Tree Alignment: Algorithms and Applications

Sitong Li, Shabnam Shabni

Dec, 15, 2018

Department of Computer Science, University of Western Ontario

Abstract

Similarity between trees is an important problem in various areas of computer science, including computational biology, geometric computing and cluster analysis [1, 2, 3]. Many approaches have been proposed to measure similarity between trees, such as tree edit distance, largest common subtrees, and transferable ratio [4, 5, 6]. In this report, we describe a feasible and realistic measure: tree alignment that is first proposed by T. Jiang, L. Wang and K. Zhang [7]. This report will introduce the classical Jiang-Wang-Zhang's Algorithm, which achieves general time complexity of $O(|T_1| \cdot |T_2| \cdot (deg(T_1) + deg\ (T_2))^2)$ and average time complexity of $O(|T_1| \cdot |T_2|)$ for ordered trees [7, 8]. While for unordered trees, tree alignment is MAX SNP-hard. Besides, average time complexity for the alignment of two RNA secondary structures is the same as the average time complexity for tree alignment. For "similar" trees, this algorithm can be modified by only considering the so-called d-relevant pairs of subtrees and subforests, with time complexity of $O((|T_1| + |T_2|)\log(|T_1| + |T_2|)(\deg(T_1) + \deg(T_2))^4 d^2)$[9]. Yet another application is to extract fields from HTML search result by combining tree and string alignment algorithms. The parameterization of cost function, in contrast to previous approaches that use hand tune approach, is fulfilled by a machine learning technic called support vector machine (SVM) [36].

Keywords: Tree Alignment, Jiang-Wang-Zhang's Algorithm, Dynamic Programming, MAX SNP-hard, Similar Trees, Average Complexity, HTML, Support Vector Machine.

1. Background

Tree and forest, as *Definition 1.1* describes, are widely used data structures in computer science. Tree structure can be defined to represent or capture the characteristic of various types of data, such as RNA secondary structure, natural language parsing and image segmentation [1, 10, 11]. Hence, it is natural to study the problem of similarity between trees.

Definition 1.1. *(Trees and forests). A tree is a node (called the root) connected to an ordered sequence of disjoint trees. Such a sequence is called a forest [12].*

Approach	Algorithm and reference	Time complexity		Space complexity	
		Ordered trees	Unordered trees	Ordered trees	Unordered trees
Tree edit distance	Zhang and Shasha's algorithm'89 [4]	$O(mn\times colldepth(T_1)\times colldepth(T_1))$	MAX SNP-hard	$O(mn)$	
	Klein's algorithm'98 [13]	$O(m^2 n\log n), m\le n$		$O(m^2 n\log n), m\le n$	
	Demaine's algorithm'09 [14]	$O\left(m^2 n(1+\log\frac{n}{m}\right), m\le n$		$O(mn)$	
Tree alignment	Jiang–Wang–Zhang's algorithm'95 [7]	$O(mn\times(\deg(T_1)+\deg(T_2))^2)$	MAX SNP-hard	$O(mn\times(\deg(T_1)+\deg(T_2)))$	
	Modified for similar trees'01 [9]	$O((m+n)\times\log(m+n)\times(\deg(T_1)+\deg(T_2))^4\times d^2$		$O((m+n)\times\log(m+n)\times(\deg(T_1)+\deg(T_2))^4\times d^2$	
Tree inclusion	Kilpel*äinen* and Mannilaalgorithm'95 [15]	$O(mn)$	NP-complete	$O(mn)$	

*Note: $|T_1|=m$, $|T_2|=n$; $colldepth(T)=\min\{depth(T),\#leaves(T)\}$; $\deg(T)$: The maximum degree of tree T;
d: The maximum # of blank symbol used for optimal alignment of two ordered trees T_1 and T_2.

Table 1.1 *The general view of the time complexity and space complexity of representative algorithms for several approaches used to measure similarity between trees.*

The comparison between two trees, has been studied for years, especially on its natural applications in bioinformatics, more specifically, in the analysis of RNA structure area. *Table 1.1* lists several already developed methods to measure similarity between trees and their corresponding information. Actually, there exist other methods such as largest common subtree and maximum agreement subtrees [16, 17]. Among those methods, tree edit and tree alignment are two approaches are developed both on substitution/insertion/deletion operations on nodes. During a substitution operation, the label of the node is changed while deletion operation includes deleting a node and assigning all its children to become children of its father. Insertion operation occurs between two siblings, all the nodes among these siblings are assigned as children of the new node. Tree edition problem concerns minimization of the cost of substitution, insertion and deletion operations to change T_1 to T_2. While tree alignment only contains substitution and insertion operations as defined in *Definition 1.2* and an example in *Figure 1.1*. The objective is also to minimize the total cost of operations, which is the sum of

the scores of each apposing label defined by an affined function. The tree alignment distance problem is a special case of the tree editing problem. In fact, it corresponds to a restricted edit distance where all insertions must be performed before any deletions.

Definition 1.2. *(Alignment between two labeled trees). 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 [9].*

For sequences, edit and alignment are equivalent in terms of complexity [18]. While it is not the same situation for trees, which is easily found in *Table 1.1*. Since tree edit distance problem has been covered in the course, here we are going to figure out tree alignment problem, including introduction of several algorithms, analysis of time complexity and an application.

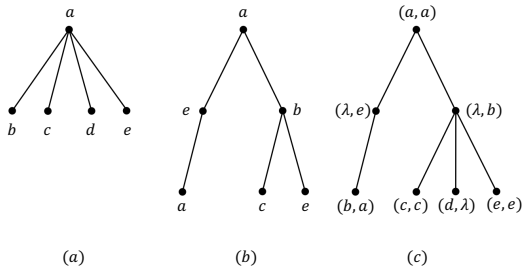


Figure 1.1 *Let* $\Sigma = \{a, b, c, d, e\}$, *scoring function* μ: $\mu(x, x) = 3$, $\mu(x, y) = -1$, $\mu(x, \lambda) = \mu(\lambda, x) = -2$, $\mu(\lambda, \lambda) = -2$ *for all* $x, y \in \Sigma$ *with* $x \neq y$. *Then the score of the alignment in (c) of the two ordered trees shown in (a) and (b) is equal to 2.*

We will start with the classical Jiang-Wang-Zhang's algorithm, which is proposed in 1995 [7]. The main idea of the algorithm is dynamic programming utilizing two formulae for computing alignment distance between two subtrees and subforests. The trick is that there is no need to compute all pairs of subtrees separately. The superiority of the algorithms shows in two aspects. First, this algorithm performs better than tree edit distance algorithms when trees are unordered but with bounded degree. Second, this algorithm has a considerably faster average time complexity ($O(|T_1| \cdot |T_2|)$) than tree edit distance algorithms ($O(|T_1|^{\frac{3}{2}}|T_2|^{\frac{3}{2}})$) [8, 19]. The average time complexity still holds when applied to align RNA secondary structure [1].

A modified version of the original algorithm is applied based on assumption that two trees are similar. The definition of the so called "similar" will be defined. Then a concept of d-relevance is generated to filter useless calculation which concerns pairs that are not d-relevance. By this trick the time complexity of original algorithm can be reduced for similar trees.

Later, we will introduce an algorithm for extracting fields form HTML as the result during the search through the internet. The algorithm matches the data which is semantically related to each other by composing a tree alignment algorithm and also domain specific feature extraction. The improvement of a previous algorithm is done in this report but there is a huge different between this algorithm and others.in this approach, we have used support vector machine rather than using hand tuned approach to parameterize the tree.

2. Jiang-Wang-Zhang's tree alignment algorithm

2.1 The algorithm

The main idea of the algorithm is dynamic programming similar to tree edit distance but different in some way. The formulae used in this algorithm are different from tree edit distance algorithm. The three lemmas used for computing alignment distance of two ordered trees are as following:

Lemma 2.1. $D(\theta,\theta)=0$; $(F_1[i],\theta)=\sum_{k=1}^{m_i} D(T_1[i_k],\theta)$; $D[T_1[i],\theta]=D[F_1[i],\theta]+\mu(l_1[i],\lambda)$; $D(\theta,F_2[j])=\sum_{k=1}^{n_j} D(\theta,T_2[j_k])$; $D(\theta,T_2[j])=D(\theta,F_2[j]])+\mu(\lambda,l_2[j])$.

