Finding Largest Subtrees and Smallest Supertrees

Arvind Gupta[1] Naomi Nishimura[2]

July 6, 1995

[1]School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada, V5A 1S6. email: arvind@cs.sfu.ca, FAX (604) 291-3045. Research supported by the Natural Sciences and Engineering Research Council of Canada and the Advanced Systems Institute.

[2]Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1. email: nishi@plg.uwaterloo.ca, FAX (519) 885-1208. Research supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre.

Abstract

As trees are used in a wide variety of application areas, the comparison of trees arises in many guises. Here we consider two generalizations of classical tree pattern matching, which consists of determining if one tree is isomorphic to a subgraph of another. For the embedding problems of subgraph isomorphism and topological embedding, we present algorithms for determining the largest tree embeddable in two trees T and T' (or a *largest subtree*) and for constructing the smallest tree in which each of T and T' can be embedded (or a *smallest supertree*). Both subtrees and supertrees can be used in a variety of different applications. For example, when each of the two trees contains partial information about a data set, such as the evolution of a set of species, the subtree or supertree corresponds to a structuring of the data in a manner consistent with both original trees. The size of a subtree or supertree of two trees can also be used to measure the similarity between two arrangements of data, whether images, documents, or RNA secondary structures.

In this paper, we present a general paradigm for sequential and parallel subtree and supertree algorithms for subgraph isomorphism and topological embedding. Our sequential algorithms run in time $O(n^{2.5} \log n)$ and our parallel algorithms in time $O(\log^3 n)$ on a randomized CREW PRAM using a polynomial number of processors. In addition, we produce better algorithms for these problems when the underlying trees are ordered, that is, when the children of each node have a left-to-right ordering associated with them. In particular, we obtain $O(n^2)$ time sequential algorithms and $O(\log^3 n)$ time *deterministic* parallel algorithms on CREW PRAMs for both embeddings.

1 Introduction

Trees and their generalizations are among the most common and best studied of all combinatorial structures arising in computer science, due in large part to the number of areas of research in which they are applicable. For example, in data structure design, trees are the primary vehicle for storing data; many efficient data structures are tree-based. Trees have also been used in such diverse areas as compiler design [27, 1], structured text databases [18, 19], and the theory of natural languages [24, 8]. More recently, labeled trees have been used in phylogeny and molecular biology, where many of the underlying structures can be modeled with trees [7, 22, 31, 5, 6, 17]. Due to the large number of application areas, often the same or similar problems are studied under different terminology. Of particular interest are methods for combining or comparing the data associated with a pair of trees, as in the following three classes of problems, which can be viewed as tree, subtree (or subgraph), and supertree problems, listed in order of attention given by previous researchers to date.

Embeddable Tree Problem: Given two trees T and T', determine whether T is embeddable in T'.

Largest Common Embeddable Subtree Problem (LCES): Given two trees T and T', determine the largest tree L such that L is embeddable in both T and T'.

Smallest Common Embeddable Supertree Problem (SCES): Given two trees T and T', determine the smallest tree S such that both T and T' are embeddable in S.

Each class of problems can be considered with respect to embedding relations, such as subgraph isomorphism and topological embedding. For a particular embedding relation, a class of problems contains variants for various assumptions, for example whether the children of each node are unordered (unordered trees) or have a left-to-right ordering (ordered or planar-planted trees). It is our goal to unify a particular class of tree problems into a common framework and to present general sequential and parallel algorithms for problems in the framework.

Over the years, researchers have developed a systematic theory through the study of the algorithmic aspects of trees. Recently, there has been a renewed focus on their combinatorial aspects. In part this is due to the central role of trees in the work by Robertson and Seymour on graph minors [29, 30]. Their far-ranging work on general graphs begins with a treatment of trees as a base case. Variations of the embeddable tree problem arise in this work. As a problem on ordered trees, the Embeddable Tree Problem has been approached as pattern matching [21, 4, 20]; there has also been work done on the unordered tree case [25, 3, 23, 10, 13]. For many embedding relations, the *Embeddable Tree Problem* reduces to the *Largest Common Embeddable Tree Problem* since T is embeddable in T' if and only if T is the largest common embeddable tree.

Determining the largest common embeddable tree is a problem of interest in its own right, even when T is not embeddable in T'. Quantifying the similarity between two trees, or more generally between a given tree and each tree in a fixed set of templates, is a natural problem arising in many application areas; the size of the largest common embeddable tree is one such measure. In addition, identifying the largest common embeddable tree may make it possible to merge two or more trees representing slightly differing views of the same data set. Grossi [11] developed an algorithm for the restricted case in which leaves of L must map to leaves of T and T'. Previous work on the more general case has included the examination of the topological embedding problem on trees with distinct leaf labels, also known as the *Maximum Agreement Subtree Problem.* The problem has the following application: given two evolutionary trees derived using different methods, the largest subtree is a more robust evolutionary tree [7, 22, 31, 5, 6, 17]. Farach and Thorup [6] have an $O(n^{1.5} \log n)$ algorithm; Keselman and Amir [17] consider the problem when there are more than two input trees.

A supertree is of interest in the context of editing, image clustering, genetics, and chemical structure analysis, as it gives a measure of the similarity of trees [15], and in the context of computational biology, as it provides a method for forming an evolutionary tree [33]. In the case of the paper by Jiang, Wang, and Zhang, the problem solved is that of ordered minor containment (a generalization of ordered topological embedding); in the case of the paper by Warnow, the problem is that of minor containment in a setting where leaves have distinct labels and are constrained to map to other leaves.

For the case of unordered trees, our main result is an $O(n^{2.5} \log n)$ time sequential algorithm for finding subtrees or supertrees under either subgraph isomorphism or topological embedding. The parallel algorithms for the same problems run in $O(\log^3 n)$ time on a randomized CREW PRAM using $O(n^{7.5})$ processors. When the trees are ordered, we obtain $O(n^2)$ time sequential algorithms and $O(\log^3 n)$ time parallel algorithms with $O(n^8)$ processors. In this latter case, the parallel algorithms are deterministic instead of randomized.

After introducing notation and some basic results in Section 2, in Section 3 we compare the sequential algorithm for subgraph isomorphism on unordered trees to the technique used to solve the *Embeddable Tree Problem*, and outline general ways in which this can be extended. Next, in Section 4 we give the basic definitions and technical lemmas that are required for the development of algorithms for the subtree problem. We present a sequential algorithm for subtrees under topological embedding in Section 5 and a parallel algorithm for the same problem in Section 6. Sequential and parallel algorithms for topological embedding for supertrees are considered in Section 7. Variations needed for both problems on ordered trees are considered in Section 8, with the modifications needed for solving the subtree and supertree problems for subgraph isomorphism outlined in Section 9. Finally, in Section 10 we present some directions for further research.

2 Preliminaries

2.1 Trees

For all the problems discussed in this paper, we consider trees (graphs with no cycles) that are finite and have one distinguished node, the root. We use the notation $V(T)$, $E(T)$, and $r(T)$ to denote the node set, edge set, and root, respectively, of a tree T. The size of T, $|V(T)|$, is denoted by $|T|$. For u and v nodes of T, we denote the path from u to v by $\pi_T(u, v)$; subscripts are omitted when the tree is clear from context.

In our algorithms, we will refer to several types of trees other than the original input trees. One such tree, the *Brent tree* (defined in Section 2.3), is used in our parallel algorithms. For clarity, we will refer to *nodes* of a tree T using the Roman alphabet and *vertices* of a Brent tree using the Greek alphabet. We associate with each vertex in a Brent tree a *level number*, where the root of the tree is at level 1 and the child of a vertex at level t is at level $t + 1$.

In processing a tree T, we will distinguish between an arbitrary connected *subgraph* of T, and a *subtree* of T consisting of a node in $V(T)$ and all its descendants, where T_v denotes the subtree of T rooted at v. In the course of our algorithms, we will often be concerned with subgraphs that arise from removing one subtree from another. For S a subtree of T and $v \in V(S)$, $S \setminus^{+} S_v$ denotes the subgraph obtained by removing from S all proper descendants of v. In particular, note that the node $v \notin S \setminus S_v$ but that $v \in S \setminus^{+} S_v$. We say that $S \setminus^{+} S_v$ is a *scarred subtree* of T, where v is a *scar*, and that $S \setminus^{+} S_v$ is *scarred at* v.

The generally accepted definition of trees does not place an ordering on the children. We will also be working with trees in which a left-to-right ordering is placed on the children of each node; we call such trees *ordered trees*; these are also known as *planar-planted trees* in the literature. When it is not clear from context, we will use the term *unordered trees* to mean trees in which there is no ordering on the children.

2.2 Embedding problems

In this section we present formal definitions of the embedding problems considered in this paper. Our definitions are in terms of unordered trees; the definitions for ordered trees can be derived by adding the further restriction that the mappings preserve the ordering on children of each node.

The problem of subgraph isomorphism is that of determining if one tree is a subgraph of another. A function ψ is a *subgraph isomorphism embedding* from a tree T to a tree T' by which we mean $\psi : V(T) \to V(T')$ is a one-to-one function such that $(a, b) \in E(T)$ if and only if $(\psi(a), \psi(b)) \in E(T')$. We say that ψ is a *root-to-root subgraph isomorphism embedding* if, in addition, $\psi(r(T)) = r(T')$.

Topological embedding can be seen as a generalization of subgraph isomorphism, as follows:

Definition: A tree T is *topologically embeddable* in a tree T', $T \leq_e T'$, if there is a one-to-one function $\psi : V(T) \to V(T')$ such that for any $a, b, c \in V(T)$ the following properties hold. If b is a child of a, then $\psi(b)$ is a descendant of $\psi(a)$. If b and c are distinct children of a, then the path from $\psi(a)$ to $\psi(b)$ and the path from $\psi(a)$ to $\psi(c)$ have exactly the node $\psi(a)$ in common. Equivalently, we will say that there is a *topological embedding* of T in T'. The topological embedding is a *root-to-root topological embedding* if $\psi(r(T)) = r(T')$.

Intuitively, $T \leq_e T'$ if we can map each node in T to a node in T' such that the edges in T map to node-disjoint paths in T'. For convenience, we may consider function ψ to map edges in T to paths in T', and describe an edge in the path $\psi(e)$ as an edge in the image of e. More generally, we extend the definition of ψ to paths of T. For $P = v_1, v_2, \ldots, v_k$ a path in T, we define $\psi(P)$ to be the concatenation of the paths $\psi((v_1, v_2)), \psi((v_2, v_3)), \ldots, \psi((v_{k-1}, v_k))$. Since there is a unique path between every pair of nodes in a tree, the following lemma is straightforward to prove.

Lemma 2.1. *For ψ a topological embedding of T into T', and for any pair of nodes u, v of T, $\psi(\pi_T(u, v))$ is a path in T'.*

Throughout the remainder of the paper, we distinguish between topological embedding, as defined above, and embedding, referring to the class of problems containing subgraph isomorphism and topological embedding. In addition, we will use T and T' to denote input trees, λ and λ' to denote mappings from a largest common subtree (denoted L) to T and T', respectively, and σ and σ' to denote mappings from T and T' to a smallest common supertree (denoted S).

2.3 Brent restructuring

To solve our tree problems in parallel, we apply results of Brent [2] to divide a tree into a number of subgraphs, each a fixed fraction smaller than the original tree. We can obtain a recursive solution by solving the problem on the subgraphs, with the depth of recursion at most $O(\log n)$.

The method used by Brent in performing the division forms two different types of subgraphs of the original tree T, namely unscarred and scarred subtrees of T. The lemmas below, slight generalizations of results of Brent, contain the essential components of the division for the first case (Lemma 2.2) and the second case (Lemma 2.3). Proofs of these lemmas can be found in an earlier paper [13].

Lemma 2.2. *For T a tree with at least two nodes, there is a unique node v of T with children $c_0, \ldots, c_{k-1}$ such that:*

1. $|T \dot{\setminus} T_v| \leq \frac{|T|}{2}$, *(or equivalently $|T_v| > \frac{|T|}{2}$); and*
2. $|T_{c_i}| \leq \frac{|T|}{2}$, *for all $0 \leq i \leq k$.*

Lemma 2.3. *For T a tree with more than two nodes and ℓ a leaf of T, there is a unique ancestor v of ℓ such that if c is the child of v for which $\ell \in T_c$ then:*

1. $|T \dot{\setminus} T_v| \leq \frac{|T|}{2}$, *(or equivalently $|T_v| > \frac{|T|}{2}$); and*
2. $|T_c| \leq \frac{|T|}{2}$.

