Efficient Chaining of Seeds in Ordered Trees

Julien Allali[1,2,3], Cédric Chauve[3],
Pascal Ferraro[1,2,4], and Anne-Laure Gaillard[1]

[1] LaBRI, Université Bordeaux 1, IPB, CNRS
{julien.allali,anne-laure.gaillard,pascal.ferraro}@labri.fr
[2] Pacific Institute for Mathematical Sciences and CNRS UMI3069
[3] Department of Mathematics, Simon Fraser University
cedric.chauve@sfu.ca
[4] Department of Computer Science, University of Calgary

Abstract. We consider here the problem of chaining seeds in ordered trees. Seeds are mappings between two trees Q and T and a chain is a subset of non overlapping seeds that is consistent with respect to postfix order and ancestrality. This problem is a natural extension of a similar problem for sequences, and has applications in computational biology, such as mining a database of RNA secondary structures. For the chaining problem with a set of m constant size seeds, we describe an algorithm with complexity $O(m^2 \log(m))$ in time and $O(m^2)$ in space.

1 Introduction

Comparing sequences is a basic task in computational biology, either for mining genomics database, or for filtering large sequence datasets. A fundamental application of sequence comparison is to search efficiently in a database a set of sequences close to a query sequence. The exponential increase of available sequence data motivates the need for very efficient sequence comparison algorithms. In particular, pairwise comparison relying on computing the exact edit distance between the query and every every sequence of the database can not practically be applied due to the quadratic time complexity of edit distance computation. A typical approach to tackle this issue is to rely on short sequences, called *seeds*, present in the query. Seeds can be detected very quickly in the database using indexing techniques; then an optimal set of seeds, called a *chain*, that tiles both the query and a sequence of the database, must be identified while conserving the same order in both sequences. Widely used programs such as BLAST [2] and FASTA [11,14] rely on such an approach. We refer the reader to [3,7] for surveys of sequence comparison in computational biology. From an algorithmic point of view, an optimal chain between two sequences, given m seeds, can be computed in $O(m \log(m))$ time and $O(m)$ space [10] (see [13] for a recent survey).

With the recent development of high-throughput genome annotation methods, similar problems appear to be relevant for the analysis of more complex biological structures [15]. For instance, an RNA secondary structures can be

C.S. Iliopoulos and W.F. Smyth (Eds.): IWOCA 2010, LNCS 6460, pp. 260–273, 2011.

represented by a tree or a graph whose nodes are the nucleotides and whose edges are the chemical bonds between them [16]. Mining large RNA secondary structure databases, such as Rfam [6], is now an important computational biology problem. An initial approach, adapting the notion of edit distance to ordered trees, was pioneered by Zhang and Shasha [17]. The tree edit approach has been extended in several ways since then, leading either to hard problems, when a comprehensive set of edit operations is considered [9], or to algorithms with a worst-case time complexity at best cubic, even with a minimal set of edit operations [5,17].

Recently, Heyne *et al.* [8] introduced a chaining problem on an alternative representation of ordered trees called arc annotated sequences, motivated by pairwise RNA secondary structure comparison: once an optimal chain of seeds between two given RNA secondary structures is detected, the regions between successive seeds are processed independently using an edit distance algorithm, which speeds up significantly the comparison process. They considered seeds defined as *exact common patterns* and designed a dynamic programming algorithm to solve the seeds chaining problem. To the best of our knowledge, [8] is the first paper addressing a chaining problem in trees.

After some preliminaries (Sections 2 and 3), we describe in Section 4 an algorithm for finding the score of an optimal chain between two ordered trees (Maximal Chaining Problem) in $O(m^2 \log(m))$ time and $O(m^2)$ space when there are m seeds of constant size, thus improving on the result of Heyne *et al.* [8]. We conclude with further research avenues.

2 Background and Problem Statement

Let T be an ordered rooted tree of size n. Nodes of T are identified with their postfix-order index from 0 to $n-1$. Thus, $n-1$ represents the root of T. T_i is the subtree of T rooted at i. We denote by $T[i,j]$ the forest induced by the nodes that belong to the interval $[i,j]$; if $i > j$, then $T[i,j]$ is empty. The partial relationship "i is an ancestor of j" is denoted by $i \prec j$. For a tree T and a node i of T, the first leaf visited during a postfix traversal of T_i is denoted by $l(i)$ and called the *leftmost leaf* of the node i. The ordered forest induced by the proper descendants of i is denoted by $\widehat{T}_i = T[l(i), i-1]$.

Definition 1. Let T be an ordered rooted tree:

1. Let $G = \{g_0, \ldots, g_{k-1}\}$ be an ordered set of k nodes of T, with $0 \leq g_j < n$. If the subgraph of T induced by G is connected, then G is called an *internal tree* rooted at g_{k-1} also referred to as r_G.
2. The set of leaves of the internal tree G is denoted by $L(G)$.
3. A node g_j of G is said to be *completely inside* G if g_j is not a leaf of T and all its children belong to G. The set of nodes of G that are not completely inside G is called the *border of* G and is denoted by $B(G)$.
4. Two internal trees G^1 and G^2 *overlap* if they share at least one node, *i.e.* $G^1 \cap G^2 \neq \emptyset$.

We now recall the central notion of *valid mapping* between two trees introduced in [16] for the tree edit distance. Given two trees Q and T, a valid mapping P between Q and T is a set of pairs of $Q \times T$ such that, if (q_i, t_i) and (q_j, t_j) belong to P, then

1. $q_i = q_j$ if and only if $t_i = t_j$,
2. $q_i < q_j$ if and only if $t_i < t_j$,
3. $q_i \prec q_j$ if and only if $t_i \prec t_j$.