Lemma 2.2. *Computing alignment distance of two ordered trees.*

$$D(T_1[i],T_2[j])$$
$$=min\begin{cases} D(\theta,T_2[j])+min_{1\le r\le n_j}\{D(T_1[i],T_2[j_r])-D(\theta,T_2[j_r])\}, \\ D(T_1[i],\theta)+min_{1\le r\le m_i}\{D(T_1[i_r],T_2[j])-D(T_1[i_r],\theta)\}, \\ D(F_1[i],F_2[j])+\mu(l_1[i],l_2[j]). \end{cases}$$

Lemma 2.3. *Computing alignment distance of two ordered forests. For any* s,t *such that* $1\le s\le m_i$ *and* $1\le t\le n_j$,

$$D(F_1[i_1,i_s],F_2[j_1,j_t])$$
$$=min\begin{cases} D(F_1[i_1,i_{s-1}],F_2[j_1,j_t])+D(T_1[i_s],\theta), \\ D(F_1[i_1,i_s],F_2[j_1,j_{t-1}])+D(\theta,T_2[j_t]), \\ D(F_1[i_1,i_{s-1}],F_2[j_1,j_{t-1}])+D(T_1[i_s],T_2[j_t]), \\ \mu(\lambda,l_2[j_t])+min_{1\le k\le s}\{D(F_1[i_1,i_{k-1}],F_2[j_1,j_{t-1}])+D(F_1[i_k,i_s],F_2[j_t])\}, \\ \mu(l_1[i_s],\lambda)+min_{1\le k\le t}\{D(F_1[i_1,i_{s-1}],F_2[j_1,j_{k-1}])+D(F_1[i_s],F_2[j_k,j_t])\}. \end{cases}$$

The correctness of these three lemmas are explicitly proved in the paper [7]. *Lemma 2.1* is immediately obtained by definition, and it provides base cases of *Lemma 2.2* and *Lemma 2.3*. Here, we use sketches to show how *Lemma 2.2* and *2.3* are generated. As illustrated in *Figure*

2.1, there are three cases resulting in three equations in *Lemma 2.2* when calculating the alignment distance between two trees. The three cases cover all situations of the possible labels of root of alignment $\mathcal{A}$. From *Lemma 2.2* we can see that $D(F_1[i], F_2[j])$ is prerequisite for computing $D(T_1[i], T_2[j])$. The calculation of alignment distance of two subforests also consists three cases as *Figure 2.2* shows. It is notable that in case 2, where the label of the rightmost tree in alignment $\mathcal{A}$ is labeled by $(l_1[i_s], \lambda)$, then $T_1[i_s]$ is aligned with the subforest $F_2[j_{t-k+1}, j_t], 0 \leq k \leq t$. This case can be divided into three subcases depending on the value of k. The subcase 2.2 when $k = 1$ is actually a special situation in case 1, so there is no need to add another formula based on it. Combining *Lemma 2.2* and *3.3*, it can be deducted that to obtain distance of each pair of subtrees $T_1[i]$ and $T_2[j]$, we only have to align $F_1[i]$ with each subforest of $F_2[j]$, and conversely align $F_2[j]$ with each subforest of $F_1[i]$. The reason comes from the fact that all the intermediate terms can be obtained as by-products. Finally, The *Procedure 2.1* and *Algorithm 2.1* can be employed to compute the alignment distance.

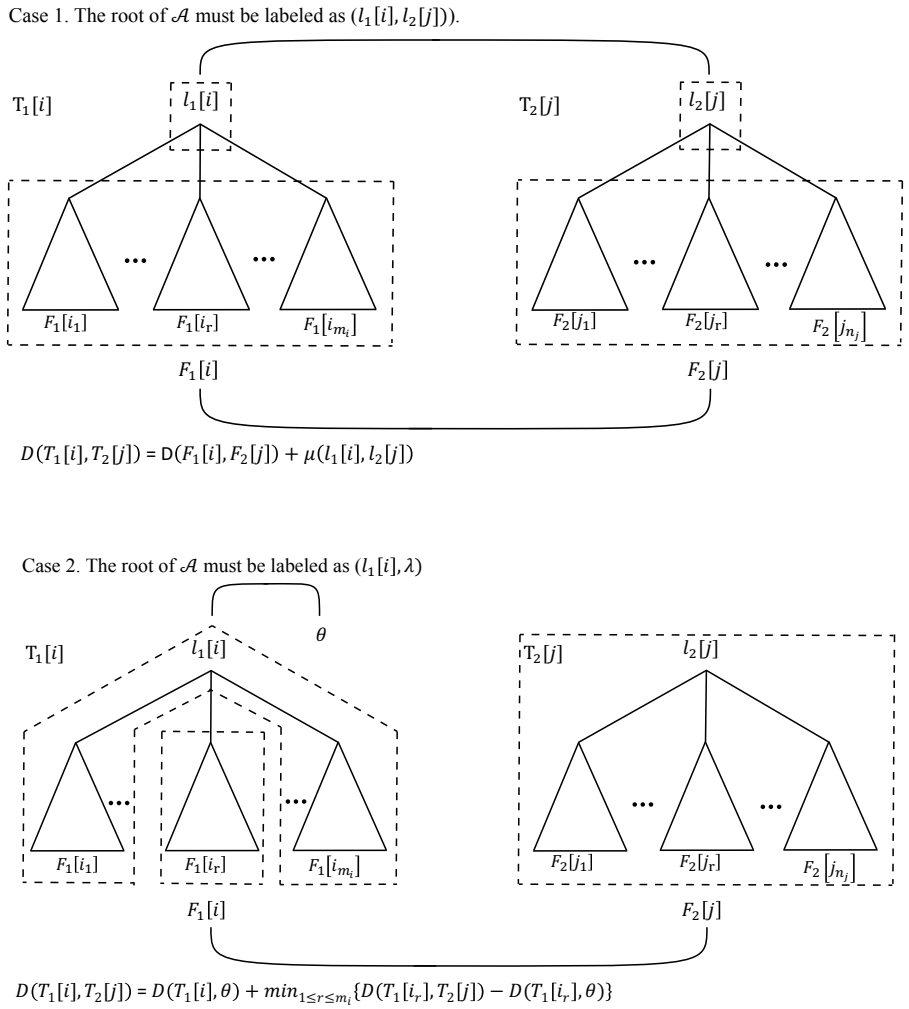
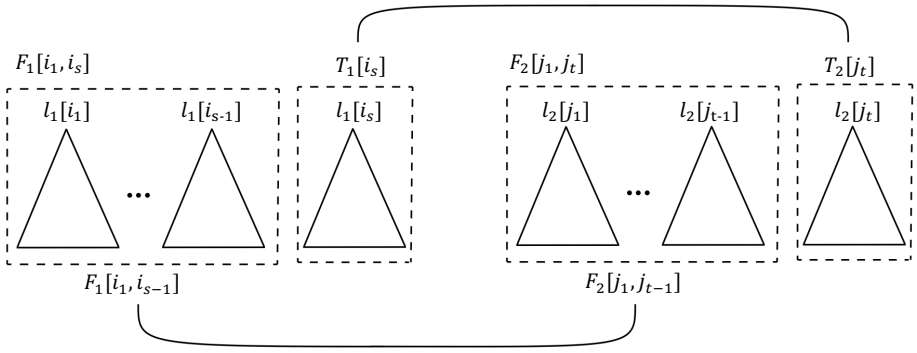


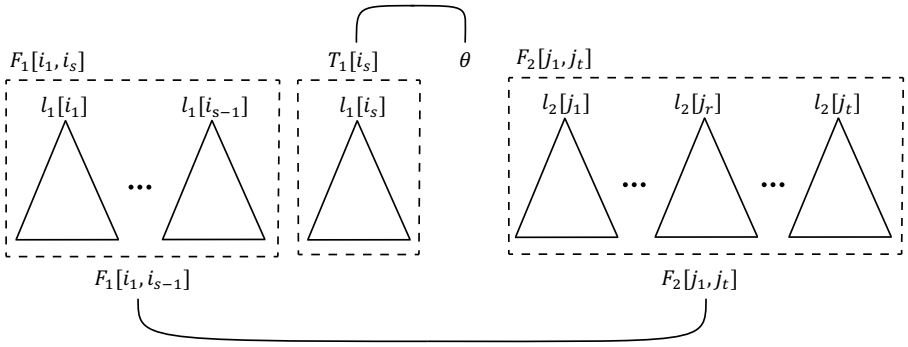
Figure 2.1 *The illustration of Lemma 2. Case 3 is similar to Case 2, the root of $\mathcal{A}$ is labeled as $(\lambda, l_2[j])$ instead.*

Case 1. The label is $(l_1[i], l_2[j])$.