We obtain a division of the tree into subgraphs by starting with a tree T and recursively applying the two lemmas depending on whether or not a subgraph has a scar. In practice, we will view the applications of both Lemma 2.2 and Lemma 2.3 as two-step operations: first T is split into subgraphs $T \dot{\setminus} T_v$ and T_v (a *Brent break*) and then T_v is split into subtrees $T_{c_0}, \ldots, T_{c_{k-1}}$ where $c_0, \ldots, c_{k-1}$ are the children of v (a *child break*). By applying a Brent break and then a child break to a graph (also known as *Brent restructuring*), we obtain subgraphs containing disjoint sets of the nodes. After $O(\log n)$ recursive applications of the lemmas, the resulting subgraphs will be of constant size.

In our parallel algorithms, we store all subgraphs arising from the Brent restructurings in a representation tree. For the tree T, this tree, denoted $\mathcal{B}_T$, is called the *Brent tree* of T. In $\mathcal{B}_T$, a vertex will correspond to a subgraph of the original tree and an edge between two vertices will indicate that the child is derived from the parent by a restructuring step. We can form $\mathcal{B}_T$ in $O(\log |T|)$ time using an $O(|T|^4)$-processor CREW PRAM [13]. The reader is referred to the bibliography for various papers in which this technique is applied [14, 12, 13] and to one paper in particular for a detailed discussion of its use [13].

3 The basic techniques

Our results are based on a dynamic programming technique first employed in a sequential algorithm of Matula and Reyner [25, 28, 32] for determining if one tree is a subgraph of another. In this section, we begin by outlining Matula's technique and then describe how it can be combined with Brent restructuring to yield a parallel algorithm. We then discuss modifications needed to handle topological embedding. Details of all these algorithms, with their complexities, can be found in earlier work [13]. Finally, we give an outline of the structure of the subtree and supertree algorithms.

3.1 Subgraph isomorphism on trees

To determine whether or not a tree T is isomorphic to a subgraph of a tree T', Matula's approach is to work up from the leaves of T', in turn labeling each node $u \in V(T')$ with the set of nodes $a \in V(T)$ such that each T_a is a subgraph of T'_u. For a particular a and u, we can determine whether or not T_a is a subgraph of T'_u by making use of such information about children of a and

u: we must determine whether or not the subgraph rooted at each child of a can be embedded in the subgraph rooted at a distinct child of u. This problem is equivalent to solving a matching problem on a bipartite graph $G(X, Y, E)$, where X corresponds to the children of a, Y corresponds to the children of u, and there is an edge in E from child b of a to child v of u if and only if T_b is a subgraph of T'_v. The matching must include every node in X, since this implies that each child of a is assigned to a distinct child of u. Since bipartite matching can be solved in time $O(n^{2.5})$, this step will dominate the complexity; it is not difficult to show that the total running time is in $O(n^{2.5})$.

3.2 Parallelizing Matula's algorithm

To obtain parallel algorithms, the Brent tree of T, $\mathcal{B}_T$, is used to decompose the original problem into subproblems. We process $\mathcal{B}_T$ level by level from the bottom up; when a level has been completely processed, we will have determined all the possible locations in T' where we can embed those subgraphs of T that correspond to vertices at that level in $\mathcal{B}_T$. Since each level of $\mathcal{B}_T$ corresponding to child breaks partitions the nodes of T, the subgraphs to be embedded are disjoint; processing a level constitutes solving, in parallel, a number of independent problems, one associated with each vertex of $\mathcal{B}_T$.

To determine the time complexity of such an algorithm, we consider the time to process one level of the Brent tree and multiply it by the number of levels. The processing is dominated by the cost of bipartite matching. Since bipartite matching can be accomplished in time $O(\log^2 n)$ on a randomized CREW PRAM and since the Brent tree has height in $O(\log n)$, the total running time for the algorithm is in $O(\log^3 n)$. The total number of processors required can be shown to be in $O(n^{6.5})$.

3.3 Topological embedding

To solve the same problems when the underlying relation is topological embedding, we use a similar approach. In particular, we once again label each node u in T' by a set of nodes a in T; in this case, a is in the set if T_a is root-to-root topologically embeddable in T'_u. Since topological embedding maps edges to paths, for each child b of a and v of u, we must know whether T_b is topologically embeddable in T'_v (it does not suffice to know whether or not T_b is root-to-root topologically embeddable in T'_v). As in the algorithm for subgraph isomorphism, such a determination can be made by setting up and solving a matching problem. Since edges are mapped to paths, once we have information about a and u, we label each ancestor w of u by the information that T_a is topologically embeddable in T'_w. This idea of propagating information to ancestors is also used in the parallel algorithm.

3.4 Subtree and supertree problems

Each of the algorithms discussed in this paper consists of determining for each pair of nodes a in T and u in T' either the largest common subtree or the smallest common supertree of T_a and T'_u. As in Matula's original algorithm, such information is obtained by making use of a dynamic programming approach; in essence, information concerning children is used to determine information about parents.

Unlike in the *Embeddable Tree Problem*, in the LCES and SCES problems it does not matter which of the input graphs is designated as T and which as T'. As a consequence of this symmetry, at times it will be necessary to compare information about a and children of u, about u and children of a, and about children of a and children of u. To enable such processing to proceed by dynamic programming, we must ensure at all times that all necessary quantities have been or can be computed. Accordingly, each algorithm description in the paper will indicate how quantities can be calculated as well as the set of quantities needed.

Although ideally T and T' should be treated equally, in the case of the parallel algorithms, asymmetry may arise from applying Brent breaks to one tree and not the other. Here it will be important to show that despite this asymmetry, it is possible to obtain all necessary intermediate results for each calculation.

4 The LCES problem - preliminaries

In the next two sections we describe sequential and parallel algorithms for the largest common embeddable subtree problem (LCES) for topological embedding. Algorithms for finding subtrees for subgraph isomorphism will be outlined in Section 9. In this section we give some basic background and technical lemmas that are needed in all these algorithms.

For input trees T and T', our algorithms will actually compute a largest common subtree of the trees T_a and T'_u for all nodes $a \in V(T)$ and $u \in V(T')$. We denote the set of largest common subtrees of T_a and T'_u under subgraph isomorphism by $LCS_i(a,u)$ and under topological embedding by $LCS_e(a,u)$. In the algorithms, it suffices to compute the size of the elements in each of these sets; the output subtree can be recovered from the size information. We use $\mathcal{L}_i(a,u)$ to denote the size of the subtrees in $LCS_i(a,u)$ and $\mathcal{L}_e(a,u)$ to denote the size of the subtrees in $LCS_e(a,u)$. When it is clear from context, we will omit the subscripts i and e.

Lemma 4.1. *For any $L \in LCS(r(T), r(T'))$ (under either subgraph isomorphism or topological embedding) and for every pair of embeddings λ and λ' of L into T and T', either $\lambda(r(L)) = r(T)$ or $\lambda'(r(L)) = r(T')$ (or both).*

Proof. Let $L \in LCS(r(T), r(T'))$ and suppose there are embeddings λ and λ' of L into T and T' such that $\lambda(r(L)) \neq r(T)$ and $\lambda'(r(L)) \neq r(T')$. We then form L' from L by adding a new node g as the parent of $r(L)$. Now we can easily extend λ and λ' to form embeddings of L' into both T and T' by mapping g to the parents of $\lambda(r(T))$ and $\lambda'(r(T'))$, thereby contradicting the maximality of L. ■

The proof of the following lemma is omitted; it can easily be derived from the definition of the largest common subtree and the fact that subgraph isomorphism and topological embedding are transitive relations.

Lemma 4.2. *For any node u in $V(T')$, $\mathcal{L}(r(T), u) \leq \mathcal{L}(r(T), r(T'))$.*

We rely extensively on solving weighted bipartite matching problems for our algorithms; these problems are slightly different for topological embedding and subgraph isomorphism. Here we give the problem for topological embedding; the problem for subgraph isomorphism is given in Section 9.

For $b_1, \ldots, b_k$ nodes of T and $v_1, \ldots, v_\ell$ nodes of T', $MaxWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\})$ is the maximum weight bipartite matching in the graph $G(X, Y, E)$ defined by $X = \{b_1, \ldots, b_k\}$, $Y = \{v_1, \ldots, v_\ell\}$, and E consisting of an edge of weight $\mathcal{L}(b_i, v_j)$ between each pair (b_i, v_j).

Lemma 4.3. *Let $X = \{b_1, \ldots, b_k\}$ and $Y = \{v_1, \ldots, v_\ell\}$ be the children of $r(T)$ and $r(T')$, respectively. If there is a matching $\mathcal{M}$ of weight $W_{\mathcal{M}}$ in $G(X, Y, E)$, then there is a tree L of size at least $W_{\mathcal{M}} + 1$ such that L root-to-root topologically embeds in T and T'. Conversely, if there is a tree L of size W that root-to-root topologically embeds in T and T', then there is a matching of weight at least $W - 1$ in $G(X, Y, E)$.*

Proof. For $\mathcal{M}$ a matching in $G(X,Y,E)$ of weight $W_{\mathcal{M}}$, we can relabel the b_i's and v_i's such that $\mathcal{M} = \{(b_1,v_1),\ldots,(b_j,v_j)\}$. Then for $1 \le i \le j$, there is a tree L_i of size $\mathcal{L}(b_i,v_i)$ such that L_i embeds in T_{b_i} and T'_{v_i}. We define L to have a root with j children, where the subtree of L rooted at the ith child is L_i. It is straightforward to verify that L root-to-root embeds in T and T' (under either subgraph isomorphism or topological embedding).

To prove the converse, let L be a tree of size W that root-to-root embeds in T and T' under λ and λ'. Let $L_1,\ldots,L_j$ be the trees rooted at the children of $r(L)$. Without loss of generality (by relabeling the b_i's and v_i's) we can assume that $\lambda(r(L_i))$ is a node in T_{b_i} and $\lambda'(r(L_i))$ is a node in T'_{v_i}, for $1 \le i \le j$. Clearly, for each i, the size of L_i is at most $\mathcal{L}(b_i,v_i)$. Therefore, $\{(b_1,v_1),\ldots,(b_j,v_j)\}$ is a matching in $G(X,Y,E)$ of weight at least $W-1$. ∎

Corollary 4.4. *For X and Y as in Lemma 4.3, $MaxWM(X,Y,E)+1$ is the size of the largest tree that is root-to-root topologically embeddable in both T and T'.*

In our parallel algorithms, it is necessary to extend the definitions of LCS and $\mathcal{L}$ to handle scarred trees. We give the definitions here without specifying the embedding used, as they are the same for both subgraph isomorphism and topological embedding.

Definition: For s a descendant of $a \in V(T)$ and y a descendant of $u \in V(T')$, $LCS(a{\setminus}^{\dagger}s, u{\setminus}^{\dagger}y)$ is the set of trees L such that

1. there are embeddings λ and λ' from L to $T_a{\setminus}^{\dagger}T_s$ and $T'_u{\setminus}^{\dagger}T'_y$ and a distinguished node d of L such that $\lambda(d) = s$ and $\lambda'(d) = y$; and

2. no tree larger than L satisfies condition 1.

Furthermore, we define $LCS(a, u{\setminus}^{\dagger}y)$ to be the set of largest common subtrees of T_a and T'', where T'' is the tree $T'_u \backslash T'_y$ (i.e. the tree $T'_u{\setminus}^{\dagger}T'_y$ with the node y removed). Therefore, for every $L \in LCS(a, u{\setminus}^{\dagger}y)$ there is an embedding λ' from L to $T'_u{\setminus}^{\dagger}T'_y$ such that $\lambda'(d) \neq y$ for every node d of L. As in our previous definitions, we define $\mathcal{L}(a{\setminus}^{\dagger}s, u{\setminus}^{\dagger}y)$ to be the size of the trees in $LCS(a{\setminus}^{\dagger}s, u{\setminus}^{\dagger}y)$ and $\mathcal{L}(a, u{\setminus}^{\dagger}y)$ to be the size of the trees in $LCS(a, u{\setminus}^{\dagger}y)$.

