SIMPLE SOLUTIONS FOR APPROXIMATE TREE MATCHING PROBLEMS

Fabrizio Luccio **Linda Pagli**
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, 56100 Pisa, Italy

1. Introduction

Tree isomorphism and matching is widely studied. Aside from its mathematical interest, the problem has a number of important motivations in the theory of programming [4] and in molecular biology [9]. Three recent papers in this field are particularly innovative by a computational point of view. Given a pattern tree P and a text tree T, Mäkinen [7] shows how to determine all the exact occurrences of P as subtree of T, in time O(|P|+|T|), using a proper string representation for trees, and applying a fast string matching algorithms on such a string. In [5] Kosaraju solves a generalization of the same problem, generally called *tree pattern matching*, where the leaves of P may match with subtrees of T (instead of single nodes). The time required by the algorithm of [5] is $O(|T||P|^{0.75}\text{polylog}|P|)$ against the O(|T||P|) of the naive bound. In fact the algorithm of [5] solves the more general problem of determining whether P is isomorphic to a subtree T' of T, where an arbitrary number of subtrees may be cancelled from T'. The result of [5] has been improved in [1], to $O(n\sqrt{m}\ \text{polylog}(m))$. Zhang and Shasha [9] measure the edit distance between ordered trees (weighted number of insert, delete and modify operations to trasform one tree into another), and consider several approximate tree matching problems based on this distance, or other weighted distances derived from subtree removal or pruning. Due to the intrinsic difficulty of their problems, all their algorithms run in more than linear time. Among the classical contributions, Hoffmann and O'Donnell [4] propose several algorithms for the subtree replacement problem in an ordered ranked tree. Unless for special cases, the algorithms of [4] are at least quadratic in the size of the input.

In this paper we consider ordered h-ary trees, that is trees whose nodes have exactly h sons; and arbitrary ordered trees. The former appear, for example, in biochemical structures such as glycogen [8]; the latter are encountered in term rewriting systems and in RNA comparisons. We define the subtree distance between two ordered h-trees T_1, T_2 as the number of subtrees to be inserted or deleted in T_1 to obtain T_2, and introduce an approximate tree matching problem (Problem 1), that consists of finding all the occurrences of P in T, with bounded distance k. On one side, this problem is an extension of the general problem solved in [5], where P may contain subtrees which are not present in T (note however the h-ary restriction on P and T). On another side, it is an extension of the approximate string matching problem [2,6] to h-ary trees. We give an algorithm to solve this problem in time O(h|P| + max(h,k)|T|).

For arbitrary ordered trees we solve an extension of the tree pattern matching problem. We define the leaf distance between two trees T_1, T_2 as the total number of subtrees to be inserted in T_1 in place of its leaves, or to be deleted from T_1, leaving leaves in their place, to obtain T_2.

Our new problem (Problem 2) consists of determining all the occurrences of P as a subtree of T, with bounded distance k. Note that, unlike in the tree pattern matching problem, P is not necessarily contained in T. The complexity of our algorithm is $O(|P|+k|T|)$. With an obvious restriction, the algorithm can be adapted to solve the tree pattern matching problem with the same complexity. If k is chosen independently of $|P|$, as it is normally assumed in approximate string matching, our algorithm compares favourably with the previous ones.

Problems 1 and 2 are defined for unlabelled trees; the extension to labelled trees require minor modifications of the algorithms. If up to k_1 label differences are allowed between the matching nodes of P and T, the two problems are solved in time $O(h|P|+\max(h, k, k_1)|T|)$, and $O(|P|+\max(k, k_1)|T|)$, respectively.

2. Matching h-ary trees

Our study begins with a definition of tree distance. Let an ordered h-ary tree T be completed by the insertion of $(h-1)|T|+1$ *external* nodes, in place of the empty sons of the original (*internal*) nodes. If $|T| = 0$ (i.e., $T = \emptyset$) one external node is placed at the root. For a node $N \in T$, N internal or external, let T[N] denote the subtree of T rooted at N. We define the following operations on T:

1) for $N \in T$, N external, *insert* a new tree T_1 in T as a subtree rooted at N. This is denoted by: Ins $(T_1,T[N])$;
2) for $N \in T$, N internal, *delete* T[N] from T (i.e., substitute T[N] with an external node). This is denoted by: Del (T[N],T).