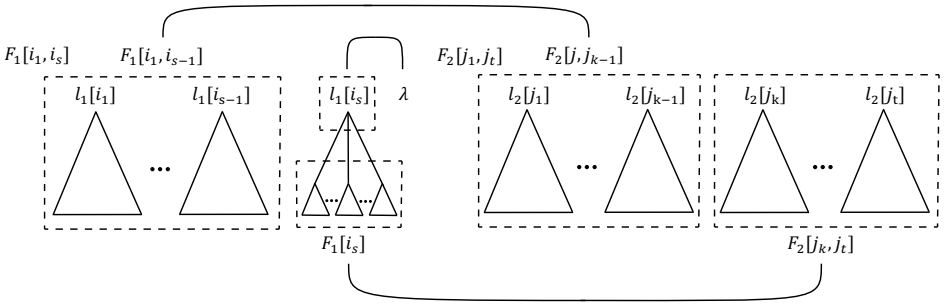
$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = D(F_1[i_1, i_{s-1}], F_2[j_1, j_{t-1}]) + D(T_1[i_s], T_2[j_t])$$

Case 2.1. The label is $(l_1[i_s], \lambda)$, and $k = 0$.



$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = (F_1[i_1, i_{s-1}], F_2[j_1, j_t]) + D(T_1[i_s], \theta)$$

Case 2.3. The label is $(l_1[i_s], \lambda)$, and $k \geq 2$.



$$D(F_1[i_1, i_s], F_2[j_1, j_t]) = \mu(l_1[i_s], \lambda) + min_{1 \leq k \leq t}\{D(F_1[i_1, i_{s-1}], F_2[j_1, j_{k-1}]) + D(F_1[i_s], F_2[j_k, j_t])\}$$

Figure 2.2. *The illustration of Lemma 3. Case 3 is similar to Case 2, the labeled is* $(\lambda, l_2[j_t])$ *instead.*

Procedure 2.1. Computing $\{D(F_1[i_s, i_p], F_2[j_t, j_q]) \mid s \leq p \leq m_i,\ t \leq q \leq n_j\}$ for fixed s and t.

Input: $F_1[i_s, i_{m_i}]$ and $F_2[j_t, j_{n_j}]$.

$D(F_1[i_s, i_{s-1}], F_2[j_t, j_{t-1}]) \coloneqq 0$;

for $p \coloneqq s$ to m_i

$D(F_1[i_s, i_p], F_2[j_t, j_{t-1}]) \coloneqq D(F_1[i_s, i_{p-1}], F_2[j_t, j_{t-1}]) + D(T_1[i_p], \theta)$;

for $q \coloneqq t$ to n_j

$D(F_1[i_s, i_{s-1}], F_2[j_t, j_q]) \coloneqq D(F_1[i_s, i_{s-1}], F_2[j_t, j_{q-1}]) + D(\theta, T_2[j_q])$;

for $p \coloneqq s$ to m_i

for $q \coloneqq t$ to n_j

Compute $D(F_1[i_s, i_p], F_2[j_t, j_q])$ as in *Lemma 3*.

Output: $\{D(F_1[i_s, i_p], F_2[i_t, i_q]) | s \le p \le m_i, t \le q \le n_j\}$.

Algorithm 2.1. Computing $D(T_1, T_2)$.

Input: T_1 and T_2.

$D(\theta, \theta) \coloneqq 0$;

for $i \coloneqq 1$ to $|T_1|$

Initialize $D(T_1[i], \theta)$ and $D(F_1[i], \theta)$ as in *Lemma 1*;

for $j \coloneqq 1$ to $|T_2|$

Initialize $D(\theta, T_2[j])$ and $D(\theta, F_2[j])$ as in *Lemma 1*;

for $i \coloneqq 1$ to $|T_1|$

for $j \coloneqq 1$ to $|T_2|$

for $s \coloneqq 1$ to m_i

Call Procedure 1 on $F_1[i_s, i_{m_i}]$ and $F_2[j]$;

for $t \coloneqq 1$ to n_j

Call Procedure 1 on $F_1[i]$ and $F_2[j_t, j_{n_j}]$;

Compute $D(T_1[i], T_2[j])$ as in *Lemma 2*.

Output: $D(T_1[|T_1|], T_2[|T_2|])$.

2.2 The time complexity

For any input $F_1[i_s, i_{m_i}]$ and $F_1[j_t, j_{n_j}]$, running time of *Procedure 2.1* is bounded by $O(m_i \cdot n_i \cdot (m_i + n_j))$. So, for each pair of i and j, *Algorithm 1* spends $O(m_i \cdot n_j \cdot (m_i + n_j)^2)$. Therefore, the time complexity of *Algorithm 2.1* is $\sum_{i=1}^{|T_1|} \sum_{j=1}^{|T_2|} O(m_i \cdot n_j \cdot (m_i + n_j)^2) \le O(|T_1| \cdot |T_2| \cdot (deg(T_1) + deg\ (T_2))^2)$. If both T_1 and T_2 have bounded degrees, the time complexity becomes $O(|T_1| \cdot |T_2|)$.

2.3 The alignment of unordered tree

This algorithm can be naturally extended to unordered trees. If the degrees of the unordered trees are bounded, we can use a modified version of *Algorithm 2.1*. The only difference is in the computation of distance between subforests as *Lemma 2.3*. We can see that since there is no order of the subtrees in each subforests, they can be divided randomly then aligned, as *Lemma 2.4* indicates. Since degrees are bounded, this can be computed in polynomial time. Actually, when both $T_1$and T_2 are binary trees, the time complexity of the algorithm is $O(|T_1| \cdot |T_2|)$.

Lemma 2.4. *Computing alignment distance of two unordered forests with bounded degree. For each* $A \subseteq \{T_1[i_1], \ \dots, T_1[i_{m_i}]\}$ *and each* $\mathrm{B} \subseteq \{T_2[j_1], \ \dots, T_1[j_{n_j}]\}$,

$D(A, B)$

$$= min \begin{cases} min_{T_1[i_p] \in A.\, T_2[j_q] \in B} D\big(A - \{T_1[i_p]\},\ B - \{T_2[j_q]\}\big) + D\big(T_1[i_p], T_2[j_q]\big), \\ min_{T_1[i_p] \in A.\ B' \subseteq B} D\big(A - \{T_1[i_p]\},\ B - B'\big) + D\big(F_1[i_p], B'\big) + \mu(l_1[i_p], \lambda), \\ min_{A' \subseteq A,\, T_2[j_q] \in B} D\big(A - A',\ B - \{T_2[j_q]\}\big) + D\big(A', F_2[j_q]\big) + \mu\big(\lambda, l_2[j_q]\big). \end{cases}$$

If one of the trees has an arbitrary degree, the alignment problem becomes MAX SNP-hard. To prove MAX-SNP hardness of a problem, we need to find a problem in this class that can linearly reduce (or L-reduce) to it [20]. The authors use an instance of problem Maximum 3-bounded Covering by 3-Sets (MAX 3SC-3), which belongs to MAX SNP-hard problem [21]. The instance is marked by I, which is like: $A = \{a_1, a_2, \dots, a_m\}$ and $C = \{c_1, c_2, \dots, c_n\}$, where $c_{\mathrm{i}} = \{c_{i,1}, c_{i,2}, c_{i,3}\}$ and $c_{i,j} \in A$. They assume w.l.o.g. that $m \leq 3n$. Let C' be a largest exact partial cover of A.

Definition 2.1. ***(****Problem MAX 3SC-3).*

Input: *A collection of subsets* $C = \{c_1, c_2, \dots, c_n\}$ *of a finite set* $A = \{a_1, a_2, \dots, a_m\}$, *where every subset* $c \in C$ *contains three elements and each element appears in at most 3 subsets.*

Output: *A subcollection composed of mutually disjoint subsets (exact partial cover).*

Measure of a solution: *The number of subsets in the returned subcollection.*

Objective: *Maximizing the measure of the returned solution.*

The authors construct two trees T_1 and T_2, which can be constructed from I in polynomial time. The transformation from MAX 3SC-3 to alignment of unordered trees is called f, so the

alignment problem is marked by $f(I)$. Let $\mathcal{A}'$ be an optimal alignment of T_1 and T_2, which is to say that $OPT\big(f(I)\big) = v(\mathcal{A}')$. The relation of $v(\mathcal{A}')$ and C' can be obtained as *Lemma 2.5*.

Lemma 2.5. $v(\mathcal{A}') = |T_1| + |T_2| - 2(|C'| + 5n + 1)$.

Given an alignment $\mathcal{A}''$ of T_1 and T_2, we can use a polynomial-time algorithm g to construct an exact partial cover C'', which is a solution of *I*. The relationship between $v(\mathcal{A}'')$ and $C''$is as *Lemma 2.6* shows.

Lemma 2.6. $v(\mathcal{A}'') \geq |T_1| + |T_2| - 2(|C''| + 5n + 1)$.

Using *Lemma 2.5* and *2.6*, The correctness of the two criteria for L-reduction from I to the tree alignment problem can be proved. Therefore, the problem of determining the optimal alignment of two unordered tree is MAX SNP-hard.