Our algorithms will be based on computing the quantities $\mathcal{L}(a, u{\setminus}^{\dagger}y)$ and $\mathcal{L}(a{\setminus}^{\dagger}s, u{\setminus}^{\dagger}y)$.

5 The LCES problem - sequential topological embedding

In this section we describe a sequential topological embedding algorithm for determining $LCS_e(a,u)$ for any pair of nodes $a \in V(T)$ and $u \in V(T')$. We use a dynamic programming approach whereby $LCS_e(a,u)$ is determined from sets computed for descendants of a and u. We will show that the largest common subtree for a and u can be found by computing the maximum of three quantities, each associated with a different condition suggested by Lemma 4.1.

Lemma 5.1. *For any $a \in V(T)$ with children $b_1,\ldots,b_k$ and any $u \in V(T')$ with children $v_1,\ldots,v_\ell$, one of the following three conditions must hold for every $L \in LCS(a,u)$:*

1. *$L \in LCS(a,v_p)$ for some p, $1 \le p \le \ell$;*

2. *$L \in LCS(b_q,u)$ for some q, $1 \le q \le k$; or*

3. *there are topological embeddings λ and λ' of L into T and T' such that:*

(a) $\lambda(r(L)) = a$, $\lambda'(r(L)) = u$;

(b) for each child g *of* $r(L)$ *there is a distinct child* $b(g)$ *of* a *such that* $\lambda(g)$ *is a descendant of* $b(g)$;

(c) for each child g *of* $r(L)$ *there is a distinct child* $v(g)$ *of* u *such that* $\lambda'(g)$ *is a descendant of* $v(g)$;

(d) the subtree of L *rooted at* g *is in* $LCS(b(g), v(g))$; *and*

(e) there is no other tree L' *bigger than* L *and topological embeddings of* L' *into* T *and* T' *such that conditions (a)-(d) are satisfied.*

Proof. As a consequence of Lemma 4.1, we can characterize each L in the set $LCS_e(a, u)$ by observing that for any pair of embeddings τ and τ' from such an L into T and T', one of the following occurs:

1. $\tau(r(L)) = a$ and $\tau'(r(L)) = w$ for w a proper descendant of u;
2. $\tau'(r(L)) = u$ and $\tau(r(L)) = c$ for c a proper descendant of a; or
3. $\tau(r(L)) = a$ and $\tau'(r(L)) = u$.

We will prove the lemma by considering each of these possible conditions in turn.

First suppose that $\tau(r(L)) = a$, $\tau'(r(L)) = w$ for w a descendant of u, and v_p is the child of u such that $w \in T'_{v_p}$. Since $L \leq_e T_a$ and $L \leq_e T'_w \leq_e T'_{v_p}$, we can conclude that $|L| \leq \mathcal{L}(a, v_p)$. Moreover, by Lemma 4.2, we know that $\mathcal{L}(a, v_p) \leq |L|$. Thus $L \in LCS(a, v_p)$, satisfying condition 1. The case in which $\tau(r(L))$ is a descendant of a and $\tau'(r(L)) = u$ is similar.

Finally, we consider the case in which $\tau(r(L)) = a$ and $\tau'(r(L)) = u$. By the definition of topological embedding, each child g of $r(L)$ must be mapped by τ to the subtree rooted at a distinct child $b(g)$ of a and by τ' to the subtree rooted at a distinct child $v(g)$ of u. We can then conclude that $L_g \in LCS(b(g), v(g))$ by noting that $L_g \leq_e T_{b(g)}$, $L_g \leq_e T'_{v(g)}$ and if there were a larger tree L' such that $L' \leq_e T_{b(g)}$ and $L' \leq_e T'_{v(g)}$, it would be possible to make L larger by substituting L' for L_g, yielding a contradiction. Thus, conditions 3(a)- 3(d) are satisfied. To see that condition 3(e) holds, notice that if a larger L' with appropriate topological embeddings existed, then L would not be a largest subtree, contradicting the assumption that L is in $LCS(a, u)$. ■

Notice that when $r(L)$ maps to both a and u, we are matching some children of a with some children of u in such a way that we maximize the sum of the sizes of the largest common subtrees of the matched children.

In our algorithm, it will be the size of the largest common subtrees that will be used to work our way up T and T'. The lemma below follows from Lemma 5.1; it suggests the structure of the algorithm itself.

Lemma 5.2. *For* a, u, $b_1, \ldots, b_k$, *and* $v_1, \ldots, v_\ell$ *as in Lemma 5.1, we define the following three quantities:*

$$\begin{aligned} M_1 &= \max\{\mathcal{L}(a, v_i) \mid 1 \leq i \leq \ell\} \\ M_2 &= \max\{\mathcal{L}(b_j, u) \mid 1 \leq j \leq k\}; \text{ and} \\ M_3 &= MaxWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\}) + 1. \end{aligned}$$

Then, $\mathcal{L}(a, u) = \max\{M_1, M_2, M_3\}$.

Proof. As before, let $L \in LCS(a,u)$ and let λ and λ' be the topological embeddings of L into T_a and T'_u. Once again we consider the three possible ways in which the root of L can be embedded.

If $\lambda(r(L)) = a$ and $\lambda'(r(L)) = w$ for w a descendant of u, $\mathcal{L}(a,u) = \mathcal{L}(a,v_p)$ for $w \in T'_{v_p}$ and therefore $\mathcal{L}(a,u) = M_1$. Similarly, if $\lambda'(r(L)) = u$ and $\lambda(r(L)) = c$ for c a descendant of a, then $\mathcal{L}(a,u) = M_2$. Finally if $\lambda(r(L)) = a$ and $\lambda'(r(L)) = u$ the lemma holds by Corollary 4.4. ∎

It is now clear how the algorithm can be structured. For every node a of T proceeding from leaves to root and for every node u of T' proceeding from leaves to root, $\mathcal{L}(a,u)$ can be computed as follows: if a or u is a leaf, $\mathcal{L}(a,u)$ is set to 1; otherwise, we can recursively compute M_1, M_2, and M_3 and then apply Lemma 5.2 to determine $\mathcal{L}(a,u)$.

The above algorithm can easily be modified to compute the largest common subtree by keeping track of both the size and the subtree at every step.

Theorem 5.3. *For trees T and T' of size $O(n)$, $\mathcal{L}(T,T')$ and $LCS(T,T')$ can be computed in time $O(n^{2.5}\log n)$.*

Proof. The proof of correctness follows from the preceding discussion. It is not difficult to see that the computation of $\mathcal{L}(a,u)$ is dominated by the cost of determining M_3, as the cost of determining M_1 is at most ℓ and that of M_2 at most k. A bipartite weighted matching on an m node graph with weights in $O(n)$ can be computed in time $O(m^{2.5}\log n)$ [9]. Thus, summing over all values of a and u, the total complexity of the algorithm is

$$\begin{aligned} &O\Big(\sum_{a\in V(T)}\sum_{u\in V(T')}(|\text{children of } a| + |\text{children of } u|)^{2.5}\log n\Big)\\ &\quad\subseteq O\Big(\Big(\sum_{a\in V(T)}|\text{children of } a| + \sum_{u\in V(T')}|\text{children of } u|\Big)^{2.5}\log n\Big)\\ &\quad\subseteq O(n^{2.5}\log n). \end{aligned}$$

as claimed. ∎

6 The LCES problem - parallel topological embedding

As a starting point for discussing the parallel largest common embeddable subtree algorithm, we note that a naive parallelization of the sequential algorithm described in Section 5 would be superquadratic as both T and T' could have depth $O(n)$. To reduce the running time, we restructure T' using Brent restructuring to form the tree $\mathcal{B}_{T'}$ of depth $O(\log n)$. We then work up level by level in $\mathcal{B}_{T'}$ from the leaves to the root. Suppose a vertex α of $\mathcal{B}_{T'}$ is labeled by the subgraph $T'_u \backslash\!\!\!+ T'_y$. We compute the largest common subtree of $T'_u \backslash\!\!\!+ T'_y$ and every subtree $T_a \backslash\!\!\!+ T_s$ of T. Similarly, if α is labeled by the subtree T'_u, we compute the largest common subtree of T'_u and T_a for every subtree T_a of T'_u.

For the remainder of this section, we only consider the case in which the label of α is a scarred tree; the unscarred case is very similar.

6.1 Case 1: α occurs at a Brent break

In this section, we show how the $\mathcal{L}$ values for α can be computed using previously computed $\mathcal{L}$ values. For a Brent break node α labeled $T'_u \backslash\!\!\!+ T'_y$ in $\mathcal{B}_{T'}$, we can assume that the children of α are labeled by $T'_u \backslash\!\!\!+ T'_x$ and $T'_x \backslash\!\!\!+ T'_y$, where $x \in \pi(u,y)$. Recall that our algorithm works up $\mathcal{B}_{T'}$ from leaves to root, computing $\mathcal{L}$ values for the label of each vertex in $\mathcal{B}_{T'}$ and for all possible scarred and unscarred trees in T. In particular, when we reach α, we will have computed, for all $a, s \in V(T)$ and $t \in \pi(a, s)$, the quantities

1. $\mathcal{L}(a\setminus^{+}t, u\setminus^{+}x)$;
2. $\mathcal{L}(t\setminus^{+}s, x\setminus^{+}y)$;
3. $\mathcal{L}(a, u\setminus^{+}x)$; and
4. $\mathcal{L}(a, x\setminus^{+}y)$.

In the course of the algorithm, we must compute $\mathcal{L}(a\setminus^{+}s, u\setminus^{+}y)$ for every pair of nodes a and s of T with a an ancestor of s and compute $\mathcal{L}(a, u\setminus^{+}y)$ for every node a of T.

Lemma 6.1. *For a, s, u, x, and y defined as above,*

$$\mathcal{L}(a\setminus^{+}s, u\setminus^{+}y) = \max_{t\in\pi(a,s)}\{\mathcal{L}(a\setminus^{+}t, u\setminus^{+}x) + \mathcal{L}(t\setminus^{+}s, x\setminus^{+}y) - 1\}.$$

Proof. We prove the lemma by finding a node t on $\pi(a, s)$ such that $\mathcal{L}(a\setminus^{+}s, u\setminus^{+}y) = \mathcal{L}(a\setminus^{+}t, u\setminus^{+}x) + \mathcal{L}(t\setminus^{+}s, x\setminus^{+}y) - 1$. Let $L \in LCS(a\setminus^{+}s, u\setminus^{+}y)$ such that λ and λ' are topological embeddings of L into $T_a\setminus^{+}T_s$ and $T'_u\setminus^{+}T'_y$ respectively, and let $d \in V(L)$ such that $\lambda(d) = s$ and $\lambda'(d) = y$. Then, $\lambda(\pi(r(L), d))$ is a subpath of $\pi(a, s)$ and $\lambda'(\pi(r(L), d))$ is a subpath of $\pi(u, y)$. Since $x \in \pi(u, y)$ and $L \leq_e T'_u\setminus^{+}T'_y$, one of the following cases must hold:

1. x is $\lambda'(h)$ for some node h in L;
2. $\lambda'(r(L))$ is a proper descendant of x; or
3. there exists an edge (g, h) in $E(L)$ such that $x \in \pi(v, w)$ for $\lambda'((g, h)) = \pi(v, w)$.

We consider each of these cases in turn.
Case 1:
If x is $\lambda'(h)$ for some node h of L, we let $t = \lambda(h)$. It is not difficult to see that $|L_h| = \mathcal{L}(t\setminus^{+}s, x\setminus^{+}y)$, since L_h is clearly a common subgraph of $T_t\setminus^{+}T_s$ and $T'_x\setminus^{+}T'_y$: if there were a larger common subgraph L', then the graph formed by replacing L_h by L' in L would contradict the assumption of the maximality of L. By a similar argument we can show that $|L\setminus^{+}L_h| = \mathcal{L}(a\setminus^{+}t, u\setminus^{+}x)$. Since h is counted twice when the two quantities are added together, we subtract one to obtain the value given in the statement of the lemma.
Case 2:
If $\lambda'(r(L))$ is a proper descendant of x then we define t to be a. Then clearly $\mathcal{L}(a\setminus^{+}t, u\setminus^{+}x) = 1$, $\mathcal{L}(t\setminus^{+}s, x\setminus^{+}y) = |L|$, and the result follows.
Case 3:
It will suffice to show that for $t = \lambda(h)$, the lemma holds, that is, $|L\setminus^{+}L_h| = \mathcal{L}(a\setminus^{+}t, u\setminus^{+}x)$ and $|L_h| = \mathcal{L}(t\setminus^{+}s, x\setminus^{+}y)$. If $h = d$, the argument is similar to that given in Case 1; we now assume that $h \neq d$.

