Efficient Construction of the Evolutionary Tree from Quartets

Che-Hao Wu and Damon Shing-Min Liu
Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 621
{wch91, damon}@cs.ccu.edu.tw

Abstract

Scientists often need to use the information of the species to infer the evolutionary relationship among them. The evolutionary relationships are generally represented by a labeled binary tree, called the evolutionary tree (or phylogenetic tree). The problem of constructing the evolutionary tree for a set of species is often known as the phylogeny problem. The difficulty of such problem is that the number of possible evolutionary trees is very large. There are many methods for solving the phylogeny problem including parsimony, maximum likelihood, divide-and-conquer approach and distance matrix methods [1, 5, 6, 7, 8]. As the number of species increases, so does the number of possible trees, making it difficult to exhaustively enumerate all possible relationships. The quantitative nature of species relationships therefore requires the development of more rigorous methods for tree construction.

In this paper, we proposed a new effective approach to construct the evolutionary tree. Our approach consists of three steps. First, we will partition the species and dispatch them onto different computers in order to quickly find the candidate quartets. Second, each computer will examine its local candidate quartets for compatibility. Third, a procedure will be applied to merge and evaluate compatible quartets found on each computer to form even larger quartets, with which we can construct evolutionary trees.

Keywords : Bioinformatics, evolutionary tree, divide-and-conquer approach, split, quartet.

1 Introduction

Bioinformatics is a newly emerging interdisciplinary research area, which may be defined as the interface between biological and computational sciences. Bioinformatics is the recording, annotation, storage, analysis, and searching of nucleic acid sequence, protein sequence and structural information. Due to the advances of biotechnological methods for gathering biological data, the amount of available data grows much faster than the growth in available computing power. Therefore, the collected data makes a strong demand for efficient algorithms and computational techniques so that bioinformatic activities are expanding rapidly in both academia and industry.

In biological research, it is often necessary to describe the relationship among species. An understanding of evolutionary relationships is at the heart of modern pharmaceutical research for drug discovery, helping researchers understand (and defend against) rapidly mutating viruses such as HIV [7, 8], is also the basis for the design of genetically enhanced organisms. Evolutionary history is typically represented by an evolutionary tree. Figure 1 shows two kinds of evolutionary tree: rooted and unrooted evolutionary tree [5, 6, 7, 8]. In an evolutionary tree, leaf nodes (and only leaf nodes) denote species. A tree of species with the root being the oldest common ancestor and the children of a node being the species show that evolved directly from that node. A path from the root to a species shows the evolutionary path of that species.

The problem of constructing the evolutionary tree for a set of species is known as the phylogeny problem. The difficulty of this problem is that the number of possible evolutionary trees is very large. For example, when you are given n species to construct an unrooted evolutionary tree, the number

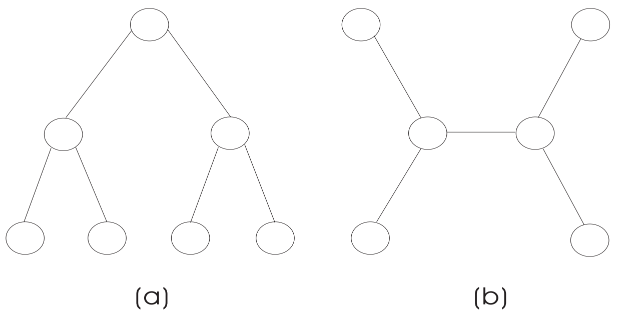

Figure 1: Rooted and unrooted evolutionary tree.

of possible unrooted trees is: N=(2n-5)!! [6], so when n gets large (>10), this number is huge (see Table 1). Consequently, exhaustive enumeration is not feasible.

The methods for solving the phylogeny problem include parsimony, maximum likelihood, and distance matrix methods [1, 5, 6, 7, 8]. Use of these traditional methods to construct trees with large collections of species is problematic for two reasons: very time consuming and optimization methods do not work well.

In addition to forementioned methods, we can also use divide-and-conquer approaches [8] to solve the phylogeny problem. The reconstruction of larger evolutionary trees from smaller subtrees is currently receiving considerable attention in the computational biology community. There is a clear computational advantage to analyze small subsets of species. There are also biological and statistical advantages of considering only small subsets of sequences at a time. The main difficulty is how to effectively build large trees from such smaller ones.

In this paper, we present an effective

Table 1: Number of unrooted trees constructed.

