JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 20, 181-190 (2004)

Short Paper

Constructing Evolutionary Trees from Rooted Triples*

BANG YE WU
Department of Computer Science and Information Engineering
Shu-Te University
Kaohsiung, 824 Taiwan

In this paper, we propose a new heuristic algorithm for the maximum consensus tree of rooted triples. By means of the experimental results, we show that the algorithm is better than the three previous heuristics and runs in a reasonable amount of time. Furthermore, using the algorithm, it is possible to achieve a trade-off between the running time and the quality of the solution.

We also investigate the computational complexity of the maximum compatible set problem. We show that it is NP-hard to find the maximum vertex set compatible with given rooted triples.

Keywords: computational biology, evolutionary tree, heuristic algorithm, NP-hard, consensus tree

1. INTRODUCTION

Evolutionary trees are used to present the relationships among a set of species. An evolutionary tree is a rooted tree, in which each of the leaves corresponds to one species, and each of the internal nodes is the inferred common ancestor of the species in the subtree. Constructing evolutionary trees is an important problem in computational biology, and there are different approaches to it. A rooted triple, or simply triple for the sake of brevity, represents the relationship between three species. A triple ($a(bc)$) specifies $lca(a, b) = lca(a, c) \rightarrow lca(b, c)$, in which $lca(a, b)$ is the lowest common ancestor of the two leaves and the relation $\rightarrow$ means "*is an ancestor of*". We say that a tree satisfies a triple or that a triple is compatible with a tree if the relationship represented by the triple is satisfied in the tree. A triple set is compatible if there exists a tree that satisfies all the triples in the set, and this tree is called as the *exact consensus tree*.

Given a set of triples, the existence of an exact consensus tree can be determined in polynomial time. For a set of constraints of the form $lca(a, b) \rightarrow lca(c, d)$, the algorithm developed in [1] can determine if there is a tree satisfying all the constraints , and it finds such a tree if it exists. A triple ($a(bc)$) is equivalent to $lca(a, c) \rightarrow lca(b, c)$; therefore, the problem of determining the existence of an exact consensus tree from triples is also

Received January 31, accepted July 4, 2003.
Communicated by Ming-Syan Chen.
*An earlier version of this paper has been presented at the 2002 International Computer Symposium, 2002.

polynomial-time solvable. An algorithm for constructing all exact consensus trees from triples has also been developed [6]. Unfortunately, it is often the case that the given triples are not compatible, in which case it is impossible to find the exact consensus tree. This motivated us to study the optimization problems of consensus trees.

We considered two optimization problems. Given a set Y of triples over species set V, the *Maximum Consensus Tree with Triples* (MCTT) problem is to construct a tree with leaf set V such that there are as many satisfied triples as possible, and the *Maximum Compatible Set with Triples* (MCST) problem is to find the compatible species subset of maximum cardinality. A species subset U is compatible with a triple set Y if there exists a tree with leaf set U such that all the triples over U are satisfied. As an example, Fig. 1 illustrates the MCTT problem of four triples over four species. The four triples are not compatible since there does not exist any evolutionary tree satisfying all four triples. The maximum consensus tree shown in the figure satisfies all the triples except (c(bd)). Set {a, b, c} is a maximum compatible set since there is only one triple (a(bc)) over the three species, thus, the set is obviously compatible. In fact, in this example, any subset of three species is a maximum compatible set.

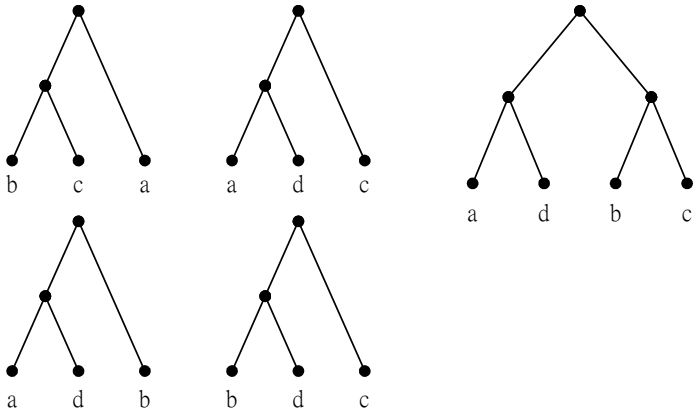

Fig. 1. (a) The rooted trees corresponding to the four rooted triples (a(bc)), (c(ad)), (b(bd)). (b) The maximum consensus tree of the triples.