Suppose instead that $|L\setminus^{+}L_h| < \mathcal{L}(a\setminus^{+}t, u\setminus^{+}x)$. Then, there is a tree L' bigger than $L\setminus^{+}L_h$ such that $L' \leq_e T_a\setminus^{+}T_t$ and $L' \leq_e T'_u\setminus^{+}T'_x$, with topological embeddings τ and τ' respectively such that there is a node d' of L' with $\tau(d') = t$ and $\tau'(d') = x$. But then we can form the tree L'' from L' and L_h by identifying d' and $r(L_h)$; L'' is bigger than L and clearly $L'' \leq_e T_a\setminus^{+}T_s$ and $L'' \leq_e T'_u\setminus^{+}T'_y$.

Similarly, if $|L_h| < \mathcal{L}(t\setminus^{+} s, x\setminus^{+} y)$, let L' be a tree bigger than L_h such that $L' \leq_e T_t\setminus^{+} T_s$ and $L' \leq_e T'_x\setminus^{+} T'_y$. But then the tree L'' formed by identifying h in $L\setminus^{+} L_h$ with $r(L')$ is bigger than L and embeds in both $T_a\setminus^{+} T_s$ and $T'_u\setminus^{+} T'_y$, a contradiction. ■

Lemma 6.2. *For a, u, x, and y defined as above,*

$$\mathcal{L}(a, u\setminus^{+} y) = \max\{\mathcal{L}(a, u\setminus^{+} x), \max_{t\in V(T_a)} \{\mathcal{L}(a\setminus^{+} t, u\setminus^{+} x) + \mathcal{L}(t, x\setminus^{+} y)\} - 1\}.$$

Proof. If $L \in LCS(a, u\setminus^{+} y)$, then there are topological embeddings λ and λ' from L to T_a and T'_u respectively such that λ' maps no node of L to y. If for every node $h \in V(L)$, $\lambda'(h) \notin V(T'_x)$, then clearly $L \in LCS(a, u\setminus^{+} x)$ and the lemma holds.

Now, suppose that there is a node $h \in V(L)$ such that $\lambda'(h) \in V(T'_x)$. We must consider two cases depending on whether or not λ' maps some node of L to x; these cases are similar to those in the proof of Lemma 6.1. ■

6.2 Case 2: α occurs at a child break

We now consider a child break node α in $\mathcal{B}_{T'}$. Let the label of α be a tree $T'_u\setminus^{+} T'_y$ where u has children $v_1, \ldots, v_\ell$; assume the scar y is a descendant of v_p, $1 \leq p \leq \ell$. Let the children of α be $\beta_1, \ldots, \beta_\ell$ where β_i is labeled by the tree T'_{v_i} when $i \neq p$ and by the tree $T'_{v_p}\setminus^{+} T'_y$ when $i = p$.

As in Case 1, we can conclude that certain values have been computed when we reach α in $\mathcal{B}_{T'}$. In particular, for every pair of nodes a, s in T with $b_1, \ldots, b_k$ children of a and with s a descendant of b_q, we will have computed the following quantities:

1. $\mathcal{L}(a, v_j)$, $1 \leq j \leq \ell, j \neq p$;
2. $\mathcal{L}(a, v_p\setminus^{+} y)$;
3. $\mathcal{L}(a\setminus^{+} s, v_p\setminus^{+} y)$;
4. $\mathcal{L}(b_i, u\setminus^{+} y), 1 \leq i \leq k, i \neq q$;
5. $\mathcal{L}(b_q\setminus^{+} s, u\setminus^{+} y)$;
6. $\mathcal{L}(b_i, v_j)$, $1 \leq i \leq k, 1 \leq j \leq \ell, i \neq q, j \neq p$;
7. $\mathcal{L}(b_i, v_p\setminus^{+} y)$, $1 \leq i \leq k, i \neq q$; and
8. $\mathcal{L}(b_q\setminus^{+} s, v_p\setminus^{+} y)$.

As in the Brent break case, we must compute $\mathcal{L}(a\setminus^{+} s, u\setminus^{+} y)$ and $\mathcal{L}(a, u\setminus^{+} y)$ for $a, s \in V(T)$ with a an ancestor of s.

Lemma 6.3. *For $a, s, u, y, b_1, \ldots, b_k, v_1, \ldots, v_\ell, p$, and q as above, let*

$$\begin{aligned} M_1 &= \mathcal{L}(a\setminus^{+} s, v_p\setminus^{+} y); \\ M_2 &= \mathcal{L}(b_q\setminus^{+} s, u\setminus^{+} y); \text{ and} \\ M_3 &= MaxWM(\{b_1, \ldots, b_k\}\setminus\{b_q\}, \{v_1, \ldots, v_\ell\}\setminus\{v_p\}) + \mathcal{L}(b_q\setminus^{+} s, v_p\setminus^{+} y) + 1. \end{aligned}$$

Then, $\mathcal{L}(a\setminus^{+} s, u\setminus^{+} y) = \max\{M_1, M_2, M_3\}$.

Proof. Let $L \in LCS(a\backslash^{\dagger} s, u\backslash^{\dagger} y)$ and let λ and λ' be topological embeddings of L into $T_a\backslash^{\dagger} T_s$ and $T'_u\backslash^{\dagger} T'_y$ respectively such that there exists a node $d \in V(L)$ for which $\lambda(d) = s$ and $\lambda'(d) = y$. By a generalization of Lemma 4.1 to include scars, we can conclude that either $\lambda(r(L)) = a$ or $\lambda'(r(L)) = u$ (or both); we consider each case in turn.

Suppose $\lambda(r(L)) = a$ but $\lambda'(r(L)) \neq u$. Since topological embeddings preserve descendant relationships, and y is in the image of the embedding, $\lambda'(r(L))$ is a node (say w) in $T'_{v_p}\backslash^{\dagger} T'_y$. Clearly $L \in LCS(a\backslash^{\dagger} s, w\backslash^{\dagger} y)$ and by a generalization of Lemma 4.2 to scars, it can be shown that $\mathcal{L}(a\backslash^{\dagger} s, u\backslash^{\dagger} y) = M_1$. The case when $\lambda(r(L)) \neq a$ but $\lambda'(r(L)) = u$ is similar, yielding $\mathcal{L}(a\backslash^{\dagger} s, u\backslash^{\dagger} y) = M_2$.

Finally, suppose $\lambda(r(L)) = a$ and $\lambda'(r(L)) = u$. Consider the child f of $r(L)$ such that d is a descendant of f. Since topological embeddings preserve descendant relationships, it must be the case that $\lambda(f)$ is a node of $T_{b_q}\backslash^{\dagger} T_s$ and $\lambda'(f)$ is a node of $T'_{v_p}\backslash^{\dagger} T'_y$. Then, $L_f \in LCS(b_q\backslash^{\dagger} s, v_p\backslash^{\dagger} y)$ since any common subtree larger than L_f could be substituted for L_f in L, thus contradicting the choice of L as a largest common subtree. Similarly, for any other child g of $r(L)$, there must be children b_i of a and v_j of u such that $L_g \in LCS(b_i, v_j)$. By Corollary 4.4, maximizing the size of a common subtree involves forming a maximum weighted matching between the children of a (except b_q) and children of u (except v_p). Thus, we must solve the indicated matching problem, yielding $\mathcal{L}(a\backslash^{\dagger} s, u\backslash^{\dagger} y) = M_3$. ■

Lemma 6.4. *For $a, u, y, b_1, \ldots, b_k, v_1, \ldots, v_\ell$, and p as defined above, let*

$$\begin{aligned}
M_1 &= \max_{1 \leq j \leq \ell, j \neq p} \{\mathcal{L}(a, v_j)\}; \\
M_2 &= \max_{1 \leq i \leq k} \{\mathcal{L}(b_i, u\backslash^{\dagger} y)\}; \\
M_3 &= \mathcal{L}(a, v_p\backslash^{\dagger} y); \text{ and} \\
M_4 &= MaxWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\}) + 1.
\end{aligned}$$

Then, $\mathcal{L}(a, u\backslash^{\dagger} y) = \max\{M_1, M_2, M_3, M_4\}$.

Proof. As this proof is similar to that of Lemma 6.3, here we present only an outline.

Consider $L \in LCS(a, u\backslash^{\dagger} y)$ and let λ and λ' be topological embeddings of L into T_a and $T'_u\backslash^{\dagger} T'_y$ respectively such that no node of L maps to y. If $\lambda(r(L)) = a$ and $\lambda'(r(L)) \neq u$, then $\lambda'(r(L)) \in V(T'_{v_j})$ for some v_j a child of u. If $j \neq p$ then $\mathcal{L}(a, u\backslash^{\dagger} y) = M_1$ and if $j = p$ then $\mathcal{L}(a, u\backslash^{\dagger} y) = M_3$. Similarly, if $\lambda(r(L)) \neq a$ and $\lambda'(r(L)) = u$ then $\mathcal{L}(a, u\backslash^{\dagger} y) = M_2$. Finally, if $\lambda(r(L)) = a$ and $\lambda'(r(L)) = u$ we must solve a matching problem of the children of a and u and it follows that $\mathcal{L}(a, u\backslash^{\dagger} y) = M_4$. ■

6.3 Handling T

The only remaining bottleneck to a fast parallel algorithm is the handling of T. In this section we show that when processing a vertex α of $\mathcal{B}_{T'}$ labeled by the tree $T'_u\backslash^{\dagger} T'_y$, we can compute $\mathcal{L}(a\backslash^{\dagger} s, u\backslash^{\dagger} y)$ in parallel for all node pairs a and s in T. It is not difficult to verify that when α is a Brent break, this is possible by using the same procedure as outlined in the previous section. The case of a child break, however, is more complicated. In particular, since we are handling all pairs a, s in T simultaneously, we will not know the quantity $\mathcal{L}(b_q\backslash^{\dagger} s, u\backslash^{\dagger} y)$ before determining $\mathcal{L}(a\backslash^{\dagger} s, u\backslash^{\dagger} y)$, and hence we will not be able to compute M_2 in Lemma 6.3. Similarly, we will not know $\mathcal{L}(b_i, u\dagger y)$, $1 \leq i \leq k, i \neq q$, before we determine $\mathcal{L}(a\dagger s, u\dagger y)$ and therefore will not be able to compute M_2 in Lemma 6.4.

