On the Approximation of Largest Common Subtrees and Largest Common Point Sets

Tatsuya Akutsu[1] Magnús M. Halldórsson

29 Oct 1997
RH-08-97

Science Institute
University of Iceland
Dunhaga 3, IS-107 Reykjavk
Iceland
mmh@rhi.hi.is

[1]Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108, Japan. tatutsu@ims.u-tokyo.ac.jp

Abstract

This paper considers the approximability of the largest common subtree and the largest common point-set problems, which have applications in molecular biology. It is shown that the problems can not be approximated within a factor of $n^{1-\epsilon}$ in polynomial time for any $\epsilon > 0$ unless $NP \subseteq ZPP$, while a general search algorithm which approximates both problems within a factor of $O(n/\log n)$ is presented. For trees of bounded degree, an improved algorithm which approximates the largest common subtree within a factor of $O(n/\log^2 n)$ is presented. Moreover, several variants of the largest common subtree problem are studied.

1 Introduction

In computational biology and chemistry, there is a frequent need to extract a common pattern from multiple data. A good example is the automatic extraction of a common pattern from multiple amino acid sequences, which has been studied extensively both theoretically and from a practical point of view.

We consider two such applications. First, we consider the case of finding a common structural pattern from multiple molecules. Substructure search, which can be formalized as the subgraph isomorphism problem, is important for database systems in chemistry. For example, it is useful for deciding whether or not a chemical reaction rule can be applied to a chemical structure. Indeed, many methods and systems have been proposed and developed [5, 15]. Relating to substructure search, common substructure search is also important [11, 16]. For example, it is useful for classifying chemical reactions. Assume that structural descriptions of the molecules before and after the reaction are given. Knowing the substructures changed by the chemical reaction is helpful for inferring which type of chemical reaction is applied. Since the changed substructures can be identified if a common substructure is given, common substructure search is useful. For another example, common substructure search is important for finding the relationship between structural patterns and chemical activities because structures which have a common substructure often show similar chemical activities. For common substructure search, several methods have been proposed [11, 16]. In these methods, common substructure search is formalized as a problem of enumerating maximal common subgraphs or finding the largest common connected subgraph.

The other application is finding common substructures of a collection of 3-dimensional protein structures. Many methods have been given for finding common substructures of two protein structures [14, 17]. Russel and Barton have developed a method for multiple protein structures, and have applied it to multiple sequence alignment [14]. However, neither of these two applications have seen many studies from a theoretical perspective.

This paper gives a theoretical analysis of the complexity of finding nearly optimal solutions for the abovementioned problems. The problems are formalized under the names *largest common subtree problem* (LCST) and the *largest common point-set problem* (LCP). Given a collection of trees, possibly with labels on the vertices, LCST is to find a tree of maximum size that is isomorphic to a subtree in each of the input trees. It is a special case of a problem considered in [16]. Although we consider the special case, it is meaningful because a lot of chemical structures have tree-like structures, and a polynomial time algorithm for finding a largest common connected subgraph for fixed number of tree-like structures was proposed [2], which was an extension of the algorithm for trees [13]. Moreover, this special case seems important from a viewpoint of the comparison of evolutionary trees because a similar problem is considered, where homeomorphic subtrees are used for the comparison [9]. We are particularly interested in the bounded-degree case, which is the case in molecular structures. Given a collection of geometric point sets, LCP is to find a set of points that is congruent to a subset of each input point-set. It is closely related to the problem of finding common substructures of multiple protein structures, since 3-dimensional protein structures are frequently treated as point sequences.

1.1 Results

We find that both problems are very hard to approximate, when the number of input sets (trees/point-sets) is unbounded. In particular, it is unlikely that either problem can be approximated within a factor of $n^{1-\epsilon}$ in polynomial time for any $\epsilon > 0$, where n is the size of the smallest input set. Similar results hold for restricted cases of LCST and LCP.

From the positive side, we present a general search algorithm that yields a non-trivial performance ratio of $O(n/\log n)$ for both problems. We further consider several variants of LCST, both in terms of the format of the input trees (e.g. unbounded/bounded number of input trees, labels, and vertex degree; ordered trees), and in terms of the definition used for a subtree (e.g. induced subgraph; required to include the root) and prove hardness and approximability results.

We further give improved algorithms for LCST for the important cases when the degree or the number of labels is restricted. In particular, when the maximum degree is constant, we obtain a performance ratio of $O(n/\log^2 n)$. When the trees are unlabeled, even if the degree is unbounded, we get a ratio of $O(n \log\log n/\log^2 n)$. The same ratio also holds if the number of labels is bounded by $\log^c n$, for some constant c.

We also give some performance ratios in terms of the size *opt* of the optimal solution. These are incomparable to the ratios in terms of n; in particular, they are better whenever $opt < n/\log n$. For bounded-degree (and arbitrary labels) we get a ratio of $O(opt/\log opt)$, and for at most polylogarithmic number of labels $O(opt \log\log opt/\log opt)$. If, further, the trees are *unlabeled*, we get a ratio of $O(opt \log\log opt/\log^2 opt)$ on bounded-degree trees and $O(opt(\log\log opt/\log opt)^2)$ independent of degree. Lower and upper bounds on performance ratios for various cases of LCST are summarized in Table 1.

	Lower Bound	Upper Bound
General instance	$\Omega(n^{1-\epsilon})$	$O(n/\log n)$
bounded-degree	$\Omega(n^{1/4-\epsilon})$	$O(n/\log^2 n)$
Poly-log labels	$\Omega(n^{1/4-\epsilon})$	$O(n \log\log n/\log^2 n)$ $O(opt \log\log opt/\log opt)$
Unlabeled	$\Omega(n^{1/4-\epsilon})$	$O(n \log\log n/\log^2 n)$ $O(opt(\log\log opt/\log opt)^2)$
Few trees	$1+c$	
Few trees of bounded-degree [1]	1	1

Table 1: Summary of results on the approximability of LCST

Let us briefly review related results. LCST for two trees can be solved in polynomial time [6] while LCST for three trees is NP-hard [1]. LCST can be solved in polynomial time if the degree is bounded and the number of input trees is fixed [1]. Approximate and exact matching problems between two point-sets have been studied extensively in computational geometry, but we are not aware of results on larger collections of point-sets.

