アルゴリズム 37－3
（1994．1．25）

最小共通包含木問題に対する並列近似アルゴリズム

山口 敦子† 中野 浩嗣† 宮野 悟‡

†(株) 日立製作所 基礎研究所
〒350-03 埼玉県比企郡鳩山町赤沼 2520
{atsuko,nakano}@harl.hitachi.co.jp
‡ 九州大学理学部基礎情報学研究施設
〒812 福岡市東区箱崎 6-10-1
miyano@rifis.sci.kyushu-u.ac.jp

最小共通包含木 (Minimum Common Supertree) 問題は与えられたラベル付き完全 k 分木の集合に対し、その全ての木を部分木として含む辺数最小の k 分木を求める問題である。この問題は NP 困難な問題であることが示されている。本稿では、この問題に対する多項式時間近似アルゴリズムを与える。さらに、このアルゴリズムによって最適解の $1+1/(2k-2)$ 倍以下の大きさの解を求めることができることを示す。また、この近似アルゴリズムを並列化し、並列近似アルゴリズムとしての計算量も評価する。

A Parallel Approximation Algorithm for the Minimum Common Supertree Problem

Atsuko Yamaguchi† Koji Nakano† Satoru Miyano‡
†Advanced Research Laboratory, Hitachi, Ltd.
Hatoyama, Saitama 350-03, Japan
‡Research Institute of Fundamental Information Science
Kyushu University 33, Fukuoka, 812, Japan

The minimum common supertree problem is a problem to find a minimum k-ary common supertree for a given set of labeled complete k-ary trees. This problem was shown to be NP-hard. This paper gives a polynomial-time approximation algorithm for the problem. The algorithm constructs a common supertree of size at most $1+1/(2k-2)$ times as large as a minimum one. Furthermore, a parallel version of this algorithm, an NC approximation algorithm for the problem, is shown.

1 Introduction

The shortest common superstring problem is to find the shortest string u that contains all strings in a given set S as substrings. This problem is known to be NP-complete [2, 3]. There have been some studies on approximation algorithms for the shortest common superstring problem [1, 5, 7, 8]. In particular, it is shown in [1] that a simple greedy algorithm produces a common superstring of length at most three time as large as the length of shortest common superstring.

This paper deals with *the minimum common supertree problem*, a generalization of the shortest common superstring problem. Let *a tree over* Σ be a directed tree whose edges are labeled with symbols in an alphabet Σ. *A k-ary tree* is a tree such that each vertex has at most k children. *A complete k-ary tree* is a tree such that every internal vertex has exactly k children and the leaves are of the same depth. For a set T of complete k-ary trees, *a common supertree* for T is a k-ary tree such that each tree in T is a subtree of it. *The minimum common supertree problem for complete k-ary trees* (abbreviated as MCSP(k)) is defined as follows:

INSTANCE A finite set T of complete k-ary trees over some alphabet Σ,

PROBLEM Find a k-ary tree S with the minimum number of edges such that each tree in T is a subtree of S.

The problem MCSP(1) is essentially the same as the shortest common superstring problem. Hence, MCSP(k) is a generalization of the shortest common superstring problem.

We can consider two cases: given trees are *ordered* or *unordered*. An ordered tree and an unordered tree are trees so that children of every internal vertex in it are ordered and unordered, respectively. It is easy to determine whether or not two ordered trees over Σ are identical: Each edge of one tree corresponds to an edge of the other tree in accordance with the order of children. Then, the two trees are identical iff they have the same label for every corresponding edges. But, it is not particularly easy to determine it for two unordered trees, because the order of children is undefined. However, we can convert an unordered tree to an ordered one such that two unordered tree are identical iff the converted ordered trees are identical. The conversion, that shall be shown in Section 2, runs in polynomial sequential time and in poly-logarithmic parallel time using a polynomial numbers of processors. Therefore, by converting the unordered trees into ordered trees beforehand, the unordered version of the problem can be converted to the ordered version. Therefore, we do not have to pay heed of whether the children are ordered or unordered.

This paper deals with MCSP(k) for $k \geq 2$. It was shown that MCSP(k) is a NP-hard problem for every $k \geq 2$ [9]. If P$\neq$NP that is a widely accepted conjecture, there exists no polynomial-time algorithm for MCSP(k). Thus, it is interesting to find an approximation algorithm that computes a small common supertree for an MCSP(k) instance: The common supertree computed by the algorithm is not always the minimum one, but the number of edges of it is not particularly large compared with the minimum one.