The *transformation* from T_1 to T_2, denoted by $T_1 \alpha_S T_2$, is a sequence S of insert or delete operations on T_1. In this transformation we establish a correspondence between pairs of internal nodes $N_1 \in T_1$, $N_2 \in T_2$, denoted by $N_1 \equiv N_2$. Let R_1,R_2 be the roots of T_1,T_2, respectively. The sequence S is such that:

1) if $T_1=T_2=\emptyset$ then $S=\emptyset$;
2) if $T_1=\emptyset$ (R_1 is an external node) and $T_2 \neq \emptyset$, then $S = \text{Ins}(T_2,T_1[R_1])$;
3) if $T_1 \neq \emptyset$ and $T_2=\emptyset$, then $S = \text{Del}(T_1[R_1],T_1)$;
4) if $T_1 \neq \emptyset$ and $T_2 \neq \emptyset$, then $R_1 \equiv R_2$ and $S = S_1 \dots S_h$, where S_i defines the transformation: $T_1[(i\text{-th})\text{son}(R_1)] \ \alpha_{Si} \ T_2[(i\text{-th})\text{son}(R_2)]$, $1 \leq i \leq h$.

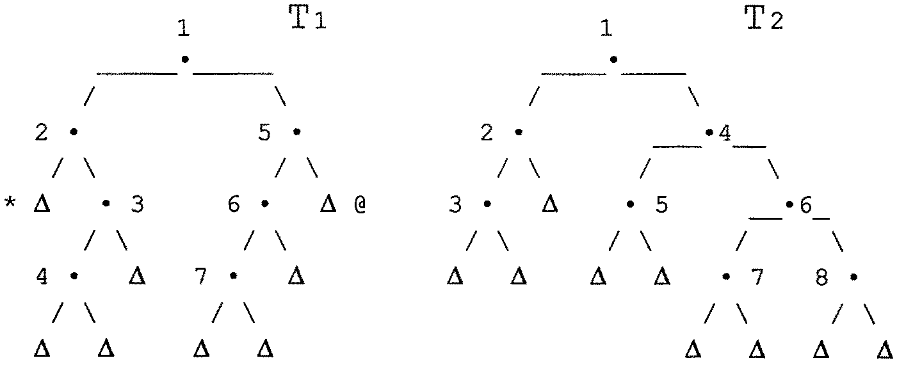
Note the transformation from T_1 to T_2 is uniquely determined. We can now pose:

Definition 1. For two ordered h-ary trees T_1, T_2, with $T_1 \alpha_S T_2$, the *subtree dinstance* $d_s(T_1,T_2)$ is given by $|S|$.

For h=2, consider the binary trees T_1, T_2 of fig.1 (external nodes are denoted by Δ). The transformation sequence is the following:

$S = \text{Ins}(T_2[3],T_1[*]), \text{Del}(T_1[3],T_1), \text{Del}(T_1[7],T_1), \text{Ins}(T_2[6],T_1[@])$.

We then have $d_s(T_1,T_2) = 4$, while the node correspondences $1\equiv1$, $2\equiv2$, $5\equiv4$, $6\equiv5$ are established.



position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
					Z'												
Z_1	1	1	0	1	1	0	0	0	1	1	1	0	0	0	0		
$LAST_1$	15	8	0	8	7	0	0	0	15	14	13	0	0	0	0		
Z_2	1	1	1	0	0	0	1	1	0	0	1	1	0	0	1	0	0
$LAST_2$	17	6	5	0	0	0	17	10	0	0	17	14	0	0	17	0	0

Fig. 1

An ordered h-ary tree T can be represented with a binary *Zaks sequence* Z, obtained by labelling the internal nodes of T by 1, and the external nodes by 0, and then visiting T in preorder [7]. Z has n ones and (h-1)n+1 zeroes, and no prefix of Z has n' ones and (h-1)n'+1, n'<n. Moreover, each subtree T' of T corresponds to a subsequence Z' of Z, which is the Zaks sequence for T'. See for example fig.1, where Z_1 and Z_2 are the Zaks sequences for T_1 and T_2, respectively. The Zaks subsequence Z' of Z_1 corresponds to $T_1[2]$.