3. Tree alignment algorithm between similar ordered tree

3.1 d-relevant pairs

This algorithm is inspired by optimal alignment between similar sequences [22]. Two sequences are similar when they "look alike" in some sense. The time complexity of algorithm for optimal alignment between two similar sequences is bounded by $O(dn)$, where n is the length of the longer sequence and d is the difference between the maximum possible score and the optimal score. Thus, the higher the similarity, the faster the algorithm. This implies an improvement comparing to $O(n^2)$ in original algorithm, when d is relatively small.

In algorithm for alignment between similar ordered tree, the general idea is just a modification of the dynamic programming algorithm in Jiang–Wang–Zhang's algorithm. Only the so-called d-relevant pairs of subtrees or subforests are considered [9]. A pair of subtrees $(S[u], S[v])$ is d-relevant if both the following two conditions are satisfied:

1. The difference of the number of nodes in $S[u]$ and $S[v]$ is bounded by d;
2. The difference of the number of leaves that are to the left of the leaves of $S[u]$ and $S[v]$ are bounded by d.

d-relevant pair of subforests is defined similarly. Besides, the concepts of d-descendent and d-ancestor for subtrees and subforests are introduced too. With simple deduction, *Lemma 3.1* to *3.3* are easily deducted, which can be used to bound the number of d-relevant pairs of

subtrees as illustrated in *Theorem 3.1*. In similar manner, the number of d-relevant pairs of subforests in which at least one is complete is bounded by $O(n \cdot d^2(maxdeg)^2)$ (*Theorem 3.2*).

Lemma 3.1. *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 a descendent (or, ancestor) of* v *of* T *then* $T[w]$ *is a 2d-descendant (or, 2d-ancestor) of* $T[v]$.

Lemma 3.2. *For a node* u *of an ordered tree* S, *the number of d-ancestors of* $S[u]$ *is at most d.*

Lemma 3.3. *Let* $\{(S[u], T[v_i])\}_{i=0}^{l}$ *be a sequence of distinct d-relevant pairs in two ordered trees* S, T *such that for any* $0 \leq i, j \leq l$, v_i *is not descendent of* v_j. *Then,* $l \leq 2d$ *holds.*

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

3.2 Preprocessing

Before application of the algorithm, some preprocessing is needed. First, all vectors in form of $(|S[u]|, |L(S,u)|)$ *and* $(|T[v]|, |L(T,v)|)$ are calculated. This can be finished in linear total time by using the Eulerian tour technique. Then, all the vectors are inserted into a standard range search data structure, such as layered range tree, which will take $O(|T| \cdot \log |T|)$ [23]. Query the data structure with square centered at $(|S[u]|, |L(S,u)|)$ having side length $2d$ will take $O(\log n + d^2)$ time according to *Theorem 1*. Thus, for a tree with n nodes, all d-relevant pairs of subtrees can be reported in $O(n(\log n + d^2))$ time. Similarly, all d-relevant pairs of subforests, in which at least one is complete can be reported in $O(n \cdot (maxdeg)^2 \cdot (\log n + d^2))$ time.

Once all d-relevant pairs are obtained, all d-relevant pairs of subtrees are inserted into a balanced binary search tree (BST) $\mathcal{B}_1$, and all d-relevant pair of subforests are inserted into a balanced BST $\mathcal{B}_2$. This operation will take $O(n \cdot d^2(maxdeg)^2 \cdot \log(n \cdot d^2(maxdeg)^2))$ time. Therefore, the total time for preprocessing is: $O(n \cdot (maxdeg)^2 \cdot (\log n + d^2) + n \cdot d^2(maxdeg)^2 \cdot \log(n \cdot d^2(maxdeg)^2)) = O(n \cdot \log n \cdot (maxdeg)^2 \cdot d^2)$.

3.3 The algorithm

According to *Lemma 2.2* and *2.3* in Jiang-Wang-Zhang's algorithm, when the left handside is d-relevant, the component on the right handside which yields the left handside must also be d-relevant. This means we only need to test each of the components of the sums on the right handside for members in $\mathcal{B}_1$ and $\mathcal{B}_2$. Querying the balanced BST takes $O(\log n)$ time. Consequently, determine scores on left won't cost more than the original algorithm multiplied by $O(log\, n)$. According to *Lemma 2.2* and *Lemma 2.3,* with already known scores for pairs of smaller subtrees or subforests, the cost for determine the score for a pair of subtrees or subforests is $O(deg(z) \cdot (maxdeg)^2)$, where z is node in S or T. So, for d-relevant pairs, its $O(deg(z) \cdot (maxdeg)^2 \cdot log\, n)$. Therefore, the total time is $O(\sum_{z \epsilon S \cup T} deg(z) \cdot (maxdeg)^2 \cdot log\, n \cdot d^2 (maxdeg)^2)$, which is $O(nd^2(maxdeg)^4\, log\, n)$.

One thing that needs to be noticed is that this algorithm cannot be generalized to unordered trees directly. Since the proof of lemmas used for bounding the number of d-relevant pairs rely on the ordering of the trees.

4. Average time complexity of Jiang Wang Zhang's Algorithm

Here, we will show that the average time complexity for tree alignment problem is $O(nm)$ and also the average time complexity of alignment algorithm for RNA is $O(|S_1|.|S_2|)$, S_1 and S_2 are the sequences [24, 25, 26]. Before the definition of the alignment distance, some notations are necessary to define the recurrence relation which was presented by [7]. $p \circ s$ shows the decomposition of a forest (or a tree is composed by the concatenation of two subforests p and s). γ Shows cost function and $\gamma(-, v)$, $\gamma(v, -)$, $\gamma(v, v')$ are the notations for insertion, deletion and substitution cost function.

Definition 4.1. *(Recurrence relation). Tree alignment distance between two trees follows these rules which is called recurrence relation.*

We assume that we have two close sub forests like $v(f) \circ g$ and $v(f') \circ g'$ and want to align them. According to recurrence relation we have

$$align(v(f) \circ g, v'(f') \circ g') =$$

$$min \begin{Bmatrix} \gamma(v, -) + \min\{align\,(f,\, p') + align\,(g, s') | such\ that\ in\ this\ case\ p' \circ s' equals\ to\ v'(f') \circ g'\} \\ \gamma(-, v) + \min\{\{align\,(p, f') + align\,(s, g') | such\ that\ in\ this\ case\ p \circ s\ equals\ to\ v(f) \circ g\}\} \\ \gamma(v, v') + \gamma(f, f') + \gamma(g, g') \end{Bmatrix}$$

The alignment distance consists of three operations, first, second and third ones are related to insertion, deletion and substitution in order. For insertion part, a cost operation for insertion and the minimum alignment between two sub forests (regarding with the definition of alignment, we are looking for the minimum cost function for transformation) are needed. In deletion part, a cost function for deletion is needed. In addition, the minimum alignment between two sub forests$p\ o\ f'$ shows the composition of two sub forests p andf'. The third part illustrates that all the sub forests must be substituted with each other. In tree alignment distance algorithm, the computation of alignment of T_1 and T_2 are done by the mentioned recurrence relation. The result is preserved and then the recurrent stage will give the alignment and since last stage is related to the largest size of the tree, the time complexity is linear and can be ignored in comparison to the alignment stage. To get the time complexity, we only focus on alignment stage.

In this part we want to compute the average time complexity for alignment algorithm. To get this goal, we mention and prove *Theorem 4.1*.

Theorem 4.1. *The average time complexity between T_1 and T_2 is $O(|T_1|.|T_2|)$. And || shows the number of nodes in the trees.*

By proving this theorem, we can achieve what we were looking for. To prove this theorem we use *Lemma 4.1* to *4.4* [25, 26], then we can take advantage of the outcome of lemmas to prove the theorem.

Lemma 4.1. *Assume that there are two lemmas, like T_1 having $n+1$ nodes and T_2 having $m+1$ nodes. The average number of operations to compute alignment distance between them is* $\frac{\sum_{|T_1=n+1|}\sum_{|T_2=m+1|}N(T_1,T_2)}{\frac{1}{n+1}\binom{2n}{n}\frac{1}{m+1}\binom{2m}{m}}$, *where* $N(T_1,T_2) = |\ \mathcal{s}(T_1)\ |x\sum_{f\in\mathcal{C}(T_2)}w(f) + \sum_{f\in\mathcal{s}(T_1)}w(f)\ x\ \mathcal{C}(T_2) + \sum_{f\in\mathcal{C}T_1}w(f)\ x|\mathcal{S}(T_2)|+|\ \mathcal{C}\ T_1|x\sum_{f\in\mathcal{S}T_2}w(f)$.

According to this lemma, it is obviously evident that the average number of operation will be achieved by dividing the number of operations by the total number of those, in the second part $|\mathcal{S}(T)|$ denotes the number of suffix sub forests of a tree and $|\mathcal{C}(T)|$ is called the number of closed sub forests. $w(f)$ Shows the number of trees in f, as an instance, the number of nodes does not have any parents. The proof of this lemma has been mentioned in [25, 26].