Section 2 is a preliminary for a polynomial-time algorithm for MCSP(k). We show a polynomial-time approximation algorithm in Section 3 and evaluate the error bound of it in Section 4. The common supertree computed by the algorithm is at most $1+\frac{1}{2k-2}$ times as large as the minimum common supertree. Section 5 shows a parallel version of the algorithm with running time $O(\log^4 n)$ using n^6 processors. The error bound of the algorithm is the same as the serial one.

2 Preliminary

This section prepares some definitions for an approximation algorithm. For a k-ary tree s, $|s|$ denotes the number of edges in s. Let $label(e)$ denote a label of an edge e. For any alphabet Σ, we can fix an order of elements in Σ easily. Then, we suppose an order of elements in an alphabet.

Difinition 1. Let s,t be ordered complete k-ary trees over an alphabet Σ with height i, and r_s and r_t be the roots of s and t respectively. We denote $s \leq t$ if s and t satisfy the following conditions:

1. If $i = 1$: let $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ denote edges sorted according to their labels in the order of Σ in s and t respectively. Then $label(a_1) \cdots label(a_k)$ is less than $label(b_1) \cdots label(b_k)$ in lexicographic order.

2. If $i > 1$: it is supposed that $\leq$ is well-defined for complete k-ary trees over Σ with height $i-1$. We consider a subtree p_e of a complete k-ary tree p over Σ with height i composed by an edge $e = (r, v)$ where r is the root of p and a subtree q_e rooted at v. Then we denote $e \leq_{tree} e'$ if $label(e) < label(e')$ or $label(e) = label(e')$ and $q_e \leq q_{e'}$. Let $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ be edges

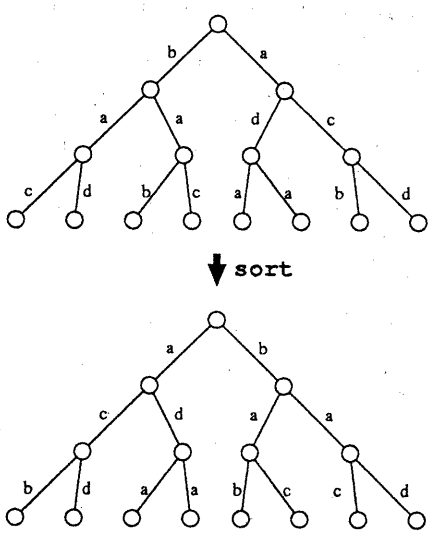

Figure 1: Sorting of unordered tree over $\{a, b, c, d\}$.

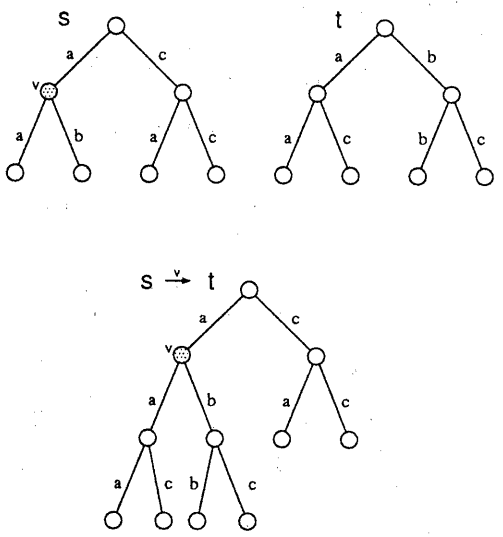

Figure 2: The tree $s \xrightarrow{v} t$. $ov(s \xrightarrow{v} t) = 2$.

sorted in order of $\leq_{tree}$ in s and t with depth 1, respectively. Then $a_1 \cdots a_k$ is less than $b_1 \cdots b_k$ in the lexicographic order of $\leq_{tree}$.

Difinition 2. An ordered tree t over Σ is called *sorted* if $a_1 \leq_{tree} a_2 \leq_{tree} \cdots \leq_{tree} a_k$ for the edges $a_1, \ldots, a_k$ starting from the root of t.

As shown in Fig. 1, an unordered tree t can be converted to the sorted ordered tree $sort(t)$. The conversion can be done in $O(|t|)$ time. Moreover, the conversion can be also performed in $O(\log |t|)$ time using $|t|$ processors on the PRAM by sorting in the bottom-up way.

