Supertree construction by matrix representation with flip

by

Duhong Chen

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Iowa State University

Ames, Iowa

2003

DEDICATION

I would like to dedicate this thesis to my wife Jing and to my daughter Vivian without whose support I would not have been able to complete this work. I would also like to thank my friends and family for their loving guidance during the writing of this work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

This thesis is about a new method for evolutionary tree inference and phylogenetic supertree construction. A phylogenetic tree or simply phylogeny represents the evolutionary history of a set of species. It can be used to determine the genetic connections and relationships between species. In recent years the explosive growth in phylogenetic trees on different small sets of species, systematic biologists have become interested in assembling those trees in one supertree called the "tree of life" which represents evolutionary history among all living things(1).

1.1 Basic Concepts and Methods

In this section, I will introduce some basic terminology, theorems, and notation that are used in this thesis.

1.1.1 Phylogenetic Trees

A *tree* is a acyclic, connected graph which consists of a set of nodes(vertices) connected by a set of branches (edges). The *degree* of a vertex equals to the number of edges to which it is connected. Let $M = \{m_1, \ldots, m_s\}$ be a taxon set. A *phylogenetic tree* T over M is a tree with exactly s leaves, each of which is labelled with a distinct element of M. Such a tree is used to represent biological evolutionary relationships among the taxon set M. A phylogenetic tree can be rooted or unrooted. A *rooted tree* is a tree in which one of the nodes is stipulated to be the root, from which a unique path leading to every other node. An *unrooted tree*, as could be imagined, has no pre-determined root and therefore induces no hierarchy.

Suppose $T = (V, E)$ is a tree. A vertex $v \in V$ is internal if its degree is greater than one; otherwise, it is a *leaf.* An edge $e = (u, v) \in E$ is internal if both u and v are internal vertices;

otherwise we say e is an *external* edge. We write $\mathcal{L}(T)$ to denote the leaf set of tree T. A vertex a is a *descendent* of a vertex b if the path from a to the root passes through b. Then b is called the *ancestor* of a. For a vertex $v \in V$, the set containing all leaves that are descendants of v is called a *cluster.* The set M is always a cluster of T; every other cluster is said to be *proper.*

A rooted tree is *fully resolved* if its root has degree two and all other internal nodes have degree three. Any binary tree is fully resolved. A *triplet* is a tree with three leaves. There are two types of triplets: *resolved* and *unresolved.* In Figure 1.1, Tree 1.1(a) is an *unresolved triplet,* it contains no internal branches and is thus uninformative about cladistic relationships. Tree 1.1(b) is a *resolved triplet* which contains one internal branch and conveys cladistic information.

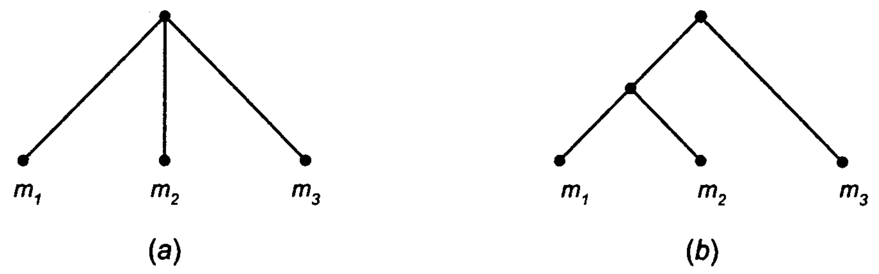


Figure 1.1 (a) an unresolved triplet and (b) a resolved triplet

1.1.2 Phylogenetic Inference

Under the natural assumption, species with similar characters(features) are genetically close. Thus, the basic idea of phylogenetic inference is to compare specific characters of the species. Phylogenetic analysis was originally developed by biological systematists who wanted to reconstruct evolutionary genealogies of species based on morphological similarities. The German entomologist Willi Hennig was the first author to propose an explicit method of phylogenetic analysis, and the publication of his work in English quickly led to the widespread use of his approach. Classic phylogenetic inference dealt mainly with physical, or morphological features - size, color, number of legs, etc. Modern phylogeny uses information directly from genetic material - mainly DNA or protein sequences. The characters used are usually DNA or protein *sites*,where a site is a single position in the sequence after alignment with several such sequences. Since phylogenetic trees play an important role in the study of molecular evolution, determining the evolutionary history of species is one of the fundamental tasks of computational biology.

The construction of optimal evolutionary trees is a very challenging problem, many different instances of this problem have been defined and studied and most of them are $\mathcal{NP}$-complete. Many of the popular phylogenetic inference methods attempt to solve $\mathcal{NP}$-hard optimization problems, such as maximum parsimony(2), maximum character compatibility(3), and maximum likelihood which has not been proved but behaves as it is(4). Polynomial time methods, such as neighbor-joining, also exist. Even though these problems have been studied extensively, phylogenetic tree construction is still an open problem (5).

1.1.3 Character Compatibility and Phylogenetic Inference

Given a set of species M, a character over M is a partition of M. A partial character of M is a partition of a subset of M. The elements of a character are called *states.* A *character state* tree is a tree whose node is labelled by a distinct element of that character states. A character is *directed* if its character state tree is rooted; otherwise, it's *undirected.* A character with exact two states is a bipartition $C = \{C_\circ, C_\bullet\}$ of a subset M_C of M, called a *binary character.* The sets $C_\circ$ and $C_\bullet$ are, respectively, the *0-state* and the *1-state* of C. If $M_C = M$, C is said to be *complete*, otherwise, it is *incomplete.* The set $M - M_C$ is called the *unknown state or ?-state* and is denoted by $C_?$. A complete character C' is a *completion* of C if $C_\circ \subseteq C'_\circ$ and $C_\bullet \subseteq C'_\bullet$. In the rest part of this thesis, without special specification, we will simply call rooted phylogenetic trees as trees, and binary directed characters as characters.

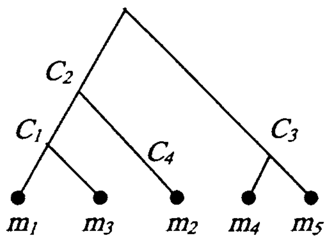
In what follows (Figure 1.2 (a) and (b)), $\mathcal{C} = (C_i)_{i=1}^r$ denotes a tuple of characters over M. A *completion* of $\mathcal{C}$ is a tuple $\mathcal{C}' = (C'_i)_{i=1}^r$ of complete characters such that, for $i \in \{1, \ldots, r\}$, C'_i is a completion of C_i. The characters $\mathcal{C}$ over M can be represented by a $\{0,1,?\}$ $s \times r$ matrix $\mathcal{A} = [a_{ij}]$ where a_{ij} is 0, 1, or ?, depending on whether m_i is in $C_{j\circ}$, $C_{j\bullet}$, or $C_{j?}$. A *completion* of $\mathcal{A}$ is a binary matrix $\mathcal{B} = [b_{ij}]$ where $a_{ij} \in \{b_{ij}, ?\}$ for all i, j. Thus, each column of $\mathcal{B}$ represents the completion of a character of $\mathcal{C}$ and $\mathcal{B}$ represents a completion of $\mathcal{C}$.

	C_1	C_2	C_3	C_4
m_1	1	1	0	0
m_2	0	1	0	1
m_3	1	?	0	0
m_4	0	0	1	?
m_5	?	0	1	0

(a)

	C_1	C_2	C_3	C_4
m_1	1	1	0	0
m_2	0	1	0	1
m_3	1	*1*	0	0
m_4	0	0	1	*0*
m_5	*0*	0	1	0

(b)



(c)

Figure 1.2 (a) a tuple of characters $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$ over $M = \{m_1, m_2, m_3, m_4, m_5\}$ is denoted by a 5×4 matrix $\mathcal{A}$. (b) A binary matrix $\mathcal{B}$ represents the completion of $\mathcal{A}$ and the set of characters $\mathcal{C}$. (c) A tree $T_{\mathcal{C}}$ is consistent with the set of characters $\mathcal{C}$ where each $C_{j\bullet}$ induces a cluster in $T_{\mathcal{C}}$.

Definition 1 *Two characters C_1 and C_2 are said to be* ***pairwise compatible****, written as $PC(C_1, C_2)$, if there exists a tree T such that both $C_{1\bullet}$ and $C_{2\bullet}$ induce a cluster.*

Two binary characters are incompatible if and only if all three possible combinations of states (0-1; 1-0; 1-1) are present in the characters.

Definition 2 *A tuple of characters $\mathcal{C}$ is said to be* ***compatible*** *if and only if there is a phylogenetic tree $T_{\mathcal{C}}$ in which every character in $\mathcal{C}$ its $C_{j\bullet}$ induces a cluster. The tree $T_{\mathcal{C}}$ is said to* ***consistent with*** *the character set $\mathcal{C}$.*

As shown in figure 1.2 (c) the tree $T_{\mathcal{C}}$ is consistent with the set of characters $\mathcal{C}$ with each $C_{j\bullet}$ induces a cluster in $T_{\mathcal{C}}$. The following basic result is proved by Estabrook et al (4).

Theorem 1 *A tuple $\mathcal{C}$ of complete characters is compatible, if and only if for every pair $C, C' \in \mathcal{C}$, $C_\bullet \cap C'_\bullet \in \{\emptyset,\ C_\bullet,\ C'_\bullet\}$.*

Note that this theorem *does not* hold for incomplete characters. Here is an example: the matrix is taken from (6),whose rows represent species and whose columns correspond to characters. Every pair of characters is compatible, but the whole set is not since for the character C_1, if the ? is set to 0, then it is incompatible with C_2; and incompatible with C_3 if the ? set to 1.

	C_1	C_2	C_3
m_1	1	0	0
m_2	1	1	0
m_3	?	1	1
m_4	0	?	1

Figure 1.3 A set of pairwise compatible incomplete characters, but the whole set is not compatible

Definition 3 (Perfect Phylogeny Problem)

Given: *A character tuple $\mathcal{C}$ over the taxon set M.*

Questions: *Does there exist a phylogenetic tree $T_{\mathcal{C}}$ consistent to $\mathcal{C}$? If so, build one.*

The tree $T_{\mathcal{C}}$ is called *perfect phylogeny* for the set of characters $\mathcal{C}$ which has the property that for each state of a character, the set of all nodes that have that state induces a subtree. The general perfect phylogeny problem is NP-hard(7). When considering the number of possible states per character as a parameter, the problem is fixed parameter tractable(8). For binary characters, having only two states for each character, perfect phylogeny problem is linear time solvable (6). When one of character in $\mathcal{C}$ has ?-state, this problem is call *Incomplete Perfect Phylogeny Problem.*

When no perfect phylogenies possible for the full set of characters,(the most common case in real phylogenetic analysis), we need some ways of choosing among them a best estimate. For example we may consider a subset of characters for which a perfect phylogeny exist.

Definition 4 (Maximum Character Compatibility Problem)

Given: *A character tuple $\mathcal{C}$ over the taxon set M.*

Questions: *Find the largest subset $\mathcal{C}'$ of $\mathcal{C}$ such that a phylogenetic tree $T'_{\mathcal{C}}$ can be inferred from $\mathcal{C}'$.*

Maximum character compatibility problem is $\mathcal{NP}$-hard(9). The proof is based on a polynomial reduction from the clique problem.

