Approximate Labelled Subtree Homeomorphism

Ron Y. Pinter* Oleg Rokhlenko* Dekel Tsur†
Michal Ziv-Ukelson*

Abstract

Given two undirected trees T and P, the *Subtree Homeomorphism Problem* is to find whether T has a subtree t that can be transformed into P by removing entire subtrees, as well as repeatedly removing a degree-2 node and adding the edge joining its two neighbors. In this paper we extend the Subtree Homeomorphism Problem to a new optimization problem by enriching the subtree-comparison with node-to-node similarity scores. The new problem, called *Approximate Labelled Subtree Homeomorphism (ALSH)*, is to compute the homeomorphic subtree of T which also maximizes the overall node-to-node resemblance. We describe an $O(m^2 n/\log m + mn \log n)$ algorithm for solving ALSH on unordered, unrooted trees, where m and n are the number of vertices in P and T, respectively. We also give an $O(mn)$ algorithm for rooted ordered trees and $O(mn \log m)$ and $O(mn)$ algorithms for unrooted cyclically ordered and unrooted linearly ordered trees, respectively.

Keywords: tree similarity, approximate labelled matching.

1 Introduction

The matching of labelled tree patterns has many important applications in areas such as bioinformatics, semistructured databases, and linguistics. Examples include the comparison among metabolic pathways, the study of alternative evolutionary trees (phylogenies), processing queries against databases and documents represented in e.g. XML, and many fundamental operations in the analysis of natural (and formal) languages. In all these scenarios, both the labels on the nodes as well as the structure of the underlying trees play a major role in determining the similarity between a pattern and the text in which it is to be found.

There are several, increasingly complex ways to model these kinds of problems. A starting point is the *subtree isomorphism problem* [17, 18, 25]: Given a pattern tree P and a text

*Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel. {pinter, olegro, michalz}@cs.technion.ac.il

†Dept. of Computer Science and Engineering, University of California, San Diego. dtsur@cs.ucsd.edu

tree T, find a subtree of T which is isomorphic to P (i.e. find whether a subtree of T that is identical in structure to P can be obtained by removing entire subtrees of T) or decide that there is no such tree. The *subtree homeomorphism problem* [3, 22] is a variant of the former problem, where degree 2 nodes can be deleted from the text tree (see Figure 1). The *constrained tree inclusion* problem [28] is a variant of labelled subtree homeomorphism where label equality between pairs of aligned nodes in the compared subtrees is required. Note that all the tree matching problems mentioned so far have efficient algorithms. The *tree inclusion problem* [16] is the problem of locating the smallest subtrees of T that include P, where a tree is included in another tree if it can be obtained from the latter by deleting nodes. Here, deleting a node v from a tree entails deleting all edges incident on v and inserting edges connecting the parent of v with the children of v. *Tree inclusion* on unordered trees is NP-complete [16]. Schlieder and Naumann [23] extended the tree inclusion problem to an *approximate tree embedding problem* in order to retrieve and rank search results using a tree-similarity measure whose semantics are tailored to XML data. The complexity of their algorithm, which is based in dynamic programming and processes the query tree in a bottom up fashion, is exponential.

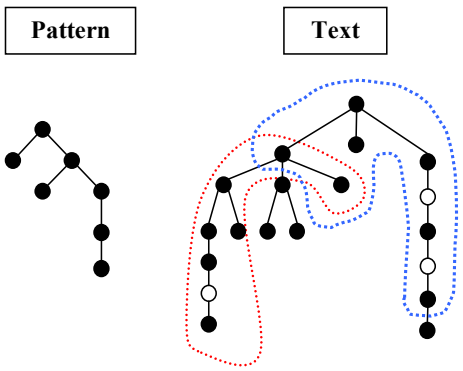

Figure 1: Example for Subtree homeomorphism. Two subtrees in the text tree that are homeomorphic to the pattern tree are circled by the dashed lines. The white nodes in the text tree are degree-2 nodes that are removed by the homeomorphism.

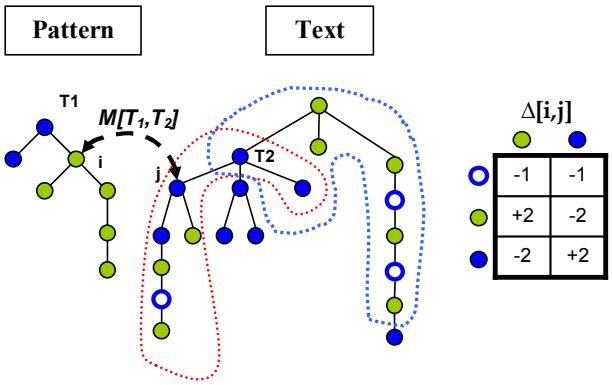

Figure 2: Approximate Labelled Subtree Homeomorphism. The node-label similarity scores are specified in table Δ. Note that $\delta = -1$. Two subtrees in the text that are homeomorphic to the pattern are circled by the dashed lines. The left homeomorphic subtree has an LSH score of 5, while the right one has an LSH score of 12.

This paper addresses queries on labelled trees. The trees can be either ordered or unordered, rooted or unrooted — depending on the application at hand. The main point is, however, that even though the trees are labelled, label equality is not required in a match. Instead, a node-to-node similarity measure is used to rank the resemblance or distance between pairs of aligned nodes. This extension is motivated by the aforementioned applications, where a more meaningful match can be found if — in addition to the topological structure similarity between subtrees — node-to-node resemblance is also taken into account. For example, in

bioinformatics, the similarity between metabolic pathways [19, 24] is based both on the resemblance of the elements constituting the pathways (e.g. proteins) as well as on the likeness of their network structure (the former would reflect closeness between proteins, based on either a BLAST sequence comparison score or on functional homology, and the latter would be the topological equivalent of multi-source unrooted trees [21]). A query that searches for a small pathway fragment in a larger tree should take into account the topology similarities (in the form of subtree homeomorphism) as well as the pairwise similarity between proteins which make up the aligned subtree nodes. Another example is when designing a semantic query language for semistructured databases that are represented as XML documents [23]: here the node-to-node similarity score may reflect content or tag resemblance, and the topological difference allows flexibility in document structure matching [2]. Finally, in natural language processing, trees are often used to represent sentences, where nodes are labelled by words and sentential forms (or constituents); matching that takes into account synonyms and elisions is very useful in order to detect semantically close phrases.

Thus, this paper addresses the challenge of extending labelled subtree homeomorphism into a new optimization problem, by introducing node-to-node similarity measures that are combined with the topological distance between the pattern and text to produce a single, comprehensive score expressing how close they are to each other. More precisely, let Δ be a node-to-node similarity score table and δ denote a predefined (usually negative) score for deleting a node from a tree (Figure 2). An *homeomorphism preserving mapping* $\mathcal{M}$ from a tree T_1 to a tree T_2 is a one-to-one map from the nodes of T_1 to the nodes of T_2 that preserves the ancestor relations of the nodes. We define the following similarity measure for two homeomorphic trees.

Definition 1. *Let T_1 and T_2 be two labelled trees such that T_2 is homeomorphic to T_1, and let $\mathcal{M}$ be a homeomorphism preserving mapping from T_1 to T_2. The* ***Labelled Subtree Homeomorphism (LSH) Similarity Score*** *of $\mathcal{M}$, denoted* $\mathrm{LSH}(\mathcal{M})$*, is*

$$\mathrm{LSH}(\mathcal{M}) = \delta \cdot (|T_2| - |T_1|) + \sum_u \Delta[u, \mathcal{M}(u)].$$

Definition 2. *The* ***Approximate Labelled Subtree Homeomorphism (ALSH) problem*** *is, given two undirected labelled trees P and T, and a scoring table which specifies the similarity scores between the label of any node appearing in T and the label of any node appearing in P, as well as a predefined node deletion penalty, to find a homeomorphism-preserving mapping $\mathcal{M}$ from P to some subtree t of T, such that* $\mathrm{LSH}(\mathcal{M})$ *is maximal.*

In this paper we show how to compute optimal, bottom-up alignments between P and any homeomorphic subtree t of T, which maximize the LSH score between P and t. Our algorithms are based on the close relationship between subtree homeomorphism and maximum weight matchings in bipartite graphs. The ALSH problem is recursively translated into a collection of smaller ALSH problems, which are solved using maximum weight matching algorithms (Figure 3). (Simpler, maximum matching algorithms were applied in the algorithms for exact subtree morphisms [3, 10, 15, 25, 28].)

