
Journal of Combinatorial Optimization, 8, 29–39, 2004

Constructing the Maximum Consensus Tree from Rooted Triples

BANG YE WU bangye@mail.stu.edu.tw
Department of Computer Science and Information Engineering, Shu-Te University, YenChau, KaoShiung, Taiwan 824, R.O.C.

Received May 25, 2001; Revised October 31, 2001; Accepted February 28, 2002

Abstract. We investigated the problem of constructing the maximum consensus tree from rooted triples. We showed the NP-hardness of the problem and developed exact and heuristic algorithms. The exact algorithm is based on the dynamic programming strategy and runs in $O((m+n^2)3^n)$ time and $O(2^n)$ space. The heuristic algorithms run in polynomial time and their performances are tested and shown by comparing with the optimal solutions. In the tests, the worst and average relative error ratios are 1.200 and 1.072 respectively. We also implemented the two heuristic algorithms proposed by Gasieniec et al. The experimental result shows that our heuristic algorithm is better than theirs in most of the tests.

Keywords: computational biology, evolutionary trees, algorithms, dynamic programming, NP-hardness

1. Introduction

Evolutionary trees are used to present the relationship among a set of species. The leaves in an evolutionary tree correspond to the species and internal nodes are the ancestors of the species. Constructing evolutionary trees is an important problem in computational biology and there are different approaches. We investigated the problem of constructing evolutionary trees from rooted triples.

A rooted triple, or triple for brevity, represents the relationship of three species. As shown in figure 1, a triple $(a(bc))$ specifies $lca(a,b) = lca(a,c) > lca(b,c)$, in which $lca(a,b)$ is the lowest common ancestor of the two leaves and relation ">" means "is an ancestor of". For a set of triples, the exact consensus tree is the tree satisfies all given triples.

Given a set of triples, the existence of the exact consensus tree can be determined in polynomial time (Aho et al., 1981). For a set of constraints of the form $lca(a,b) > lca(c,d)$, the algorithm in Aho et al. (1981) determines if there is a tree satisfying all constraints and finds such a tree if it exists. A triple $(a(bc))$ is equivalent to $lca(a,c) > lca(b,c)$ and is a special case of the constraints considered in Aho et al. (1981). An algorithm for constructing all exact consensus trees from triples was also developed (Ng and Wormald, 1996). Unfortunately, it is often impossible to find the exact consensus tree and we want to find the tree satisfying as many given triples as possible. We shall call the optimization problem the *maximum consensus tree from rooted triples* problem, or the MCTT problem for brevity. Gasieniec et al. (1999) showed that the problem to find the maximum consensus

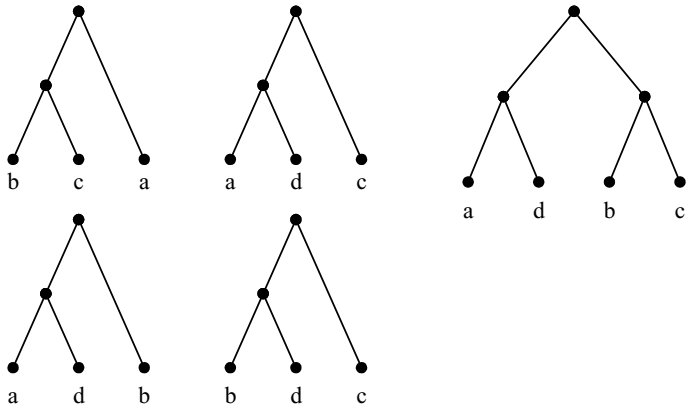

Figure 1. Left: rooted triples $(a(bc))$, $(c(ad))$, $(b(ad))$, $(c(bd))$; Right: the maximum consensus tree. The tree satisfies all triples except $(c(bd))$.

tree from constraints of the form $lca(a, b) > lca(c, d)$ is NP-hard. They also proposed two heuristic algorithms and discussed the theoretical performance ratios. The algorithms also work for the MCTT problem but the complexity of the MCTT problem was left open.

Similar problems for unrooted trees were also investigated. A quartet represents the relationship of four species. To determine if there is a tree satisfying a given set of quartets were shown to be NP-complete (Steel, 1992). Therefore the corresponding optimization problem is obviously NP-hard.