$Numberofspecies$	$Numberofunrootedtrees$
2	1
3	1
4	3
:	:
17	6,190,283,353,629,375
18	191,898,783,962,510,625
19	6,332,659,870,762,850,625
20	221,643,095,476,699,771,875

method for solving such problems.

2 Preliminaries

In this section, we describe some relevant notations and terminologies.

2.1 Splits

Let T be an unrooted tree and let e be an edge of T. If we remove e then we divide T into two components. Let A be the leaves in one component and B be the leaves in the other component. Then $A|B$ is a partition of L(T) into two blocks, called a split or bipartition of L(T). The split $A|B$ is said to be the split corresponding to the edge e. The set of those splits corresponding to edges in T is called the set of splits of T or just the splits in T and is denoted β (T) [2, 3]. We say that a split $A|B$ is in T if $A|B$ corresponds to an edge of T. If |A| = 1 or |B| = 1 then $A|B$ is trivial, otherwise it is non-trivial. Generally, a tree can be reconstructed in linear time from its set of splits.

2.2 Quartet

For every three leaves a, b, c there is only one unrooted tree with the leaf set {a, b, c} although there are three unrooted trees for any set of four leaves. The three binary trees with four leaves are called quartets [2, 3, 4]. A quartet is an unrooted binary tree on four species. A quartet induces a unique bipartition of the four species and can be denoted by that bipartition.

The quartet with two pendant pairs {a, b} and {c, d} is denoted ab|cd. We say that a quartet ab|cd fits a tree T if the path from a to b in T does not share any vertices with the path from c to d in T. The quartet set of a tree or just the set of quartets in a tree is the set of quartets that fits T, and is denoted q(T).

To illustrate, we give a simple example (Figure 2).

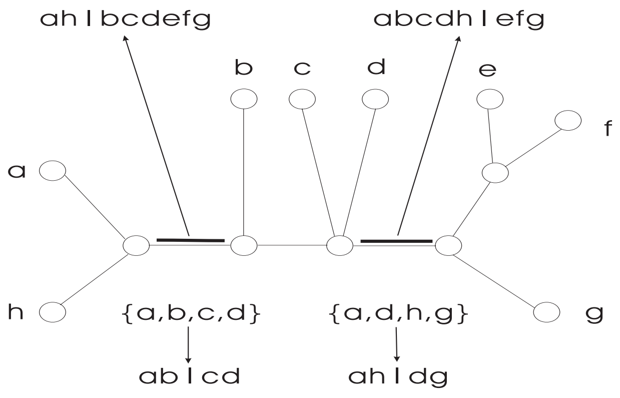

Figure 2: Splits and quartets of an unrooted evolutionary tree.

We have selected two internal edges to get two splits (ah|bcdefg and abcdh|efg) and give the induced quartets (ab|cd and ah|dg).

3 Tree Construction Algorithm

The problem is: suppose that we are given a set of quartets, how can we build trees? If we can find an unrooted tree T that corresponds to quartet sets Q, we call quartet sets Q are compatible or T extends Q. To date, there have been no polynomial-time algorithms for building a tree from quartets, thus the set of quartets would have to be small.

Here we introduce an algorithm to determine compatibility of any small- to medium-sized quartet sets. This algorithm is called ConstructTree algorithm [2]. The inputs are leaves (i.e., species) L=$\{a_1 , ... , a_n\}$ and some quartet sets Q.

We first label the leaves L=$\{a_1 , ... , a_n\}$ in order and construct one star tree with leaves L=$\{a_1, a_2, a_3\}$. The resulting tree is named T_3. Parameters to the procedure ConstructTree are k, T_k and the quartet set Q (shown as below).

```
Procedure ConstructTree (k, $T_k$ ,Q)
1. FOR all edges e in $T_k$ DO
2.     Construct $T_{k+1}$ from the tree $T_k$ by subdividing e with a new internal vertex
       and appending $a_{k+1}$ to this vertex.
3.     IF $T_{k+1}$ extends Q THEN
4.         IF k + 1 = n THEN output this tree
5.         ELSE ConstructTree(k + 1, $T_{k+1}$,Q)
6.     END(IF)
7. END(FOR)
```

For example, we are given leaves L={a, b, c, d, e}and quartet sets Q = {ab|cd, ab|ce}. First we construct one star tree with leaves a, b, c as shown in Figure 3(a). Then we insert a new leave d. We can get three possible results shown as in Figure 3(b)(c)(d). However, only tree in Figure 3(d) extends Q. Next we insert leave e and obtain five possible results as shown in Figure 3(e)(f)(g)(h)(i). Eventually there are three trees (Figure 3(g)(h)(i)) extend Q after inserting all leaves.

