Largest Weight Common Subtree Embeddings with Distance Penalties

Andre Droschinsky

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
andre.droschinsky@tu-dortmund.de
https://orcid.org/0000-0002-7983-3739

Nils M. Kriege

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
nils.kriege@tu-dortmund.de
https://orcid.org/0000-0003-2645-947X

Petra Mutzel

Department of Computer Science, TU Dortmund University
Otto-Hahn-Str. 14, 44221 Dortmund, Germany
petra.mutzel@tu-dortmund.de
https://orcid.org/0000-0001-7621-971X

Abstract

The largest common embeddable subtree problem asks for the largest possible tree embeddable into two input trees and generalizes the classical maximum common subtree problem. Several variants of the problem in labeled and unlabeled rooted trees have been studied, e.g., for the comparison of evolutionary trees. We consider a generalization, where the sought embedding is maximal with regard to a weight function on pairs of labels. We support rooted and unrooted trees with vertex and edge labels as well as distance penalties for skipping vertices. This variant is important for many applications such as the comparison of chemical structures and evolutionary trees. Our algorithm computes the solution from a series of bipartite matching instances, which are solved efficiently by exploiting their structural relation and imbalance. Our analysis shows that our approach improves or matches the running time of the formally best algorithms for several problem variants. Specifically, we obtain a running time of $\mathcal{O}(|T|\,|T'|\Delta)$ for two rooted or unrooted trees T and T', where $\Delta = \min\{\Delta(T), \Delta(T')\}$ with $\Delta(X)$ the maximum degree of X. If the weights are integral and at most C, we obtain a running time of $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|,|T'|\}))$ for rooted trees.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases maximum common subtree, largest embeddable subtree, topological embedding, maximum weight matching, subtree homeomorphism

Digital Object Identifier 10.4230/LIPIcs.MFCS.2018.54

Funding This work was supported by the German Research Foundation (DFG), priority programme "Algorithms for Big Data" (SPP 1736).

1 Introduction

The maximum common subgraph problem asks for a graph with a maximum number of vertices that is isomorphic to induced subgraphs of two input graphs. This problem arises in many domains, where it is important to find the common parts of objects which can be represented as graphs. An example of this are chemical structures, which can be interpreted directly as labeled graphs. Therefore, the problem has been studied extensively in cheminformatics [5, 16, 17]. Although elaborated backtracking algorithms have been developed [16, 2], solving large instances in practice is a great challenge. The maximum common subgraph problem is NP-hard and remains so even when the input graphs are restricted to trees [6]. However, in trees it becomes polynomial-time solvable when the common subgraph is required to be connected, i.e., it must be a tree itself. This problem is then referred to as *maximum common subtree problem* and the first algorithm solving it in polynomial-time is attributed to J. Edmonds [12]. Also requiring that the common subgraph must be connected (or even partially biconnected) several extensions to tree-like graphs have been proposed, primarily for applications in cheminformatics [19, 17, 3]. Some of these approaches are not suitable for practical applications due to high constants hidden in the polynomial running time. Other algorithms are efficient in practice, but restrict the search space to specific common subgraphs. Instead of developing maximum common subgraph algorithms for more general graph classes, which has proven difficult, a different approach is to represent molecules simplified as trees [15]. Then, vertices typically represent groups of atoms and their comparison requires to score the similarity of two vertices by a weight function. This, however, is often not supported by algorithms for tree comparison. Moreover, it maybe desirable to map a path in one tree to a single edge in the other tree, skipping the inner vertices. Formally, this is achieved by *graph homeomorphism* instead of isomorphism.

Various variants for comparing trees have been proposed and investigated [18]. Most of them assume rooted trees, which may be ordered or unordered. Algorithms tailored to the comparison of evolutionary trees typically assume only the leaves to be labeled, while others support labels on all vertices or do not consider labels at all. The well-known agreement subtree problem, for example, considers the case, where only the leaf nodes are labeled, with no label appearing more than once per tree [11]. We discuss the approaches most relevant for our work. Gupta and Nishimura [8] investigated the *largest common embeddable subtree* problem in unlabeled rooted trees. Their definition is based on topological embedding (or homeomorphism) and allows to map edges of the common subtree to vertex-disjoint paths in the input trees. The algorithm uses the classical idea to decompose the problem into subproblems for smaller trees, which are solved via bipartite matching. A solution for two trees with at most n vertices is computed in time $\mathcal{O}(n^{2.5} \log n)$ using a dynamic programming approach. Fig. 1a illustrates the difference between maximum common subgraph and largest common embeddable subtree. Lozano and Valiente [10] investigated the *maximum common embedded subtree* problem, which is based on edge contraction. In both cases the input graphs are rooted unlabeled trees. Note, the definition of their problems is not equivalent. The first is polynomial time solvable, while the second is NP-hard for unordered trees, but polynomial time solvable for ordered trees. Many algorithms do not support trees, where leaves and the inner vertices both have labels. A notable exception is the approach by Kao et al. [9], where only vertices with the same label may be mapped. This algorithm generalizes the approach by Gupta/Nishimura and improves its running time to $\mathcal{O}(\sqrt{d}D \log \frac{2n}{d})$, where D denotes the number of vertex pairs with the same label and d the maximum degree of all vertices.

We consider the problem of finding a *largest weight common subtree embedding* (LaWeCSE), where matching vertices are not required to have the same label, but their degree of agreement is determined by a weight function. We build on the basic ideas of Gupta and Nishimura [8].

To prevent arbitrarily long paths which are mapped to a common edge we study a linear distance penalty for paths of length greater than 1. Note that, by choosing a high distance penalty, we solve the maximum common subtree (MCS) problem as a special case. By choosing weight 1 for equal labels and sufficiently small negative weights otherwise, we solve a problem equivalent to the one studied by Kao et al. [9].

Our contribution. We propose and analyze algorithms for finding largest weight common subtree embeddings. Our method requires to solve a series of bipartite matching instances as subproblem, which dominates the total running time. We build on recent results by Ramshow and Tarjan [13, 14] for unbalanced matchings. Let T and T' be labeled rooted trees with $k := |T|$ and $l := |T'|$ vertices, respectively, and $\Delta := \min\{\Delta(T), \Delta(T')\}$ the smaller degree of the two input trees. For real-valued weight functions we prove a time bound of $\mathcal{O}(kl\Delta)$. For integral weights bounded by a constant C we prove a running time of $\mathcal{O}(kl\sqrt{\Delta}\log(\min\{k,l\}C))$. This is an improvement over the algorithm by Kao et al. [9] if there are only few labels and the maximum degree of one tree is much smaller than the maximum degree of the other. In addition, we support weights and a linear penalty for skipped vertices.

Moreover, the algorithm by Kao et al. [9] is designed for rooted trees only. A straight forward approach to solve the problem for unrooted trees is to try out all pairs of possible roots, which results in an additional $\mathcal{O}(kl)$ factor. However, our algorithm exploits the fact that there are many similar matching instances using techniques related to [1, 4]. This includes computing additional matchings of cardinality two. For unrooted trees and real-valued weight functions we prove the same $\mathcal{O}(kl\Delta)$ time bound as for rooted trees. This leads to an improvement over the formally best algorithm for solving the maximum common subtree problem, for which a time bound of $\mathcal{O}(kl\,(\Delta + \log d))$ has been proven [4].

