Approximation Algorithms for Constructing Evolutionary Trees from Rooted Triplets

Kazuya Maemura[1] **Jesper Jansson**[1]
Hirotaka Ono[2] **Kunihiko Sadakane**[2] **Masafumi Yamashita**[2]

[1,2]Dept. of Computer Science and Communication Engineering, Kyushu University
(744 Motooka Nishi-ku Fukuoka 819-0395, Japan
e-mail:[1]{maemura, jj}@tcslab.csce.kyushu-u.ac.jp,
[2]{ono, sada, mak}@csce.kyushu-u.ac.jp)

Abstract. We study the evolutionary tree construction from rooted triplets from the viewpoint of approximation algorithms. An evolutionary tree is a rooted tree structure that represents the evolutionary interrelationship among various species in which species are represented by leaves. A triplet is a small evolutionary tree which consists of three leaves. Given a set S of species, and a set T of triplets whose leaves are from S, we construct a tree preserving (we say "satisfying") the evolutional divergence structures in T. It is shown that the problem of determining whether there exits a tree satisfying all of the triplets in T is polynomially solvable, while finding a tree that maximizes the number of satisfied triplets in T is NP-hard. For this maximization problem, Wu proposed simple bottom-up heuristics called Algorithm BPMF [5]. In this paper, we study the performance of BPMF-type algorithms from both theoretical and simulation-viewpoints.

1 Introduction

It is believed that all of the species on the Earth evolved from only one common ancestor, and they have evolutionary relationship among them. An evolutionary tree is a tree structure showing the evolutionary interrelationship among various species. Evolutionary trees are rooted and unordered trees. The labeled leaves in an evolutionary tree represent species, and the root represents the common ancestor of all species, and internal nodes represent ancestors of species. In this paper, we focus on the case that evolutionary trees are binary. Figure 1 is an example of evolutionary trees [6].

For a set of species, a standard way to construct a evolutionary tree for the set is as follows. First we construct evolutionary trees for subsets of the species based on knowledges of biology. Then we combine the trees into an evolutionary tree for all the species. Here we are faced with a difficulty that not all the trees for the subsets can be combined into an evolutionary tree due to errors. Therefore, in this paper we consider a mathematical problem that given a set of species and a set of trees for the species, construct an evolutionary tree for the species which is consistent with as many of the trees (a formal definition is given in Section 2). We call the problem *Maximum Rooted Triplets Consistency problem* (MRTC) or *the Maximum Inferred Local Consensus Tree problem* (MILCT).

Several studies are done for the MRTC. Bryant [2] proved that the problem is NP-hard. Gasieniec et al. [3] gave a factor-3 approximation algorithm. However, because the algorithm outputs only caterpillar trees, actual approximation ratio is not good. Wu [5] proposed heuristic algorithms which have better approximation ratios than [3] in practice, though the worst-case ratios are not analyzed.

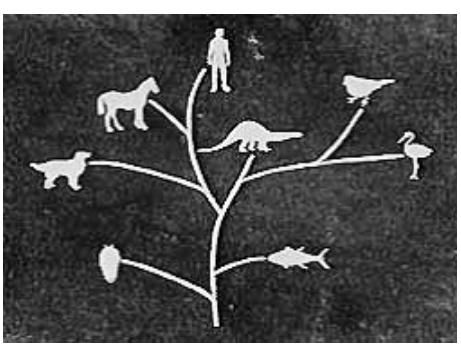

Figure 1: A example of evolutionary tree

The contributions of this paper are:

- We show that one of the six heuristic algorithms proposed in [5] has approximation ratio 3.
- We give some experimental results on approximation ratio of the algorithms. The results show that the algorithm that is analyzed in this paper has better approximation ratio for real inputs than that of [3], though they have the same worst-case ratio.
- We give a heuristic algorithm which improves the approximation ratio in practice.

The rest of this paper is organized as follows. In Section 2 we explain triplets and evolutionary tree construction problems. In Section 3 we describe heuristic algorithms which Wu proposed [5]. Section 4 and 5 are our contributions. In Section 4 we show the approximation ratio of Algorithm BPMF, in Section 5we propose improved algorithm of Algorithm BPMF and show experimental results. Section 6 gives conclusion of this paper.

2 Preliminaries

First we explain rooted triplets and evolutionary tree construction problems as preliminaries.