Our approach yields an $O(m^2 n/\log m + mn \log n)$ algorithm for solving ALSH on unordered, unrooted trees, where m and n are the number of vertices in P and T, respectively. Note that the time complexity of the exact subtree isomorphism/homeomorphism algorithms [25, 28], which do not take into account the node-to-node scores, is $O(m^{1.5} n/\log m)$. Thus, the enrichment of the model with the node similarity information only increases the time complexity by half an order. We also give an $O(mn)$ algorithm for the problem on rooted ordered trees and $O(mn \log m)$ and $O(mn)$ algorithms for unrooted cyclically ordered and unrooted linearly ordered trees respectively.

Also note that a related set of problems, where dynamic programming is combined with weighted bipartite matching, is that of finding the maximum agreement subtree and the maximum refinement subtree of a set of trees [14, 26]. Such algorithms utilize the special constraints imposed by the properties of evolutionary trees (internal nodes contain less information than leaves, similarity assumption allows for scaling, etc).

The rest of the paper is organized as follows. Section 2 includes weighted bipartite matching preliminaries. In Section 3 we describe the basic $O(m^2 n + mn \log n)$ time ALSH algorithm for rooted unordered trees and show how to extend it to unrooted unordered trees without increasing the time complexity. These solutions are further improved in Section 4 to yield an $O(m^2 n/\log m + mn \log n)$ solution for both rooted and unrooted unordered trees. In Section 5 we describe the $O(mn)$ time ALSH algorithm for rooted ordered trees,$O(mn \log m)$ time ALSH algorithm or unrooted cyclically ordered trees and $O(mn)$ time ALSH algorithms for unrooted linearly ordered trees.

2 Weighted Matchings and Min-Cost Max Flows

Definition 3. *Let $G = (V, E)$ be a bipartite graph with edge weights $w(x, y)$ for all edges $[x, y] \in E$. The **Maximum Weight Matching Problem** (also called the assignment problem) is to compute a matching M that maximizes $\sum_{[x,y]\in M} w(x, y)$.*

Many researchers have developed several different algorithms for the assignment problem [1], and some of these algorithms share common features. The successive shortest path algorithm for the minimum cost max flow problem appears to lie at the heart of many assignment algorithms [13]. The reduction from the assignment problem to the minimum cost max flow problem is as follows. Let s, t be two new vertices. Construct a graph G' with vertex set $V \cup \{s, t\}$, source s, sink t, and capacity-one edges: an edge $[s, x]$ of cost zero for every $x \in X$, an edge $[y, t]$ of cost zero for every $y \in Y$, and and edge $[x, y]$ of cost $-w(x, y)$ for every $[x, y] \in E$. An integral flow f on G' defines a matching on G of size $|f|$ and weight $-\text{cost}(f)$ given by the set of edges $[x, y]$ such that $[x, y]$ has one unit of flow. Conversely, a matching M on G defines a flow of value $|M|$ and cost $\sum_{[x,y]\in M} -w(x, y)$. This means that one can solve a matching problem on G by solving a flow problem on G'.

Edmonds and Karp's algorithm [6] finds a minimum cost maximum flow in G' in $O(EV \log V)$ time. In each stage of this algorithm, Dijkstra's algorithm [5] is used to compute an aug-

mentation path for the existing flow f. Each run of Dijkstra's algorithm is guaranteed to increase the size of M by 1, and thus the algorithm requires at total of $O(V)$ phases to find a maximum score matching. Fredman and Tarjan [8] developed a new heap implementation, called *Fibonacci heap*, that improves the running time of Edmond and Karp's algorithm to $O(VE + V^2 \log V)$. The latter bound is the best available strongly polynomial time bound for solving the assignment problem.

Under the assumption that the input costs are integers in the range $[-C, \ldots, C]$, Gabow and Tarjan [9] use cost scaling and blocking flow techniques to obtain an $O(V^{1/2}E \log(VC))$ time algorithm for the assignment problem. Two-phase algorithms with the same running time appeared in [11, 20]. The scaling algorithms are conditioned by the *similarity assumption* (i.e. the assumption that $C = O(n^k)$ for some constant k).

The following lemma, which was first stated and proven in [13], will serve as the basis for the proofs of Lemma 2 and Lemma 4.

Lemma 1. *([6], [13],[27]) A flow f is minimum cost **iff** its residual graph R has no negative cost cycle.*

3 Dynamic Programming Algorithms for ALSH

3.1 The Basic Algorithm for Rooted Unordered Trees

Let $T^r = (V_T, E_T, r)$ be the text tree which is rooted at r, and $P^{r'} = (V_P, E_P, r')$ be the pattern tree which is rooted at r', respectively. For a node v of T^t, let t_v^r denote the subtree of T^r that contains v and all its descendents, and whose root is v. Similarly, we use $p_u^{r'}$ to denote a subtree of $P^{r'}$.

Definition 4. *For each node $v \in V_T$ and for each node $u \in V_P$,* RScores$[v, u]$ *is the maximum LSH similarity score between $p_u^{r'}$ and some homeomorphic subtree of t_v^r, if such subtrees exist. Otherwise,* RScores$[v, u]$ *is $-\infty$.*

The computation of RScores$[v, u]$ is done recursively, in a postorder traversal of T^r. First, RScores$[v, u]$ is computed for every leaf nodes of T^r and $P^{r'}$. Next, RScores$[v, u]$ is computed for each node $v \in V_T$ and $u \in V_P$, based on the values of the previously computed scores for the children of v and u as follows. Let u be a node of $P^{r'}$ with children $x_1, \ldots, x_{c(u)}$ and v be a node of T^r with children $y_1, \ldots, y_{c(v)}$. After computing RScores$[y_j, x_i]$ for $i = 1, \ldots, c(u)$ and $j = 1, \ldots, c(v)$, a bipartite graph G is constructed with bipartition X and Y, where X is the set of children of u, Y is the set of children of v, and each node in X is connected to each node in Y. An Edge (x_i, y_j) of G is annotated with weight RScores$[y_j, x_i]$ (Figure 3).

RScores$[v, u]$ is then computed as the maximum between the following two terms:

1. The node-to-node similarity value $\Delta[v, u]$, plus the maximum weight of a matching in G. Note that this term is only computed if $c(u) \leq c(v)$.

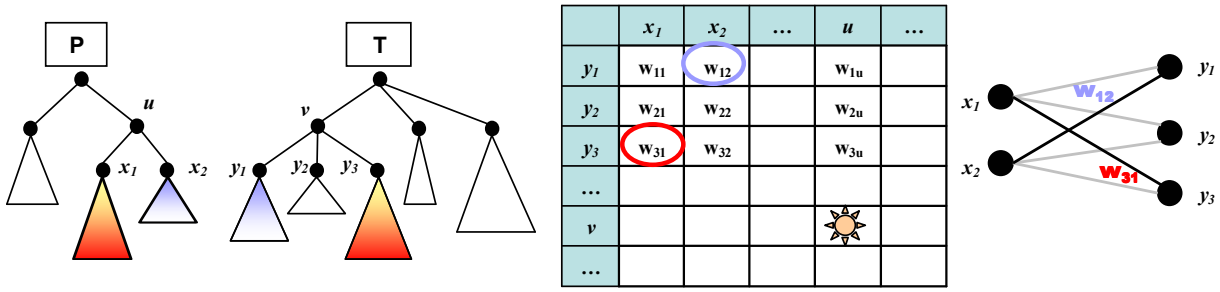

Figure 3: The bipartite graph that is built by Algorithm 1 in order to compute RScores$[v, u]$. w_{ij} marks the optimal LSH similarity score for aligning the subtrees rooted at $y_j \in T$ and $x_i \in P$.

