Tree pattern matching with a more general notion of occurrence of the pattern

Cedric Chauve

LaBRI, Université de Bordeaux I

Information Processing Letters, 82(4):197-201, 2002.

Tree pattern matching with a more general notion of occurrence of the pattern

Cedric Chauve

LaBRI, Université de Bordeaux I
351 Cours de la Libération,
33405 Talence Cedex, France

Abstract. Since the initial paper [7] of Hoffmann and O'Donnell, tree pattern matching has been studied by many authors. However, all of the papers proposing algorithms for this problem deal with a restricted notion of pattern occurrence in the subject. In this note, we consider a more general notion of occurrence and we give an extension of the top-down algorithm of Hoffmann and O'Donnell for this notion.
Keywords: design of algorithms, tree pattern matching.

1 Introduction

The tree pattern matching problem is a natural generalization of the classical string pattern matching problem (a string is a tree where every vertex has at most one child). In this problem, one considers two ordered trees with vertices labeled on an alphabet Σ, a pattern P and a subject S, and one searches all the vertices v of S admitting an occurrence of P (see [2] for a survey on this problem and its applications). The definition of an occurrence of the pattern tree in the subject tree was considered in papers about tree pattern matching as the following (we call this kind of occurrence a *compact occurrence*).

Definition 1. *Let v be a vertex of the subject S. The vertex v admits a* compact occurrence *of P if there exists a one-to-one map from the vertices of P to the vertices of S such that the root of P maps to v, and, if x maps to y then x has the same label as y and the i^{th} child of x maps to the i^{th} child of y.*

The study of the tree pattern matching problem started with the paper [7] of Hoffmann and O'Donnell, who presented the first non trivial algorithm (called the top-down algorithm in their paper) to find all the vertices of the subject admitting a compact occurrence of the pattern. However, this algorithm has a quadratic $O(n \times m)$ worst-case time complexity, where n and m are respectively the number of vertices of the subject S and of the pattern P. The first subquadratic algorithm is due to Kosaraju [9], and has been improved by Dubiner, Galil and Magen [6] (see also [5, Section 15.2]). Recently, Cole and Hariharan proposed a randomized algorithm with $O(n \times \log^3 m)$ time [3], and Cole, Hariharan and Indyk obtained the same complexity with a deterministic algorithm [4]. We can also cite the works of Luccio and Pagli, based on representing a tree with strings [10, 11].

In this paper, we are interested in the tree pattern matching problem with a more general notion of occurrence, defined below.

Definition 2. *Let v be a vertex of the subject S. The vertex v admits a* general occurrence *of P if there exists a one-to-one map from the vertices of P to the vertices of S such that the root of P maps to v, if x maps to y, then x has the same label as y, and if x and x' are two vertices of P such that x is a left sibling of x' and x (resp. x') maps to a vertex y (resp. y') of S, then y is a left sibling of y'.*

This notion of occurrence is a natural generalization of the notion of compact occurrence (every compact occurrence is a general occurrence[1]) and of the notion of occurrence of a pattern string in a subject string. However, as far as we know, there is no non trivial algorithm for this notion

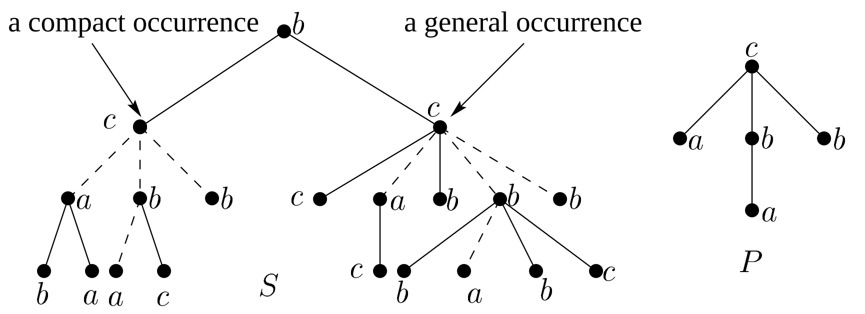

Fig. 1. The two notion of occurrences of the pattern in the subject

of occurrence: all the algorithms proposed in the literature on tree pattern matching consider the notion of compact occurrence. Indeed, either they assume some conditions on the trees S and P (such as imposing that two siblings do not have the same label) or they translate the problem in terms of binary trees. (It is easy to see that, using the classical encoding of ordered trees by binary trees [8], the notion of compact occurrence has a natural expression in terms of binary trees, unlike the notion of general occurrence).