From now we use the term *mapping* to refer to a valid mapping. Given a mapping P between Q and T, the smallest internal tree of Q (resp. T) that contains all nodes of Q (resp. T) belonging to a pair of P is denoted by Q_P (resp. T_P). Q_P and T_P are respectively called the internal trees of Q and T induced by P.

Definition 2. Let Q and T be two ordered trees.

1. A *seed* P between Q and T is a mapping between Q and T such that $(r_{Q_P}, r_{T_P}) \in P$ and all the nodes of the border of Q_P (resp. T_P) belong to a pair of P.
2. The border (resp. leaves) $B(P)$ (resp. $L(P)$) of the seed P is the set of pairs $(x, y) \in P$ such that $x \in B(Q_P)$ and $y \in B(T_P)$ (resp. $x \in L(Q_P)$ and $y \in L(T_P)$).
3. The *size* $|P|$ of the seed P is the number of pairs its mapping contains.
4. For a set S of seeds, $\|S\|$ is the sum of the sizes of the $|S|$ seeds in S.

Note that, theoretically, the number of seeds between Q and T can be exponential in the size of Q and T, although in applications such as RNA secondary structure comparison, this exponential upper bound is unlikely to be reached (see [8] for example).

Definition 3. Let Q and T be two ordered trees.

1. A pair (P^1, P^2) of seeds between Q and T is *chainable* if Q_{P^1} does not overlap Q_{P^2}, T_{P^1} does not overlap T_{P^2}, and $P^1 \cup P^2$ is a mapping.
2. A *chain* is a set $C = \{P^0, P^1, \ldots, P^{\ell-1}\}$ of seeds between Q and T such that any pair (P^i, P^j) of distinct seeds in C is chainable.
3. Given a scoring function v for the seeds P^i, the score of a chain C is the sum of the scores of its seeds: $v(C) = \sum_i v(P^i)$.
4. Given a set S of possibly overlapping seeds between Q and T, $\mathcal{C}_S(Q, T)$ denotes the set of all possible chains between Q and T included in S.

We can now define the main problem we consider in the present paper (illustrated in Fig. 1).

Problem. Maximum Chaining Problem (MCP):
Input: A pair (Q, T) of ordered rooted trees, a set $S = \{P^0, \ldots, P^{m-1}\}$ of m possibly overlapping seeds between Q and T, a scoring function v on the seeds P^i.
Output: The maximum score chain C included in S:

$$MCP(Q, T, S) = \max\{v(C); C \in \mathcal{C}_S(Q, T)\}.$$

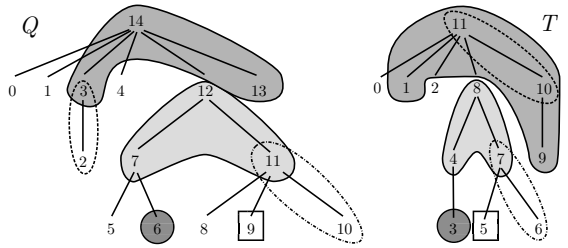

Fig. 1. An instance of the MCP with 6 seeds: $P^0 = \{(2,10),(3,11)\}$, $P^1 = \{(6,3)\}$, $P^2 = \{(9,5)\}$, $P^3 = \{(10,6),(11,7)\}$, $P^4 = \{(7,4),(11,7),(12,8)\}$, $P^5 = \{(3,1),(13,9),(14,11)\}$. If for every seed $v(P^i) = |P^i|$, an optimal chain is composed of $\{P^1, P^2, P^4, P^5\}$ and has a score of 8.

Remark 1. The notion of mapping extends naturally to ordered forests. Hence, if S is a set of seeds such that each seed is a seed between a tree of F_1 and a tree of F_2, then the MCP can naturally be extended to ordered forests.

Remark 2. To compare with chaining algorithms for sequences, we represent a sequence $u = (u_0, \ldots, u_{n-1})$ by a unary tree, rooted at a node labeled by u_{n-1}, where every internal node has a single child and u_0 is the unique leaf: the sequence of nodes visited by the postfix-order traversal of this tree is exactly u.

Motivation and background. As far as we know, [8] is the only work that attacks the MCP in tree structures, although the authors describe the problem in terms of arc-annotated sequences. They proposed a dynamic programming algorithm to solve the maximum chaining problem with some restrictions on the seeds (precisely, seeds are maximal exact pattern common to the considered sequences). This dynamic programming technique is different from the approach used for the currently best known algorithms for Maximum Chaining Problem in sequences [10,13]. Moreover, when applied to arc-annotated sequences with no arc (*i.e.* sequences) and m seeds, it can be shown this algorithm has a worst-case time complexity in $O(m^2)$.

Result statement. Our main result is the following:

Theorem 1. *Let S be a set of m seeds between two ordered trees Q and T. After an $O(\|S\|)$ time preprocessing of the m seeds of S one can solve the Maximum Chaining Problem in $O(\|S\| \log(\|S\|) + m\|S\| \log(m))$ time and $O(m\|S\|)$ space.*

Note that we described the complexity of our algorithm using uniquely the set of seeds S, unlike Heyne *et al.* [8], who, for the same problem, also consider the sizes of Q and T (see [1] for a detailed analysis of the complexity of the algorithm of [8]). We prove in Section 4, that our algorithm solves the maximal chaining problem on sequences (*i.e.* unary trees as described in Remark 2 above) in $O(m \log(m))$ time and $O(m)$ space complexity.