For alignment computation between T_1 and T_2 we need to compute sub forests F_1 and F_2 which belong to the union suffix sub forests of T_1, closed forests of T_2 and suffix sub forests T_2 and closed forest T_1. As we know $w(f)$ denotes the number of trees contained in subforest f to compute the alignment(F_1, F_2) we need find the minimum cost (best decomposition) between the possible decompositions, hence we need to compute as long as the width of each forest plus 1 possibility [1]. According to the first part of recurrence relation, we need to find the best decomposition of F_2 .which can be the concatenation of p_2 and s_2. It shows that there is $w\ (F_2) + 1$ possible decompositions. The same happens for the second line of the recurrence relation, we are looking for the best decomposition of F_1 like p_1 and s_2 among the $(F_1) + 1$ possible decomposition. For the last line of the recurrence relation, there is no need for the comparison and the number of possibilities is only one. Now the total number of operations to compute alignment of two trees (T_1, T_2) equals to sum of all the decompositions of F_1 and F_2: $\sum_{(F_1,F_2)\epsilon(\mathcal{S}(T_1)\times\mathcal{C}(T_2)\cup\mathcal{S}(T_2)\times\mathcal{C}(T_1))}(w(F_1) + w(F_2))$. By developing the summations and using the rules of union and intersection, the former summations equals to: $\sum_{F_1\in(\mathcal{S}(T_1)}\ \sum_{F_2\in\mathcal{C}(T_2)}(w(F_1) + w(F_2)) + \sum_{F_1\epsilon\mathcal{C}(T_1)}\sum_{F_2\epsilon\mathcal{S}(T_2)}(w(F_1) + w(F_2)) - \ \sum_{F_1\in(\mathcal{S}(T_1)}\sum_{F_2\epsilon\mathcal{S}(T_2)}(w(F_1) + w(F_2))$.

Lemma 4.2. *If T is a tree, then:*

1. $|\mathcal{S}(T)| = n_T + l_T - 1$, *$n_T$ is the number of nodes and l_T is the number of leaves in tree T.*
2. $\mathcal{C}(T) = 1 + \sum_{v\in T}\binom{d_v + 1}{2}$, *$d_v$ Is the degree of each node.in the other words, the number of children of each node.*
3. $\sum_{f\in\mathcal{S}(T)} w(f) = l_T + \sum_{v\in T}\binom{d_v + 1}{2}$.
4. $\sum_{f\in\mathcal{C}(T)} w(f) = \sum_{v\in T}\binom{d_v + 2}{3}$.

Lemma 4.3. $\sum_{|T|=n+1}\sum_{v\in T}\binom{d_v + 1}{2} = \binom{2n + 1}{n + 1}$ [28].

Lemma 4.4. $\sum_{|T|=n+1}\sum_{v\in T}\binom{d_v + 2}{3} = \binom{2n + 2}{n - 1}$.

The proof for *Lemma 4.3* and *4.4* is achieved by considering the *Lemma 4.2*. The proof of *Lemma 4.2* is in [25, 26] and the proof of *Lemma 4.3* and *4.4* also in [1].

Now, regarding to these lemmas, we can prove the theorem aforementioned before to reach the average time complexity for tree alignment problem.

Theorem 4.2. *The average time complexity between T_1 and T_2 is O ($|T_1|.|T_2|$). And $|T_1|.|T_2|$ shows the number of nodes in the trees.*

According to *Lemma 4.1*, when the number of nodes for T_1 is $n+1$ and the number of nodes for T_2 is $m+1$ the average number of required operations for the alignment tree problem is: $\frac{\sum_{|T_1|=n+1}\sum_{|T_2|=m+1}N(T_1,T_2)}{\frac{1}{n+1}\binom{2n}{n}\frac{1}{m+1}\binom{2m}{m}}$. By following the mathematical rules and replacement, we can get the $N(T_1,T_2)$. Now by considering the second part of *Lemma 4.1*, we reach the expression containing 4 main parts. T is for both T_1 and T_2.

1. In *Lemma 4.2* part 1, it was stated that the number of suffix sub forests and subtree is $n_T + l_T - 1$ for a tree which has $n+1$ nodes. Hence the average number of those are achieved by dividing by the total number of nodes: $\frac{\sum_{|T|=n+1}(n+1+l_T-1)}{\frac{1}{n+}\binom{2n}{n}}$ And the time complexity for this expression is $O(n)$.
2. Regarding with the *Lemma 4.2*, the number of closed subforests for a tree including $n+1$ nodes is 1+$\sum_{v\in T}\binom{d_v+1}{2}$, by dividing this expression by the total number of nodes, we can get the average number of closed sub forests: $\frac{\sum_{|T|=n+1}\left(1+\sum_{v\in T}\binom{d_v+1}{2}\right)}{\frac{1}{n+1}\binom{2n}{n}}$.

   Now, we can replace the equivalent value which was stated in *Lemma 4.3* for $1+\sum_{v\in T}\binom{d_v+1}{2}$ and also the summation of 1, so we can have: $\frac{\frac{1}{n+1}\binom{2n}{n}+\binom{2n+1}{n-1}}{\frac{1}{n+1}\binom{2n}{n}}$ $= 1 + \frac{(n+1)(2n+1)!n!n!}{(n-1)!(n+2)!2n!} = 1 + \frac{n(2n+1)}{n+2}$, which is also $O(n)$.
3. According to *Lemma 4.2,* the number of suffix sub forests including $n+1$ Nodes, is $l_T + \sum_{v\in T}\binom{d_v+1}{2}$ and trough dividing it by the total number of nodes, we will have: $\frac{\sum_{|T|=n+1}\left(l_T+\sum_{v\in T}\binom{d_v+1}{2}\right)}{\frac{1}{n+1}\binom{2n}{n}}$.

   By adding *Lemma 4.3* and replacing its equivalent value, we can get: $\frac{\sum_{|T|=n+1}l_T}{\frac{1}{n+}\binom{2n}{n}} + \frac{\binom{2n+1}{n-1}}{\frac{1}{n+1}\binom{2n}{n}}$ can be computed in $O(n)$.
4. According to *Lemma 4.2*, the number of closed sub forests in a tree with $n+1$ is $1+\sum_{v\in T}\binom{d_v+1}{3}$ by dividing that by the total number of nodes, we can

obtain the average number of operations, the *Lemma 4.4,* is added to replace its equivalent value:

$$\frac{\sum_{|T|=n+1|}\left(1+\sum_{v\in T}\binom{d_v+1}{3}\right)}{\frac{1}{n+1}\binom{2n}{n}} = \frac{\frac{1}{n+1}\binom{2n}{n}+\binom{2n+2}{n}}{\frac{1}{n+1}\binom{2n}{n}} = 1 + \frac{(n+1)(2n+2)!n!n!}{(n-1)!(n+3)!2n!}$$

$$= 1 + \frac{n(2n+1)(2n+2)}{(n+2)(n+3)} = O(n).$$

So far we have illustrated that the average number of operation to compute the alignment between two tress, is in order of the number of nodes for first and second tress.

4.2 RNA secondary structure

In this part, we talk about RNA secondary structure and also introduce the operations [24] done for edit and alignment problems. In additionally, we define the algorithm for alignment RNA secondary structures [24, 25, 26] and the average complexity in this algorithm is proved as well.

Definition 4.2. An RNA secondary structure without pseudoknots is a sequence of alphabet [1] like *S* and a set *P* including a pair of positions (i, j) for insertions. The alphabets belonging to *S* are also called arcs. If a letter incident to an arc, it is called pair-base. Otherwise it is called non-pair base.

An algorithm is introduced to build a tree form a sequence of letters as *Figure 4.1(a)* shows. In the first step of algorithm, when the letter is not incident, a leaf is added to the current letter and it becomes the current vertex [1], and the next vertex is added to this node as the right siblings. The *Figure 4.1(b)* shows this step of algorithm, C is a non-pair base letter, and it is added to the tree as a leaf having the label of the current letter.

The second part of the algorithm says if the current letter is incident to the left part of the arc, an internal node is added. The label of this node is the concatenation of the left and right label of the arc. And the next vertex will be added as the child of this node [1]. By continuing the algorithm, we can see that, after letter C, letter A has incident to an arc, the label is AU, and this letter will be added as an internal node. CG and GC are the next arcs and according to the algorithm, they will be added as internal nodes as *Figure 4.1(c).*

The next letters, is U. Regarding with the first step of the algorithm, it will like a leave and will be added to the tree. UG and UA are those arcs will be added to the tree according to the second

part of the algorithm, C, C, U, A, G, U are the letters are added to the tree base on the first part of the algorithm. *Figure 4.1(d)* is achieved by following these steps.

