_i)|\,|(\Gamma/\mathcal{P}_\Gamma)|\,(|V| + |\hat{\mathcal{R}}_{i+1}|)\Big) \ .$$

To simplify the complexity expression, we use the following inequalities:

1. $|(V/\mathcal{R}_i)| \leq |(V/\mathcal{R}_g)| \leq |V| \leq |g|$,
2. $|\hat{\mathcal{R}}_{i+1}| \leq |\,(V/\mathcal{R}_g)\,|^2$, and
3. $|(\Gamma/\mathcal{P}_\Gamma)| \leq |\mathcal{P}_\Gamma|$.

The cost per refinement step then becomes $O(|g|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$. □

Theorem 16 (Preprocess). *Let g be a DAG and $\mathcal{R}_g$ the coarsest simulation on g. On input g,* Preprocess *can be computed in time $O(|\mathcal{P}_\Sigma| + \mathit{height}(g)\,|g|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$.*

Proof. By Lemma 15, $\mathcal{R}_0$ can be initialised in $O(|V| + |\mathcal{P}_\Sigma|)$ computation steps, and by Corollary 10, $\mathit{height}(g)$ refinement steps are needed. Also by Lemma 15, refining the relation $\mathcal{R}_i$ requires $O(|g|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$ operations. Combining the cost of the initialisation step and the $\mathit{height}(g)$ approximation refinements we have our result. □

Theorem 17 (Search). *Let g be a DAG, t a tree and $\mathcal{R} = \{\mathcal{R}_i \mid i \in [\mathit{height}(g)]\}$ the family of approximated simulations computed by* Preprocess *on input g. The time complexity of* Search *is $O(|\mathcal{P}_\Sigma| + \mathit{height}(t)\,|t|\,|(V/\mathcal{R}_g)|^2\,|\mathcal{P}_\Gamma|)$.*

Proof (Sketch). Initially checking the height of t can be done in time $O(\mathit{height}(t))$, which is assumed to be negligible in comparison with e.g. traversing g once.

To find the occurrences $\mathit{occur}_g(t)$, the routine computes the $\mathit{height}(t)$-approximated simulation $\mathcal{R}_h$ on $g \uplus t$. Since g and t are disjoint, the initialisation of $\mathcal{R}_0$ and refinement steps can be organised so that the vertices of g and t are processed separately. Since $\mathcal{R}_h$ has already been computed on g by Search, what remains is to complete the computation of $\mathcal{R}_h$ by retracing the refinement steps done by Preprocess for the vertices in t. To find a bound of the execution time, it suffices to substitute $\mathit{height}(t)$ and $|t|$ for $\mathit{height}(g)$ and $|g|$, respectively, in the proof of Lemma 15. □

5. Restrictions and extensions

5.1. Graph structure

Recall that $\mathcal{P}_\Sigma$ and $\mathcal{P}_\Gamma$ are preorders on the vertex alphabet Σ and edge alphabet Γ, respectively. The simplest case of the general investigation is when both alphabets are trivial (or, equivalently, when both preorders are trivial). In this case, Problem 2 reduces to a purely structural problem in the theory of directed graphs. Absent considerations of the ordering of edge- and vertex labels, the following proposition can be easily proven, by considering the respective definitions of occurrence and graph isomorphism.

Propostition 18. *If t is isomorphic to a subgraph of g, then t occurs in g.*

The converse of the proposition is obviously not true, and examples of this can be found easily. If we denote by $C(g)$ the set of graphs that can be produced from g by means of the operation of identifying pairs of vertices v, v' for which there exists a w such that $(v, w) \in E_g$ and $(v', w) \in E_g$ and removing multiple edges to keep the graph simple, then we can formulate the following result.

Propostition 19. *The tree t occurs in g, if and only if t is isomorphic to a subgraph of some graph in $C(g)$.*

The proof of this proposition consists in little more than translating the definition of occurrence into the terminology employed here.