Remark 3. Without loss of generality, from now we assume that the seeds P^i are sorted increasingly according to the postfix number of their roots in Q, that is: $r_{Q_{P^0}} \leq \cdots \leq r_{Q_{Pi}} \leq \cdots \leq r_{Q_{Pm-1}}$. For a given chain C, the *last* seed of C is then the seed with the highest postfix index in Q.

3 Combinatorial Properties of Seeds and Chains

We first describe combinatorial properties of seeds and chains, that naturally lead to a recursive scheme to compute a maximum chain. Indeed, we show that given a chain C and its last seed P, the root and border of P define a partition of both $Q - Q_P$ and $T - T_P$ into pairs of forests that contain the seeds $C - \{P\}$ and form sub-chains of C. More precisely, for every border nodes (x, y) of a seed P, we define the couples of forests included in $(\widehat{Q_x}, \widehat{T_y})$, that is composed of descendants of (x, y), such that any seed included into such couple of forest is chainable with P.

Definition 4. Let P be a seed on two trees Q and T and $(a, b; c, d)$ be a quadruple such that $l(r_{Q_P}) \leq a < b < r_{Q_P}$, $l(r_{T_P}) \leq c < d < r_{T_P}$ and the pair of forests $(Q[a, b], T[c, d])$ does not contain any node involved in P ($Q_P \cap Q[a, b] = \emptyset$ and $T_P \cap T[c, d] = \emptyset$). $(a, b; c, d)$ is a *chainable area* if for all $i \in [a, b]$ and all $j \in [c, d]$, $P \cup (i, j)$ is a valid mapping. $(a, b; c, d)$ is a *maximal chainable area* for P if neither $(a-1, b; c, d)$ or $(a, b+1; c, d)$ or $(a, b; c-1, d)$ or $(a, b; c, d+1)$ are chainable areas for P.

For example, in Fig. 1, let us consider the seed $P = P^5$; then, $(4, 12; 2, 8)$ is a maximal chainable area. See also Figure 2.

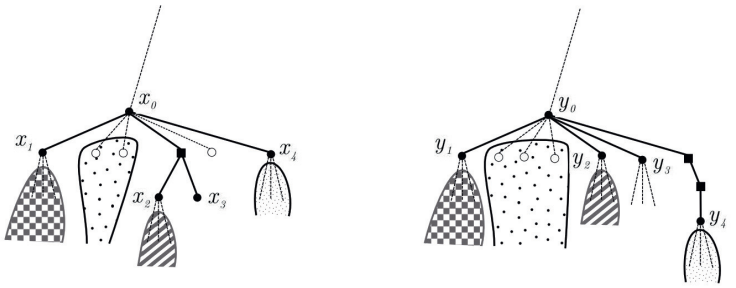

Fig. 2. Illustration of the notion of chainable areas of a seed of size 5: $P = \{(x_0, y_0), \ldots, (x_4, y_4)\}$ and there are 4 chainable areas for P each indicated by a different filling pattern

Definition 5. Let $(x, y) \in B(P)$ for a seed P between Q and T. We define by $F(x, y) = \{(a_i, b_i; c_i, d_i)\}$ the set of all maximal chainable areas for P included in $(Q_x; T_y)$ such that there is no border node of P in Q (resp. T) on the path from b to x (resp. d to y). We call this set the *chainable areas* of (x, y).

For example, let us consider a pair (x, y) in $L(P)$ such that x and y are not a leaf of respectively Q and T, then $F(x, y)$ represents the couple of forests $\widehat{Q_x}$ and $\widehat{T_y}$, $F(x, y) = \{(l(x), x-1; l(y), y-1)\}$. In Fig. 1, with $P = P^4$ and $(x, y) = (11, 7)$, $F(x, y) = \{(8, 10; 5, 6)\}$); if $(x, y) = (14, 11) \in B(P^5) - L(P^5)$, $F(x, y) = \{(0, 1; 0, 0), (4, 12; 2, 8)\}$. See also Figure 3.

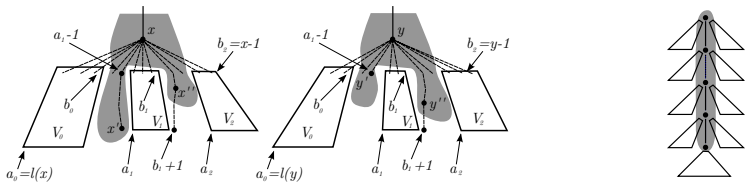

Fig. 3. (Left) Illustration of Definition 5 for a seed P (the shaded zone) and $(x, y) \in B(P) - L(P)$: $F(x, y) = \{(a_0, b_0, c_0, c_1), (a_1, b_1, c_1, c_1), (a_2, b_2, c_2, c_2)\}$. (Right) Illustration of the maximum number of chainable areas of a seed.

Definition 6. The *chainable areas* of a seed P, denoted by $CA(P)$, is the union of the sets of quadruples $F(x, y)$ for all pairs $(x, y) \in B(P)$.

Notation. For a seed P (resp. chain C) and a chainable area $(a, b; c, d)$, we say that $P \subset (a, b; c, d)$ (resp. $C \subset (a, b; c, d)$) if $a \leq r_{Q_P} \leq b$ and $c \leq r_{T_P} \leq d$.

The following property is a relatively straightforward consequence of the definitions of seeds and chainable areas (Fig. 3).

Property 1. Given a seed P between trees Q and T, $|CA(P)| \leq 2 \times |B(P)| - 1$.