Lemma 1. Let T be a set of unordered complete k-ary trees over an alphabet Σ and T' the set of sorted trees in T. That is, $T' = \{sort(t) \mid t \in T\}$. Then, the number of edges in a minimum common supertree for T is equal to that for T'.

Therefore, we assume that every tree is converted beforehand in the unordered version of MCSP(k).

Difinition 3. A set T of trees is called *reduced* if no tree in T is a subtree of another tree in T.

Let T be a set of complete k-ary trees. Then $T' = T - \{t \mid t$ is a subtree of some $s \in T$ with $s \neq t\}$ is obviously reduced. The minimum common supertree for T is also that for T'. Furthermore, T' is computable from T in polynomial-time as well as in poly-logarithmic time using a polynomial number of processors. Therefore, without loss of generality, we assume that T is a reduced set of complete k-ary trees.

For complete k-ary trees s and t, we introduce the following notations:

1. $s \xrightarrow{v} t$ denotes a common supertree for s and t so that the root of t is identified with a vertex v of s, provided that every pair of corresponding edges of the two trees has the same label(Fig. 2). If there is a pair of corresponding edges whose labels are different, $s \xrightarrow{v} t$ is undefined.
2. For defined $s \xrightarrow{v} t$, $ov(s \xrightarrow{v} t)$ denotes $|s| + |t| - |s \xrightarrow{v} t|$, the number of pairs of corresponding edges in $s \xrightarrow{v} t$. $ov(s \xrightarrow{v} t)$ is called the *overlap* of t on s at v.

A complete k-ary tree over Σ is denoted by $t = (v(t), e(t))$ where $v(t)$ is a set of vertices, $e(t)$ is a set of labeled edges (u, u', l) $(u, u' \in v(t), l \in \Sigma)$ connecting two vertices u and u' in $v(t)$ with label l. For a set T of complete k-ary trees, let $V_T = \bigcup_{t \in T} v(t)$. Without loss of generality, for any t, t' in T, we assume $v(t) \cap v(t') = \emptyset$. Now we define *the overlap graph* G_S that expresses a common supertree S for T as follows: $G_S = (T, E_S)$ is a directed labeled graph such that E_S is a subset of the edge set

$\{(t,t',u) \mid t,t' \in T, u \in v(t)\}$ satisfying the following conditions:

(1) For all (t,t',u) in E_S, $t \xrightarrow{u} t'$ is defined.

(2) $G_S = (T, E_S)$ is a directed spanning tree,

(3) For any $t,t' \in T$, if both (s,t,u) and (s,t',u') are in E_u, the two vertices u and u' is not on a single directed path of s.

For every $e = (t,t',u) \in E_S$, the edge e express that t is put in t' such that the root of t' is identified with the vertex u of t.

It is clear that an overlap graph G_S can be determined uniquely from a common supertree S. Conversely, for a labeled directed tree $G = (T,E)$, we shall prove that we can construct a common supertree for T corresponding to G if G satisfies the conditions (1), (2), and (3). The proof is based on the induction on the size of the graph. That is, we assume that a common supertree can be constructed from every labeled directed spanning tree satisfying the three conditions whose size is less than that of G, and shall show that a common supertree can be constructed from G. Suppose that G is separated into some subtrees by removing the root from G. Then, each of them satisfies the three conditions. Hence, from the inductive assumption, we can construct a common supertree for each subgraph. For each subgraph, there is an edge between the root of it and that of G. Since G satisfies the three conditions, the common supertree for each subgraph can be placed on the tree corresponding the root of G in accordance with the edge. The tree thus obtained is the common supertree for G.

For an overlap graph $G_S = (V,E)$, let $SIZE(G_S)$ be the number of edges in a common supertree S and $||G_S|| = \sum_{(s,t,u)\in E} ov(s \xrightarrow{u} t)$, the sum of the overlap. Let $n = \sum_{t\in T} |t|$, the total numbers of edges in T. Then we have,

Lemma 2. For an overlap graph $G_S = (T, E_S)$ of a (reduced) set T of complete k-ary trees, $SIZE(G_S) = n - ||G_S||$ holds.

Therefore, the problem MCSP(k) corresponds to the following problem:

INSTANCE A finite set T of complete k-ary trees over some alphabet Σ.