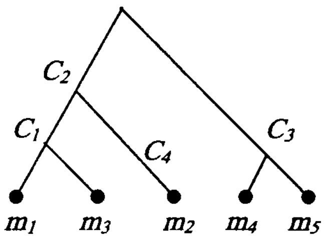
Several factors can cause incompatibility among characters. For example, errors in data, loss of function during evolution, convergent evolution, etc. Convergent evolution is a trait developed independently by two evolutionary paths (e.g. wings in birds and bats). If only error in data causes lack of perfect phylogeny, we are interested in building a perfect phylogeny with minimum changes in the input data. For binary characters, assuming the states are 0 and 1, then the changes or error corrections, simply called *flips*, are either $0 \rightarrow 1$ or $1 \rightarrow 0$. Figure 1.4 (a) shows a tuple of characters $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$ over $M = \{m_1, m_2, m_3, m_4, m_5\}$. Character C_1 is incompatible with C_2. So are C_3 and C_4. After two flips, the set of characters (Figure 1.4(b)) is compatible and thus a perfect phylogeny (Figure 1.4(c)) can be constructed and called such phylogeny as *minimum flip tree.* This problem is going to be described formally in chapter 2.

	C_1	C_2	C_3	C_4
m_1	1	1	0	0
m_2	0	1	0	1
m_3	1	$\underline{0}$	0	0
m_4	?	0	1	$\underline{1}$
m_5	0	0	1	0

(a)

	C_1	C_2	C_3	C_4
m_1	1	1	0	0
m_2	0	1	0	1
m_3	1	*1*	0	0
m_4	?	0	1	*0*
m_5	0	0	1	0

(b)



(c)

Figure 1.4 Flip to achieve perfect phylogeny

1.1.4 Parsimony

Parsimony is one of the most popular methods for phylogenetic inference. In order to define this method, we begin with the following definition.

The *Hamming distance* between two vectors x and y of same length is $|\{i : x_i \neq y_i\}|$ and is denoted by $H(x, y)$. The *parsimony length* of a tree $T = (V, E)$ in which each node $v \in V$ is labelled by a sequence is the sum of the Hamming distance of the sequences labelling endpoints of the edges in the tree, i.e., $\sum_{(a,b)\in E} H(a,b)$.

Definition 5 (Maximum Parsimony Problem)

Given: *A set of sequences S with each has r sites.*

Questions: *Find a tree leaf labelled by S and assigned sequences for the internal nodes of minimum parsimony length.*

Maximum parsimony or simply parsimony chooses that phylogeny on which the characters can evolve with the fewest evolutionary events. Unfortunately, this is an $\mathcal{NP}$-hard problem, even when the sequences are binary(10).

1.2 Review of Supertree Reconstruction Methods

The trees that will be used to construct the supertree are the *source trees.* For the purposes of this thesis, all trees will be considered rooted. Species found on only one source tree are *unique.* Those found on two or more are *shared.* Any tree containing all the species found among the source trees is a *supertree.* If all the input trees have same leaves, the supertree is usually called *consensus tree.*

A rooted phylogenetic tree T' is said to be obtained from T by *contraction* if T' can be obtained from T by contracting a sequence of internal edges. A rooted phylogenetic tree T *displays* a rooted phylogenetic tree t if t can be obtained from an induced subtree of T by contraction (or, equivalently, t is an induced subtree of a contraction of T).

Definition 6 (Tree Compatibility Problem)

Given: *A set of trees $\mathcal{T}$.*

Questions: *Does there exist a phylogenetic tree T that displays every tree $t \in \mathcal{T}$?.*

Unfortunately, the general version of determining whether a set of source trees are compatible is NP-complete(7). However, if the source trees are rooted then it is polynomial time solvable(7)(11).

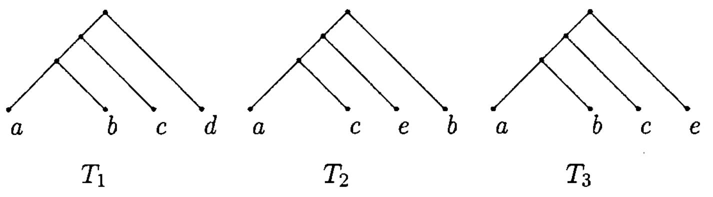
1.2.1 Phylogenetic Trees and their Binary Character Coding

Let T be a phylogenetic tree over a subset of M and let X be a non-trivial cluster in T. We ignore the trivial clusters that are useless in supertree reconstruction. The *character representation of* X is the incomplete binary character C over M, where $C_\bullet$ contains all taxa in the cluster, $C_\circ = \mathcal{L}(T) - C_\bullet$ and $C_? = M - \mathcal{L}(T)$. Let $X_1, \ldots, X_k$ be the non-trivial clusters in T. The *character representation of* T is the tuple $\mathcal{C}_T = (C_1, \ldots, C_k)$, where, for $i \in \{1, \ldots, k\}$, C_i is the character representation of X_i. The *matrix representation of* T is the matrix representation of $\mathcal{C}_T$.

Let $\mathcal{T} = \{T_i\}_{i=1}^{l}$ be a set of phylogenetic trees over subsets of M. The *character representation of* $\mathcal{T}$ is the character tuple

$$\mathcal{C} = (C_1^1, \ldots, C_{k_1}^1, \ldots, C_1^l, \ldots, C_{k_l}^l),$$

where, for $i \in \{1, \ldots, l\}$, $(C_1^i, \ldots, C_{k_i}^i)$ is the character representation of T_i. The *matrix representation of* $\mathcal{T}$ is the matrix $A_\mathcal{T}$ obtained by concatenating the matrix representations of $T_1, \ldots, T_r$. A set of trees $\mathcal{T} = \{T_1, T_2, T_3\}$ and its matrix representation is shown in Figure 1.5.



	C_1^1	C_2^1	C_1^2	C_2^2	C_1^3	C_2^3
a	1	1	1	1	1	1
b	1	1	0	0	1	1
c	0	1	1	1	0	1
d	0	0	?	?	?	?
e	?	?	0	1	0	0

Figure 1.5 A collection of trees and its matrix representation

1.2.2 Matrix Representation with Parsimony(MRP) and Matrix Representation with Flip(MRF)

When the source trees are compatible, they can be included in a single full consistent supertree. In fact, source trees conflict is very common during tree assembly. Thus the relationships between certain shared species must be taken into account during tree building since they cause incompatible. One of the common supertree methods is to re-code trees by incomplete binary characters and then apply phylogenetic inference. The most widely used supertree method, MRP,is based on this idea. Unfortunately, this method is $\mathcal{NP}$-hard due to finding maximum parsimonious tree. MRF does the same way as MRP except that the objective of MRF is to find the minimum flip trees instead of the most parsimonious trees.

1.2.3 MinCut(MC) and Modified MinCut(MMC) Algorithm

The mincut supertree algorithm(12) can construct a rooted supertree for a set of rooted input trees in polynomial time. The MC algorithm is derived from the Algorithm 1 described by Aho et al(13) which determines the the rooted tree compatibility problem in polynomial time. That is, whether there is a supertree with which each input tree is compatible can be answered in polynomial time. The Aho's algorithm construct a graph $\mathcal{H}=(V,E)$, where the vertices V are species, a edge $(a,b) \in E$ if and only if species a and b are in a proper cluster in at least one of the input trees. Let $\mathcal{T}$ be a family of rooted phylogenies on subsets of M. Given a set $X \subseteq M$, $\mathcal{T}|X$ denotes $\{T|X : T \in \mathcal{T}\}$.

Semple and Steel modified step 8 in Algorithm 1 so that it always returns a tree. The idea is when only one component occurs in step 8 MC algorithm would find a min cut set in $\mathcal{H}$ to break it up into several connected components. Thus, the computation can be continued until return a tree. MC algorithm also assigns weight to the graph $\mathcal{H}$ and might collapse some edges in $\mathcal{H}$ in order to preserve strict cluster nestings which appears in the source trees. The cut set is helpful to solve conflict and preserve some desirable nesting property in the source trees. That is, any nesting cluster in all the source trees will appear in the mincut supertree. However, the original algorithm has two drawbacks: (i) it is not efficient in calculating all the minimum

Algorithm 1 RootedTreeCompatibility($\mathcal{T}, M$)

```
▷ $\mathcal{T}$ is a set of input(source) trees over species $M = \{m_1, \ldots, m_s\}$
if $size[M]$=1 then
    return a single node labelled by $m_1$
end if
if $size[M]$=2 then
    return a tree with two taxa labelled by $m_1$ and $m_2$.
end if
Otherwise,construct graph $\mathcal{H}$ as described
if $\mathcal{H}$ has only one connected component then
    return nil
end if
for each component $M_i$ of $\mathcal{H}$ do
    $T_i \leftarrow RootedTreeCompatibility(\mathcal{T}|M_i, M_i)$
    if $T_i$ = nil then
        return nil
    end if
end for
construct a new tree $T$ by connecting the roots of trees $T_i$ to a new root.
return $T$
```

cut set of the graph; (ii) it is too sensitive to the size of the input tree and fails to include the compatible clusters in the supertree. Page's modified mincut approach (1) improved MC algorithm by avoiding these two drawbacks.

1.3 Outline

Chapter 2, **Complexity of the Minimum Flip Problem**, commences with the Minimum Flips Problem(MFP) and proceeds with a study of the complexity of the problem.

Chapter 3, **Algorithms for Solving the Minimum Flip Problem**, deals with the issue of solving minimum flips problem. Since MFP is NP-hard, it's impossible to find an efficient algorithm unless P=NP. A branch-and-bound algorithm is given in this chapter to solve small problem (the number of taxa is less than 16), and a uphill searching heuristic is discussed for large input trees.

Chapter 4, **Simulation Studies on Supertree Construction Methods**, studies the performance of different methods in supertree construction using simulation data. Those algorithms include: Matrix Representation with Parsimony, Matrix Representation with Flip, Semple and Steel's MinCutSupertree, and Page's Modified mincut suptertree algorithms.

The last chapter is the **Conclusion and Future Study**.

CHAPTER 2. COMPLEXITY OF THE MINIMUM FLIP PROBLEM

2.1 Introduction

We have introduced the goal and some basic concepts of phylogenetic inference in the previous chapter. In this chapter we will introduce our new tree inference method called minimum flip method which is base on smallest changes on input data to achieve perfect phylogenies. Its application is used in supertree construction and the results of simulation study is reported in chapter 4.

2.2 Flip and the Minimum Flip Problem

In this section, we will discuss some relevant terminology and notations. M is a finite set of taxa, $\wp(N)$ denotes the power set of a set N. We define $\mathcal{C}_M$ to be the set of all characters over the set M, $\mathcal{T}$ is a collection of phylogenetic trees over the set M .

Definition 7 *For a character $C \in \mathcal{C}$ and a set $F \in \wp(M)$, define the* flip-operation $\triangle$ *as follows,*

$$C' = C \triangle F :\Longleftrightarrow C'_\circ = C_\circ \Delta F \wedge C'_\bullet = C_\bullet \Delta F$$

.