In the earlier version of the current paper [3] only the $O(n/\log n)$ ratio was given. Recently, Khanna, Motwani and Yao [10] independently developed an approximation algorithm with performance ratio $O(n \log\log n/\log^2 n)$ for trees of bounded degree. They also extended this to trees of unbounded degree with at most poly-log labels, obtaining a ratio of $O(n(\log\log n)^2/\log^2 n)$. The basic ideas are similar to the ones given here, but the implementation has an added complication (only choosing vertices from the same level of the first tree) which causes the performance ratio to be weaker by a factor of $\log\log n$. Also, their deterministic algorithm is a rather expensive derandomization of their randomized method.

2 The hardness of approximating common subtrees

In this section, we consider the largest common subtree problem (LCST), and show that it is hard to compute even approximately.

Trees are assumed to be unordered unless otherwise stated. Two trees are identical if they are isomorphic. The trees may have labels associated with the vertices, and isomorphism must preserve the labels. A tree T is a *subtree* of a tree T' if, when viewed as special types of graphs, T is a connected vertex-induced subgraph of T'. Equivalently, T is a subtree of T' if there is a surjective mapping from $V(T)$ to $V(T')$ that preserves labels and adjacency. The size of a tree and the length of a path are defined to be their number of edges.

LCST is defined formally as follows: Given a collection of vertex-labeled trees $\{T^1, \cdots, T^t\}$, find a common subtree with the maximum number of edges. Let S denote the smallest tree, and let n denote its size. Let t denote the number of trees. Although labels refer to node-labels, the arguments can be modified for trees with edge-labels.

An approximation algorithm for a maximization problem *approximates* the optimal cost *opt within a factor of* $f(n)$ if, for all instances Π of the problem of size n, it produces a solution of size at least $opt(\Pi)/f(n)$ in time polynomial in the size of the input. We say that a problem is *hard to approximate* within a certain factor, if such an approximation would lead to the conclusion $NP \subseteq ZPP$ (i.e. that there exist zero-error randomized algorithms for all problems in NP).

The INDEPENDENT SET problem is that of finding a maximum cardinality set of vertices I in a graph $G = (V, E)$ that are mutually non-adjacent, i.e. satisfying $(\forall v_i, v_j \in I)(\{v_i, v_j\} \notin E)$. Recent work on approximation hardness, based on interactive proof systems, has culminated in the result of Håstad [7] that INDEPENDENT SET is hard to approximate within a factor of $N^{1-\epsilon}$, for any $\epsilon > 0$, where N is the number of vertices.

We shall consider directed (i.e. rooted) trees, while the same argument holds also for undirected trees. First, we consider a simple case when both the maximum degree and the number of labels are unbounded. In this case, we use almost the same reduction as in [8] and [12].

Theorem 2.1 *If* LCST *for trees with unbounded labels and unbounded degree can be approximated within a factor of $f(n)$ (on all instances where the smallest tree is of size n), then* INDEPENDENT SET *can be approximated within a factor of $f(N)$ (on all graphs on N vertices).*

Proof. Given a graph $G = (V, E)$ with $V = \{v_1, \ldots, v_N\}$, we construct a collection of trees $\{T^1, T^2, \ldots, T^{N+1}\}$. The last tree T^{N+1} consists of a root node labeled by 0 and its children w_j^{N+1}, $j = 1, \ldots, N$, labeled by j. Tree T^i, $i = 1, \ldots, N$, consists of a root, two children w^i and u^i, and grandchildren $w_1^i, \cdots, w_N^i$ and $u_1^i, \cdots, u_N^i$. Roots and internal nodes are labeled by 0, while the leaves are labeled by:

$$label(w_j^i) = \begin{cases} -1, & \text{if } (v_i, v_j) \in E, \\ j, & \text{otherwise,} \end{cases} \quad label(u_j^i) = \begin{cases} -1, & \text{if } j = i, \\ j, & \text{otherwise.} \end{cases}$$

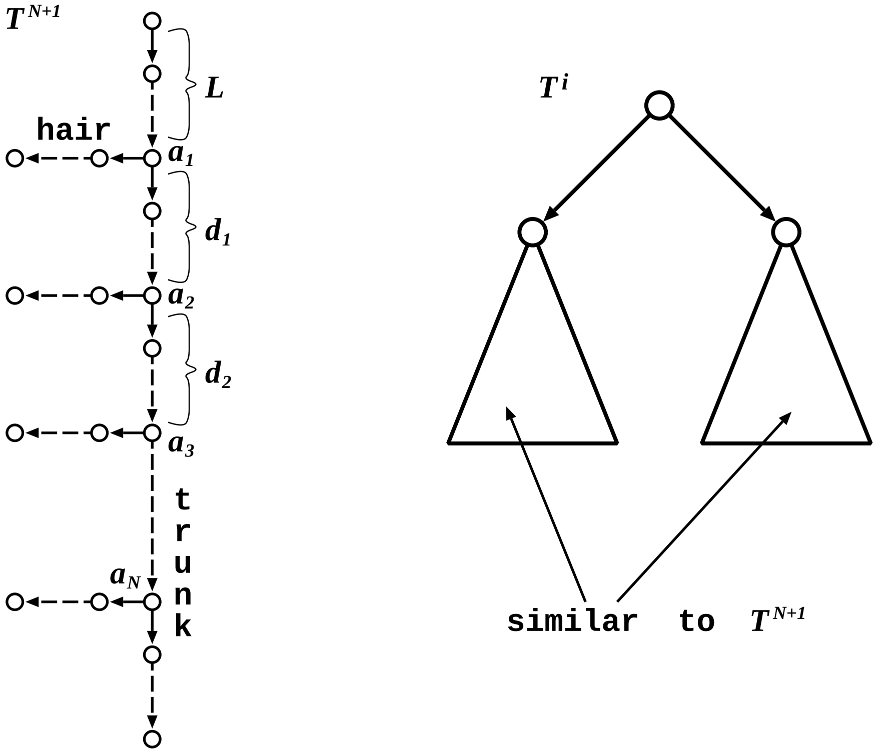

Figure 1: Forms of trees constructed

Intuitively, the labels of vertices w^i_j encode the adjacency list of graph vertex v_i.