2.1 Rooted Triplet

A rooted triplet is an evolutionary tree which consists of three leaves. Let x, y and z be species, rooted triplet is described as Figure 2. This tree is described such as $\{x, y\} < \{x, z\}$ or $\{x, y\} < \{y, z\}$. In this description, $\{x, y\}$ denotes an internal node as the lowest common ancestor of x and y. $\{x, z\}$ or $\{y, z\}$ also denotes the lowest common ancestors of x and z or y and z, respectively Moreover

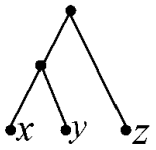

Figure 2: A example of triplet

$\{x, y\} < \{x, z\}$ means that $\{x, y\}$ is a proper descendant of $\{x, z\}$ or $\{y, z\}$. In other words, when we think x, y and z on evolutionary process, firstly z is divergent from the root and next x and y are divergent.

$\{x, y\} < \{x, z\}$ or $\{x, y\} < \{y, z\}$ are described as $(\{x, y\}, z)$ for short.

2.2 Tree Construction Problems

Now we explain evolutionary tree construction problems. Let S be a finite set of species, and T be a finite set of rooted triplets whose three different leaves are elements of S; n denotes cardinality of S, and m denotes cardinality of T. As inputs of evolutionary tree construction problems, we are given two sets, S and T.

We treat the evolutionary interrelationship of species on triplets in T as constraints for constructing trees. In our study, our problem is to construct a tree satisfying constraints in T made of S. If evolutionary interrelationship of three species of a triplet T_i in T correspond to the interrelationship of same species of a constructed tree, we say the tree "satisfies" triplet T_i.

Several studies are done for constructing evolutionary trees so far. It is shown that the problem of determining whether there exists a tree satisfying all of the triplets in T is polynomially solvable [1], while finding a tree that maximizes the number of satisfied triplets in T is NP-hard [2, 4]. This problem is called *Maximum Rooted Triplets Consistency problem* (MRTC) or *the Maximum Inferred Local Consensus Tree problem* (MILCT), and studied in the field of approximation algorithm.

3 Algorithm BPMF

In this section, we explain approximation algorithm in [5], named Algorithm Best Pair Merge First. As the first step in this section we show a simple process of Algorithm Best Pair Merge First, we call this Algorithm BPMF for short.

Let τ be a set of trees. In the initial condition of τ, each element of τ is only single node (it becomes a leaf of trees) labeled x, where x is an element of S. In other words, τ contains n nodes uniquely labeled n species. In Algorithm BPMF, we construct new trees made from trees in τ. Until τ contains only one tree, we merge two trees in τ and construct a new tree at each iteration.

The question is how we choose two trees to merge from τ. For this purpose author defines a function $e\_score(V(t_a), V(t_b))$.

Definition3.1: $V(t_a)$ denotes set of leaves which tree t_a has.

Let t_a, t_b be elements of τ, this function evaluates the score of merging trees t_a and t_b. At each iteration, we choose the two trees with maximum score from τ and merge them. The details of the algorithm is the following.

Algorithm Best Pair Merge First

(1) $\tau = \{t_x | 1 \leq x \leq n, t_x$ is a single node labeled $x\}$
(2) **while** $|\tau| > 1$ **do**
(2-1) Choose two trees t_a and t_b from τ which maximize $e\_score(V(t_a), V(t_b))$ among all pairs of two trees in τ.
(2-2) Merge t_a and t_b by adding a root of a new tree as a common ancestor, and replace t_a and t_b by the new tree.
endwhile
(3) Return the tree in τ

Next we show the function e_score defined in [5].

Definition3.2: The function e_score is defined as combinations of below three functions.
• $w(V(t_a), V(t_b)) = \{(\{x, y\}, z) \in T | x \in V(t_a), y \in V(t_b), z \in S \backslash (V(t_a) \cup V(t_b))\}$: this means the set of triplets satisfying the new tree when t_a and t_b are merged.
• $p(V(t_a), V(t_b)) = \{(\{x, z\}, y), (\{y, z\}, x) \in T | x \in V(t_a), y \in V(t_b), z \in S \backslash (V(t_a) \cup V(t_b))\}$:this means the set of triplets remaining the new tree when t_a and t_b are merged.
• $t(V(t_a), V(t_b)) = \{(\{x, y\}, z) \in T | x \in V(t_a),$

	Ratio-type		
if-penalty	0	1	2
0	w	$\frac{w}{w+p}$	$\frac{w}{t}$
1	$w-p$	$\frac{w-p}{w+p}$	$\frac{w-p}{t}$

Table 1: A table of *e_score* for combinations of parameters.