2 Preliminaries

We consider finite simple undirected graphs unless stated otherwise. Let $G = (V, E)$ be a graph, we refer to the set of *vertices* V by $V(G)$ or V_G and to the set of *edges* by $E(G)$ or E_G. An edge connecting two vertices $u, v \in V$ is denoted by uv or vu. The *order* $|G|$ of a graph G is its number of vertices. The *neighbors* of a vertex v are defined as $N(v) := \{u \in V_G \mid vu \in E_G\}$. The *degree* of a vertex $v \in V_G$ is $\delta(v) := |N(v)|$, the *degree* $\Delta(G)$ of a graph G is the maximum degree of its vertices. In case of a directed graph (*digraph*) we call its edges *arcs*, denoted by (u, v), i.e., an edge from u to v.

A *path* P is a sequence of pairwise disjoint vertices connected through edges (or arcs) and denoted as $P = (v_0, e_1, v_1, \ldots, e_l, v_l)$, where $e_i = v_{i-1}v_i$ (or $e_i = (v_{i-1}, v_i)$), $i \in \{1, \ldots, l\}$. We alternatively specify the vertices $(v_0, \ldots, v_l)$ or edges $(e_1, \ldots, e_l)$ only. The *length* of a path is its number of edges. A connected graph with a unique path between any two vertices is a *tree*. A tree T with an explicit root vertex $r \in V(T)$ is called *rooted* tree, denoted by T^r. In a rooted tree T^r we denote the set of *children* of a vertex v by $C(v)$ and its *parent* by $p(v)$, where $p(r) = r$. For any tree T and two vertices $u, v \in V(T)$ the *rooted subtree* T^u_v is induced by the vertex v and its descendants related to the tree T^u. If the root r is clear from the context, we may abbreviate $T_v := T^r_v$. We refer to the root of a rooted tree T by $r(T)$.

If the vertices of a graph G can be separated into exactly two disjoint sets V, U such that $E(G) \subseteq V \times U$, then the graph is called *bipartite*. In many cases the disjoint sets are already given as part of the input. In this case we write $G = (V \sqcup U, E)$, where $E \subseteq V \times U$.

For a graph $G = (V, E)$ a *matching* $M \subseteq E$ is a set of edges, such that no two edges share a vertex. An edge $e \in M$ is denoted *matched*. A vertex incident to an edge $e \in M$ is denoted *matched*; otherwise it's *free*. For an edge $uv \in M$, the vertex u is the *partner* of v and vice versa. The *cardinality of a matching* M is its number of edges $|M|$. A *weighted graph* is a graph endowed with a function $w : E \to \mathbb{R}$. The weight of a matching M in a weighted graph is $W(M) := \sum_{e \in M} w(e)$. We call a matching M of a weighted graph G a *maximum weight matching* (MWM) if there is no other matching M' of G with $W(M') > W(M)$. A matching M in G is a *MWM of cardinality* k (MWM_k) if there is no other matching M' of cardinality k in G with $W(M') > W(M)$.

For convenience we define the maximum of an empty set as $-\infty$.

3 Gupta and Nishimura's algorithm

In this section we formally define a *Largest Common Subtree Embedding* (LaCSE) and present a brief overview of Gupta and Nishimura's algorithm to compute such an embedding. The following two definitions are based on [8].

▶ **Definition 1** (Topological Embedding)**.** A rooted tree T is *topologically embeddable* in a rooted tree T' if there is an injective function $\psi : V(T) \to V(T')$, such that $\forall a, b, c \in V(T)$

i) If b is a child of a, then $\psi(b)$ is a descendant of $\psi(a)$.

ii) For distinct children b, c of a, the paths from $\psi(a)$ to $\psi(b)$ and from $\psi(a)$ to $\psi(c)$ have exactly $\psi(a)$ in common.

T is *root-to-root topologically embeddable* in T', if $\psi(r(T)) = r(T')$.

▶ **Definition 2** ((Largest) Common Subtree Embedding; (La)CSE)**.** Let T and T' be rooted trees and S be topologically embeddable in both T and T'. For such a S let $\psi : V(S) \to V(T)$ and $\psi' : V(S) \to V(T')$ be topological embeddings.

- Then $\varphi := \psi' \circ \psi^{-1} : \psi(V_S) \to \psi'(V_S)$ is a *Common Subtree Embedding*.
- If there is no other tree S' topologically embeddable in both T and T' with $|S'| > |S|$, then S is a *Largest Common Embeddable Subtree* and φ is a *Largest Common Subtree Embedding*.
- An CSE with $\varphi(r(T)) = r(T')$ is a *root-to-root CSE*.
- A root-to-root CSE is *largest*, if it is of largest weight among all root-to-root common subtree embeddings.

Algorithm from Gupta and Nishimura. Gupta and Nishimura [8] presented an algorithm to compute the size of a largest common embeddable subtree based on dynamic programming, which is similar to the computation of a largest common subtree, described in, e.g., [12, 4]. Let T and T' be rooted trees and $\mathcal{L}$ be a table of size $|T||T'|$. For each pair of vertices $u \in T, v \in T'$ the value $\mathcal{L}(u, v)$ stores the size of a LaCSE between the rooted subtrees T_u and T'_v. Gupta and Nishimura proved, that an entry $\mathcal{L}(u, v)$ is determined by the maximum of the following three quantities.

- $M_1 = \max\{\mathcal{L}(u, c) \mid c \in C(v)\}$
- $M_2 = \max\{\mathcal{L}(b, v) \mid b \in C(u)\}$
- $M_3 = W(M)+1$, where M is a MWM of the complete bipartite graph $(C(u) \sqcup C(v), C(u) \times C(v))$ with edge weight $w(bc) = \mathcal{L}(b, c)$ for each pair $(b, c) \in C(u) \times C(v)$.

Here, M_1 represents the case, where the vertex v is not mapped. To satisfy ii) from Def. 1, we may map at most one child $c \in C(v)$. M_2 represents the case, where u is not mapped and at most one child $b \in C(u)$ is allowed. M_3 represents the case $\varphi(u) = v$. To maximize

the number of mapped descendants we compute a maximum weight matching, where the children of u and v are the vertex sets $C(u)$ and $C(v)$, respectively, of a bipartite graph. The edge weights are determined by the previously computed solutions, i.e., the LaCSEs between the children of u and v and their descendants, namely between T_b and T'_c for each pair of children (b, c). The algorithm proceeds from the leaves to the roots. From the above recursive formula, we get $\mathcal{L}(u, v) = 1$ if u or v is a leaf, which was separately defined in [8].

A maximum value in the table yields the size of a LaCSE. We obtain the size of a root-to-root LaCSE from M_3 of the root vertices $r(T), r(T')$. Note, with storing $\mathcal{O}(|\mathcal{L}|)$ additional data, it is easy to obtain a (root-to-root) LaCSE φ.