Observe that a common subtree T consists of the root of T^{N+1} and a subset of its children. Moreover, the root of T must correspond to either w^i or u^i for each $i \neq N+1$. The leaves of T describe exactly an independent set in the graph. Namely, we can construct an independent set I by:

$$I \;=\; \{v_i \mid w \text{ is a leaf in } T \text{ and } label(w) = i\}.$$

Thus, given a common subtree of size at least k, we obtain an independent set of size k. Furthermore, this can be mapped back; thus, the independence number of the graph is k iff the LCST size is k. Since n, the minimum size of the constructed trees, is N, the theorem follows. $\square$

We obtain a strong hardness result using the result of [7].

Corollary 2.2 LCST *is hard to approximate within a factor of* $n^{1-\epsilon}$, *for any* $\epsilon > 0$.

Next we show that a similar result holds even if trees are unlabeled and of maximum degree three, by modifying the construction of Theorem 2.1. In fact, we can prove this for a still further restricted class of trees.

A *caterpillar* is a binary tree, all of whose degree-three vertices lie on a single path, known as the *trunk*. The paths extending from degree-three vertices are called *hairs*.

Theorem 2.3 LCST *of caterpillars is hard to approximate within a factor of* $n^{1/4-\epsilon}$,*for any* $\epsilon > 0$.

Proof. Given a graph $G = (V, E)$ on N vertices, we construct $N+1$ caterpillars, $T^1, T^2, \ldots, T^{N+1}$. The trunk of T^{N+1} contains degree-three vertices $a_1, a_2, \ldots, a_N$ in order, where a_i and a_{i+1} are separated by a simple path of length $d_i = N^2 + i$. The length of all hairs, and the length of the paths before a_1 and after a_N on the trunk, is $L = N^3$.

For $i \neq N+1$, T^i is obtained by modifications similar to Theorem 2.1. First, construct a tree with a root with two children, with both of them roots of subtrees isomorphic to T^{N+1}. Rename the branching nodes as $b_1^i, b_2^i, \ldots, b_N^i$ in the left subtree and $c_1^i, c_2^i, \ldots, c_N^i$ in the right subtree. Then, delete several hairs from both subtrees: the hair extending from b_j^i is deleted if $(v_i, v_j) \in E$, and the hair extending from c_j^i is deleted if $i = j$.

Let q be the length of the trunk of T^{N+1}, i.e. $q = 2L + \sum_{i=1}^{N-1}(N^2+i) = 2N^3 + (N-1)N(N+1/2)$. Let *opt* denote the size of the maximum common subtree of the trees. Note that n, and the size of all the trees, is $\theta(N^4)$. Let $\alpha(G)$ denote the size of the maximum independent set of G.

Claim 2.4 *$\alpha(G) = k$ iff $opt = kL + q$.*

Assuming this claim, we have that deciding whether $opt \geq N^\delta L + q$ or $opt \leq N^{1-\delta} L + q$ is as hard as deciding whether $\alpha(G) \geq N^\delta$ or $\alpha(G) \leq N^{1-\delta}$. Thus, for some δ, approximating $LCST$ within $\theta(\frac{N^{1-\delta}L+q}{N^\delta L+q}) = \theta(N^{1-2\delta}) = \theta(n^{1/4-\delta/2})$ is *hard*, and thus the theorem follows.

To prove one direction of the claim, consider an independent set $\{v_{j_1}, v_{j_2}, \ldots, v_{j_k}\}$. Take the subtree of T^{N+1} of size $kL + q$ formed by the trunk along with those of the hairs that extend from $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$. This is a common subtree, since the hairs extending from $b_{j_1}^i, b_{j_2}^i, \ldots, b_{j_k}^i$ are intact in T^i when $i \in \{j_1, j_2, \ldots, j_k\}$, and the hairs extending from $c_{j_1}^i, c_{j_2}^i, \ldots, c_{j_k}^i$ are intact in T^i when $i \notin \{j_1, j_2, \ldots, j_k\}$.

For the other direction, we argue that a common subtree with at least two branching nodes has the property that, whenever a branching node matches to some a_j in T^{N+1}, it also matches in each T^i to either b_j^i or c_j^i. This is because the distances between any pair of branching nodes of T^{N+1} are unique. Namely, there are no distinct positive integers i, j, l, m such that $d_i + d_{i+1} \cdots + d_j = d_l + d_{l+1} + \cdots d_m$. We see that a maximum common subtree must contain the whole trunk, as well as all nodes of any hair that it contains. Hence, it's size is exactly $kL + q$, where k is the number of hairs it contains. Suppose the hairs contained extend from $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ in T^{N+1}. It is then easy to verify that $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ must form an independent set in G. Hence, the claim and the theorem follow. $\square$

2.1 Restrictions and variations

Constant number of trees In the case when the number of trees in the input is bounded by a constant, the problem can be solved in polynomial time if the degree is also bounded, even for labeled trees [1]. On the other hand, if the degree is unbounded, the reduction of [1] that proves the NP-hardness for three trees of unbounded degree can be seen to prove that there is some constant larger than one within which it hard to approximate.

Ordering among children Another variation of the input format concerns whether there is an order among the children that must be preserved. This is largely a concern for unlabeled graphs, since an ordering can always be simulated with labels. Again, the previous hardness result holds.