The set F is called a flip for the character C, or simply a flip, we refer only when the character C is obvious. A flip F is called a d-flip *if $F \subseteq C_\bullet$, and an* i-flip *if $F \subseteq C_\bullet = \emptyset$.*

The flip operator is generalized to character tuples $\mathcal{C} = (C_i)_{i=1}^r$ as follows. Let $\mathcal{F} = (F_i)_{i=1}^r$ be a flip tuple. Then $\mathcal{C}' = \mathcal{C} \triangle \mathcal{F}$ is the character tuple $(C'_i)_{i=1}^r$ where, for $i \in \{1, \dots, r\}$, $C'_i = C_i \triangle F_i$. The size *of $\mathcal{F}$ is defined as $s(\mathcal{F}) = \sum_{i=1}^r |F_i|$.*

Flipping can be viewed as an operation on the matrix representation of character tuples. Suppose that A is the matrix representation of $\mathcal{C}$ and that $\mathcal{F}$ is a flip tuple. A *flip* for entry a_{ij} such that $a_{ij} \neq ?$ is the operation of replacing a_{ij} by its complement. The matrix representation of $\mathcal{C} \triangle \mathcal{F}$ is the matrix obtained by, for all i, j, flipping entry a_{ij} if $m_i \in F_j$.

The *minimum flip problem* is, given a character tuple $\mathcal{C}$, find a minimum-size flip tuple $\mathcal{F}$ such that $\mathcal{C}' = \mathcal{C} \triangle \mathcal{F}$ is compatible. Figure 2.1 shows a matrix representing a collection of incompatible characters, a flip tuple $\mathcal{F}$, the matrix of compatible characters obtained by flipping according to $\mathcal{F}$. The number of flip $s(\mathcal{F})$ in this example is 2. Note that the number of changes of flipping the character C to the character C' is the Hamming distance between C and C'. Thus, in the following example, $s(\mathcal{F}) = H(C_1, C_1') + H(C_3, C_3') = 2$.

	C_1	C_2	C_3	C_4
m_1	1	1	1	1
m_2	1	0	0	0
m_3	0	1	0	0
m_4	0	0	1	0
m_5	1	1	0	1

$$\mathcal{F} = (\{m_3\}, \emptyset, \{m_4\}, \emptyset)$$

	C_1'	C_2'	C_3'	C_4'
m_1	1	1	1	1
m_2	1	0	0	0
m_3	*1*	1	0	0
m_4	0	0	*0*	0
m_5	1	1	0	1

Figure 2.1 Example of a flip tuple $\mathcal{F}$, and $s(\mathcal{F}) = 2$.

Definition 8 (Flip Problem (FP) - the decision version of minimum flip problem)

Given: *A character tuple* $\mathcal{C} \in \mathcal{C}_M^r$ *where* $r \in \mathbb{N}$ *and a number* $k \in \mathbb{N}$.

Questions: *Does there exist a flip tuple* $\mathcal{F} \in \wp(M)^r$ *where* $s(\mathcal{F}) \leq k$, *such that* $\mathcal{C} \triangle \mathcal{F}$ *is compatible ?*

2.3 Complexity of the Minimum Flip Problem

In this section we show that FP is $\mathcal{NP}$-complete if the input is constrained to complete characters. From this follows directly the $\mathcal{NP}$-completeness for (partial) characters. All characters in this section are complete characters in $\mathcal{C}_M$. The proofs shown in this subsetcion use only set theoretical definitions of characters.

Lemma 2 *Let $\mathcal{C} = (C_1, C_2) \in \mathcal{C}_M^2$, such that $|C_{1_\bullet}| = |C_{2_\bullet}| = 3$ and $|C_{1_\bullet} \cap C_{2_\bullet}| \leq 1$. If $\mathcal{F} = (F_1, F_2) \in \wp(M)^2$ is a flip-tuple, such that $|F_1| = 0$, $|F_2| = 1$, and $\mathcal{C} \triangle \mathcal{F} = (C'_1, C'_2)$ is compatible, then $C'_{1_\bullet} \cap C'_{2_\bullet} = \emptyset$.*

Proof. Let $F_2 = \{e\}$. Since C'_1 and C'_2 are compatible, it follows from Theorem 1 that either $C'_{1_\bullet} \supseteq C'_{2_\bullet}$, or $C'_{1_\bullet} \subseteq C'_{2_\bullet}$, or $C'_{1_\bullet} \cap C'_{2_\bullet} = \emptyset$. We have $C'_1 = C_1 \triangle F_1 = C_1$, since $|F_1| = 0$. It follows that either $C_{1_\bullet} \supseteq C'_{2_\bullet}$, or $C_{1_\bullet} \subseteq C'_{2_\bullet}$, or $C_{1_\bullet} \cap C'_{2_\bullet} = \emptyset$. We will prove the lemma by showing that the first two cases are impossible.

$|C'_{2_\bullet}| \in \{2, 4\}$, since $|C'_{2_\bullet}| = |C_{2_\bullet} \triangle \{e\}|$ and $|C_{2_\bullet}| = 3$.

- *Case $C'_{2_\bullet} \subseteq C_{1_\bullet}$:* It follows that $|C'_{2_\bullet}| \leq |C_{1_\bullet}| = 3$. Thus, $|C'_{2_\bullet}| = 2$ and further, $\{e\}$ is a d-flip. Since $\{e\}$ is a d-flip we have $C'_{2_\bullet} = C_{2_\bullet} - \{e\}$ where $e \in C_{2_\bullet}$. It follows that $C_{2_\bullet} - \{e\} \subseteq C_{1_\bullet}$. Hence, $|C_{1_\bullet} \cap C_{2_\bullet}| = |C_{2_\bullet} - \{e\}| = 2$, which is in contradiction to $|C_{1_\bullet} \cap C_{2_\bullet}| \leq 1$.

- *Case $C_{1_\bullet} \subset C'_{2_\bullet}$:* It follows that $3 = |C_{1_\bullet}| < |C'_{2_\bullet}|$. Thus, $|C'_{2_\bullet}| = 4$ and further $\{e\}$ is an i-flip. Since $\{e\}$ is an i-flip we have $C'_{2_\bullet} = C_{2_\bullet} \cup \{e\}$ where $e \notin C_{2_\bullet}$. It follows that $C_{1_\bullet} \subseteq C_{2_\bullet} \cup \{e\}$. So $|C_{1_\bullet} \cap C_{2_\bullet}| \geq |C_{1_\bullet}| - 1 = 2$, which is in contradiction to $|C_{1_\bullet} \cap C_{2_\bullet}| \leq 1$.

□

Lemma 3 *Let $\mathcal{C} = (C_1, \dots, C_r) \in \mathcal{C}_M^r$ for some $r \in \mathbb{N}$, such that $|C_{i_\bullet}| = 3$, and $|C_{i_\bullet} \cap C_{j_\bullet}| \leq 1$ for any $i, j \in \{1, \dots, r\}$ where $i \neq j$. If there exists a flip tuple $\mathcal{F} = (F_1, \dots, F_r) \in \wp(M)^r$, such that $|F_i| = 1$ for any $i \in \{1, \dots, r\}$, and $(C'_1, \dots, C'_r) = \mathcal{C} \triangle \mathcal{F}$ is compatible, then $2r \leq |\bigcup_{i \in \{1, \dots, r\}} C'_{i_\bullet}|$.*

Proof. A flip in $\mathcal{F}$ is either a d-flip or an i-flip, since $|F_i| = 1$ for any $i \in \{1, \dots, r\}$. W.l.o.g. we assume that $\mathcal{F}_d = (F_1, \dots F_l)$ consists only of d-flips and $\mathcal{F}_i = (F_{l+1}, \dots, F_r)$ only of i-flips, for some $l \in \{0, \dots, r\}$. Thus

$$|C'_{i_\bullet}| = 2 \text{ for } i \in \{1, \dots, l\} \text{ and } |C'_{i_\bullet}| = 4 \text{ for } i \in \{l+1, \dots, r\} \qquad (*)$$

Claim 1: $C'_{i_\bullet} \cap C'_{j_\bullet} = \emptyset$ for either $i, j \in \{1, \ldots, l\}$ or $i, j \in \{l+1, \ldots, r\}$, where $i \neq j$.

Proof of claim: From theorem 1 it follows either $C'_{i_\bullet} \subseteq C'_{j_\bullet}$, or $C'_{i_\bullet} \supseteq C'_{j_\bullet}$, or $C'_{i_\bullet} \cap C'_{j_\bullet} = \emptyset$, since C'_i and C'_j are compatible. We will prove the claim by showing that the first two cases are impossible. It is sufficient to show this by proving that $C'_{i_\bullet} \subseteq C'_{j_\bullet}$ is impossible, since either $i, j \in \{1, \ldots, l\}$ or $i, j \in \{l, \ldots, r\}$.

Suppose $C'_{i_\bullet} \subseteq C'_{j_\bullet}$. We have $|C'_{i_\bullet}| = |C'_{j_\bullet}| = 2$ in the case $i, j \in \{1, \ldots, l\}$ and $|C'_{i_\bullet}| = |C'_{j_\bullet}| = 4$ otherwise. It follows that $|C'_{i_\bullet}| = |C'_{j_\bullet}|$. From this and our assumption that $C'_{i_\bullet} \subseteq C'_{j_\bullet}$ it follows that $C'_{i_\bullet} = C'_{j_\bullet}$. Now, if F_i and F_j are d-flips we have $|C_{i_\bullet} \cap C_{j_\bullet}| \geq (|C_{i_\bullet} - F_i) \cap (C_{j_\bullet} - F_j)| = |(C_{i_\bullet} \triangle F_i) \cap (C_{j_\bullet} \triangle F_j)| = |C'_{i_\bullet} \cap C'_{j_\bullet}| = |C'_{i_\bullet}| = 2$. Otherwise F_i and F_j are i-flips and we have $|C_{i_\bullet} \cap C_{j_\bullet}| + 2 \geq |(C_{i_\bullet} \cup F_i) \cap (C_{j_\bullet} \cup F_j)| = |(C_{i_\bullet} \triangle F_i) \cap (C_{j_\bullet} \triangle F_j)| = |C'_{i_\bullet} \cap C'_{j_\bullet}| = |C'_{i_\bullet}| = 4$. Hence in both cases we have $|C_{i_\bullet} \cap C_{j_\bullet}| \geq 2$, which is in contradiction the pre-condition $|C_{i_\bullet} \cap C_{j_\bullet}| \leq 1$.

Claim 2: For any $i \in \{l+1, \ldots, r\}$ there exists at most one $j \in \{1, \ldots, l\}$, such that $C'_{i_\bullet} \cap C'_{j_\bullet} \neq \emptyset$.

Proof of claim: Suppose there exist $v, w \in \{1, \ldots, l\}$ where $v \neq w$, such that $C'_{i_\bullet} \cap C'_{v_\bullet} \neq \emptyset$ and $C'_{i_\bullet} \cap C'_{w_\bullet} \neq \emptyset$.

First we show $C'_{v_\bullet} \subset C'_{i_\bullet}$. From Theorem 1 follows that $C'_{i_\bullet} \cap C'_{v_\bullet} \in \{C'_{i_\bullet}, C'_{v_\bullet}, \emptyset\}$, since the characters C'_i, C'_v, C'_w are compatible. By our assumption $C'_{i_\bullet} \cap C'_{v_\bullet} \neq \emptyset$ this reduces to $C'_{i_\bullet} \cap C'_{v_\bullet} \in \{C'_{i_\bullet}, C'_{v_\bullet}\}$. From this, $|C'_{i_\bullet}| = 4$, and $|C'_{v_\bullet}| = 2$ it follows that $C'_{v_\bullet} \subset C'_{i_\bullet}$. By a similar reasoning $C'_{w_\bullet} \subset C'_{i_\bullet}$ follows, and we conclude $C'_{v_\bullet} \cup C'_{w_\bullet} \subseteq C'_{i_\bullet}$.