In this paper, we propose an extension of the top-down algorithm of Hoffmann and O'Donnell (which is itself a generalization of the classical Aho-Corasick algorithm for multi-pattern matching in strings [1]) allowing all general occurrences of the pattern in the subject to be found in time $O(n \times l(P))$, where $l(P)$ is bounded by the number of leaves of P, after $O(n + m \times |\Sigma|)$ time and space preprocessing and with $O(n + m \times |\Sigma|)$ extra space.

2 The algorithm

Preprocessing of the subject. We assume that for every vertex x of S, we have computed its DFS (Depth First Search) number $\text{num}(x)$ ($\text{num}(x) = i$ if x is the i^{th} vertex visited during a DFS traversal of S). This information can be computed for all the vertices of S in $\Theta(n)$ time and space.

Preprocessing of the pattern. First, we associate to every leaf x of P a triple $(\text{lnum}(x), \text{height}(x), \text{nca}(x))$ of integers defined as follows (see Figure 2):

- if x is the k^{th} leaf visited during a DFS traversal of P, then $\text{lnum}(x) = k$ (x is called the k^{th} leaf of P),
- $\text{height}(x)$ is the number of edges on the path from x to the root of P,
- if x is the first leaf of P then $\text{nca}(x) = 0$, else (x is the k^{th} leaf of P, $k > 1$) $\text{nca}(x)$ is the number of edges on the path from x to the nearest common ancestor of x and the $(k-1)^{th}$ leaf of P.

Notation. Suppose vertex x is an ancestor of vertex y in tree T. $W_T(x, y)$ denotes the word corresponding to the labels of the vertices on the path from x to y (including x and y).

For the next part of the preprocessing of P, we compute the deterministic automaton recognizing the set of words $\{W_P(r_P, f_1), \ldots, W_P(r_P, f_l)\}$ where r_P is the root of P, f_i is the i^{th} leaf of P and l the number of leaves of P. This construction is based on a simple transformation of P. Moreover, we augment this automaton with the following information (see Figure 3):

- we associate to every final state s the list of triples associated with the leaves f of P such that reading $W_P(r_P, f)$ in the automaton leads to the state s ;

[1] We should notice that computing the vertices admitting a general occurrence of the pattern in the subject does not give in general the number of occurrences, which differs from the similar problem with compact occurrences.

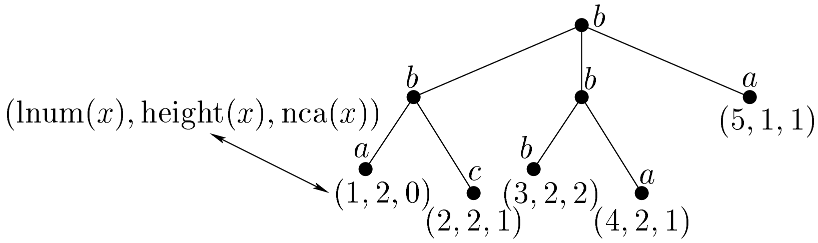

Fig. 2. Information associated to the leaves of the pattern

- for a symbol a of Σ and a state s such that no transition labeled a leaves s, we add a transition, labeled with a, from s to the state s' recognizing the longest proper suffix[2] of the word $w(s).a$, where $w(s)$ is the word recognized by the state s ;
- we add a *final transition* to every final state s to the final state s' recognizing the longest proper suffix of $w(s)$ (to the initial state if such a state s' does not exist).

We denote this automaton by ACA(P). In the following figure, the initial state is displayed with an empty circle, the final states with squares and the final transitions by dashed lines.