▶ **Theorem 3** (Gupta, Nishimura, [8]). *Computing a LaCSE between two rooted trees of order at most n is possible in time $\mathcal{O}(n^{2.5} \log n)$.*

4 Largest Weight Common Subtree Embeddings

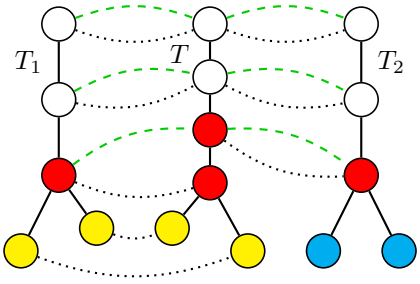
First, we introduce weighted common subtree embeddings between labeled trees. Part of the input is an integral or real weight function on all pairs of the labels. Next, we consider a linear distance penalty for skipped vertices in the input trees. After formalizing the problem and presenting an algorithm, we prove new upper time bounds.

Vertex Labels. In many application domains the vertices of the trees need to be distinguished. A common representation of a vertex labeled tree T is (T, l), where $l : V(T) \to \Sigma$ with Σ as a finite set of labels. Let $\omega : \Sigma \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ assign a weight to each pair of labels. Instead of maximizing the number of mapped vertices, we want to maximize the sum of the weights $\omega(l(u), l(\varphi(u)))$ of all vertices u mapped by a common subtree embedding φ. For simplicity, we will omit l and l' for the rest of this paper and define $\omega(u, v) := \omega(l(u), l'(v))$ for any two vertices $(u, v) \in V(T) \times V(T')$.

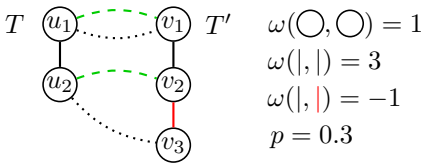
Edge Labels. Although not as common as vertex labels, edge labels are useful to represent different bonds between atoms or relationship between individuals. In a common subtree embedding we do not map edges to edges but paths to paths. Since in an embedding inner vertices on mapped paths do not contribute to the weight, we do the same with edges. I.e., both paths need to have length 1 for their edge labels to be considered. Here again, we want to maximize the weight $\omega(e, e') := \omega(l(e), l'(e'))$ of these edges mapped to each other (additional to the weight of the mapped vertices).

Distance Penalties. Depending on the application purpose it might be desirable that paths do not have an arbitrary length. Here, we introduce a linear distance penalty for paths of length greater than 1. I.e., each inner vertex on a path corresponding to an edge of the common embeddable subtree lowers the weight by a given penalty p. By assigning p the value ∞ we effectively compute a maximum common subtree. The following definition formalizes a LaCSE under a weight function ω and a distance penalty p.

▶ **Definition 4** (Largest Weight Common Subtree Embedding; LaWeCSE). Let (T, l) and (T', l') be rooted vertex and/or edge labeled trees. Let φ be a common subtree embedding between T and T'. Let $\omega : \Sigma \times \Sigma \to \mathbb{R} \cup \{-\infty\}$ assign a weight to each pair of labels. Let $p \in \mathbb{R}^{\geq 0} \cup \{\infty\}$ be a distance penalty. We refer to a path $P = (u_0, e_1, u_1, \ldots, u_k)$ in the tree T corresponding to a single edge in the common embeddable subtree as *topological path*. Let $\varphi(P)$ be the corresponding path $(v_0, e'_1, v_1, \ldots, v_l)$ in T'. Then



(a) Labeled MCS (green dashed lines) and LaCSE (black dotted lines) between T and T_i, $i \in \{1, 2\}$.



(b) The black embedding has weight 1.7, since the vertex v_2 is skipped and therefore the penalty p is applied; the weight between the edges is not added. The green embedding has weight 5; 2 from the vertices, 3 from the topological path (u_1, u_2) mapped to (v_1, v_2).

Figure 1 a) Although 'intuitively' T is more similar to T_1 than to T_2, both MCSs have size 3. However, the LaCSE between T and T_1 has 6 mapped vertices. **b)** Two weighted embeddings; one with a skipped vertex, the other where the edge labels contribute to the weight.

- $\omega_{\mathrm{p}}(P, \varphi(P)) = \omega(e_1, e_1') := \omega(l(e_1), l'(e_1'))$, if $l = k = 1$, or
- $\omega_{\mathrm{p}}(P, \varphi(P)) = -p \cdot (l + k - 2)$, otherwise.
- The *weight* $\mathcal{W}(\varphi)$ is the sum of the weights $\omega(u, \varphi(u))$ of all vertices u mapped by φ plus the weights $\omega_{\mathrm{p}}(P, \varphi(P))$ of all topological paths P.
- If φ is of largest weight among all common subtree embeddings, then φ is a *Largest Weight Common Subtree Embedding* (LaWeCSE).

The definition of root-to-root LaWeCSE is analogue to Def. 2. A closer look at the definition of ω_{p} reveals that each inner vertex (the skipped vertices) on a topological path or its mapped path subtracts p from the embedding's weight. Fig. 1b illustrates two weighted common subtree embeddings.

The dynamic programming approach. To compute a LaWeCSE, we need to store some additional data during the computation. In Gupta and Nishimura's algorithm there is a table $\mathcal{L}$ of size $|T||T'|$ to store the weight of LaCSEs between subtrees of the input trees. In our algorithm we need a table $\mathcal{L}$ of size $2|T||T'|$. An entry $\mathcal{L}(u, v, t)$ stores the weight of a LaWeCSE between the rooted subtrees T_u and T_v' of *type* $t \in \{\curlywedge, \diamond\}$. Type $\curlywedge$ represents a *root-to-root* embedding between T_u and T_v'; $\diamond$ an embedding, where u or v is *skipped*. Skipped in the sense, that at least one of u, v will be an inner vertex when mapping some additional ancestor nodes of u or v during the dynamic programming. For type $\diamond$ we subtract the penalty p from the weight for the skipped vertices before storing it in our table. We obtain the weight of a LaWeCSE and a root-to-root LaWeCSE, respectively, from the maximum value of type $\curlywedge$ and from $\mathcal{L}(r(T), r(T'), \curlywedge)$, respectively. The following lemma specifies the recursive computation of an entry $\mathcal{L}(u, v, t)$.

▶ **Lemma 5.** *Let $u \in V(T)$ and $v \in V(T')$. For $t \in \{\curlywedge, \diamond\}$ let $M_t^T = \max\{\mathcal{L}(b, v, t) \mid b \in C(u)\}$ and $M_t^{T'} = \max\{\mathcal{L}(u, c, t) \mid c \in C(v)\}$. Then*

- *$\mathcal{L}(u, v, \diamond) = \max\{M_\diamond^T, M_\curlywedge^T, M_\diamond^{T'}, M_\curlywedge^{T'}\} - p$*

Let $G = (C(u) \sqcup C(v), C(u) \times C(v))$ be a bipartite graph with edge weights $w(bc) = \max\{\mathcal{L}(b, c, \diamond), \mathcal{L}(b, c, \curlywedge) + \omega(ub, vc)\}$ for each pair $(b, c) \in C(u) \times C(v)$. Then

- *$\mathcal{L}(u, v, \curlywedge) = \omega(u, v) + W(M)$, where M is a MWM on G.*

Proof. Since we defined the maximum of an empty set as $-\infty$, this covers the base case, where one vertex is a leaf, e.g., if u is a leaf, then $\max\{M_\diamond^T, M_\curlywedge^T, M_\diamond^{T'}, M_\curlywedge^{T'}\} = \max\{M_\diamond^{T'}, M_\curlywedge^{T'}\}$. Further, $W(M) = 0$, if G has no positive weight edges. Then $\mathcal{L}(u, v, \curlywedge) = \omega(u, v)$.