From now, for every (x, y) of a seed P^j, we denote by x^j the unique node y of T associated with x in P^j. We also denote by $F_j(x)$ the set of quadruples $F(x, x^j)$ for the pair of nodes $(x, x^j) \in P^j$.

The next property describes the structure of any chain between two forests $Q[a, b]$ and $T[c, d]$ included in a set of m seeds $S = \{P^0, \ldots, P^{m-1}\}$. It is a direct consequence of the constraints that define a valid mapping and the fact that seeds are non-overlapping in a chain.

Property 2. Let P^j be the last seed of a chain C included into two forests $Q[a, b]$ and $T[c, d]$.

1. C can be decomposed into $|CA(P^j)|+2$ (possibly empty) distinct sub-chains: P^j itself, $|CA(P^j)|$ chains: for each $(e, f; g, h) \in CA(P^j)$ a (possible empty) chain included into $Q[e, f]$ and $T[g, h]$ and a chain included into the forests $Q[a, l(r_j) - 1]$ and $T[c, l(r_j^j) - 1]$.
2. Moreover, C is a chain of maximum score among all chains in $Q[a, b]$ and $T[c, d]$ that contain P^j if and only if all of its sub-chains described above are chains of maximum score with respect to the corresponding forests defined by $CA(P^j)$.

Property 2.2 naturally leads to a recursive scheme to compute an optimal chain between two forests $Q[a,b]$ and $T[c,d]$ that ends by the last seed of a set. If $MCP'(Q[a,b],T[c,d],\{P^0 \dots P^j\})$ is the score of a maximum chain between $Q[a,b]$ and $T[c,d]$ and that contains P^j:

$$MCP'(Q[a,b],T[c,d],\{P^0 \dots P^j\}) = \tag{1}$$

$$\begin{cases} 0 & \text{if } P^j \not\subset (a,b;c,d), \\ v(P^j) + \sum\limits_{(e,f;g,h)\in CA(P^j)} MCP(Q[e,f],T[g,h],\{P^0 \dots P^{j-1}\}) & \text{otherwise.} \\ \quad +MCP(Q[a,l(r_j)-1],T[c,l(r_j^j)-1],\{P^0 \dots P^{j-1}\}) & \end{cases}$$

and thus $MCP(Q,T,S)$ can be computed using MCP' as follow[1]:

$$MCP(Q[a,b],T[c,d],\{P^0 \dots P^j\}) = \max_{i=0\dots j} MCP'(Q[a,b],T[c,d],\{P^0 \dots P^i\}) \tag{2}$$

$$MCP(Q,T,S) = MCP(Q[0,r_Q],T[0,r_T],S) \tag{3}$$

The main challenge in designing an algorithm for the MCP is then to implement efficiently this recursive formula, that was already central in the dynamic programming algorithm of [8]. In Section 4, we will rely on the fact that for every seed P^j, $CA(P^j)$ and, for every border node x of P^j, $F_j(x)$, have been computed during a preprocessing phase. A journal version will discuss the issues related to this preprocessing and will show that it can be done in $O(\|S\|)$ time and space (see also [1]).

4 Algorithms for the Maximum Chaining Problem

From now, we consider that we are given two ordered trees Q and T, a set $S = \{P^0,\dots,P^{m-1}\}$ of seeds and a scoring score v on S. Furthermore, we assume that the score $v(j)$ of a seed P^j can be accessed in constant time and the seeds of S are given as a list I of triples (i,f,j) such that: (1) i is the postfix number of either the root of P^j_Q or a border node of P^j_Q (*ie.* $i \in B(P^j_Q) \cup \{r_j\}$) and (2) f is a flag indicating if i is either border ($f=0$) or root ($f=1$) for P^j_Q. Thus if i is both in $B(P^j_Q)$ and the root of P^j_Q then i appears in two distinct triples[2]. Moreover, for a node i in Q belonging to a seed P^j, we assume that the corresponding node in T, i^j (or more precisely its postfix number in T) can be accessed in constant time. Finally, for every node i in Q and T, its leftmost leaf $l(i)$ is also supposed to be accessed in constant time.

As a preprocessing, I is sorted in lexicographic order. Thus, if a node is both in the border and root of P^j, it first appears in I as a border, then as a root. This sorting can be done in $O(||S||\log(||S||))$ time. In our algorithms, we visit

[1] We remind that the seeds are supposed to be sorted incrementally (see Remark 3).

[2] Hence, we do not require as input the whole seeds mappings but just the borders and roots of the seeds, as it is usual when chaining seeds in sequences.

successively the elements of I in increasing order, and a seed P^j is said to be *processed* after its root has been processed (*i.e.* the current element of I is greater than $(r_j, 1, j)$ for the order defined above).

In the following, we first introduce a simple but non optimal algorithm to compute the MCP between Q and T which does not require any special data structure. In a second step, we will present a more efficient method based on a simple modification of this algorithm.

4.1 A Simple Non Optimal Algorithm

In order to compute in constant time the partial MCP for any pair of forests in $CA(P^j)$ as described in equation (1), we introduce a data structure M indexed by quadruples of integers $(a, b; c, d)$ defining the forests $Q[a, b]$ and $T[c, d]$. These quadruples $(a, b; c, d)$ belong to a set $Y = Y_1 \cup Y_2 \cup Y_3$ defined as follows:

$$Y_1 = \bigcup_{j=0}^{m-1} CA(P^j),\ Y_2 = \{(0, r_Q, 0, r_T)\},$$

$$Y_3 = \{(a, l(r_j) - 1; c, l(r_j^j) - 1) \mid \exists (b, d); (a, b; c, d) \in Y_1 \cup Y_2 \text{ and } P^j \subset (a, b; c, d)\}$$

In algorithm 1, the function `Update` replaces the value of $M[a, b, c, d]$ by a real number w if w is greater than $M[a, b, c, d]$. We also use an array V of m integers to store the intermediate quantities of $MCP^{'}$. The correctness of the algorithm relies on the following invariants for the two data structures V and M, that we prove later:

M1. After P^j has been processed, then $M[a, b, c, d] = MCP(Q[a, b], T[c, d], \{P^0, \ldots, P^j\})$ for every $(a, b; c, d) \in Y$.
V1. After P^j has been processed, then $V[j] = MCP'(Q, T, \{P^0, \ldots, P^j\})$.