However, notice that in both these lemmas we can compute M_1 and M_3 (and in Lemma 6.4, M_4) directly for all descendants c of a. In particular, to compute M_2 in Lemma 6.3 consider an $L \in LCS(a\dagger s, u\dagger y)$ such that λ and λ' are topological embeddings of L into $T_a\dagger T_s$ and $T'_u\dagger T'_y$ respectively such that $\lambda(r(L)) \neq a$ and $\lambda'(r(L)) = u$. Then, there is a descendant c of a such that $\lambda(r(L)) = c$ with b_q, c, and s on a root-leaf path in $T_a\dagger T_s$. Let $d_1, \ldots, d_{k'}$ be the children of c with s a descendant of d_m. Then,

$$\mathcal{L}(c\dagger s, u\dagger y) = MaxWM(\{d_1, \ldots, d_{k'}\}\backslash\{d_m\}, \{v_1, \ldots, v_\ell\}\backslash\{v_q\}) + \mathcal{L}(d_m\dagger s, v_p\dagger y) + 1.$$

Although we do not know which node of $\pi(a,s)$ corresponds to c, we can try all of them in parallel and choose the one that maximizes the weight of the matching; this will be the value of M_2. Similarly, we can compute the value of M_2 in Lemma 6.4 as follows:

$$\begin{aligned} M_2 &= \max_{c \in V(T_a), c \neq a} \{\mathcal{L}(c, u\dagger y)\} \\ &= \max_{c \in V(T_a), c \neq a} \{MaxWM(\{d_1, \ldots, d_{k'}\}, \{v_1, \ldots, v_\ell\}) + 1 \mid d_1, \ldots, d_{k'} \text{ children of } c\} \end{aligned}$$

We outline the algorithm below for the case in which we are computing $\mathcal{L}(a\dagger s, u\dagger y)$ at a child break.

```
LP1.   Form Brent Tree $\mathcal{B}_{T'}$ of $T'$.
LP2.   For every level in $\mathcal{B}_{T'}$, proceeding from leaves to root
LP3.       In parallel for each $\alpha$ at that level:
LP4.           If $\alpha$ is scarred then
LP5.               Let the label of $\alpha$ be $T'_u\dagger T'_y$.
LP6.           In parallel, for every pair of nodes $a, s$ in $T$ with $a$ an ancestor of $s$:
LP7.               If $\alpha$ is a leaf (i.e. $y = u$) then
LP8.                   $\mathcal{L}(a\dagger s, u\dagger y) = 1$; Exit
LP9.               Let $b_1, \ldots, b_k$ be the children of $a$, $s \in V(T_{b_q})$
LP10.              Let $v_1, \ldots, v_\ell$ be the children of $u$, $y \in V(T'_{v_p})$
LP11.              $M_1(a,s) = \mathcal{L}(a\dagger s, v_p\dagger y)$
LP12.              $M_3(a,s) = MaxWM(\{b_1, \ldots, b_k\}\backslash\{b_q\}, \{v_1, \ldots, v_\ell\}\backslash\{v_q\}) + \mathcal{L}(b_q\dagger s, v_p\dagger y) + 1$
LP13.              $\mathcal{L}(a\dagger s, u\dagger y) = \max\{M_1(a,s), M_3(c,s) \mid c$ on $\pi(a,s)\}$
LP14.      End For
LP15.  End For
```

Notice that $M_2(a,s)$ is not explicitly computed but rather is implicit in step LP13 of the algorithm.

Theorem 6.5. *For trees T and T' of size $O(n)$, $\mathcal{L}(T,T')$ and $LCS(T,T')$ can be computed in time $O(\log^3 n)$ on a randomized $O(n^{7.5})$-processor* CREW PRAM.

Proof. We need only show that the algorithm outlined above meets the resource constraints in the theorem; the correctness follows from the preceding discussion. Since the Brent tree of T' can be created in time $O(\log^2 n)$ with $O(n^4)$ processors [13], it will suffice to show that each of the $O(\log n)$ levels of the Brent tree can be processed in time $O(\log^2 n)$ using $O(n^{7.5})$ processors. We give details of the calculations for handling a scarred child break using Lemma 6.3; the remaining cases require fewer resources.

For a particular vertex α, and a particular pair of nodes a and s in T, M_1 can be determined using table look-up in $O(1)$ time using $O(1)$ processors. M_3 can be determined using a maximum weight perfect matching, which can be set up and solved in randomized time $O(\log^2 n)$ with $O(n^{5.5})$ processors for a problem of size n [26]. $\mathcal{L}(a\overset{+}{\setminus}s, u\overset{+}{\setminus}y)$ can be determined in $O(\log n)$ time with $O(n)$ processors. Therefore, to compute $M_3(a,s)$ for all nodes a and s in T requires a total of $O(n^{7.5})$ processors. Since a particular level of the Brent tree is a partition of the nodes of T', $O(n^{7.5})$ processors suffice overall. ■

7 The SCES problem - topological embedding

The algorithms for the smallest common embeddable supertree (SCES) problem under topological embedding are similar in structure to those for the largest common embeddable subtree problem. However, some of the statements and proofs of lemmas differ; for the sake of completeness, where appropriate, we give these lemmas with their proofs in full. The algorithms for SCES under subgraph isomorphism will be presented in Section 9.

7.1 Technical lemmas

For tree nodes $a \in V(T)$ and $u \in V(T')$, we define $SCS_e(a,u)$ to be the set of smallest common supertrees of T_a and T'_u under topological embedding. This definition can immediately be extended for use in conjunction with Brent restructuring: for s a descendant of a and y a descendant of u, $SCS_e(a\overset{+}{\setminus}s, u\overset{+}{\setminus}y)$ is the set of smallest common supertrees of $T_a\overset{+}{\setminus}T_s$ and $T'_u\overset{+}{\setminus}T'_y$ such that for every $S \in SCS_e(a\overset{+}{\setminus}s, u\overset{+}{\setminus}y)$, there are embeddings that map s and y to a distinguished node of S. The set $SCS_e(a, u\overset{+}{\setminus}y)$ is the set of smallest common supertrees of T_a and $T'_u\overset{+}{\setminus}T'_y$, where y maps to no node of a tree in $SCS_e(a, u\overset{+}{\setminus}y)$. The notation $\mathcal{S}$ is used to indicate the size of trees in sets, with $\mathcal{S}_e(a,u)$, $\mathcal{S}_e(a\overset{+}{\setminus}s, u\overset{+}{\setminus}y)$, and $\mathcal{S}_e(a, u\overset{+}{\setminus}y)$ referring to sets $SCS_e(a,u)$, $SCS_e(a\overset{+}{\setminus}s, u\overset{+}{\setminus}y)$, and $SCS_e(a, u\overset{+}{\setminus}y)$ respectively. For subgraph isomorphism, we define SCS_i and $\mathcal{S}_i$ similarly and omit subscripts when the embedding is clear from context. Furthermore, we extend the definition of $\mathcal{S}$ so that $\mathcal{S}(\emptyset, u) = |T'_u|$ and $\mathcal{S}(a, \emptyset) = |T_a|$.

The first lemma follows directly from the fact that we are working with embeddings on rooted trees.

Lemma 7.1. *For any $S \in SCS(T,T')$ and for any pair σ and σ' of embeddings (subgraph isomorphism or topological embedding) from T and T' to S, one of the following conditions must hold: $\sigma(r(T)) = \sigma'(r(T'))$; $\sigma(r(T))$ is a proper ancestor of $\sigma'(r(T'))$; or $\sigma(r(T))$ is a proper descendant of $\sigma'(r(T'))$.*

Next, we prove an analog of Lemma 4.1.

Lemma 7.2. *For any $S \in SCS(T,T')$ (under either subgraph isomorphism or topological embedding) and for every pair of embeddings σ and σ' from T and T' to S, either $\sigma(r(T)) = r(S)$ or $\sigma'(r(T')) = r(S)$ (or both).*

Proof. Suppose $\sigma(r(T)) = g \neq r(S)$ and $\sigma'(r(T')) = h \neq r(S)$; we will show that this contradicts the minimality of S. If S_g and S_h are disjoint, then we can form S' by taking S_g and S_h and identifying g and h; S' has size one smaller than S yet T and T' embed in S'. If S_g and S_h have a node in common, then by Lemma 7.1, we can assume without loss of generality that $g \in V(S_h)$. It is not difficult to see that both T and T' embed in S_h, a contradiction to the minimality of S. ■

For $b_1, \ldots, b_k$ nodes of T and $v_1, \ldots, v_\ell$ nodes of T', we define $MinWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\})$ to be the minimum weight perfect matching in the weighted bipartite graph $G(X, Y, E)$ where

1. $X = \{b_1, \ldots, b_k, n_1, \ldots, n_\ell\}$;
2. $Y = \{v_1, \ldots, v_\ell, m_1, \ldots, m_k\}$;
3. for $1 \leq i \leq k$ and $1 \leq j \leq \ell$, the edge (b_i, v_j) has weight $\mathcal{S}(b_i, v_j)$ and the edge (n_j, m_i) has weight 0;
4. for $1 \leq i \leq k$, the edge (b_i, m_i) has weight $\mathcal{S}(b_i, \emptyset)$; and
5. for $1 \leq j \leq \ell$, the edge (n_j, v_j) has weight $\mathcal{S}(\emptyset, v_j)$.

Results analogous to Lemma 4.3 and Corollary 4.4 are given below.

Lemma 7.3. *Let $X = \{b_1, \ldots, b_k\}$ and $Y = \{v_1, \ldots, v_\ell\}$ be the children of $r(T)$ and $r(T')$, respectively. If there is a perfect matching $\mathcal{M}$ of weight $W_\mathcal{M}$ in $G(X, Y, E)$ then there is a tree S of size $W_\mathcal{M} + 1$ such that T and T' root-to-root embed in S. Conversely, if T and T' root-to-root embed in a tree S of size W then there is a perfect matching of weight at most $W - 1$ in $G(X, Y, E)$.*

Proof. For $\mathcal{M}$ a perfect matching in $G(X, Y, E)$ of weight $W_\mathcal{M}$, we can partition the edges of $\mathcal{M}$ into sets $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ and $\mathcal{M}_4$ as follows.

1. $\mathcal{M}_1$ is the set of all edges of the form (b_i, v_j), without loss of generality we can assume that $\mathcal{M}_1 = \{(b_1, v_1), \ldots, (b_h, v_h)\}$ for some h;
2. $\mathcal{M}_2$ is the set of edges $\{(b_{h+1}, m_{h+1}), \ldots, (b_k, m_k)\}$;
3. $\mathcal{M}_3$ is the set of edges $\{(n_{h+1}, v_{h+1}), \ldots, (n_\ell, v_\ell)\}$; and
4. $\mathcal{M}_4$ is the set of edges of the form (n_j, m_i).

For each edge (b_i, v_i) in $\mathcal{M}_1$, there is a tree S_i of size $\mathcal{S}(b_i, v_i)$ such that T_{b_i} and T'_{v_i} embed in S_i. For an edge (b_i, m_i) in $\mathcal{M}_2$, let P_i be a tree isomorphic to T_{b_i} and for an edge (n_j, v_j) in $\mathcal{M}_3$, let Q_j be a tree isomorphic to T'_{v_j}. We define S to be a root with children such that the subtrees rooted at the children are the trees $S_1, \ldots, S_h, P_{h+1}, \ldots, P_k, Q_{h+1}, \ldots, Q_\ell$. Then S has size $W_\mathcal{M} + 1$ and T and T' root-to-root embed in S.

For the converse, suppose T and T' root-to-root embed in a tree S of size W and suppose σ and σ' are root-to-root embeddings of T and T' into S respectively. Then, the edges

$$\begin{aligned}
&\{(b_i, v_j) \mid \text{ for some child } f \text{ of } r(S),\ \sigma(b_i) \in V(S_f) \text{ and } \sigma'(v_j) \in V(S_f)\} \\
&\quad \cup\{(b_i, m_i) \mid \text{ for some child } f \text{ of } r(S),\ \sigma(b_i) \in V(S_f) \text{ and } \sigma'(v_j) \notin V(S_f) \text{ for all } j\} \\
&\quad \cup\{(n_j, v_j) \mid \text{ for some child } f \text{ of } r(S),\ \sigma'(v_j) \in V(S_f) \text{ and } \sigma(b_i) \notin V(S_f) \text{ for all } i\} \\
&\quad \cup\{\text{a perfect matching of } n_j\text{'s and } m_i\text{'s unmatched above}\}
\end{aligned}$$

form a perfect matching in $G(X, Y, E)$. It is straightforward to show that this matching has weight at most $W - 1$. ■

Corollary 7.4. *For X and Y as in Lemma 7.3, $MinWM(X, Y, E) + 1$ is the size of the smallest tree S such that both T and T' root-to-root embed in S.*

7.2 Sequential algorithm

The sequential algorithm is based on the following analog of Lemma 5.2.

Lemma 7.5. *For any $a \in V(T)$ with children $b_1, \ldots, b_k$ and $u \in V(T')$ with children $v_1, \ldots, v_\ell$, we define the following three quantities:*

$$\begin{aligned} M_1 &= \min\{\mathcal{S}(a, v_i) + 1 + \sum_{j \neq i} \mathcal{S}(\emptyset, v_j) \mid 1 \leq i \leq \ell\}; \\ M_2 &= \min\{\mathcal{S}(b_i, u) + 1 + \sum_{j \neq i} \mathcal{S}(b_j, \emptyset) \mid 1 \leq i \leq k\}; \text{ and} \\ M_3 &= MinWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\}) + 1. \end{aligned}$$

Then, $\mathcal{S}(a, u) = \min\{M_1, M_2, M_3\}$.