The problem of finding the maximum consensus tree from constraints of the form $lca(a, b) \rightarrow lca(c, d)$ was shown to be NP-hard, and a 3-approximation algorithm was proposed [3]. The approximation algorithm also can be applied to the MCTT problem, but the complexity of the MCTT problem was left open. Recently, it has been shown that the MCTT problem is also NP-hard, and exact and heuristic algorithms have been developed [8]. The NP-hardness of the MCTT problem was also shown by Jansson independently [4]. Similar problems for unrooted trees have also been investigated. A quartet represents the relationship between four species on an unrooted tree. The problem of determining if there is a tree satisfying a given set of quartets has been shown to be NP-complete [7]. Therefore, the corresponding optimization problem is obviously NP-hard.

In this paper, we propose another heuristic algorithm for the MCTT problem. The performance of the algorithm was tested by means of experiments. Based on the experimental results, the heuristic algorithm is better than the previously proposed ones and runs in a reasonable amount of time. We also show here that the MCST problem is NP-hard.

2. A HEURISTIC ALGORITHM FOR THE MCTT PROBLEM

In this section, we present a heuristic algorithm for the MCTT problem. The algorithm is derived from the exact algorithm [8], and the performance is analyzed by comparing with the exact algorithm and with previous heuristic algorithms. For completeness, we first briefly introduce those algorithms.

2.1 The Exact Algorithm

The algorithm **ExactMCTT** uses the dynamic programming strategy and is based on the following formula:

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

Let U be the set of all considered species. For a subset $V \subset U$, B(V) is the set of all bipartitions of V. The value $score(V)$ is the maximum number of satisfiable triples over V, and $w(V_1, V_2)$ of two disjoint sets represents the number of triples ($x(v_1v_2)$) in which $v_1 \in V_1$, $v_2 \in V_2$ and $x \in U \backslash V$. The algorithm computes the score of a set by trying all of its bipartitions. A set V corresponds to an internal node of the evolutionary tree, and two subsets of the best bipartition correspond to the two subtrees of the internal node. By computing the scores of subsets with cardinalities from small to large, the algorithm takes $O((m + n^2)3^n)$ time and $O(2^n)$ space, in which m and n are the numbers of triples and species respectively.

2.2 Previous Heuristics

We introduce the following three heuristics proposed in the previous papers.

- **BOSF:** The Best-One-Split-First algorithm [3] uses the top-down splitting strategy. The algorithm repeatedly splits the species set into bipartitions of the form (V_1, V_2), in which V_1 contains only a singleton. Therefore the algorithm always constructs a *linear* tree. In each iteration, the split species is chosen greedily by finding the maximum ratio of the number of satisfied triples to the number of conflicted triples.
- **MCSF:** The Min-Cut-Split-First algorithm [3] also uses the top-down splitting strategy. The algorithm is derived from the exact algorithm for compatible triples [1]. For compatible triples, it is always possible to find a bipartition without conflicting any triple in each iteration. For incompatible triples, the MCSF algorithm repeatedly splits the species set into bipartitions such that the number of conflicted triples is minimized. The bipartition in each iteration is found by computing the minimum cut of an auxiliary graph.

- **BPMF:** The Best-Pair-Merge-First algorithm [8] uses the bottom-up merging strategy. The algorithm repeatedly merges two subtrees with best scores. Six different scoring functions were tested. Basically, in each iteration, it tries to maximize the number of satisfied triples and to minimize the number of conflicted triples. It was reported that none of the six scoring functions is absolutely better than the others. In our experiments, we ran the six algorithms for each data instance and took the best one as the result of the algorithm.

2.3 The Heuristic Algorithm

The Dynamic-Programming-With-Pruning (DPWP) algorithm, the heuristic algorithm we propose in this paper, is derived from the exact algorithm with the dynamic programming strategy. The exact algorithm runs in exponential time since the number of subsets is exponential. Instead of all the subsets, we use an array Q_i to keep at most K subsets for each possible cardinality i. The algorithm merges the subsets with cardinalities from small to large. When a subset V of cardinality i is considered, it is merged with each of the subsets in Q_j for each $j \le i$. The resulting set is then considered for placement into the array. If the set is already in the array, we keep the best score of the set, otherwise into the array. However, if the array is full, the set with the minimum score is discarded. Instead of $score(V)$ defined in Eq. (1), the scoring function to measure the goodness of merging two set V_1 and V_2 is

$$score_2(V_1, V_2) = \frac{w(V_1, V_2) - c(V_1, V_2)}{w(V_1, V_2) + c(V_1, V_2)},$$

in which $w(V_1, V_2)$ is the number of satisfied triples as defined in Eq.(1) and $c(V_1, V_2)$ is the number of triples conflicted by merging V_1 and V_2. Precisely speaking, $c(V_1, V_2)$ is the number of triples of the form $(i(xj))$ or $(j(xi))$, in which $i \in V_1$, $j \in V_2$, and $x \notin V_1 \cup V_2$. The score of two intersecting sets is $-\infty$ since the merge is invalid. The algorithm is as follows.