2. The maximum value among RScores$[y_1, u], \ldots,$ RScores$[y_{c(v)}, u]$, plus δ (the penalty for deleting v).

The optimal LSH similarity score, denoted *best_score*, is *best_score* $= \max_{j=1}^{n}$ RScores$[y_j, x_m]$. Every node $y_j \in T$ with RScores$[y_j, x_m]$ = *best_score* is reported as a root of a subtree of T which bears maximal similarity to P under the LSH measure.

An outline of the algorithm described above, denoted Algorithm 1, is given below.

Time Complexity Analysis

Theorem 1. *Algorithm 1 computes the optimal ALSH solution for two rooted unordered trees in $O(m^2n + mn \log n)$ time. Moreover, under the similarity assumption, the algorithm has an $O(m^{1.5}n \log(nC))$ time complexity.*

Proof. The time complexity analysis of the algorithm is based in the following observation.

Observation 1. $\sum_{u=1}^{m} c(u) = m - 1$ *and* $\sum_{v=1}^{n} c(v) = n - 1$.

Algorithm 1 calls procedure ComputeScoresForTextNode once for each node pair ($v \in T, u \in P$). The dominant part of this procedure's work is spent in computing a maximum weight matching in a bipartite graph with $c(v) + c(u)$ nodes and $c(v) \cdot c(u)$ edges. Therefore, using Fredman and Tarjan's algorithm [8], each maximum weight matching computation takes $O(c(u)^2 \cdot c(v) + c(u) \cdot c(v) \log c(v))$ time.

Summing up the work over all node pairs we get

$$O(\sum_{u=1}^{m}\sum_{v=1}^{n}(c(u)^2c(v)+c(u)c(v)\log c(v)) \overset{\text{obs. 1}}{=} O(\sum_{u=1}^{m} c(u)^2n+c(u)n\log n) \overset{\text{obs. 1}}{=} O(m^2n+mn\log n).$$

Under the similarity assumption (namely, all scores assigned to the edges of G are integers in the range $[-C, \ldots, C]$, where $C = O(n^k)$ for some constant k), the algorithm can be modified to run in time $O(m^{1.5}n \log(nC))$ by employing the algorithm of Gabow and Tarjan [9] for weighted bipartite matching. □

Algorithm 1 ALSH for Rooted Unordered Trees

Input: Rooted trees $T = (V_T, E_T, r)$ and $P = (V_P, E_P, r')$.

Output: The root of the subtree t of T that has the highest similarity score to P, if T has a subtree which is homeomorphic to P.

for each node u of P in **postorder do**
  for each node v of T in **postorder do**
    if u is leaf **then**
      if v is leaf **then**
        RScores$(v, u) \leftarrow \Delta[v, u]$
      else
        RScores$(v, u) \leftarrow$ ComputeScoresForTextNode(v, u)
      end if
    else
      if Level$(u) >$ Level(v) **then**
        RScores$(v, u) \leftarrow -\infty$
      else
        RScores$(v, u) \leftarrow$ ComputeScoresForTextNode(v, u)
      end if
    end if
  end for
end for

Procedure ComputeScoresForTextNode(v, u)

Let k denote the out-degree of node u and ℓ denote the out-degree of node v.
if $k > \ell$ **then**
  AssignmentScore $\leftarrow -\infty$
else
  Construct a bipartite graph G with node bipartition X and Y such that $X = \{x_1, \ldots, x_k\}$ is the set of children of u, $Y = \{y_1, \ldots, y_\ell\}$ is the set of children of v, and every node $u_i \in X$ is connected to every node $v_j \in Y$ via an edge whose weight is RScores(v_j, u_i).
  Set AssignmentScore to the maximum weight of a matching in G.
end if
BestChild $\leftarrow \max_{j=1}^{\ell}$ RScores(y_j, u)
return $\max\{\Delta[v, u] +$ AssignmentScore, BestChild $+ \delta\}$

3.2 Extending the Algorithm to Unrooted Unordered Trees

Let $T = (V_T, E_T)$ and $P = (V_P, E_P)$ be two unrooted trees. The ALSH problem on P and T can be solved in a naive manner as follows. Select an arbitrary node r of T to get the rooted tree T^r. Next, for each $u \in P$ solve the rooted ALSH problem on P^u and T^r. This method yields an $O(m^3n + m^2n \log n)$ strongly-polynomial algorithm for ALSH on unrooted unordered trees, and an $O(m^{2.5}n \log(nC))$ time algorithm under the similarity assumption with integrality cost function restriction. In this section we show how to reduce these bounds back to $O(m^2n + mn \log n)$, and $O(m^{1.5}n \log(nC))$ time, respectively, by utilizing the decremental properties of the maximum weight matching algorithm.

The algorithm starts by selecting a vertex r of T to be the designated root. T^r is then traversed in a postorder, and each internal vertex $v \in T^r$ is compared with each vertex $u \in P$. Let $y_1, \ldots, y_{c(v)}$ be the children of $v \in T^r$, and let $x_1, \ldots, x_{d(u)}$ be the neighbors of $u \in P$. When computing the score for the comparison of u and v we need to take into account the fact that, since P is unrooted, each of the neighbors of u may serve as a parent of u in a mapping of P with some subtree t^r_v. Suppose node $x_i \in P$, which is a neighbor of $u \in P$, serves as the parent of u in one of these subtree alignments. Since x_i is the chosen parent, the children set of u includes all its neighbors but x_i. Therefore, the computation of the similarity score for v versus u requires computing the maximum weight matching between the children of v and all neighbors of u except x_i. Since each neighbor x_i of u is a potential parent of a subtree rooted in u, our algorithm will actually need to compute maximum weight matching in a series of graphs $G_i = (X_i \cup Y, E)$ for $i = 1, \ldots, d(u)$, where Y consist of all the children of v in T^r, while X_i consists of all the neighbors of u except x_i. Therefore, we define the table UScores as follows.

Definition 5. *For every vertex $v \in T^r$, every vertex $u \in P$. and every vertex $x_i \in$* neighbors(u)*,* UScores$[v, u, x_i]$ *is the maximum LSH similarity score between $p^{x_i}_u$ and some homeomorphic subtree of t^r_v, if such subtrees exist. Otherwise,* UScores$[v, u, x_i]$ *is $-\infty$. Moreover,* UScores$[v, u, \phi]$ *is the maximum LSH similarity score between P^u and some homeomorphic subtree of t^r_v, if one exists.*

The computation of UScores$[v, u, x_i]$ is carried out as follows. UScores$[v, u, x_i]$ is set to the maximum between the following two terms:

1. The node-to-node similarity value $\Delta[v, u]$, plus the maximum weight of a matching in G_i. Note that this term is only computed if $d(u) - 1 \leq c(v)$.
2. The maximum value among UScores$[y_1, u, x_i], \ldots,$ UScores$[y_{c(v)}, u, x_i]$ plus δ.

The maximal LSH similarity score between P and T is *best_score* $= \max_{v \in V_T, u \in V_P}$ UScores$[v, u, \phi]$. An outline of the second algorithm, denoted Algorithm 2, is given below.