Proof. Let σ and σ' be topological embeddings of T_a and T'_u respectively into $S \in SCS(a, u)$ such that $\sigma(a) = g$ and $\sigma'(u) = h$. If $h = r(S)$ and $g \neq r(S)$ then $|S| = M_1$ since $S_g \in SCS(a, v_i)$ for some child v_i of u and $S \backslash S_g = T'_u \backslash T'_{v_i}$. Similarly, if h is a proper descendant of g then $|S| = M_2$. Finally, if $g = h$ then by Corollary 7.4, $\mathcal{S}(a, u) = M_3$. ∎

Our algorithm has the same structure as that for determining the largest common subtree using the values of M_1, M_2 and M_3. It is not difficult to show the following result.

Theorem 7.6. *For T and T' trees of size $O(n)$, a smallest common supertree of T and T' under topological embedding can be computed in time $O(n^{2.5} \log n)$.*

7.3 Parallel algorithm

For our parallel algorithm, we will proceed by forming the Brent tree of T' and processing it level by level from leaves to root. At each vertex α labeled by, for example, the scarred tree $T'_u \setminus^{+} T'_y$, we will compute the quantities $\mathcal{S}(a \setminus^{+} s, u \setminus^{+} y)$ and $\mathcal{S}(a, u \setminus^{+} y)$ for every pair of nodes a and s of T, s a descendant of a. We will only consider the case of scarred labels of the Brent tree, as unscarred labels are handled analogously.

For α occurring at a Brent break, let the children of α be labeled by $T'_u \setminus^{+} T'_x$ and $T'_x \setminus^{+} T'_y$ for some $x \in \pi(u, y)$. Then we have:

Lemma 7.7. *For $a, s, u, x,$ and y defined as above,*

$$\mathcal{S}(a \setminus^{+} s, u \setminus^{+} y) = \min_{t \in \pi(a,s)} \{\mathcal{S}(a \setminus^{+} t, u \setminus^{+} x) + \mathcal{S}(t \setminus^{+} s, x \setminus^{+} y) - 1\}$$

and

$$\mathcal{S}(a, u \setminus^{+} y) = \min\{\mathcal{S}(a, u \setminus^{+} x) + \mathcal{S}(\emptyset, x \setminus^{+} y) - 1, \min_{t \in V(T_a)} \{\mathcal{S}(a \setminus^{+} t, u \setminus^{+} x) + \mathcal{S}(t, x \setminus^{+} y) - 1\}\}.$$

Proof. Let $S \in SCS(a \setminus^{+} s, u \setminus^{+} y)$ with σ and σ' topological embeddings from $T_a \setminus^{+} T_s$ and $T'_u \setminus^{+} T'_y$ to S, respectively. Since either $\sigma(a)$ is an ancestor of $\sigma'(u)$ or vice-versa, it follows that $\sigma(a)$, $\sigma'(u)$, $\sigma'(x)$, and $\sigma'(y) = \sigma(s)$ all occur on a common root-leaf path in S. To prove the first part of the lemma, it will suffice to find a node $t \in \pi(a, s)$ such that $\mathcal{S}(a \setminus^{+} s, u \setminus^{+} y) = \mathcal{S}(a \setminus^{+} t, u \setminus^{+} x) + \mathcal{S}(t \setminus^{+} s, x \setminus^{+} y) - 1$.

If $\sigma(a) \in V(S_{\sigma'(x)})$, then all of $T_a \backslash T_s$ must map to $S_{\sigma'(x)}$, and $S \backslash S_{\sigma'(x)}$ is identical to $T'_u \backslash T'_x$. Choosing $t = a$, we can conclude that $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = |S| = |T'_u \backslash T'_x| + |S_{\sigma'(x)}| = \mathcal{S}(a \backslash^{+} a, u \backslash^{+} x) + \mathcal{S}(a \backslash^{+} s, x \backslash^{+} y) - 1$, as required.

When $\sigma(a) \notin V(S_{\sigma'(x)})$, we can choose as t the unique node in $\pi(a, s)$ such that $\sigma(t)$ is in $V(S_{\sigma'(x)})$ but the parent of t is not in $V(S_{\sigma'(x)})$. We then have $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = |S| = |S \backslash S_{\sigma'(x)}| + |S_{\sigma'(x)}| = \mathcal{S}(a \backslash^{+} t, u \backslash^{+} x) + \mathcal{S}(t \backslash^{+} s, x \backslash^{+} y) - 1$, as required.

The argument for $\mathcal{S}(a, u \backslash^{+} y)$ is similar. ■

We next consider the case in which α occurs at a child break.

Lemma 7.8. *For a, s, u, and y as above, $b_1, \ldots, b_k$ the children of a such that s is a descendant of b_q and $v_1, \ldots, v_\ell$ the children of u with y a descendant of v_p, let*

$$\begin{aligned}
M_1 &= \mathcal{S}(a \backslash^{+} s, v_p \backslash^{+} y) + \mathcal{S}(\emptyset, u \backslash^{+} v_p) - 1; \\
M_2 &= \mathcal{S}(b_q \backslash^{+} s, u \backslash^{+} y) + \mathcal{S}(a \backslash^{+} b_q, \emptyset) - 1; \ \textit{and} \\
M_3 &= \mathit{MinWM}(\{b_1, \ldots, b_k\} \backslash \{b_q\}, \{v_1, \ldots, v_\ell\} \backslash \{v_p\}) + \mathcal{S}(b_q \backslash^{+} s, v_p \backslash^{+} y) + 1.
\end{aligned}$$

Then, $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = \min\{M_1, M_2, M_3\}$.

Proof. Let σ and σ' be the required topological embeddings from $T_a \backslash^{+} T_s$ and $T'_u \backslash^{+} T'_y$ into $S \in SCS(a \backslash^{+} s, u \backslash^{+} y)$ such that $\sigma(s) = \sigma'(y)$. Then, by Lemma 7.2, either $\sigma(a) = r(S)$, $\sigma'(u) = r(S)$ or both; we consider each case in turn.

If $\sigma(a) = r(S)$ and $\sigma'(u) \neq r(S)$, then $\sigma'(u) \in S_{\sigma(b_q)}$. Hence, the image of $T'_u \backslash^{+} T'_y$ is contained entirely in $S_{\sigma(b_q)}$ and $S_{\sigma(b_q)} \in SCS(b_q \backslash^{+} s, u \backslash^{+} y)$. Moreover, the subtrees of T rooted at the children of a (with the exception of T_{b_q}) are all contained in $S \backslash S_{\sigma(b_q)}$. Thus, since b_q is counted twice, $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = M_2$. Similarly, we can show that if $\sigma(a) \neq r(S)$ and $\sigma'(u) = r(S)$ then $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = M_1$.

If $\sigma(a) = \sigma'(u)$, then since $\sigma(u) = \sigma'(y)$ there must be a child f of S such that $\sigma(b_q), \sigma'(v_p) \in S_f$ and no other node of either $T_a \backslash T_{b_q}$ or $T'_u \backslash T'_{v_p}$ maps to S_f. Therefore, $S_f \in SCS(b_q \backslash^{+} s, v_p \backslash^{+} y)$. By Corollary 7.4, $S \backslash S_f$ has size $\mathit{MinWM}(\{b_1, \ldots, b_k\} \backslash \{b_q\}, \{v_1, \ldots, v_\ell\} \backslash \{v_p\})$ and it follows that $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = M_3$. ■

Due to its similarity to the preceding lemma, the following lemma is stated without proof.

Lemma 7.9. *For $a, u, y, b_1, \ldots, b_k, v_1, \ldots, v_\ell, p$, and q as defined above, let*

$$\begin{aligned}
M_1 &= \min_{1 \leq j \leq \ell, j \neq p} \{\mathcal{S}(a, v_j) + \mathcal{S}(\emptyset, u \backslash^{+} v_j) - 1\}; \\
M_2 &= \min_{1 \leq i \leq k, i \neq q} \{\mathcal{S}(b_i, u \backslash^{+} y) + \mathcal{S}(a \backslash^{+} b_i, \emptyset) - 1\}; \\
M_3 &= \mathcal{S}(a, v_p \backslash^{+} y) + \mathcal{S}(\emptyset, u \backslash^{+} v_p) - 1; \ \textit{and} \\
M_4 &= \mathit{MinWM}(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\} \backslash \{v_p\}) + \mathcal{S}(\emptyset, v_p \backslash^{+} y) + 1.
\end{aligned}$$

Then, $\mathcal{S}(a, u \backslash^{+} y) = \min\{M_1, M_2, M_3, M_4\}$.

We again note that working up the Brent tree of T', we can compute M_1, M_3 (and M_4 in Lemma 7.9) using previously computed values. The value of M_2 cannot be computed directly. However, in that case $\sigma'(u)$ is the same as $\sigma(c)$ for some $c \in \pi(a, s)$. Then, $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y) = \mathcal{S}(c \backslash^{+} s, u \backslash^{+} y)$ in Lemma 7.8, and $\mathcal{S}(a, u \backslash^{+} y) = \mathcal{S}(c, u \backslash^{+} y)$ in Lemma 7.9. The value $\mathcal{S}(c \backslash^{+} s, u \backslash^{+} y)$ is obtained by solving a matching problem. This yields the following parallel algorithm for computing values of the form $\mathcal{S}(a \backslash^{+} s, u \backslash^{+} y)$ at a child break.

```
SP1.   Form Brent Tree $\mathcal{B}_{T'}$ of $T'$.
SP2.   For every level in $\mathcal{B}_{T'}$, proceeding from leaves to root
SP3.       In parallel for each $\alpha$ at that level:
SP4.           If $\alpha$ is scarred then
SP5.               Let the label of $\alpha$ be $T'_u \setminus^+ T'_y$.
SP6.           In parallel, for every pair of nodes $a, s$ nodes in $T$ with $a$ an ancestor of $s$:
SP7.               If $\alpha$ is a leaf (i.e. $y = u$) then
SP8.                   $\mathcal{S}(a \setminus^+ s, u \setminus^+ y) = |T_a \setminus^+ T_s|$; Exit
SP9.               Let $b_1, \ldots, b_k$ be the children of $a$, $s \in V(T_{b_q})$
SP10.              Let $v_1, \ldots, v_\ell$ be the children of $u$, $y \in V(T'_{v_p})$
SP11.              $M_1(a,s) = \mathcal{S}(a \setminus^+ s, v_p \setminus^+ y) + \mathcal{S}(\emptyset, u \setminus^+ v_p) - 1$
SP12.              $M_3(a,s) = MinWM(\{b_1, \ldots, b_k\} \setminus \{b_q\}, \{v_1, \ldots, v_\ell\} \setminus \{v_q\}) + \mathcal{S}(b_q \setminus^+ s, v_p \setminus^+ y) + 1$
SP13.              $\mathcal{S}(a \setminus^+ s, u \setminus^+ y) = \min\{M_1(a,s), M_3(c,s) \mid c$ on $\pi(a,s)\}$
SP14.      End For
SP15.  End For
```

Theorem 7.10. *For trees T and T' of size $O(n)$, $\mathcal{S}(T,T')$ and $SCS(T,T')$ can be computed in time $O(\log^3 n)$ on a randomized $O(n^{7.5})$-processor* CREW PRAM.

8 Ordered trees

Despite the same overall structure, there are a number of distinctions between our algorithms for the LCES and SCES problems for unordered and ordered trees. The key difference is the type of matching problem that must be solved. In particular, we must find matchings that preserve the ordering of the children.

8.1 Planar Matchings

Let $G(X,Y,E)$ be a bipartite graph with $X = \{x_1, \ldots, x_k\}$ and $Y = \{y_1, \ldots, y_\ell\}$. A *planar matching* on G is a subset $\mathcal{M}$ of E such that for any two edges $e = (x_i, y_j)$ and $e' = (x_{i'}, y_{j'})$ of $\mathcal{M}$, $i > i'$ if and only if $j > j'$. If G is edge weighted (with edge e having weight $w(e)$), the *maximum weight planar matching problem* is the problem of finding the planar matching that maximizes the sum of the edge weights of the edges in the matching. We denote the maximum weight of a planar matching by $MaxWPM(X,Y,E)$.

Our algorithm for computing $MaxWPM(X,Y,E)$ is based on the problem of leveling a weighted directed acyclic graph.