In this paper, we show that the MCTT problem is NP-hard. (Recently the NP-hardness was also independently shown by a different reduction (Jansson, 2001).) Exact and heuristic algorithms are also presented. The exact algorithm is based on the dynamic programming strategy and runs in $O((m + n^2)3^n)$ time and $O(2^n)$ space. The performances of the heuristic algorithms were tested by comparing their outputs with the exact solutions. In the tests, the worst and average relative error ratios are 1.200 and 1.072 respectively. We also implemented the two heuristic algorithms proposed by Gasieniec et al. (1999). The experimental result shows that our heuristic algorithm is better than theirs in most of the tests.

The time complexity of the MCTT problem is shown in Section 2. In Section 3, we present the exact and heuristic algorithms and the experimental results. We give a discussion in Section 4.

2. The computational complexity

In this section, we shall show the NP-hardness of the MCTT problem by reducing the Feedback Arc Set problem to it. We first give the definition of the Feedback Arc Set problem.

Definition 1. Let $G = (V, A)$ be a directed graph. A subset A' of A is a feedback arc set if every directed cycle in G contains at least one arc in A'. Given a directed graph $G = (V, A)$ and an integer k, the *Feedback Arc Set* problem asks if there is a feedback arc set A' with $|A'| \leq k$.

The Feedback Arc Set problem is NP-complete (Garey and Johnson, 1979; Karp, 1972).

Definition 2. Let a and b be nodes of a tree. The *lowest common ancestor* of a and b is denoted by $lca(a, b)$. We write $a > b$ if a is an ancestor of b.

Definition 3. A rooted triple, or triple for brevity, over a species set is a constraint on the relationship of three species. Let V be a species set and $a, b, c \in V$, the rooted triple $(a(bc))$ over V represents $lca(a, b) = lca(a, c) > lca(b, c)$ in the desired tree.

We say that a tree satisfies a triple or a triple is compatible with a tree if the relationship represented by the triple is satisfied in the tree.

Definition 4. Given a set Y of rooted triples over leaf set V, the maximum consensus tree from triples (MCTT) problem looks for a binary tree T with leaf set V such that the number of triples compatible with T is maximum.

The computational complexity is shown in the next theorem.

Theorem 1. *The MCTT problem is NP-hard.*

Proof: We reduce the Feedback Arc Set problem to the MCTT problem. Given an instance $G = (V, A)$ and k of the Feedback Arc Set problem, we shall construct a set of rooted triples Y and show that the directed graph G contains a feedback arc set of k arcs if and only if there is a tree compatible with $|A| - k$ triples from Y.

Let $x \notin V$. For every arc $(u, v) \in A$, there is a corresponding triple $(u(xv))$ in Y. Suppose that A' is a feedback arc set of G and $|A'| = k$. Since A' is a feedback arc set, removing A' from G results in a directed acyclic graph $G_1 = (V, A_1)$, in which $A_1 = A \backslash A'$. Since G_1 contains no cycle, we may assign each vertex v a label $f(v) \in \{1 \ldots p\}$ such that $f(u) < f(v)$ for every $(u, v) \in A_1$, where $p \leq |V|$ is number of nodes of the longest path in G_1. Let $V_i = \{v \mid f(v) = i\}$ and T_i be an arbitrary evolutionary tree of V_i for $1 \leq i \leq p$. We construct an evolutionary tree T of $V \cup \{x\}$ as in figure 2. For any arc $(u, v) \in A_1$, since $f(u) < f(v)$, the corresponding triple $(u(xv))$ in Y is compatible with T. Therefore all triples corresponding to arcs in A_1 are satisfied, and T compatible with $|A| - k$ triples in Y.

Conversely suppose that there is a tree T compatible with $|A| - k$ triples in Y. Let Y_1 be the set of satisfied triples in Y. As in figure 2, let the path from root to x be $(r_1, r_2, \ldots, r_p, x)$ and V_i denote the set of leaves whose common ancestor with x is r_i. For each triple $(u(xv)) \in Y_1$ in which $u \in V_i$ and $v \in V_j$, since $lca(u, x) = lca(u, v) > lca(x, v)$, we have $j > i$. Let A_1 be the set of arcs corresponding to the triples in Y_1, that is $A_1 = \{(u, v) \mid (u(xv) \in Y_1\}$.