Second, we show $C'_{v_\bullet} \cup C'_{w_\bullet} = C'_{i_\bullet}$. $C'_{v_\bullet} \cap C'_{w_\bullet} = \emptyset$ follows directly from Claim 1, and hence $|C'_{v_\bullet} \cup C'_{w_\bullet}| = |C'_{v_\bullet}| + |C'_{w_\bullet}|$. As shown before in $(*)$, it is $|C'_{v_\bullet}| = |C'_{w_\bullet}| = 2$ and $|C'_{i_\bullet}| = 4$. So $|C'_{i_\bullet}| = |C'_{v_\bullet}| + |C'_{w_\bullet}| = |C'_{v_\bullet} \cup C'_{w_\bullet}|$. Now, from $|C'_{i_\bullet}| = |C'_{v_\bullet} \cup C'_{w_\bullet}|$ and the prior result, $C'_{v_\bullet} \cup C'_{w_\bullet} \subseteq C'_{i_\bullet}$, it follows that $C'_{v_\bullet} \cup C'_{w_\bullet} = C'_{i_\bullet}$.

We are prepared to give the contradiction. Let $F_v = \{e_v\}$, $F_w = \{e_w\}$ and $F_i = \{e_i\}$. F_v, F_w are d-flips and F_i is an i-flip. Thus $C'_{v_\bullet} = C_{v_\bullet} - \{e_v\}$, $C'_{w_\bullet} = C_{w_\bullet} - \{e_v\}$ where $e_v \in C_{v_\bullet}$, $e_w \in C_{w_\bullet}$, and $C'_{i_\bullet} = C_{i_\bullet} \cup \{e_w\}$ where $e_i \notin C_{i_\bullet}$. Replacing $C'_{w_\bullet}$, $C'_{w_\bullet}$ and $C'_{i_\bullet}$ in $C'_{v_\bullet} \cup C'_{w_\bullet} = C'_{i_\bullet}$ leads to $(C_{v_\bullet} - \{e_v\}) \cup (C_{w_\bullet} - \{e_w\}) = C_{i_\bullet} \cup \{e_i\}$ where $e_v \in C_{v_\bullet}$, $e_w \in C_{w_\bullet}$, $C_i \notin C_{i_\bullet}$ and

$C_{v_\bullet} - \{e_v\} \cap C_{w_\bullet} - \{e_v\} = \emptyset$. So, either $e_i \in C_{v_\bullet} - \{e_v\}$ or $e_i \in C_{w_\bullet} - \{e_w\}$. W.l.o.g. we assume $e_i \in C_{v_\bullet} - \{e_v\}$. Thus $C_{w_\bullet} - \{e_w\} \subseteq C_{i_\bullet}$. From this follows $|C_{w_\bullet} \cap C_{i_\bullet}| \geq |(C_{w_\bullet} - \{e_w\}) \cap C_{i_\bullet}| = 2$, since $|C_{w_\bullet} - \{e + w\}| = 2$, which is in contradiction to the pre-condition $|C_{w_\bullet} \cap C_{i_\bullet}| \leq 1$.

From Claim 1 it follows that $|\bigcup_{i \in \{1,\ldots,l\}} C'_{i_\bullet}| = 2l$ and $|\bigcup_{i \in \{l+1,\ldots,r\}} C'_{i_\bullet}| = 4l$. Claim 2 states that for any $i \in \{l+1,\ldots,r\}$ there exists at most one $j \in \{1,\ldots,l\}$, such that $C'_{i_\bullet} \cap C'_{j_\bullet} \neq \emptyset$. Hence, any $C_{i_\bullet}$ for $i \in \{l+1,\ldots,r\}$ counts for at least 2 new elements. It follows $|\bigcup_{i \in \{1,\ldots,r\}} C'_{i_\bullet}| \geq 2r$. □

In the following we use a polynomial time reduction from CX3C to show that FP is $\mathcal{NP}$-complete. The problem CX3C is an input constrained version of exact 3-cover (X3C) (14) and is known to be $\mathcal{NP}$-complete (15).

Definition 9 (Constrained Exact 3-Cover (CX3C)**)**

Given *A set X with $|X| = 3q$ for some $q \in \mathbb{N}$, and a collection $\mathcal{S}$ of 3-element subsets of X, such that for any set $\{S, S'\} \subseteq \mathcal{S}$, $|S \cap S'| \leq 1$.*

Question *Does there exist a set $\mathcal{S}' \subseteq \mathcal{S}$ that covers X? ($\mathcal{S}'$* covers *X, iff $X = \bigcup_{S \in \mathcal{S}'} S$ and for any $\{S, S'\} \subseteq \mathcal{S}'$ holds $S \cap S' = \emptyset$.)*

Theorem 4 FP *is $\mathcal{NP}$-complete.*

Proof. Clearly, FP $\in$ NP. The reduction CX3C $\leq^m_p$ FP, will be shown by a modification of a proof given in (15).

Construction: Let a possible instance for CX3C be a set X, where $|X| = 3bq$ for some $q \in \mathbb{N}$, and $\mathcal{S} = \{S_1, \ldots, S_r\}$ be a collection of 3-element subsets of X. From this instance a possible instance for FP is constructed as follows: For every $i \in \{1,\ldots,r\}$ a character C_i is constructed, such that $C_{i_\bullet} = S_i$ and $C_{i_\circ} = X - S_i$. From the resulting characters the character r-tuple $\mathcal{C} = (C_1, \ldots, C_r)$ is constructed and the number $k = 2(r - q)$ is calculated.

Obviously, $\mathcal{C}$ and k can be calculated in polynomial time. Thus, the $\mathcal{NP}$-completeness of FP derives from the following statement.

Claim: $\mathcal{S}$ contains a 3-cover for X, iff there exists a flip-tuple $\mathcal{F}$ where $s(\mathcal{F}) \leq k$ and $\mathcal{C} \triangle \mathcal{F}$ is compatible.

Proof of claim:

"$\Longrightarrow$": Let $\mathcal{S}' = \{S_1, \ldots, S_q\}$ be a subset of $\mathcal{S}$ that is a 3-cover for X. Let $\mathcal{F} = (F_1, \ldots, F_q, F_{q+1}, \ldots F_r)$ be a flip tuple where $F_i = \emptyset$ for any $i \in \{1, \ldots, q\}$ and $F_i \subset C_{i_\bullet}$, such that $|F_i| = 2$ for any $i \in \{q+1, \ldots, r\}$. Thus, $s(\mathcal{F}) = k$. Now we show in two steps that $\mathcal{C} \triangle \mathcal{F}$ is compatible.
First we show that $C_i - F_i$ is compatible with any character in $\mathcal{C}_M$ for any $i \in \{q+1, \ldots, r\}$. We have $C_i \triangle F_i = C_i - F_i$ where $F_i \subset C_i$, since F_i is a d-flip for C_i. From this follows that $|C_i \triangle F_i| = |C_i - F_i| = 1$, since $|F_i| = 2$ and $|C_i| = 3$. From Theorem1 it follows that a character $C \in \mathcal{C}_M$, where $|C_\bullet| = 1$, is compatible with any character in $\mathcal{C}_M$. Thus, the character $C_i \triangle F_i$ is compatible with any character in $\mathcal{C}_M$.
Second we show that $C'_i = C_i \triangle F_i$ and $C'_j = C_j \triangle F_j$ are compatible for different $i, j \in \{1, \ldots, q\}$. It is $C'_i = C_i$ and $C'_j = C_j$, since $F_i = F_j = \emptyset$. Next we have $C_{i_\bullet} \cap C_{j_\bullet} = \emptyset$, since by our construction $C_{i_\bullet} = S_i$, $C_{j_\bullet} = S_j$, and S_i, S_j are elements of the 3-cover for X. Thus, $C'_{i_\bullet} \cap C'_{j_\bullet} = \emptyset$. From this it follows by Theorem 1 that C'_i and C'_j are compatible.
From the first and second part follows directly that $\mathcal{C} \triangle \mathcal{F}$ is compatible.

"$\Longleftarrow$": Suppose that there exists a flip-tuple $\mathcal{F}$ such that $s(\mathcal{F}) \leq k$ and $\mathcal{C} \triangle \mathcal{F} = (C'_1, \ldots, C'_r)$ is compatible. W.l.o.g. let $\mathcal{F} = (F_0, \ldots, F_i, F_{i+1}, \ldots, F_j, F_{j+1}, \ldots F_r)$ such that $|F_k| = 0$ for $k \in \{1, \ldots, i\}$, $|F_k| = 1$ for $k \in \{i+1, \ldots, j\}$, and $|F_k| > 1$ for $k \in \{j+1, \ldots, r\}$. Let $f_0 = i$, $f_1 = j - i$, and $f_{2\leq} = r - j$. We show $3/2(f_0 - q) \geq f_1 \geq 2(f_0 - q)$. From this it follows $q = f_0$ and hence f_0 is a 3-cover for X.

- It holds that $k = 2(n-q) \geq f_1 + 2f_{2\leq}$ and replacing n by $f_0 + f_1 + f_{2\leq}$ yields $f_1 \geq 2(f_0 - q)$.
- From the construction it follows that $3q \geq |\bigcup_{l \in \{1,\ldots j\}} C'_l|$. By Lemma 2 we have $|\bigcup_{l \in \{1,\ldots j\}} C'_l| = 3f_0 + |\bigcup_{l \in \{i+1,\ldots j\}} C'_l|$. Lemma 3 states $|\bigcup_{l \in \{i+1,\ldots j\}} C'_l| \geq 2f_1$. Thus $3q \geq 3f_0 + 2f_1$ which is equivalent to $3/2(f_0 - q) \geq f_1$.

□

CHAPTER 3. ALGORITHMS FOR SOLVING THE MINIMUM FLIP PROBLEM

3.1 Introduction

This chapter describes methods used for solving the minimum flip problem. We have proved that the FP problem is $\mathcal{NP}$-complete so there is probably no efficient algorithm to achieve a perfect phylogeny based on minimum change of the input characters. The search space of possible trees increases exponentially with the number of species. In general, the number of phylogenetic trees for a data set of s taxa is given by $\prod_{n=1}^{s-1}(2n-3)$ for $n \geq 2$, since for an arbitrary tree with n taxa, there are $2n-1$ possible places for the $(n+1)^{th}$ to be added. An exact solution to the minimum flip problem can be obtained using exhaustive search or branch-and-bound algorithm when the number of taxa is less than or equal to 15. Given a character tuple $\mathcal{C} = (C_1, \ldots, C_r)$ our algorithm exhaustively evaluates every tree T over M by the minimum number of flips to make $\mathcal{C}$ compatible with T. A tree is in our solution if it has the minimum number of flips over all of the possible trees. For large data sets, heuristics must be employed. Exact and heuristic methods will be discussed in section 2 and section 3 respectively.