Corollary 2.5 LCST *is no easier to approximate if the order among children must be maintained.*

Rooted trees The other case is a slight modification of the canonical case, where the input graphs are rooted and the subtrees are required to contain the root. The hardness results from the previous section hold equally for this case by a slight modification of the construction. We modify the tree T^{N+1} by adding a new root adjacent only to the old root. The new root will match the roots of the other trees, while the old root will match either of the roots of the T^{N+1}-like subgraph of each T^i. As before, this holds for unlabeled trees of unbounded degree.

Corollary 2.6 LCST *is no easier to approximate if the common subtree must match the roots of the trees.*

3 Algorithms for common subtrees

In this section we consider algorithms that find common subtrees with a guaranteed performance. We first give in Section 3.1 a general-purpose algorithm that yields a performance guarantee for all input trees. Then, improved ratios are given in Section 3.2 for the case of trees of low degree. At last, we give in Section 3.3 incomparable performance ratios that depend on the size of the optimal solution, rather than the size of the input.

Recall that the input to LCST consists of a collection $\{T^1, \cdots, T^t\}$ of trees. Let S denote the smallest tree, and let n be its size.

We introduce some new notation. Let OPT denote some optimal solution (i.e. a maximum common subtree), *opt* its size, and D its maximum degree. Denote $k = \max\{n/opt, 70\}$, and $m = \log_k n$. In the following, $\log n$ denotes $\log_2 n$ and we ignore all ceilings and floors.

3.1 General purpose approximation algorithm

We begin with a general purpose approximation algorithm, which we also apply to LCP in Section 4.

A subset X of vertices in a tree induces a unique subtree containing X, any vertex on a path between two vertices in X, and the edges connecting these vertices into a tree. Namely, there is a unique minimal subtree containing X.

Consider the following algorithm.

> Partition the vertices of the smallest tree S arbitrarily into $n/\log n$ sets, each of size $\log n$. For each subset of each set, check if the subtree of S induced by this vertex subset is a common subtree. Output the largest common subtree found.

The total number of subtrees checked is $n/\log n \cdot 2^{\log n} < n^2$, hence the process runs in polynomial time. At least one of these $n/\log n$ sets contains a subset of the vertices of OPT of size at least $opt/(n/\log n)$ (and thus induces a tree at least that large). That is thus a lower bound on the solution found by the algorithm. We have:

Proposition 3.1 LCST *can be approximated within a factor of $n/\log n$.*

Observe that this algorithm and its performance also applies to the more general case where there is a cost function associated with each label and the objective is to maximize the sum of the labels on the common subtree found.

3.2 Improved approximations for low-degree trees

For trees of bounded degree, we can obtain an improved algorithm. We can also apply it to get an improved approximation for trees of unbounded degree in the case that the number of labels is polylogarithmically bounded. We first describe a simple randomized algorithm that motivates an improved deterministic one.

Two vertices are *distant* if the distance between them is at least $\frac{1}{2}\log_D n$; otherwise, they are *close*. A vertex set X is *dispersed* if every pair of vertices in X is distant.

Randomized algorithm Assume to begin with that we know k, the reciprocal of the fraction of vertices belonging to OPT. We apply the following simple random sampling approach:

> Randomly select a subset X of $m = \log_k n$ vertices from S
> Extend X to a subtree T' of S and verify that T' is a common subtree of the input trees.

It is not hard to see that X is both dispersed and contained in OPT with probability $\Omega(1/n)$. By repeating the sampling $O(n^2)$ times, the success probability can be amplified to $1 - 1/n$. When successful, the size of the induced subtree T' is $\Omega(\log_k n \log_D n)$. The performance ratio is then $O(\frac{n/k}{\log_k n \cdot \log_D n}) = O(\frac{n}{\log^2 n} \cdot \frac{\log D \log k}{k})$, expected and with high probability. This is $O(\frac{n}{\log^2 n})$ for trees of bounded degree.

To get around the requirement of knowing the size of the optimal solution, we can run the algorithm in parallel for the $\log n$ possible values of *opt* that are powers of 2. In fact, it suffices for our purpose to use the $\log \log n$ double powers of 2.

Deterministic algorithm As in the randomized algorithm, assume that we know the value of k within a reasonable precision. We use the following algorithm:

1. Partition S into a collection of at most km forests, each containing between $\frac{n}{km}$ and $2\frac{n}{km}$ vertices. The roots of the trees in each forest are siblings in S.

2. For each tree in each forest, eliminate from further consideration all vertices that are *close* to the root of that tree.

3. Partition the remaining vertices into $2\frac{n}{km}$ groups, with each group containing at most one vertex from each forest.

4. Try all subsets of size $\frac{m}{3}$ of each group, and check if it induces a common subtree.

To partition the tree, we can apply the following greedy method. Determine an internal node v such that the subtree rooted at v contains at least $\frac{n}{km}$ vertices, while none of the subtrees rooted at the children of v reach that size. If the size of the subtree rooted at v is at most $2\frac{n}{km}$, cut this subtree, and make it the first forest in the partition. Otherwise, use any subset of the subtrees rooted at the children of v that is within the size limits as the first forest. Repeat this procedure on the remaining tree, until empty.

Time complexity Steps one through three are easily performed in time linear in the size of the tree. Step four depends on the number of subsets examined, along with the complexity of testing for subtree inclusion.

The number of sets of size $\frac{m}{3}$ in each group is at most

$$\binom{km}{m/3} \approx (3ek)^{\frac{m}{3}} \approx n^{\frac{1}{3}(1+3.03/\log k)}.$$

For some $\delta > 0.005$, this is at most $n^{1/2-\delta}$. Testing if tree X is contained in tree Y as a subtree can be performed in time $|X| \cdot |Y|^{1.5}$ [13] (see [10]). Since we cannot guarantee finding a common subtree of size larger than $\log^2 n$, we can arbitrarily cut each candidate to that size. The total time complexity is then bounded by

$$O(n^{1/2-\delta}) \sum_{i=1}^{t} (\log^2 n)|T^i|^{1.5}.$$

This is $O(n^{1/2-\delta}\nu^{1.5})$, where ν denotes the size of the input (i.e. sum of the sizes of the tree).

