Approximate Labelled Subtree Homeomorphism

Ron Y. Pinter[1,⋆], Oleg Rokhlenko[1,⋆], Dekel Tsur[2], and Michal Ziv-Ukelson[1,⋆⋆]

[1] Dept. of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel
`{pinter, olegro, michalz}@cs.technion.ac.il`
[2] Caesarea Rothschild Institute of Computer Science, University of Haifa, Haifa 31905, Israel
`dekelts@cs.haifa.ac.il`

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, denoted *ALSH* (Approximate Labelled Subtree Homeomorphism) 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.

1 Introduction

The matching of labelled tree patterns, as opposed to linear sequences (or strings), 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* [15, 16, 23]: Given a pattern tree P and a text tree T, find a subtree of T which is isomorphic to P (i.e. find if some 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.

⋆ Research supported in part by the Bar-Nir Bergreen Software Technology Center of Excellence.

⋆⋆ Research supported in part by the Aly Kaufman Post Doctoral Fellowship and by the Bar-Nir Bergreen Software Technology Center of Excellence.

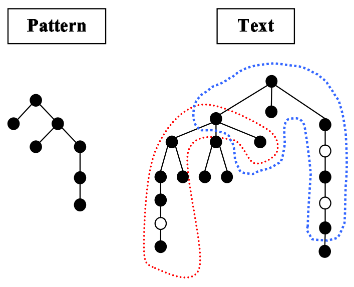

Fig. 1. Subtree homeomorphism. The white nodes in the text are degree-2 nodes that have been removed. Two subtrees in the text that are homeomorphic to the pattern are circled by the dashed lines.

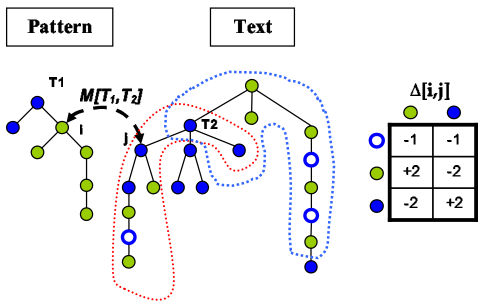

Fig. 2. Approximate Labelled Subtree Homeomorphism. The node-label similarity scores are specified in table Δ, and $D = -1$. Two subtrees in the text that are homeomorphic to the pattern are circled by the dashed lines. The left homeomorphic subtree has an LSH score of 5, while the right one has an LSH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

The *subtree homeomorphism problem* [3, 20] 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 [26] 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 polynomial-time algorithms. The more advanced *tree inclusion problem* [14] is that 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 to v and inserting edges connecting the parent of v with the children of v. *Tree inclusion* on unordered trees is NP-complete [14]. Schlieder and Naumann [21] 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.

This paper addresses queries on labelled trees. The trees could 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 [17, 22] 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 [19]). 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 [21]: 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, in this paper we address 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.

Let Δ denote a predefined node-to-node similarity score table and $\mathcal{D}$ denote a predefined (usually negative) score for deleting a node from a tree (Figure 2). A *mapping* $\mathcal{M}[T_1, T_2]$ from T_1 to T_2 is a partial 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. *Consider two labelled trees T_1 and T_2, such that T_2 is homeomorphic to T_1, and let $\mathcal{M}[T_1, T_2]$ denote a node-to-node, homeomorphism-preserving mapping from T_1 to T_2. The* **Labelled Subtree Homeomorphism Similarity Score** *of $\mathcal{M}[T_1, T_2]$, denoted $LSH(\mathcal{M}[T_1, T_2])$, is*

$$LSH(\mathcal{M}[T_1, T_2]) = \mathcal{D} \times (|T_2| - |T_1|) + \sum_{(u,v) \in \mathcal{M}} \Delta[u, v]$$