Before solving the minimum flip problem, we first define a basic estimation problem. Remember that a rooted binary phylogenetic tree T over M can be represented as a tuple of characters called *character coding of* T and denoted by $\mathcal{C}_T = (C_1^T, \ldots, C_{2s-1}^T)$, where $s = |M|$, C_i^T is a s-dimensional complete binary character which corresponds to a cluster in the tree T, with $C_{i\bullet}$ containing all taxa in the cluster, $C_{i\circ} = M - C_{i\bullet}$. The *matrix representation of* T is the matrix representation of $\mathcal{C}_T$. Figure 3.1 is the matrix representation of the tree T in Figure 1.2 (c).

	C_1^T	C_2^T	C_3^T	C_4^T	C_5^T	C_6^T	C_7^T	C_8^T	C_9^T
m_1	1	1	0	0	0	0	1	1	0
m_2	1	0	1	0	0	0	0	1	0
m_3	1	0	0	1	0	0	1	1	0
m_4	1	0	0	0	1	0	0	0	1
m_5	1	0	0	0	0	1	0	0	1

Figure 3.1 A binary matrix representing a tree T

Definition 10 Flips Estimation Problem (FEP)

Given *A set of characters $\mathcal{C} = (C_1, \ldots, C_r)$ over M, where $r \in \mathbb{N}$ and a phylogenetic tree T over taxa M.*

Questions *What is the minimum number of flips in order to make $\mathcal{C}$ consistent with the T?*

Algorithm 2 Flips-Estimate ($\mathcal{C}$, T)

```
1: Codes $T$ into a set of characters $\mathcal{C}_T$
2: $flips \leftarrow 0$
3: for all $x \in \mathcal{C}$ do
4:   $distance \leftarrow \infty$
5:   for all $y \in \mathcal{C}_T$ do
6:     if $distance > H(x,y)$ then
7:       $distance \leftarrow H(x,y)$
8:     end if
9:   end for
10:  $flips \leftarrow flips + distance$
11: end for
12: return $flips$
```

Algorithm 2 first codes the tree T as a set of characters $\mathcal{C}_T$ and then maps each character $C_i \in \mathcal{C}$ to a character $C'_j \in \mathcal{C}_T$ such that $H(C_i, C'_j)$ is minimal among all $H(C_i, C')$ forall $C' \in \mathcal{C}_T$. Let $\mathcal{F}_{ij}$ be the flip tuple such that $C'_j = C_i \triangle \mathcal{F}_{ij}$, and $s(\mathcal{C}, \mathcal{C}_T)$ be the number of flips to make input characters $\mathcal{C}$ compatible with the phylogenetic tree T. Thus, $s(\mathcal{C}, \mathcal{C}_T) = \sum_{i=1}^{r} \min\{|\mathcal{F}_{ij}|, 1 \le j \le 2s-1\}$. Clearly, this can be done in $O(r \times s^2)$ because there are r binary characters in $\mathcal{C}$, and each of them maps to one of the $2s-1$ characters in $\mathcal{C}_T$ with minimum Hamming distance. Note that the calculation of Hamming distance $H(x, y)$

takes $O(s)$.

Since the phylogenetic tree T is binary, any character $C' \notin \mathcal{C}_T$ must be incompatible with T. Algorithm 2 maps each character in $\mathcal{C}$ to a character in $\mathcal{C}_T$ with minimum number of flips. Let $\mathcal{C}_{\mathcal{F}} = \mathcal{C} \triangle \mathcal{F}$. Then the algorithm guarantees that each character in $\mathcal{C}_{\mathcal{F}}$ (after flip) is compatible with the tree T and the number of flips is minimal. i.e. $\mathcal{C}_{\mathcal{F}} \subseteq \mathcal{C}_T$ and $s(\mathcal{C}, \mathcal{C}_T)$ is minimal. Therefore, Algorithm 2 solves the FEP.

3.2 Exact Methods

3.2.1 Exhaustive Enumeration

For a small input data set (generally, one where the number of taxa is less than 15), it is possible to find the optimal flipping tree by enumerating all of the possible trees and applying Algorithm 2 to evaluate the minimum number of flips for each of these trees. A simple method, "transverse add" (Figure 3.2), is used to enumerate all the trees with s taxa. Initially, a phylogenetic tree with two taxa is built. All possible three taxon trees are then constructed by adding the the third taxon at all possible places in the initial tree. The remaining taxa are added in this manner until all taxa have been added, which results in the enumeration of all of the possible trees for s taxa. For each complete tree, a tree with s taxa, the minimum number of flips is evaluated by applying Algorithm 2. The trees with smallest score are kept.

3.2.2 Branch-and-bound Method

We first specify an upper bound, U, on the number flips to make the input character set $\mathcal{C}$ compatible. We then consider the enumeration of the set of all complete trees with s taxa. Specifically, we begin with a tree with two taxa and then consider the three taxon tree constructed from the initial tree by adding the third taxon. From the remaining $s-3$ taxa, we can calculate the lower bound using Algorithm 3 described below. If the value of the minimum flip of three taxon tree plus its lower bound exceeds the upper limit U then we do not need to consider any of the trees that are obtained by adding taxa to this three taxon tree. Otherwise,

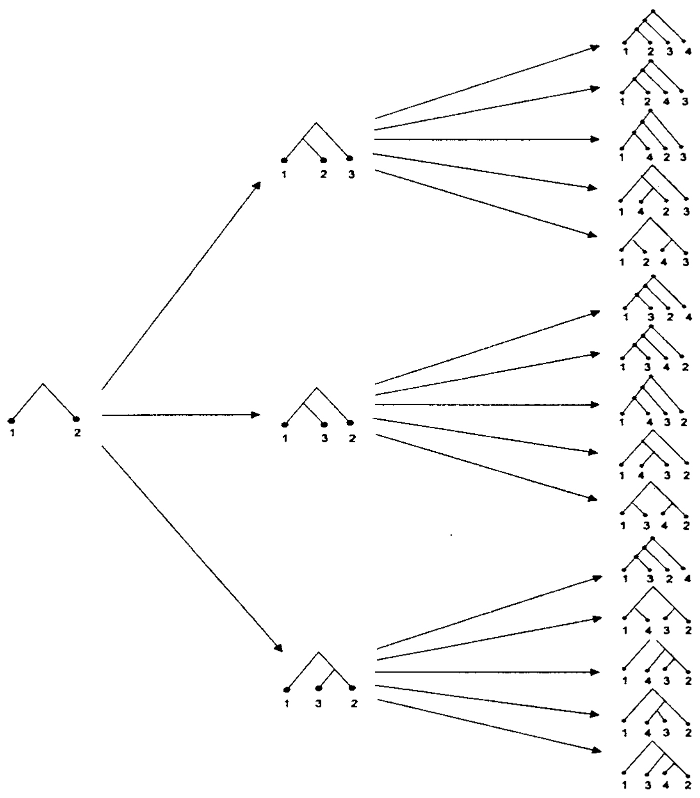


Figure 3.2 Enumeration of all possible trees for 4 species, An initial tree is formed from the first two taxa. The third taxon is added transversely to each possible location of the initial tree to obtain all the three taxon trees. The four taxon tree is formed from adding the fourth taxon transversely to each location of every possible three taxon trees.

we evaluate each of the four taxon trees obtained by adding the fourth taxon to this tree. The remaining taxa are added in the manner. If at any time a complete tree is encountered with a value U^* which is less than U, U is reset to U^* and the search is continued. Algorithm 4 is a branch-and-band algorithm to solve the minimum flip problem.

The branch-and-bound algorithm explores each possible topology until the upper limit is reached. Thus, it ensures that all trees are tested and guarantees that a phylogenetic tree which optimizes the criteria is found.

3.2.2.1 Lower Bound for Minimum Flip

A *maximum weight matching* in an undirected edge weight graph $G = (V, E, w)$ is a collection of node-disjoint edges with maximum total weight. A lower bound for the flip problem is the sum of a maximum weight matching for the following edge weighted graph $G = (V, E, w)$: $i \in V$, if C_i is a character in $\mathcal{C}$; $\{i, j\} \in E$, if C_i and C_j are incompatible, and $w(\{i, j\})$ is the minimum number of flips to make C_i and C_j compatible. Note that a tuple $\mathcal{C}$ of complete characters is compatible if and only if every pair $C, C' \in \mathcal{C}$ is compatible. A maximum weight matching for the graph G is a subset of flips making the characters pairwise compatible. Thus, it is a lower bound. We used Algorithm 3 to calculate the lower bound. In the following example (Figure 3.3), the lower bound for the set of $\mathcal{C} = (C_1, \ldots C_5)$ characters is 3.

	C_1	C_2	C_3	C_4	C_5
m_1	1	0	1	1	1
m_2	0	0	1	0	1
m_3	0	1	1	0	1
m_4	0	1	0	1	0
m_5	1	?	0	1	0
m_6	1	0	0	1	1

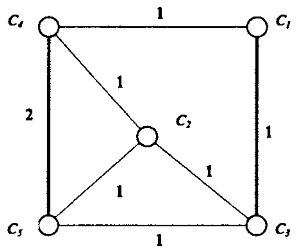


Figure 3.3 An example of lower bound for minimum flip problem, the thick edges is a maximum weighted matching whose total weight is 3. Thus, the lower bound for the input characters $\mathcal{C} = (C_1, \ldots C_5)$ is 3.

Algorithm 3 GetLowerBound ($\mathcal{C}$)

```
1: construct the edge weighted graph $G = (V, E, w)$ from $\mathcal{C}$
2: $M \leftarrow$ MAX_WEIGHT_MATCHING($G$)   ▷ returns maximum weighted matching as a list of edges
3: $L \leftarrow 0$
4: for all $e \in M$ do
5:     $L \leftarrow L + w(e)$
6: end for
7: return $L$
```

3.2.2.2 Upper Bound for Minimum Flip

An initial upper bound can be computed by a quick heuristic search or obtained from a random tree T. In the second case, a bound is given by the minimum number of flips to make $\mathcal{C}$ compatible with T. During the search, if a complete tree over M with a smaller minimum flip number is found, we replace the upper bound with the smaller value. The better the upper limit, the faster the search. Clever programming, including an appropriate choice of the initial upper bound can greatly improve the running time of the algorithm.

3.2.2.3 Branch-and-bound algorithm

Let A be a subset of M, and $\mathcal{C}|A$ denote the restriction of set of characters $\mathcal{C}$ to A. For each $C \in \mathcal{C}$, $C|A$ is a character C' over A such that $C'_\bullet = C_\bullet \bigcap A$, $C'_\circ = C_\circ \bigcap A$, and $C'_? = C_? \bigcap A$. We wrote $\mathcal{C}|A$ to denote the multiset $\{C' : C' = C|A \textit{ for } C \in \mathcal{C}\}$. Let T be a tree over taxa set A, and x, y, and z are nodes where x is a node in tree T, y is a new leaf node to be added to the tree, and z is a new internal node. Now, we introduce two functions *AddTaxon* and *RemoveTaxon* that are used in the Algorithm 5.

The function $AddTaxon(T, x, y, z)$ inserts the nodes z and y to the tree T such that y and x become z's left and right children respectively.

The function $RemoveTaxon(T, y, z)$ removes the nodes y and its parent z from the tree T.