Parallel algorithm Steps 2 through 4 are easily performed in parallel. A partitioning into $O(km)$ groups (not necessarily of bounded minimum size) sufficient for Step 1 can be achieved in parallel by the following approach:

> Associate with each vertex v a processor. Count the number d_v of descendants of each vertex v. Let the *level* of v be $\lfloor \frac{d_v}{n/(km)} \rfloor$. Each vertex of non-zero level different from its parent, removes the edge to its parent, and thereby assigning it and all its descendants to separate partitions. This results in at most $2km$ connected components. The vertices of non-zero level of each component are of the same level and lie on a single path. Separate the children of the lowest node from this component; the remainder contains at most as many nodes as the difference in the number of descendents of the highest and the lowest nodes on the path, or $n/(km)$.
>
> The processor assigned to the lowest vertex on the path will group together the subtrees of the children into forests, so as to get a good approximate partitioning. This can be achieved similar to parallel approximate bin packing, e.g. by rounding subtree sizes to powers of two, grouping identical sizes, and sequentially solving the now logarithmic-sized collection. The total number of groups is then $O(km)$.

Analysis

Lemma 3.2 *At most $\sqrt{n}km$ vertices from the optimal solution are removed in Step 2.*

Proof. The roots of the trees of a given forest are all children of the same node v in S. The number of vertices in OPT that are close to v (or adjacent to a close vertex) is at most $D^{(\log_D n)/2} = \sqrt{n}$. Since there are at most km forests, the lemma follows. □

Lemma 3.3 *One of the sets of size $m/3$ tried by the algorithm is a dispersed subset of OPT.*

Proof. Observe that vertices in the same group are dispersed, since each group contains at most one vertex from the same forest and vertices in different forests are distant.

We now argue that some group contains a subset of OPT of size $\frac{m}{3}$, and our exhaustive search must therefore uncover such a set. In Step 3, we partition at least $\frac{n}{k} - \sqrt{n}km = \frac{n}{k}(1 - o(1))$ vertices from OPT to at most $2\frac{n}{km}$ groups. Thus, some group must contain at least

$$\frac{1}{2}\left(m - \frac{k^2m^2}{\sqrt{n}}\right) = \frac{m}{2}(1 - o(1))$$

vertices from OPT, since we may assume that k grows no faster than $O(n^{1/5})$ and m is logarithmic in n. This is at least $\frac{m}{3}$ for sufficiently large values of n. □

The following observation has been used innumerably about the minimum size Steiner tree in an arbitrary metric.

Observation 3.4 *A dispersed set of size s induces a tree of size at least $\frac{s}{2} \log_D n$.*

Thus we have obtained the following bound on the size of the solution found by the algorithm.

Lemma 3.5 *The algorithm finds a solution of size at least $\frac{1}{6} \log_D n \log_k n$.*

This immediately yields a performance ratio for bounded-degree trees.

Theorem 3.6 LCST *can be approximated within a factor of $O(n/\log^2 n)$ on bounded-degree trees.*

Proof. By definition, $opt = O(n/k)$. By Lemma 3.5, the algorithm finds a tree of size $\Omega(\log^2 n/\log k)$ on bounded-degree instances. The ratio is $O(n/\log^2 n \cdot \log k/k) = O(n/\log^2 n)$. □

For trees of high degree, we can use alternative approximation algorithms as suggested in [10].

Theorem 3.7 LCST *can be approximated within a factor of $O(n \log\log n/\log^2 n)$, when the number of labels is bounded by $O(\log^c n)$, for c constant.*

Proof. If D is at most $\log^{c+2} n$, the result follows from Lemma 3.5. Thus, assume D is greater than $\log^{c+2} n$. Then, there is some symbol that appears on at least $\Omega(\log^2 n)$ sibling vertices within OPT, and thus on some node in each input tree. We can detect such an occurrence in polynomial time, by iterating through each internal node of S, and each label. The *star* tree formed by those siblings and their parent is then a common subtree of size $\Omega(\log^2 n)$, yielding a $O(n/\log^2 n)$ performance ratio. □

3.3 Approximation in terms of size of optimal solution

We can also argue incomparable performance ratios, which are functions of the size of the optimal solution, rather than of the input size. The results of the preceding subsections are useful only if the optimal solution is some constant fraction of the whole input, while the following results apply equally well when the size of the optimal solution is, e.g., only a square root of the input size.

Theorem 3.8 LCST *can be approximated within a factor of $O(opt \log\log opt/\log opt)$, when the number of labels is bounded by $O(\log^c opt)$, for some integer c.*

Proof. If D is greater than $\log^{c+2} opt$, then we can find a common star of size $\log^2 opt$, by the argument of Theorem 3.7, for a performance ratio of $opt/\log^2 opt$.

On the other hand, if D is at most $\log^{c+2} opt$, then the height of OPT must be at least $\log opt/\log(\log^{c+2} opt) = \Omega(\log opt/\log\log opt)$. Since there are at most n^2 paths in S, we can try all of them to obtain the largest common path, which is necessarily of length $\Omega(\log opt/\log\log opt)$. Hence, the theorem. □

We now give a stronger ratio in the case of unlabeled trees (or, equivalently, when the number of labels is one).

Recall from Section 2, that a caterpillar is a tree of maximum degree three where all vertices of degree three lie on a single path.

Lemma 3.9 *Let T be a tree on n vertices with maximum degree bounded by D. Then, either T contains a path of length $\log^2 n$, or it contains a caterpillar with at least $\log_D(\sqrt{n}/\log^2 n)$ hairs of length at least $(\log_D n)/2$.*

Proof. Let Q denote the height of T. The lemma is trivially true if Q is greater than $\log^2 n$, thus we assume from now that $Q < \log^2 n$.