Algorithm 2 ALSH for Unrooted Unordered Trees

Input: Unrooted trees $T = (V_T, E_T)$ and $P = (V_P, E_P)$).

Output: The root of the subtree t of T which has the highest similarity score to P, if T has a subtree which is homeomorphic to P.

Pick a vertex r of T to be the root of T.
for all $u \in P$, $v \in T$ $x_i \in P$ **do**
  UScores$[v, u, x_i] \leftarrow -\infty$.
end for
for each leaf v of T_r **do**
  for each leaf u of P **do**
    UScores$[v, u, \text{parent}(u)] \leftarrow \Delta[v, u]$.
  end for
end for
for each internal node v of T in postorder **do**
  ComputeScoresForTextNode(v)
end for
best_score $\leftarrow \max_{i=1,\ldots,m,j=1,\ldots,n}$ UScores$[j, i, \phi]$

Procedure ComputeScoresForTextNode(v)

for each node u of P **do**
  BestChild$(v, u, x_i) \leftarrow \max_{j=1,\ldots,\ell,i=1,\ldots,k}$ UScores$[y_j, u, x_i]$
  Construct a bipartite graph G with node bipartition X and Y such that $X = \{x_1, \ldots, x_k\}$ is the set of neighbors of u, $Y = \{y_1, \ldots, y_\ell\}$ is the set of children of v, and every node $x_i \in X$ is connected to every node $y_j \in Y$ via an edge whose weight is UScores$[y_j, x_i, u]$.
  Let $X_0 = X$ and $X_i = X - \{x_i\}$.
  for all $1 \leq i \leq k$ **do**
    if Level$(v) <$ Level(u, x_i) **then**
      UScores$[u, v, x_i] \leftarrow -\infty$
    else
      if $k > \ell$ **then**
        AssignmentScore$(X_i, Y) \leftarrow -\infty$.
      else
        Compute the score AssignmentScore(X_i, Y) of the maximum weight matching in G.
      end if
      UScores$[v, u, x_i] \leftarrow \max\{\Delta[v, u]+$AssignmentScore$(X_i, Y)$, BestChild$(v, u, x_i)+\delta\}$
    end if
  end for
  if $k > l$ **then**
    UScores$[v, u, \phi] = -\infty$
  else
    Set UScores$[v, u, \phi]$ to the maximum weight matching in G.
  end if
end for

Time Complexity Analysis

The time complexity bottleneck of Algorithm 2 is in the need to compute for each pair $u \in P$ and $v \in T$, maximum weight matchings in a series of bipartite graphs $G_i = (X_i \cup Y, E)$ for $i = 1, \ldots, d(u)$. This, in contrast to Algorithm 1, where only one maximum weight matching is computed for each pair $u \in P$ and $v \in T$. However, the next lemma shows that the decremental nature of weighted bipartite matching makes it possible to compute the matchings for $G_1, \ldots, G_{d(u)}$ in the same asymptotic time complexity as the maximum weight matching computation for the single graph $G = (X \cup Y, E)$.

For the following lemma, denote $k = |X|$ and $\ell = |Y|$. Note that $k \leq \ell$.

Lemma 2. *Given a maximum weight matching M of G, a maximum weight matching of G_i can be computed via one run of a single-source shortest-path algorithm.*

Proof. The proof is based on the reduction of the assignment problem to min-cost max-flow, as described in Section 2. Let f denote the minimum cost maximum flow in the graph G' which corresponds to the matching M, and let R denote the residual graph for f (Figure 4.A). Clearly, since f is optimal, by Lemma 1, there are no negative cost cycles in R. Also, since $\ell \geq k$, $|M| = k$.

Let y_j be the vertex that is matched to x_i in M. Let $M' = M \setminus \{(x_i, y_j)\}$ denote the matching obtained by removing the pair (x_i, y_j) from M. We say that y_j is "newly widowed" in M'. Let f' be the flow defined by M'. Let G'' denote the subgraph of G' obtained by removing the vertex x_i and all the edges that are incident on x_i. Let R' be the "decremented" residual subgraph of R, obtained by removing vertex x_i and all edges incident on x_i, as well as cancelling the flow on edge $[s, y_j]$ in R. (Figure 4.B). Our objective is to compute a min-cost max-flow (of value $k - 1$) of G'', which will correspondingly define a maximum weight matching in G_i.

Clearly, the value of f' is $k - 1$. But is f' min-cost? By Lemma 1, if there are no negative cost cycles in R', then f' is a min-cost flow among all $k - 1$-valued flows in G''. Otherwise, there are one or more negative cost cycles in R'. Each one of these negative cost cycles in R' must contain the edge $[s, y_j]$, since this edge is the only edge of R' which does not exist in R, and R has no negative cost cycles.

Among all the negative cost cycles in R', we wish to find the one which contributes the most to decrementing the value of f'. More precisely, define the *correction path* for R', to be the minimum cost cycle in R' that contains the edge $[s, y_j]$ (Figure 4.C). Denote by p the correction path for R'.

We claim that:

1. If p has negative total cost, then M' is not optimal.
2. If M' is not optimal, then p is the optimal correction path for R'. That is, the flow obtained by pushing one unit of flow in R' from s through y_j and back to s through

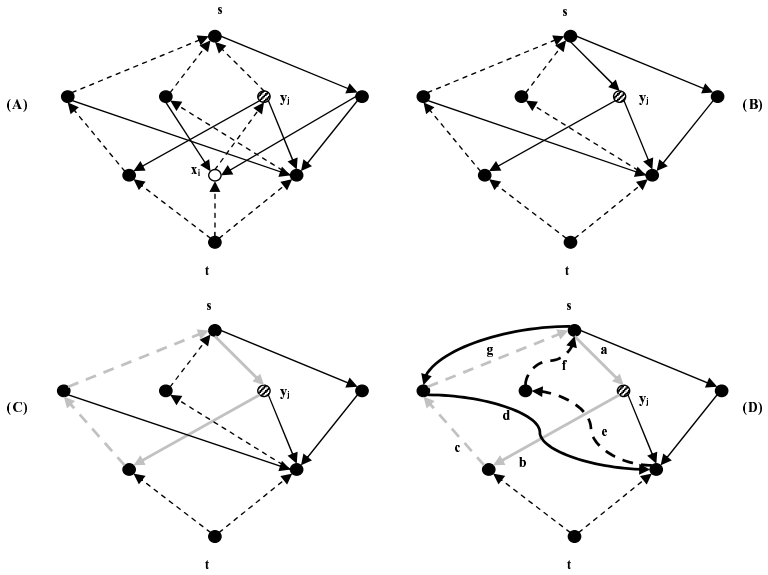

Figure 4: The computation of a maximum weight matching in G_i. Note that, for the sake of graphical clarity, some of the edges of the graph are not shown in the figure. (A) The residual graph R for flow f.The dashed lines denote the backward edges and the solid lines denote the forward edges. The white circle denotes vertex x_i and the lined circle denotes its matched neighbor y_i. (B) The residual graph R' received from R by removing x_i and its adjacent edges, as well as reversing the flow on edge $[s, y_i]$. (C) The augmentation path p in R' is marked by grey edges. (D) The negative cost cycle c is marked by curves. Edges [a,b,c,g] correspond to the correcting path p, edges [g,d,e,f] correspond to the negative cost cycle c, and edges [a,b,c,d,e,f] mark the path p' with a cost which is less then that of p.

the path p, is a minimal cost maximum flow for the network graph G'' and defines, a maximum weight matching in G_i.

The above claims are proven as follows.

