Approximation Algorithms for the Largest Common Subtree Problem

Sanjeev Khanna*
Department of Computer Science
Stanford University

Rajeev Motwani†
Department of Computer Science
Stanford University

Frances F. Yao
Palo Alto Research Center
Xerox Corporation

ABSTRACT

The *largest common subtree* problem is to find a largest tree which occurs as a common subgraph in a given collection of trees. Let n denote the number of vertices in the largest tree in the collection. We show that in the case of bounded degree trees, it is possible to achieve an approximation ratio of $O(n(\log\log n)/\log^2 n)$. For unbounded degree trees, we give an algorithm with approximation ratio $O(n(\log\log n)^2/\log^2 n)$ when the trees are unlabeled. An approximation ratio of $O(n(\log\log n)^2/\log^2 n)$ is also achieved for the case of labeled unbounded degree trees provided the number of distinct labels is $O(\log^{O(1)} n)$.

*Supported by an OTL grant, NSF Young Investigator Award CCR-9357849, and a Schlumberger Foundation Fellowship.

†Supported by Alfred P. Sloan Research Fellowship, IBM Faculty Development Award, an OTL grant, and NSF Young Investigator Award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

1 Introduction

The problem of finding largest common substructures often arises in chemistry and computational biology. Such substructures, for instance, may help explain the common properties shared by a group of compounds (see e.g. [8]). We study a restricted version of this problem, namely, that of searching for the largest common subtree in a given collection of trees. We refer to this problem as the *largest common subtree* (LCST) problem.

The LCST problem for two trees is known to be in both P and RNC, while it is NP-hard for three or more trees [1]. It is also known that the LCST problem for a fixed number of bounded degree trees is in NC [1]. Recently, Akutsu and Halldórson [2] showed that there exists a constant $\epsilon > 0$ such that the LCST problem is hard to approximate within a factor of n^ϵ on an unbounded collection of trees independent of whether the vertices are labeled or not (here n denotes the maximum number of vertices in any tree of the given collection). These results hold even when the vertex degrees are bounded and the trees are ordered. They also showed that the LCST problem is MAX SNP-hard when the number of trees is bounded but the vertex degrees are unbounded. On the positive side, it was shown that the LCST problem can be approximated to within a factor of $O(n/\log n)$.

The focus of this paper is to present improved approximation algorithms for the LCST problem. Specifically, we show that in the case of bounded degree trees we can achieve an approximation ratio of $O(n(\log\log n)/\log^2 n)$. For unbounded degree trees, we give an algorithm with an approximation ratio of $O(n(\log\log n)^2/\log^2 n)$ if the trees are unlabeled. In the case of labeled, unbounded degree trees, we can achieve the same approximation ratio when the number of distinct labels is $O(\log^{O(1)} n)$. The approximation guarantees achieved in this paper are independent of the number of trees in the input collection. It would be interesting to see if these guarantees can be improved when there is a restriction on the number of input trees; of particular interest is the case when the number of input trees is a constant but the vertex degrees are unbounded.

2 Preliminaries

We are given a collection $\mathcal{C} = \{T_1, T_2, \ldots, T_m\}$ of trees such that each tree contains at most n vertices. The vertices in the tree may have labels associated with them. The LCST problem is that of determining the largest tree which occurs as a subgraph in each $T_i \in \mathcal{C}$.

We say that an algorithm approximates a maximization problem (such as the LCST problem) to a factor $f(N)$ if on any instance of size N, it outputs a solution such that the ratio of the optimal solution value to this solution value is at most $f(N)$.

3 The Bounded Degree Case

We first consider the case when the maximum degree of any vertex is bounded by a constant, say Δ. We show that there is a simple randomized algorithm that achieves an approximation ratio of $O(n(\log\log n)/\log^2 n)$; subsequently, we indicate how this algorithm may be derandomized.

Let OPT and |OPT| respectively denote an optimal solution and the size of an optimal solution. If $|\mathrm{OPT}| \leq n/\log^2 n$, then the desired approximation ratio is trivially achieved.

Similarly, if the height of OPT is $\Omega(\log^2 n)$, then there is a simple algorithm to achieve the desired approximation: exhaustively compute all $O(n^2)$ pairwise paths in T_1; verify the ones which occur as a subgraph in each of the $T_i \in \mathcal{C}$; and, output the largest one. Without loss of generality, we assume from now on that:

- $|\text{OPT}| \geq n/\log^2 n$, and
- the height of OPT is at most $\log^2 n$.