We construct a caterpillar as follows. Choose the trunk by taking the root u_1 of T and append it to the path $\langle u_2, u_3, \ldots u_Q \rangle$ obtained recursively from the largest subtree rooted at a child of u_1. For hairs, use the longest path from each u_i to a leaf. A hair is said to be *long* if its length is at least $\frac{1}{2}\log_D n$, and otherwise *short*. Let L denote the number of long hairs.

Let r_i denote the number of vertices in the subtree of T rooted at u_i, i.e. the number of descendants of u_i (including itself). Observe that $r_1 = n$, while $r_Q = 1$.

We claim that $r_{i+1} \geq r_1/D^L - i\sqrt{n}$, for any $i \geq 1$. When the hair extending from u_i is short we have that $r_{i+1} \geq r_i - \sqrt{n}$, since a tree of depth less than $(\log_D n)/2$ contains at most $D^{(\log_D n)/2} = \sqrt{n}$ nodes. On the other hand, when the hair extending from u_i is long, we have that $r_{i+1} \geq (r_i - 1)/D$, since we recursively chose the largest among at most D subgraphs. When we expand these recursive relations, we obtain in the end a term with r_1/D^L, since there are L long hairs, while a term of $\sqrt{n}$ is subtracted at most i times. Hence, the claim. In particular, we have that

$$1 = r_Q \geq n/D^L - Q\sqrt{n}.$$

Since $Q < \log^2 n$, it follows that $L \geq \log_D(\sqrt{n}/\log^2 n)$. Hence, the total size of the caterpillar found is at least $(\log_D n/2) \cdot \log_D(\sqrt{n}/\log^2 n)$. □

Definition 1 *An n-*bounded caterpillar *is a caterpillar with at most $\log n/\log\log n$ branching nodes, and at most $\log^2 n$ vertices.*

From the previous lemma, we can easily deduce the following application to the restricted caterpillar type. Note that $\log_D(\sqrt{n}/\log^2 n) \geq (\log_D n)/3$, for sufficiently large n.

Lemma 3.10 *Consider a tree T on N vertices of degree D, and let $n \geq N$. Then, T contains an n-bounded caterpillar of size $\Omega(\log_D N \cdot \min(\log_D N, \log n/\log\log n))$.*

Proof. Lemma 3.9 guarantees the existence of a caterpillar of one of two types. Either it is a path of length $\log^2 N$, in which case we are done. Otherwise, it has at least $\log_D(\sqrt{N}/\log^2 N)$ hairs of length at least $(\log_D N)/2$ each. Delete any hairs that are in excess of $\log n/\log\log n$. The remaining caterpillar is now n-bounded and satisfies the desired size-bound. □

Lemma 3.11 *The number of n-bounded caterpillars is polynomially bounded in n.*

Proof. Each n-bounded caterpillar can be represented by a sequence $\langle b_0, a_1, b_1, a_2, b_2, \ldots, a_t, b_t \rangle$, with $t = \log n/\log\log n$, where a_i denotes the length of the i-th hair and b_i denotes the length of the trunk to the right of the i-th branching vertex. Each a_i, b_i is at most $\log^2 n$, and can be represented with $2\log\log n$ bits. Hence, the full representation uses at most $2\log\log n \cdot (\log n/\log\log n) = 2\log n$ bits. □

Theorem 3.12 *LCST can be approximated within a factor of $O(opt \log\log opt/\log^2 opt)$ on bounded-degree unlabeled trees, and within a factor of $O(opt(\log\log opt/\log opt)^2)$ on arbitrary unlabeled trees.*

Proof. The algorithm is simply to try all n-bounded caterpillars (or all contained in the smallest tree), and output the largest one that is a common subtree.

Let D be the maximum degree of the optimal common subtree. By Lemma 3.9, the optimal common subtree includes a bounded caterpillar of size $\Omega(\log_D opt \cdot \min(\log_D opt, \log n / \log\log n)) = \Omega(\log_D opt \cdot \log opt / \log\log opt)$. Thus, for bounded-degree instances, the performance ratio is $O(opt \log\log opt / (\log opt)^2)$.

If the optimal solution is of degree more than $\log^2 opt$, then we can find a star of this size. Otherwise, we apply the above approach. In either case, the solution is at least $\Omega((\log opt / \log\log opt)^2)$, for a performance ratio of $O(opt(\log\log opt / \log opt)^2)$. □

Finally, we observe that for the special case where the inputs are caterpillars, as in the hardness proof of Theorem 2.3, we obtain a $O(\sqrt{opt})$ approximation by finding the largest common subpath.

4 Largest common edge subgraph problem

We now consider a different largest common subtree problem, based on a variation of the definition of the term *subtree*. The canonical definition that we have used is that of a tree induced by a set of vertices/edges. Because the input graphs are also trees, this coincides both with the definition of an *induced connected subgraph* as well as with subtrees uniquely obtained by specifying only the leaves. Another natural version asks for an induced – not necessarily connected – subgraph, i.e. a forest.

The *subgraph induced by a set of edges* E' contains E' as an edge set and all vertices incident on edges in E' as a vertex set. A graph $G' = (V', E')$ is a *common edge subgraph* if, for each graph G_i (here, tree) in the input, there is a subset $E_i \subseteq E(G_i)$ such that G' is isomorphic to the subgraph induced by E_i. The LARGEST COMMON EDGE SUBGRAPH problem is that of finding a common subgraph with the largest number of edges.

In the case of unlabeled trees, the following yields a fast approximation algorithm. Compute a maximum matching of each tree and use the smallest of these as a common edge subgraph.

Let Δ be the maximum degree in all the input. We claim that there always exists a matching in all the input trees of size $(n-1)/\Delta$. This follows from a greedy matching: pick any edge incident on a leaf, and remove it and all incident edges, and iterate. At most Δ edges are eliminated in each step, and there are $(n-1)$ edges to start with, so there must be at least $(n-1)/\Delta$ edges chosen.