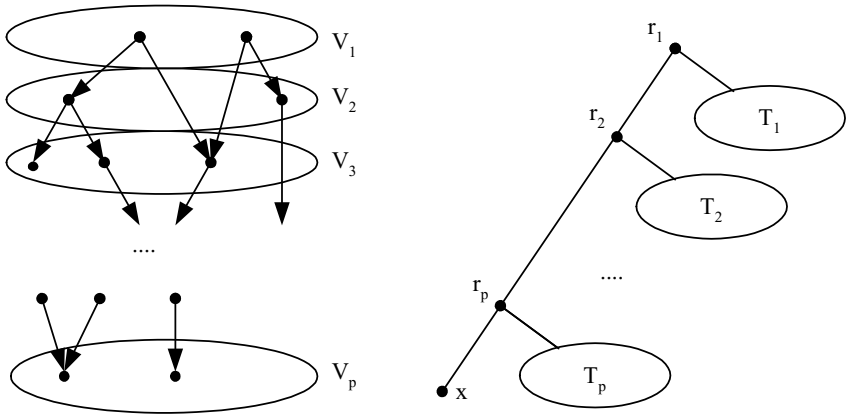

Figure 2. Transformation of an instance of the Feedback Arc Set problem into that of the MCTT problem. Left: the labeling of a directed acyclic graph; Right: A maximum consensus tree of the MCTT problem.

Consider the graph $G_1 = (V, A_1)$ and label each vertex v with i if $v \in V_i$. Since all the arcs in A_1 are from vertices with small labels to larger labels, G_1 contains no directed cycle. Therefore $A \backslash A_1$ is a feedback arc set of G and contains k arcs.

The above transformation reduces the Feedback Arc Set problem to the MCTT problem in polynomial time. Since the Feedback Arc Set problem is NP-complete, the MCTT problem is NP-hard. □

3. Algorithms and experimental results

In this section, exact and heuristic algorithms will be presented. In the remaining of this paper, Y is the set of the input triples over species set U. Let n and m be the cardinalities of U and Y respectively.

3.1. An exact algorithm

In this subsection, we shall present an algorithm to find the exact solution of the MCTT problem.

Definition 5. Let $V \subset U$, we use $score(V)$ to denote the maximum number of satisfiable triples in $\{(a(bc)) \mid b, c \in V\} \subset Y$.

Definition 6. Let $V \subset U$, the set of all bipartitions of V is denoted by $\mathcal{B}(V)$.

Definition 7. Let $V \subset U$ and $(V_1, V_2) \in \mathcal{B}(V)$. We use $w(V_1, V_2)$ to denote the number of triples $(x(v_1 v_2))$ in which $v_1 \in V_1$, $v_2 \in V_2$ and $x \notin V$.

If we label each internal node of the tree by the set of leaves of the subtree rooted at it, any internal node will be labelled by a subset of U, and the labels of its two children form a bipartition of its label. For any $V \subset U$, once the scores of all its subsets are found, we may compute $score(V)$ by trying all its possible bipartitions. The exact algorithm uses the dynamic programming strategy and is based on the following formula:

$$score(V) = \max_{(V_1, V_2) \in \mathcal{B}(V)} \{score(V_1) + score(V_2) + w(V_1, V_2)\} \tag{1}$$

Obviously $score(U)$ is the maximum number of satisfiable triples in Y. The exact algorithm is list below.

1. **Algorithm Exact_MCTT**
2. **Input:** A set Y of rooted triples over species set U.
3. All triples are stored in a matrix M of lists.
4. $M[i, j]$ is a list of the elements of set $\{x \mid (x(ij)) \in Y\}$.
5. **Output:** A rooted tree T satisfying maximum number of triples in Y.
6. **Step 1: Compute the maximum number of satisfied triples.**
7. For $i = 1$ to n do
8. For each subset V with cardinality i do
9. For each bipartition (V_1, V_2) of V do
10. Compute $w(V_1, V_2)$ by counting the number of elements
11. in $M[i, j] \backslash V$ for each $i \in V_1$ and $j \in V_2$;
12. $score(V) = \max\{score(V_1) + score(V_2) + w(V_1, V_2)\}$, in which
13. the maximum is taken over all bipartitions of V.
14. Record the best bipartition of V at $Partition(V)$.
15. **Step 2: Construct the tree by backtracking** $Partition(U)$.
16. Start from $V = U$.
17. If V contains only one species, create a leaf node for it.
18. Otherwise recursively construct trees T_1 and T_2 for V_1 and V_2 respectively,
19. where (V_1, V_2) is the best bipartition of V recorded at Step 1.
20. **Step 3: Output the tree.**

Theorem 2. *The algorithm* **Exact_MCTT** *computes the maximum consensus tree from rooted triples with time complexity* $O((m + n^2)3^n)$ *and space* $O(2^n)$.

