Fundamenta Informaticae 56 (2003) 105–120
IOS Press

A Fast Algorithm for Optimal Alignment between Similar Ordered Trees[*]

Jesper Jansson[†]
Department of Computer Science
Lund University, Box 118
SE-221 00 Lund, Sweden
Jesper.Jansson@cs.lth.se

Andrzej Lingas
Department of Computer Science
Lund University, Box 118
SE-221 00 Lund, Sweden
Andrzej.Lingas@cs.lth.se

Abstract. We present a fast algorithm for optimal alignment between two similar ordered trees with node labels. Let S and T be two such trees with $|S|$ and $|T|$ nodes, respectively. If there exists an optimal alignment between S and T which uses at most d blank symbols and d is known in advance, it can be constructed in $O(n \log n \cdot (maxdeg)^3 \cdot d^2)$ time, where $n = \max\{|S|, |T|\}$ and $maxdeg$ is the maximum degree of all nodes in S and T. If d is not known in advance, we can construct an optimal alignment in $O(n \log n \cdot (maxdeg)^3 \cdot f^2)$ time, where f is the difference between the highest possible score for any alignment between two trees having a total of $|S| + |T|$ nodes and the score of an optimal alignment between S and T, if the scoring scheme satisfies some natural assumptions. In particular, if the degrees of both input trees are bounded by a constant, the running times reduce to $O(n \log n \cdot d^2)$ and $O(n \log n \cdot f^2)$, respectively.

1. Introduction

Let R be a rooted tree. R is called a *labeled tree* if each node of R is labeled by a symbol from a fixed finite set Σ. R is an *ordered tree* if the left-to-right order among siblings in R is given.

[*]A preliminary version of this article appeared in [3].
[†]Address for correspondence: Department of Computer Science, Lund University, Box 118, SE-221 00 Lund, Sweden

The problem of determining the similarity between two labeled trees occurs in several different areas of computer science. For example, in computational biology, methods for measuring the similarity between ordered labeled trees of bounded degree can be used in the comparison of RNA secondary structures [4, 6, 10]. The problem also occurs in evolutionary trees comparison, organic chemistry, pattern recognition, and image clustering [4, 6, 9, 14].

The similarity between two labeled trees can be defined in various ways analogous to the ways of defining the similarity between two strings [7, 9]. For example, one can look for the largest maximum agreement subtree, the largest common subgraph, the smallest common supertree, the minimum tree edit distance, *etc.* [4, 5, 6, 11, 14].

Jiang, Wang, and Zhang [4] generalized the concept of an alignment between strings to include labeled trees as follows. An *insert operation* on a labeled tree adds a new node u, labeled by a blank symbol '$-$' which does not belong to Σ. The operation either (1) turns the current root of the tree into a child of u and lets u become the new root, or (2) makes u the parent of a subset of (if the tree is unordered) or consecutive subsequence of (if the tree is ordered) children[1] of an existing node v, and u a child of v. See Figure 1.

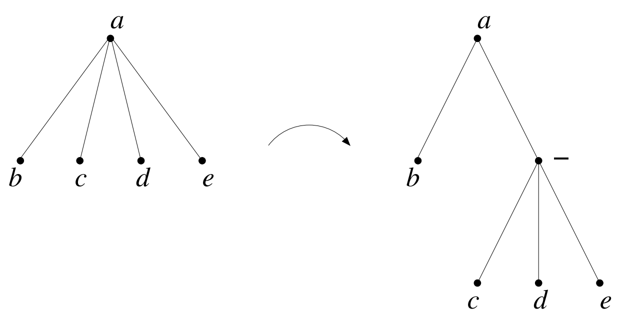

Figure 1. An insert operation of type (2). The new node becomes the parent of a consecutive subsequence of children of the node labeled by a, and then becomes a child of that node.

An *alignment between two labeled trees* is obtained by performing insert operations on the two trees so they become isomorphic when labels are ignored, and then overlaying the first augmented tree on the other one. The *score* of the alignment is the sum of the scores of all matched pairs of labels, where the score of a pair of labels is defined by a given function $\mu : (\Sigma \cup \{-\}) \times (\Sigma \cup \{-\}) \to \mathbb{Z}$. An *optimal alignment* between a pair of labeled trees is an alignment between them achieving the highest[2] possible score using μ. See Figure 2 for an example.

In [4], Jiang *et al.* presented an algorithm for computing an optimal alignment between two ordered trees S and T with node labels in $O(|S| \cdot |T| \cdot (maxdeg)^2)$ time, where $|S|$ and $|T|$ are the number of nodes in S and T, respectively, and $maxdeg$ is the maximum degree[3] of all nodes in S and T. They also provided a polynomial time algorithm for finding an optimal alignment of two *unordered* trees in case $maxdeg = O(1)$, and showed the latter problem to be MAX SNP-hard if at least one of the trees is allowed to have an arbitrary degree.

[1]Observe that subsets and consecutive subsequences can consist of zero elements.
[2]In [4], Jiang *et al.* defined an optimal alignment as one with the *lowest* possible score.
[3]The *degree* of a node is the number of its children.

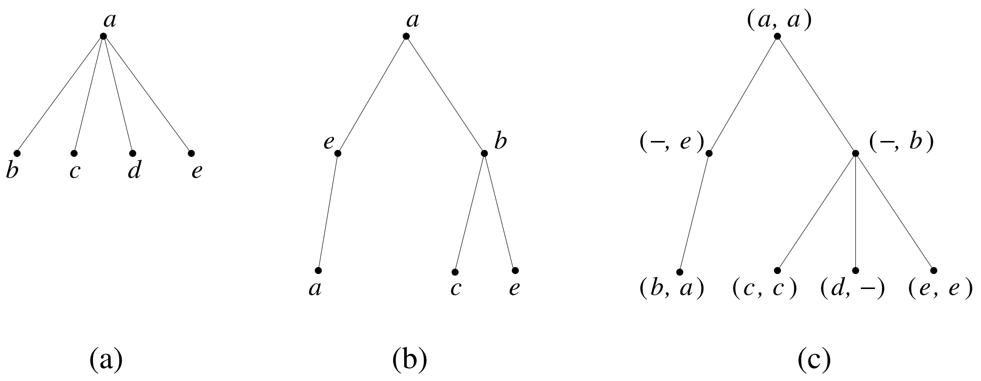

Figure 2. Let $\Sigma = \{a, b, c, d, e\}$ and define the scoring function μ as follows: for every $x, y \in \Sigma$ with $x \neq y$, let $\mu(x, x) = 3,\ \mu(x, y) = -1,$ and $\mu(x, -) = \mu(-, x) = \mu(-, -) = -2$. Then the score of the alignment in (c) between the two labeled trees shown in (a) and (b) is equal to 2.