$y \in V(t_b), z \in S\}$

Author sets two parameter **if-penalty** and **ratio-type**. If-penalty is set 0 or 1: if if-penalty is 0, we use $|w(V(t_a), V(t_b))|$, otherwise we use $|w(V(t_a), V(t_b))| - |p(V(t_a), V(t_b))|$. Ratio-type is set 0, 1 or 2: we use respectively 1,$|w(V(t_x), V(t_y))| + |p(V(t_x), V(t_y))|$ or $|t(V(t_x), V(t_y))|$ for each value. This two parameters give six scoring functions. For different combinations of the two parameters, the function *e_score* is defined as Table 1. In this table, $|w(V(t_x), V(t_y))|$, $|p(V(t_x), V(t_y))|$ and $|t(V(t_x), V(t_y))|$ are described as w, p and t for short.

Algorithm BPMF merges two trees in τ and heuristically constructs a tree which satisfies as many of the triplets as possible in bottom-up manner. In [5], there are no mentions about time complexity and approximation ratio of Algorithm BPMF. Thus we need to study about this two topics.

Here we shortly mention the time complexity. At each iteration, the number of times we compute *e_score* is $\binom{|\tau|}{2}$. Then number of the iterations is $n-1$; this means the number of internal nodes of constructed trees. Therefore the total sum of the number of times in all iterations is

$$\sum_{l=2}^{n} \binom{l}{2} = \frac{1}{6}(n-1)n(n+1).$$

The time complexity of computing an *e_score* is at most $O(m)$. Therefore the time complexity of Algorithm BPMF is $O(mn^3)$.

4 Approximation Ratio of Algorithm BPMF

We pointed out the approximation ratio of Algorithm BPMF is not analyzed in [5]. In this section, we discuss and compute the approximation ratio of Algorithm BPMF. We derive approximation ratio of Algorithm BPMF using how to compute approximation ratio of Algorithm Heuristic1 [3]. Here we shortly mention approximation ratio in MRTC.

Definition4.1: Let a function $Sat(R)$ be the number of triplets tree R which satisfies an instance S and T.

Let R_{Opt} be the tree of optimal solution, and R_{App} be a tree constructed in approximation algorithm. Then the approximation ratio of MRTC is $\max\{\frac{Sat(R_{Opt})}{Sat(R_{App})}\}$.

We compute the approximation ratio of Algorithm BPMF referring [3]. However, we could compute approximation ration in the case that *e_score* is if-penalty=0 and ratio-type=1.

Before discussing the approximation ratio of Algorithm BPMF, we have some propositions about the function *e_score*. We use these propositions later to discuss the approximation ratio.

Lemma 4.2: We assume that we merge t_a and $t_b (t_a, t_b \in \tau)$ at an iteration. Then *e_score* after this iteration does not contain triplets in *e_score*$(V(t_a), V(t_b))$. In other words, it is unnecessary that we count triplets in *e_score*$(V(t_a), V(t_b))$ to compute *e_score* after this iteration.

Proof: We use the function *e_score* for evaluation when merging two different trees of τ: triplets in $w(V(t_a), V(t_b))$ or $p(V(t_a), V(t_b))$ are contained if a element of $V(t_a)$ and a element of $V(t_b)$ are in different trees of τ. Merging t_a and t_b means that $V(t_a)$ and $V(t_b)$ are united and become the set of leaves the new tree has. Therefore, above holds. □

For any iteration, let T' be the set of triplets which are contained in *e_score*$(V(t_a), V(t_b))$ until all of the trees in τ at this iteration are constructed. By the definition of w and p, any triplet in T' has at least two leaves from the same tree of τ. Therefore, by Lemma 4.2 the following corollary is derived.

Corollary 4.3: Any triplet in $T \backslash T'$ consists of three leaves which belong to different trees in τ.

Next, we have some definitions.

Definition4.4: For a triplet $(\{x, y\}, z)$, where all of the leaves are different, a pair of species $\{x, y\}$ is said to be a *lower-lower-pair* in the triplet, and $\{x, z\}$(or $\{y, z\}$) are said to be a *lower-upper-pair* in the triplet.