Proof: The correctness of the algorithm is from Eq. (1). The algorithm computes the scores of subsets with cardinalities from small to large. When computing the score of set V, the scores of all its subsets have been found. The storage space used by the algorithms is $O(2^n + m + n^2)$, $O(2^n)$ for the scores and partitions of all subsets and $O(m + n^2)$ for the triples. Since 2^n is larger than $m + n^2$, the space complexity is $O(2^n)$. For each bipartition (V_1, V_2) of any subset, the time complexity for computing $w(V_1, V_2)$ is no more than $n^2 + m$ since there are totally m triples and $O(n^2)$ pairs (i, j) of species with $i \in V_1$ and $j \in V_2$. Since there are 2^k bipartitions for a set of cardinality k and there are $\binom{n}{k}$ subsets of U with

cardinality k, the time complexity is

$$(n^2 + m) \sum_{k=1}^{n} 2^k \binom{n}{k} = (n^2 + m)(1 + 2)^n = (n^2 + m)3^n$$

□

3.2. *Heuristic algorithms*

In this subsection, we shall present heuristic algorithms for the MCTT problem. The heuristic algorithms do not ensure the optimality of the found solutions but it runs in polynomial time. The performance of the heuristics will be shown by comparing with the optimal solutions found by the exact algorithm represented in the previous subsection.

Our heuristic algorithm **Best-Pair-Merge-First** works as follows: Initially there are n subsets and each contains one of the species. The algorithms then repeatedly merge pair of subsets until there is only one set left. But it is a question to determine the two subsets to be merged at each iteration. We shall define a function $e\_score(V_1, V_2)$ to evaluate the score of merging sets V_1 and V_2. At each iteration, the algorithm chooses the two sets with maximum evaluation score.

To evaluate the score, an intuitive method is to choose sets V_1 and V_2 with maximum $w(V_1, V_2)$. That is, we greedily merge two of the subsets such that the number of triples to be satisfied is as many as possible. Besides the intuitive method, the following two points were also considered and the scoring function is depends on two parameters **if-penalty** and **ratio-type**.

- Merging two set not only satisfies some triples but also makes some triples unsatisfiable. Precisely speaking, merging V_1 and V_2 satisfies the triples $(x(ij))$ but conflicts with the triples $(i(xj))$ and $(j(xi))$, where $i \in V_1$, $j \in V_2$ and $x \notin V_1 \cup V_2$. We define the penalty $p(V_1, V_2)$ as the number of triples conflicted by merging the two sets. When the input parameter **if-penalty** is true, the algorithm uses $w(V_1, V_2) - p(V_1, V_2)$ to select the two sets to be merged. Otherwise only $w(V_1, V_2)$ is considered.
- There may be bias to evaluate the subset pairs by the number of satisfied triples since the distribution of the triples may be not uniform and the cardinalities of the subsets are different while the program is running. Therefore it may be better to use relative score than the number of satisfied triples. Two ratios were considered in our algorithm. One is $w(V_1, V_2)/(w(V_1, V_2) + p(V_1, V_2))$, and the other is $w(V_1, V_2)/t(V_1, V_2)$, in which $t(V_1, V_2)$ is the total number of triples $(x(v_1v_2))$ for all $v_1 \in V_1$ and $v_2 \in V_2$. When the penalty is considered, the numerator is replaced with $w(V_1, V_2) - p(V_1, V_2)$ in either ratio. A parameter **ratio-type** is used to determine which ratio will be used. If it is zero, the algorithm does not use the relative ratio.

The two parameters give us six scoring functions. The performance of all the alternatives were tested. The heuristic algorithm is list below. For different combinations of the two parameter, the function *e_score* is defined in Table 1.

