Constructing evolutionary trees from rooted triples

Submitted to

Workshop on Algorithms and Computational Molecular Biology

Corresponding author:

Bang Ye Wu

Dept. of Computer Science and Information Engineering,

Shu-Te University, YenChau, Kaohsiung, Taiwan 824, R.O.C.

E-mail: bangye@mail.stu.edu.tw

TEL: 07-6151000-4612

FAX: 07-6151000-4699

Abstract

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

Key Words: computational biology, evolutionary tree, heuristic algorithm,

NP-nard

Constructing evolutionary trees from rooted triples

Bang Ye Wu
Dept. of Computer Science and Information Engineering,
Shu-Te University, YenChau, Kaohsiung, Taiwan 824, R.O.C.
E-mail: bangye@mail.stu.edu.tw

1 Introduction

Evolutionary trees are used to present the relationship 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. A rooted triple, or triple for brevity, represents the relationship of 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 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 satisfies all the triples in the set, and the tree is called as the *exact consensus tree.*

Given a set of triples, the existence of the 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 [1] determines if there is a tree satisfying all the constraints and finds such a tree if it exists. A triple $(a(bc))$ is equivalent to $lca(a,c) \rightarrow lca(b,c)$ and therefore the problem of determining the existence of the exact consensus tree from triples is also polynomial-time solvable. An algorithm for constructing all exact consensus trees from triples was also developed [6]. Unfortunately, it is often that the given triples are not compatible and it is impossible to find the exact consensus tree. It motivates the study of the optimization problems of the consensus trees.

We considered two optimization problem. Given a set Y of triples over species set V, the *maximum consensus tree* (MCTT) problem is to construct a tree with leaf set V such that the satisfied triples is as many as possible, and the *maximum compatible set* (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, Figure 1 illustrates the MCTT of four triples over four species. The four triples are not compatible since there does not exists any evolutionary tree satisfying all the 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 and the set is obviously compatible. In fact, in this example, any subset of three species is a maximum compatible set.

The problem to find 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

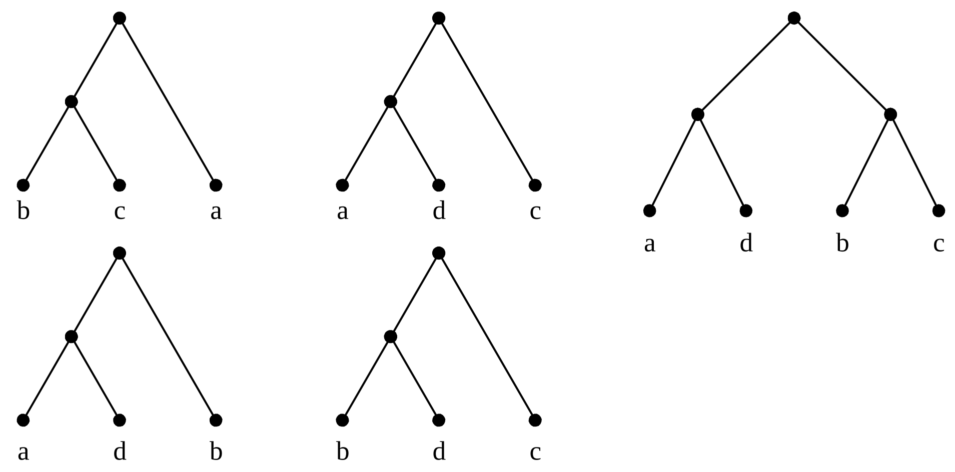

Figure 1: The maximum consensus tree of four rooted triples $(a(bc))$, $(c(ad))$, $(b(ad))$, and $(c(bd))$.

algorithm was proposed [3]. The approximation algorithm also works for the MCTT problem but the complexity of the MCTT problem was left open. Recently it was shown that the MCTT problem is also NP-hard, and exact and heuristic algorithms were developed [8]. The NP-hardness of the MCTT problem was also shown by Jansson independently [4]. Similar problems for unrooted trees were also investigated. A quartet represents the relationship of four species on an unrooted tree. To determine if there is a tree satisfying a given set of quartets were 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 was tested by experiments. By the experimental results, the heuristic algorithm is better than the previous ones and runs in reasonable time. We also show that the MCST problem is NP-hard.