Algorithm DPWP(*K*)
Input: A set Y of rooted triples over species set U of cardinality n.
All triples are stored in a matrix M of lists. $M[i, j]$ is a list of the elements of set $\{x \mid (x(ij)) \in Y\}$.
Output: A rooted evolutionary tree T.
Step 1: (Initialization)
Array Q_1 contains all subsets of singleton, and Q_i is empty for $2 \le i \le n$.
For each subset V in the arrays, $score_2(V)$ is the currently best score and $partition(V)$ is the bipartition corresponding to $score_2(V)$.
Step 2: (Compute the number of satisfied triples)
For $i = 1$ to $n - 1$ do
For $j = 1$ to i do
For each V_1 in Q_i and V_2 in Q_j do
Compute the score $score_2(V_1, V_2)$;
Search Q_{i+j} for $V = V_1 \cup V_2$;

If V exists, keep the better score
else put V into Q_{i+j};
If $|Q_{i+j}| > K$, delete the set with smallest score;

Step 3: Construct the tree by backtracking *partition*(U).
Step 4: Output the tree.

The complexity of the algorithm is given in the next theorem. Since the proof is obvious, we ignore it here.

Theorem 1 The algorithm DPWP runs in $O(n^2K^2(n^3 + K))$ time and uses $O(nK)$ space.

2.4 The Experimental Results

2.4.1 The environment of the experiments

Both the exact and heuristic algorithms were coded in the **C** language and ported to a personal computer equipped with an Intel Pentium IV-1.8 CPU and 128M bytes of memory. The platform was Microsoft WIN32. The triples were generated randomly over all species. In this subsection, n is the number of species, m is the number of triples, and K is the array size of the DPWP algorithm. We ran the exact algorithm only for $n \leq 20$, and the other heuristics for $n \leq 30$. For the exact algorithm with $n = 20$, only a few instances were tested. For the other cases, hundreds of data were tested.

2.4.2 Running time

The heuristic algorithms BOSF, MCSF, and BPMF ran quickly. In all of our tests, the number of species was no more than 30, and the three algorithm obtained results within one second. We measured the running time of the exact algorithm with n ranging from 12 to 20, and the time of DPWP with n ranging from 12 to 30. The results are shown in Table 1, in which DPWP(K) means the algorithm DPWP with array size K.

Table 1. The running time (second).

n	12	15	18	20	24	27	30
Exact	1	18	752	8314	NA	NA	NA
DPWP(300)	1	2	5	7	16	21	34
DPWP(600)	2	7	13	19	54	84	130
DPWP(900)	3	15	34	78	128	201	273

2.4.3 Performances

The performance ratios of the heuristic algorithms are shown in the following tables, The ratio was obtained using $opt(Y)/heu(A, Y)$, where $opt(Y)$ 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. Table 2 shows the worst ratios for different numbers of triples. Tables 3 and 4 show the average and worst ratios for different numbers of species. Table 5 shows how much the DPWP algorithm improved the previous heuristics. The ratio (in

Table 2. The worst error ratios for different numbers of triples with $n = 15$.

	BPMF	BOSF	MCSF	DPWP(300)	DPWP(600)	DPWP(900)
$m = 200$	1.1630	1.2250	1.4000	1.0106	1.0000	1.0000
$m = 400$	1.1081	1.1486	1.3248	1.0066	1.0063	1.0000
$m = 600$	1.0885	1.1270	1.2321	1.0048	1.0000	1.0000

Table 3. The average error ratios for different numbers of species.

	BPMF	BOSF	MCSF	DPWP(300)	DPWP(600)	DPWP(900)
$n = 12$	1.0511	1.0835	1.1699	1.0000	1.0000	1.0000
$n = 15$	1.0635	1.0932	1.1889	1.0003	1.0000	1.0000
$n = 18$	1.0676	1.0903	1.1614	1.0008	1.0005	1.0001
$n = 20$	1.0849	1.0920	1.1838	1.0026	1.0000	1.0000

Table 4. The worst error ratios for different numbers of species.