Definition4.5: We define two sets.

$$\begin{aligned} LL(k,l;T) &:= \{(\{k,l\},z) \in T\} \\ LU(k,l;T) &:= \{(\{x,y\},z) \in T | (x=k, z=l) \\ &\quad \vee (y=k, z=l) \vee (x=l, z=k) \\ &\quad \vee (y=l, z=k)\} \end{aligned}$$

Lemma 4.6: For any instance S and T, at each iteration of the algorithm the following holds. In initial condition, T' is empty set.

$$\frac{\sum_{k,l\in S} |LL(k,l;T\backslash T')|}{\sum_{k,l\in S} |LL(k,l;T\backslash T')| + \sum_{k,l\in S} |LU(k,l;T\backslash T')|} = \frac{1}{3}$$

Proof: For any triplet, there exists a pair of species which is a lower-lower-pair and two pairs of species which are lower-upper-pairs. Therefore, for all pairs of species in S the ratio between the total number of triplets in $LL(k, l)$ and the total number of triplets $LU(k, l)$ is always 1:2. □

In particular, Lemma 4.6 implies that given a non empty set S and T, at any iteration there always exists a pair of species for which $\frac{|LL(k,l;T\backslash T')|}{|LL(k,l;T\backslash T')|+|LU(k,l;T\backslash T')|}$ is equal to or more than $\frac{1}{3}$.

Lemma 4.7: For any instance S and T, at any iteration the following holds.

$$\begin{aligned} \sum_{k\in V(t_a), l\in V(t_b)} |LL(k,l;T\backslash T')| &= |w(V(t_a), V(t_b))| \\ \sum_{k\in V(t_a), l\in V(t_b)} |LU(k,l;T\backslash T')| &= |p(V(t_a), V(t_b))| \end{aligned}$$

Proof: First, we discuss above formula. We can describe $|w(V(t_a), V(t_b))|$ and $\sum_{k\in V(t_a), l\in V(t_b)} |LL(k,l;T\backslash T')|$ in detail as the following.

$$\begin{aligned} & |w(V(t_a), V(t_b))| \\ = \; & |\{(\{x,y\},z) \in T | x \in V(t_a), y \in V(t_b), \\ & z \in S\backslash(V(t_a) \cup V(t_b))\}| \\ & \sum_{k\in V(t_a), l\in V(t_b)} |LL(k,l;T\backslash T')| \\ = \; & \sum_{k\in V(t_a), l\in V(t_b)} |\{(\{k,l\},z) \in T\backslash T'\}| \\ = \; & |\{(\{x,y\},z) \in T\backslash T' | x \in V(t_a), y \in V(t_b)\}| \end{aligned}$$

Both $w(V(t_a), V(t_b))$ and $\bigcup_{k\in V(t_a), l\in V(t_b)} LL(k, l; T\backslash T')$ are sets of triplets in which the elements of the lower-lower-pairs are from $V(t_a)$ and $V(t_b)$.

Firstly we prove that $w(V(t_a), V(t_b)) \subseteq \bigcup_{k\in V(t_a), l\in V(t_b)} LL(k, l; T\backslash T')$. By Corollary-4.3, $T\backslash T'$ contains triplets which consist of three leaves in three different trees in τ, and triplets in $w(V(t_a), V(t_b))$ consist of three leaves from three trees in τ. As a result, all the triplets in $w(V(t_a), V(t_b))$ are elements of $\bigcup_{k\in V(t_a), l\in V(t_b)} LL(k, l; T\backslash T')$.

Now we discuss the opposite. By Corollary 4.3, a leaf z of $(\{x, y\}, z)$ is an element of $S\backslash(V(t_a) \cup V(t_b))$. As a result, all the triplets in $\bigcup_{k\in V(t_a), l\in V(t_b)} LL(k, l; T\backslash T')$ are elements of $w(V(t_a), V(t_b))$.

Thus, we can derive $\sum_{k\in V(t_a), l\in V(t_b)} |LL(k, l; T\backslash T')| = |w(V(t_a), V(t_b))|$.