Correctness of the algorithm. Obviously, V1 implies that $\max_j V[j]$ contains the score of the maximum chain (equations (2) and (3)). Let us assume now that M1 is satisfied. If the seed P^j has been processed, then $V[j]$ contains the sum of $v(j)$ (line 1), the MCP scores of the chainable areas of all its border nodes (line 5) and the MCP score between forests $Q(0, l(r_j) - 1)$ and $T(0, l(r_j^j) - 1)$ (line 11). From Property 2 and (1), $V[j] = MCP'(Q, T, \{P^0, \ldots, P^j\})$ and V1 is satisfied.

We prove M1 by induction. Initially, since no seed has been processed, line 2 ensures that M1 is satisfied. Now let us assume that M1 is satisfied for all processed seeds $\{P^0, \ldots, P^{j-1}\}$ and the input $(i, 1, j)$ is being processed. If $P^j \not\subset (a, b; c, d)$, then by induction, M1 is satisfied for $M[a, b, c, d]$. Otherwise, the loop in lines 7 and 8 ensures that M1 is satisfied for all entries $M[a, b, c, d]$ such that $(a, b; c, d) \in Y_1 \cup Y_2$, as $(a, l(r_j) - 1; c, l(r_j^j) - 1)$ does not contain P^j ; thus by induction M1 is satisfied for this index. Finally, the loop in line 9 update all $(a, b; c, d) \in Y_3$ including P^j,and M1 is satisfied for all entries of M.

Algorithm 1. MCP_1: compute the score of a maximum chain.

```
1  for j from 0 to m − 1 do V[j] = v(j)
2  foreach (a, b; c, d) ∈ Y do M[a, b, c, d] = 0
3  foreach (i, f, j) in I do
4    if f = 0 then # i.e. (i, i^j) ∈ B(P^j)
5      foreach (a, b; c, d) ∈ F_j(i) do V[j] = V[j] + M[a, b, c, d]
6    else # i.e. f = 1 and i is the root of Q_{P^j}, i = r_j
7      foreach (a, b; c, d) ∈ Y_1 ∪ Y_2 s.t. P^j ⊂ (a, b; c, d) do
8        Update M[a, b, c, d] with w = V[j] + M[a, l(r_j) − 1, c, l(r_j^j) − 1]
9        foreach P^g ⊂ (r_j + 1, b; r_j^j + 1, d) do
10         Update M[a, l(r_g) − 1, c, l(r_g^g) − 1] with w
11     V[j] = V[j] + M[0, l(r_j) − 1, 0, l(r_j^j) − 1]
12 return max_j V[j]
```

Complexity analysis. From Property 1, the space required to encode the entries of M indexed by Y_1 is in $O(\|S\|)$. The space required to encode the entries of M indexed by Y_3 is in $O(m^2)$, as for every pair of seeds P^i and P^j, there is at most one chainable area of $CA(P^i)$ that contains P^j.

We now address the worst-case time complexity. We do not factor the preprocessing required to compute the F_j and CA and we assume I has been sorted in time $O(\|S\| \log(\|S\|))$. The amortized cost of lines 4–5 is $O(\|S\|)$, as each chainable area is considered once, there are $O(\|S\|)$ such areas, and we assumed we can access them in amortized constant time. A naive implementation of lines 6–11 would require $O(m^2\|S\|)$ operations: indeed, there are m iterations of the loop in line 6, the loop in line 7 considers only entries indexed by $Y_1 \cup Y_2$ (there are $O(\|S\|)$ such entries) and the loop on line 9 iterates $O(m)$ times. However, we can notice that there are $O(m)$ entries $(a, b; c, d) \in Y_1 \cup Y_2$ such that $P^j \subset (a, b; c, d)$, and it is possible to preprocess I in time and space $O(m\|S\|)$ in such a way that the loop in line 7 can be implemented to perform $O(m)$ iterations, leading to a total time complexity of $O(\|S\| \log(\|S\|) + m\|S\| + m^3)$ (respectively for sorting the input, preprocessing and then the main algorithm).

4.2 A More Efficient Algorithm

We describe and analyze now a more efficient algorithm, which proves our main result, Theorem 1.

The key ideas are to access less entries from M (while maintaining property M1 on the remaining entries though) and to complement M with a data structure R that can be queried in $O(\log(m))$ instead of $O(1)$, but whose maintenance does not require a loop with $O(m^2)$ iterations. Formally, let $X = \{(a, c) \text{ s.t. } \exists (a, b; c, d) \in Y_1 \cup Y_2\}$ and R be a data structure indexed by X such that, for a given index $(a, c) \in X$, $R[a, c]$ is a set of pairs (j, s) where j is the index of the seed P^j and s is the maximum score of chains in $Q[a, r_j], T[c, r_j^j]$

that ends with P^j. Roughly, M is used to access, still in $O(1)$ time, the values $MCP(a, l(r_j)-1, c, l(r_j^j)-1, \{P^0 \dots P^{j-1}\})$ required to compute MCP' in equation (1) and $R[a,c]$ is used to access, in time $O(\log(m))$, the scores of the best chains included in $(Q[a, r_Q], T[c, r_T])$ (the values $MCP(Q[e,f], T[g,h], \{P^0 \dots P^{j-1}\})$ in equation (1)) and replace the entries $M[a,b,c,d]$ with $(a,b;c,d) \in Y_1 \cup Y_2$, which were used in the previous algorithm.