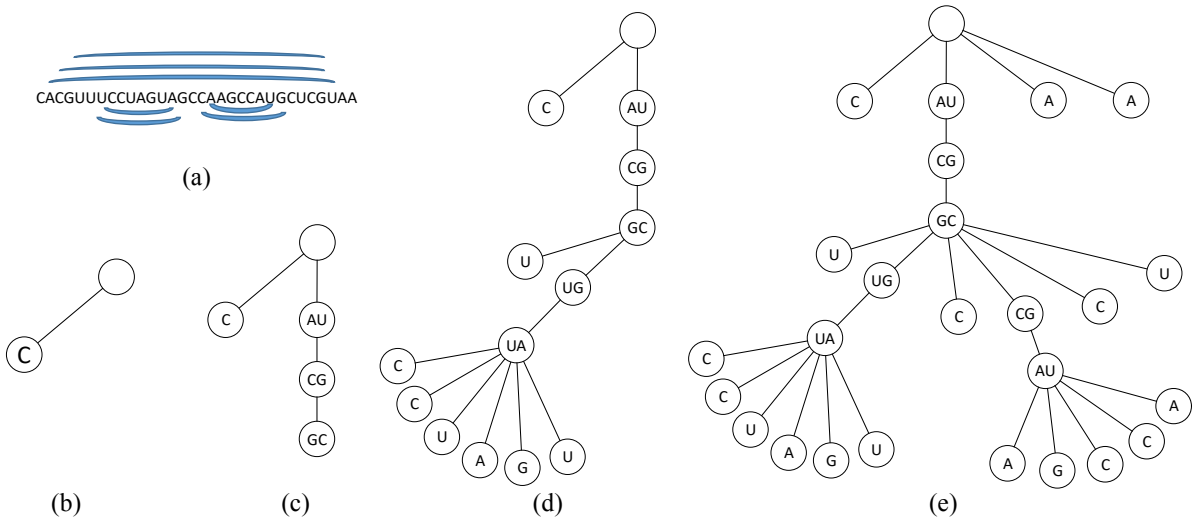


Figure 4.1. *The transformation from sequence to tree.*

When the next letter is A. As it is shown on the sequence, A is incident to the right place of an arc, here the third part of the algorithm is added. If the current letter is incident to the right branch of an arc, no vertex is added and the parent of the current letter, is the current vertex and the next vertex is added as right siblings of it [1]. So, our tree will not have any changes, and GC is the current letter according the algorithm. Other nodes are added according the algorithm and finally the tree will be built as *Figure 4.1(e)* shows.

Since we have a tree, we can define some edit operations on it and also we can mention the cost of each operation.

The *Figure 4.2(a, b)* shows the Base-Deletion and Base insertion respectively. In the first one, a node is deleted and in the second one a node is inserted.

The operation *Figure 4.2(c)* shows is the base substitution and occures when a node is changed by another one.

According to pair deletion- insertion which is shown through the *Figure 4.2(d, e)*. Arc deletion happens when an internal node is deleted and its children become the children of their parents, while arc insertion happens when an internal node is inserted as a parent of consecutive siblings.

Pair substitution is another edit operation which is shown in the *Figure 4.2(f)*. It happens when a label of an internal node is changed.

Fusion and scission are two other operations have been shown in the Figure 4.2(g, h). When an internal node is replaced by two siblings. Scission is defined when two sibling leaves are replaced while one is on the left side of the other. The siblings between these two siblings are

children for an internal node.

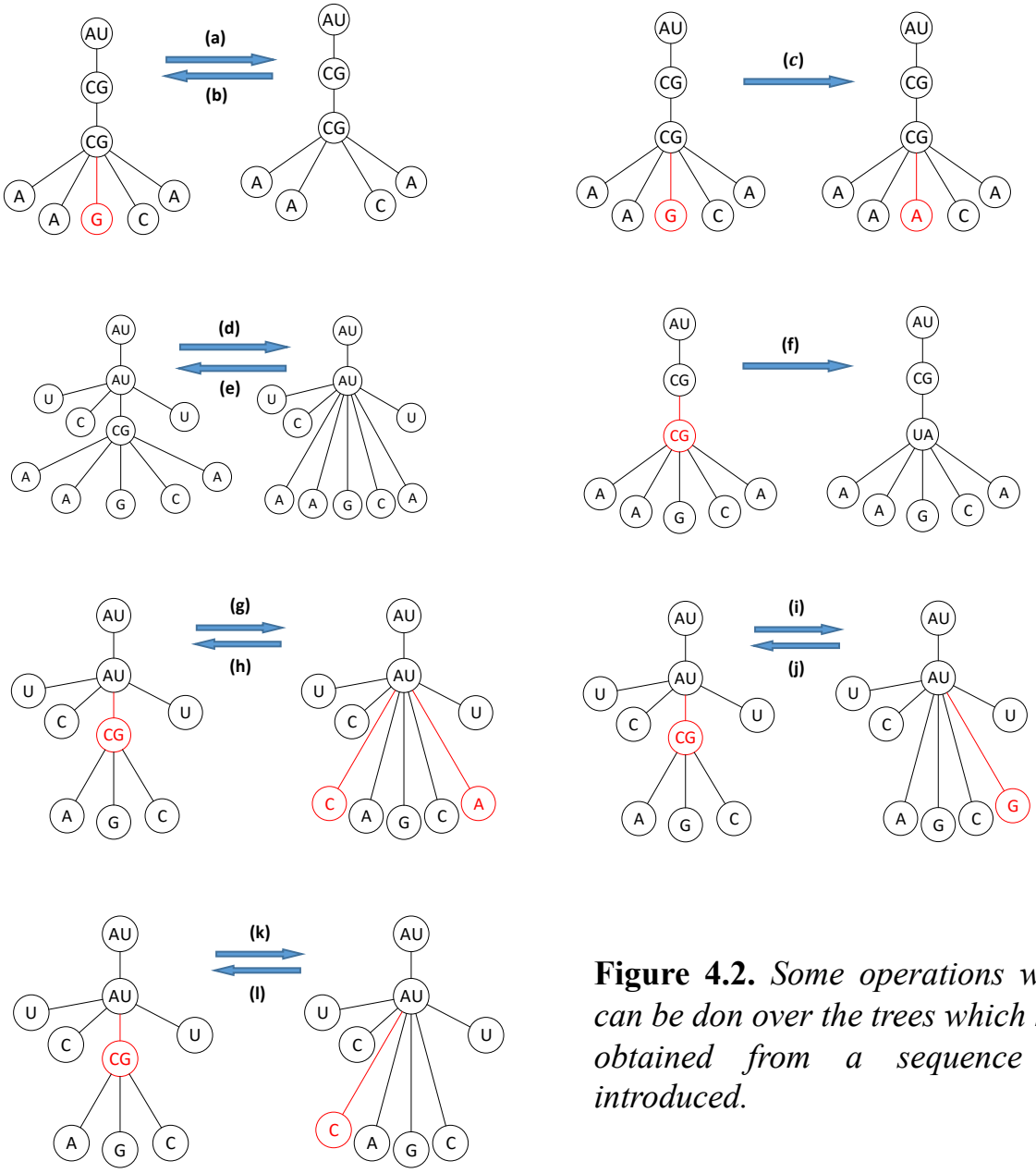


Figure 4.2. *Some operations which can be don over the trees which have obtained from a sequence are introduced.*

Left alteration and left completion are illustrated in *Figure 4.2(i, j)* means when a node is replaced by a leaf. If the letter removed is on the left side, the children of internal node are left sibling of the leaf.it is called left alteration. The second is introduced by replacing a leaf by an internal node. If the inserted letter is the left one, then left sibling of the leaf is children of internal node [1]. Right alteration and right completion are opposite of the left ones and are shown in *Figure 4.2(k, l)*.

In this part, we mention *Theorem 4.3* to show the time complexity for the alignment problem between two RNA Secondary Structure.

Recurrence relation is also true for RNA secondary structure and we can compute the time complexity.

Theorem 4.3. *The average time complexity of the secondary structure alignment algorithm for two secondary structures* S_1 *and* S_2 *is in* $O\ |S_1||S_2|$ *where* $|S|$ *stands for the length of the RNA sequence*S.

Proof: we show that we can build a tree from a sequence of letters. The number of nodes (vertices) in tree is equal to the number of letters in RNA. So, if the size of RNA is n, there are $\frac{n}{2}$ pairs.

As in the tree T, we can have either pair-base and non-pair base nodes from the sequence, so there is $\frac{n}{2}$ and n number of nodes at least and at last in order. Finally, we can conclude that the number of vertices in a tree equals to the size of a sequence.

5. An application of tree alignment algorithm

Nowadays, according to the vast demand of researches, authors have focused on the field of knowledge discovery in databases to give the highest valued data to be extracted from the data bases [27, 28]. As the huge amount of contents on the internet are semi structures, they are unusable for many algorithms. IE is a field of study which tries to solve this problem by changing semi-structured text to highly-structured databases [29 -35].