5.2. Ordered trees

With non-trivial alphabets and preorders, there is more structure to work with, and so called *ordered* graphs are of particular interest for many applications. Simply put, an acyclic graph g is *ordered* if the preorder $\mathcal{P}_\Gamma$ on the edge alphabet is a total order when restricted to the outgoing edges of each and every vertex v in V_g. More formally:

Definition 20 (Ordered acyclic graph). An acyclic graph $g = (V, E, s, l)$ is *ordered* if, for every $v \in V$, $\mathcal{P}_\Gamma$ is a total order on $\{l((v, v')) \mid (v, v') \in E\}$.

To search for patterns in an ordered tree, we can use pattern trees in which every edge e is labelled with a subset of Γ. When we execute the search algorithm, we interpret an edge label γ in the object tree as the label $\{\gamma\}$ and use set specification as our preorder: Somewhat counter-intuitively, the set S dominates the set S' if $S \subseteq S'$.

Let us illustrate this principle with an example.

Example 21. Suppose that we have a corpus of syntactically annotated texts by Shakespeare, and that we want to search this corpus for the following: a sentence in which there is a verb phrase that begins with the word "are", immediately followed by a noun phrase, but which is not the first constituent of the sentence. These semantics are captured by the pattern tree in Figure 1. The sets that label its edges are given as intervals of natural numbers (augmented with an additional maximal element ∞).

Figure 2 depicts the syntax tree assigned by the Stanford parser to a quote from Shakespeare's Hamlet. For our purposes, we consider the outgoing edges of each vertex to be labelled by $\{1\}, \{2\}, \{3\}, \ldots$. The edge labels are not written out explicitly in the figure, but can be deduced from the left-to-right order of the edges; at each node the first edge from the left is labelled $\{1\}$, the second edge is labeled $\{2\}$ and so on. Under the specification preorder, the pattern tree in Figure 2 occurs at the root of the parse tree, so it it contains the syntactical fragment that we were looking for.

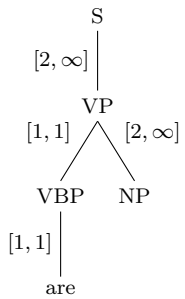

Figure 1: A pattern tree t that matches against a language of syntactical fragments.

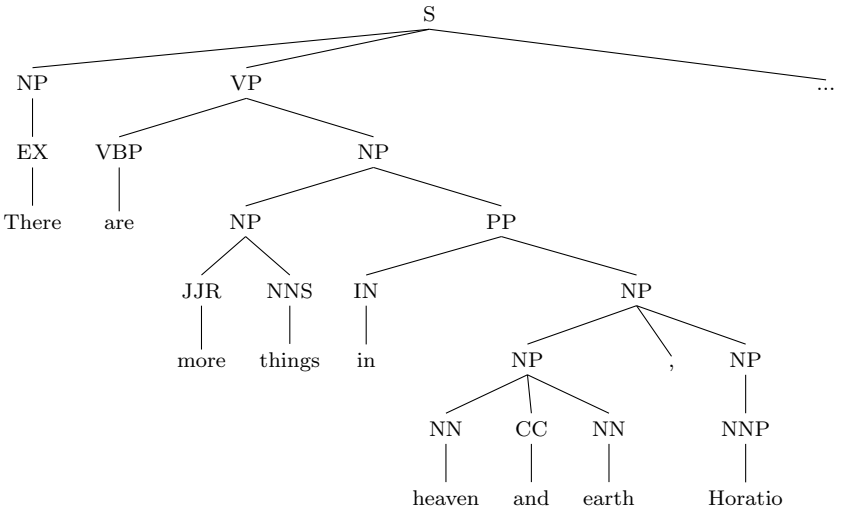

Figure 2: A syntax tree for the sentence *There are more things in heaven and earth, Horatio....*

6. Conclusion and future work

By introducing and building on a theoretical framework of approximated simulations, we obtained a search algorithm that is only affected indirectly by the size of the object treebank T, namely, through the size of the coarsest simulation relation on a DAG representation of T. The trade-off is the need for a relatively expensive preprocessing step, but this may well be justifiable if the object treebank seldom updated.

Future work will aim to combine the ideas presented in this paper with techniques to lessen the computational complexity with respect to the pattern tree. We are also interested in investigating the usefulness of our notion of occurrence for algorithmic query learning, and in lifting the results to more general families of graphs.