We can also derive $\sum_{k\in V(t_a), l\in V(t_b)} |LU(k, l; T\backslash T')| = |p(V(t_a), V(t_b))|$, in the same way. □

Theorem 4.8: Algorithm BPMF(e_score is if-penalty=0 and ratio-type=1) constructs a tree which satisfies a subset of T whose total number of triplets satisfied is at least $\frac{m}{3}$.

Proof: By Corollary 4.3, we can treat a tree in τ as a labeled leaf or species. Therefore, for all the triplets in $T\backslash T'$, if both T_k and T_l ($T_k, T_l \in T\backslash T'$) have leaves from the same tree in τ, we can describe the leaves as a leaf equivalent to the tree: for example, x denotes all of the leaves t_x has. Therefore, we can describe $\sum_{k\in V(t_x), l\in V(t_y)} |LL(k, l; T\backslash T')|$ as $|LL(x, y; T\backslash T')|$, and $\sum_{k\in V(t_x), l\in V(t_y)} |LU(k, l; T\backslash T')|$ as $|LU(x, y; T\backslash T')|$.

By Lemma 4.6 ,Lemma 4.7 and how to choose t_a and t_b in step (2-1) of the algorithm, the ratio $\frac{|w(V(t_a),V(t_b))|}{|w(V(t_a),V(t_b))|+|p(V(t_a),V(t_b))|}$ is at least $\frac{1}{3}$. The new tree satisfies all of the triplets which a pair $\{x, y\}$ is a lower-lower-pair in $T\backslash T'$.

Let T'' be the set whose elements are triplets which consist of x, y and other species in $T\backslash T'$. Thus, every time the algorithm has performed in step (2-2), the new tree satisfies a subset of T'' whose total number is at least $\frac{1}{3} \cdot |T''|$.

Let R_{BPMF} be q tree constructed by Algorithm BPMF. $Sat(R_{BPMF})$ is the sum of the size of w at each iteration. Therefore, $Sat(R_{BPMF})$ is equal to or more than $\frac{m}{3}$. □

Algorithm BPMF constructs a tree which

satisfies a subset of T whose total number of triplets satisfied is equal to or more than $\frac{m}{3}$. Therefore, the approximation ratio of Algorithm BPMF is derived by the following proposition.

Proposition 4.9: The approximation ratio of Algorithm BPMF (e_score is if-penalty=0, ratio-type=1) is 3.

Proof: Let R_{BPMF} be a tree constructed by Algorithm BPMF. $Sat(R_{Opt})$ is at most m, and $Sat(R_{BPMF})$ is at least $\frac{m}{3}$. Thus, the following holds.

$$\frac{Sat(R_{Opt})}{Sat(R_{BPMF})} \leq \frac{m}{m/3} = 3$$

By above formula, the approximation ratio of Algorithm BPMF is 3. □

At last of this section, we discuss this approximation ratio in Proposition 4.10.

Proposition 4.10: There exists no tight example such that the approximation ratio of Algorithm BPMF (e_score is if-penalty=0 and ratio-type=1) is 3.

Proof: We assume that there is a tight example, so $Sat(R_{Opt}) = 3 \cdot Sat(R_{BPMF}) = m$ ($\frac{Sat(R_{Opt})}{3} = Sat(R_{BPMF}) \geq \frac{m}{3}$).

Let two leaves be x and y in the optimal tree. We discuss $w(x, y)$ and $p(x.y)$ is the following.

$$\begin{aligned} w(x,y) &= \{(\{x,y\},z) \in T | c \in S\} \\ p(x,y) &= \{(\{x,z\},y), (\{y,z\},x) \in T | c \in S\} \end{aligned}$$

From the assumption, $e\_score(x, y)$ must be $\frac{1}{3}$. In addition, the optimal tree satisfies all the triplets in $w(x, y)$ and $p(x, y)$. However, triplets in $w(x, y)$ conflict triplets in $p(x, y)$, and there exists no trees which satisfies all the triplets in $e\_score(x, y)$. This contradicts the assumption. □

5 Improved Algorithm

In this section, we propose an improved algorithm which we improve Algorithm BPMF. In addition, we show experimental results for Algorithm BPMF, the improved algorithm and Heuristic1, which implies a improved algorithm is equal to or better than original algorithm in practice.

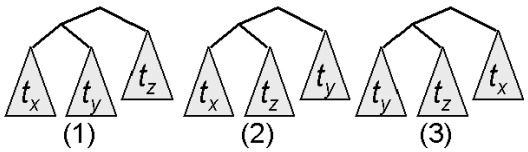

Figure 3: Examples of trees constructed by t_x, t_y and t_x.