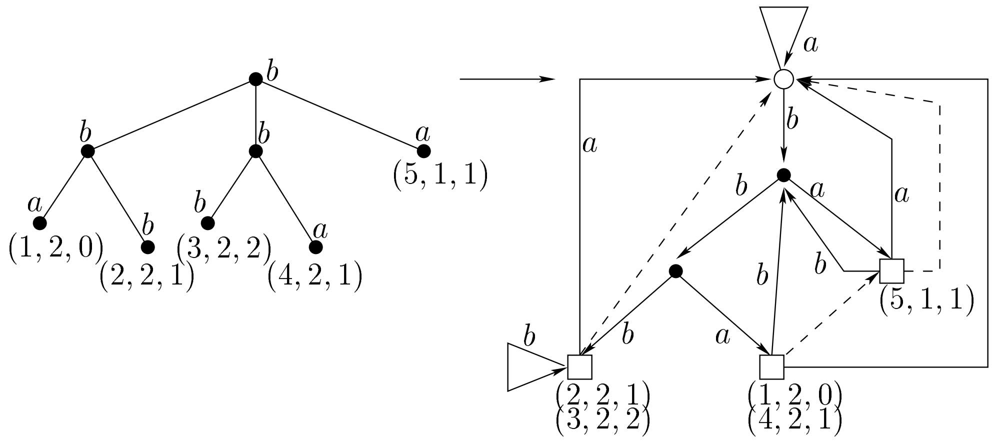

Fig. 3. A tree P and the automaton ACA(P)

The pattern matching phase. For a tree T having l leaves and an integer $1 \leq i \leq l$, we call the i^{th} prefix of T, denoted by T_i, the restriction of T to the vertices visited before the i^{th} leaf during a DFS traversal of T (including the i^{th} leaf). Our pattern matching algorithm is based on the following lemma relating the occurrence of two consecutive prefixes of the pattern in the subject.

Lemma 1. Let x be a vertex of S admitting a general occurrence of P_i, y be the descendant of x such that the leaf f_i of P maps to y in this occurrence, z be a descendant of x such that $W_S(x, z) = W_P(r_P, f_{i+1})$, z' be the $\text{nca}(f_{i+1})^{th}$ ancestor of z and z'' be the child of z' on the path from z' to z.
Then x admits a general occurrence of P_{i+1} such that f_i maps to y and f_{i+1} maps to z if and only if

$$\text{num}(z') < \text{num}(y) < \text{num}(z''). \tag{1}$$

[2] A proper suffix of a word w is a suffix of w different from w.

Proof. With the hypothesis assumed in the lemma, it is easy to check that x admits a general occurrence of P_{i+1} such that f_i maps to y and f_{i+1} maps to z if and only if z' is a common ancestor of y and z and the path from z' to y goes through a left sibling of z''. These two conditions are clearly equivalent to inequalities (1).

The principle of the pattern matching phase is to process S in DFS order reading at the same time the labels of the visited vertices in ACA(P) (after visiting the vertex x of S, we are on the state s of ACA(P) recognizing the word $W_S(r_S, x)$, where r_S is the root of S).

It will maintain an additional data structure for the current path of the DFS, storing the vertices of this path and the associated states of the automaton in an array in path order, so as to allow the i^{th} ancestor of the current vertex to be accessed in $O(1)$ time. The key information stored with each vertex v on the path is the largest index j such that v admits an occurrence of P_j with f_j within the subtree of v that has already been explored (recall that f_j is the j^{th} leaf of P). To allow this information to be updated quickly as the current vertex is processed, a second field is needed, denoted k_j, or k for short ; in the occurrence of P_j at x, k is the DFS number of the vertex mapping to f_j.

Suppose now that ACA(P) enters a final state s while we are visiting vertex z of S. Then for each leaf f_i of P that this final state could correspond to (s recognizes $W_P(r_P, f_i)$), a check is made to see if for the corresponding ancestor x of z (we follow the notations of Lemma 1), z provides the next leaf in the candidate match of P at x. More specifically, suppose one of the triples for this final state is (i, h_1, h_2). Then we are seeking to check if x has matched P_{i-1} so far, and if so, whether z corresponds to f_i in this candidate match. Let z' (resp. z'') be the h_2^{th} (resp. $(h_2-1)^{th}$) ancestor of z. Following (1) (Lemma 1), we have to check the condition $\mathrm{num}(z') < \mathrm{num}(y) < \mathrm{num}(z'')$, where $\mathrm{num}(y)$ is the record k associated with vertex x (x is exactly the h_1^{th} ancestor of the current vertex z) in the path array described above. If the condition holds, the record associated with x is updated correspondingly.

The pattern matching algorithm follows, where δ denote the transition function of ACA(P), γ the final transition function for its final states, and, for a vertex v of S, label(v) denotes the label of v.