PROBLEM Find an overlap graph $G_S = (T, E_S)$ with maximum $||G_S||$.

3 Approximation algorithm

This section shows a polynomial-time approximation algorithm that computes an overlap graph $G = (T,E)$ for a given set T of complete k-ary trees.

The algorithm determines the overlap graph based on the greedy technique. It has $|T| - 1$ phases. In each phase, it select (s,t,u) such that $ov(s \xrightarrow{u} t)$ is the maximum value on the condition that

(a) $s \xrightarrow{u} t$ is defined,

(b) $G = (T, E \cup \{(s,t,u)\})$ is a directed forest,

(c) For any $(s,t',u') \in E$, the two vertices u and u' is not on a single directed path of s.

Then, (s,t,u) is added to E. After $|T| - 1$ iterations of this phase, an overlap graph can be obtained.

The algorithm *GreedyOverlap* is formally written as follows: The algorithm uses a table of (s,t,u) where $s,t \in T$ and $u \in V_T$. Each element of the table has the capability of storing the value of $ov(s \xrightarrow{u} t)$ and keeping one of two status *candidate* and *defeated*. Initially, every element is *candidate*.

Step 1 Let $E = \emptyset$. For each element (s,t,u) of the table, compute $ov(s \xrightarrow{u} t)$ and store it to the table.

Step 2 Repeat the following substeps $|T| - 1$ times:

Step 2.1 For every element (s,t,u) in the table, determine it satisfies conditions (a), (b), and (c). If it does not satisfy every conditions, its element becomes *defeated*.

Step 2.2 Select (s,t,u) such that $ov(s \xrightarrow{u} t)$ is the maximum over all candidate elements.

Step 2.3 Add (s,t,u) to E.

For each s,t,u, $ov(s \xrightarrow{u} t)$ can be computed in $O(n)$ time. Since u is a vertex in s, the size of table (s,u) is less than n. Then, Step 1 takes $O(|T|n^2) = O(n^3)$ time. For each s,t,u, it takes constant time to determine whether or not the condition (a) is satisfied by looking up the table. By using the depth-first search technique, $O(n)$ time is sufficient for the condition (b). Since the number of the ancestors of u is at most $\log n$, $O(|T|^2 \log n) = O(n^2 \log n)$ is sufficient for the condition (c) by looking up the table for each ancestors. Hence, for each s,t,u, $O(n^2 \log n)$ is sufficient for the three conditions and Step 2 takes $O(|T|n^2 \log n) = O(n^3 \log n)$. Therefore, we have

Lemma 3. GreedyOverlap runs in $O(n^3 \log n)$ time.

4 Error bound

We prove the upper bound of the size of the solution obtained by using the approximation algorithm *GreedyOverlap*.

For a given set T of complete k-ary trees, let S_{opt} be a minimum common supertree for T, and $G_{S_{opt}} = (T, E_{S_{opt}})$ the overlap graph of S_{opt}. Furthermore, let S and $G_S = (T, E_S)$ be a common supertree for T and the overlap graph of S computed by the algorithm *GreedyOverlap*. The main theorem of this section is the following:

Theorem 4. For G_S and $G_{S_{opt}}$ defined above,

$$SIZE(G_S) \leq (1 + \frac{1}{2k-2})SIZE(G_{S_{opt}}).$$

In other words, the size of the common supertree computed by the algorithm *GreedyOverlap* is at most $1 + 1/(2k-2)$ times as large as that of the minimum common supertree.

Theorem 4 can be proved by using the following Lemmas 5 and 6:

Lemma 5. For any overlap graph $G_S = (T, E_S)$ of a set T of complete k-ary trees, the relation $||G_S|| \leq n/k$ holds.

Proof. Let $h(t)$ denote the height of a k-ary tree t. Then, $|t| = k^1 + k^2 + \cdots + k^{h(t)} = (k^{h(t)+1} - k)/(k-1)$. For any $(s,t,u) \in E$, we have $ov(s \xrightarrow{u} t) \leq (k^{h(t)} - k)/(k-1) \leq |t|/k$. Furthermore, for any $t \in T$, there is at most one tree s that satisfies $(s,t,u) \in E$. Therefore, we have

$$\begin{aligned} ||G|| &= \sum_{(s,t,u)\in E} ov(s \xrightarrow{u} t) \leq \sum_{(s,t,u)\in E} |t|/k \\ &\leq \sum_{t\in T} |t|/k \leq n/k \end{aligned}$$

□

Lemma 6. For G_S and $G_{S_{opt}}$ mentioned above, $||G_{S_{opt}}|| \leq 2||G_S||$ holds.