5.1 Algorithm BPMR

In each iteration of Algorithm BPMF, we choose the two trees which maximize function the e_score value from τ, and merge them to construct a new tree: in this algorithm we heuristically construct a tree. Consequently, we assume that first t_x and t_y are merged and next the tree and t_z are merged, so this tree is not always the tree which satisfies the most triplets. More accurately, it is possible that a tree in Figure 3 (2) or (3) is better than a tree Figure 3(1) constructed in Algorithm BPMF. An improved algorithm which we propose is an algorithm which this point are considered.

Now we explain the improved algorithm. The main process of the algorithm is the same as Algorithm BPMF: a finite set τ and its initial condition, and function e_score to merge two trees in τ.

At each iteration in the algorithm, we assume that t_x and t_y are the two trees maximize e_score; t_x consists of two subtrees t_{x1} and t_{x2}, t_y also consists of two subtrees t_{y1} and t_{y2}. At this condition, we compute the value of the function Sat (Definition 4.1) for each tree of Figure 4. We reconstruct the tree such that maximum value of Sat in the five trees: in Algorithm BPMF, we can regard only the tree t_α is selected. This improved algorithm reconstructs trees, so we named this Algorithm

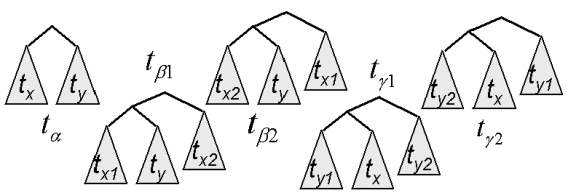

Figure 4: Trees: $t_\alpha, t_{\beta 1}, t_{\beta 2}, t_{\gamma 1}, t_{\gamma 2}$

Best Pair Merge with Reconstruction; Algorithm BPMR for short. This algorithm's more detailed process is below.

Algorithm Best Pair Merge with Reconstruction

(1) $\tau = \{t_x | 1 \leq x \leq n, t_x$ is a single node labeled $x\}$
(2) **while** $|\tau| > 1$ **do**
(2-1) Choose two tree t_a and t_b from τ which maximize $e\_score(V(t_a), V(t_b))$ among all pairs of two trees in τ.
(2-2) Compute $Sat(t_\alpha)$, $Sat(t_{\beta 1})$, $Sat(t_{\beta 2})$, $Sat(t_{\gamma 1})$ and $Sat(t_{\gamma 2})$.
(2-3) Construct the tree such that the Sat value is maximum.
(3) Return the new tree in τ.

We mention the time complexity of Algorithm BPMR. The total number of times that computations for reconstructing trees is $n-1$. At the worst case of computing Sat value, the time complexity is $O(mn)$ by trees' height: in the case of complete binary tree, it is $O(m\log n)$. Therefore, by Section3 the time complexity of Algorithm BPMR is $O(mn^3)$.

Proposition 5.1: Let R_{BPMF} or R_{BPMR} be a tree constructed by Algorithm BPMF or BPMR. The following holds.

$$Sat(R_{BPMF}) \leq Sat(R_{BPMR})$$

Proof: At each iteration, we assume that $e\_score(V(t_a), V(t_b))$ is max. We can consider the following two cases.
(i) There is no reconstruction.

In this case, tree t_α in Figure 4 is constructed. It is the same as the tree in Algorithm BPMF, so $Sat(R_{BPMF})$ is equal to $Sat(R_{BPMR})$.
(ii) There is reconstruction.

At each iteration, the two trees which are constructed in Algorithm BPMF or BPMR are determined only by the value of function e_score. In addition, the arguments of function e_score are sets of leaves which the two trees have. It is obvious that $V(t_\alpha) = V(t_{\beta 1}) = V(t_{\beta 2}) = V(t_{\gamma 1}) = V(t_{\gamma 2})$. Therefore, the arguments for e_score are the same whether reconstruct t_α or not. Thus, R_{BPMR} and R_{BPMF} are different only at the subtree.

The difference between $Sat(R_{BPMF})$ and $Sat(R_{BPMR})$ is the difference between the number of triplets in the original and reconstructed trees. We presume that t_α is reconstructed, so $Sat(R_{BPMR})$ is more than $Sat(R_{BPMF})$.