4 Approach

Obviously, splits provide more useful information compared to quartets. For an unrooted tree, if we are given n species, there are (2n-3) edges [6]. It also means that if there are n species we can find (2n-3) splits. However, in it only (n-3) splits are non-trivial.

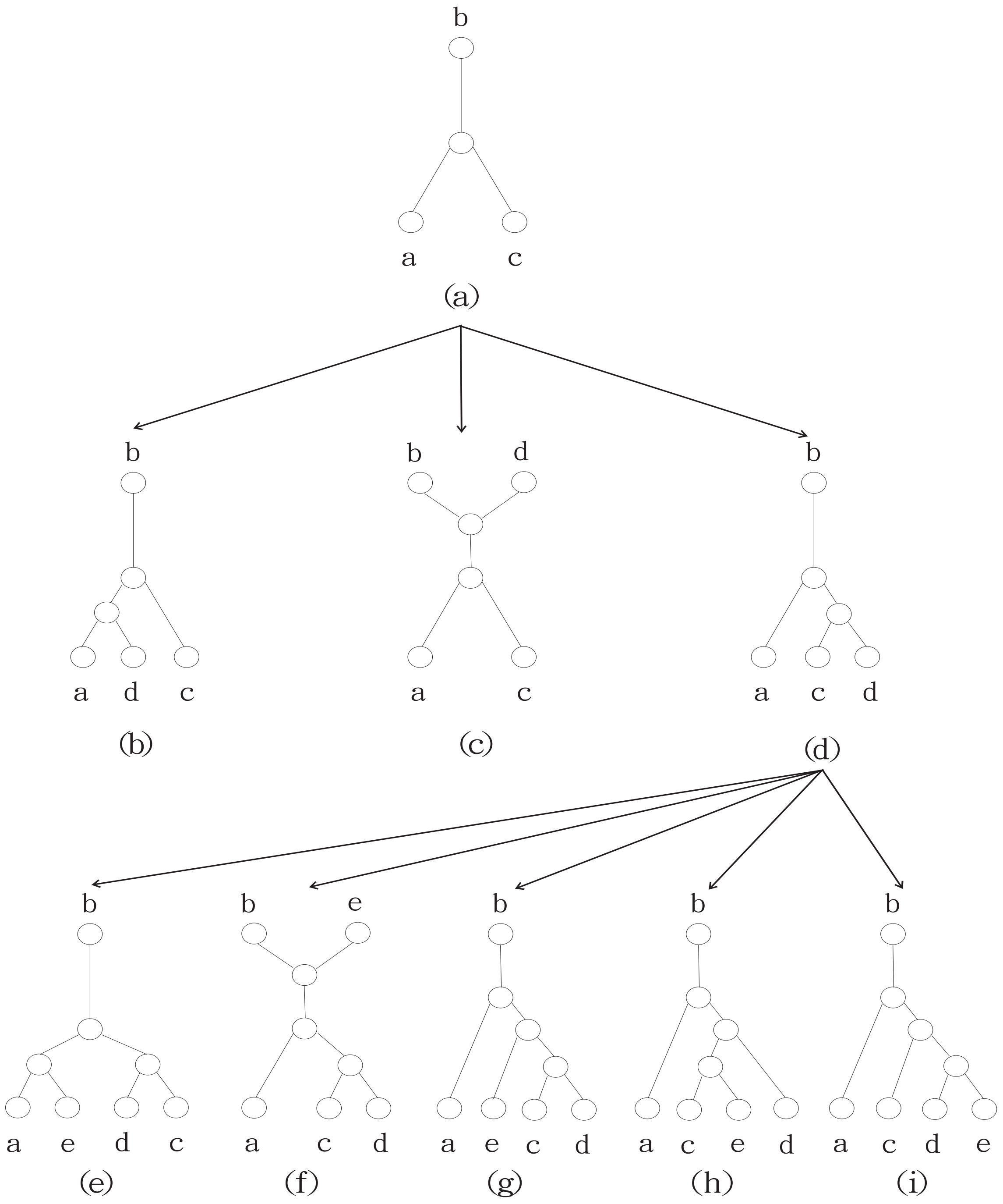

Figure 3: The example of ConstructTree algorithm.