1. This is immediate from Lemma 1, since p is by definition a cycle in R'.

2. Let f'' be obtained from f' by correcting along p, and let R'' denote the residual graph for flow f''. We will prove that f'' is min-cost in G'', by proving that there are no negative cost cycles in R''. Suppose conversely that there is some negative cost cycle c in R''. Since R has no negative cost cycles, and the only new edge $[s, y_j] \in R'$ has been saturated by p in R'' (therefore all previous cycles are broken in R''), we know that c has at least one edge which is the reversal of an edge in p. Therefore, the cycle c consists of edges in R' and one or more edges whose reversals are on p. In particular, p and c share at least one vertex.

   Let $p \oplus c$ be the set of edges on p or on c except for those occurring on p and reversed on c. The cost of $p \oplus c$ is $\text{cost}(p) + \text{cost}(c)$, since the $\oplus$ operation has removed pairs of edges and their reversals (the score is negated in the reversal) in $p \cup c$. Since c is a

negative cost cycle, $\text{cost}(p) + \text{cost}(c) < \text{cost}(p)$. Furthermore, $p \oplus c$ can be partitioned into a cycle p' that contains the edge $[s, y_j]$, plus a collection of additional cycles (note that none of these cycles includes the edge $[s, y_j]$, since this edge is already occupied by p'). By Lemma 1 all the cycles must have nonnegative cost since they consist of edges only from R' which has no negative cost cycles. Thus, the cost of the cycle p' is less than the cost of p (Figure 4.D). This contradiction implies the lemma. □

Lemma 3. *Computing the maximum weight matchings for the bipartite graphs $G_1, \ldots, G_k$ can be done in $O(k^2\ell + k\ell \log \ell)$ time. Moreover, under the similarity assumption, computing these matchings can be done in $O(k^{1.5}\ell \log(\ell C))$ time.*

Proof. The computation can be carried on as follows. First construct the network flow graph corresponding to G and find the minimum cost maximum flow which defines a maximum weight matching in G. This can be done in $O(VE + V^2 \log V) = O(k^2\ell + \ell^2 \log \ell)$ time using the algorithm of [8]. Then, for each x_i, $i = 1, \ldots, k$, construct the corresponding residual network graph, and compute the correction path p_i in order to obtain an maximum weight matching in G_i. By Lemma 2, the amount of work for each correction path computation is bounded by the complexity of a single-source shortest path computation on a graph with *ell* vertices and $k\ell$ edges. Using the variation of Dikjstra's algorithm described in [8], each correction path computation can be done in $O(k\ell + \ell \log \ell)$ time. Summing over the graphs $G_1, \ldots, G_k$, we get a total time of $O(k^2\ell + k\ell \log \ell)$.

The second part of the lemma follows from [14]. □

Theorem 2. *Algorithm 2 computes the optimal ALSH solution for two unrooted unordered trees in $O(m^2 n + mn \log n)$ time. Under the similarity assumption, the time complexity of the algorithm is $O(m^{1.5} n \log(nC))$.*

Proof. For the proof of the theorem, we use the following observation.

Observation 2. *The sum of vertex degrees in an unrooted tree P is $\sum_{u=1}^{m} d(u) = 2m - 2$.*

Algorithm 2 calls to procedure ComputeScoresForTextNode once for each node pair ($v \in T, u \in P$). The algorithm computes, in the loop of step 4 of procedure ComputeScoresForTextNode, maximum weight matchings in several bipartite graphs. By Lemma 3, computing these matchings takes $O(d(u)^2 c(v) + d(u)c(v) \log c(v))$ time. Therefore, the total complexity is:

$$O(\sum_{u=1}^{m}\sum_{v=1}^{n}(d(u)^2 c(v)) + d(u)c(v)\log c(v)) \overset{\text{obs. } 1}{=} O(\sum_{u=1}^{m} d(u)^2 n + d(u) n \log n) \overset{\text{obs. } 2}{=} O(m^2 n + mn \log n).$$

Under the (similarity) assumption that all scores assigned to the edges of G are integers in the range $[-C, \ldots, C]$, where $C = O(n^k)$ for some constant k, the algorithm can be modified to run in time $O(m^{1.5} n \log(nC))$ by employing the algorithm of Gabow and Tarjan [9] for weighted bipartite matching with scaling and [14] for cavity matching. □

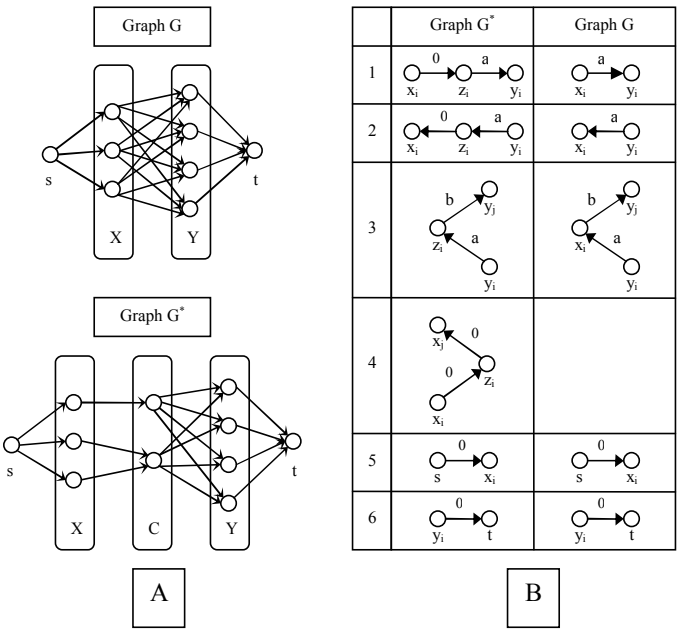

Figure 5: (A) Graph G with a bipartition of its vertices into sets X and Y, and the compressed graph G^* that contains an additional layer C. (B) The mapping $\mathcal{M}$ of subunits of sub-paths from G^* into subunits of sub-paths in G.

Note that the first term of the sum $O(m^2n + mn \log n)$ dominates the time complexity of Algorithm 2, since the second term only takes over when $m \leq \log n$. Therefore, in the next section we will show how to reduce the first term in the time complexity of Algorithm 2 by a $\log m$ factor.

4 A More Efficient ALSH Algorithm for Unordered Trees

In this section we show how the dominant term in the time complexity of the algorithm described in the previous section can be reduced by a $\log m$ factor, assuming a constant-sized label alphabet. We will use the previous algorithm but solve the matching problems more efficiently by employing the notion of clique partition of a bipartite graph [7, 25]. The modified algorithm, denoted Algorithm 3, is the same as Algorithm 2 with the exception that, in step 12 we solve the assignment problems differently. Let v be some vertex in T^r whose children are $y_1, \ldots, y_{c(v)}$ and let u be a vertex in P whose neighbors are $x_1, \ldots, x_{d(u)}$. Denote $X = \{x_1, \ldots, x_{d(u)}\}$ and $Y = \{y_1, \ldots, y_{c(v)}\}$. Let G be the corresponding complete bipartite graph on $X \cup Y$. We compute a maximum weight matching in G as follows. Let $N(x_i)$ be the sequence of the weights of the edges $[x_i, y_1], \ldots, [x_i, y_{c(v)}]$. We sort the vertices of X where the key of a vertex x is $N(x)$. Afterwards, we split X into sets of equal keys $X^1, X^2, \ldots, X^{\text{clusters}_u}$ (i.e., for every two vertices $x, x' \in X^i$, every edge $[x, y_j]$ has the same

cost as the edge $[x', y_j]$). Note that every set $X^i \cup Y$ induces a complete bipartite subgraph of G.