Proof. To evaluate the overlap of G_S relative to that of $G_{S_{opt}}$, let us trace the algorithm *GreedyOverlap*. Let $e_i = (s_i, t_i, u_i)$ $(1 \leq i \leq |T|-1)$ be the edge added to E such that the root of t_i is identified with the vertex u_i of s_i at the i-th iteration of Step 2.3. Suppose that the algorithm transforms $G_{S_{opt}}$ into G_S as follows: Let $G_0 = G_{S_{opt}}, G_1, G_2, \ldots, G_{|T|-2}, G_{|T|-1} = G_S$ be the sequence of graphs such that G_{i-1} is transformed into G_i. To transform G_{i-1} to G_i, the edge $e_i = (s_i, t_i, u_i)$ is added into G_{i-1}. Furthermore, some edges are removed from G_{i-1} and some edges are added to G_{i-1} so that the graph G_i obtained may satisfy the conditions (a), (b), and (c) and $||G_i|| \geq ||G_{i-1}|| - ov(e_i)$ may hold.

We shall show how to transform from G_{i-1} into G_i. If e_i is an element in E_{i-1}, let $G_i = G_{i-1}$. Then, G_i satisfies obviously the conditions (a), (b), (c), and $||G_i|| \geq ||G_{i-1}|| - ov(e_i)$ hold. Hence, from now on, we assume that E_{i-1} does not have e_i. Next, let us focus on the two cases whether or not E_{i-1} has an edge $e_i' = (s', t_i, u')$.

First, we consider the case that E_{i-1} does not have e_i': To begin with, $e_i = (s_i, t_i, u_i)$ is added to E_{i-1} and e_i' is removed from E_{i-1} to satisfy (b). Let F_i be the set consists of every edge (s_i, t', u') in E_{i-1} satisfying the condition that u' is a descendant of u_i. To satisfy (c), remove all edges in F_i from E_{i-1}. Note that there is no edge (s_i, t', u') in E_{i-1} such that u' is an ancestor of u_i. Otherwise, the algorithm must select (s_i, t', u') to add E_{i-1} in stead of e_i because $ov((s_i, t', u')) > ov(e_1)$. G_i thus obtained may have a cycle. Hence, we select an edge c_i in E_{i-1} such that c_i is on the cycle and for all j $(1 \leq j \leq i)$, $c_i \neq e_j$. For such an c_i, we remove c_i from E_{i-1}, and E_{i-1} satisfies (b). Note that $ov(c_i) \leq ov(e_i)$ holds. Otherwise the algorithm must select c_i in stead of e_i to add E_{i-1}. Then, let E_i be E_{i-1} thus obtained, and we have

$$\begin{aligned} ||G_i|| &= ||G_{i-1}|| + ov(e_i) - ov(F_i) - ov(c_i) \\ &\geq ||G_{i-1}|| - ov(e_i), \end{aligned}$$

where $ov(F_i)$ denotes $\sum_{t\in F_i} ov(t)$.

Secondly, we consider the case that E_{i-1} has e_i': Similarly to the first case, e_i is added to E_{i-1}, and F_i and c_i are removed from E_{i-1}. Then, e_i' is removed from E_{i-1}. Furthermore, we add every H_i to E_{i-1} where H_i is determined as follows: For each removed edge (s_i, t', u') in F_i, we can find the vertex u'' of s' such that $s' \xrightarrow{u''} t'$ is defined if corresponding vertex of u' exists in s' and otherwise u'' is an arbitrary leaf of s'. Let H_i be the set of all corresponding edge (s', t', u'') to (s_i, t', u'). Then we have the following Fact:

Fact 1. $ov(e_i) + ov(H_i) \geq ov(e_i') + ov(F_i)$.