We now pose:

Problem 1. Given two ordered h-ary trees P, T, and an integer $k\geq0$, find all the nodes $N\in T$ such that $d_s(P,T[N]) \leq k$.

For k=0 and h=2 this problem becomes the well known subtree isomorphism problem between ordered binary trees. For k>0 we have an *approximate* tree matching problem. As noted in [9], such a problem can be seen as a generalization of the approximate string matching problem applied to trees.

We solve problem 1 using the Zaks representation of the trees, and a suffix tree built on it.

Recall that, for a given sequence X of n elements, the suffix tree on X allows to compute, in time O(1), the lenght PREFIX(X, i, j) of the longest common prefix between the subsequences X(i),...,X(n) and X(j),...,X(n) [3].

Let Z(i) denote the i-th element of a Zaks Z, and N_i be the tree node corresponding to Z(i). Define the new sequence LAST, with |LAST| = |Z|, as follows. For Z(i)=0 (N_i external), LAST(i)=0. For Z(i)=1 (N_i internal), LAST(i)=j, where j is the last position in Z of the subsequence Z' corresponding to the subtree rooted at N_i. For example, the sequences $LAST_1$, $LAST_2$ for the trees T_1, T_2 are shown in fig.1. Similarly define the sequence PROX, with |PROX| = |Z|, as follows. if Z(i)=0, then PROX(i) = 0; if Z(i)=1, then PROX(i) = j, where j>i is the minimum value for which Z(j)=1 (if Z(j) = 0 for all j>i, then PROX(i) = |Z|+1).

Problem 1 can be solved with the following method, consisting of a preprocessing phase followed by algorithm 1:

Preprocessing

1. build the Zaks sequences Z_P, Z_T for P and T;
2. build the LAST sequences $LAST_P$, $LAST_T$ for P and T;
3. build the sequence PROX for T;{note: PROX contains pointers to the successive nonvoid subtrees, in preorder};
4. build the suffix tree for the string $X=Z_T\$Z_P\&$ ($,& are markers).

```
Algorithm 1
   t:=1;
   while t≤|Z_T| do
   begin
      j:=1; i:=t; d:=0;
      while i≤ LAST_T(t) and j≤|Z_P| and d≤k do
         if Z_T(i)=Z_P(j)
1:       then begin r:=PREFIX(X,i,j+|Z_T|+1);
                    i:=i+r; j:=j+r end
2:       else if Z_T(i)=0 and Z_P(j)=1
            then begin i:=i+1; j:=LAST_P(j)+1; d:=d+1 end
            else begin i:=LAST_T(i)+1; j:=j+1; d:=d+1 end;
      if d≤k then PRINT(t); {we have d_s(P,T[t])≤k}
      t:=PROX(t)
   end
end algorithm 1.
```

Theorem 1. Problem 1 is solved in time O(h·|P|+max(h,k)·|T|)), using preprocessing and algorithm 1.

Proof. *Correctness.* Let N_t be the node of T corresponding to $Z_T(t)$. The inner **while** loop of algorithm 1 computes the distance between P and $T[N_t]$. The outer **while** loop iterates such