2 A heuristic algorithm for the MCTT

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

2.1 The exact algorithm

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

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

For a species set V, $\mathcal{B}(V)$ 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 is the number of triples $(x(v_1v_2))$ in which $v_1 \in V_1$, $v_2 \in V_2$ and $x \notin V_1 \cup V_2$. 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.

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 split the species set into bipartitions of the form (V_1, V_2) in which V_1 contains only singleton. Therefore the algorithm always construct 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 uncompatible triples, the MCSF algorithm repeatedly split the species set into the bipartition 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 score. There are six different scoring functions are tested in the paper. 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 the paper, is derived from the exact algorithm with dynamic programming strategy. The exact algorithm runs in exponential time since the number of subsets is exponential. Instead of all 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 $j \leq i$. The resulted set is then considered to be put into the array. If the set is already in the array, we keep the best score of the set. Otherwise it is put into the array. However, if the array is full, the set with minimum score is discarded. The scoring function we used is $\frac{s-c}{s+c}$, in which s is the number of satisfied triples and c is the number of conflicted triples. 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 $\{x|(x(ij)) \in Y\}$.

Output: A rooted evolutionary tree T.

Step 1: (Initialization)

```
Array $Q_1$ contains all subsets of singleton,
and $Q_i$ is empty for $2 \leq i \leq n$.
For each subset $V$ in the arrays, $score(V)$ is the currently best
score and $partition(V)$ is the bipartition corresponding to $score(V)$.
```

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 $score(V_1, V_2)$;
            Search $Q_{i+j}$ for $V = V_1 \cup V_2$;
            If $V$ already 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 lemma. Since it is obvious, we ignore the proof.

Lemma 1: The algorithm DPWP runs in $O(n^2K^3)$ 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 **C** language and ported on a personal computer equipped with Intel Pentium IV-1.8 CPU and 128M bytes memory. The platform is 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 few instances were tested. For the other cases, over hundreds of data were tested.

2.4.2 Running time

The heuristic algorithms BOSF, MCSF, and BPMF run quickly. In all of our tests, the number of species is no more than 30, and the three algorithm obtained results within one second. We measured the running time of the exact algorithm for n from 12 to 20, and the time of DPWP for n from 12 to 30. The result is shown in Table 1.

Table 1: The running time (second)

	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)	NA	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 is obtained by $opt(Y)/heu(A, Y)$, 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. The column labeled by DPWP(K) shows the results for the algorithm DPWP with array size K. Table 2 shows the worst ratios for different numbers of triples. Table 3 and 4 show the average and worst ratios for different number of species. Table 5 shows how much the DPWP algorithm improves the previous heuristics. The ratio (in percentage) is calculated by $(x - y)/y$, in which x is the result (number of satisfied triples) by the DPWP and y is the best one of the results by BPMF, BOSF, and MCSF.

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	1.0000	1.0063
$m = 600$	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

2.5 Discussion

By the experimental results, we observed the following.

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	1.0000	1.0064
$n = 18$	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%

- For all data in our tests, the DPWP algorithm performs better than the previous heuristics.
- The DPWP algorithm finds optimal solution in most of the cases for small data instances. In our tests, the percentages of that the DPWP(900) found the optimal solution are 100% for $n = 12$, 99.3% for $n = 15$, and 98% for $n = 18$.
- The running time of the DPWP algorithm is much more reasonable than that of the exact algorithm.
- By the DPWP algorithm, it is possible to make trade-off between the running time and the quality of 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 first give the definition of 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 [5, 2].