Proof. Let $H_i = \{f_1, \ldots, f_l\}$, and $F_i = \{f_1', \ldots, f_l'\}$(Fig. 3). Let $h(e_i)$ and $h(f_j')$ be the heights of the subtrees of s corresponding to the edges e_i and f_j', and $h(e_i')$ and $h(f_j)$ the heights of the subtrees of t corresponding to the edges e_i' and f_j, respectively. Then, we see $h(f_j) \geq h(e_i') + h(f_j') - h(e_i)$. Moreover, since the subtree of s corresponding to

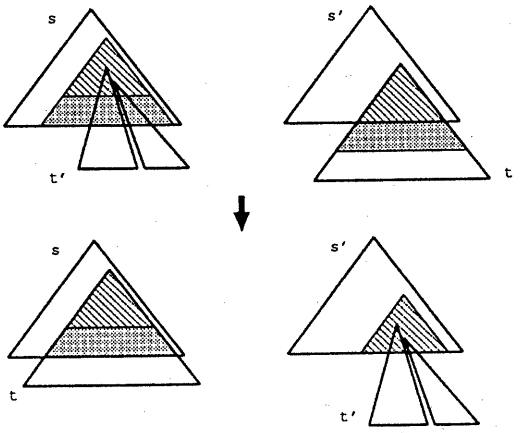

Figure 3: The trees expressed by the edges $e_i' = (s,t'), f_j' = (s',t), e_i = (s,t), f_j = (s,t')$.

the edge e_i includes the subtrees corresponding to the edges f_j' as they do not overlap each other, $\sum_{1\le j\le l} k^{h(f_j')} \le k^{h(e_i)}$ holds. Therefore,

$$
\begin{aligned}
& ov(e_i) + ov(H_i) - ov(e_i') - ov(F_i) \\
&= \frac{k^{h(e_i)}-k}{k-1} + \sum_{1\le j\le l}\frac{k^{h(f_j)}-k}{k-1} \\
&\quad -\frac{k^{h(e_i')}-k}{k-1} - \sum_{1\le j\le l}\frac{k^{h(f_j')}-k}{k-1} \\
&\ge \frac{k^{h(e_i)}-k}{k-1} + \sum_{1\le j\le l}\frac{k^{h(e_i')+h(f_j')-h(e_i)}-k}{k-1} \\
&\quad -\frac{k^{h(e_i')}-k}{k-1} - \sum_{1\le j\le l}\frac{k^{h(f_j')}-k}{k-1} \\
&= \frac{k^{h(e_i)} - k^{h(e_i')} + (k^{h(e_i')-h(e_i)}-1)\sum_{1\le j\le l} k^{h(f_j')}}{k-1} \\
&\ge \frac{k^{h(e_i)} - k^{h(e_i')} + (k^{h(e_i')-h(e_i)}-1)k^{h(e_i)}}{k-1} \\
&= 0
\end{aligned}
$$

□

Then, we have,

$$
\begin{aligned}
\|G_i\| &= \|G_{i-1}\| + ov(e_i) - ov(e_i') \\
&\quad -ov(F_i) + ov(H_i) - ov(c_i), \\
&\ge \|G_{i-1}\| - ov(c_i), \\
&\ge \|G_{i-1}\| - ov(e_i).
\end{aligned}
$$

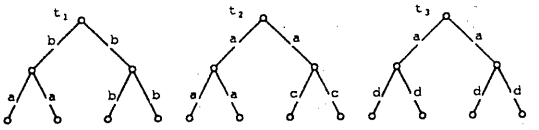

Figure 4: The set T of complete k-ary trees.

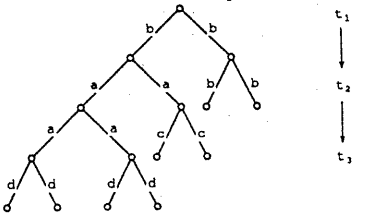

Figure 5: The minimum supertree for T.

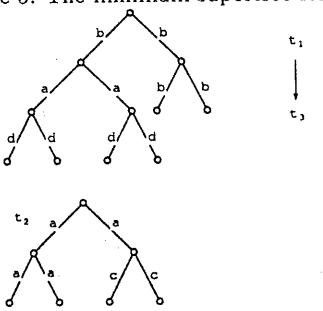

Figure 6: The worst case using *GreedyOverlap*.

From this relation, we have,

$$
\begin{aligned}
\|G_{S_{opt}}\| &\le \|G_S\| + \sum_{i=1}^{|T|-1} ov(e_i) \\
&\le 2\|G_S\|.
\end{aligned}
$$

□

Now we shall give the following example showing that the result of Lemma 6 is best possible for the algorithm *GreedyOverlap*.