Correspondingly,

Definition 2. *The* **Approximate Labelled Subtree Homeomorphism (ALSH) Problem** *is, given two undirected labelled trees P and T, and a scoring table that 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 $\mathcal{M}[P, t]$ from P to some subtree t of T, such that $LSH(\mathcal{M}[P, t])$ is maximal.*

In this paper we show how to compute optimal, bottom-up alignments between P and every 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 weighted assignments in bipartite graphs. The ALSH problem is recursively translated into a collection of smaller ALSH problems, which are solved using weighted assignment algorithms (Figure 3). (Simpler, maximum bipartite perfect matching algorithms were applied in the algorithms for exact subtree morphisms [3, 9, 13, 23, 26].)

Our approach yields an $O(m^2n/\log m+mn\log n)$ 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 [23, 26], 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.

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 [12, 24]. 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 $O(m^2n + mn\log n)$ 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 $O(m^2n/\log m + mn\log n)$ solution for both rooted and unrooted unordered trees. In Section 5 we describe the $O(mn)$ time ALSH algorithm for rooted ordered trees.

Note that due to lack of space, most proofs are omitted. The proofs will be given in the journal paper.

2 Weighted Assignments and Min-Cost Max Flows

Definition 3. *Let $G = (V = X \cup Y, E)$ be a bipartite graph with edge weights $w(x, y)$ for all edges $(x, y) \in E$, where $x \in X$ and $y \in Y$. The Assignment Problem is to compute a matching M, i.e. a list of monogamic pairs $(x \in X, y \in Y)$, such that the size of M is maximum among all the matchings in G, and $\sum_{(x,y)\in M} w(x, y)$ is maximum among all the matchings with maximum size.*

Although researchers have developed several different algorithms for the assignment problem [1], many 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 [11]. The reduction from an assignment problem to a 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 $-cost(f)$ given by the set of edges $\{x, y\}$ such that (x, y) has flow one. 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 we can solve a matching problem on G by solving a flow problem on G'.

Edmonds and Karp's algorithm [5] finds a minimum cost maximum flow in G' in $O(EV \log V)$ time. In each stage of this algorithm, Dijkstra's algorithm [4] is used to compute an augmentation 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 match. Fredman and Tarjan [7] 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 (the running time is polynomially bounded in only V and E, independent of the edge costs) for solving the assignment problem.

Under the assumption that the input costs are integers in the range $[-C, \ldots, C]$, Gabow and Tarjan [8] used 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 [10, 18]. The scaling algorithms are conditioned by the *similarity assumption* (i.e. the assumption that $C = O(n^k)$ for some constant k) and have the integrality of cost function restriction.

The following lemma, which was first stated and proven in [11], will serve as the basis for the proofs of Lemma 2 and Lemma 4.

Lemma 1 ([5, 11, 25]). *A flow f has 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

We use $d(v)$ to denote the number of neighbors of a node v in an unrooted tree, and $c(v)$ to denote the number of children of a node v in a rooted tree.

Let $T^r = (V_T, E_T, r)$ be the text tree which is rooted in r, and $P^{r'} = (V_P, E_P, r')$ be the pattern tree which is rooted in r'. Let $p_u^{r'}$ denote a subtree of $P^{r'}$ which is rooted in node u of $P^{r'}$, and t_v^r denote a subtree of T^r which is rooted in node v of T^r.

We define $RScores[v \in V_T, u \in V_P]$ as follows.