Algorithm 1: Pattern matching algorithm

Data : ACA(P), S

begin

let init be the initial state of ACA(P) and r_S the root of S;
$top := 1,\ s := \delta(\text{init}, \text{label}(r_S))$;
$ST[top] := (r_S, 0, 0, s)$;
while $top > 0$ **do**
  $(v, j, k, s) := ST[top]$;
  if all the children of v have been visited **then**
    $top := top - 1$;
    if j is equal to the number of leaves of P **then**
      v admits a general occurrence of P;
  else
    let z be the first unvisited child of v;
    $top := top + 1$ and $s := \delta(s, \text{label}(z))$;
    $ST[top] := (z, 0, 0, s)$;
    if s is a final state of ACA(P) **then**
      while $s \neq$ init **do**
1   **for** every triple (i, h_1, h_2) associated with the state s **do**
          $(x, j, k, s_x) := ST[top - h_1]$;
          $(z', j', k', s') := ST[top - h_2]$;
          $(z'', j'', k'', s'') := ST[top - h_2 + 1]$;
          if $(j = i - 1)$ and $(j = 0$ or $\text{num}(z') < k < \text{num}(z''))$ **then** $ST[top - l_1] := (x, j + 1, \text{num}(z), s_x)$;
        $s := \gamma(s)$;

end

Proposition 1. *Let $l(P)$ be the size of the greatest set of leaves $\{f_{i_1}, \ldots, f_{i_k}\}$ of P such that for every $j \in [k-1]$, $W_P(r_P, f_{i_j})$ is a suffix of $W_P(r_P, f_{i_j+1})$. After a preprocessing in $O(n + m \times |\Sigma|)$ time and space, Algorithm 1 finds the vertices of S admitting a general occurrence of P in $O(n \times l(P))$ time, using $O(n + m \times |\Sigma|)$ extra space.*

Proof. Except for the list of triples associated to the final states, the automaton ACA(P) is identical to the automaton defined by Hoffmann and O'Donnell (see [7]) and can be constructed in $O(m \times |\Sigma|)$ time and space. The computation of the DFS number of every vertex of S is done in $\Theta(n)$ time. The extra space is due to the automaton ACA(P) and to the stack ST, that contains, at every time, at most n elements (in fact $h(S)$ elements, where $h(S)$ is the height of S). The $O(n \times l(P))$ time complexity of the pattern matching phase follows from the fact that the time spent when visiting a vertex z of S is proportional to the number of iterations of the **for** loop marked 1 (all the other operations being done in constant time), which is at most $l(P)$. The fact that this algorithm computes all the occurrences of the pattern in the text follows immediately from Lemma 1.

3 Concluding remarks

In this note, we propose an algorithm for the tree pattern matching problem with a more general notion of occurrence than in the previous literature on this problem. However our algorithm does not break the quadratic time bound of the naive algorithm (because $l(P)$ can be $O(n)$). It is still an open problem to break this bound for this notion of occurrence.

Acknowledgements We thank two anonymous referees whose comments greatly improved the writing of this note.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Comm. ACM*, 18:333–340, 1975.
2. A. Apostolico and Z. Galil, editors. *Pattern matching algorithms.* Oxford University Press, New York, 1997.
3. R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. In *ACM Symposium on Theory of Computing, STOC'97*, pages 66–75. Assoc. Comput. Mach. Press, 1997.
4. R. Cole, R. Hariharan, and P. Indyk. Tree pattern matching and subset matching in deterministic $O(n \log^3 n)$-time. In *ACM-SIAM Symposium on Discrete Algorithms, SODA'99*, pages 245–254. Assoc. Comput. Mach. Press, 1999.
5. M. Crochemore and W. Rytter. *Text algorithms.* Oxford University Press, New York, 1994.
6. M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. *J. Assoc. Comput. Mach.*, 41(2):205–213, 1994.
7. C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *J. Assoc. Comput. Mach.*, 29(1):68–95, 1982.
8. D. E. Knuth. *The art of computer programming.* Addison-Wesley Publishing Company, Reading (MA), third edition, 1997. Volume 1: Fundamental algorithms.
9. S. R. Kosaraju. Efficient tree pattern matching. In *IEEE Symposium on Foundations of Computer Science, FOCS'89*, pages 178–183. IEEE Comput. Soc. Press, 1989.
10. F. Luccio and L. Pagli. An efficient algorithm for some tree matching problems. *Inform. Process. Lett.*, 39(1):51–57, 1991.
11. F. Luccio and L. Pagli. Approximate matching for two families of trees. *Inform. and Comput.*, 123(1):111–120, 1995.