We now build a network G^* whose vertices are $V^* = X \cup Y \cup \{c_1, \ldots, c_{\text{clusters}_u}, s, t\}$. The edges are $E^* = E_1 \cup E_2 \cup E_3$ where $E_1 = \{[s, x_i] : x_i \in X\} \cup \{[y_i, t] : y_i \in Y\}$, $E_2 = \{[x_i, c_j] : j \leq \text{clusters}_u, x_i \in X^j\}$, and $E_3 = \{[c_j, y_i] : j \leq \text{clusters}_u, y_i \in Y\}$. All edges have capacity 1. Edges from sets E_1 and E_2 are assigned a cost of zero. An edge of type E_3 from c_j to y_i is assigned a cost which is identical to the cost of the edge $[x, y_i]$ where x is any vertex belonging to set X^j. The source is s and the sink is t. We find a min-cost max flow f^* in G^* and construct from this flow a maximum weight matching in G.

Let G' be the network that corresponds to G as defined in Section 2.

Lemma 4. *A min-cost mas flow in G' can be found by finding a min-cost max flow in G^*.*

Proof. We define a mapping $\mathcal{M}$ that maps paths in G^* to paths in G'. The mapping $\mathcal{M}$ is by partitioning sub-paths in G^* into subunits which are then mapped onto their corresponding subunits in G'. Consider the 6 cases of sub-unit mappings shown in Figure 5. Note that this mapping preserves the endpoints and the costs of each subunit except the subunit of type 4 which does not have a mapping. Thus we can partition a path in G^* into subunits, map each subunit into subunit in G' and then reconstruct the path in G' with exactly the same cost end the same endpoints. A subunit of type 4 can be omitted during the mapping since it donates only 0 cost to the pathway in which it participates.

Following the algorithm of [4] for computing minimum cost maximum flow, a series of $d(u)$ stages is executed, where in each stage a new flow f_i^* is computed which is an increment by one of the previous flow. Each increment step is realized by computing the minimum cost path from s to t and infusing one flow unit along this path. Using the mapping $\mathcal{M}$ as demonstrated in Figure 5, each such increment path in the graph G^* can be mapped into a corresponding minimum cost increment path in G'. Therefore, in any stage i of the min-cost max-flow computation on G^*, f_i^* is equal in cost and value to the corresponding flow f_i in G', as induced by $\mathcal{M}$. Clearly, any cycle in G^* is mapped by $\mathcal{M}$ onto a cycle in G' with exactly the same cost. Thus, if there is a negative cost cycle in the residual graph of G^* at any stage of the min-cost max flow computation, then it immediately follows that there is a corresponding negative cost cycle in the residual graph of G'. According to Lemma 1, if there is a negative cost cycle in the residual graph of G', then the flow is not minimum cost. Thus if there is a negative cost cycle in residual graph of G^* then the corresponding flow in graph G' is not minimal, in contradiction to [4]. □

We denote by $D(u)$ the number of distinct trees in the forest $p^u_{x_1}, \ldots, p^u_{x_{d(u)}}$.

Lemma 5. *The maximum weight matching in the bipartite graph corresponding to $u \in P$ and $v \in T$ can be computed in $O(d(u) \cdot (D(u)c(v) + c(v) \log c(v)))$ time.*

Proof. If some pair of rooted trees $p^u_{x_i}$ and $p^u_{x_j}$ are isomorphic, then in the graph G, the vertices x_i and x_j have exactly the same neighbors, and this remains true in G^*. Therefore

$\text{clusters}_u \leq D(u)$. The time for constructing G is $O(d(u)c(v))$ and the time for constructing G^* is $O(d(u)c(v)\log c(v))$. We now bound the time for finding a min-cost max flow in G^*: The size of E_1 is at most $|X|+|Y| = d(u)+c(v)$. The size of E_2 is $|X| = d(u)$ and the size of E_3 is $|\text{clusters}_u \cdot c(v)|$. Hence, the number of edges in G^* is $O(\text{clusters}_u \cdot c(v))$. The number of vertices in G^* is at most $d(u) + c(v) + \text{clusters}_u + 2 = O(c(v))$, since $d(u) - 1 \leq c(v)$. Now, the maximum weight matching algorithm performs $d(u)$ stages, and each stage takes $O(E^* + V^* \log V^*)$ time. Hence, the total time is $O(d(u) \cdot (\text{clusters}_u \cdot c(v) + c(v)\log c(v))) = O(d(u) \cdot (D(u)c(v) + c(v)\log c(v)))$ time. □

Time Complexity Analysis

By Lemma 5, the total time complexity is

$$O(\sum_{u=1}^{m}\sum_{v=1}^{n}(d(u)(\text{clusters}_u \cdot c(v)) + c(v)\log c(v))) \overset{\text{obs. 1}}{=} O(\sum_{u=1}^{m} d(u) \cdot \text{clusters}_u \cdot n + d(u)n\log n)$$
$$\overset{\text{obs. 2}}{=} O(n\sum_{u=1}^{m} d(u)D(u) + mn\log n).$$

We now turn to bound the summation $O(\sum_{u=1}^{m} d(u)D(u))$. The following lemma sets an asymptotically tight bound on the number of distinct labelled rooted trees in a forest of n vertices, denoted $f(n)$.

Lemma 6. *Assuming constant label alphabet,* $f(n) = O(n/\log n)$

Proof. We will prove that for any forest with n vertices labelled by an alphabet of size σ, $f(n) = O(n\log\sigma/\log n)$. It is known that the number of distinct rooted unlabeled trees with i vertices is at most c^i for some constant c. Therefore, the number of distinct rooted labelled trees with i vertices is at most $(c\sigma)^i$.

If we have a forest of labeled rooted trees and r_i is the number of trees with i vertices, then the number of distinct trees in this forest is at most $\sum_{i=1}^{n}\min(r_i, (c\sigma)^i)$. Hence,

$$f(n) \leq \max\left\{\sum_{i=1}^{n}\min(r_i,(c\sigma)^i : r_1,\ldots,r_n \in \text{N}, \sum_{i=1}^{n} ir_i \leq n\right\}$$

Let x be the minimum integer for which $\sum_{i=1}^{x} i(c\sigma)^i \geq n$. Let $r_1,\ldots,r_n$ be the integers that maximize $\sum_{i=1}^{n}\min(r_i,(c\sigma)^i)$ under the constraint $\sum_{i=1}^{n} ir_i \leq n$. Using the arguments of [25], we can assume that $r_i \leq (c\sigma)^i$ for all i and $r_i = 0$ for all $i > x$. Therefore, $f(n) \leq \sum_{i=1}^{x}(c\sigma)^i = O((c\sigma)^x)$. The lemma follows from the fact that $x = \log_{c\sigma} n - \log_{c\sigma}\log_{c\sigma} n - O(1)$. □

Lemma 7. $\sum_{u=1}^{m} d(u)D(u) = O(m^2/\log m)$.

Proof. Let ϵ denote some constant $0 < \epsilon < 1/2$. We call a vertex of P *heavy* if its degree is at least $2m^{1-\epsilon}$, otherwise the vertex is called *light*. Clearly, the number of heavy vertices is at most m^ϵ. If u is a heavy vertex and v is a neighbor of u such that p_v^u does not contain a heavy vertex, then we call every vertex in p_v^u a *private vertex of* u. Denote the vertices of P by $v_1, \ldots, v_m$. We denote by ℓ_j the number of the private vertices of a heavy vertex of v_j. We split $\sum_{u=1}^{m} d(u)D(u)$ into two sums. Summing over the light vertices of P we have

$$\sum_{j:v_j \text{ is light}} d(v_j)D(v_j) \leq \sum_{j:v_j \text{ is light}} d(v_j)^2 \leq (2m^{1-\epsilon}) \sum_{j:v_j \text{ is light}} d(v_j) \overset{\text{obs. 2}}{\leq} (2m^{1-\epsilon}2m) = 2^2 m^{2-\epsilon}.$$

Shamir and Tsur [25] proved that, for any heavy vertex v_j, $d(v_j) \leq m^\epsilon + \ell_j$ and $D(v_j) \leq m^\epsilon + f(\ell_j)$. Therefore, summing over the heavy vertices we have

$$\sum_{j:v_j \text{ is heavy}} d(v_j)D(v_j) \leq \sum_{j:v_j \text{ is heavy}} (m^\epsilon + \ell_j)(m^\epsilon + f(\ell_j)).$$