Therefore, if we can find those non-trivial splits, we can use the information to construct the unrooted tree. Nevertheless, it is not easy and straightforward to find those non-trivial splits.

On the contrary, it is much easier to compute quartets than to compute splits. Given any four species, there are three possible quartets. If there are n species, we can compute $\binom{n}{4}$ combinations of any four species. Each combination has three possible quartets. So, there are totally $3^{\binom{n}{4}}$ combinations of candidate quartets. If n >= 5, for example, it is impossible to emulate all combinations of quartets in a reasonable time frame (see Table 2).

In fact, there exists more than one quartet combination to reconstruct evolutionary trees, however, it is impossible to emulate all of them. Our approach is to find the first quartets that are compatible, and use those quartets to reconstruct the tree. The proposed approach is described as follows.

In the first step, we will exhaustively enumerate and partition species into clusters. Each cluster has a multiple of four species. However, clusters may not be disjoint. We will then dispatch clusters onto different computers for processing. The dispatching strategy will take into account each machines computing capability. Machine that has higher computing power will process larger clusters, while those are less capable will process smaller clusters. If we have n species and m computers, we will dispatch f_1, f_2, ... , f_m clusters to each computer according to its computing power, where f_1, f_2, ... , f_m may not be equal and $|f_1|+|f_2|+...+|f_m|=\binom{n}{4}$. In order to determine the quartet we will add some rules.Each computer will calculate its own possible quartet sets, and all candidate quartet sets are computed in parallel.

However, not all quartet sets are compatible. Therefore, in the second step we will evaluate their compatibility by using ConstructTree algorithm. The iteration starts with a compatible quartet set, every time we add a new quartet and re-evaluate the compatibility of the newly formed quartet set to decide whether we should reject the last added quartet or keep adding another one to form a larger quartet set. This iteration may be parallelized to for best performance. If there exists a tree that extends all quartets, it means that all quartets must be compatible. In other words, if there exists no quartet set that is compatible, there is no solution for the tree. If this is the case, we should terminate at the second step. Otherwise, we may continue.

In the last step, we will try to merge compatible quartets found on each machine to form even larger quartet sets. The procedure is realized by exhaustively enumerating all possible unions of compatible quartets between any two machines. Note that the resulting evolutionary tree may not be unique, thus we have the choice to either stop on finding the first good answer, or keep on searching for all valid tree topologies. In either case, the technique of parallelism can be applied to achieve better efficiency. Furthermore, step 2 and step3 may also be integrated to exploit concurrent computing.

Table 2: Number of combinations of quartets.

$Number of species$	$Number combinations of all quartets$
4	3
5	243
6	14,348,907
7	50,031,545,098,999,707
:	:

5 Conclusion and Future Work

In this paper, we present an effective method to create and determine quartets which are compatible, and then use these quartets to construct evolutionary trees. Although we can find a correct evolutionary tree to describe the relationship among the species, it may not be the optimal solution. There is still room for improvement of our method. In the future, we will explore heuristics in order to find an optimal tree topology.

References

[1] H. O. Andrzej Lingas and A. Ostlin. Efficient merging and construction of evolutionary trees. In *Journal of Algorithms*, volume 41, pages 41–51, 2001.

[2] D. Bryant. Building trees, hunting for trees, and comparing trees. In *Ph.D.Thesis. University of Canterbury, NZ.*, 1997.

[3] D. Bryant and M. Steel. Constructing optimal trees from quartets. In *Journal of Algorithms*, volume 38, pages 237–259, 2001.

[4] K. S. John, T. Warnow, B. M. Moret, and L. Vawter. Performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001.

[5] J. A. Jones and K. A. Yelick. Parallelizing the phylogeny problem. 1995.

[6] R. C. T. Lee. Computational biology. In *Department of Computer Science and Information Engineering, National Chi-Nan University*, 2001.

[7] B. M. E. Moret, D. A. Bader, and T. Warnow. High performance algorithm engineering for computational phylogenetics. In *Lecture Notes in Computer Science*, volume 2074, page 1012, 2001.

[8] L. A. Salter. Algorithms for phylogenetic tree reconstruction. In *Proceeding of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, volume 2, pages 459–465, 2000.