We also assume that both OPT and T_1 are rooted at the same vertex r, since we can exhaustively try all possible vertices of T_1 as the root for OPT, and output the best solution among them.

Definition 1 [Dense Level] *A level of vertices in T_1 is called* dense *if it contains at least $n/\log^4 n$ vertices of OPT.*

Lemma 1 *There exists a dense level in T_1.*

Proof: All vertices in OPT occur in the first $\log^2 n$ levels of T_1. The result follows by an application of the pigeonhole principle to the first $\log^2 n$ levels of T_1. ■

Given two vertices x, y in T_1, let $LCA(x, y)$ denote their least common ancestor in T_1.

Definition 2 [Scattered Set] *A set of vertices U from a given level of the tree T_1 is called* scattered *if for any two vertices x and y in U, $LCA(x, y)$ is at a distance $h \geq \log n/(2\log\Delta)$ from x and y.*

Lemma 2 *Let U be a set of size $k = \log n/(5\log\log n)$ chosen uniformly at random from the vertices in L. Then, with probability $\Omega(1/n)$, U consists entirely of vertices from* OPT *and is a scattered set.*

Proof: Consider a vertex x at level L and let Y be the set of all vertices in L such that $LCA(x, y)$ for $y \in Y$ is at distance $h < \log n/(2\log\Delta)$ from x and y. Since the degree of each vertex is bounded by Δ, it is easy to verify that $|Y| \leq \sqrt{n} - 1$. Thus if p denotes the probability that a randomly chosen set U of size k is scattered, then

$$p \geq \left(\frac{n/\log^4 n}{n}\right) \times \left(\frac{n/\log^4 n - \sqrt{n}}{n-1}\right) \times \ldots \times \left(\frac{n/\log^4 n - (k-1)\sqrt{n}}{n-(k-1)}\right).$$

It is easy to verify that for any $i = O(\log n)$ and sufficiently large n,

$$\left(\frac{n/\log^4 n - i\sqrt{n}}{n-i}\right) \geq \frac{1}{2\log^4 n},$$

and therefore, for $k = \log n/(5\log\log n)$,

$$p \geq 2^{-k(4\log\log n + 1)} \geq \frac{1}{n}.$$

■

Definition 3 [Subtree Extension] *Given a collection of vertices U in T_1, the subtree extension of U, denoted by $\mathcal{T}_U$, is the unique smallest subtree in T_1 which contains every vertex in U.*

Lemma 3 *Let U be any scattered subset of vertices from a given level of the tree T_1. Then $\mathcal{T}_U$, the subtree extension of U, contains at least $\Omega((|U|-1)\log n)$ vertices.*

Proof: We can construct $\mathcal{T}_U$ iteratively by adding vertices from U one at a time. Each newly added vertex contributes a path P to the partial subtree $\mathcal{T}_i$ constructed so far. But observe that P must comprise of at least $\log n/(2\log\Delta)$ vertices not in $\mathcal{T}_i$; otherwise there would be a pair of vertices x, y in U with $LCA(x, y)$ at a distance $h < \log n/(2\log\Delta)$, contradicting the assumption that U is scattered. ■

3.1 A Randomized Algorithm

Our randomized algorithm is now easy to state:

A). Select a vertex in T_1 as the root. We will try all possible vertices as candidates for the root; there are at most n such choices.

- Assume from now on that T_1 is rooted at the same vertex as OPT.

B). Select a level in T_1 as a candidate for being a dense level. As before, we will try all possible choices; only the first $\log^2 n$ levels need be considered since we assume that OPT and T_1 are rooted at a common vertex and the height of OPT is at most $\log^2 n$.

- Assume from now on that L is a dense level.

C). Randomly choose a subset U of $k = (\log n)/(5\log\log n)$ vertices from L. By Lemma 2, with probability $\Omega(1/n)$ the subset U contains only vertices from OPT and is a scattered set. The probability of success can be suitably amplified by repeating the trials.

D). Compute the subtree extension of U in T_1, namely $\mathcal{T}_U$, and verify that it is indeed a common subtree of $\{T_1, T_2, ..., T_m\}$.

3.2 The Analysis

Assuming that the algorithm has made the right choices in steps (A) and (B), it is easy to see that it outputs a solution of size $\Omega((\log^2 n)/\log\log n)$.

The running time is simply given by $\#A \times \#B \times \#C \times \#D$, where $\#S$ denotes the number of times step S gets executed; the steps are nested with (A) being the outermost and (D) being the innermost.