Since $f(\ell_j) \leq \ell_j \leq m$ and since the number of heavy vertices is at most m^ϵ, we have

$$\leq \sum_{j:v_j \text{ is heavy}} (m^{2\epsilon} + m^{1+\epsilon} + m^{1+\epsilon} + \ell_j(f(\ell_j))) \leq 3m^{1+2\epsilon} + \sum_{j:v_j \text{ is heavy}} \ell_j f(\ell_j).$$

By Lemma 6, the fact that $\sum_{j:v_j \text{ is heavy}} \ell_j \leq m$ (as each vertex can be a private vertex of at most one heavy vertex), and the fact that the function $h(x) = x^2/\log x$ is convex,

$$\leq 3m^{1+2\epsilon} + \sum_{j:v_j \text{ is heavy}} c{\ell_j}^2/\log \ell_j \leq 3m^{1+2\epsilon} + cm^2/\log m. \qquad \square$$

We have thus proved the following theorem.

Theorem 3. *Algorithm 3 computes the optimal ALSH solution for two unrooted unordered trees in* $O(m^2 n/\log m + mn \log n)$ *time.*

5 Solving ALSH on Ordered Trees

5.1 Ordered Rooted Trees

A simplified version of Algorithm 1 can be used to solve the ALSH problem on ordered rooted trees. The simplification is based on the fact that the assignment problems now turn into maximum weight matching problems on ordered bipartite graphs, where no two edges are allowed to cross in the matchings. (We refer the reader to [28] for a discussion of non-crossing plain bipartite matching in the context of exact ordered tree homeomorphism.) Also note that, in the context of our ALSH solutions, we apply a rectangular case of the

perfect assignment problem on the graph $G = (X \cup Y, E)$, i.e. $|X| \leq |Y|$ and all nodes in X must eventually be paired with some node in Y. Therefore, the assignment computation reduces to the *Approximate Weighted Episode Matching* optimization problem, as defined below.

Definition 6. *The* ***Approximate Weighted Episode Matching Problem*** *is given a pattern string X, a source string Y, and a character-to-character similarity table $\Delta[\Sigma_X, \Sigma_Y]$, to find among all $|X|$-sized subsequences of Y, a subsequence Q which is most similar to X under Δ (i.e. a subsequence Q that maximizes the sum $\sum_{i=1}^{|X|} \Delta[Q_i, X_i]$).*

Lemma 8. *The Approximate Weighted Episode Matching problem can be solved in $O(|X| \cdot |Y|)$ time.*

Proof. The matching problem is solved by applying the classical dynamic programming string alignment algorithm on a $|X| + 1$ rows by $|Y| + 1$ columns graph (Figure 6). All horizontal edges in the graph, corresponding to character deletions from Y, are assigned a score of zero. All vertical edges in the graph, corresponding to character deletions from X, are assigned a score of $-\infty$. A diagonal edge leading into vertex (x_i, y_j) corresponds to the string-edit operation of substituting the ith character of X with the jth character of Y, and is therefore assigned the score $\Delta[i, j]$. During the initialization stage, scores of all vertices in the first row of the dynamic programming graph are set to zeros. The dynamic programming algorithm is then applied to this alignment graph.

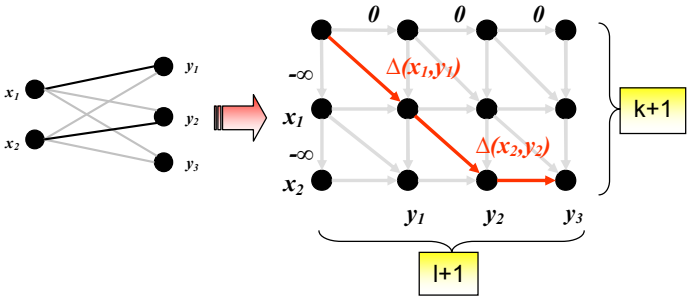

Figure 6: Approximate Weighted Episode Matching Calculation. The matching is computed on the dynamic programming graph with score $-\infty$ on vertical the edges, score 0 on the horizontal edges and $\Delta[x_i, y_j]$ scores on the diagonal edges.

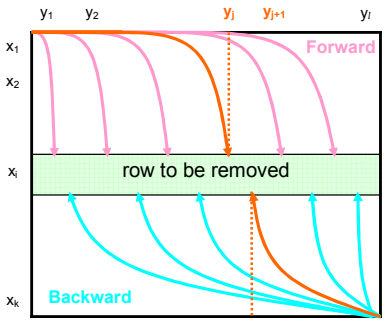

Figure 7: The calculation of the best possible alignment between $X - x_i$ and Y.

When the algorithm completes, all vertices in the last row of the graph are scanned for the best score. Any vertex in the last row which carries the optimal score represents a highest scoring approximate weighted episode of X in Y. □

Clearly, after running the algorithm described in the proof of Lemma 8 on the alignment graph constructed for strings X and Y, a highest-scoring path P in the alignment graph will correspond to a maximum weight matching M in the corresponding ordered bipartite graph

$G = (X \cup Y, E)$, where no two edges in the matching are allowed to cross. A diagonal edge in P connecting vertex $(x_i - 1, y_j - 1)$ with vertex (x_i, y_j), corresponds to a pairing of x_i and y_j in M.

Time Complexity Analysis

Theorem 4. *The ALSH problem on rooted ordered trees can be solved in $O(mn)$ time.*

Proof. By Lemma 8, the time complexity is

$$\sum_{v=1}^{n}\sum_{u=1}^{m} O(c(v)\cdot c(u)) \overset{\text{obs. 1}}{=} \sum_{v=1}^{n} O(m\cdot c(v)) \overset{\text{obs. 1}}{=} O(mn). \qquad \square$$

5.2 Ordered Unrooted Trees

In this section we extend the algorithm for ordered rooted trees to apply to unrooted trees. There are two standard ways to order unrooted trees: the cyclic order and the linear one.

A linearly ordered set X of children of a node u is a poset $(X, \leq)$ (partial order relation on X) which has the property of comparability:

$$\text{for all } x, y \in X, \text{ either } x \leq y \text{ or } y \leq x.$$

The binary relation $\leq$ is then called a linear ordering.

A cyclic order on a set X of children of node u with k elements is an arrangement of X as on a clock face, for an k-hour clock. That is, rather than an order relation on X, we define on X just functions "element immediately before" and "element immediately following" any given x_i, in such a way that taking predecessors, or successors, cycles once through the elements as $x_1, x_2, \ldots, x_k$.

Both orders are illustrated in Figure 8.

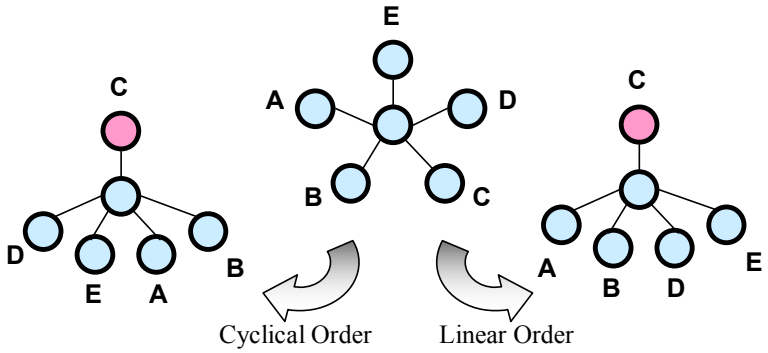

Figure 8: Cyclic and linear orders of unrooted trees.