Example 1. Let T be a set of complete k-ary trees over $\Sigma = \{a,b,c,d\}$ shown as Fig. 4. The overlap of the minimum common supertree for T is 4 (Fig. 5). However, the algorithm *GreedyOverlap* may select the edge at first as Fig. 6. Then, the overlap of this common supertree is 2, the half of minimum one.

Theorem 4 can be proved as follows:

Proof of Theorem 4.

$$\begin{aligned}\frac{SIZE(G_S)}{SIZE(G_{S_{opt}})} &= \frac{n-\|G_S\|}{n-\|G_{S_{opt}}\|} && (from\ \text{Lemma 2})\\ &\leq \frac{n-\frac{1}{2}\|G_{S_{opt}}\|}{n-\|G_{S_{opt}}\|} && (from\ \text{Lemma 6})\\ &\leq 1+\frac{1}{2k-2} && (from\ \text{Lemma 5})\end{aligned}$$

□

5 Parallelization of Greedy-Overlap

This section shows the *ParallelGreedyOverlap*, a parallel version of the *GreedyOverlap*, that solves MCSP(k) in poly-logarithmic time using polynomial numbers of processors on the CRCW-PRAM.

Before showing the *ParallelGreedyOverlap*, we review the *GreedyOverlap*. The *GreedyOverlap* has $|T|-1$ phases. In each phase, a single edge of the overlap graph is determined as follows: The largest overlap of all edges satisfying the three conditions is computed, and one arbitrary edge is selected among the edges whose overlaps are equal to the largest overlap.

The *ParallelGreedyOverlap* simulates the *GreedyOverlap*. Let V_U be the set of edges whose overlap is the largest of all edges satisfying the three conditions. The edges in V_U must be selected simultaneously to attain the poly-logarithmic time. Let E be the set of the previously selected edges. For the simultaneous selection, a graph $U=(V_U,E_U)$ such that $E_U=\{((s,t,u),(s',t',u'))\mid t=t' \text{ or } u=u'\}$ is constructed and a maximal independent set I of U is computed, i.e., I is a maximal subset of V_U such that no two vertices in I are not joined by an edge in E_U. If all edges of an overlap graph corresponding to vertices in I would be selected, the selected edges may make cycles. This violates the condition (b). Hence, to break the cycles, the weighted graph is constructed such that the edge set is $E\cup I$ and the weight of each (s,t,u) in I is 2 and that in E is 1. For the graph, the minimum spanning forest is computed. From the assignment of the weight, every edge in E is included in the minimum spanning tree, and some edges in I are removed. Then, every edge of I in the spanning forest is added to E. This selection is repeated until $|T|-1$ edges are selected.

The *ParallelGreedyOverlap* is written as follows:

Step 1 Let $E=\emptyset$. For each element (s,t,u) of the table, compute $ov(s \xrightarrow{u} t)$ and store it to the table.

Step 2 Repeat the following substeps until $|E|=|T|-1$.

Step 2.1 For every element (s,t,u) in the table, determine whether or not it satisfies conditions (a), (b), (c).

Step 2.2 Compute the maximum overlap of all edges satisfying the three conditions and select edges with the maximum overlap. Let V_U be a set of the selected edges.

Step 2.3 Construct a graph $U=(V_U,E_U)$ such that E_U is a set $\{((s,t,u),(s',t',u'))|t=t' \text{ or } u=u'\}$ of edges.

Step 2.4 Compute the independent set I of U.

Step 2.5 Construct the weighted graph such that the edge set is $E\cup I$ and the weight of each (s,t,u) in I is 2 and that in E is 1.

Step 2.6 Compute the minimum spanning forest of the weighted graph. Then, let E be the set of edges in the spanning forest.

In the *ParallelGreedyOverlap*, previously known algorithms are used:

MSF(Minimum Spanning Forest) For a graph $G=(V,E)$, the minimum spanning forest of G can be computed in $O(\log^2|V|)$ time using $O(|V|^2/\log^2|V|)$ processors [4]. Using this algorithm, it can be also determined whether or not the graph has cycles.

MF(Maximum Finding) For given n integers, the maximum of them can be computed in $O(\log n)$ time using n processors [4].

MIS(Maximal Independent Set) For a graph $G=(V,E)$, the maximal independent set of G can be computed in $O(\log^2|V|)$ time using $O(|V|^2|E|)$ processors [6].