By (i) and (ii), we can prove that $Sat(R_{BPMF}) \leq Sat(R_{BPMR})$ holds. □

5.2 Experimental Comparison

We had experiments to compare the performance of Algorithms BPMF, BPMR and Heuristic1.

For $n = 15$, we experimented that m =50, 100, 200, 300. We randomly generated integer numbers from 0 to 14 as species, and made triplets from groups of three species generated randomly. We experimented ten times for each m.

For any instance S and T, R_{Opt} and R_{App} respectively denote trees of optimal and the constructed one by approximation algorithm. Table 2 and 3 summarize the experimental results. For example, BPMF01 denotes the algorithm with if-penalty=0 and ratio-type=1. The each value in Table 2 means the average value of $\frac{Sat(R_{Opt})}{Sat(R_{App})}$ for the same m of each approximation algorithms. The each value in Table3 means the worst value of $\frac{Sat(R_{Opt})}{Sat(R_{App})}$ for same m.

m	50	100	200	300
BPMF00	1.199	1.222	1.190	1.199
BPMF10	1.106	1.108	1.094	1.091
BPMF01	1.097	1.117	1.084	1.076
BPMF11	1.103	1.113	1.082	1.077
BPMF02	1.254	1.320	1.254	1.237
BPMF12	1.099	1.120	1.087	1.082
BPMR00	1.190	1.207	1.187	1.194
BPMR10	1.106	1.100	1.083	1.079
BPMR01	1.097	1.111	1.077	1.073
BPMR11	1.103	1.105	1.075	1.076
BPMR02	1.225	1.250	1.186	1.196
BPMR12	1.099	1.111	1.079	1.080
Heuristic1	1.192	1.169	1.137	1.118

Table 2: The comparison of experimental results(average)

m	50	100	200	300
BPMF00	1.357	1.327	1.306	1.293
BPMF10	1.226	1.173	1.158	1.145
BPMF01	1.226	1.192	1.119	1.114
BPMF11	1.226	1.192	1.119	1.131
BPMF02	1.500	1.452	1.429	1.345
BPMF12	1.188	1.192	1.130	1.167
BPMR00	1.310	1.300	1.306	1.293
BPMR10	1.226	1.173	1.137	1.145
BPMR01	1.226	1.192	1.119	1.114
BPMR11	1.226	1.192	1.119	1.131
BPMR02	1.500	1.370	1.286	1.288
BPMR12	1.188	1.192	1.130	1.167
Heuristic1	1.333	1.283	1.178	1.168

Table 3: The comparison of experimental results(worst)

We cannot decide which is the best function of Algorithm BPMF or BPMR, but *e_score*s of BPMF01 or 11 are roughly better function. In the experiments, Algorithm BPMF01 and 11 perform better than Algorithm Heuristic1 for most instances.

6 Conclusions

In this paper, we computed the approximation ratio of the approximation algorithm named Algorithm BPMF proposed in [5], and proposed an improved algorithm of Algorithm BPMF named Algorithm BPMR. In addition, we showed experimental results for the algorithms. The most important contribution of this paper is a study of the approximation ratio of Algorithm BPMF which has never been studied so far. However, there is room for this algorithm to discussion.

A further direction of this study is analysis of exact approximation ratio. Furthermore, we will compute approximation ratio of Algorithm BPMF for other *e_score*'s, and propose other approximation algorithms and analyze its approximation ratio.

Acknowledgment

This work is supported in part by the Grant-in-Aid of the Ministry of Education, Science, Sports and Culture of Japan.

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 of Computing*, Vol.10, No.3, pp.405-421, 1981.

[2] D.Bryant, "Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis," PhD thesis, University of Canterbury , Christchurch, New Zealand, 1997.

[3] 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.

[4] J.Jansson, "On the Complexity of Inferring Rooted Evolutionary Trees," *Proceedings of the Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO 2001), Electronic Notes in Discrete Mathematics*, Vol. 7, pp. 121-125, Elsevier B.V., 2001.

[5] B.Y.Wu, "Constructing the Maximum Consensus Tree from Rooted Triples," *Journal of Combinatorial Optimization*, Vol.8, No.1, pp.29-39, 2004.

[6] http://www.christiananswers.net/home.html