Definition: For G a weighted directed acyclic graph where $w(x,y)$ is the weight of the edge (x,y), the *weighted level number* $\ell(v)$ of a node v is defined as follows:

1. for v a source node, $\ell(v) = 0$; and
2. for v a non-source node and $u_1, \ldots, u_k$ the set of nodes with edges to v, $\ell(v) = \max\{\ell(u_1) + w(u_1, v_1), \ldots, \ell(u_k) + w(u_k, v_k)\}$.

The weighted level number of a node is the weight of the heaviest path from a source to the node. For a graph G with a single source (which can be formed from any directed acyclic graph by adding one node connected to all sources), the following algorithm computes all weighted level numbers.

```
LNS1.   Let $v_1, \ldots, v_n$ be a topological sort of $V(G)$
LNS2.   Set $\ell'(v_i) = 0$ for every $i$
LNS3.   For $i = 1$ to $n$
LNS4.       Set $\ell(v_i) = \ell'(v_i)$
LNS5.       For every $j > i$ such that $(v_i, v_j) \in E(G)$
LNS6.           Let $\ell'(v_j) = \max\{\ell'(v_j), \ell(v_i) + w(v_i, v_j)\}$
LNS7.       End For
LNS8.   End For
```

For the parallel algorithm we use pointer doubling to propagate information more quickly.

```
LNP1.   For every node $v$ in parallel, set $\ell'(v) = \max\{w(u,v) \mid (u,v) \in E(G)\}$
LNP2.   For $\log n + 1$ rounds
LNP3.       In parallel, for every triple of nodes $v, x, y$
LNP4.           If $(v,x), (x,y) \in E(G)$ then
LNP5.               Let $w_{v,x,y} = \ell'(v) + w(v,x) + w(x,y)$
LNP7.               Add the edge $(v,y)$ to $E(G)$
LNP8.           Else $w_{v,x,y} = 0$
LNP9.       For each node $y$, let $w_y = \max\{w_{v,x,y} \mid v, x \in V(G)\}$
LNP10.      For each node $y$, let $\ell'(y) = \max\{\ell'(y), w_y\}$
LNP11.  End For
LNP12.  For each node $v$ in parallel, set $\ell(v) = \ell'(v)$
```

Lemma 8.1. *For G a weighted acyclic directed graph, the weighted level numbers of the nodes in G can be computed sequentially in time $O(n^2)$ and in parallel in time $O(\log^2 n)$ on an $O(n^3)$-processor* CREW PRAM.

Proof. It is clear that the sequential algorithm above correctly computes weighted level numbers; the complexity is obtained by noting that each edge of the graph is examined only once.

For the parallel algorithm, a straightforward proof by induction shows that for any node v at distance i from the source, after $\log i + 1$ iterations of Step LNP2, $\ell'(v)$ will equal $\ell(v)$. In the for loop at Step LNP2, step LNP9 takes $O(\log n)$ time with all other steps running in $O(1)$ time. Therefore the total running time is $O(\log^2 n)$. For the processor count, we see that $O(n^3)$ triples of nodes are selected in step LNP3; assigning a processor to each of these triples allows us to compute the maximum at step LNP9. ∎

We are now ready to compute maximum weight planar matchings.

Lemma 8.2. *For $G(X, Y, E)$ a bipartite graph with edge weights such that $|X| = n$ and $|Y| = n$, the maximum weight planar matching problem can be solved sequentially in time $O(n^2)$ and in parallel in time $O(\log^2 n)$ on an $O(n^6)$-processor* deterministic CREW PRAM.

Proof. We will reduce this problem to finding the level numbers of a weighted directed acyclic graph.

We denote the weight of the edge from a node x to a node y by $w(x, y)$. Without loss of generality we assume there is an edge between every x_i and y_j; if no edge exists between some x_i and y_j, we place an edge of weight 0 between these two nodes.

Our algorithm is based on a similar formulation of Jiang, Wang, and Zhang in their work on tree alignment [15]:

$$MaxWPM(X,Y) = \max \begin{cases} MaxWPM(X\backslash\{x_n\}, Y\backslash\{y_n\}) + w(x_n, y_n) \\ MaxWPM(X\backslash\{x_n\}, Y) \\ MaxWPM(X, Y\backslash\{y_n\}) \end{cases}$$

Therefore, to compute $MaxWPM(X,Y)$ we compute $MaxWPM(\{x_1,\ldots,x_i\},\{y_1,\ldots,y_j\})$ for every i and j; for a particular i and j this is computed from $MaxWPM(\{x_1,\ldots,x_{i-1}\},\{y_1,\ldots,y_j\})$, $MaxWPM(\{x_1,\ldots,x_i\},\{y_1,\ldots,y_{j-1}\})$, and $MaxWPM(\{x_1,\ldots,x_{i-1}\},\{y_1,\ldots,y_{j-1}\})$.

We construct a weighted directed acyclic graph H where

1. $V(H)$ consists of one node for each pair (i,j) with $0 \leq i,j \leq n$, where the nodes are labeled by the pairs;
2. there are edges from (i,j) to $(i-1,j)$ and to $(i,j-1)$ of weight 0; and
3. there is an edge from (i,j) to $(i-1,j-1)$ of weight $w(x_i,y_j)$.

We claim that the weighted level number of $(0,0)$, $\ell((0,0))$, is $MaxWPM(X,Y)$. First consider a path in H from (n,n) to $(0,0)$ of weight $\ell((0,0))$; we construct a matching $\mathcal{M}$ in $G(X,Y,E)$ of the same weight. For two consecutive nodes (i_1,j_1) and (i_2,j_2) in this path, if $i_1 = i_2+1$ and $j_1 = j_2+1$, we put the edge (x_{i_1}, y_{i_1}) into $\mathcal{M}$. Then $\mathcal{M}$ will form a planar matching of weight $\ell((0,0))$; hence $\ell((0,0)) \leq MaxWPM(X,Y)$.

To show that $MaxWPM(X,Y) \leq \ell((0,0))$, we consider a maximum weight planar matching $\mathcal{M}$ in $G(X,Y,E)$ with edges $(x_{i_1},y_{j_1}), (x_{i_2},y_{j_2}),\ldots,(x_{i_k},y_{j_k})$ with $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$. Since $\mathcal{M}$ has maximum weight, for consecutive edges $(x_{i_{h-1}}, y_{j_{h-1}})$ and (x_{i_h}, y_{j_h}), either $i_h = i_{h-1}+1$ or $j_h = j_{h-1}+1$. We construct paths $P_1,\ldots,P_{k+1}$ as follows:

1. P_{k+1} is any path of weight 0 edges from (n,n) to (i_k,j_k);
2. for $1 < h \leq k$, if $i_h = i_{h-1}+1$ then P_h is the path $(i_h,j_h),(i_h,j_h-1),\ldots,(i_h,j_{h-1}+1),(i_{h-1},j_{h-1})$;
3. for $1 < h \leq k$, if $j_h = j_{h-1}+1$ then P_h is the path $(i_h,j_h),(i_h-1,j_h),\ldots,(i_{h-1}+1,j_h),(i_{h-1},j_{h-1})$; and
4. P_0 is any path of weight 0 edges from (i_1,j_1) to $(0,0)$.

We form the path P from (n,n) to $(0,0)$ by concatenating $P_{k+1},P_k,\ldots,P_1$ and P_0; the sum of the weights along this path is the weight of $\mathcal{M}$ and is at least $\ell((0,0))$.

Since the graph H has $O(n^2)$ nodes and edges, using our leveling algorithm to compute $\ell((0,0))$ results in the stated resource bounds. ∎

For the supertree problem, we will instead find a planar matching of minimum weight. However, because edge weights are nonnegative, the matching will always consist of the trivial empty set of edges. To circumvent this problem, we impose a penalty for not matching a node. In particular, suppose $G(X,Y,E)$ is an edge- and node-weighted bipartite graph. Then, $\mathcal{M} \subseteq E$ is a minimum weight planar matching if it is a planar matching which minimizes the sum of the edge weights of $\mathcal{M}$ plus the weight of the vertices not adjacent to any edge of $\mathcal{M}$. The weight of this matching is denoted by $MinWPM(X,Y,E)$.

Algorithms for minimum weight planar matchings are similar in structure to those for maximum weight planar matchings. We outline these results below.

Lemma 8.3. *For $G(X, Y, E)$ a bipartite graph with edge and node weights such that $|X| = |Y| = n$, the minimum weight planar matching problem can be solved sequentially in time $O(n^2)$ and in parallel in time $O(\log^2 n)$ on a* deterministic *$O(n^6)$-processor* CREW PRAM.

Proof. Let $MinWPM(X, Y)$ be the weight of the maximum weight planar matching in G and let the weight of edge (x_i, y_j) be $w(i, j)$ and the weight of vertex v be $w(v)$. Without loss of generality, we assume there is an edge between every x_i and y_j.

The algorithms follow from the formulation:

$$MinWPM(X,Y) = \min \begin{cases} MinWPM(X \backslash \{x_k\}, Y \backslash \{y_\ell\}) + w(x_k, y_\ell) \\ MinWPM(X \backslash \{x_k\}, Y) + w(x_k) \\ MinWPM(X, Y \backslash \{y_\ell\}) + w(y_\ell) \end{cases}$$

We now form the graph H as in the proof of Lemma 8.2 with the only difference being that an edge from (i, j) to $(i-1, j)$ has weight $w(x_i)$ and from (i, j) to $(i, j-1)$ has weight $w(y_j)$. Then, $MinWPM(X, Y, E)$ is the length of the shortest path in H from (n, n) to $(0, 0)$; such lengths can be computed sequentially in time $O(n^2)$ using Dijkstra's algorithm and in parallel in time $O(\log^2 n)$ with $O(n^6)$ processors [16]. The proof of correctness is similar to that in Lemma 8.2. ■

8.2 Algorithms for Ordered Trees

Algorithms for ordered trees can be formed from algorithms for unordered trees by substituting $MaxWPM$ for $MaxWM$ in the largest common embeddable subtree algorithms and $MinWPM$ for $MinWM$ in the smallest common embeddable supertree algorithms. Unlike the use of randomization in parallel algorithms in the unordered case, here all algorithms are deterministic.

Theorem 8.4. *For T and T' ordered trees of size $O(n)$, the largest common embeddable subtree problem and smallest common embeddable supertree problem can be solved sequentially in time $O(n^2)$ and in parallel in time $O(\log^3 n)$ on a* deterministic *$O(n^8)$-processor* CREW PRAM.

Proof. The sequential algorithm follows from the fact that the matching problems being solved are all disjoint. The parallel algorithm takes $O(\log^2 n)$ time for the matching problems on each level of the Brent tree with $O(\log n)$ levels in total. Matchings at a particular level of the Brent tree can be accomplished by $O(n^6)$ processors. Since there are $O(n^2)$ choices for the nodes a and s in T, a total of $O(n^8)$ processors suffices. ■

9 Subgraph isomorphism

Our algorithms for subgraph isomorphism are slight modifications of those for topological embedding. In this section we outline the modifications and prove that they suffice. We begin with the largest common subtree problem and then show that for subgraph isomorphism, the smallest common supertree can be obtained directly from the largest common subtree.

9.1 Largest common subtrees

Under both subgraph isomorphism and topological embedding, when we are computing a largest subtree L of trees T_a and T'_u, $r(L)$ maps to a, or to u, or to both a and u. It is the last case in which subgraph isomorphism differs from topological embedding. In particular, the children of $r(L)$ must map to the children of a and u rather than arbitrary descendants of these children. Therefore, we must keep track of not only the largest supertree of T_{b_i} and T'_{v_j} for every child b_i of a and v_j of u, but also the size of the largest supertree under a root-to-root embedding. We define the following notation: $LCS^r(a,u)$ is the set of largest trees L that are root-to-root subgraph isomorphic to T_a and T'_u and $\mathcal{L}^r(a,u)$ is the size of the trees in $LCS^r(a,u)$.

For $X = \{b_1, \ldots, b_k\}$ nodes of T_a and $Y = \{v_1, \ldots, v_\ell\}$ nodes of T'_u, we define $G^r(X,Y,E)$ to be the same graph as $G(X,Y,E)$ except that the weight on the edge (b_i, v_j) is $\mathcal{L}^r(b_i,v_j)$ instead of $\mathcal{L}(b_i,v_j)$. Then, $MaxWM^r(X,Y,E)$ is the size of the maximum weight matching in $G^r(X,Y,E)$. It is straightforward to prove the following variant of Corollary 4.4.