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

The first algorithm, denoted Algorithm 1, computes the values $RScores[v,u]$ recursively, in a *postorder* traversal of T^r. First, $RScores[v,u]$ are computed for all leaf nodes of T^r and $P^{r'}$. Next, $RScores[v,u]$ are computed for each node $v \in V_T$ and $u \in V_P$, based on the values of the previously computed scores for all children of v and u as follows. Let u be a node of $P^{r'}$ with children

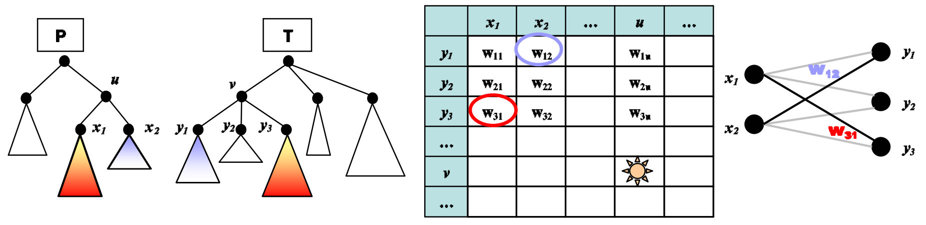

Fig. 3. The work done by the first algorithm during the computation of $RScores[v, u]$. The bipartite graph is constructed in order to compute the optimal weighted assignment between the children of u and those of v. w_{ij} marks the optimal LSH similarity score for aligning the subtrees rooted at $y_j \in T$ and $x_i \in P$.

$x_1, \ldots, x_{c(u)}$ 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) in 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 sum of the weights of the matched edges in the maximal assignment over G. Note that this term is only computed if $c(u) \leq c(v)$.
2. The weight $RScores[y_j, u]$ for the comparison of u and the best scoring child y_j of v, updated with the penalty for deleting v.

Upon completion, the optimal LSH similarity score, denoted $best\_score$, where $best\_score = \max_{j=1}^{n} RScores[y_j, x_m]$, is found, and 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.

Time Complexity Analysis.

Theorem 1.

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

Proof. The time complexity analysis of Algorithm 1 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 computes a score once for each node pair $(v \in T, u \in P)$. The dominant part of this computation is spent in obtaining the weighted assignment between a bipartite graph with $c(v)+c(u)$ nodes and $c(v)\times c(u)$ edges. Therefore, based on Fredman and Tarjan's algorithm [7], each assignment computation takes $O(c(u)(c(u)\times c(v)+c(v)\log c(v))) = O(c(u)^2\times c(v)+c(u)\times c(v)\log c(v))$ time.

Summing up the work spent on assignment computations over all node pairs, we get

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

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, Algorithm 1 can be modified to run in time $O(m^{1.5}n\log(nC))$ by employing the algorithm of Gabow and Tarjan [8] for weighted bipartite matching with scaling. □

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 between P and T could be computed in a naive manner as follows. Select an arbitrary node r of T to get the rooted tree T^r. Next, for each $u\in P$ compute rooted ALSH between P^u and T^r. This method will yield 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, correspondingly, by utilizing the decremental properties of the weighted assignment algorithm.

The second algorithm, denoted Algorithm 2, starts by selecting a vertex r of T to be the designated root. T^r is then traversed in 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 eventually serve as a parent of u in a considered mapping of a subtree of P, which is rooted in u, with a homeomorphic subtree of t^r_v. Suppose that node x_i, which is a neighbor of u, 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 the consideration of the highest scoring weighted matching between the children of v, and all neighbors of u except x_i. Since each neighbor x_i of u, $i=1,\ldots,d(u)$, is a potential parent of a subtree rooted in u, our algorithm will actually need to compute a series of weighted assignments for the graphs $G_i=(X_i\cup Y,E_i)$, $i=1,\ldots,d(u)$, where Y consists of the children of v in T^r, while X_i consists of all neighbors of u except x_i. Therefore, we define $\mathit{UScores}[v \in V_T, u \in V_P, x_i \in \mathit{neighbors}(u)]$ as follows.

Definition 5. *For each vertex $v \in T^r$, each vertex $u \in P$, and every vertex $x_i \in \mathit{neighbors}(u)$, $\mathit{UScores}[v, u, x_i]$ is the maximal LSH similarity score between a subtree $p_u^{x_i}$ of P and a corresponding homeomorphic subtree of t_v^r, if one exists. Otherwise, $\mathit{UScores}[v, u, x_i]$ is set to $-\infty$.*

For the graphs G_i defined above, the weight of an edge $(x_j, y_{j'})$ in G_i is $\mathit{UScores}[y_{j'}, x_j, u]$.