Algorithm 4 BandBMinFlip($\mathcal{C}$)

```
▷ $\mathcal{C} = \{C_1, \ldots C_r\}$ is a tuple of characters over $M = \{m_1, \ldots, m_s\}$
1: generate a random tree $T$
2: $U \leftarrow Flips\_Estimate(\mathcal{C}, T)$   ▷ obtain upper bound
3: create the initial tree $T$ with the taxa $m_1$ and $m_2$
4: $A \leftarrow \{m_1, m_2\}$   ▷ $A$ is the set of taxa that have been added to the tree $T$
5: for $i \leftarrow 3$ to $s$ do
6:     $A \leftarrow A \bigcup \{m_i\}$
7:     $R \leftarrow M - A$   ▷ $R$ is the set of taxa what not appear in tree $T$
8:     $L[i] \leftarrow GetLowerBound(\mathcal{C}|R)$   ▷ Get the lower bounds for the added taxa
9: end for
10: return $TraverseAdd(\mathcal{C}, T, root[T], 3, \{m_1, m_2\}, U, L)$
```

Algorithm 5 TraverseAdd($\mathcal{C}, T, x, i, A, U, L$)

```
▷ $T$ is a tree on the leaves set $A$, $x$ is one of its nodes, $i$ is the index of the taxon to be added to the tree $T$
▷ $U$ and $L$ are upper bound and lower bounds repectively
1: $y \leftarrow CreateNode(m_i)$   ▷ create a leaf node labelled by $m_i$
2: $z \leftarrow CreateNode()$   ▷ create a empty internal node
3: $AddTaxon(T, x, y, z)$
4: $A \leftarrow A \bigcup \{m_i\}$
5: $flip = Flip\_Estimate(\mathcal{C}|A, T) + L[i]$
6: if $flip < U$ then
7:    if $i < s$ then
8:       $TraverseAdd(\mathcal{C}, T, x, i+1, A \bigcup \{m_i\}, U, L)$
9:    else
10:      $U \leftarrow flip$   ▷a complete tree is reached, update the upper bound
11:   end if
12: end if
13: $RemoveTaxon(T, y, z)$
14: $A \leftarrow A - \{m_i\}$
15: if $x$ is not a leaf node then
16:    $TraverseAdd(\mathcal{C}, T, left[x], i, A, U, L)$
17:    $TraverseAdd(\mathcal{C}, T, right[x], i, A, U, L)$
18: end if
19: return $U$
```

3.3 Heuristics Algorithm for Minimum Flip

We now describe a heuristic algorithm for solving minimum flip problem with a large data set. This heuristic uses the uphill searching strategy employed in PAUP and commonly used among biologists. The main idea of this strategy is to start from a tree, called starting point, and rearrange branches in this tree to form different trees. If a better tree, that is, a tree with smaller number of flips, is found, then replace the starting point with this better tree. This uphill algorithm continues until no possible rearrangements of a given tree result in a better tree and that tree is returned as the estimate of the minimum flip tree. In this section, we first introduce three types of branch-swapping methods that are widely used to create new trees by rearranging branches. Then, we describe a greedy algorithm to obtain a starting point. Finally, we will give a detailed uphill algorithm which is used for minimum flip problem.

3.3.1 Branch-swapping Operations

3.3.1.1 Nearest Neighbor Interchange (NNI)

For a rooted binary tree T, a *NNI operation* on an internal node consists of swapping one of its children with its sister. For one NNI operation, there are $2s-4$ different trees that can be possibly obtained from the tree T with s taxa, because there are $s-1$ internal nodes in the tree T and only two possible rearrangements for each internal node except the root (as shown in Figure 3.4).

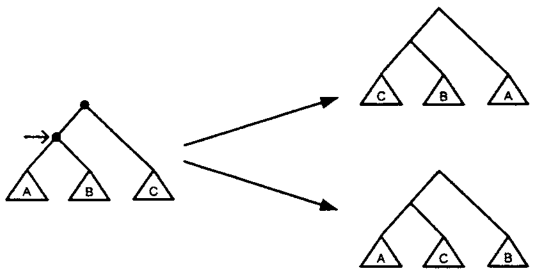


Figure 3.4 Two rearrangements are possible for one NNI operation on an internal node.

3.3.1.2 Subtree Pruning and Regrafting (SPR)

A SPR operation proceeds by removing a subtree t from a rooted binary tree T and then attaching t to each possible branch of the remaining subtree $T - t$. The root of the pruned subtree t is called a *prune node.* A general case of SPR is illustrated in Figure 3.5. A special case of the SPR operation (Figure 3.6) occurs when the the parent of the prune node is the root of the original tree T. In this case, after regrafting subtree t to the subtree $T - t$, the prune node's sister is set as new root. Note that the NNI operation can be considered as a SPR operation but not vice versa.

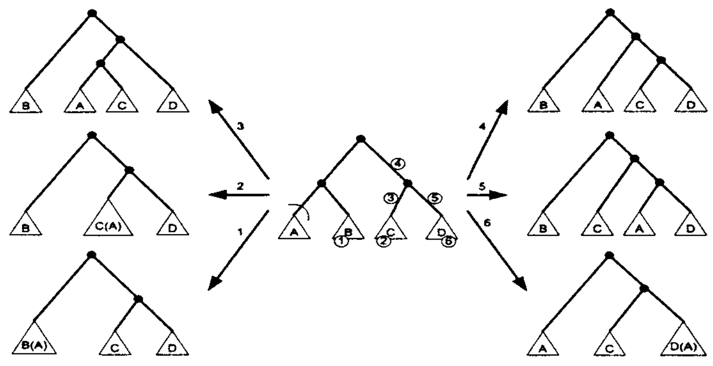


Figure 3.5 A general case of SPR operation. Subtree A is detached from the original tree T, and attached to any possible branch of remaining part of the tree $T - A$, such that different trees are created. 1,2 and 6 are represented any branch inside subtree B,C and D respectively. B(A) denotes a tree formed by regrafting subtree A to a branch in subtree B. C(A), D(A) have similar meanings.

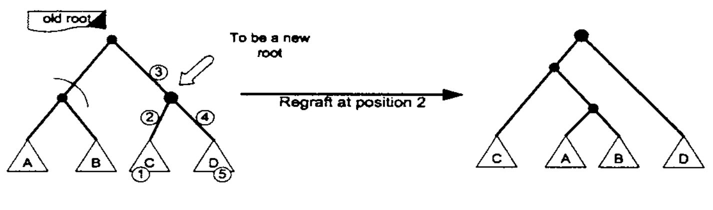


Figure 3.6 A special case of SPR operation. The prune node's parent is the root of original tree T. A general SPR operation is performed, and then the root is changed to the prune node's sister.

3.3.1.3 Tree Bisection and Reconnection (TBR)

A TBR operation cuts off an edge of a rooted binary tree T, dividing the tree T into two parts: t and $T - t$. It then chooses a subtree, say t without loss of generality, which does not contain the root node of the original tree T, from the subtree t, create a new subtree t' by "bending" each possible edge to obtain a new root, and then attaching t' to any possible branch in the subtree $T - t$. This is similar to regrafting in SPR operation. An example is shown in figure 3.7. Note that SPR operation can be viewed as a TBR operation but not vice versa.

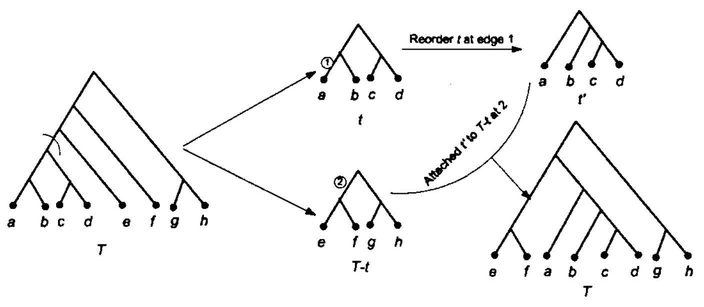


Figure 3.7 A tree T is divided into two subtrees t and $T - t$ by a cut. Subtree t' is formed by "bending" the branch 1 in subtree t. A TBR tree is created when subtree t' is regrafted to one of the branches in subtree $T - t$.

3.3.2 Greedy Algorithm

This algorithm is used to get a starting point for the heuristic algorithm described later. Similarly to exhaustive enumeration, the greedy search begins with the unique initial tree formed from the first two taxa in the input data set. The third taxon is inserted into every possible branch in the initial tree to form all possible three taxon trees. Algorithm 2 is then applied to estimate the minimum number of flips for each of those three taxon trees. The best three taxon tree, a tree with the minimum number of flips among all possible three taxon trees, is chosen to add the fourth taxon. Taxa are added to the tree in such a way until a complete tree is reached. The running time for this algorithm is $O(s^4 \times r)$.

3.3.3 Heuristics

For large data sets, Algorithm 6 is used for minimum flip problem.

Algorithm 6 HeuristicMinFlip ($\mathcal{C}$, *branchswap*)

```
▷ $\mathcal{C}$ is a set of characters, branchswap is a function pointer referring to one of the procedures in
▷ {NNI, SPR, and TBR } and returns all the neighbor trees according to {NNI, SPR, and TBR } .
1: order taxa randomly, and obtain starting point $T$ using above Greedy Algorithm.
2: initialize besttreelist empty
3: add the tree $T$ to besttreelist
4: $bestscore \leftarrow Flips\_Estimate(\mathcal{C}, T)$
5: for all $T_1 \in besttreelist$ do
6:   for all $T_2 \in branchswap(T_1)$ do
7:     if $Flips\_Estimate(\mathcal{C}, T_2) \leq bestscore$ then
8:       if $Flips\_Estimate(\mathcal{C}, T_2) < bestscore$ then
9:         $bestscore \leftarrow Flips\_Estimate(\mathcal{C}, T_2)$
10:        empty besttreelist
11:      end if
12:      add $T_2$ to besttreelist
13:    end if
14:  end for
15: end for
16: return besttreelist
```

This algorithm does not test all possible trees and thus do not guarantee that the resulting trees are optimal. It is clear that the result trees obtained by such a algorithm depends on the starting point. From this reason, we usually run this algorithm several times and choose the best result. This is equivalent to repeating the procedure with different taxon addition orderings.

CHAPTER 4. SIMULATION STUDIES OF SUPERTREE CONSTRUCTION METHODS

4.1 Introduction

To test the effectiveness of flipping as a supertree construction method, we conducted a series of tests on artificial data, and ran both the exact branch-and-bound algorithm and the uphill searching heuristics described in the previous chapter. Our goal was to examine several parameters that affect the performance of building supertrees. By fixing those parameters, we can compare MRP, MRF, MC and MMC supertrees. The simulation studies are divided into two kinds of experiments: those where exact solutions were computed and those where heuristics were used. The former are based on very small artificial data set and use a branch-and-bound algorithm to construct exact MRP and MRF supertrees; while the latter using uphill searching heuristics to build heuristic MRP and MRF supertrees. The two experiments will be described in the section 3 and section 4 respectively.

4.2 Criteria Used for Tree Comparison

In this section, we first define some metrics used for tree comparison. Then, we describe the criteria used in our simulation result analysis. Let $M_{\mathcal{T}}$ denote the true tree, $\mathcal{I}_{\mathcal{T}}$ denote a set of source trees, and $\mathcal{S}_{\mathcal{T}}$ denote a set of supertrees.