The type $\diamond$ represents the case of an embedding between T_u and T'_v which is not root-to-root. From the definition of M_t^T the vertex u is skipped and from the definition of $M_t^{T'}$ the vertex v is skipped, so it is indeed not root-to-root. Since either u or v was skipped, we subtract the penalty p. This ensures we have taken inner vertices of later steps of the dynamic programming into account.

The type $\curlywedge$ implies that u is mapped to v. Each edge in G represents the weight of a LaWeCSE from one child of u to one child of v. A maximum matching yields the best combination which satisfies Def. 1. If a child b of u is mapped to a child c of v, the paths bu and cv have length one. Then from Def. 4 we have to add the weight $\omega(bu, cv)$. Otherwise at least one path has length greater than one and we have to subtract the distance penalty p for each inner vertex. We already did that while computing $\mathcal{L}(b, c, \diamond)$. ◀

Time and space complexity. We next analyze upper time and space bounds. Thereby we distinguish between real- and integer-valued weight functions ω. If we use dynamic programming starting from the leaves to the roots, we need to compute each value $\mathcal{L}(u, v, t)$ only once.

▶ **Theorem 6.** *Let T and T' be rooted vertex and/or edge labeled trees. Let ω be a weight function, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and p be a distance penalty.*

- *A LaWeCSE between T and T' can be computed in time $\mathcal{O}(|T|\,|T'|\Delta\}$ and space $\mathcal{O}(|T|\,|T'|)$.*
- *If the weights are integral and bounded by a constant C, a LaWeCSE can be computed in time $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|, |T'|\}))$.*

We first need to provide two results regarding maximum weight matchings.

▶ **Lemma 7** ([14])**.** *Let G be a weighted bipartite graph containing m edges and vertex sets of sizes s and t. W.l.o.g. $s \leq t$. We can compute a MWM on G in time $\mathcal{O}(ms + s^2 \log s)$ and space $\mathcal{O}(m)$. If G is complete bipartite, we may simplify the time bound to $\mathcal{O}(s^2 t)$.*

If the weights are integral and bounded by a constant, the following result for a minimum weight matching was shown in [7]. We solve MWM by multiplying the weights by -1.

▶ **Lemma 8.** *Let G as in Lemma 7 and the weights be integral and bounded by a constant C. We can compute a MWM on G in time $\mathcal{O}(m\sqrt{s}\log C)$. If G is complete bipartite, we may simplify the time bound to $\mathcal{O}(s^{1.5} t \log C)$.*

Unfortunately, there is no space bound given in [7]. However, since their algorithm is based on flows, the space bound is probably $\mathcal{O}(m)$. If we assume this to be correct, the space bound in Theorem 6 applies to integral weights too.

Proof of Theorem 6. We observe that the entries of type $\curlywedge$ in the table $\mathcal{L}$ dominate the computation time. We assume the bipartite graphs on which we compute the MWMs to be complete. We further observe that each edge bc representing the weight of the LaWeCSE between the subtrees T_b and T'_c is contained in exactly one of the matching graphs.

Let us assume real weights first. $\mathcal{L}$ requires $\mathcal{O}(|T|\,|T'|)$ space. We may also compute each MWM within the same space bound. This proves the total space bound. From Lemma 7 the

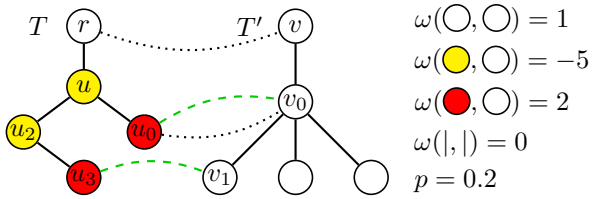


Figure 2 The weight of a LaWeCSE between T^r and T'^v is 2.8 (black dotted lines), since we have one skipped vertex u for penalty 0.2. The weight of a LaWeCSE between T^{u_0} and T'^{v_0} is 3.6 (green dashed lines) for two skipped vertices. The latter one is also a LaWeCSE_u.

time to compute all the MWMs is bounded by

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T}|C(u)|\,|C(v)|\min\{|C(u)|,|C(v)|\}\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}}|C(v)|\sum_{u\in V_T}|C(u)|\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$

Let us assume ω to be integral and bounded by a constant C next. This implies a weight of each single matching edge of at most $\bar{C} := 2C\cdot\min\{|T|,|T'|\}$, since no more than $2\min\{|T|,|T'|\}$ edges and vertices in total may contribute to the weight. Negative weight edges never contribute to a MWM and may safely be omitted. From Lemma 8 the time bound is

$$\mathcal{O}\left(\sum_{v\in V_{T'}}\sum_{u\in V_T}|C(u)|\,|C(v)|\sqrt{\min\{|C(u)|,|C(v)|\}}\log\bar{C}\right)$$
$$\subseteq \mathcal{O}\left(\sum_{v\in V_{T'}}|C(v)|\sum_{u\in V_T}|C(u)|\sqrt{\Delta}\log\bar{C}\right) = \mathcal{O}\left(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|,|T'|\})\right). \quad \blacktriangleleft$$

5 Largest Weight Common Subtree Embeddings for Unrooted Trees

In this section we consider a LaWeCSE between unrooted trees. I.e., we want to find two root vertices $r\in V(T)$, $s\in V(T')$ and a common subtree embedding φ between T^r and T'^s such that there is no embedding φ' between $T^{r'}$ and $T'^{s'}$, $r'\in V(T)$, $s'\in V(T')$, with $\mathcal{W}(\varphi')>\mathcal{W}(\varphi)$. We abbreviate this as LaWeCSE_u. In Sect. 5.1 we present a basic algorithm and a first improvement by fixing the root of T. In Sect. 5.2 we speed up the computation by exploiting similarities between the different chosen roots of T'. In each section we prove the correctness and upper time bounds of our algorithms.

5.1 Basic algorithm and fixing one root

The basic idea is to compute for each pair of vertices $(u,v)\in V(T)\times V(T')$ a (rooted) LaWeCSE from T^u to T'^v and output a maximum solution. This is obviously correct and the time bound is $\mathcal{O}(|T|^2\,|T'|^2\Delta)$.

In our previous work [4] we showed how to compute a maximum common subtree between unrooted trees by arbitrarily choosing one root vertex r of T and then computing MCSs between T^r and T'^s for all $s\in V(T')$. The key idea in the proof is that for any maximum

common subtree isomorphism φ between T and T', either r is mapped by φ, or there is a unique vertex $u \in V(T)$ with shortest distance to r, such that all vertices mapped by φ are contained in T_u. The dynamic programming approach for LaWeCSE already considers the maximum solutions between the rooted subtrees of T^r and T'^s. However, Fig. 2 shows that this strategy alone sometimes fails, if we want to find a LaWeCSE_u between the input trees. A LaWeCSE between T^r and T' rooted at any vertex results in a weight of at most 2.8. In contrast, rooting T at u_0 results in a LaWeCSE of weight 3.6.