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

1. The node-to-node similarity value $\Delta[v, u]$, plus the sum of the weights of the matched edges in the maximal weighted assignment over G_i. Note that this term is computed only if $d(u) - 1 \leq c(v)$.
2. The weight $\mathit{UScores}[y_j, u, x_i]$ of the comparison of u and the best scoring child y_j of v, updated with the penalty for deleting v.

Since each node $u \in P$ is also a potential root for P in the sought alignment, an additional entry $\mathit{UScores}[v, u, \phi]$ is computed, which stores the maximal LSH similarity score between a subtree of P^u and a corresponding homeomorphic subtree of t_v^r, if one exists. $\mathit{UScores}[v, u, \phi]$ is computed from the weighted assignment for the graph $G = (X \cup Y, E)$, where X is the set of neighbors of u.

Upon completion, the maximal LSH similarity score $\mathit{best\_score} = \max_{v \in V_T, u \in V_P} \mathit{UScores}[v, u, \phi]$ is computed, and every node pair $(j \in T, i \in P)$ with $\mathit{UScores}[j, i, \phi] = \mathit{best\_score}$ is reported as a subtree t_j^r of T^r which bears maximal similarity to tree P^i under the LSH measure.

Time Complexity Analysis.

The time complexity bottleneck of Algorithm 2 is based in the need to compute, for each comparison of a pair $u \in P$ and $v \in T$, weighted assignments for 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 weighted assignment is computed for each comparison of a 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 assignments for the full series of G_i graphs in the same time complexity as the assignment computation for the single graph G. In the following, let $k = |X|$ and $\ell = |Y|$. Note that $k - 1 \leq l$.

Lemma 2. *Let G be the bipartite graph constructed by Algorithm 2 for some pair of vertices $v \in V_T$ and $u \in V_P$. Given an optimal assignment of G, the optimal assignment of G_i can be computed via one run of a single-source shortest-path computation on a subgraph of G.*

Proof. The proof is based on the reduction of the assignment problem to min-cost max-flow, as described in Section 2. We will show that after having computed the

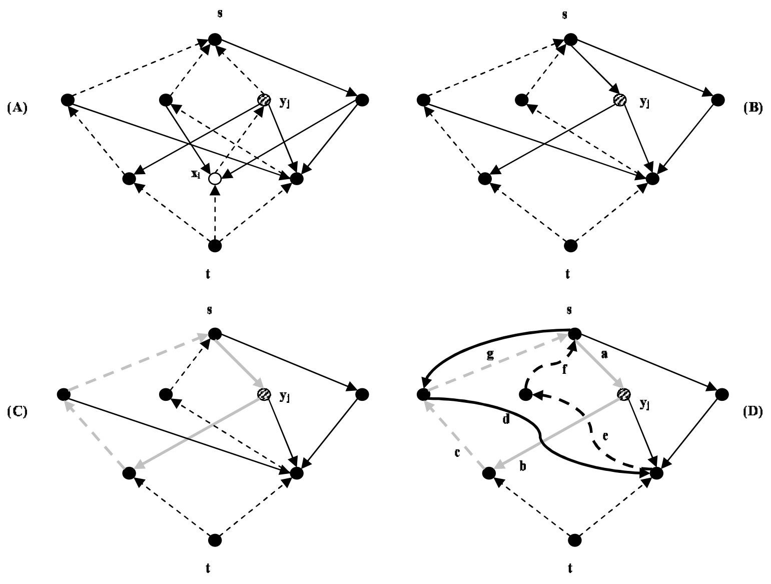

Fig. 4. The computation of the optimal assignment for $X_i \cup Y$. Note that, for the sake of graphical clarity, some of the edges of the graph are not shown in the figure. (A) The residual graph R for flow f. The dashed lines denote the backward edges and the solid lines denote the forward edges. The 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 "correction 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.

min-cost max-flow of G', which defines the optimal assignment for G, the optimal assignment for G_i can be obtained via one more augmenting path computation.