Table 1. The evaluation score $e\_score(V_1, V_2)$ for combinations of parameters.

	Ratio-type		
if-penalty	0	1	2
False	$w(V_1, V_2)$	$\frac{w(V_1,V_2)}{w(V_1,V_2)+p(V_1,V_2)}$	$\frac{w(V_1,V_2)}{t(V_1,V_2)}$
True	$w(V_1, V_2) - p(V_1, V_2)$	$\frac{w(V_1,V_2)-p(V_1,V_2)}{w(V_1,V_2)+p(V_1,V_2)}$	$\frac{w(V_1,V_2)-p(V_1,V_2)}{t(V_1,V_2)}$

1. **Algorithm Best-Pair-Merge-First(if-penalty, ratio-type)**
2. **Step 1: Initialization**
3. Let $\mathcal{T} = \{T_i | \ 1 \le i \le n\}$, in which T_i is the tree contains only one leaf i.
4. **Step 2: Iteratively merging**
5. While there are more than one trees in $\mathcal{T}$ do
6. Select two trees T_i and T_j in $\mathcal{T}$ such that $e\_score(V(T_i), V(T_j))$
7. is maximum, in which $e\_score(V(T_i), V(T_j))$ depends on the
8. parameters **if-penalty** and **ratio-type** as defined in Table 1;
9. Merge T_i and T_j by adding an common ancestor and replace T_i and T_j
10. by the merged tree;
11. **Step 3:** Output the tree in $\mathcal{T}$;

3.3. GJLO's heuristic algorithms

We also implemented the two heuristic algorithms proposed by Gasieniec et al. (1999) and tested their performances. In this subsection, we briefly describe the algorithms, and the experimental results are shown in the next subsection. The two algorithms are called GJLO1 and GJLO2.

While Algorithm **Best-Pair-Merge-First** using a bottom-up strategy, both GJLO1 and GJLO2 construct the tree by recursively top-down splitting. Initially there is only one node labelled by the set of all species. The algorithms find a bipartition of the set and split the node into two children. Then the children are recursively splitted until every node contains only one species. The difference between GJLO1 and GJLO2 is how to find the bipartition.

Algorithm GJLO1 always finds bipartitions of the form (V_1, V_2) in which V_1 or V_2 contains only singleton. For each $v \in V$, it computes

$$\frac{|\{(v(ij)) : i, j \in V\}|}{|\{(i(vj)) : i, j \in V\}|}$$

as the score of v. By choosing the element x with maximum score, the set is bipartitioned into $(\{x\}, V \backslash \{x\})$. Note that $|\{(v(ij))\}|$ is the number of triples which satisfied by the bipartition, and $|\{(i(vj))\}|$ is the number of triples which conflict with the bipartition.

The way that Algorithm GJLO2 finds a bipartition of a set is to minimize the number of conflicted triples. That is, Algorithm GJLO2 finds a bipartition (V_1, V_2) of a set V such that $p(V_1, V_2)$ is minimum, where $p(V_1, V_2)$, as defined in the previous subsetion, is the

number of triples conflicted with the bipartition. In our heuristics, we only need to compute $p(V_1, V_2)$ for given V_1 and V_2, since our algorithms use a bottom-up strategy. However, Algorithm GJLO2 is to find the desired bipartition, and is therefore more complicated. To partition a set V, construct an auxiliary graph $G = (V, E, \lambda)$, where the edge weight $\lambda(i, j) = |\{(v(ij)) : v \in V\}|$ is the number of conflicted triples if i and j are partitioned into different subsets. Obviously the desired bipartition can be found by computing the minimum cut of the auxiliary graph. In the original paper, GJLO2 uses the techniques in Henzinger et al. (1996) and Karger (1996) to reduce the time complexity for finding the minimum cut. Since the time complexity is not our major concern and the number of species is not large in our tests, we implemented the Ford and Fulkerson's algorithm for the minimum cut (Ford and Fulkerson, 1962; Liu, 1998).

3.4. The experimental results

3.4.1. The environment of the experiments. Both the exact and heuristic algorithms were coded in **ANSI C** and ported on a personal computer equiped with Intel Pentium III-733 CPU and 64M bytes memory. The platform is Microsoft WIN32. The triples were generated randomly over all species.