Inspired by the known fast method for an optimal alignment between similar strings (see Section 3.3.4 in [9]), we present an algorithm for optimal alignment between two similar ordered trees with node labels which is faster than the algorithm of Jiang *et al.* when the score of an optimal alignment between the two input trees is high and the scoring scheme satisfies some natural assumptions. If there is an optimal alignment between the two input ordered trees which uses at most d blank symbols and d is specified in advance, then our algorithm called Algorithm *Fast Score* computes its score in $O(n \log n \cdot (maxdeg)^3 \cdot d^2)$ time, where $n = \max\{|S|, |T|\}$. Our more general algorithm called Algorithm *Unspecified* d for when no upper bound on d is provided computes the score of an optimal alignment in $O(n \log n \cdot (maxdeg)^3 \cdot f^2)$ time, where (assuming that the scoring scheme satisfies certain properties) f is the difference between the highest possible score for any alignment between two trees having a total of $|S| + |T|$ nodes and the score of an optimal alignment between S and T. Furthermore, if there exists an optimal alignment between S and T with b blank symbols and $O(b)$ node pairs of the form (x, y), where $x \neq y$, then (under some slightly stronger assumptions on the scoring scheme) Algorithm *Unspecified* d runs in $O(n \log n \cdot (maxdeg)^3 \cdot b^2)$ time, even if b is not known in advance. In particular, if the degrees of both input trees are bounded by a constant, the running times stated above reduce to $O(n \log n \cdot d^2)$, $O(n \log n \cdot f^2)$, and $O(n \log n \cdot b^2)$, respectively. The algorithms can be modified to return an alignment corresponding to the optimal score without increasing the asymptotic running times in the same way as for the algorithm of Jiang *et al.*

The rest of this paper is organized as follows. In Section 2, we describe the algorithm of Jiang, Wang, and Zhang from [4]. In Section 3, we introduce a new concept we call d-relevance, and in Section 4, we show how to construct d-relevant pairs of subtrees and subforests efficiently. Then, in Sections 5 and 6, we present and analyze Algorithm *Fast Score* and Algorithm *Unspecified* d.

2. The algorithm of Jiang, Wang, and Zhang

The algorithm of Jiang, Wang, and Zhang [4] for aligning two labeled, ordered trees is based on the standard dynamic programming algorithm for the string alignment problem which calculates the scores of optimal alignments between pairs of *prefixes* (or symmetrically, *suffixes*) of the two input strings in bottom-up order by using a two-dimensional table to store the computed scores, and then, when the table is complete, performs a traceback to obtain an optimal alignment (see, e.g., [2, 7, 9, 13]). The algorithm of Jiang *et al.* computes and stores the scores of optimal alignments between pairs of *ordered subtrees* of S and T and between pairs of *ordered subforests* of S and T in a bottom-up fashion. After the algorithm is finished, an actual optimal alignment between S and T can also be recovered by doing a traceback.

Some notation is necessary to describe the algorithm in more detail.

Definition 2.1. For an ordered tree S and a node u of S, let $S[u]$ denote the ordered subtree of S rooted at u (i.e., the minimal subgraph of S which includes u and all of its descendants). Let $\deg(u)$ be the degree of u, and denote the children of u by $u_1, ..., u_{\deg(u)}$ according to their left-to-right order. $S(u, i, j)$ refers to the ordered subforest $S[u_i], ..., S[u_j]$, and $S(u)$ is short for $S(u, 1, \deg(u))$. The number of nodes in a subtree or subforest S_* is denoted by $|S_*|$. Finally, $\deg(S)$ is defined as the maximum degree of all nodes in S.

Thus, $S(u)$ is the *complete ordered forest* obtained by removing u and all edges incident to u from $S[u]$. Also observe that $S(u, i, i) = S[u_i]$.

Definition 2.2. The score of an optimal alignment between two subtrees or two subforests S_* and T_* is denoted by $D(S_*, T_*)$.

To obtain a bottom-up ordering of the subtrees and subforests suitable for dynamic programming, the nodes in an ordered tree with n nodes are numbered 1 through n according to postorder so that $D(S[|S|], T[|T|])$ will contain the score of an optimal alignment between S and T. Henceforth, Θ represents the empty tree and $g_S(u)$ is the label of node number u in the labeled tree S.

The next lemma forms the basis of the algorithm of Jiang *et al.*

Lemma 2.1. Let S and T be two labeled ordered trees with $u \in S$ and $v \in T$. Then

1. $D(\Theta, \Theta) = 0$

$$D(S[u], \Theta) = D(S(u), \Theta) + \mu(g_S(u), -)\ , \qquad D(S(u), \Theta) = \sum_{q=1}^{\deg(u)} D(S[u_q], \Theta)$$

$$D(\Theta, T[v]) = D(\Theta, T(v)) + \mu(-, g_T(v))\ , \qquad D(\Theta, T(v)) = \sum_{q=1}^{\deg(v)} D(\Theta, T[v_q])$$

2. $D(S[u], T[v]) =$

$$\max \begin{cases} D(S(u), T(v)) + \mu(g_S(u), g_T(v)) \\ D(S[u], \Theta) + \max\limits_{1 \le q \le \deg(u)} \{D(S[u_q], T[v]) - D(S[u_q], \Theta)\} \\ D(\Theta, T[v]) + \max\limits_{1 \le q \le \deg(v)} \{D(S[u], T[v_q]) - D(\Theta, T[v_q])\} \end{cases}$$