Let f denote the minimum cost maximum flow on G', defining the the optimal weighted assignment M, and let R denote the residual graph for f (Figure 4.A).

Since f is optimal, by Lemma 1, there are no negative cost cycles in R. Also, since $\ell \geq k$, $|M| = k$. Let y_i be the matched neighbor of x_i in M, i.e., $(x_i, y_i) \in M$. Let M' denote the matching $M \setminus (x_i, y_i)$, i.e. the matching obtained by removing the pair (x_i, y_i) from M. We say that y_i is "newly widowed" in M'. Clearly, $|M'| = k - 1$. Let f' be the flow defined by M'. Let G'' denote the subgraph of G' obtained by removing vertex x_i and all edges adjacent to x_i in G'. Let R' be the "decremented" residual subgraph of R, obtained by removing vertex x_i and all edges adjacent to x_i in R, as well as cancelling the flow on edge (s, y_i) 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 the optimal assignment of size $k - 1$ on 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 begin with edge (s, y_i), since this edge is the only edge of R' which did not exist in R, and R had no negative cost cycles.

Among all such negative cost cycles in R', all of which start with edge (s, y_i), we wish to find the one which contributes the most to decrementing the value of f'. Therefore, let p denote the **"correction path"** for R', defined as the the min-cost path in R' among all paths which originate in s, pass through the newly widowed vertex y_i and end back in s (Figure 4.C). Clearly, p can not start from some other edge than (s, y_i) since all $\ell - k$ vertices in Y which were unmatched in M are "out of the game" and will remain unmatched in the optimal assignment of size $|M|-1$ on G_i. Otherwise, the assumed optimality of M would be contradicted.

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_i and back to s via path p, is the minimal cost maximum value flow for network graph G'' and defines, respectively, the optimal assignment for 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 there was some negative cost cycle c in R''. Since R had no negative cost cycles, and the only new edge $(s, y_i) \in R'$ 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, cycle c consists of edges in R' and one or more edges whose reversals are on p. Therefore, 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 $\mathit{cost}(p) + \mathit{cost}(c)$, 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, $\mathit{cost}(p) + \mathit{cost}(c) < \mathit{cost}(p)$. Furthermore, $p \oplus c$ can be partitioned into a path p' originating in s, going through y_i and ending in s, plus a collection of cycles. (Note that none of these cycles includes edge (s, y_i), since this edge is already occupied by p'). Clearly, all the cycles must have nonnegative cost since they consist of edges only from R' which has no negative cost cycles by Lemma 1. Thus path p' is a "correction path" of cost less than p (Figure 4.D). This contradiction implies the lemma.

Note that the above proof assumes $k \leq \ell$, in which case $|M| = k$. However, the special case $k = \ell + 1$ can be easily handled by adding a dummy vertex to Y and connecting it by edges to all the vertices in X. The weight of these edges should be set to N for some very large number N. $\square$

Lemma 3.

1. *Computing the weighted assignments for the bipartite graphs $G_1, \ldots, G_k$ can be done in $O(k^2\ell + k\ell \log \ell)$ time.*
2. *Under the similarity assumption, computing the weighted assignments for the bipartite graphs $G_1, \ldots, G_k$ can be done in $O(k^{1.5}\ell \log(\ell C))$ time [12].*

Theorem 2.