In a maximum common subtree between trees T and T', let $r \in V(T)$ be an arbitrarily chosen root of T. If any two children u_1, u_2 of their parent node $u \in V(T)$ are mapped to vertices of T', then u is also mapped. This statement is independent from the chosen root $r \in V(T)$, since a common subtree is connected. If we want to compute a (rooted) LaWeCSE, the statement is also true (for the given root). This follows from Def. 1 ii). However, if we choose u_1 as root in a LaWeCSE_u, we may skip u and map u_2, forming the topological path (u_1, u, u_2). Whatever we do, if we skip vertex u as an inner vertex of a topological path, this is the only path containing u; otherwise we violate Def. 1. We record this as a lemma.

▶ **Lemma 9.** *Let T and T' be unrooted trees. Let φ be a LaWeCSE_u from T to T' and $u \in V(T)$ be an inner vertex of a topological path with its neighbors $N(u) = \{u_1, u_2, \ldots, u_k\}$. Then φ maps vertices from exactly two of the rooted subtrees $T^u_{u_1}, \ldots, T^u_{u_k}$ to T'.*

To compute a LaWeCSE_u φ, additionally to the strategy from [4], we need to cover the case, that there is no single vertex u mapped by φ, such that all vertices mapped by φ are contained in T^r_u, cf. Fig. 2 with r as chosen root. In this case let u be the unique inner vertex of a topological path P, such that all vertices mapped by φ are contained in T^r_u. An example is the yellow vertex u in Fig. 2. Further, let $P_1 = (u_0, \ldots, u_{i-1}, u_i = u, u_{i+1}, \ldots, u_k)$ be the topological path containing u and $\varphi(P_1) = P_2 = (v_0, \ldots, v_l)$ with $\varphi(u_0) = v_0$ and $\varphi(u_k) = v_l$.

Then there is a LaWeCSE ϕ_1 between the rooted subtrees $T^r_{u_{i-1}}$ and $T'^{v_1}_{v_0}$ containing u_0 and all its descendants mapped by φ. There is another LaWeCSE ϕ_2 between the rooted subtrees $T^r_{u_{i+1}}$ and $T'^{v_0}_{v_1}$ containing u_k and all its descendants mapped by φ. Note, choosing $T'^{v_{j+1}}_{v_j}$ and $T'^{v_j}_{v_{j+1}}$ for any $j \in \{0, \ldots, l+1\}$ as rooted subtrees yields the same LaWeCSEs ϕ_1 and ϕ_2, i.e., it does not matter where we split the path P_2.

For any vertex $v \in V(T')$ let $\mathcal{L}^v$ refer to the table $\mathcal{L}$ corresponding to T^r and T'^v. Then $L_1 := \max\{\mathcal{L}^{v_1}(u_{i-1}, v_0, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_1 minus the penalty for the inner vertices $u_1, \ldots, u_{i-1}$; the penalty is 0, if $i = 1$. $L_2 := \max\{\mathcal{L}^{v_0}(u_{i+1}, v_1, t) \mid t \in \{\curlywedge, \diamond\}\}$ is the weight of the LaWeCSE ϕ_2 minus the penalty for the inner vertices $u_{i+1}, \ldots, u_{k-1}$ and $v_1, \ldots, v_{l-1}$. I.e., the penalty p for each inner vertex on the paths excluding u is included in $L_1 + L_2$. Therefore $\mathcal{W}(\varphi) = L_1 + L_2 - p$. Before summarizing this strategy in the following Lemma 10, we exemplify it on Fig. 2.

The LaWeCSE_u φ is depicted by green dashed lines with mapping $u_0 \mapsto v_0$ and $u_3 \mapsto v_1$. The yellow inner vertex u fulfills the condition that T^r_u contains all vertices mapped by φ. We have paths $P_1 = (u_0, u, u_2, u_3)$ and $\varphi(P_1) = P_2 = (v_0, v_1)$. Then $L_1 = \mathcal{L}^{v_1}(u_0, v_0, \curlywedge) = 2$ for the mapping $u_0 \mapsto v_0$. Further $L_2 = \mathcal{L}^{v_0}(u_2, v_1, \diamond) = 1.8$ for the mapping $u_3 \mapsto v_1$ and the skipped vertex u_2. We obtain $\mathcal{W}(\varphi) = L_1 + L_2 - p = 2 + 1.8 - 0.2 = 3.6$.

▶ **Lemma 10.** *Let T and T' be trees. Let $r \in V(T)$ be arbitrarily chosen. Let $\mathcal{W}(r, v)$ be the weight of a LaWeCSE from T^r to T'^v and $\mathcal{T}(u, v, w) = \max\{\mathcal{L}^w(u, v, t) \mid t \in \{\curlywedge, \diamond\}\}$. Then the weight of a LaWeCSE_u is the maximum of the following two quantities.*

- $M_1 = \max\{\mathcal{W}(r, v) \mid v \in V(T')\}$
- $M_2 = \max_{u \in V(T),\ vw \in E(T')}\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\} - p$

▶ **Lemma 11.** *Let the preconditions be as in Lemma 10. Then M_1 can be computed in time $\mathcal{O}(|T|\,|T'|^2\Delta)$ and M_2 in time $\mathcal{O}(|T|\,|T'|)$; both can be computed in space $\mathcal{O}(|T|\,|T'|)$.*

Proof. For any vertices $s, v \in V(T')$ consider the rooted subtree T'^s_v. Let $p(v)$ be the parent vertex of v with $p(s) = s$. Then $T'^s_v = T'^{p(v)}_v = T'^w_v$ for each vertex $w \in V(T') \setminus V(T'^s_v)$, i.e., w is a vertex of T', which is not contained in the rooted subtree T'^s_v. Therefore we can identify each table entry $\mathcal{L}^s(u, v, t)$ by $\mathcal{L}^{p(v)}(u, v, t)$. In other words, all the table entries needed to compute M_1 are determined first by a node $u \in V(T)$, and second by either an edge $wv \in E(T')$ or the root vertex s. Therefore, the space needed to store all the table entries and thus compute M_1 is $\mathcal{O}(|T|\,|T'|)$. We will use these values to retrieve $\mathcal{T}(u, v, w)$ in constant time.