a computation to all the nonvoid subtrees of T, pointed by the function PROX[t].
Complexity. Preprocessing. Steps 1 and 2 can be executed in time $O(h(|P|+|T|))$ (size of the output from preprocessing), since Z_P, Z_T, $LAST_P$ and $LAST_T$ can be directly built visiting the two trees. Step 3 amounts to scanning Z_T, hence takes time $O(h|T|)$. Step 4 takes time $O(h(|P|+|T|))$ [3]. The total time of preprocessing is therefore $O(h(|P|+|T|))$. To analyze algorithm 1, consider the inner **while** loop first. The **else** statement, labelled 2, is executed at most k+1 times (at this point the while is terminated). When a **then** statement, labelled 1, is executed, the function PREFIX bring the indices i, j to two values for which $Z_T(i) \neq Z_P(j)$. That is, at the next **while** iteration, the **else** statement is again executed. Therefore, the **then** statement is executed at most as many times as the **else** statement. Since PREFIX can be computed in $O(1)$, the total time required by the inner **while** loop is $O(k)$, for $k>0$. For $k=0$ the **while** loop is executed exactly once, with a single computation of PREFIX. Each iteration of the outer loop performs a comparison between P and a subtree of T, and this comparison is repeated for all the internal nodes of T. Therefore, the outer loop is executed $O(|T|)$ times, and the overall time complexity of algorithm 1 is $O(k|T|)$ for $k>0$, or $O(|T|)$ for $k=0$. []

If the internal nodes of the two trees are labelled, another distance may be considered in addition to d_s. Let Σ be a finite alphabet, $0 \notin \Sigma$, and let $\lambda(N) \in \Sigma$ be the label of a node N. Disregarding the labels, consider the transformation $T_1 \; \alpha_s \; T_2$, and let $A_1,\dots,A_h \in T_1$ and $B_1,\dots,B_h \in T_2$ be the ordered sets of nodes put into correspondance in the trasformation. That is $A_i \equiv B_i$, $1 \leq i \leq h$. (Note that A_i and B_i belong to the *common* portion of T_1 and T_2). We pose:

Definition 2. For two labelled ordered h-ary trees T_1, T_2, the *mismatch distance* $d_m(T_1, T_2)$ is given by the number of pairs A_i, B_i such that $\lambda(A_i) \neq \lambda(B_i)$.

Problem 1'. Given two labelled ordered h-ary trees P,T, and two integers k, $k_1 \geq 0$, find all the nodes $N \in T$ such that $d_s(P,T[N]) \leq k$, $d_m(P,T[N]) \leq k_1$.

Problem 1' can be solved with a variant of algorithm 1, using *the labelled Zaks sequences* $Z_{\lambda T}$, $Z_{\lambda P}$, that is, Zaks sequences with the 1's replaced by the internal node labels; and taking into account that $Z_{\lambda T}(i) \neq Z_{\lambda P}(j)$ may occur for a label mismatch. With an argument similar to the one used in theorem 1, it can be easily shown that the inner **while** loop of the algorithm is executed at most $2(\max(k, k_1)+1)$ times. Therefore we have:

Corollary 1. Problem 1' can be solved in time $O(h \cdot |P| + \max(h, k_1, k_2) \cdot |T|)$.

3. Matching arbitrary ordered trees

If the node degree is arbitrary, the transformation of a tree into another by inserting and deleting subtrees becomes less obvious. For a nonvoid ordered tree T, and a node
$N \in T$, we newly define these operations as follows:

1) for $N \in T$, N leaf, ***insert*** a non void tree T_1 in T in the place of N. Denoted by: $Ins(T_1,T[N])$;

2) for $N \in T$, N non leaf, *delete* T[N] from T by replacing T[N] with the single node N (N becomes a leaf). Denoted by Del (T[N],T).

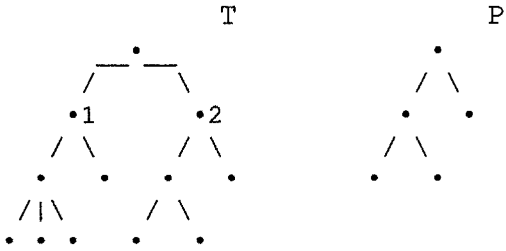
Note that we do not use now the concept of extenal and internal nodes. Let a sequence S of insert and delete operations determine the transformation of a tree T_1 into another treeT_2, denoted by $T_1 \, \beta_S \, T_2$. Letting R_1, R_2 be the roots of T_1,T_2, respectively, S is such that:

1) $R_1 \equiv R_2$,
2) if $|T_1|=|T_2|=1$ then $S=\emptyset$;
3) if $|T_1|=1$ and $|T_2|>1$, then $S = \mathrm{Ins}(T_2,T_1[R_1])$;
4) if $|T_1|>1$ and $|T_2|=1$, then $S = \mathrm{Del}\,(T_1[R_1],T_1)$;
5) if $|T_1|>1$ and $|T_2|>1$, and deg R_1 = deg R_2 = h, then $S = S_1 \ldots S_h$, where S_i defines the transformation: $T_1[(\text{i-th})\mathrm{son}(R_1)] \, \beta_{S_i} \, T_2[(\text{i-th})\mathrm{son}(R_2)]$, $1 \le i \le h$.

S is *undefined* if $|T_1|>1$, $|T_2|>1$, and deg $R_1 \neq$ deg R_2; or $|T_1|>1$, $|T_2|>1$, and deg R_1= deg R_2, but one of the sequences S_i of point 5 is undefined. In this case β_S does not exist, that is, T_1 cannot be transformed into T_2 by insert or delete operations. If S is defined, the transformation is uniquely determined. We now pose:

Definition 3. For two ordered trees T_1 , T_2, with $T_1 \, \beta_S \, T_2$, the *leaf distance* $d_l(T_1,T_2)$ is given by $|S|$.

In the sample trees T, P of fig.2 we have $d_l(T,P)=2$, $d_l(T[2],P)=0$, while $d_l(T[1],P)$ is undefined.



pos.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
W_T	1	1	1	1	0	1	0	1	0	0	1	0	0	1	1	1	0	1	0	0	1	0	0	0
LST_T	24	13	10	5	0	7	0	9	0	0	12	0	0	23	20	17	0	19	0	0	22	0	0	0
PROX	2	3	4	6	0	8	0	11	0	0	14	0	0	15	16	18	0	21	0	0	25	0	0	0
W_P	1	1	1	0	1	0	0	1	0	0														
LST_P	10	7	4	0	6	0	0	9	0	0														

Fig. 2

An ordered tree T can be represented by a binary sequence *W*, obtained visting T in preorder. A 1 is entered in W for each node encountered in the visit, and a 0 is entered for each return to the previous level. The syntactic stucture of the sequence W is the following:

W ::= 1 W* 0

where a term W occurs for each subtree of the root. In fact, each subtree T' of T corresponds to a subsequence W' of W. Clearly W has the following properties:

1. W has |T| ones and |T| zeroes;
2. all prefixes of W have a number of ones greater than the number of zeroes.

For example, see the sequences W_T, W_P for the trees T and P reported in fig.2. An easy inductive argument shows that here is a one to one correspondence between ordered trees and sequences W. We denote by W(i) the i-th position of W, and define the sequence LAST as for problem 1 (see fig.2).

We then pose:

Problem 2. Given two ordered trees P, T, and an integer $k \geq 0$, find all the nodes $N \in T$ such that $d_l(P,T[N]) \leq k$.

If we compute d_l excluding case 4 from the definition of S (that is we consider S to be undefined for $|P|>1$ and $|T|=1$), and take $k=|P|$, then problem 2 becomes the tree pattern matching defined in [5].

Problem 2 is solved with the preprocessing phase and algorithm 2 reported below (refer to the example of fig.2). We have:

Theorem 2. Problem 2 is solved in time $O(|P|+k \cdot |T|))$, using preprocessing and algorithm 2.

Proof. *Correctness.* The inner **while** loop of algorithm 2 cheks the matchings of P with a subtree of T. A point of mismatch, e. g. $W_T(i)=1$, $W_P(j)=0$, is reached via a PREFIX or a LAST computation. In both cases, $W_T(i-1)=W_P(j-1)$. If these carachters are 0, we are attempting to matching internal nodes of P and T of different degrees. Hence P is not found (defined=**false**). If $W_T(i-1)=W_P(j-1)=1$, then $W_T(i-1)$ corresponds to an internal node N of T and $W_P(j-1)$ corresponds to a leaf L of P. Then, the process goes on matching the subtree routed in N with the leaf L, and increasing d. The case $W_T(i)=0$, $W_P(j)=1$ is symmetrical. The outer **while** loop iterates the procedure, matching P with to all the nonvoid subtrees of T.