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

Proof. We use the following simple observation:

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

Algorithm 2 computes a score for each node pair $(v \in T, u \in P)$. During this procedure, the algorithm computes weighted assignments of several bipartite graphs. Due to the decremental relationship between these graphs, it is shown in Lemma 3 that computing the maximal assignment for all graphs in the series can be done in the same time bound taken by computing a single assignment. This computation, which is the dominant part of the procedure's time, can be done in $O(d(u)^2c(v) + d(u)c(v) \log c(v))$ time. Therefore, the total complexity is

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

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 [8] for weighted bipartite matching with scaling and [12] for cavity matching. $\square$

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 time complexity of the first, dominant term in the time complexity of Algorithm 2.

4 A More Efficient ALSH Algorithm for Unordered Trees

In this section we show how the dominant term in the time complexity of the algorithms 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 maximum matching problems more efficiently by employing the notion of clique partition of a bipartite graph [6, 23]. The modified algorithm, denoted Algorithm 3, is the same as Algorithm 2 with the exception that, in step 12, we solve the assignment problem differently. Let v denote some vertex in T^r whose children are $Y = y_1, \ldots, y_{c(v)}$ and let u denote a vertex in P whose neighbors are $X = x_1, \ldots, x_{d(u)}$. We now attempt to compute the Assignment of $G = (X \cup Y, E)$. We will partition the edges of G into complete bipartite graphs $C_1, C_2, \ldots, C_{clusters_u}$. Let $Key(x_i)$ be a vector containing the weights of the edges $(x_i, y_1), \ldots, (x_i, y_{c(v)})$. We do the partition in the following way: First, we sort the vertices of X in lexicographic order where the key of a vertex x is $Key(x)$. Afterwards, we split X into sets of equal keys $X^1, X^2, \ldots, X^{clusters_u}$ (i.e., for any two vertices $x, x' \in X_i$, any edge (x, y_j) has the same weight as edge (x', y_j) for all vertices $y_j \in Y$).

Now, for $1 \leq i \leq clusters_u$ we set C_i to be the subgraph induced by the vertices of X^i and all their neighbors in Y. We now follow the method of [6, 23] and build a network G^* whose vertices are $V^* = X \cup Y \cup \{c_1, \ldots, c_{clusters_u}, s, t\}$. The edges are $E^* = E_1 \cup E_2 \cup E_3$ where $E_1 = \{[s, x_i] : x_i \in X\} \cup \{[y_i, t] : y_i \in Y\}$, $E_2 = \{[x_i, c_j] : j \leq clusters_u, x_i \in C_j\}$, and $E_3 = \{[c_j, y_i] : j \leq clusters_u\}$. 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 edge (x, y_i) where $x \in X$ is any vertex belonging to the set C_j. The source is s and the sink is t. We find a min-cost max flow f^* in G^* using Fredman and Tarjan's algorithm, and construct from this flow the assignment in G.

Lemma 4. *A min-cost max flow in f^* corresponds directly to a min-cost max flow in f.*

We denote by $D(u)$ the number of distinct trees in the forest $P^u_{u_1}, \ldots, P^u_{u_k}$.

Lemma 5. *The assignment between $u \in P$ and $v \in T$ can be computed in $O(d(u)\ (D(u)c(v) + c(v)\log c(v))\ )$ time.*

Time Complexity Analysis.

Algorithm 3 activates 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 the weighted assignment between a bipartite graph with $c(v) + d(u)$ nodes and $c(v) \times d(u)$ edges.

Therefore, based on Lemma 5, the total work spent on assignment computations is

$$O(\sum_{u=1}^{m}\sum_{v=1}^{n}(d(u)(clusters_u \times c(v)) + c(v)\log c(v))) \overset{\text{observation 1}}{=}$$

$$O(\sum_{u=1}^{m} d(u) \times clusters_u \times n + d(u) n \log n) \overset{observation\ 2}{=}$$
$$O(n \sum_{u=1}^{m} d(u)\ D(u) + mn \log n).$$

We will now turn to bound the summation $O(\sum_{u=1}^{m} d(u)\ D(u))$. We start with next lemma, which 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, $f(n) = O(n/\log n)$.*