From Theorem 6 for each $v \in V(T')$ we can compute $\mathcal{W}(r, v)$ in time $\mathcal{O}(|T|\,|T'|\Delta)$. Thus, the time for M_1 is bounded by $\mathcal{O}(|T|\,|T'|^2\Delta)$. For any edge $vw \in E(T')$ we observe that the only rooted subtrees from T' to consider are T'^v_w and T'^w_v. Let $L(b, v)$ and $L(b, w)$, $b \in C(u)$ be the weight of a LaWeCSE from T^r_b to T'^w_v and T'^v_w, respectively. Let G be a bipartite graph with vertices $C(u) \sqcup \{v, w\}$ and edges between these vertices with weights defined by $L(b, v)$ and $L(b, w)$, respectively. Let M be a MWM_2 on G. Then $W(M) = \max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$. This follows from the construction of G. Note, a MWM_2 contains exactly 2 edges. Let $C(u) = \{b_1, \ldots, b_k\}$ such that $L(b_i, v) \geq L(b_{i+1}, v)$ for any $i < k$. I.e., the vertices b_i are ordered, such that $L(b_1, v)$ and $L(b_2, v)$ have weight at least $L(b_i, v)$ for all $i > 2$. We remove all edges incident to v except b_1v and b_2v. Analog we remove all but the two edges of greatest weight incident to w. Let G' be the graph with those edges removed and M' be a MWM_2 on G'. We next prove $W(M') = W(M)$. Let M be a matching on G. Assume the partner of v is b_i, $i > 2$. Then let $b = b_1$ if b_1 is not the partner of w, and $b = b_2$ otherwise. Replacing $vb_i \in M$ by vb results in a matching M' such that $W(M') \geq W(M)$. We may argue analog for w.

Since G' contains at most 4 edges we may compute M' in constant time. The time to remove the edges from G to G' is $\mathcal{O}(k)$. Therefore the time to compute $\max\{\mathcal{T}(u_1, v, w) + \mathcal{T}(u_2, w, v) \mid u_1 \neq u_2 \in C(u)\}$ for given u and vw is $\mathcal{O}(|C(u)|)$. The time to compute M_2 is $\mathcal{O}(\sum_{u \in V_T, vw \in E_{T'}} |C(u)|) = \mathcal{O}(|T|\,|T'|)$.

We may compute M_2 from $\mathcal{L}$ and additional space $\mathcal{O}(|T|)$, which is $\mathcal{O}(|T|\,|T'|)$ in total. ◀

5.2 Exploiting similarities

In this section we improve the running time from $\mathcal{O}(|T|\,|T'|^2\Delta)$ to $\mathcal{O}(|T|\,|T'|\Delta)$. To this end, we need to speed up the computation in Lemma 5. Specifically, we exploit similarities between the graphs on which we compute the maximum weight matchings. We further need to speed up the computation of $M_t^{T'}$ related to the root vertices from T'. We have to take special care of the sequence, in which we compute the table entries, to avoid circular dependencies.

Speeding up the dynamic programming approach. In Lemma 5 the recursion computes maximum values among certain table entries. We first include the current root $s \in V(T')$ into the notation. We use the definition of $\mathcal{L}^s$ from Sect. 5.1 refering to the table where s is the root of $V(T')$. Let $u \in V(T)$ and $v \in V(T')$ be the vertices in the current recursion of Lemma 5. For all $t \in \{\curlywedge, \diamond\}$ let $M_t^{T,s} = \max\{\mathcal{L}^s(b, v, t) \mid b \in C(u)\}$ and $M_t^{T',s} = \max\{\mathcal{L}^s(u, c, t) \mid c \in C(v)\}$. From the proof of Lemma 11 we know that $T'^s_v = T'^w_v$

for each vertex $w \in V(T') \setminus V(T'^s_v)$. This implies $M_t^{T,s} = M_t^{T,w}$ and $M_t^{T',s} = M_t^{T',w}$ for all w as before. I.e., it is sufficient to distinguish all the $M_t^{T,s}$ and $M_t^{T',s}$ first by a node $u \in V(T)$, and second by either an edge $wv \in E(T')$ or a single vertex $s \in V(T')$.

This observation allows us to upper bound the time to compute all the $M_t^{T,s}$ by

$$\mathcal{O}\left(\sum_{u\in V_T, wv\in E_{T'}} |C(u)| + \sum_{u\in V_T, s\in V_{T'}} |C(u)|\right) = \mathcal{O}\left(\sum_{wv\in E_{T'}} |T| + \sum_{s\in V_{T'}} |T|\right) = \mathcal{O}\left(|T|\,|T'|\right).$$

Let $N(v) = \{c_1, c_2, \dots c_l\}$ and $C_i := \{c_1, \dots, c_{i-1}, c_{i+1}, \dots, c_l\}$ for $1 \le i \le l$, i.e, C_i contains all the vertices from $N(v)$ except c_i. We observe $M_t^{T',v} = \max\{\mathcal{L}^v(u,c,t) \mid c \in N(v)\}$ and $M_t^{T',c_i} = \max\{\mathcal{L}^{c_i}(u,c,t) \mid c \in C_i\} = \max\{\mathcal{L}^v(u,c,t) \mid c \in C_i\}$ for each $i \in \{1,\dots,l\}$. Let j be an index, such that $M_t^{T',v} = \mathcal{L}^v(u,c_j,t)$. Then for each $i \neq j$ we have $M_t^{T',c_i} = M_t^{T',v}$. Therefore, we may compute $M_t^{T',v}$ and M_t^{T',c_i} for all $i \in \{1,\dots,l\}$ in time $\mathcal{O}(\delta(v))$. Hence, the time to compute all the $M_t^{T',s}$ is bounded by $\mathcal{O}\left(\sum_{u\in V_T, v\in V_{T'}} \delta(v)\right) = \mathcal{O}\left(\sum_{u\in V_T} |T'|\right) = \mathcal{O}(|T|\,|T'|)$.

▶ **Lemma 12.** *Assume there is a sequence of all pairs (u,v) such that all necessary values are available to compute $M_t^{T,s}$ and $M_t^{T',s}$ for $s \in N(v) \cup \{v\}$; then the total time is $\mathcal{O}(|T|\,|T'|)$.*

Exploiting similarities between the matching graphs. In Lemma 5 we need to compute a MWM for each $(u,v) \in V(T) \times V(T')$. When considering all roots $s \in V(T')$, we have one matching graph G with vertices $C(u) \sqcup N(v)$, $N(v) = \{c_1, \dots, c_l\}$ as well as l graphs G_{c_i}, $1 \le i \le l$. This follows analog to the observation regarding $M_t^{T',v}$ from the previous paragraph. A graph G_c, $c \in N(v)$, is the same as G except that the vertex c and incident edges are removed. Let $s := \min\{\delta(u), \delta(v)\}$ and $t := \max\{\delta(u), \delta(v)\}$. We now prove a total time bound of $\mathcal{O}(s^2 t)$ for computing a MWM on G as well as on G_c for all $c \in N(v)$. We distinguish two cases.

i) $s \ge \log t$. In our previous work we presented an algorithm to compute a maximum common subtree of maximum weight, a special case of LaWeCSE_u [4]. A subproblem is to compute MWMs on graphs structurally identical to G and G_{c_i}, $1 \le i \le l$. We showed that we can compute all those MWMs in time $\mathcal{O}(st(s + \log t))$. Under the premise $s \ge \log t$ the time bound is $\mathcal{O}(s^2 t)$.

ii) $s < \log t$. Then one vertex set is much smaller than the other. From Lemma 14 we can compute all those MWMs in time $\mathcal{O}(s^4 + s^2 t)$. Under the premise $s < \log t$ that is $\mathcal{O}(s^2 t)$.