A *maximum agreement subtree*(MAST) of two phylogenetic trees T_1 and T_2, denoted by MAST(T_1,T_2), is an agreement subtree with the largest possible number of leaves. The maximum agreement subtree can best represent the common information provided by the two input trees. The *MAST fit* of two input trees T_1 and T_2 is the size of MAST(T_1,T_2) divided by the number of common leaves in both T_1 and T_2. That is,

$$\text{MAST fit } (T_1, T_2) = \frac{|MAST(T_1,T_2)|}{|\mathcal{L}(T_1) \bigcap \mathcal{L}(T_2)|}$$

To assess the accuracy of supertree methods, we need to consider two issues: the degree to which a supertree agrees with the true tree and the degree to which a supertree recovers the information in the source trees.

The first issue concerns how well different supertree building methods based on subsets of taxa reflect the true tree. Since some methods such as MRP and MRF might produce more than one supertree, we used the average MAST fit of supertrees and the true tree as a metric to measure how well those supertrees agree with the true tree. The average MAST fit is calculated as following:

$$\text{average MAST fit } (\mathcal{S}_T, M_T) = \frac{\sum_{t \in \mathcal{S}_T} MASTfit(M_T,\ t)}{|\mathcal{S}_T|}$$

The second issue concerns how well a supertree incorporates information in source trees. Here we used the average MAST fit of supertrees and source trees as one of metrics, which was calculated by

$$\text{average MAST fit } (\mathcal{S}_T, \mathcal{I}_T) = \frac{\sum_{t \in \mathcal{S}_T} \sum_{t' \in \mathcal{I}_T} MASTfit(t,\ t')}{|\mathcal{S}_T| \times |\mathcal{I}_T|}$$

However, this MAST based metric has drawbacks to measure tree similarity especially for those input trees that are not fully resolved. Following a suggestion by Roderic Page, we used *average triplet fit* to measure how well a supertree agrees with source trees (1). Given a supertree T_1 and a source tree T_2, let $d(T_1, T_2)$ be the number of triplets resolved differently in T_1 and T_2; $s(T_1, T_2)$ be the number of triplets resolved identically in T_1 and T_2; $r1(T_1, T_2)$ be the number of triplets resolved in T_1 but not in T_2; and $r2(T_1, T_2)$ be the number of triplets resolved in tree T_2 but not in T_1. A triplet fit is defined and calculated by the following:

$$\text{triplet fit } (T_1, T_2) = 1 - \frac{d(T_1,T_2)+r2(T_1,T_2)}{d(T_1,T_2)+s(T_1,T_2)+r2(T_1,T_2)}$$

The average triplet fit of supertrees and source trees is the average value of triplet fit over each pair of supertree and source tree. That is,

$$\text{average triplet fit } (\mathcal{S}_\mathcal{T}, \mathcal{I}_\mathcal{T}) = \frac{\sum_{t \in \mathcal{S}_\mathcal{T}} \sum_{t' \in \mathcal{I}_\mathcal{T}} tripletfit(t,\ t')}{|\mathcal{S}_\mathcal{T}| \times |\mathcal{I}_\mathcal{T}|}$$

Note that the metrics described above is only used in heuristic solution comparison experiment.

4.3 Exact Solution Comparison Experiment

4.3.1 Materials and Methods

- **Hardware**: Four Linux based Dell workstations each with one 667MHz Pentium-III CPU and 512MB memory was used in this study.

- **Software**: Several programs were involved in this simulation study. They are listed below:

  1. **r8s version 1.5**(16): A program from Michael J.Sanderson that is used to generate a set of random trees as "true" trees.
  2. **Seq-Gen version 1.2.3**(17): Several artificial data sets (simulation of DNA sequences)were created by this program based on the "true" trees.
  3. **PAUP version 4.0**(18): This is used to build most parsimonious trees.
  4. **BBMFT**: A program written by myself for constructing exact minimum flip trees using branch-and-bound algorithm described in previous chapter.
  5. **MCSupertree**: My implementation of Semple and Steel's mincut algorithm. This is used to construct mincut supertree.

- **Method**:

  The parameters we chose to study are the size and the number of source trees, the fraction of shared taxa between source trees, and the level of disagreement between source trees (19). The level of disagreement is largely a product of evolutionary noise that was added. Each parameter combination was replicated 50 times. A single replicate of the simulation is shown in figure 4.1. The details of each step are described below.

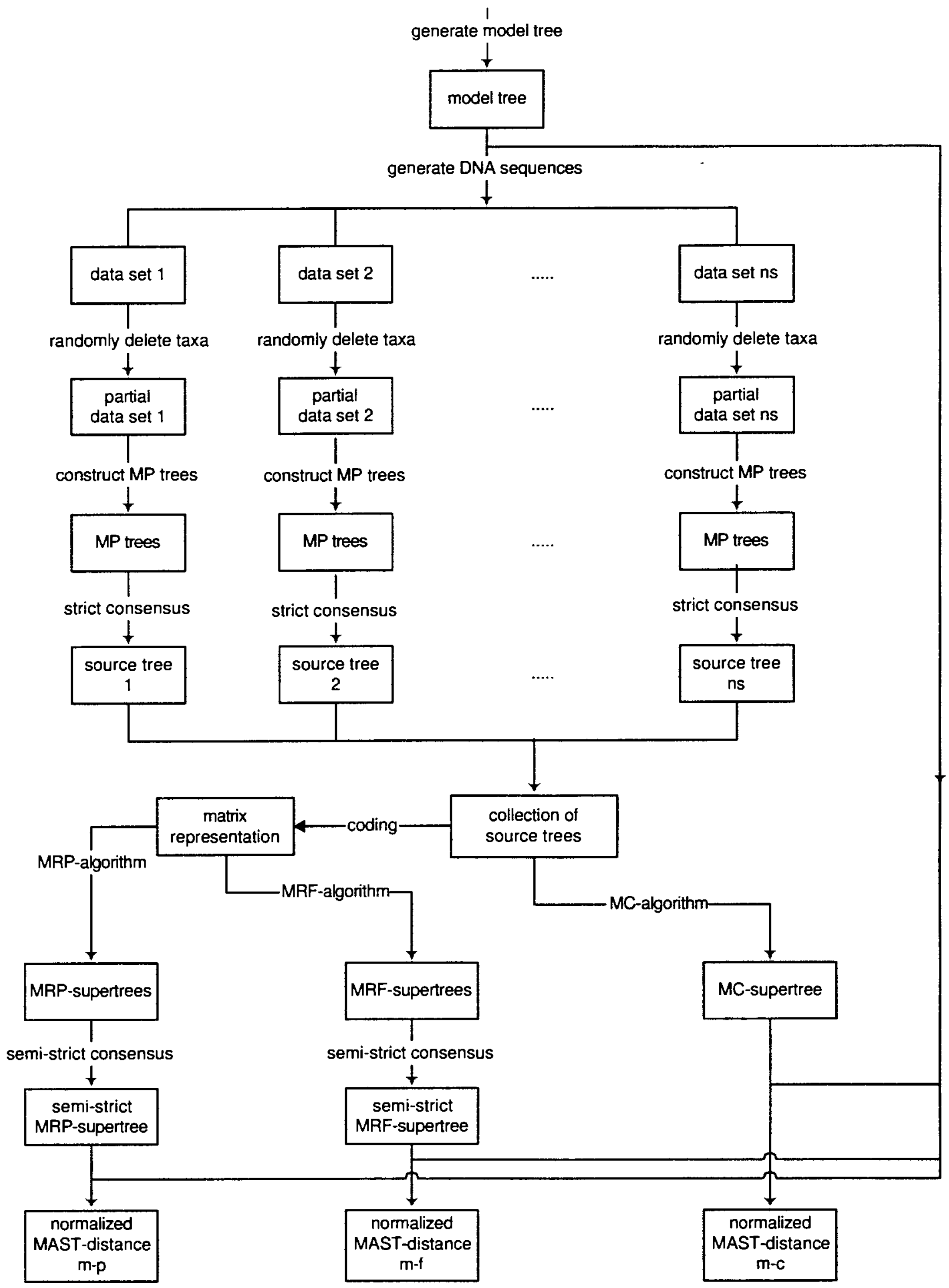


Figure 4.1 Flowchart of the experiment to assess the accuracy of different supertree methods.

1. **Randomly generate a model tree:** In our simulation study, we generated a model tree with n taxa using the program R8S with the default parameter setting. This produced a model tree according to a stochastic Yule birth process conditional on a fixed number of tips and a fixed time between the root of the tree and the present (20). Thus, the model tree fulfilled the desirable property of preserving a constant age distribution of node times independent of the number of tips, allowing comparisons between different tree sizes.

2. **Generate data sets of DNA sequences, and construct source trees from them:** We used the program SEQ-GEN to generate a certain number, ns data sets (data sets 1 to ns, in Figure 4.1). Each data set contained n DNA-sequences of length l generated according to a standard HKY85 Markov substitution model (21), assuming equal base frequencies, a transition-transversion ratio of 2.0 and gamma distributed rates with a shape-parameter of 0.5. We imposed partial overlap between data sets by randomly deleting taxa in each of them (partial data sets 1 to ns, in Figure 4.1). A taxon was deleted by a fixed probability 25%, 50% or 75%.

   For each data set, we constructed MP trees using PAUP* 4.0 (MP trees in Figure 4.1), and used their strict consensus as a *source tree.* As a result of the deletion process our source trees varied in size according to a binomial probability distribution. According to the number of data sets and their sequences length, we specified two experiments.

   - *fixed sequence length experiment*: We generated a large data set in which each taxon has a fixed sequence length L. By partitioning the large data set, we obtained small data sets in which each taxon has the same sequence length $l = L/ns$ (thus l declines as ns increases). From the small data sets, the source trees were generated.

   - *proportional sequence length experiment*: Each of the ns data sets has a fixed sequence length l and the total sequence length L is proportional to l and ns. That is, $L = ns * l$.

Note that in the model of character evolution described above, evolution is independent and identical at each site and independent along each lineage, so it is valid to generate one large data set and then partition it into smaller data sets.

3. **Coding source trees:** For a collection of source trees derived from the same model tree, we generated its matrix representation.

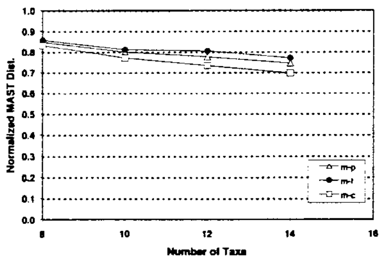
4. **Building MRP, MRF, and MC supertrees:** For each matrix obtained from step 3, we used *exact* MRP, MRF, and MC algorithms to construct MRP, MRF, and the MC supertrees respectively. MRP, and MRF supertrees are not necessarily unique. Thus, we used the semi-strict consensus of MRP, and MRF supertrees respectively for comparison against the model tree.

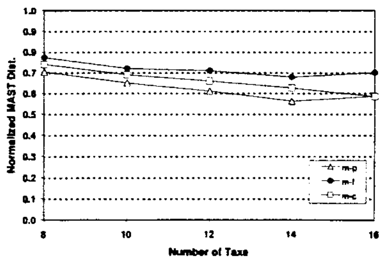
5. **Assessing accuracy:** To assess the accuracy of supertree methods, we calculated the *normalized* MAST-*distance* of the semi-strict supertrees (semi-strict MRP, semi-strict MRF, and MC supertree), against the model tree respectively. The *normalized* MAST-*distance* of a supertree against the model tree is the number of the leaves of their maximum agreement subtree (22) normalized by the number of leaves of the supertree. The average normalized MAST-distance of all replicates (50) was used to evaluate the MRP, MRF and MC algorithms.