References

Abdulla, P. A., Bouajjani, A., Kaati, L., March 2008. Computing simulations over tree automata: Efficient techniques for reducing tree automata. In: Ramakrishnan, C. R., Rehof, J. (Eds.), 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, 2008. Springer, Heidelberg, Germany, pp. 93–108.

Baader, F., Nipkow, T., 1998. Term rewriting and all that. Cambridge University Press, New York, NY, USA.

Bille, P., June 2005. A survey on tree edit distance and related problems. Theoretical Computer Science 337 (1–3), 217–239.

Buchholz, P., 2008. Bisimulation relations for weighted automata. Theoretical Computer Science 393 (1-3), 109–123.

Chen, L., Gupta, A., Kurul, M. E., September 2005. Stack-based algorithms for pattern matching on dags. In: Böhm, K., Jensen, C. S., Haas, L. M., Kersten, M. L., Larson, P.-Å., Ooi, B. C. (Eds.), Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 2005. ACM, pp. 493–504.

Chen, W., 1998. More efficient algorithm for ordered tree inclusion. Journal of Algorithms 26, 370–385.

Chen, Y., Thurley, M., Weyer, M., 2008. Understanding the complexity of induced subgraph isomorphisms. In: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I. Springer Verlag, Heidelberg, Germany, pp. 587–596.

Cheng, J., Yu, J. X., Ding, B., Yu, P. S., Wang, H., April 2008. Fast graph pattern matching. Proceedings of the 24th International Conference on Data Engineering, Cancun, Mexico, 2008, 913–922.

Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M., 1997. Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, release October, 2002.

Droste, M., Kuich, W., Vogler, H., 2009. Handbook of Weighted Automata, 1st Edition. Springer Verlag, Heidelberg, Germany.

Fan, W., Li, J., Ma, S., Tang, N., Wu, Y., Wu, Y., 2010. Graph pattern matching: From intractable to polynomial time. In: Jagadish, H. V. (Ed.), Proceedings of the 37th International Conference on Very Large Data Bases, Seattle, WA, 2010. Vol. 3. pp. 264–275.

Kilpelinen, P., Mannila, H., 1995. Ordered and unordered tree inclusion. Siam Journal on Computing 24, 340–356.

Koehn, P., January 2010. Statistical Machine Translation. Cambridge University Press, Cambridge, England.

Kolaczyk, E. D., 2009. Statistical analysis of network data: Methods and models. Springer Series in Statistics, 386.

Lee, J. K., 2010. Statistical bioinformatics: a guide for life and biomedical science researchers. Methods of Biochemical Analysis. Wiley-Blackwell, Oxford, England.

Maletti, A., 2009. A backward and a forward simulation for weighted tree automata. In: Proceedings of the 3rd International Conference on Algebraic Informatics. CAI '09. Springer-Verlag, Berlin, Heidelberg, pp. 288–304.

Maneth, S., Mihaylov, N., Sakr, S., 2008. XML tree structure compression. In: Proceedings of the International Workshop on Database and Expert Systems Applications, Turin, Italy, 2008. IEEE Computer Society, Los Alamitos, CA, USA, pp. 243–247.

Milner, R., 1982. A Calculus of Communicating Systems. Springer Verlag, Heidelberg, Germany.

Paige, R., Tarjan, R., 1987. Three partition refinement algorithms. SIAM Journal on Computing 16 (6), 973–989.

Sundaram, G., Skiena, S. S., 1995. Recognizing small subgraphs. Networks 25, 183–191.

Ullmann, J. R., 1976. An Algorithm for Subgraph Isomorphism. Journal of the ACM 23 (1), 31–42.

Vogt, H. G., 1985. Leçons sur la résolution algébrique des équations. Vuibert et Nony, Paris, France.

Wang, J. T. L., Zhang, K., Chang, G., Shasha, D., 2002. Finding approximate patterns in undirected acyclic graphs. Pattern Recognition 35.

Wuu, H.-T. L., tzu Lu, H., Yang, W., December 2000. A simple tree pattern-matching algorithm. In: Proceedings of the Workshop on Algorithms and Theory of Computation, Chiyayi, Taiwan. pp. 1–8.

Yao, J. T., Zhang, M., 2004. A fast tree pattern matching algorithm for XML query. In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, 2004. IEEE Computer Society, Washington, DC, USA, pp. 235–241.