Finally, the algorithm iterates on a list of triples $J = I \bigcup \left(\cup_{j=0}^{m-1}(l(r_j), -1, j)\right)$, sorted using the lexicographic order used in the previous section, with the following modification: if we have two seeds P^j and P^g with $g > j$ such that $(l(r_j), l(r_j^j)) = (l(r_g), l(r_g^g))$ then only $(l(r_j), -1, j)$ occurs in J. This preprocessing requires $O(||S|| \log(||S||))$ time.

Algorithm 2. $MCP_2(Q, T, S, v)$: compute a maximum chaining from S.

```
1  for j from 0 to m − 1 do V[j] = v(j)
2  foreach (a, b; c, d) ∈ Y3 do M[a, b, c, d] = 0
3  foreach (a, c) ∈ X do R[a, c] = ∅
4  foreach (i, f, j) in J do
5    if f = −1 then # i = l(r_j)
6      foreach (a, c) ∈ X s.t. a, c < l(r_j), l(r_j^j) do
7        M[a, l(r_j) − 1, c, l(r_j^j) − 1]= value s of the last (y, s) of R[a, c] s.t. r_y^y < l(r_j^j)
8    else if f = 0 then # (i, i^j) ∈ B(P^j)
9      foreach (a, b; c, d) ∈ F_j(i) do
10       Add to V[j] the value s of the last entry (y, s) of R[a, c] s.t. r_y^y ≤ d
11   else # f = 1 and i is the root of Q_{P^j}, i = r_j
12     foreach (a, c) ∈ X s.t. a, c < l(r_j), l(r_j^j) do
13       w = V[j] + M[a, l(r_j) − 1, c, l(r_j^j) − 1]
14       Insert entry (j, w) into R[a, c] and update R[a, c] as follow:
15             Find the last entry (y, s) s.t. r_y^y < r_j^j
16             if s < w then
17                   Insert (j, w) just after (y, s) in R[a, c]
18                   Remove from R[a, c] all entries (z, t) s.t. r_j^j ≤ r_z^z and t < w
19     V[j] = V[j] + M[0, l(r_j) − 1, 0, l(r_j^j) − 1]
20 return max_j V[j]
```

Correctness of the algorithm. We consider the following invariants.

M2. After P^j has been processed, then $M[a,b,c,d] = MCP(Q[a,b], T[c,d], \{P^0, \dots, P^j\})$ for every $(a,b;c,d) \in Y_3$.

V1. After P^j has been processed, then $V[j] = MCP'(Q, T, \{P^0, \dots, P^j\})$.

R1. After P^j has been processed, then for all $(a,c) \in X$, $R[a,c]$ contains all (y,s) that satisfies
   a. $y \le j$ and $s = MCP'(Q[a, r_y], T[c, r_y^y], \{P^0, \dots, P^y\})$.
   b. $\forall (z,t) \in R[a,c]$, $r_z^z < r_y^y \Rightarrow t < s$.

R2. $\forall (a,c) \in X$, $R[a,c]$ is totally ordered as follows: $(y,s) < (z,t)$ iff $r_y^y < r_z^z$.

We first assume that R1 and R2 are satisfied. As previously, if V1 is satisfied, then the algorithm computes $MCP(Q, T, S)$. The initialization line 1 ensures that $V[j]$ contains $v(j)$. Next to prove V1 we only need to show that, when we process a border i of a seed P^j, in line 10 we add to $V[j]$ the best chain of each chainable area $(a, b; c, d)$ of the border; it follows from (1) the fact that every seed P^{j+e} with $e > 0$ does not belong to the forest $Q[a, b]$ (because $b < i \leq r_{j+e}$) and thus can not belong to a chain in the $(a, b; c, d)$ area, (2) the fact that the score of this chain is present in $R[a, c]$ (from R1) and (3) the fact that it is the last entry (y, s) such that $r^y_y \leq d$ (from R2).

M2 is similar to M1 but restricted to entries $M[a, b, c, d]$ such that $(a, b; c, d) \in Y_3$. To check it is satisfied, we only need to focus on line 7, as it is the only line that updates M. For entries $M[a, b, c, d]$ such that $a \geq l(r_j)$ or $c \geq l(r^j_j)$, then $M[a, b, c, d] = 0$ due to the initialization in line 1. For all other entries, M2 follows immediately from R1 and R2, using argument similar to the previous ones.

Finally, we need to check that R1 and R2 are satisfied. First, as previously, in the case where $a \geq l(r_j)$ or $c \geq l(r^j_j)$, $R[a, c] = \emptyset$ which is ensured by the initialization in line 3. So we need only to consider the case where $a, c < l(r_j), l(r^j_j)$, that is handled in lines 11 to 18. Every seed P^y such that $y < j$ has already been processed and $s = MCP'(Q[a, r_y], T[c, r^y_y], \{P^0, \ldots, P^y\})$ can not be modified after P^y has been processed, so lines 12 and 13, together with M2, ensure that (y, s) has been inserted into $R[a, c]$ previously, and the same argument applies if $y = j$. Entries (z, t) removed at line 18 do not belong to any of these (y, s), which implies that R1.a and R1.b, and so R1, are satisfied. R2 is obviously satisfied from the position where (j, w) is inserted into $R[a, c]$ in line 17.