3. For any j and l such that $1 \leq j \leq \deg(u)$ and $1 \leq l \leq \deg(v)$,

   $D(S(u,1,j),T(v,1,l)) =$

$$\max \begin{cases} D(S(u,1,j-1),T(v,1,l)) + D(S[u_j],\Theta) \\ D(S(u,1,j),T(v,1,l-1)) + D(\Theta,T[v_l]) \\ D(S(u,1,j-1),T(v,1,l-1)) + D(S[u_j],T[v_l]) \\ \mu(g_S(u_j),-) + \max\limits_{1 \leq q < \deg(l)} \{D(S(u,1,j-1),T(v,1,q-1)) + D(S(u_j),T(v,q,l))\} \\ \mu(-,g_T(v_l)) + \max\limits_{1 \leq q < \deg(j)} \{D(S(u,1,q-1),T(v,1,l-1)) + D(S(u,q,j),T(v_l))\} \end{cases}$$

Proof:
See [4]. □

The algorithm of Jiang *et al.* (Algorithm *Score*) is displayed in Figure 3. As the various values of $D(S_*,T_*)$ are computed using the recurrences in Lemma 2.1, they are stored in a data structure which allows them to be retrieved in $O(1)$ time from then on.

Algorithm *Score* employs an auxiliary procedure called Procedure 1 (not shown here) that takes as input two subforests of the form $S(u,i,\deg(u))$ and $T(v,k,\deg(v))$, where at least one of i and k is equal to 1, and then computes $D(S(u,i,j),T(v,k,l))$ for all j and l such that $i \leq j \leq \deg(u)$ and $k \leq l \leq \deg(v)$ by repeatedly applying Lemma 2.1.3 in a straightforward manner.

Each call to Procedure 1 is proved in [4] to take $O((\deg(u)+\deg(v)) \cdot \deg(u) \cdot \deg(v))$ time, and the total running time of the algorithm is shown to be $O(|S| \cdot |T| \cdot (maxdeg)^2)$.

Note that for every pair of subtrees $S[u]$ and $T[v]$, although the algorithm computes $D(S(u,i,j), T(v))$ for all $1 \leq i \leq j \leq \deg(u)$ and $D(S(u),T(v,k,l))$ for all $1 \leq k \leq l \leq \deg(v)$, it does *not* need to compute the values of $D(S(u,i,j),T(v,k,l))$ for all $1 \leq i \leq j \leq \deg(u)$ and $1 \leq k \leq l \leq \deg(v)$.

By adding a traceback step at the end, the algorithm can be extended to return an alignment corresponding to the optimal score without increasing the asymptotic running time[4]. Hence, Jiang *et al.* proved the following result.

Theorem 2.1. An optimal alignment between two node-labeled, ordered trees S and T can be computed in $O(|S| \cdot |T| \cdot (maxdeg)^2)$ time, where $maxdeg$ is the maximum degree of all nodes in S and T.

[4]An optimal alignment can be recovered by recalculating the terms on the right-hand side of Lemma 2.1 for each pair of subtrees or subforests encountered during the traceback to determine which of the possibilities that resulted in the highest score; alternatively, one can modify the algorithm to also record information about how each value $D(S_*,T_*)$ is obtained as it is computed, e.g., by saving pointers.

```
Algorithm  Score
Input:  Two labeled ordered trees $S$ and $T$.
Output: The score of an optimal alignment of $S$ and $T$.

  $D(\Theta, \Theta) := 0$
  for $u := 1$ to $|S|$ do
      Initialize $D(S[u], \Theta)$ and $D(S(u), \Theta)$ according to Lemma 2.1.1.
  endfor
  for $v := 1$ to $|T|$ do
      Initialize $D(\Theta, T[v])$ and $D(\Theta, T(v))$ according to Lemma 2.1.1.
  endfor
  for $u := 1$ to $|S|$ do
      for $v := 1$ to $|T|$ do
          for $i := 1$ to $\deg(u)$ do
              Call Procedure 1 on $S(u, i, \deg(u))$ and $T(v)$.
          endfor
          for $k := 1$ to $\deg(v)$ do
              Call Procedure 1 on $S(u)$ and $T(v, k, \deg(v))$.
          endfor
          Compute $D(S[u], T[v])$ as in Lemma 2.1.2.
      endfor
  endfor
  return $D(S[|S|], T[|T|])$
End Score
```

Figure 3. The algorithm of Jiang, Wang, and Zhang.

3. d-relevance

The main idea of our algorithm is to modify the dynamic programming algorithm of Jiang *et al.* outlined in Section 2 to only consider what we call d-relevant pairs of subtrees and subforests.

3.1. d-relevant pairs of subtrees

In order to introduce our slightly technical concept of d-relevance, we need some definitions.

Definition 3.1. Let S be a labeled ordered tree and u a node of S. When u is not the root of S, $\overline{S[u]}$ stands for the ordered subtree of S resulting from removing $S[u]$ and the edge between u and the parent of u from S. Next, $L(S[u])$ denotes the set of leaves in S that are to the left of the leaves of $S[u]$.

Now, we are ready to define the concept of a d-relevant pair of subtrees as well as those of a d-descendant and a d-ancestor.

Definition 3.2. Let d be a positive integer. For two ordered trees S and T containing nodes u and v respectively, the pair of subtrees $(S[u], T[v])$ is called *d-relevant* if and only if both of the following conditions hold:

1. $||S[u]| - |T[v]|| \leq d$
2. $||L(S[u])| - |L(T[v])|| \leq d$

Definition 3.3. Let d be a positive integer, and let T be an ordered tree containing two nodes v and w. $T[w]$ is called a d-*descendant* of $T[v]$ if w is a descendant of v and $|T[v]| - |T[w]| \leq d$. Symmetrically, $T[w]$ is called a d-*ancestor* of $T[v]$ if w is an ancestor of v and $|T[w]| - |T[v]| \leq d$.

The definition of d-relevance immediately yields the following lemma.

Lemma 3.1. Let S and T be two labeled ordered trees, and let u and v be two nodes in S and T respectively. If there is an alignment between S and T which uses at most d blank symbols and consists of an alignment between $S[u]$ and $T[v]$ and an alignment between $\overline{S[u]}$ and $\overline{T[v]}$ then $(S[u], T[v])$ is d-relevant for S and T.

The next three lemmas will be useful for bounding the number of d-relevant pairs from above.