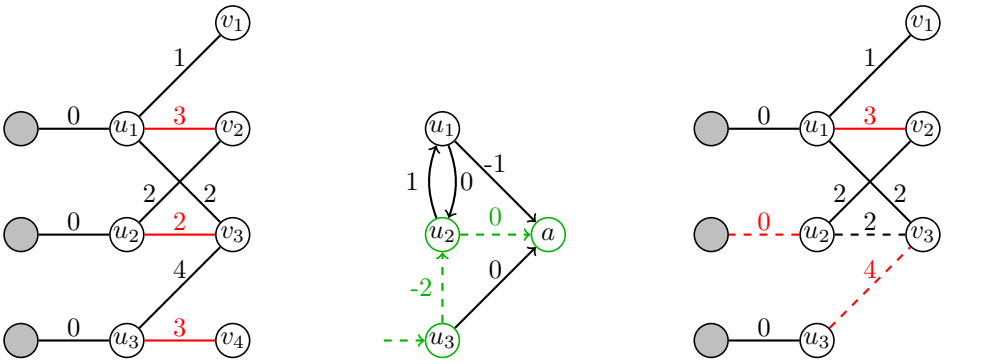
▶ **Lemma 13.** *Assume there is a sequence of all pairs (u,v) such that all necessary values are available to compute the MWMs; then the total time is $\mathcal{O}(|T|\,|T'|\Delta)$.*

Proof. The time to compute all the MWMs is

$$\mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\min\{\delta(u),\delta(v)\}\right) \subseteq \mathcal{O}\left(\sum_{u\in V_T}\sum_{v\in V_{T'}} \delta(u)\delta(v)\Delta\right) = \mathcal{O}(|T|\,|T'|\Delta).$$ ◀

▶ **Lemma 14.** *Let G be a weighted bipartite graph with vertex sets U and V, $s := |U| \le |V| =: t$. Let either $C = U$ or $C = V$. We can compute a MWM on G and a MWM on each graph G_c, $c \in C$, in total time $\mathcal{O}(s^4 + s^2 t)$.*

Proof. From Lemma 7 we know there is an algorithm which computes a MWM on G in time $\mathcal{O}(s^2 t)$. This algorithm first copies the s vertices of U and then adds an edge of weight 0



(a) G' and a MWM M' (red).

(b) $\overline{G'_{v_4}}$ and a shortest path P from v_4's partner u_3 to a (green, dashed lines).

(c) A MWM (red) on G'_{v_4} obtained from a single augmenting path (dashed lines).

Figure 3 **(a)** A MWM$_s$, $s = |U| = 3$, on G', which is also a MWM. **(b)** The graph $\overline{G'_{v_4}}$, on which we compute a shortest path from u_3 to a. Such a path corresponds to a augmenting path of maximal weight in G'_{v_4}. **(c)** Applying the path yields a MWM$_3$ on G'_{v_4}, which is also a MWM.

between each vertex of U and its copy. We denote this graph by G'. The algorithm computes a MWM$_s$ M' on G' (M' is also a MWM), which corresponds to a MWM on G. An example is depicted in Figure 3a.

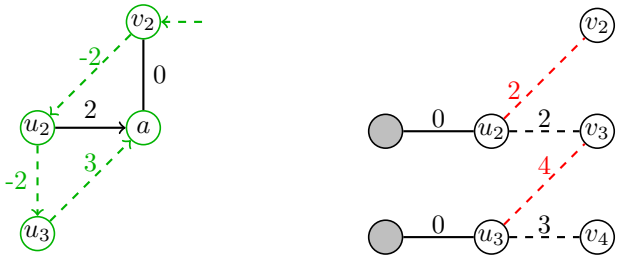
The graph G' with one vertex $c \in C$ removed is denoted by G'_c. If c is not matched, we are done. If c is matched, let u (in case $c \in V$) respectively v (in case $c \in U$) be the partner of c. Let $M'_c := M' \setminus \{cu\}$ or $M'_c := M' \setminus \{cv\}$, respectively. We observe $|M'_c| = s - 1$.

First, assume $c \in V$. In a MWM$_s$ of G'_c each vertex of U including u must be matched. An odd length M'_c-augmenting path P of maximal weight (the path's weight refers to the difference in the matching's weight after augmentation) incident to u yields a MWM$_s$ on G'_c and thus a MWM on G_c. This follows from the fact, that any M'_c-alternating cycle or path on G'_c not incident to u has nonpositive weight; otherwise M' was no MWM. We can find such a path using the Bellman-Ford algorithm in time $\mathcal{O}(st + s^3)$ as follows. Let $\overline{G'_c} = (U \cup \{a\}, A)$ be the digraph, where A is the union of the following two sets of directed arcs.

1. For each alternating path $\bar{u}vu'$ in G'_c, where $\bar{u}, u' \in U, v \in V$, and vu' is matched, we add the arc $(\bar{u}, u')$ with weight $w(vu') - w(\bar{u}v)$.
2. For each vertex $\bar{u} \in U$ let v be a free vertex adjacent to u, such that the edge uv has maximum weight among all such edges. We add an arc (u, a) of weight $-w(uv)$.

The time to construct the graph is bounded by $\mathcal{O}(st)$. Since $\overline{G'_c}$ has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we may compute a shortest path P from c's partner u to a in time $\mathcal{O}(s^3)$. We obtain a MWM on G'_c by augmenting M'_c with the edges that correspond to P in $\overline{G'_c}$. Figure 3b depicts an example for $\overline{G'_c}$, as well as a shortest path P. Figure 3c depicts the resulting MWM. Since at most s vertices of V are matched by M, the total time to compute a MWM on each of the graphs G_c, $c \in V$, is $\mathcal{O}(s^4 + s^2 t)$.

Second, assume $c \in U$. Each vertex in U is matched. Therefore the cardinality of M'_c is $s - 1$. This time we need to find an alternating path of even length (we removed a vertex from U) and of maximal weight incident to c's partner v. Any alternating cycle or path not incident to v cannot augment M'_c to greater weight; otherwise M' was no MWM. This path may have length 0, e.g., if M'_c is a MWM of M'. The total time to compute such a path is $\mathcal{O}(s^3)$ as follows. Let $\overline{G'_c} = (U \cup \{v, a\}, A)$, where A is the union of the following four sets of directed arcs.



(a) $\overline{G'_{u_1}}$ and a shortest path P from u_1's partner v_2 to a (green, dashed).

(b) A MWM (red) on G'_{u_1} (isolated vertices removed) obtained from a single augmenting path (dashed lines).

Figure 4 **(a)** The graph $\overline{G'_{u_1}}$, on which we compute a shortest path from v_2 to a. Such a path corresponds to an augmenting path of maximal weight in G'_{u_1}. **(b)** Applying the path yields a MWM_2 on G'_{u_1}, which is also a MWM.

1. For each edge $vu \in E(G'_c)$, $u \in U$, we add the arc (v, u) with weight $-w(vu)$.
2. For each alternating path $uv'u'$ in G'_c, where $u, u' \in U, v' \in V$, and uv' is matched, we add the arc (u, u') with weight $w(uv') - w(v'u')$.
3. For each matching edge uv', where $u \in U, v' \in V$, we add the arc (u, a) with weight $w(uv')$.
4. We add the arc (v, a) with weight 0.