One of the existent problems during the search on the internet is the high degree of repetition among the extracted data. The proposed algorithm has addressed this issue by identifying the repetitive elements trough the automatic parameterization tree alignment. As we know, a tree alignment assigns a cost of pairing vertices from two tress [36]. When the minimum cost of vertices is found, the textual and structural elements of the tree are identified.

Since by having good parameters, the model is resilient to changes of returned dynamic HTML from the websites, it is necessary to use a good approach to achieve this goal. In order to assign the parameters to the tree, the machine learning technic is used. A support vector machine is used for these parameters. In addition, this proposed approach is useful for generating and representing schema. In the past, the authors used a set of rules for processing unseen data, while we can have a comparison between new data and the one has already been seen.

4.1 Cost function

In this method, we use HTML parse tree as trees (for edition and alignment). The labels of the tree are tag identifier or text. The trees are labeled, ordered and rooted. In this approach, the alignment is defined as a list of pairs of nodes, form each tree, each of them is paired at most. In the bellow figure is an example of two parse HTML tree.

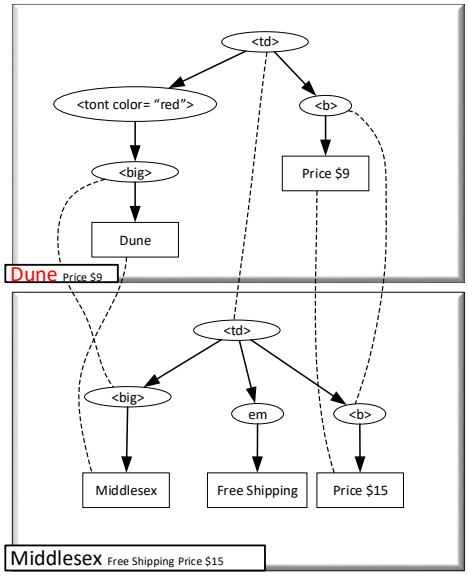


Figure 5.1. *An example of an alignment between two HTML parse trees. The rendered HTML text is illustrated at the bottom left of each box. Vertices labeled with HTML tags are shown as ellipses and textual vertices are shown as squares [36].*

The cost function of the parse tree alignment (like normal tree) consists of the summation of the cost function for relabeling, insertion and deletion:

$$\sum_{(a,b)\in A} C_\theta\big(R(a,b)\big) + \sum_{\substack{a\in T_1 \\ \forall b\in T_2 \\ (a,b)\notin \mathrm{A}}} C_\theta\big(D(a)\big) + \sum_{\substack{b\in T_2 \\ \forall a\in T_1 \\ (a,b)\notin A}} C_\theta(I(a))$$

Cost function is one of the most important parts of information extraction, sometimes, the returned data contains different string with same semantic. Prices and dates are the examples. To address this issue and the vertices are aligned well, the cost function must assign the lowest cost for the alignment of strings including similar content. For example, although the text "Price 4.99 $" and "100 $" have different synthetic, both have similar cost function. To remove such probes, three cost functions are considered with different degree of specificity.

1. Simple cost function :

   There are only three cost functions to parameterize, θ_M to copy any vertex, $\theta_{I/D}$ for insertion and deletion and θ_R to relabel the nodes. Since in this cost function, the semantics of the labels are totally ignored, the alignment of "$ 100" with "$5" have the same cost as "$ 100" with "July 24th".

2. Sed cost function:

It is similar to simple cost function with the difference about aligning textual vertices. It also has one extra parameter which is θ_S. To align two text labeled vertices, we need to multiplyθ_S times the normalized edit distance of two strings.

3. Semantic cost function:

   The third cost function uses the composition of some simple rules to determine the relationship of semantic between two text strings. The cost function to align two vertices with strings S_1 and S_2 is: $\theta_{s_1}.f(S_1, S_2)$. The $f(.,.)$ is a feature vector. And θ_{s_1} is the weights for each vector. Feature vector shows if both strings represent for example a street or not.

4.2 Learning operation costs with SVM structures

In this part we want to use a learning algorithm to train the parameters for unlabeled data. This approach has been done before by [37]. We assume that we have a set of HTML parse trees like T, each of these trees are labeled with a site (like google, amazon...). We also assume that each parse tree equals to one data record from the result of the page. T_s Denotes all the trees from site S.

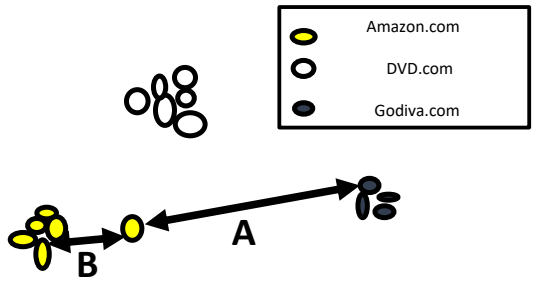


Figure 5.2. *The points represent records from three different sites. The task is to find operation costs such that preserves this grouping under tree-edit distance. A represents inter-site distance and B represents site width [36].*

There is a limitation we have imposed on the algorithm, regarding with -tree edit distance- that two trees from one site, must be closer to another than a tree from other sites. We want to find a θ which is the cost function of the parameters in such way $\forall s_1, s_2 \in S$, $\forall T_1, T_1' \in T_{s1}, \forall T_2 \in T_{s2}$: $d_\theta(T_1, T_1') \leq d_\theta(T_1, T_2)$. Sometimes it is difficult to find a θ which satisfies the limitation, to address this problem, we introduce slack variable which is denoted by ξ_s for s ∈ S penalizes the solution by the sum of the slack variable, the optimization problem is:

Min $\sum_{s \in S} \xi_s$ $\forall s_1, s_2 \in S$,$\forall T_1, T_1' \in T_{s1}, \forall T_2 \in T_{s2}$

$d_\theta(T_1, T_1')$- $d_\theta(T_1, T_2) \leq \xi_{s1}$

The width of s is the maximum distance between any two trees and the minimum distance between any two trees in tow sites S_1 and S_2 is called inter-site distance. But there is a problem,

the optimization problem is very difficult to be solved because the distance between two trees is a function of hidden variable. To solve this issue, we use EM-algorithm [38]:

Algorithm 5.1. EM-algorithm.

Input: $T_1$and T_2

Set θ_0 *to be all zeros and* $t = 0$

Do

Expectation-like step: find the minimum cost alignment, $A_{\theta_t}(.,.)$ between pairs of trees

Maximization –like step: solve the above optimization problem but replace the constraint with $C_{\theta_t}\left(A_{\theta_t}(T_1, T'_1)\right) - C_{\theta_t}\left(A_{\theta_t}(T_1, T_2)\right) \leq \xi_{s1} - 1$