Lemma 4 *Given a set of vertices U in a rooted tree T such that each vertex is at distance at most h from the root, the subtree extension of the set U in T can be computed in $O(|U|h)$ time.*

Proof: This can be done by a simple algorithm that iteratively constructs the subtree extension. Suppose we have the subtree extension of a subset of vertices in U and let r' be the root of this tree. When we add another vertex x from U, we simply need to add the vertices along the paths from x to $LCA(x, r')$ and from r' to $LCA(x, r')$ to update the current subtree and the root r'. Since $LCA(x, r')$ and these paths can be computed in $O(h)$ time by simply following the parent pointers from each of the vertices x and r', the lemma follows. ■

The following lemma was established in [7].

Lemma 5 *Given two rooted trees S and T, it can be determined in time $O(st^{1.5})$ if S occurs as a subtree of T, where s and t denote the number of vertices in S and T respectively.*

Using Lemmas 4 and 5, we know that each execution of step (D) takes

$$O\left(\left(\frac{\log n}{\log\log n} \times \log^2 n\right) \times \left(\frac{\log^2 n}{\log\log n} \times n^{1.5} \times m\right)\right)$$

time. Steps (A) and (B), within which step (D) is nested, are performed a total of $O(n \log^2 n)$ times. Finally, the number of executions of (C) depends on the desired probability of error. Suppose we want the probability of error to be at most $O(1/n)$; it suffices to execute (B) a total of $O(n \log n)$ times. Thus the total running time will be $O((mn^{3.5} \log^8 n)/(\log\log n)^2)$.

Theorem 1 *If $|\mathrm{OPT}| \geq n/\log^2 n$, then in $O((mn^{3.5} \log^8 n)/(\log\log n)^2)$ time, we can determine a common subtree of size $\Omega((\log^2 n)/\log\log n)$ with probability of failure $O(1/n)$.*

3.3 Derandomization

The algorithm of Section 3.1 uses randomization only in step (C). Since $\Omega(\log^2 n/\log\log n)$ random bits are used, the naive derandomization of exhaustively searching through the probability space would take super-polynomial time. Instead, we will derandomize (C) by the technique of two-point sampling [6]. This technique involves showing that analysis of a randomized algorithm applies even when the random choices are pairwise independent; then, since pairwise independent choices can be obtained using a polynomial size probability space, the algorithm can be derandomized by efficiently searching the underlying probability space.

Let us study the random variable R defined as $R = \sum_{i=1}^{l} R_i$, where each R_i is a binary random variable, l is to be specified later, and R_i's are pairwise independent. Let S_{i-1} denote the set of vertices specified by the first $i-1$ random variables, and let y be the vertex chosen at the ith step. The variable R_i equals 1 if and only if

- the vertex y is in OPT, and
- for any vertex $x \in S_{i-1}$, $\{x, y\}$ is a scattered set.

For $i = O(\log^{O(1)} n)$,

$$\Pr[R_i = 1] = p_i \geq \frac{n/\log^4 n - (i-1)\sqrt{n}}{n - (i-1)} = \Omega\left(\frac{1}{\log^4 n}\right).$$

Since $E(R_i) = p_i$ and $\sigma^2_{R_i} \leq p_i$, we have $E(R) = \sum_{i=1}^{l} p_i$ and $\sigma^2_R \leq \sum_{i=1}^{l} p_i$ (using the pairwise independence of R_i's). By Chebyshev's inequality, we have

$$\Pr[|R - E(R)| \geq t] \leq \frac{\sigma^2_R}{t^2}.$$

Choosing $t = E(R)/2$ and $l = \log^5 n$, we conclude that

$$\Pr\left[R \leq \frac{E(R)}{2}\right] = O\left(\frac{1}{\log n}\right).$$

Since $E(R) = \Omega(\log n)$, with some constant probability, we are guaranteed to find a scattered set of size at least $\log n/(5 \log\log n)$ among the $\log^5 n$ vertices chosen above.

Once the $\log^5 n$ vertices are chosen, we can exhaustively try all possible subsets of k vertices among them. The number of such subsets is only $o(n)$.

The derandomization is now simple; it is well known that we can construct $\Omega(n)$ pairwise independent random variables by using only $O(\log n)$ random bits [4, 3]. Hence the probability space constructed above uses only $O(\log n)$ bits and can be exhaustively searched in polynomial time. Thus we have derandomized the algorithm of Section 3.1.