	BPMF	BOSF	MCSF	DPWP(300)	DPWP(600)	DPWP(900)
$n = 12$	1.1707	1.2727	1.5484	1.0000	1.0000	1.0000
$n = 15$	1.1630	1.2250	1.4000	1.0106	1.0063	1.0000
$n = 18$	1.1463	1.1870	1.3738	1.0084	1.0068	1.0068
$n = 20$	1.1111	1.1301	1.2222	1.0078	1.0000	1.0000

Table 5. The improvement by DPWP.

n	18	21	24	27	30
Max	10.0 %	14.0 %	14.3 %	15.9 %	14.6 %
average	6.0 %	7.3 %	8.3 %	9.0 %	9.3 %

percentage) is calculated using $(x - y)/y$, in which x is the result (the number of satisfied triples) obtained by DPWP and y is the best of the results obtained by BPMF, BOSF, and MCSF.

2.5 Discussion

Based on the experimental results, we make the following observations.

- For all data used in our tests, the DPWP algorithm performed better than the previously proposed heuristics.
- The DPWP algorithm found the optimal solution in most of the cases with a small number of species. In our tests, the success rate of DPWP(900) finding the optimal solution was 100% for $n = 12$, 99.3% for $n = 15$, and 98% for $n = 18$.

- The running time of the DPWP algorithm was much more reasonable than that of the exact algorithm.
- Using the DPWP algorithm, it was possible to achieve a trade-off between the running time and the quality of the solution.

3. THE COMPUTATIONAL COMPLEXITY OF MCST

In this section, we shall show the NP-hardness of the MCST problem by reducing the Feedback Vertex Set problem to it. We will first define the Feedback Vertex Set problem.

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

The Feedback Vertex Set problem is NP-complete [2, 5].

Definition 2 Let Y be a set of triples over vertex set V, and let $U \subset V$. The reduced triple set Y_U is the subset of triples over U, i.e., $Y_U = \{(a(bc)): a, b, c \in U\} \cap Y$. A vertex set U is compatible with Y if the reduced triple set Y_U is compatible.

Definition 3 Given a set Y of rooted triples over species set V, the maximum compatible set with triples (MCST) problem looks for a subset U of V such that U is compatible with Y and the cardinality of U is the maximum.

The computational complexity is shown in the next theorem.

Theorem 2 The MCST problem is NP-hard.

Proof: We reduce the Feedback Vertex Set problem to the MCST problem. Given an instance $G = (V, A)$ and k of the Feedback Vertex Set problem, we construct a set of rooted triples Y and show that the directed graph G contains a feedback vertex set of cardinality k if and only if there is a compatible vertex set of cardinality $2n - k$, where $n = |V|$.

Let $x_i \notin V$, $1 \leq i \leq n$. For every arc $(u, v) \in A$, we construct n corresponding triples $(u(x_iv))$ in Y, where $1 \leq i \leq n$. Suppose that U is a feedback vertex set of G and $|U| = k$. Removing U and all arcs incident to any vertex in U from G results in a directed acyclic graph $G_1 = (V\backslash U, A_1)$. Since G_1 contains no cycle, we may assign to 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 the number of nodes of the longest path in G_1. Let $V_i = \{v| f(v) = i, v \in V\backslash U\}$ 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 shown in Fig. 2.

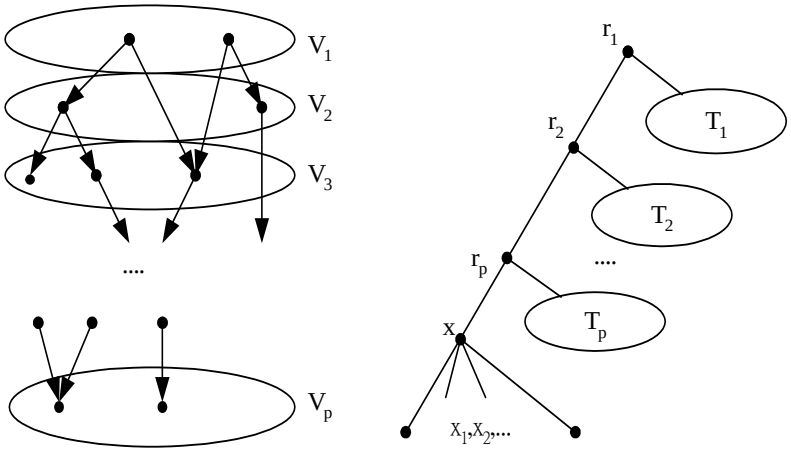

Fig. 2. Transformation of an instance of the Feedback Vertex Set problem into that of the MCST problem. (a) the labeling of a directed acyclic graph; (b) a maximum consensus tree of the MCST problem.