This implies an *absolute performance ratio* of Δ. Similarly, we can also claim a (relative) performance ratio of D, where D is the maximum degree of the optimal solution, since in each iteration of the greedy algorithm, at most D edges are eliminated from the optimal solution.

We can extend this to labeled trees, by partitioning the trees into edge sets for each combination of labels incident on an edge. Notice that there are $\binom{L+1}{2}$ distinct pairs of not necessarily distinct labels.

Theorem 4.1 *The* LARGEST COMMON EDGE SUBGRAPH *problem can be approximated within a factor of* $\binom{L+1}{2}D$*, where* L *is the number of labels and* D *is the maximum degree of the optimal solution.*

In the case when the number of trees, the number of labels, and the degree are all unbounded, we can prove a strong hardness result by using the same construction of trees as in the proof of Theorem 2.1.

Theorem 4.2 *The* LARGEST COMMON EDGE SUBTREE *problem is hard to approximate within a factor of* $n^{1-\epsilon}$*, for any* $\epsilon > 0$.

4.1 Common edge subgraph of few trees

We now turn our attention to the case when we are given simply two trees. Recall that LCST was polynomial solvable in this case. The COMMON EDGE SUBTREE problem, however, is hard to compute exactly for even the simplest of inputs.

Theorem 4.3 *The* LARGEST COMMON EDGE SUBTREE *problem is NP-hard for two paths with only two labels, as well as for two unlabeled trees.*

Proof. We reduce from 3-PARTITION [6]:

> *Instance:* An integer B and a multiset $A = \{a_1, \cdots, a_{3m}\}$ of integer numbers such that $(\forall i)(B/4 < a_i < B/2)$ and $\sum_i a_i = mB$.
> *Question:* Can A be partitioned into m disjoint multisets $S_1, S_2, \cdots, S_m$ such that $(\forall i)(\sum_{a \in S_i} a = B)$?

First, consider the paths P and Q on $m(B+1)$ and $mB + 6m$ vertices, respectively, whose sequences of labels are given by the following two binary strings:

$$\begin{aligned} P &: (1^B 0)^m, \\ Q &: 1^{a_1} 001^{a_2} 00 \cdots 1^{a_{3m}} 00. \end{aligned}$$

We claim that there exists a 3-partition of A iff there is a common edge-subgraph of P and Q with $(B-2)m$ edges. Let us refer to each substring of P of the form $1^B 0$ as a *block*.

Suppose there is a 3-partition. Then there exists a common subgraph containing all the vertices of P and missing exactly two edges from each of the m blocks as well as the edges connecting the blocks. The three strings from Q matched to a block in P correspond to the three integers assigned to a given multiset that sum to B.

On the other hand, any common subgraph must miss at least two edges separating the 1's from each block in P, since there is no substring of labels of Q with $B/2$ ones in it. Also, at least one of the edges incident on each '0' but the last must be missing. A common subgraph missing only $3m - 1$ edges similarly results in a 3-partition. Hence, we have proved that the problem is NP-hard for two labeled paths.

An *spider* is a tree with exactly one node of degree greater than two. The paths extending from the high-degree center are called hairs. Let T_1 be a spider with m hairs, each with $B+3$ edges. Let T_2 be an extended star with $3m$ hairs, where the ith hair has $a_i + 1$ edges. Note that both T_1 and T_2 have $m(B+3)$ edges.

A 3-partition of A yields a common subtree on all the vertices of T_1 with only $2m$ edges missing, or 2 per hair. We claim that any common subtree must miss at least $2m$ edges. Suppose that only one edge was missing on a given hair in T_1. The larger remainder of the hair must contain at least $B/2$ edges, which can then only be matched to two hairs of T_2 including the root. Hence, $3m - 2$ of the edges incident on the root of T_2 must be eliminated. Thus, there is a common subtree missing only two edges per hair in T_1 iff there is a 3-partition of A. □

On the positive side, we can give a constant factor approximation.

Theorem 4.4 *The* LARGEST COMMON EDGE SUBTREE *problem of two trees can be approximated within a factor of 4.*

Proof. Given two trees T_1 and T_2, partition the edges of each into two sets, corresponding to edges at even and odd levels (distance from root). Namely, we obtain forests T_1^o, T_1^e, T_2^o, and

T_2^e such that T_i^o and T_i^e partition the edges of T_i, $i = 1, 2$. Further, these forests are of height 1, i.e. a collection of stars.

For each pair (T_1^a, T_2^b), $a, b =$ "o", "e", we compute a maximum common edge subset. This can be done by forming a weighted complete bipartite graph where nodes correspond to the connected components (stars) of the forests and weight of edges correspond to the number of labeled edges that match among the two stars. An optimal solution then is obtained via a maximum weighted matching of this graph, and we output the best of solution from the four pairs.

Our approximate solution will be the best of the common edge subsets computed. The optimal solution is contained in both T_1 and T_2, so at least one fourth of it is contained in one of the four pairs of partitions of T_1 and T_2. Since the algorithm finds the optimal solution for each pair, we obtain a performance ratio of 4. $\square$

This can be generalized to the case of t trees. When matching t distinct star collections, we must solve a weighted t-dimensional matching problem. Although it is NP-hard, we can compute a t-approximate solution by a natural greedy approach in time $O(n^t)$. Then, we obtain a $t2^t$ approximation for the case of t trees.

5 Largest common point-set problem

We now consider the largest common point-set problem (LCP), defined as follows: given a collection of D-dimensional point-sets $SS = \{S^1, \cdots, S^t\}$, find a common point-set with the maximum number of elements, where a point-set C is a *common point-set* of SS if C is congruent to a subset of each S^i. We assume D is fixed, and let n denote the size of the smallest input point-set. First, we show that LCP is hard to approximate, even on the real line.