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

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 Rooted Trees

A simplified version of the first algorithm, denoted Algorithm 4, 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 weighted matching problems on ordered bipartite graphs, where no two edges are allowed to cross. (We refer the reader to [26] 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 actually apply a rectangular case of the perfect assignment problem on $G = (X \cup Y, E)$, 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:** *Given pattern string X, a source string Y, and a character-to-character similarity table $\Delta[\Sigma_X, \Sigma_Y]$, find among all $|X|$-sized subsequences of Y the subsequence Q which is most similar to X, that is, the sum $\sum_{i=1}^{|X|} \Delta[Q_i, X_i]$ is maximized.*

Lemma 8. *The highest-scoring approximate episode occurrence of a pattern X of size k characters in a source string Y of size ℓ characters can be computed in $O(k \times \ell)$ time.*

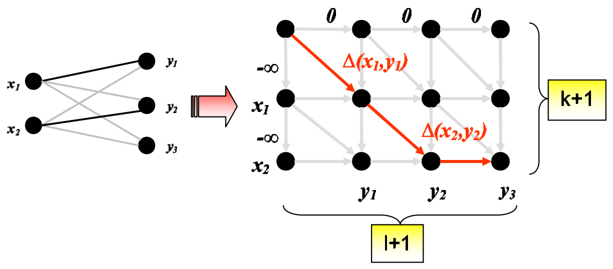

Fig. 5. Approximate Weighted Episode Matching Calculation.

Proof. This can be achieved by applying the classical dynamic programming string alignment algorithm to a ($k+1$ rows by $\ell+1$ columns) dynamic programming graph (see Figure 5). 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 $\Delta[i, j]$. □

Time Complexity Analysis.

Theorem 4. *ALSH of two rooted ordered trees can be computed in $O(mn)$ time.*

Proof. Algorithm 4 computes the assignment once for each node pair $(v \in T, u \in P)$. In this section we showed that, for rooted unordered trees, the dominant work of the assignment computation can be narrowed down to $O(c(v) \times c(u))$ time. Therefore, the total time complexity is:

$$\sum_{v=1}^{n}\sum_{u=1}^{m} O(c(v) \times c(u)) \overset{\text{observation 1}}{=} \sum_{v=1}^{n} O(m \times c(v)) \overset{\text{observation 1}}{=} O(m \times n). \quad \square$$

Acknowledgements.

We thank Seffi Naor and Gabriel Valiente for helpful discussions and comments.

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

5. 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.
6. 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.
7. 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.
8. H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1012–1036, 1989.
9. P. B. Gibbons, R. M. Karp, and D. Soroker. Subtree isomorphism is in random NC. *Discrete Applied Mathmatics*, 29:35–62, 1990.
10. 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.
11. L. R. Ford Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton, NJ: Princeton University Press, 1962.
12. 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.
13. M. Karpinski and A. Lingas. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.
14. P. Kilpelainen and H. Mannila. Ordered and unordered tree inclusion. *SIAM J. Comput.*, 24(2):340–356, 1995.
15. D. W. Matula. An algorithm for subtree identification. *SIAM Rev.*, 10:273–274, 1968.
16. D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978.
17. 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.
18. J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum cycle mean problems. *Math. Prog.*, 54:541–561, 1992.
19. R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, and M. Ziv-Ukelson. A new tool for the alignment of metabolic pathways. *Manuscript*, 2004.
20. S. W. Reyner. An analysis of a good algorithm for the subtree problems. *SIAM J. Comput.*, 6:730–732, 1977.
21. T. Schlieder and F. Naumann. *Approximate Tree Embedding for Querying XML Data.* ACM SIGIR Workshop on XML and IR, 2000.
22. 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.
23. R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.
24. M. A. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.
25. R. E. Tarjan. *Data Structures and Network Algorithms.* SIAM, Philadelphia, 1982.
26. 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.