Definition 2: Let Y be a set of triples over vertex set V and $U \subset V$. 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 vertex set (MCST) problem looks for a subset U of V such that U is compatible with Y and the cardinality of U is 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 Arc 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 $(u(x_i v))$ 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 $G_1 = (V \setminus U, A_1)$. Since G_1 contains no cycle, we may assign each vertex v a label $f(v) \in \{1 \dots p\}$ 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 $V_i = \{v | f(v) = i\}$ 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 triples $(u(x_i v))$ in Y is compatible with T. Therefore all triples corresponding to arcs in A_1 are satisfied, and the cardinality of the compatible vertex set is $|V \setminus U| + |X| = 2n - k$.

Conversely suppose that the cardinality of the maximum compatible vertex 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 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 consisted 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 $x_j \in X_1$, U_1 is not maximum since we may insert x_i into the tree without conflicting any triple. Consequently $X_1 = X$.

As in Figure 2, let the path from root to x be $(r_1, r_2, \dots, r_p, x)$ and 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 $lca(u, x) = lca(u, v) \to lca(x, v)$, 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

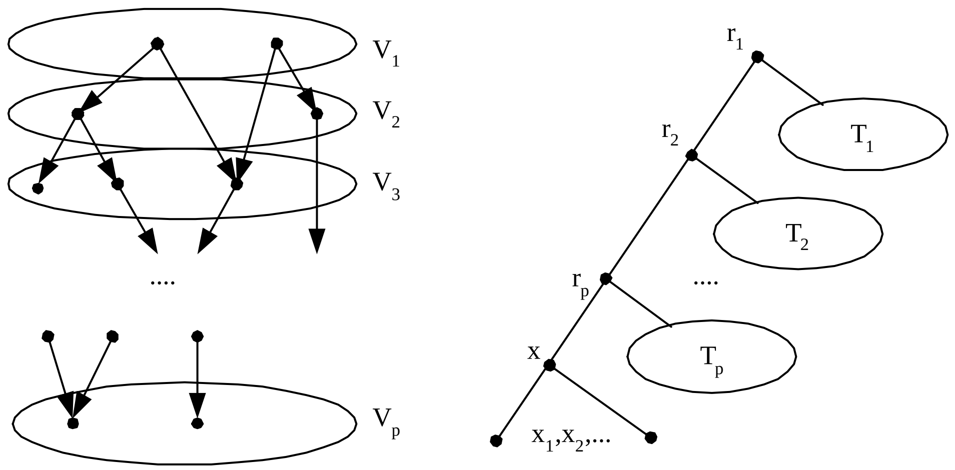

Figure 2: Transformation of an instance of the Feedback Vertex Set problem into that of the MCST problem. Left: the labeling of a directed acyclic graph; Right: A maximum consensus tree of the MCST problem.

$G_1 = (U_1, A_1)$ and label each vertex v with i if $v \in V_i$. Since all the arcs in A_1 are from vertices with small labels to larger labels, G_1 contains no directed cycle. Therefore $V \setminus U_1$ 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 propose a new heuristic algorithm DPWP for the MCTT problem. By the experimental results, we show that the algorithm performs better than the previous heuristics and runs in reasonable time. The DPWP algorithm can be easily modified to work for 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 in the paper can be extended to the case that 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 maximum number of triples but not number of input trees. We are going to open the source program when the release version is completed.

The exact algorithm for the MCTT also works for the MCST. Since the decision version of the MCST is polynomial-time solvable, another approach to the exact solution of the MCST is to determine the compatibility of each subset. Good heuristic and approximation algorithms will be interesting.

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, no. 3, pp. 405–421, 1981.

[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 Fransisco, 1979.

[3] L. Gasieniec, J. Jansson, A. Lingas and A. Ostlin, On the complexity of computing evolutionary trees, in *Proceedings of the 3th Annual International Conference COCOON'97*, pp.134–145, 1997.

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

[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, pp. 85–103, 1972.

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

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

[8] B.Y. Wu, Constructing the maximum consensus tree from rooted triples, to appear in *Journal of Combinatorial Optimization.*