The time to construct the graph is bounded by $\mathcal{O}(s^2)$. An example is depicted in Figure 4a. Figure 4b depicts the resulting MWM. Since $\overline{G'_c}$ has $\mathcal{O}(s)$ vertices and $\mathcal{O}(s^2)$ edges, we may compute a shortest path P from c's partner v to a in time $\mathcal{O}(s^3)$. Again, P yields the augmenting path in G'_c. Since all the s vertices of U are matched by M, the total time to compute a MWM on each of the graphs G_c, $c \in U$, is $\mathcal{O}(s^4)$. ◀

Sequence of computation. For given vertices $(u, v) \in V(T) \times V(T')$ we denote the matching graph G without removed vertices as *main instance* and the matching graphs G_c as its *sub instances*. Analog for $t \in \{\curlywedge, \diamond\}$ we define $M_t^{T',v}$ as main instance and $M_t^{T',c}$ for each $c \in N(v)$ as its *sub instances*.

Let $u \in V(T)$ and $vw \in E(T')$. We observe, for type $t \in \{\curlywedge, \diamond\}$ the following values depend circularly on each other. To compute $M_t^{T',w}$ recursively for the vertices u, w we need $\mathcal{L}^w(u, v, t)$. Computing $\mathcal{L}^w(u, v, t)$ requires $M_t^{T',v}$ computed recursively for the vertices u, v. The latter one requires $\mathcal{L}^v(u, w, t)$. Finally $\mathcal{L}^v(u, w, t)$ requires $M_t^{T',w}$ computed recursively for the vertices u, w, which was the start of the circular dependency.

We further observe, the MWMs depend on table entries of both types. We may break the dependencies by solving at most one sub instance before solving the main instance, as shown next.

We iterate over all roots $s \in V(T')$ and compute a rooted LaWeCSE between T^r and T'^s as in Lemma 5. During the recursion on vertices (u, v) the following cases may happen.

1. The first instance to compute on (u, v) is a main instance. Then we instantly compute all its sub instances from it.
2. The first instance to compute on (u, v) is a sub instance. Then we compute only the sub instance without deriving it from the main instance.
   - **a.** If the second instance is a main instance, we instantly compute its sub instances.
   - **b.** Otherwise let $c_1 \in N(v)$ and $c_2 \in N(v)$ be the vertices corresponding to the first and second sub instance, respectively. Let us consider table entries first. When

we computed the sub instance corresponding to c_1, all necessary table entries for $M_t^{T',v}$ except $\mathcal{L}^v(u, c_1, t)$ were available. For the second sub instance $\mathcal{L}^v(u, c_1, t)$ is also available. Thus we may instantly compute the main instance and all other sub instances including the one corresponding to c_2. We may argue analog for the MWMs.

▶ **Theorem 15.** *Let T and T' be (unrooted) vertex and/or edge labeled trees. Let ω be a weight function, $\Delta = \min\{\Delta(T), \Delta(T')\}$, and p be a distance penalty. A LaWeCSE$_u$ between T and T' can be computed in time $\mathcal{O}(|T|\,|T'|\Delta)$ and space $\mathcal{O}(|T|\,|T'|)$.*

6 Conclusions

We presented an algorithm which solves the largest weight common subtree embedding problem in time $\mathcal{O}(|T|\,|T'|\Delta)$. For rooted trees of integral weights bounded by a constant we proved a bound of $\mathcal{O}(|T|\,|T'|\sqrt{\Delta}\log(C\min\{|T|, |T'|\}))$. Our approach generalizes the maximum common subtree problem [4] and the largest common subtree embedding problem, both unlabeled [8] and labeled [9], by supporting weights between labels and a distance penalty for skipped vertices.

A remaining open problem is whether the time bound for unrooted trees can be improved when the weights are integral and bounded by a constant. Since weight scaling algorithms for matchings do not work incrementally [13], there is no obvious way to exploit the similarities in the given matching graphs.

References

1. Moon Jung Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987. doi:10.1016/0196-6774(87)90030-7.
2. James Trimble Ciaran McCreesh, Patrick Prosser. A partitioning algorithm for maximum common subgraph problems. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 712–719, 2017. doi:10.24963/ijcai.2017/99.
3. Andre Droschinsky, Nils Kriege, and Petra Mutzel. *Finding Largest Common Substructures of Molecules in Quadratic Time*, pages 309–321. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-51963-0_24.
4. Andre Droschinsky, Nils M. Kriege, and Petra Mutzel. Faster Algorithms for the Maximum Common Subtree Isomorphism Problem. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*, volume 58 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2016.33.
5. Hans-Christian Ehrlich and Matthias Rarey. Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(1):68–79, 2011. doi:10.1002/wcms.5.
6. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
7. Andrew V. Goldberg, Sagi Hed, Haim Kaplan, and Robert E. Tarjan. Minimum-cost flows in unit-capacity networks. *Theory of Computing Systems*, 61(4):987–1010, Nov 2017. doi:10.1007/s00224-017-9776-7.
8. Arvind Gupta and Naomi Nishimura. Finding largest subtrees and smallest supertrees. *Algorithmica*, 21:183–210, 1998. doi:10.1007/PL00009212.

9 Ming-Yang Kao, Tak-Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. An even faster and more unifying algorithm for comparing trees via unbalanced bipartite matchings. *Journal of Algorithms*, 40(2):212–233, 2001. doi:10.1006/jagm.2001.1163.

10 Antoni Lozano and Gabriel Valiente. On the maximum common embedded subtree problem for ordered trees. In *In C. Iliopoulos and T Lecroq, editors, String Algorithmics, chapter 7. King's College London Publications*, 2004.

11 Daniel M. Martin and Bhalchandra D. Thatte. The maximum agreement subtree problem. *Discrete Applied Mathematics*, 161(13-14):1805–1817, 2013. doi:10.1016/j.dam.2013.02.037.

12 David W. Matula. Subtree isomorphism in $O(n^{5/2})$. In P. Hell B. Alspach and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978. doi:10.1016/S0167-5060(08)70324-8.

13 L. Ramshaw and R. E. Tarjan. A weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 581–590, Oct 2012. doi:10.1109/FOCS.2012.9.

14 Lyle Ramshaw and Robert Tarjan. On minimum-cost assignments in unbalanced bipartite graphs, 04 2012. URL: http://www.hpl.hp.com/techreports/2012/HPL-2012-40R1.html.

15 Matthias Rarey and J. Scott Dixon. Feature trees: A new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design*, 12:471–490, 1998. doi:10.1023/A:1008068904628.

16 John W. Raymond and Peter Willett. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, 16(7):521–533, 2002. URL: http://ipsapp008.lwwonline.com/content/getfile/4830/45/6/abstract.htm.

17 Leander Schietgat, Jan Ramon, and Maurice Bruynooghe. A polynomial-time maximum common subgraph algorithm for outerplanar graphs and its application to chemoinformatics. *Annals of Mathematics and Artificial Intelligence*, 69(4):343–376, 2013. doi:10.1007/s10472-013-9335-0.

18 Gabriel Valiente. *Algorithms on Trees and Graphs*. Springer-Verlag, Berlin, 2002.

19 Atsuko Yamaguchi, Kiyoko F. Aoki, and Hiroshi Mamitsuka. Finding the maximum common subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning trees. *Inf. Process. Lett.*, 92(2):57–63, 2004. doi:10.1016/j.ipl.2004.06.019.