For any arc $(u, v) \in A_1$, since $f(u) < f(v)$, the corresponding triples $(u(x_i v))$ in Y are compatible with T. Therefore, all the triples corresponding to arcs in A_1 are satisfied, and the cardinality of the compatible set is $|V \backslash U| + |X| = 2n - k$.

Conversely, suppose that the cardinality of the maximum compatible set is $2n - k$. Let $U = U_1 \cup X_1$ be the maximum compatible set, where $U_1 \subset V$ and $X_1 \subset X$. First, we will show that $X_1 = X$. If X_1 is empty, the cardinality of U_1 is at most n. However, there is a trivial compatible set consisting of X and any two vertices in V. We conclude that X_1 is not empty.

If there exists some $x_i \notin X_1$ and $x_j \in X_1$, then U_1 is not the maximum since we may insert x_i into the tree without conflicting any triple. Consequently, $X_1 = X$.

As shown in Fig. 2, we let the path from the root to x be $(r_1, r_2, \ldots, r_p, x)$, and let V_i denote the set of leaves whose lowest common ancestor with x is r_i. For each triple $(u(xv)) \in Y_U$, in which $u \in V_i$ and $v \in V_j$, since $lca(u, x) = lca(u, v) \rightarrow lca(x, v)$, we have $j > i$. Let A_1 be the set of arcs corresponding to the triples in Y_U, that is, $A_1 = \{(u, v) | (u(xv)) \in Y_U\}$. Consider the graph $G_1 = (U_1, A_1)$ and label each vertex v with i if $v \in V_i$. Since all the arcs in A_1 extend from vertices with small labels to those with larger labels, G_1 contains no directed cycle. Therefore, $V \backslash U_1$ is a feedback vertex set of G and contains k vertices.

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

4. CONCLUDING REMARKS

In this paper, we have proposed a new heuristic algorithm DPWP for the MCTT problem. By means of the experimental results, we have shown that the algorithm per-

forms better than the previously proposed heuristics and runs in a reasonable amount of time. The DPWP algorithm can be easily modified to work so that it can be applied to the weighted version of the MCTT problem, in which each triple has a weight and we want to find the tree such that the total weight of the satisfied triples is maximized. All the algorithms used in this study can be extended to the case where the input is a set of trees not restricted to triples by transforming the input trees into triples. However, the result is a tree satisfying the maximum number of triples but not the number of input trees.

The exact algorithm for the MCTT problem also can be applied to the MCST problem. Since the decision version of the MCST problem is polynomial-time solvable, another approach to obtaining the exact solution of the MCST problem is to determine the compatibility of each subset. Good heuristic and approximation algorithms would be of interest in this regard.

REFERENCES

1. 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, 1981, pp. 405-421.
2. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.
3. L. Gasieniec, J. Jansson, A. Lingas, and A. Ostlin, "On the complexity of computing evolutionary trees," in *Proceedings of the 3rd Annual International Computing and Combinatorics Conference* (*COCOON '97*), 1997, pp. 134-145.
4. J. Jansson, "On the complexity of inferring rooted evolutionary trees," in the *Proceedings of the Brazilian Symposium on Graph, Algorithms, and Combinatorics*, Vol. 7, 2001, pp. 121-125.
5. R. M. Karp, "Reducibility among combinatorial problems," in R. E. Miller and J. W. Thatcher (eds.) *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85-103.
6. M. P. Ng and N. C. Wormald, "Reconstruction of rooted trees from subtrees," *Discrete Applied Mathematics*, Vol. 69, 1996, pp. 19-31.
7. M. Steel, "The complexity of reconstructing trees from qualitative characters and subtrees," *Journal of Classification*, Vol. 9, 1992, pp. 91-116.
8. B. Y. Wu, "Constructing the maximum consensus tree from rooted triples," to appear in *Journal of Combinatorial Optimization*.

Bang Ye Wu (吳邦一) was born on May 19, 1964 in Kaohsiung, Taiwan. He received the B.S. degree in 1986 from the Department of Electrical Engineering at Chung Cheng Institute of Technology. He received the M.S. and the Ph.D. degrees in Computer Science from National Tsing Hua University in 1991 and 1999. During 1986-1989, 1991-1995, and 1999-2000, he worked at Chung-Shan Institute of Science and Technology. Since 2000, he is presently an assistant professor in the Department of Computer Science and Information Engineering at Shu-Te University. His research interests include algorithms, graph theory and computational biology.