Lemma 3.2. If the pairs $(S[u], T[v])$ and $(S[u], T[w])$ are d-relevant for two ordered trees S and T, and w is an ancestor (or, descendant) of v in T, then $T[w]$ is a $2d$-ancestor (or, $2d$-descendant) of $T[v]$.

Proof:
Since $(S[u], T[v])$ is d-relevant, it holds that $||S[u]| - |T[v]|| \leq d$ and hence $|S[u]| - |T[v]| \leq d$, which gives us $-|S[u]| + |T[v]| \geq -d$. Suppose that $T[w]$ is not a $2d$-ancestor of $T[v]$, i.e., $|T[w]| - |T[v]| > 2d$. Then we have $|T[w]| - |S[u]| = |T[w]| - |T[v]| + |T[v]| - |S[u]| > 2d + (-d) = d$, which contradicts the d-relevance of $(S[u], T[w])$. □

Lemma 3.3. For a node u of an ordered tree S, the number of d-ancestors of $S[u]$ is at most d.

Proof:
Assume that the number of d-ancestors of $S[u]$ is greater than d. By the pigeonhole principle there exists a d-ancestor $S[u']$ whose root u' is located at distance greater than d from u. But then $|S[u']| - |S[u]| > d$, which is a contradiction. □

Lemma 3.4. Let $\{(S[u], T[v_i])\}_{i=0}^{l}$ be a sequence of distinct d-relevant pairs in two ordered trees S and T such that for any $0 \leq i, j \leq l$, v_i is not a descendant of v_j. Then, $l \leq 2d$ holds.

Proof:
We may assume without loss of generality that the sequence is ordered according to the left-right order in T. Since $(S[u], T[v_l])$ is d-relevant, $||L(S[u])| - |L(T[v_l])|| \leq d$. On the other hand, we have $|L(T[v_l])| - |L(T[v_0])| \geq l$. Thus, if $l > 2d$ then $|L(S[u])| - |L(T[v_0])| = |L(S[u])| - |L(T[v_l])| + |L(T[v_l])| - |L(T[v_0])| > (-d) + 2d = d$, which contradicts the d-relevance of $(S[u], T[v_0])$. □

By combining the three lemmas above, we obtain an upper bound on the number of d-relevant pairs of subtrees.

Theorem 3.1. For two ordered trees S and T and a node u of S, the number of distinct d-relevant pairs of subtrees in which u participates is $O(d^2)$.

Proof:
Let $\{(S[u], T[v_i])\}_{i=0}^{l}$ be a maximal sequence of distinct d-relevant pairs of subtrees for two ordered trees S and T such that for each $0 \leq i \leq l$ there is no d-relevant pair $(S[u], T[v])$, where v is a descendant of v_i. It follows from Lemma 3.2 that for each d-relevant pair $(S[u], T[w])$, it either belongs to the sequence or $T[w]$ is a $2d$-ancestor of a member in the sequence. Hence, the number of d-relevant pairs in which u participates is at most $(2d+1) \cdot (l+1)$ by Lemma 3.3. Now, it is sufficient to observe that l cannot exceed $2d$ by Lemma 3.4. □

Corollary 3.1. There are $O(m \cdot d^2)$ d-relevant pairs of subtrees for S and T, where $m = \min\{|S|, |T|\}$.

3.2. d-relevant pairs of subforests

The algorithm of Jiang *et al.* computes scores not only between pairs of subtrees of the input trees, but also between certain pairs of subforests of the trees. Therefore, in order to modify the algorithm, we have to generalize the concepts of d-relevance, d-descendants, and d-ancestors for pairs of nodes inducing full subtrees to include pairs of subforests of the form $(S(u,i,j), T(v,k,l))$.

Definition 3.4. Let $S(u,i,j)$ be an ordered forest in an ordered tree S. When u is not the root of S, $\overline{S(u,i,j)}$ stands for the ordered subtree of S obtained by removing $S(u,i,j)$ and all edges incident to $S(u,i,j)$ from S. $L(S(u,i,j))$ denotes the set of leaves in S that are to the left of the leaves of $S(u,i,j)$.

Definition 3.5. Let d be a positive integer. For two ordered trees S and T containing nodes u and v respectively, the pair of ordered subforests $(S(u,i,j), T(v,k,l))$ is called *d-relevant* if and only if both of the following conditions hold:

1. $||S(u,i,j)| - |T(v,k,l)|| \leq d$
2. $||L(S(u,i,j))| - |L(T(v,k,l))|| \leq d$

Definition 3.6. Let d be a positive integer, and let T be an ordered tree containing two nodes v and w. $T(w,k',l')$ is called a *d-descendant* of $T(v,k,l)$ if w is a descendant of v, $T(w,k',l')$ is contained in $T(v,k,l)$, and $|T(v,k,l)| - |T(w,k',l')| \leq d$. Symmetrically, $T(w,k',l')$ is called a *d-ancestor* of $T(v,k,l)$ if w is an ancestor of v, $T(v,k,l)$ is contained in $T(w,k',l')$, and $|T(w,k',l')| - |T(v,k,l)| \leq d$.

The definition of d-relevance of subforests yields the following lemma analogous to Lemma 3.1.

Lemma 3.5. Let S and T be two labeled ordered trees, and let $S(u,i,j)$ and $T(v,k,l)$ be ordered forests in S and T respectively. If there is an alignment between S and T which uses at most d blank symbols and consists of an alignment between $S(u,i,j)$ and $T(v,k,l)$ and an alignment between $\overline{S(u,i,j)}$ and $\overline{T(v,k,l)}$ then $(S(u,i,j), T(v,k,l))$ is d-relevant for S and T.

The next three lemmas will be useful for bounding the number of d-relevant pairs of subforests from above. Their proofs are analogous to the corresponding proofs of Lemmas 3.2–3.4.

Lemma 3.6. If the pairs $(S(u,i,j), T(v))$ and $(S(u,i,j), T(w))$ are d-relevant for two ordered trees S and T, and w is an ancestor (or, descendant) of v in T, then $T(w)$ is a $2d$-ancestor (or, $2d$-descendant) of $T(v)$.

Lemma 3.7. For a node u of an ordered tree S, the number of d-ancestors of the form $S(w)$ of the forest $S(u)$ is at most d.