Using the algorithms above, we shall evaluate the computing time of the *ParallelGreedyOverlap*. Since the equivalence of two trees can be determined in constant time by assigning a processor to each edge, the values of $ov(s \xrightarrow{u} t)$ for each element (s,t,u) of the table can be computed in constant time using $|s|+|t|=O(n)$ processors. Since the size of tables (s,u) is $O(n)$, Step 1 takes constant time using at most $O(n)\times|T|n=O(n^4)$ processors. For each element (s,t,u), the decision of the condition (b) can be performed in constant time using a single processor by referring the table. The decision of the condition (b) can be done in $O(\log^2 n)$ time using

$O(n^2/\log^2 n)$ processors. The decision of the condition (c) can be done in constant time using n^2 processors by referring to E and s. Thus, Step 2.1 can be done in $O(\log^2 n)$ time using $O(n^2) \times n^3 = O(n^5)$ processors. Step 2.2 takes $O(\log n)$ time using n^3 processors. Since $|V_U| \leq n^3$, Step 2.3 takes constant time using $n^3 \times n^2 = n^5$ processors. Step 2.4 takes $O(\log^2 n)$ time using n^6 processors. Since elements in I are distributed in n^6 space, Step 2.5 can be done in $O(\log n)$ time using n^6 processors. Since the weighted graph has at most n nodes, Step 2.6 can be done in $O(\log^2 n)$ time using $O(n^2/\log^2 n)$ processors. Therefore, each iteration of Step 2 takes $O(\log^2 n)$ time using n^6 processors.

It remains to show that Step 2 is repeated at most poly-logarithmic times. Let m_i and U_i be the maximum overlap and U computed at the i-th iteration, respectively. Let E_i denote E just before the i-th iteration. Each m_i takes one of the values of $f(0), f(1), \ldots, f(\lfloor \log n \rfloor)$ where $f(h)$ is the number of edges of a complete k-ary tree with height h, $(k^h - k)/(k-1)$ for each h $(0 \leq h \leq \lfloor \log n \rfloor)$. Furthermore, $m_1 \geq m_2 \geq \cdots \geq m_t$ holds where t is the number of the iteration. Let us consider the subsequence $m_j, m_{j+1}, \cdots, m_{j'}$ of m's such that $m_{j-1} > m_j = m_{j+1} = \cdots = m_{j'} > m_{j'+1}$. Then, observe the connected components of $E_j, E_{j+1}, \ldots, E_{j'+1}$. Fix a connected component of $E_{j'+1}$. The component may be separated into two or more connected components on E_j. As computing $E_{j+1}, E_{j+2}, \ldots, E_{j'+1}$, these connected components are merged, and finally, they are merged into one component. Furthermore, in each iteration, every connected component is merged with the other connected components if it is not alone. Hence, the number of the components decreases by half in each iteration. Thus, $j' - j \leq \log n$ holds. As a result, $t \leq \log^2 n$ holds. Therefore, we have

Theorem 7. *ParallelGreedyOverlap* solves the MCSP(k) in $O(\log^4 n)$ time using n^6 processors with the error bound $1 + 1/(2k-2)$.

6 Conclusions

This paper has presented the approximation algorithm *GreedyOverlap* for MCSP(k) that approximates the number of edges of the minimum common supertree by $1+1/(2k-2)$ for every $k \geq 2$. Though we have shown the worst case for the overlap graph, we do not know whether or not there exists an algorithm whose error bound is less than $1 + 1/(2k-2)$. In this paper we have restricted our attention to complete k-ary trees. Then, it remains to show an approximation algorithm for more general problem, i.e., the problem of finding a minimum common supertree for a finite set T of trees, where no restriction is put on the trees such as arity, etc.

References

[1] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. In *23rd ACM STOC*, pages 328–336, 1991.

[2] J. Gallant, D. Maier, and J. Storer. On finding minimal length superstrings. *J. Comput. System Sci.*, 20:50–58, 1980.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Company, 1979.

[4] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[5] M. Li. Towards a DNA sequencing theory. In *31st IEEE FOCS*, pages 125–134, 1990.

[6] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Computing*, 15:1036–1053, 1986.

[7] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoret. Comput. Sci.*, 57:131–145, 1988.

[8] J. Turner. Approximation algorithms for the shortest common superstring problem. *Inf. Comput.*, 83:1–20, 1989.

[9] A. Yamaguchi and S. Miyano. Approximating minimum common supertrees for complete k-ary trees. Tech. rep. no. 66, 1993.