Complexity analysis. The space complexity is given by the space required for structures M and R. M requires a space in $O(m^2)$ as it is indexed by Y_3. R requires a space in $O(m\|S\|)$, as $|Y_1 \cup Y_2| \in O(\|S\|)$ and for each seed P^j, an entry (j, s) is inserted at most once in each $R[a, c]$. All together, the space complexity is then $O(m^2 + m\|S\|) = O(m\|S\|)$.

We now describe the time complexity. First, note that following the technique used for computing maximum chains in sequence [7,10,13], the structures $R[l_Q, l_T]$ can be implemented using classical data structures such as AVL or concatenable queues supporting query requests, insertions and deletions, successor and predecessor, in a set of n totally ordered elements in $O(\log(n))$ worst-case time.

Now, we analyze the complexity of lines 5 to 7. The loop of line 6 is performed at most $O(m\|S\|)$ times and each iteration requires $O(\log(m))$ in time (line 7), which gives an amortized time complexity of $O(m\|S\| \log(m))$.

Line 10 is applied at most once for each of the $O(\|S\|)$ chainable area $F_j(i)$ (Property 1), and each iteration requires $O(\log(m))$, which gives an $O(\|S\| \log(m))$ amortized time complexity.

Finally, we analyze the complexity of lines 11 to 19. First, we do not consider the operation in line 18. The loop starting in line 12 is performed in $O(m)$, and the complexity of each loop is in $O(\|S\|)$. The cost of the operations performed during each iteration is $O(\log(\|S\|))$ (lines 13 and 16 are both performed in

$O(1)$ and lines 14 and 15 in time $O(\log(\|S\|))$. The total time complexity of this part, without considering line 18, is then $O(m\|S\|\log(\|S\|))$. To complete the time complexity analysis, we show that the amortized complexity of line 18 is in $O(m\|S\|)$. Indeed, it follows from R2 that all entries removed in one step are consecutive in the total order on $R[a,c]$ defined in R2. Hence, if one call to line 18 removes k elements from $R[a,c]$, it can be done in $O((k+2)\log(m))$ time, as the successor of a given element can be retrieved in $O(\log(m))$ time. As every element of R is removed at most once during the whole algorithm, this leads to an amortized complexity of $O(m\|S\|\log(m))$ for line 18. Altogether, our algorithm solves computes $MCP(Q,T,S)$ in time $O(m\|S\|\log(m))$, using standard data structures and after a preprocessing in time $O(\|S\|\log(\|S\|))$ to compute the chainable areas and to sort J.

Additional remarks. If we consider that Q and T are sequences, or, as described in Section 2, unary trees, then each of the two trees has a single leaf and each seed is unambiguously defined by its root and border, which implies that$\|S\| = m$. There is only one $R[a,c]$, as $a = c = 0$, that contains $O(m)$ entries. Hence, all loops that were iterating on R have now a single iteration, which reduces the time complexity by a factor m to $O(\|S\|\log(m)) = O(m\log(m))$.

In the complexity analysis above, we followed the approach used for expressing the complexity of chaining in sequences, as we expressed the complexity only in terms of the size of the seeds. To express the complexity of our algorithm in terms of the size of Q and T, a finer analysis of the data structure R and of the number of different chainable areas leads to the following result: the worst-case space complexity of our algorithm is $O(|Q|^2|T|^2)$ (similar to the algorithm of Heyne et al.), and its worst-case time complexity is in $O(\|S\|\log(\|S\|) + |Q||T|\log(|T|)(|Q||T|+m))$, to compare with the complexity of the Heyne et al algorithm, which is in $O(\|S\|\log(\|S\|)+|Q|^2|T|^2(|Q||T|+m))$ [1]. This alternative complexity analysis is mostly of theoretical interest as in practice, for RNA analysis, one can expect that $m \ll |Q||T|$.

5 Conclusion

The current paper describes algorithms to solve chaining problems in ordered trees. With respect to similar problems in sequences, these methods exhibit a linear factor increase both in time and space. Chains so obtained can be used to speed-up RNA structure comparisons, as illustrated in [8,12].

A natural question related to chaining problems, that, as far as we know, has not been considered in the case of sequences, is to decide whether a given seed P of a set of seeds S belongs to *any* optimal chains or not. However a trade-off between quality and speed needs to be found. Indeed, identifying these *always optimal* seeds would probably ensure a chain of good quality, whereas the high complexity of these identifications might slow down the detection of similar structures in a large database.

Acknowledgements. Pacific Institute for Mathematical Sciences (PIMS, UMI CNRS 3069), Agence Nationale pour la Recherche project BRASERO (ANR-06-BLAN-0045), Natural Sciences and Engineering Research Council of Canada (NSERC), Multiscale Modeling of Plants associated team (INRIA).