Lemma 3.8. Let $\{(S(u,i,j), T(v_q)\}_{q=0}^{l}$ be a sequence of distinct d-relevant pairs in two ordered trees S and T such that for any $0 \leq q', q'' \leq l$, $v_{q'}$ is not a descendant of $v_{q''}$. Then, $l \leq 2d$ holds.

By combining Lemmas 3.6–3.8, we obtain an upper bound on the number of d-relevant pairs $(S(u), T(v,k,l))$ and $(S(u,i,j), T(v))$ like in Theorem 3.1.

Theorem 3.2. For two ordered trees S and T and a node u of S, the number of distinct d-relevant pairs of the form $(S(u,i,j), T(v))$ is $O(d^2 \cdot (\deg(S))^2)$. Symmetrically, for a node v of T, the number of distinct d-relevant pairs of the form $(S(u), T(v,k,l))$ is $O(d^2 \cdot (\deg(T))^2)$.

Corollary 3.2. There are $O(n \cdot d^2 \cdot (maxdeg)^2)$ d-relevant pairs of subforests of the form $(S(u), T(v,k,l))$ and $(S(u,i,j), T(v))$ for S and T, where $n = \max\{|S|, |T|\}$.

4. Constructing the d-relevant pairs

The test for d-relevance for a pair of subtrees can easily be accomplished in constant time after appropriate preprocessing. However, in order to speed up the quadratic time algorithm of Jiang *at al.*, we cannot afford testing each possible pair of subtrees for d-relevance. Instead, we proceed as follows.

First, we compute all vectors $(|T[v]|, |L(T[v]|)$, where $v \in T$. Figure 4 demonstrates how this can be done recursively in $O(|T|)$ time by using the Euler tour technique [12]. The algorithm is started by calling $Euler\ Tour\ (root, 0)$. We assume that as the values of $|T[v]|$ and $|L(T[v])|$ for various nodes v are computed, they are stored in a tree T' which is isomorphic to T and equipped with the necessary auxiliary data fields.

We then fetch the vectors $(|T[v]|, |L(T[v]|)$ one at a time by traversing T', and insert them into a standard data structure for two-dimensional range search, e.g., a layered range tree [8]. The construction of the data structure takes $O(|T| \cdot \log |T|)$ time. Then, for all u in S we compute the vectors $(|S[u]|, |L(S[u])|)$ in linear time in the same way as above. For each u in S, we query the range search data structure with the square centered at $(|S[u]|, |L(S[u])|)$ having side length $2d$. Each query takes $O(\log |T| + r)$ time, where r is the number of reported vectors. Since each of the returned vectors is in one-to-one correspondence with a node v such that the pair (u, v) is d-relevant, $r = O(d^2)$ holds by Theorem 3.1.

Putting everything together, we obtain the following theorem.

Theorem 4.1. For two ordered trees on at most n nodes each and a non-negative integer d, all d-relevant pairs of subtrees can be reported in $O(n \cdot (\log n + d^2))$ time.

We can use the same technique to precompute all pairs of d-relevant subforests. In fact, for our purposes it is sufficient to report all pairs of d-relevant subforests where at least one of the subforests is complete, i.e., is of the form $S(u)$ or $T(v)$. To report all d-relevant pairs of the form $(S(u), T(v,k,l))$, the number of vectors to insert into the layered range tree is $O(|T| \cdot (maxdeg)^2)$ since $O((maxdeg)^2)$ ordered forests of the form $T(v,k,l)$ originate from each node v in T. Thus, the construction time becomes $O(|T| \cdot (maxdeg)^2 \cdot \log(|T| \cdot (maxdeg)^2)) = O(n \cdot (maxdeg)^2 \cdot \log n)$. The number of

```
Algorithm   Euler Tour
Input:   Node $v$, integer $left$.
Output: Integer $sumNodes$, integer $sumLeaves$.

  $|L(T[v])| := left$
  $sumNodes := 1$
  if $v$ is a leaf then
    $sumLeaves := 1$
  else
    $sumLeaves := 0$
    for all children $w$ of $v$ in left-to-right order do
        $sN, sL := Euler\ Tour\ (w,\ left + sumLeaves)$
        $sumLeaves := sumLeaves + sL$
        $sumNodes := sumNodes + sN$
    endfor
  endif
  $|T[v]| := sumNodes$
  return $sumNodes, sumLeaves$
End Euler Tour
```

Figure 4. The Euler tour algorithm for computing the vectors $(|T[v]|, |L(T[v]|)$, where $v \in T$.

queries to the data structure is $O(|S|)$, and the query time is $O(\log(|T| \cdot (maxdeg)^2) + r) = O(\log n + r)$ time, where the sum of the r's over S is $O(n \cdot d^2 \cdot (maxdeg)^2)$ by Corollary 3.2. The reporting of d-relevant pairs of the form $(S(u,i,j), T(v))$ can be done symmetrically within the same (in terms of n) preprocessing and query time bounds.

Summing up, we obtain:

Theorem 4.2. For two ordered trees on at most n nodes each and a non-negative integer d, all d-relevant pairs of subforests, where at least one subforest is complete, can be reported in $O(n \cdot (maxdeg)^2 \cdot (\log n + d^2))$ time.

5. Algorithm *Fast Score*

Our Algorithm *Fast Score* for computing the score of an optimal alignment between two labeled, ordered trees S and T is displayed in Figure 5. It works under the assumption that there exists an optimal alignment which uses at most d blank symbols, for some specified positive integer d.