3.4.2. Running time. We tested the running time for the exact algorithm for n from 10 to 20. Since the algorithm uses the dynamic programming strategy. The running time does not vary for different instances. For each n, three data instances were tested. The results are shown in Table 2.

3.4.3. Error ratios. The performances of the heuristic algorithms are shown in the following tables. Tables 3 and 4 show the worst ratios for different numbers of triples. For each case, over 100 data were tested. The error ratio is obtained by $opt(Y)/heu(A, Y)$, where

Table 2. The running time for Algorithm **Exact_MCTT**.

n	10	11	12	13	14	15	16	17	18	19	20
Time in sec.	<1	1	2	9	30	104	366	1255	4322	14690	49923

Table 3. The worst error ratios for different numbers of triples with $n = 10$.

if-penalty	Without penalty			With penalty					
ratio-type	0	1	2	0	1	2	BPMF	GJLO1	GJLO2
$m = 60$	1.650	1.276	1.619	1.240	1.276	1.360	1.192	1.440	1.737
$m = 80$	1.464	1.206	1.615	1.188	1.206	1.464	1.182	1.303	1.593
$m = 100$	1.400	1.179	1.621	1.225	1.179	1.265	1.150	1.257	1.667
$m = 120$	1.514	1.256	1.556	1.256	1.256	1.343	1.136	1.227	1.583

Table 4. The worst error ratios for different numbers of triples with $n = 15$.

if-penalty	Without penalty			With penalty					
ratio-type	0	1	2	0	1	2	BPMF	GJLO1	GJLO2
$m = 100$	1.450	1.208	1.657	1.191	1.208	1.250	1.191	1.283	1.595
$m = 200$	1.373	1.200	1.507	1.186	1.200	1.200	1.141	1.238	1.365
$m = 300$	1.365	1.159	1.467	1.196	1.159	1.159	1.126	1.218	1.387

$opt(Y)$ is the maximum number of satisfiable triples in Y and $heu(A, Y)$ is the number of triples satisfied by the tree found by heuristic algorithm A. The column labeled by **BPMF** shows the results for the algorithm which runs all the six heuristics and chooses the best for each data instance. Tables 5 and 6 show the average and worst ratios in the tests for different number of species. The number of the tested data is 1200 for $n = 10$, and 300 for $n = 12$, 15, and 30 for $n = 18$, and 35 for $n = 20$. Table 7 shows the ratio (in percentage) of the instances that BPMF performed better than GJLO and vice versa, in which GJLO is to run both GJLO1 and GJLO2 and to choose the better result for each data instance. For example, in our tests with $n = 10$, BPMF got better results than the ones by GJLO on 69.5% of the data instances, while GJLO is better than BPMF on only 16.1% of the instances. Note that, on the remaining 14.4% of the instances, the two algorithms tied.

Table 5. The average error ratios for different numbers of species.

if-penalty	Without penalty			With penalty					
ratio-type	0	1	2	0	1	2	BPMF	GJLO1	GJLO2
$n = 10$	1.176	1.068	1.245	1.070	1.068	1.086	1.054	1.106	1.210
$n = 12$	1.208	1.086	1.272	1.090	1.086	1.103	1.069	1.122	1.232
$n = 15$	1.210	1.086	1.269	1.093	1.086	1.098	1.071	1.122	1.218
$n = 18$	1.210	1.098	1.268	1.101	1.098	1.114	1.082	1.118	1.223
$n = 20$	1.228	1.109	1.306	1.104	1.109	1.121	1.085	1.146	1.242

Table 6. The worst error ratios for different numbers of species.

if-penalty	Without penalty			With penalty					
ratio-type	0	1	2	0	1	2	BPMF	GJLO1	GJLO2
$n = 10$	1.650	1.276	1.621	1.256	1.276	1.464	1.192	1.440	1.737
$n = 12$	1.630	1.280	2.000	1.231	1.280	1.292	1.200	1.375	2.615
$n = 15$	1.450	1.208	1.657	1.196	1.208	1.250	1.191	1.283	1.595
$n = 18$	1.348	1.235	1.500	1.170	1.235	1.286	1.127	1.265	1.455
$n = 20$	1.373	1.208	1.571	1.167	1.208	1.208	1.139	1.314	1.535

Table 7. The comparison between **BPMF** and **GJLO**.