Theorem 5.1 *If* LCP *can be approximated within a factor of $f(n)$, then* INDEPENDENT SET *can be approximated within a factor of $f(n)$.*

Proof. The reduction is similar to that of Theorem 2.1.

Given a graph G on N vertices, we construct a collection SS of point-sets $\{S^1, S^2, \ldots, S^{N+1}\}$. Let L_1 be a number at least N^2 and L_2 be a number at least $3NL_1$. Corresponding to each vertex v_j, we define two points w_j, u_j on the real line by: $w_1 = 0$, $w_j = w_{j-1} + L_1 + j$ for $1 < j \leq N$, and $u_j = L_2 + w_j$.

Let $S^{N+1} = \{w_1, \cdots, w_N\}$, and, for $1 \leq i \leq N$,

$$S^i = \{w_j \mid \{v_i, v_j\} \notin E\} \bigcup \{u_j \mid j \neq i\}.$$

Note that the only transformations that map a subset of S^i with at least two elements to a congruent subset of S^{N+1} are $x \mapsto x$ and $x \mapsto x - L_2$, i.e. the identity mapping, and the subtraction of L_2 of all the elements. Then, a common point-set $SS = \{w_{i_1}, w_{i_2}, \ldots, w_{i_m}\}$ can be seen to correspond to an independent set $v_{i_1}, v_{i_2}, \ldots, v_{i_m}$, and vice versa. Thus, G has an independent set of size m if and only if there exists a common point set of size m. Moreover, we can construct an independent set of size m in polynomial time given a common point-set of size m. Finally, note that n, the size of the smallest point-set, is N. $\square$

Corollary 5.2 LCP *is hard to approximate within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$.*

While LCP normally requires exact correspondence, the above result holds even if a small gap is allowed between corresponding points. In addition, we can easily obtain the same results for two variants. In one, the point-sets are ordered and the order in sequences must be preserved. In the other, instead of point-sets we are given graphs where each vertex is a point in D-dimensional space and each edge is a line segment connecting its endpoints.

In spite of this hardness result, LCP can be solved in polynomial time if the number of point-sets t is bounded. Consider all combinations of $D+1$ elements from each set, which, if matching, uniquely defines an isometric transformation of the point-sets, and count the points that match. This runs in time polynomial in the size of the input, where the degree depends on t and D [4].

Theorem 5.3 LCP *can be solved in polynomial time if the number of point-sets is bounded.*

Note that LCP can be approximated within a factor of $O(n/\log n)$ by the general algorithm described in Section 3 even if t is unbounded.

6 Conclusion

We have considered the approximability of LCST and LCP, and obtained lower and upper bounds of the approximation ratios. Although we have focused on LCST of unbounded number of input trees, it would be interesting to study LCST of bounded number of input trees. In particular, it is left as an open problem whether the performance ratio can be reduced if the number of trees is a constant but the vertex degrees are unbounded.

Although we have presented hardness results, it is important and may be possible to develop practical algorithms. For example, efficient algorithms which work well in the average case, or on inputs of moderate size ($n \approx 100 \sim 1000$) would be of interest. We hope that the research reported here will guide the search for such algorithms.

References

[1] T. Akutsu, An RNC algorithm for finding a largest common subtree of two trees, *IEICE Transactions on Information and Systems* **E75-D** (1992) 95–101.

[2] T. Akutsu, A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **E76-A** (1993) 1488–1493.

[3] T. Akutsu and M. M. Halldórsson, On the approximation of largest common point sets and largest common subtrees, In *Proc. Fifth Intl. Symp. on Algorithms and Computation*, Lecture Notes in Computer Science, Vol. 834 (Springer, Berlin, 1994) 405–413.

[4] H. Alt, K. Melhorn, H. Wagener and E. Welzl, Congruence, similarity, and symmetries of geometric objects, *Discrete and Computational Geometry* **3** (1988) 237–256.

[5] S. Anderson, Graphical representation of molecules and substructure-search queries in MACCS, *Journal of Molecular Graphics* **2** (1984) 8–90.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability* (Freeman, New York, 1979).

[7] J. Håstad, Clique is hard to approximate within $n^{1-\epsilon}$, In *Proc. 37th IEEE Symp. on Foundations of Comput. Sci.* (1996) 627–636.

[8] T. Jiang and M. Li, On the approximation of shortest common supersequences and longest common subsequences, *SIAM J. Computing* **24** (1995) 1122–1139.

[9] D. Keselman and A. Amir. Maximum agreement of subtree in a set of evolutionary trees - metrics and efficient algorithms, In *Proc. 35th IEEE Symp. on Foundations of Comput. Sci.* (1994) 758–769.

[10] S. Khanna, R. Motwani, and F. F. Yao, Approximation algorithms for the largest common subtree problem, Technical report, Stanford University, 1995.

[11] J. J. Mcgregor, Backtrack search algorithms and the maximal common subgraph problem, *Software-Practice and Experience* **12** (1982) 23–34.

[12] R. Maier, The complexity of some problems on subsequences and supersequences, *J. ACM* **25** (1978) 322–336.

[13] S. W. Reyner, An analysis of a good algorithm for the subtree problems, *SIAM J. Computing* **6** (1977) 730–732.

[14] R. B. Russel and G. J. Barton, Multiple protein sequence alignment from tertiary structure comparison: Assignment of global and residue confidence levels, *PROTEINS: Structure, Function, and Genetics* **14** (1992) 309–323.

[15] R. E. Stobaugh, Chemical substructure searching, *Journal of Chemical Information and Computer Sciences* **25** (1985) 271–275.

[16] Y. Takahashi, Y. Satoh, H. Suzuki and S. Sasaki, Recognition of largest common structural fragment among a variety of chemical structures, *Analytical Sciences* **3** (1987) 23–28.

[17] G. Vriend and C. Sander, Detection of common three-dimensional substructures in proteins, *PROTEINS: Structure, Function, and Genetics* **11** (1991) 52–58.