First, we compute all d-relevant pairs of subtrees of S and T, and as each d-relevant pair is reported, we insert it into a balanced binary search tree $\mathcal{B}_1$. Next, all d-relevant pairs of subforests in which at least one subforest is complete are computed and inserted into two balanced binary search trees $\mathcal{B}_2$ and $\mathcal{B}_3$. According to Corollaries 3.1 and 3.2, there are $O(n \cdot d^2 \cdot (maxdeg)^2)$ d-relevant pairs of subtrees or subforests where at least one subforest is complete, so this preprocessing takes $O(n \cdot (maxdeg)^2 \cdot (\log n + d^2) + n \cdot d^2 \cdot (maxdeg)^2) \cdot \log(n \cdot d^2 \cdot (maxdeg)^2)) = O(n \log n \cdot (maxdeg)^2 \cdot d^2)$ time by Theorems 4.1

```
Algorithm   Fast Score
Input:   Two labeled ordered trees $S$ and $T$, positive integer $d$.
Output:  The score of an optimal alignment of $S$ and $T$ (assuming there exists an optimal alignment with at
         most $d$ blank symbols).

  Compute all $d$-relevant pairs of subtrees of $S$ and $T$ as described in Section 4.
  As each $d$-relevant pair is reported, insert it into a balanced binary search tree $\mathcal{B}_1$.

  Compute all $d$-relevant pairs of subforests of $S$ and $T$ of the form $(S(u,i,j),T(v))$ and $(S(u),T(v,k,l))$,
  and insert them into two balanced binary search trees $\mathcal{B}_2$ and $\mathcal{B}_3$.

  $D(\Theta,\Theta) := 0$
  for $u := 1$ to $|S|$ do
      Initialize $D(S[u],\Theta)$ and $D(S(u),\Theta)$ according to Lemma 2.1.1.
  endfor
  for $v := 1$ to $|T|$ do
      Initialize $D(\Theta,T[v])$ and $D(\Theta,T(v))$ according to Lemma 2.1.1.
  endfor
  for all $d$-relevant pairs of subtrees $(S[u],T[v])$, determined by doing an inorder traversal of $\mathcal{B}_1$ do
      for $i := 1$ to $\deg(u)$ do
          if $(S(u,i,\deg(u)),T(v))$ is $d$-relevant (i.e., belongs to $\mathcal{B}_2$) then
              Call Procedure $1'$ on $S(u,i,\deg(u))$ and $T(v)$.
          endif
      endfor
      for $k := 1$ to $\deg(v)$ do
          if $(S(u),T(v,k,\deg(v)))$ is $d$-relevant (i.e., belongs to $\mathcal{B}_3$) then
              Call Procedure $1'$ on $S(u)$ and $T(v,k,\deg(v))$.
          endif
      endfor

      Compute $D(S[u],T[v])$ as in Lemma 2.1.2, only considering $d$-relevant pairs of subtrees on the right
      hand side of the expression.

  endfor
  return $D(S[|S|],T[|T|])$
End Fast Score
```

Figure 5. The fast algorithm for computing the score of an optimal alignment between two ordered trees which uses at most d blank symbols.

and 4.2. The scores for pairs containing an empty subtree or subforest are also precomputed, which takes $O(|S|+|T|)=O(n)$ time.

We then modify the algorithm of Jiang *et al.* to only evaluate scores for d-relevant pairs of subforests and scores for pairs of subforests where one of the subforests is empty. Whenever one of the formulas in Lemma 2.1.2 or Lemma 2.1.3 is to be applied, we test each of the components of the right hand side which is not a pair containing an empty subtree or an empty subforest for membership in $\mathcal{B}_1$, $\mathcal{B}_2$, or $\mathcal{B}_3$. Such a membership query takes $O(\log n)$ time. If the test is positive, we fetch the score for the argument pair which should be evaluated by this time, otherwise we set that score to minus infinity. We conclude that the cost of determining the score for a d-relevant pair on the left hand side in Lemma 2.1 from the scores for d-relevant pairs occuring on the right hand side does not exceed the cost of determining the score for that pair based on the scores of pairs occuring on the right hand side multiplied by $O(\log n)$. Procedure 1′ referred to in Figure 5 is the same as Procedure 1 with such tests for d-relevance included. Therefore, each call to Procedure 1′ takes $O(\log n \cdot (\deg(u) + \deg(v)) \cdot \deg(u) \cdot \deg(v))$ time. Below, we denote the running time of one call to Procedure 1′ by P1′.

For each d-relevant pair of subtrees $(S[u], T[v])$, the algorithm tests $\deg(u)$ and then $\deg(v)$ pairs of subforests for d-relevance and makes at most this many calls to Procedure 1′. Next, it evaluates $D(S[u], T[v])$ by testing $\deg(u) + \deg(v)$ pairs of subtrees and one pair of subforests on the right hand side of the relation in Lemma 2.1.2 for d-relevance. Thus, each d-relevant pair of subtrees contributes $O(\deg(u) \cdot (\log n + \mathrm{P1}') + \deg(v) \cdot (\log n + \mathrm{P1}') + (\deg(u) + \deg(v)) \cdot \log n) = O(\log n \cdot (\deg(u) + \deg(v))^2 \cdot \deg(u) \cdot \deg(v))$ to the total running time. Summing over all d-relevant pairs of subtrees, we see that the entire main loop takes

$$\begin{aligned}
&\sum_{\substack{d\text{-relevant pairs of}\\ \text{subtrees } (S[u],T[v])}} O(\log n \cdot (\deg(u) + \deg(v))^2 \cdot \deg(u) \cdot \deg(v)) \\
&= O(\log n \cdot (maxdeg)^3 \cdot \sum_{\substack{d\text{-relevant pairs of}\\ \text{subtrees } (S[u],T[v])}} \deg(u)) \\
&= O(\log n \cdot (maxdeg)^3 \cdot \sum_{u \in S} \sum_{\substack{v \in T \text{ and}\\ (S[u],T[v])\\ \text{is } d\text{-relevant}}} \deg(u)) \\
&= O(\log n \cdot (maxdeg)^3 \cdot \sum_{u \in S} d^2 \cdot \deg(u)) \\
&= O(n \log n \cdot (maxdeg)^3 \cdot d^2)
\end{aligned}$$

time by using Theorem 3.1 and the fact that $\sum_{u \in S} \deg(u) = n$.

Including the preprocessing, the algorithm's running time is $O(n \log n \cdot (maxdeg)^2 \cdot d^2 + n + n \log n \cdot (maxdeg)^3 \cdot d^2) = O(n \log n \cdot (maxdeg)^3 \cdot d^2)$, which gives us the following theorem.

Theorem 5.1. If there exists an optimal alignment between S and T which uses at most d blank symbols and d is given, we can compute its score in $O(n \log n \cdot (maxdeg)^3 \cdot d^2)$ time.