	Number of species				
Algorithm	10 (%)	12 (%)	15 (%)	18 (%)	20 (%)
BPMF is better	69.5	70.0	80.0	73.3	74.3
GJLO is better	16.1	16.7	15.3	20.0	20.0

4. Discussion

In the following paragraphs, the heuristics will be referred as BPMF (p_1,p_2), in which the p_1 and p_2 are the input parameters. By the results of experiments, we observed the following:

- By the results of individual data instances (not shown in the paper), we found that no one of the eight heuristics (including 6 alternatives of BPMF, GJLO1, and GJLO2) is absolutely better than another. For each of them, there exist some instances that it finds better solutions than all the others. This is also the reason why the heuristic **BPMF** performs better than all the others.
- The error ratios are not sensitive to either the number of input triples or the number of species.
- Taking penalty into consideration improves the performance significantly. Note that the evaluation score of BPMF (no-penalty, ratio-type = 1) in fact involves the penalty.
- Heuristics BPMF (no-penalty, ratio-type = 1) and BPMF (penalty, ratio-type = 1) perform very similarly. In over thousands of tests, there are only few cases that the scores of their outputs are different.
- BPMF performed better than GJLO on about 70% of the data instances.

We make some remarks as the conclusion. In most of the applications, the solution quality is the major concern. Therefore, for small data instances, the exact algorithm should be used. For large data instances, we propose the heuristic BPMF since it takes the advantages of the six heuristics and runs in polynomial time. Furthermore, combining BPMF with GJLO may give us even better solutions. For example, in our tests with $n = 15$, the worst error ratio of the combined algorithm is 1.157, while the ratios of BPMF, GJLO1 and GJLO are 1.191, 1.283 and 1.595 respectively. When the running time is an important factor, any one of the heuristics with penalty considered may be a good choice.

In the exact algorithm presented in Section 3, we need to compute and keep the scores of all subsets. The time complexity is therefore exponential. While computing the subsets of some cardinality, if we keep only certain, but not all, subsets, we may have an algorithm which is adaptable in the computational time and the solution quality. It may be a good direction of future researches.

There are also some open problems. We show the performances of the heuristics by experiments. It is interesting to give a theoretic analysis of the performance. The computational complexity of the MCTT problem is shown in this paper, but the approximability is still open.

Acknowledgment

We thank the anonymous referees for their careful reading and many useful suggestions. This work is supported in part by an NSC grant NSC89-2218-E-366-003.

References

A.V. Aho, Y. Sagiv, T.G. Szymanski, and J.D. Ullman, "Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions," *SIAM Journal on Computing*, vol. 10, no. 3, pp. 405–421, 1981.

L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press: Princeton, New Jersey, 1962.

M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company: San Fransisco, 1979.

L. Gasieniec, J. Jansson, A. Lingas, and A. Östlin, "On the complexity of constructing evolutionary trees," *Journal of Combinatorial Optimization*, vol. 3, pp. 183–197, 1999.

M.R. Henzinger, V. King, and T. Warnow, "Constructing a tree from homeomorphic subtrees, with applications to computational biology," in *Proc. of the 7th ACM-SIAM SODA*, 1996, pp. 333–340.

J. Jansson, "On the complexity of inferring rooted evolutionary trees," in *Proceedings of the Brazilian Symposium on Graph, Algorithms, and Combinatorics* (GRACO01), Fortaleza, Electronic Notes in Discrete Mathematics, vol. 7, Elsevier, 2001, pp. 121–125.

D.R. Karger, "Minimum cuts in near-linear time," in *Proc. of the 28th ACM STOC*, 1996, pp. 56–63.

R.M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (Eds.), Plenum Press: New York, 1972, pp. 85–103.

C.L. Liu, *Elements of Discrete Mathematics*, 2nd edn. McGraw-Hill: New York, 1998.

M.P. Ng and N.C. Wormald, "Reconstruction of rooted trees from subtrees," *Discrete Applied Mathematics*, vol. 69, pp. 19–31, 1996.

M. Steel, "The complexity of reconstructing trees from qualitative characters and subtrees," *Journal of Classification*, vol. 9, pp. 91–116, 1992.