5.2.1 Cyclically Ordered Unrooted Trees.

The problem of ALSH on cyclically ordered unrooted trees can be reduced to the *Cyclic Sting Comparison* problem, defined as follows: Given two strings P and T, find a cyclic shift of P which can be best aligned with T. More precisely, let $T = T_1 \cdots T_\ell$ and $P = P_1 \cdots P_k$. We wish to find an index i for which the optimal alignment of $P_i P_{i+1} \cdots P_k P_1 \cdots P_{i-1}$ with T gives the best score. An $O(kl \log k)$ algorithm for the Cyclic String Comparison problem, which applies to general weights scoring schemes, was described in [**?**]. An $O(kl)$ algorithm which applies to rational number scoring schemes was given in [**?**].

An efficient solution to ALSH on Cyclically Ordered Unrooted Trees can be obtained by applying the following reduction. Consider computing the $(2k+1) \times (l+1)$ grid graph for the comparison of the string PP (P concatenated with P) versus T. (see Figure 9.A). Each cyclic shift of P defines a starting point in the source area for the alignment path to the appropriate point in the destination area. The alignment scores can be computed efficiently in an incremental manner using [**?**, **?**]. Our problem is actually a variant of this problem where in each increment step another neighbor of node u serves as its parent and thus can not participate in the alignment. Therefore, the sliding window in our problem is of size $k-1$ instead of k (see Figure 9.B).

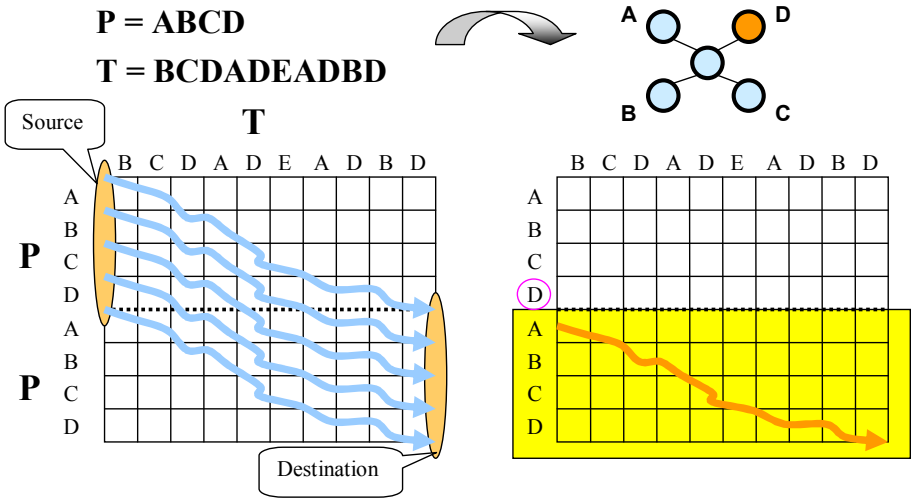

Figure 9: Cyclic string comparison problem.

Time Complexity Analysis

The time complexity for each pair of vertices is is $O(kl \log k)$ using the algorithm of Maez [**?**] or $O(kl)$ for rational number scoring scheme using the algorithm of Schmidt [**?**]. Therefore, summing up the work over pairs of vertices yields a total time complexity of $O(mn \log m)$ and $O(mn)$, respectively.

Linearly Ordered Unrooted Trees As mentioned above, the naive extension of rooted case requires an additional order in the time complexity since each of the neighbors of node u in pattern tree eventually could serve as the parent of u and will not participate in the children's matching for this computation. In this section we use the decomposition technique for the linearly ordered case to reduce the complexity to the rooted case bounds.

Our technique is based on Hirschberg's algorithm for the linear space sequence alignment [12]. Similar to Hirschberg, we calculate the Forward "alignment" using the algorithm for the rooted ordered trees and Backward "alignment" which is the same as the Forward, but in the opposite direction. Then we throw out the line i, which represent the the current father of node u and stitch Forward and Backward "alignments" by finding the best *stitching point* (see Figure 7):

$$stitching\_point = \max\{F[i-1,j] + B[i+1,j+1]\} \qquad \text{for } j = 0,\ldots,n-1.$$

Clearly, the value $F[i-1,j]$ represents the best alignment between $X[0\ldots i-1]$ and $Y[0\ldots j]$. The value $B[i+1,j+1]$ represents the best alignment between $X[m\ldots i+1]$ and $Y[n\ldots j+1]$. Therefore, the maximal sum corresponds to an an optimal alignment of $X - x_i$ and Y.

Time Complexity Analysis The calculations of both Forward and Backward alignments requires $O(mn)$ time. The calculation of the stitching point requires $O(n)$ time. Therefore, the overall time complexity is $O(mn)$. □

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[2] D. Carmel, N. Efrati, G.M. Landau, Y.S. Maarek, and Y. Mass. An extension of the vector space model for querying xml documents via xml fragments. In *XML and Information Retrieval (Workshop) Tampere, Finland*, 2002.

[3] M. J. Chung. $O(N^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *J. Algorithms*, 8:106–112, 1987.

[4] G.B. Dantzig, L. R. Ford, and D. R. Fulkerson. A primal-dual algorithm for linear programs. In H. W. Kuhn and A. W. Tucker, editors, *Linear Inequalities and Related Systems*. Princeton Univ. Press, Princeton NJ, 1956.

[5] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[6] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. of the Assoc. for Comput. Mach.*, 19(2):248–264, 1972.

[7] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proceedings 23rd Symposium on the Theory of Computing (STOC 91)*, pages 123–133, 1991.

[8] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the Association for Computing Machinery*, 34(3):596–615, 1987.

[9] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1012–1036, 1989.

[10] P. B. Gibbons, R. M. Karp, and D. Soroker. Subtree isomorphism is in random NC. *Discrete Applied Mathmatics*, 29:35–62, 1990.

[11] A. V. Goldberg, S. A. Plotkin, and P.M Vidya. Sublinear-time parallel algorithms for matching and related problems. *J. Algorithms*, 14:180–213, 1993.

[12] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[13] L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.

[14] M. Y. Kao, T. W. Lam, W. K. Sung, and H. F. Ting. Cavity matchings, label compressions, and unrooted evolutionary trees. *SIAM J.Comput.*, 30(2):602–624, 2000.

[15] M. Karpinski and A. Lingas. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.

[16] P. Kilpelainen and H. Mannila. Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995.

[17] D. W. Matula. An algorithm for subtree identification. *SIAM Rev.*, 10:273–274, 1968.

[18] D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978.

[19] H. Ogata, W. Fujibuchi, S. Goto, and M. Kanehisa. A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. *Nucleic Acids Research*, 28:4021–4028, 2000.

[20] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum cycle mean problems. *Math. Prog.*, 54:541–561, 1992.

[21] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, and M. Ziv-Ukelson. A new tool for the alignment of metabolic pathways. *Manuscript*, 2004.

[22] S. W. Reyner. An analysis of a good algorithm for the subtree problems. *SIAM J. Comput.*, 6:730–732, 1977.

[23] T. Schlieder and F. Naumann. *Approximate Tree Embedding for Querying XML Data*. ACM SIGIR Workshop on XML and IR, 2000.

[24] F. Schreiber. Comparison of metabolic pathways using constraint graph drawing. In *Proceedings of the Asia-Pacific Bioinformatics Conference (APBC'03), Conferences in Research and Practice in Information Technology*, 19, pages 105–110, 2003.

[25] R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.

[26] M. A. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

[27] R. E. Tarjan. *Data Structures and Network Algorithms*. SIAM, Philadelphia, 1982.

[28] G. Valiente. Constrained tree inclusion. In *Proceedings of 14th Annual Symposium of Combinatorial Pattern Matching (CPM '03)*, volume 2676 of *Lecture Notes in Computer Science*, pages 361–371, 2003.