We remark that Algorithm *Fast Score* can be modified to return an optimal alignment without increasing the asymptotic running time by adding a traceback step just like for the algorithm of Jiang *et al.* (see Section 2). Thus, we can construct an optimal alignment between S and T in $O(n \log n \cdot (maxdeg)^3 \cdot d^2)$ time if there exists an optimal alignment between S and T which uses at most d blank symbols and d is known in advance.

Also note that if $maxdeg = O(1)$, the running time of Algorithm *Fast Score* becomes $O(n \log n \cdot d^2)$.

6. Algorithm *Unspecified* d

Here, we extend Algorithm *Fast Score* from Section 5 to compute the score of an optimal alignment between the two input trees even if no upper bound on the number of blank symbols in an optimal alignment is given. We show that under some natural assumptions on the scoring scheme, the resulting method is faster than the algorithm of Jiang *et al.* for problem instances consisting of similar trees (i.e., instances in which the score of an optimal alignment is high). The technique we employ stems from Section 3.3.4 in [9], where it is applied to compute the score of an optimal alignment between two strings of equal length by using an algorithm which only evaluates a band of specified width around the main diagonal of the dynamic programming matrix.

Write $m = \min\{|S|, |T|\}$ and $n = \max\{|S|, |T|\}$. The algorithm of Jiang *et al.* runs in $O(m \cdot n \cdot (maxdeg)^2)$ time, regardless of the number of insertions required by an optimal solution (see Section 2). On the other hand, by Theorem 5.1, Algorithm *Fast Score* runs in $O(n \log n \cdot (maxdeg)^3 \cdot d^2)$ time, where d is the maximum number of insertions allowed. Thus, Algorithm *Fast Score* is asymptotically faster than the algorithm of Jiang *et al.* if d is small[5]. The drawback is that Algorithm *Fast Score* needs a value of d to be specified beforehand; the running time may be much worse than that of the algorithm of Jiang *et al.* if no sufficiently strong upper bound on d is known[6]. One way to overcome this difficulty is by running Algorithm *Fast Score* with successively larger values of d until a certain stop condition is satisfied, as explained below.

Let M be the maximum value of $\mu(s,t)$ over all pairs of symbols (s,t) belonging to $\Sigma \times \Sigma$, and let B be the maximum value of $\mu(s,t)$ over all pairs (s,t) in $(\Sigma \times \{-\}) \cup (\{-\} \times \Sigma)$, i.e., all pairs where precisely one of s and t is equal to the blank symbol. Assume that $M > 0$ and $B \le 0$.

Lemma 6.1. For any positive integer d, if an alignment between S and T uses at least $d+1$ blank symbols then its score is at most $(d+1) \cdot B + \frac{m+n-(d+1)}{2} \cdot M$.

Proof:
Let A be an alignment between S and T with at least $d+1$ blank symbols. Then the total number of nodes in S and T which can be paired off with each other is at most $|S| + |T| - (d+1)$. The maximum possible score of A is achieved when all such pairs of nodes have score M; thus, the score of A is at most $(d+1) \cdot B + \frac{|S|+|T|-(d+1)}{2} \cdot M$. □

For any positive integer d, let D_d be the value returned by Algorithm *Fast Score* on input (S,T,d). As d increases, D_d increases or remains the same while the value of $(d+1) \cdot B + \frac{m+n-(d+1)}{2} \cdot M$ decreases because $B \le 0$ and $M > 0$. Thus, by gradually increasing d, D_d eventually becomes larger than or equal to $(d+1) \cdot B + \frac{m+n-(d+1)}{2} \cdot M$. This yields a useful stop condition because when it occurs, Lemma 6.1 ensures that all alignments containing more blank symbols than the current value of d will have scores which are lower than or equal to D_d and therefore do not need to be considered.

[5]More precisely, if $d = o\left(\sqrt{\frac{m}{\log n \cdot maxdeg}}\right)$.

[6]For example, just plugging in the trivial upper bound $d = |S| + |T| \le 2n$ does not help here.

The algorithm is called Algorithm *Unspecified* d and is listed in Figure 6. Initially, it sets d to $(n - m) + 1$ since all alignments between S and T use at least $n - m$ blank symbols. It then finds the score of an optimal alignment by doubling d until the stop condition is satisfied.

```
Algorithm  Unspecified d
Input:  Two labeled ordered trees $S$ and $T$.
Output: The score of an optimal alignment between $S$ and $T$.

    $d := n - m + 1$
    $D_d :=$ Fast Score$(S, T, d)$
    while $D_d < (d+1) \cdot B + \frac{m+n-(d+1)}{2} \cdot M$ do
        $d := d \cdot 2$
        $D_d :=$ Fast Score$(S, T, d)$
    endwhile
    return $D_d$
End  Unspecified d
```

Figure 6. An algorithm for computing the score of an optimal alignment between two ordered trees when no upper bound on the number of blank symbols is provided.

We now analyze the running time of Algorithm *Unspecified* d. Denote the algorithm's final value of d by $\tilde{d}$. The first call to Algorithm *Fast Score* takes $O(n \log n \cdot (maxdeg)^3 \cdot (n - m + 1)^2)$ time, the second one $O(n \log n \cdot (maxdeg)^3 \cdot (2(n - m + 1))^2)$ time, etc., and the last one $O(n \log n \cdot (maxdeg)^3 \cdot \tilde{d}^2)$ time. Since

$$x^2 + (2x)^2 + (4x)^2 + (8x)^2 + \ldots + \tilde{d}^2 \;=\; x^2 \cdot \sum_{i=0}^{\log_2(\frac{\tilde{d}}{x})} (2^i)^2 \;=\; \frac{4\tilde{d}^2 - x^2}{3},$$

the running time is $O(n \log n \cdot (maxdeg)^3 \cdot (\tilde{d}^2 - (n - m + 1)^2))$.