References

1. Allali, J., Chauve, C., Ferraro, P., Gaillard, A.-L.: Efficient chaining of seeds in ordered trees. arXiv:1007.0942v1 [q-bio.QM] (2010)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. 215(3), 403–410 (1990)
3. Aluru, S. (ed.): Handbook of Computational Molecular Biology. CRC Press, Boca Raton (2005)
4. Backofen, R., Will, S.: Local sequence-structure motifs in RNA. J. Bioinform. Comput. Biol. 2(4), 681–698 (2004)
5. Demaine, E.D., Mozes, S., Rossman, B., Weimann, O.: An optimal decomposition algorithm for tree edit distance. ACM Trans. Algorithms 6(1), Article 2 (2009)
6. Gardner, P.P., Daub, J., Tate, J.G., et al.: Rfam: updates to the RNA families database. Nucleic Acids Res. 37(Database issue), D136–D140 (2009)
7. Gusfield, D.: Algorithms on Strings, Trees and Sequences. Cambridge University Press, Cambridge (1997)
8. Heyne, S., Will, S., Beckstette, M., Backofen, R.: Lightweight comparison of RNAs based on exact sequence-structure matches. Bioinformatics 25(16), 2095–2102 (2009)
9. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. J. Comput. Biol. 9(2), 371–388 (2002)
10. Joseph, D., Meidanis, J., Tiwari, P.: Determining DNA sequence similarity using maximum independent set algorithms for interval graphs. In: Nurmi, O., Ukkonen, E. (eds.) SWAT 1992. LNCS, vol. 621, pp. 326–337. Springer, Heidelberg (1992)
11. Lipman, D.J., Pearson, W.R.: Rapid and sensitive protein similarity searches. Science 227(4693), 1435–1441 (1985)
12. Lozano, A., Pinter, R.Y., Rokhlenko, O., Valiente, G., Ziv-Ukelson, M.: Seeded tree alignment. IEEE/ACM TCBB 5(4), 503–513 (2008)
13. Ohlebusch, E., Abouelhoda, M.I.: Chaining Algorithms and Applications in Comparative Genomics. In: Handbook of Computational Molecular Biology. CRC Press, Boca Raton (2005)
14. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. PNAS 85(8), 2444–2448 (1988)
15. Pedersen, J.S., et al.: Identification and classification of conserved RNA secondary structures in the human genome. PLoS Comput. Biol. 2(4), e33 (2006)
16. Shapiro, B.A., Zhang, K.: Comparing multiple RNA secondary structures using tree comparisons. CABIOS 6, 309–318 (1990)
17. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM J. Comput. 18(6), 1245–1262 (1989)

Appendix

Time Complexity of Algorithm 2

To establish the worst-case complexity of Algorithm 2, we have to study the cost of the algorithm for each f values. To ease the reading, we denote by n_1 the size of Q and n_2 the size of T. Without loss of generality, we furthermore assume that $n_2 \leq n_1$.

Following invariants $R1$ and $R2$, each list of R contains at most $min(m, n_2)$ elements, as there are no $(y, s), (y', s') \in R[a, c]$ s.t. $r_y^y = r_{y'}^{y'}$, and $|X| \leq min(\|S\|, n_1 n_2)$. Thus, in the worst-case, we have at most $O(n_1^2 n_2^2)$ different chainable areas, $|R| = O(n_1 n_2)$, for all (a, c): $|R[a, c]| = O(n_2)$ and $|X| = O(n_1 n_2)$.

$f = -1$ line 5: Over the whole execution of the algorithm each $M[a, l(r_j) - 1, c, l(r_j^j) - 1]$ is computed only once for all possible quadruplets as there is no (i, f, j), $(i', f', j') \in J$ such that $(l(r_j), l(r_j^j)) = (l(r_{j'}), l(r_{j'}^{j'}))$. Each computation require a search in $R[a, c]$ that can be done in $O(log(n_2))$. Thus, the total time complexity for this case is $O(n_1^2 n_2^2 log(n_2))$.

$f = 0$ line 8: The computation line 10 can be store in a dedicated array M' such that the best chain of the area (a, b, c, d) is computed only once. Thus, over all the execution of the algorithm, each different chainable area requires a search into a $R[a, c]$ and the total time complexity for this case is $O(\|S\| + n_1^2 n_2^2 \log(n_2))$.

$f = 1$ line 11: This case is run once peer seeds, so $O(m)$ times. Each run costs $O(n_1 n_2 \log(n_2))$ and the total time complexity is $O(m n_1 n_2 \log(n_2))$.

From above, we conclude that the worst-case time complexity of our algorithm is

$$\begin{aligned}
&O(\|S\| \log(\|S\|) + n_1^2 n_2^2 log(n_2) + \|S\| + n_1^2 n_2^2 \log(n_2) + m n_1 n_2 \log(n_2)) \\
&= O(\|S\| \log(\|S\|) + n_1 n_2 \log(n_2)(n_1 n_2 + m) + \|S\|) \\
&= O(\|S\| \log(\|S\|) + n_1 n_2 \log(n_2)(n_1 n_2 + m))
\end{aligned}$$

which represents an improvement of the worst-case complexity of Heyne et al. algorithm [8].

To conclude, we can merge the worst-case complexity analysis with the time complexity analysis of section 4.2 leading to the following time complexity for Algorithm 2:

$$\begin{aligned}
O(\ &\|S\| && \text{computing the chainable areas} \\
&+\|S\| \log(\|S\|) && \text{sorting the areas} \\
&+ \min(m, n_1 n_2) \times \min(\|S\|, n_1 n_2) \times log(\min(m, n_2)) && f = -1 \text{ case} \\
&+\|S\| + \min(\|S\|, n_1^2 n_2^2) \times \log(\min(m, n_2)) && f = 0 \text{ case} \\
&+m \times \min(\|S\|, n_1 n_2) \log(\min(m, n_2)) && f = 1 \text{ case}
\end{aligned}$$

as $|X| \leq \min(\|S\|, n_1 n_2)$ and $|R[a, c]| \leq \min(m, n_2)$ for all a, c.