Store the result as θ_{t+1}

$t \leftarrow t + 1$

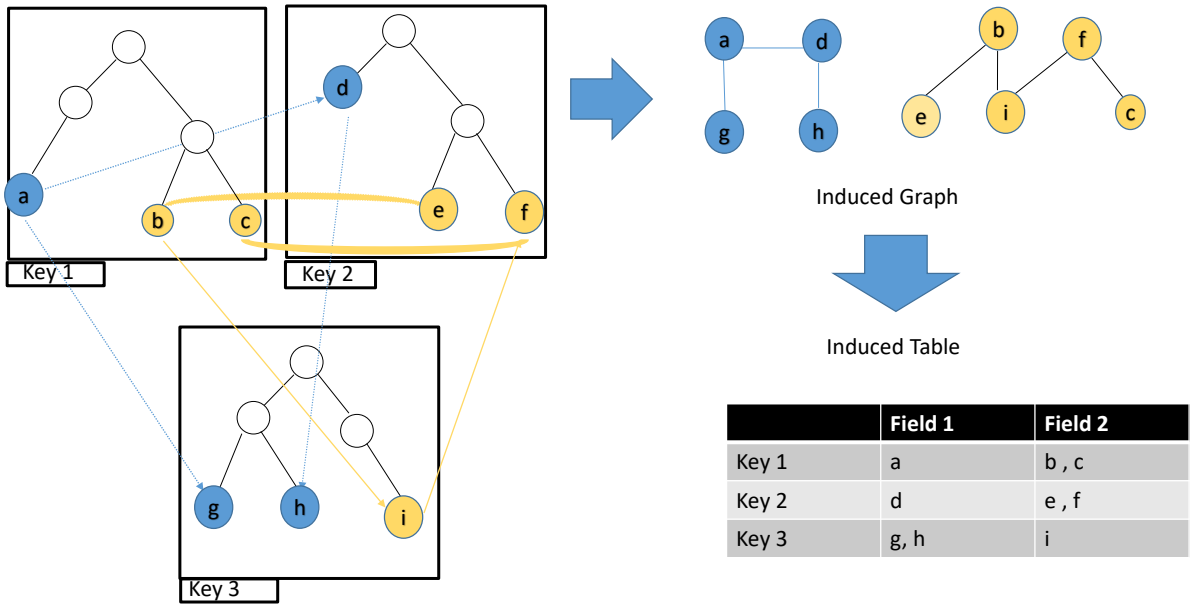
While $(\theta_t \neq \theta_{t-1})$

First step is related to initialization, at first we consider that the cost function is zero, in the expectation part of the EM- algorithm, the minimum cost function for alignment two trees is found and in maximization part, the limitation we imposed on the algorithm is replaced by $C_{\theta_t}\left(A_{\theta_t}(T_1, T'_1)\right) - C_{\theta_t}\left(A_{\theta_t}(T_1, T_2)\right) \leq \xi_{s1} - 1$. The result is cost function which is stored, and the algorithm continues until the current cost function is not equal to the previous one. There is another difference between this approach and the other approaches, the construction of the rules to extract fields [39, 40] is not explicit. We stored a set of labeled HTML parse trees as record and incoming records are aligned against the previous ones. This approach is based on nearest neighbor classification neighborhood algorithm. This approach has 2 benefits, firs, it is very simple and second when a web site changes, it is only necessary to cull a set of new keys from websites and there is no need to label data or retrain the algorithm.

- Initialize graph $G = (V, E)$
- For every pair of keys, find the minimum cost alignment A
    - For every$(u, v) \in A$, if u and v are both text vertices add (u, v) to E

Assign a unique field label to every connected component of G containing more than one vertex. The above algorithm has two arguments for generating the scheme, the alignment model and a set of keys that here are the HTML parse trees. The initial field labeling tries to

label the keys with numeric ids. A vertex must contain the same field id as all of other vertices the vertex is aligned to.



	Field 1	Field 2
Key 1	a	b , c
Key 2	d	e , f
Key 3	g, h	i

Figure 5.3. *Key labeling algorithm.*

6. Conclusion

Tree alignment algorithm, as a representative approach to measure similarity between trees, has been proposed for more than twenty year. It has been used to solve lots of scientific problems, such as aforementioned comparison of RNA secondary structures and extraction of fields from HTML search results. Its time complexity, no matter the worst case, or average time complexity has been studied for ordered tree. While same to tree edit problem, tree alignment for unordered trees is still open problem.

Reference

1. Herrbach, Claire, Alain Denise, and Serge Dulucq. "Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm." Theoretical Computer Science 411 (2010): 2423-2432.
2. Xu, Hangjun. "An Algorithm for Comparing Similarity Between Two Trees." arXiv preprint arXiv:1508.03381 (2015).
3. Krishnamachari, Santhana, and Mohamed Abdel-Mottaleb. "Hierarchical clustering algorithm for fast image retrieval." Storage and Retrieval for Image and Video Databases VII. Vol. 3656. International Society for Optics and Photonics, 1998.

4. Zhang, Kaizhong, and Dennis Shasha. "Simple fast algorithms for the editing distance between trees and related problems." SIAM journal on computing 18.6 (1989): 1245-1262.
5. Amir, Amihood, and Dmitry Keselman. "Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms." SIAM Journal on Computing 26.6 (1997): 1656-1669.
6. Kilpeläinen, Pekka, and Heikki Mannila. "Ordered and unordered tree inclusion." SIAM Journal on Computing 24.2 (1995): 340-356.
7. Jiang, Tao, Lusheng Wang, and Kaizhong Zhang. "Alignment of trees—an alternative to tree edit." Theoretical Computer Science 143.1 (1995): 137-148.
8. Herrbach, Claire, Alain Denise, and Serge Dulucq. "Average complexity of the Jiang-Wang-Zhang pairwise tree alignment algorithm and of a RNA secondary structure alignment algorithm." Theoretical Computer Science 411 (2010): 2423-2432.
9. Jansson, Jesper, and Andrzej Lingas. "A fast algorithm for optimal alignment between similar ordered trees." Annual Symposium on Combinatorial Pattern Matching. Springer, Berlin, Heidelberg, 2001.
10. Collins, Michael. "Head-driven statistical models for natural language parsing." Computational linguistics 29.4 (2003): 589-637.
11. Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation." IEEE Transactions on pattern analysis and machine intelligence 22.8 (2000): 888-905.
12. Dulucq, Serge, and Hélene Touzet. "Decomposition algorithms for the tree edit distance problem." Journal of Discrete Algorithms 3.2-4 (2005): 448-471.
13. Klein, Philip N. "Computing the edit-distance between unrooted ordered trees." European Symposium on Algorithms. Springer, Berlin, Heidelberg, 1998.
14. Demaine, Erik D., et al. "An optimal decomposition algorithm for tree edit distance." ACM Transactions on Algorithms (TALG) 6.1 (2009): 2.
15. Kilpeläinen, Pekka, and Heikki Mannila. "Ordered and unordered tree inclusion." SIAM Journal on Computing 24.2 (1995): 340-356.
16. Khanna, Sanjeev, Rajeev Motwani, and F. Frances Yao. Approximation algorithms for the largest common subtree problem. Stanford University, Department of Computer Science, 1995.
17. Amir, Amihood, and Dmitry Keselman. "Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms." SIAM Journal on Computing 26.6 (1997): 1656-1669.

18. D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, Cambridge, 1997.
19. Dulucq, Serge, and Laurent Tichit. "RNA secondary structure comparison: exact analysis of the Zhang–Shasha tree edit algorithm." Theoretical Computer Science 306.1-3 (2003): 471-484.
20. S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy, "Proof verification and the hardness of approximation problems", Journal of the ACM, vol. 45, no. 3, pp. 501-555, 1998.
21. Papadimitriou, C. H., & Yannakakis, M. (1991). Optimization, approximation, and complexity classes. Journal of computer and system sciences, 43(3), 425-440.
22. J.C. Setubal and J. Meidanis. Introduction to Computational Molecular Biology. PWS Publishing Company, Boston, 1997.
23. F. Preparata and M.I. Shamos. Computational Geometry. Springer-Verlag, New York, 1985.
24. Blin, Guillaume, and Hélene Touzet. "How to compare arc-annotated sequences: The alignment hierarchy." International Symposium on String Processing and Information Retrieval. Springer, Berlin, Heidelberg, 2006.
25. Herrbach, Claire, et al. "Alignment of rna secondary structures using a full set of operations." Rapport de Recherches 1451 (2006).
26. Blin, Guillaume, et al. "Alignments of RNA structures." IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 7.2 (2010): 309-322.
27. Chen, Ming-Syan, Jiawei Han, and Philip S. Yu. "Data mining: an overview from a database perspective." IEEE Transactions on Knowledge and data Engineering 8.6 (1996): 866-883.
28. Fayyad, Usama, Gregory Piatetsky-Shapiro, and Padhraic Smyth. "The KDD process for extracting useful knowledge from volumes of data." Communications of the ACM 39.11 (1996): 27-34.
29. Zhai, Yanhong, and Bing Liu. "Web data extraction based on partial tree alignment." Proceedings of the 14th international conference on World Wide Web. ACM, 2005.
30. Arasu, Arvind, and Hector Garcia-Molina. "Extracting structured data from web pages." Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 2003.

31. Kushmerick, Nicholas. "Wrapper induction: Efficiency and expressiveness." Artificial intelligence 118.1-2 (2000): 15-68.
32. Arlotta, Luigi, et al. "Automatic annotation of data extracted from large Web sites." WebDB. 2003.
33. Zhu, Jun, et al. "2d conditional random fields for web information extraction." Proceedings of the 22nd international conference on Machine learning. ACM, 2005.
34. Chang, Chia-Hui, and Shao-Chen Lui. "IEPAD: information extraction based on pattern discovery." Proceedings of the 10th international conference on World Wide Web. ACM, 2001.
35. Baumgartner, Robert, Sergio Flesca, and Georg Gottlob. "Visual web information extraction with lixto." In The VLDB Journal (2001).
36. Zigoris, Philip, Damian Eads, and Yi Zhang. "Unsupervised learning of tree alignment models for information extraction." Data Mining Workshops, 2006. ICDM Workshops 2006. Sixth IEEE International Conference on. IEEE, 2006.
37. sochantaridis, Ioannis, et al. "Support vector machine learning for interdependent and structured output spaces." Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.
38. https://en.wikipedia.org/wiki/Expectation–maximization_algorithm
39. Muslea, Ion, Steven Minton, and Craig A. Knoblock. "Hierarchical wrapper induction for semistructured information sources." Autonomous Agents and Multi-Agent Systems 4.1-2 (2001): 93-114.
40. Reis, Davi de Castro, et al. "Automatic web news extraction using tree edit distance." Proceedings of the 13th international conference on World Wide Web. ACM, 2004.