We then proceed as in [9] to obtain a nontrivial upper bound on $\tilde{d}$ in terms of m, n, M, B, and s, where s is the score of an optimal alignment between S and T. When the algorithm stops, there are two possibilities:

- If $D_{\tilde{d}} = D_{\tilde{d}/2}$ then $s = D_{\tilde{d}/2}$. The inequality $D_{\tilde{d}/2} < (\frac{\tilde{d}}{2} + 1) \cdot B + \frac{m+n-(\frac{\tilde{d}}{2}+1)}{2} \cdot M$ (due to the algorithm not finishing in the previous iteration) then implies that $\tilde{d} < \frac{2(m+n)M-4s}{M-2B} - 2$.
- If $D_{\tilde{d}} > D_{\tilde{d}/2}$ then any optimal alignment contains $> \frac{\tilde{d}}{2}$ blank symbols so that by Lemma 6.1, $s \le (\frac{\tilde{d}}{2} + 1) \cdot B + \frac{m+n-(\frac{\tilde{d}}{2}+1)}{2} \cdot M$. Rearranging gives us $\tilde{d} \le \frac{2(m+n)M-4s}{M-2B} - 2$.

Thus, in both cases we have the upper bound

$$\tilde{d} \le 2\left(\frac{(m+n)M - 2s}{M - 2B} - 1\right) \tag{1}$$

The score of an optimal alignment between S and T is at most $m \cdot M$. Therefore, $s \le \frac{m+n}{2} \cdot M$. By inequality (1), if the score of an optimal alignment between S and T is high (so that s is close to $\frac{m+n}{2} \cdot M$) then $\tilde{d}$ is small. Assuming that $M - 2B$ is a constant, we can express the running time of Algorithm *Unspecified* d as follows.

Theorem 6.1. If $M - 2B$ is a constant and $B \le 0$, $M > 0$ then Algorithm *Unspecified* d computes the score of an optimal alignment between S and T in $O(n \log n \cdot (maxdeg)^3 \cdot f^2)$ time, where $f = \frac{m+n}{2} \cdot M - s$ and s is the score of an optimal alignment between S and T.

We also note the following:

Corollary 6.1. If there exist constants α, β, and γ such that $\alpha > 0$, $\beta \le 0$, $\gamma \le \alpha$ and for every $x, y \in \Sigma$ with $x \ne y$ it holds that $\mu(x,x) = \alpha$, $\mu(x,-) = \mu(-,x) = \beta$, and $\mu(x,y) = \gamma$, and if there exists an optimal alignment between S and T with b blank symbols and $O(b)$ node pairs of the form (x,y) with $x \ne y$, then Algorithm *Unspecified* d runs in $O(n \log n \cdot (maxdeg)^3 \cdot b^2)$ time.

Proof:
Write $s = b \cdot \beta + q \cdot \gamma + \frac{m+n-b-2q}{2} \cdot \alpha$, where q is the number of node pairs (x,y) with $x \ne y$ and $x, y \in \Sigma$. Combining this with inequality (1) yields $\tilde{d} \le 2\left(\frac{b(\alpha-2\beta)+2q(\alpha-\gamma)}{\alpha-2\beta} - 1\right)$. Now, $q = O(b)$ implies that $\tilde{d} = O(b)$. $\square$

In particular, if $maxdeg = O(1)$ then the running times given in Theorem 6.1 and Corollary 6.1 reduce to $O(n \log n \cdot f^2)$ and $O(n \log n \cdot b^2)$, respectively.

Finally, as mentioned at the end of Section 5, it is possible to modify Algorithm *Fast Score* (and hence also Algorithm *Unspecified* d) to return an optimal alignment by performing a traceback with no increase in the asymptotic running time.

7. Final remarks

An optimal alignment between two strings whose score is at most d apart from that of a perfect alignment between the first string and its copy can be constructed in $O(nd)$ time [9]. Since a string can be interpreted as a line ordered tree with node labels, a natural question arises: is it possible to lower the time complexity of our method, especially the exponent 2 of d?

Our method does not seem to generalize to include unordered trees directly. For example, the proof of Lemma 3.4 relies on the ordering of the trees (i.e., on the sets $L(\,)$). It is an interesting open problem whether a substantial speed-up in the construction of an optimal alignment between similar unordered trees of bounded degree is achievable.

In the construction of the d-relevant pairs, we could use more sophisticated and more asymptotically efficient data structures for two dimensional range search on an integer grid [1]. However, this would not lead to an improvement of the asymptotic total time complexity of our alignment algorithm.

References

[1] Alstrup, S., Brodal, G., Rauhe, T.: New Data Structures for Orthogonal Range Searching, *Proceedings of the 41st Annual Symposium on Foundations of Computer Science* (FOCS 2000), 2000.

[2] Gusfield, D.: *Algorithms on Strings, Trees, and Sequences : Computer Science and Computational Biology*, Cambridge University Press, 1997.

[3] Jansson, J., Lingas, A.: A Fast Algorithm for Optimal Alignment between Similar Ordered Trees, *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching* (CPM 2001), 2001.

[4] Jiang, T., Wang, L., Zhang, K.: Alignment of trees – an alternative to tree edit, *Theoretical Computer Science*, **143**, 1995, 137–148, A preliminary version appeared in *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching* (CPM'94), pages 75–86, 1994.

[5] Keselman, D., Amir, A.: Maximum agreement subtree in a set of evolutionary trees – metrics and efficient algorithms, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (FOCS '94), 1994.

[6] Le, S.-Y., Nussinov, R., Maizel, J.: Tree graphs of RNA secondary structures and their comparisons, *Computers and Biomedical Research*, **22**, 1989, 461–473.

[7] Pevzner, P.: *Computational Molecular Biology : An Algorithmic Approach*, The MIT Press, Massachusetts, 2000.

[8] Preparata, F., Shamos, M.: *Computational Geometry*, Springer-Verlag, New York, 1985.

[9] Setubal, J., Meidanis, J.: *Introduction to Computational Molecular Biology*, PWS Publishing Company, Boston, 1997.

[10] Shapiro, B.: An algorithm for comparing multiple RNA secondary structures, *Computer Applications in the Biosciences*, **4**, 1988, 387–393.

[11] Tai, K.-C.: The tree-to-tree correction problem, *Journal of the ACM*, **26**(3), 1979, 422–433.

[12] Tarjan, R., Vishkin, U.: An efficient parallel biconnectivity algorithm, *SIAM Journal on Computing*, **14**(4), 1985, 862–874.

[13] Waterman, M.: *Introduction to Computational Biology : Maps, Sequences, and Genomes*, Chapman & Hall, London, 1995.

[14] Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems, *SIAM Journal on Computing*, **18**(6), 1989, 1245–1262.