Corollary 9.1. *For X and Y the sets of children of a and u respectively, $MaxWM^r(X,Y,E)+1$ is the size of the largest tree that is root-to-root subgraph isomorphic to T_a and T'_u.*

Our sequential algorithm is based on the following observation:

Lemma 9.2. *For any $a \in V(T)$ with children $b_1, \ldots, b_k$ and any $u \in V(T')$ with children $v_1, \ldots, v_\ell$, one of the following conditions must hold for every $L \in LCS(a,u)$:*

1. *$L \in LCS(a, v_p)$ for some p, $1 \le p \le \ell$;*
2. *$L \in LCS(b_q, u)$ for some q, $1 \le q \le k$; or*
3. *$L \in LCS^r(a,u)$.*

We can compute $LCS^r(a,u)$ for every pair of nodes a and u using Corollary 9.1. Recalling that it is sufficient to compute the sizes of largest subtrees instead of the actual subtrees, we can obtain a sequential algorithm by using the following lemma.

Lemma 9.3. *For a, u, $b_1, \ldots, b_k$, and $v_1, \ldots, v_\ell$ as in Lemma 9.2, we define the following three quantities:*

$$\begin{aligned} M_1 &= \max\{\mathcal{L}(a, v_i) \mid 1 \le i \le \ell\}; \\ M_2 &= \max\{\mathcal{L}(b_j, u) \mid 1 \le j \le k\}; mbox\ and \\ M_3 &= MaxWM(\{b_1, \ldots, b_k\}, \{v_1, \ldots, v_\ell\}) + 1. \end{aligned}$$

Then, $\mathcal{L}(a,u) = \max\{M_1, M_2, M_3\}$ and $\mathcal{L}^r(a,u) = M_3$.

Notice that we must keep track of both $\mathcal{L}(a,u)$ and $\mathcal{L}^r(a,u)$ since the latter quantities are required to compute future maximum weight matchings. It is not difficult to see that the complexity of this algorithm is the same as that of the topological embedding algorithm.

For the parallel algorithm, we can again use Brent restructuring on the tree T'. Here, we must compute quantities $\mathcal{L}^r(a, u\setminus^{+} y)$ and $\mathcal{L}^r(a\setminus^{+} s, u\setminus^{+} y)$ as well as the other quantities described in Section 6; the modifications of Lemmas 6.1 to 6.4 are straightforward, as is the substitution of $MaxWM^r$ for $MaxWM$ in the matchings.

Theorem 9.4. *For T and T' trees of size $O(n)$, a largest common subtree of T and T' under subgraph isomorphism can be found sequentially in time $O(n^{2.5}\log n)$ and in parallel in time $O(\log^3 n)$ on a randomized $O(n^{7.5})$-processor* CREW PRAM.

9.2 Smallest common supertree

In this section we show that the smallest supertree problem for subgraph isomorphism can be directly reduced to the largest subtree problem.

Theorem 9.5. *For any trees T and T', $\mathcal{S}(r(T), r(T')) = |T| + |T'| - \mathcal{L}(r(T), r(T'))$.*

Proof. We first show that there is a tree S of size $|T| + |T'| - \mathcal{L}(r(T), r(T'))$ such that T and T' are subgraphs of S; this will allow us to conclude that $\mathcal{S}(r(T), r(T')) \leq |T| + |T'| - \mathcal{L}(r(T), r(T'))$. We will then show that the size of any tree in $SCS(r(T), r(T'))$ is at least $|T| + |T'| - \mathcal{L}(r(T), r(T'))$, completing the proof of the theorem.

We form a tree S from T and T' by adding to T those nodes of T' that are not part of a largest common subtree in T'. More formally, for $L \in LCS(r(T), r(T'))$ and λ and λ' subgraph isomorphisms from L to T and T', let R and R' be the subgraphs of T and T' that are the images of L under λ and λ'. We define the node set of S to be $V(T) \cup V(T') \backslash V(R')$, and the edge set to be the union of the following: all edges in $E(T)$; all edges in $E(T')$ except those with one endpoint in R'; and all edges of the form (a, y) where $a \in V(R)$, $y \in V(T') \backslash V(R')$ and $(\lambda'(\lambda^{-1}(a)), y) \in E(T')$.

It is clear that T is a subgraph of S. To see that T' is a subgraph of S, consider the replacement of R' by R in S. Since $|S| = |T| + |T'| - |L|$, $\mathcal{S}(r(T), r(T')) \leq |T| + |T'| - \mathcal{L}(r(T), r(T'))$.

We now show that $\mathcal{S}(r(T), r(T')) \geq |T| + |T'| - \mathcal{L}(r(T), r(T'))$. For $S \in SCS(r(T), r(T'))$ and σ and σ' subgraph isomorphisms from T and T' to S, let Q be the subgraph of S induced on those nodes which are in the image of both T and T'. Since Q is connected (and therefore a tree), Q is a subgraph of both T and T'. Then $|Q| \leq \mathcal{L}(r(T), r(T'))$ and therefore $|S| = |T| + |T'| - |Q| \geq |T| + |T'| - \mathcal{L}(r(T), r(T'))$. This completes the proof of the theorem. ∎

Since we have already developed algorithms for finding the largest common subtree, we obtain the following result.

Theorem 9.6. *For T and T' trees of size $O(n)$, a smallest common supertree of T and T' under subgraph isomorphism can be found sequentially in time $O(n^{2.5} \log n)$ and in parallel in time $O(\log^3 n)$ on a randomized $O(n^{7.5})$-processor* CREW PRAM.

10 Conclusions and directions for further research

In this paper we have presented a basic paradigm for sequential and parallel algorithms for the *Largest Common Embeddable Subtree Problem* and the *Smallest Common Embeddable Supertree Problem* for the subgraph isomorphism and topological embedding relations where the underlying trees can be ordered or unordered. For unordered trees, we have obtained sequential algorithms running in time $O(n^{2.5} \log n)$ and randomized parallel algorithms running in $O(\log^3 n)$ time with $O(n^{7.5})$ processors. In the ordered case, our algorithms for all these problems take $O(n^2)$ time sequentially and $O(\log^3 n)$ time deterministically in parallel with $O(n^8)$ processors.

Although all the algorithms in this paper have been based on the assumption that T and T' are unlabeled, it is straightforward to extend the algorithms to handle the cases in which T and T' are labeled. The known algorithms for computing smallest supertrees for minor containment handle ordered, labeled trees [15] and unordered, leaf-labeled trees [33]; it would be interesting to solve the problem for other possible labeling schemes.

The work on the *Smallest Common Embeddable Supertree Problem*, and most of the work on the *Largest Common Embeddable Subtree Problem*, has concentrated on two-input versions of the problem. Keselman and Amir have shown that the three-input version of the *Largest Common Embeddable Tree Problem* is NP-complete for subgraph isomorphism [17]. There are no known such results for the *Smallest Common Embeddable Supertree Problem.*

It remains to be seen whether or not the basic paradigm can be further modified to hone running times and processor counts for special cases, for embeddings listed here and potentially for others as well, for both labeled and unlabeled trees. Further progress on parallel algorithms for weighted bipartite matching could have a serious impact on the possibility of such improvements.

Acknowledgements

We are grateful to Ernst Mayr for pointers to references on weighted bipartite matching, and to Jianghai Fu for bringing to our attention the work of Jiang, Wang, and Zhang and the subject of supertrees in general.

References

[1] H. Alblas, Iteration of transformation passes over attributed program trees, *Acta-Informatica* **27** (1989), pp. 1–40.

[2] R. Brent, The parallel evaluation of general arithmetic expressions, *Journal of the ACM* **21**, 2 (1974), pp. 201–206.

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

[4] M. Dubiner, Z. Galil, and E. Magen, Faster tree pattern matching, *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, pp. 145–150, 1990.

[5] M. Farach and M. Thorup, Fast comparison of evolutionary trees, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 481–488, 1994.

[6] M. Farach and M. Thorup, Optimal evolutionary tree comparison by sparse dynamic programming, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 770-779, 1994.

[7] C.R. Finden and A.D. Gordon, Obtaining common pruned trees, *Journal of Classification* **2**, (1985), pp. 255–276.

[8] J. Friedman, Expressing logical formulas in natural language. *Formal methods in the study of language, Part I,* Math. Centrum, Amsterdam, 1981, pp. 113–130.

[9] H. Gabow and R. Tarjan, Faster scaling algorithms for network problems, *SIAM Journal on Computing* **18**, 5 (1989), pp. 1013–1036.

[10] P. Gibbons, R. Karp, G. Miller, and D. Soroker, Subtree isomorphism is in random NC, *Discrete Applied Mathematics* **29** (1990), pp. 35–62.

[11] R. Grossi, On finding common subtrees, *Theoretical Computer Science* **108** (1993), pp. 345–356.

[12] A. Gupta and N. Nishimura, Finding largest common embeddable subtrees, *Proceedings of the Twelfth Annual Symposium on Theoretical Aspects of Computer Science*, pp. 397–408, 1995.

[13] A. Gupta and N. Nishimura, The parallel complexity of tree embedding problems, *Journal of Algorithms* **18**, 1 (1995), pp. 176–200.

[14] A. Gupta and N. Nishimura, Sequential and parallel algorithms for embedding problems on classes of partial k-trees, *Proceedings of the Fourth Scandinavian Workshop on Algorithm Theory*, pp. 172–182, 1994.

[15] T. Jiang, L. Wang, and K. Zhang, Alignment of trees – an alternative to tree edit, *Combinatorial Pattern Matching*, pp. 75–86, 1994.

[16] R. Karp and V. Ramachandran, Parallel Algorithms for Shared Memory Machines, *in Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity,* editor J. van Leeuwen, The MIT Press, Cambridge, pp. 869–941, 1990.

[17] D. Keselman and A. Amir, Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 758-769, 1994.

[18] P. Kilpeläinen, Tree matching problems with applications to structured text databases, Ph.D. thesis, University of Helsinki, Department of Computer Science, November 1992.

[19] P. Kilpeläinen and H. Mannila, Grammatical tree matching, *Combinatorial Pattern Matching*, 1992.

[20] P. Kilpeläinen and H. Mannila, Ordered and unordered tree inclusion, *SIAM Journal on Computing* **24**, 2 (1995), pp. 340–356.

[21] S. R. Kosaraju, Efficient tree pattern matching, *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pp. 178–183, 1989.

[22] E. Kubicka, G. Kubicki, and F.R. McMorris, On agreement subtrees of 2 binary trees, *Congressus-Numerantium* **88** (1992), pp. 217–224.

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

[24] P. Materna, P. Sgall and Z. Hajicova, Linguistic constructions in transparent intensional logic, *Prague-Bulletin on Mathematical Linguistics* **43** (1985), pp. 5–24.

[25] D. Matula, Subtree isomorphism in $O(n^{5/2})$, *Annals of Discrete Mathematics* **2** (1978), pp. 91–106.

[26] K. Mulmuley, U. Vazirani, and V. Vazirani, Matching is as easy as matrix inversion, *Proceedings of the 19th Annual ACM Symposium on the Theory of Computing*, pp. 345–354, 1987.

[27] P. Powell and V. Ngo, Complexity of optimal vector code generation, *Theoretical Computer Science* **80** (1991), pp. 105–115.

[28] S. W. Reyner, An analysis of a good algorithm for the subtree problem, *SIAM Journal on Computing* **6**, 4, (1977), pp. 730–732.

[29] N. Robertson and P. Seymour, Graph Minors III. Planar tree-width, *Journal of Combinatorial Theory (Ser. B)* **36** (1984), pp. 49–64.

[30] N. Robertson and P. Seymour, Graph Minors II. Algorithm aspects of tree-width, *Journal of Algorithms* **7** (1986), pp. 309–322.

[31] M. Steel and T. Warnow, Kaikoura tree theorems: Computing the maximum agreement subtrees. Submitted for publication.

[32] R. M. Verma and S. W. Reyner, An analysis of a good algorithm for the subtree problem, corrected, *SIAM Journal on Computing* **18**, 5 (1989), pp. 906–908.

[33] T. Warnow, Tree compatibility and inferring evolutionary history, *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 382–391, 1993.

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