Complexity. Preprocessing takes time $O(|P|+|T|)$ (see the proof of theorem 1). The inner **while** loop of algorithm 2 is executed at most 2k+1 times. In fact, $\leq k+1$ times for the common subsequences starting at $W_T(i)=W_P(j)$, interleaved with the determinations of $\leq k$ mismatches between a leaf and a subtree. If a forbidden mismatch occurs (defined=**true**) the loop is forcibly terminated. Each iteration of the outer **while** loop performs a comparison between P and a subtree of T, hence this loop is executed $O(|T|)$ times. The complexity of algorithm 2 is then $O(k|T|)$. □

We can consider labelled ordered trees, extend the definition of mismatch distance d_m (Definition 2) to this case, and apply an obvious extension of algorithm 2. We have:

Problem 2'. Given two labelled ordered P,T, and two integers $k, k_1 \geq 0$, find all the nodes $N \in T$ such that $d_l(P,T[N]) \leq k$, $d_m(P,T[N]) \leq k_1$.

Corollary 2. Problem 2' can be solved in time $O(|P|+\max(k, k_1) \cdot |T|)$.

Preprocessing

1. build the sequences W_P, W_T, $LAST_P$, $LAST_T$ for P and T and the sequence PROX for T;
2. build the suffix tree for the string $X=W_T\$W_P\&$ (\$,& are markers).

```
Algorithm 2
  t:=1;
  while t≤|$W_T$| do
  begin
    j:=1; i:=t; d:=0; defined:=true;
    while i≤$LAST_T$(t) and j≤|$W_P$| and d≤k and defined do
      if $W_T$(i)=$W_P$(j)
      then begin r:=PREFIX(X,i,j+|$W_T$|+1);
                 i:=i+r;j:=j+r end
      else if $W_T$(i)=1 and $W_P$(j)=0
        then if $W_T$(i-1)=1
          then begin i:=$LAST_T$(i-1)+1;j:=j+1;d:=d+1 end
          else defined:=false
        else if $W_P$(i-1)=1        {We have $W_T$(i)=0 and $W_P$(j)=1}
          then begin j:=$LAST_P$(j-1)+1;i:=i+1;d:=d+1 end
          else defined:=false;
    if d≤k and defined then PRINT(t);  {we have $d_I$(P,T[t])≤k}
    t:=PROX(t)
  end
end algorithm 2.
```

Acknowledgment.

This work has been partially supported by M.U.R.S.T. of Italy.

References

[1] M. Dubiner, Z. Galil and E. Magen, Faster tree pattern matching, *Proc. 31-st IEEE Symp. on Found. of Comp. Sc.*(1990) 145-149.

[2] Z. Galil and R. Giancarlo, Data structures and algorithms for approximate string matching, *J. Complexity* **4** (1988) 33-72.

[3] D. Harel and R.E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comp.* **13** (1984) 338-355.

[4] C.M. Hoffmann and M.J. O'Donnell, Pattern matching in trees, *J. ACM* **29** (1982) 68-95.

[5] S.R. Kosaraju, Efficient tree pattern matching, *Proc. 30-th IEEE Symp. on found. of Comp. Sc.*(1989) 178-183.

[6] G.M. Landau and U. Vishkin, Introducing efficient parallelism into approximate string matching and a new serial algorithm, *Proc. 18-th ACM Symp. on Theory of Comp.* (1986) 220-230.

[7] E. Mäkinen, On the subtree isomorphism problem for ordered trees, *Inf. Proc. Let.* **32** (1989) 271-273.

[8] L. Stryer, *Biochemestry*, 3-rd edition, W.H.Freeman and Co., New York, NY 1988.

[8] K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comp.* **18** (1989) 1245-1262.