Corollary 1 *LCST in bounded degree trees can be approximated in polynomial time to within a ratio of* $O(n(\log\log n)/\log^2 n)$.

3.4 Parallelization

The derandomized algorithm above can in fact be implemented in NC. Only the NC implementation of Lemma 5 needs some explanation. The algorithm in [7] essentially proceeds as follows. Delete the roots of S and T to obtain rooted subtrees $S_1, S_2, ..., S_p$ and $T_1, T_2, ..., T_q$. Form a $p \times q$ binary matrix M such that $M_{ij} = 1$ if and only if S_i is a rooted subtree of T_j (determined recursively). Check if there is a matching in the bipartite graph specified by M which matches each subtree of S to a distinct subtree of T. Since the height of the tree S in our application is only $O(\log^2 n)$, the recursion has $O(\log^2 n)$ levels. The problem of bipartite maximum matching in general is known only to be in RNC [5] and not known to be in NC. However, since an augmenting path can be determined in NC, and in our application, the size of the matching is $O(\log^2 n)$, our algorithm can be implemented in NC. Thus we have the following corollary.

Corollary 2 *There is an NC algorithm to approximate LCST in bounded degree trees to within a ratio of* $O(n(\log\log n)/\log^2 n)$.

4 The Unbounded Degree Case

We now consider the case when the maximum degree of any vertex is unbounded. To begin with, observe that we may state Theorem 1 in the following more general form.

Theorem 2 *LCST can be approximated to within a ratio of* $O(n(\log\log n \log \Delta)/\log^2 n)$ *in polynomial time, where* Δ *denotes an upper bound on the degree of any vertex in* OPT.

We use the above theorem to design two separate algorithms for the case of unlabeled and labeled trees.

4.1 Unlabeled Trees

We observe the following.

Lemma 6 *If the largest degree of any vertex in* OPT *is Δ, then the LCST of unlabeled trees can be approximated to within a ratio of $O(n/(1+\Delta))$.*

Proof: Let Δ_i be the largest degree of any vertex in the tree $T_i \in \mathcal{C}$, and let $\Delta_C = \min_i \Delta_i$. Clearly, the graph K_{1,Δ_C} (a single vertex connected to Δ_C other vertices) occurs as a common subtree and $\Delta_C \geq \Delta$. ■

Theorem 3 *LCST in unlabeled trees can be approximated in polynomial time to within a ratio of $O(n(\log\log n)^2/\log^2 n)$.*

Proof: An algorithm which outputs the largest of the two solutions implicit in Theorem 2 and Lemma 6 achieves this approximation ratio. ■

4.2 Labeled Trees

The above result cannot be directly extended to labeled trees because we do not have an immediate analogue of Lemma 6. However, if the number of distinct labels is suitably restricted, it is possible to obtain a similar result.

Theorem 4 *LCST in labeled trees can be approximated in polynomial time to within a ratio of $O(n(\log\log n)^2/\log^2 n)$, provided the number of distinct labels is $O(\log^c n)$ for some constant c.*

Proof: If the maximum degree of any vertex in OPT is $O(\log^{c+2} n)$, then Theorem 2 still gives us an approximation ratio of $O(n(\log\log n)^2/\log^2 n)$. On the other hand, if the maximum degree is $\Omega(\log^{c+2} n)$, then there exists a vertex in OPT having at least $\Omega(\log^2 n)$ children with the same label. Therefore, an algorithm which simply computes for each label b the largest star with all children having the label b, gives an $O(n/\log^2 n)$ approximation ratio. ■

Finally observe that both of the above theorems naturally yield NC algorithms.

References

[1] T. Akutsu. An RNC algorithm for finding a largest common subtree of two trees. *IEEE Trans. Inf. & Syst.*, E75-D (1992), pp. 95–101.

[2] T. Akutsu and M.M. Halldórsson. On the approximation of largest common point sets and largest common subtrees. *Proc. of 5th Ann. Int. Symp. on Algorithms and Computation*, Lecture Notes in Comput. Sci., 834 (Springer-Verlag, 1994), pp. 405–413.

[3] B. Chor and O. Goldreich. On the power of two-point sampling. *Journal of Complexity*, 5 (1989), pp. 96–106.

[4] A. Joffe. On a set of almost deterministic k-independent random variables. *Annals of Probability*, 2 (1974), pp. 161–162.

[5] R.M. Karp, E. Upfal, and A. Wigderson. Constructing a perfect matching is in random NC. *Combinatorica*, 6 (1986), pp. 35–48.

[6] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press)New York, 1995).

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

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