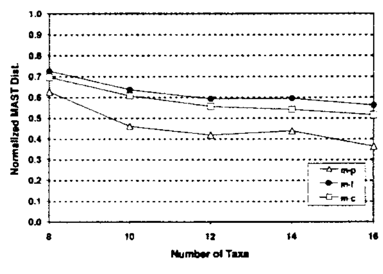
4.3.2 Results and Analysis

Two factors, the number of taxa and the number of characters, are known to influence the accuracy of all phylogenetic reconstruction methods, and supertree methods are no exception. Figure 4.2 shows an overall decline in accuracy with increasing size of the source trees, probably reflecting the increasing difficulty of estimating larger trees with the same number of characters (19). This effect gets larger as the degree of overlap between source trees declines (deletion fraction increases). Not surprisingly, overall accuracy increased with the number of characters under both fixed and proportional sequence length models (results not shown).

The 95% confidence intervals for the mean normalized MAST-distances (± 2 standard errors) were about $0.05 - 0.10$ in all experiments. In almost all comparisons between methods







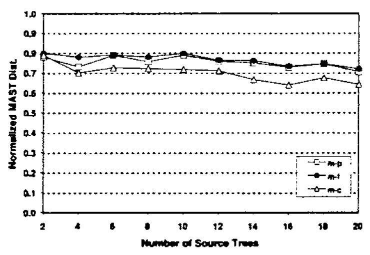
Normalized MAST-distance between model tree and MRP ("m-p"), MRF ("m-f") and MC ("m-c") semistrict consensus supertree estimates of that tree, as a function of size of source tree. Fixed sequence model with 1250 characters was used. Clockwise the figures represent deletion probabilities of 0.25, 0.50, 0.75 respectively.

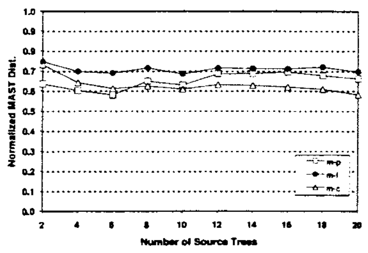
Figure 4.2 Variable taxa, and fixed sequence length experiment.

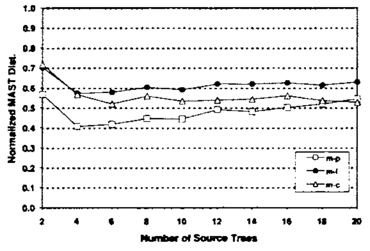
(Figures. 4.2–4.4) the performance of the MRF method was consistently better than MRP and MC methods, but was rarely distinguishable from the other two with statistical confidence.

The effect of number of source trees depended somewhat on whether the total number of characters was fixed or proportional to the number of trees. When the number was fixed (Figure. 4.3), accuracy was nearly independent of the number of source trees, possibly because the benefits of combining more independent source trees were matched by lowered accuracy of each source tree because of fewer characters. When the number of characters was allowed to be proportional to the number of trees, such that each source tree had a constant number (Figure 4.4), accuracy increased monotonically with the number of source trees, although rather slowly.

Deletion probability controls the amount of taxonomic overlap between trees. When it is high, source trees often contain very different subsets of taxa. Comparisons between columns in Figures. 4.2–4.4 show consistently that increasing deletion probability (and hence decreasing overlap) decreases accuracy. At a high deletion fraction of 0.75, for example, no method performed better than a MAST-distance of 0.6 – 0.7, regardless of how many source trees were







Distance between model tree and MRP, MRF, MC supertree estimates as a function of number of source trees, n_S. Fixed sequence length model was used with 12 taxa and a total of 1000 characters. See Figure 4.2 for abbreviations and deletion probabilities

Figure 4.3 Fixed sequence length, and fixed taxa experiment

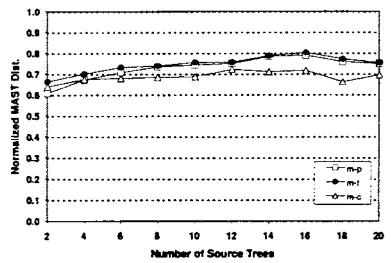
added. Accuracy improves somewhat if the accuracy of the source trees is improved by adding characters (data not shown), but deletion probability remains perhaps the most important determinant of overall success of these supertree methods (as shown also by Bininda-Emonds and Sanderson (19) for the MRP method alone).

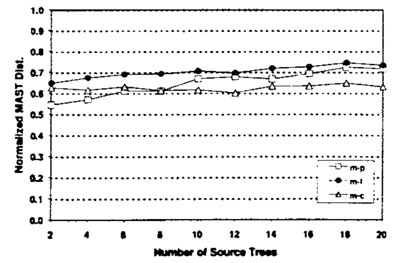
4.4 Heuristic Solution Comparison Experiment

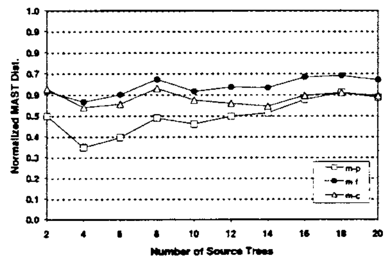
4.4.1 Materials and Methods

Since this simulation study involved a large amount of computational work, it required many powerful computers and several software packages. We now describe our experimental setup.

- **Hardware**: A 38 nodes Linux cluster in the University of California, Davis was used in our simulation study. Each of those node consists of dual AMD Athlon(tm) 1400MHz CPUs and 1GB memory.
- **Software**: Besides r8s, Seq-Gen, and PAUP, the packages described in previous section, we also used the following programs:







Distance between model tree and MRP, MRF, MC supertree estimates as a function of number of source trees, *ns*. Proportional sequence length model was used with 12 taxa and 100 characters per source tree data matrix. See Figure 4.2 for abbreviations and deletion probabilities.

Figure 4.4 Proportional sequence length, and fixed taxa experiment

1. **HeuristicMFT**: A program written by myself for constructing minimum flip trees using uphill searching heuristics.
2. **supertree**(1)(23): A program written by Roderic Page. That implements Semple and Steel's mincut algorithm and his Modified mincut algorithm.

- **Methods**: Two parameters: the number of source trees and the fraction of shared taxa between source trees were studyed in this experiment. We repeated each parameter combination 100 times. A single replicate of the simulation is shown in Figure 4.5 and 4.6. This is very similar to the methods used in exact solution comparison experiment. The details of each step are described below. We skip the detailed description if the procedure is identical to that in the exact simulation experiment.

1. **Randomly generate a model tree**
2. **Generate data sets of DNA sequences, and construct source trees and total evidence trees from them:** This is the same as step 2 in exact simulation experiment except that we added building total evidence trees and changed the scheme to control the fraction of shared taxa among source trees.

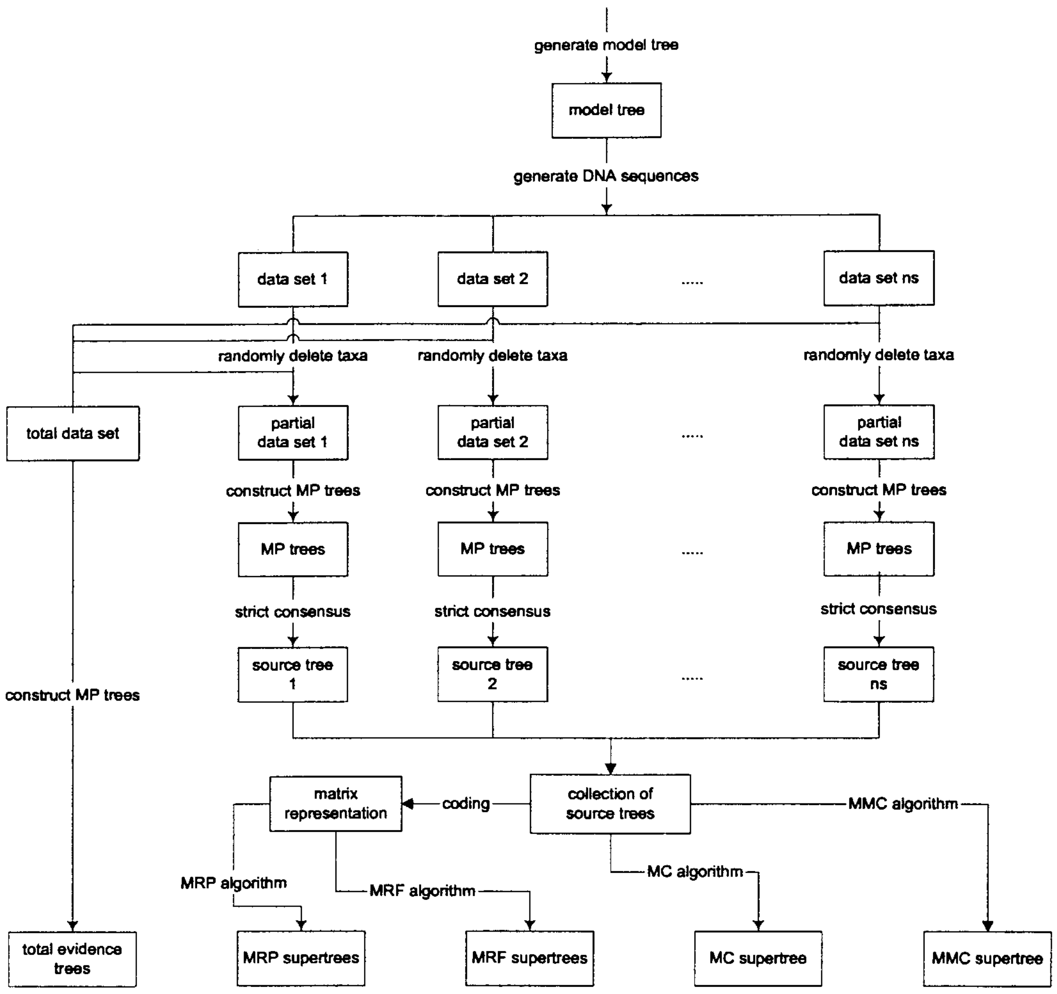


Figure 4.5 Flowchart of the heuristics solution comparison experiment

After creating those ns small data sets, we merge all of them into a big total data set by concatenating the DNA sequences in each data set. We built MP trees, called *total evidence trees*, from the total data set using PAUP's heuristic search. The parameters used for heuristic search in PAUP 4.0 are 5 random-addition-sequence addiction replications, TBR branch-swapping, and 5 equivalent parsimony trees kept.

We imposed partial overlap between data sets by randomly deleting a certain number of taxa in each of them (partial data sets 1 to ns, in Figure 4.5). A fixed probability 25%, 50% or 75% of taxa in each data set was deleted.

3. **Coding source trees**

4. **Building MRP, MRF, MC, and MMC supertrees:** We used uphill searching heuristics to build MRP and MRF supertrees instead of using bunch-and-bound algorithm, as in the exact solution comparison experiment. The parameters for building MRP tree are the same as in constructing total evidence trees. Constructing MRF supertrees used the same parameters, except the branch-swapping was SPR. We construct MC supertree by Roderic's implementation and added MMC method in this study.

5. **Assessing accuracy:** To assess the accuracy of supertree methods, we need to consider the degree to which the supertree agrees with the source trees and the supertree recovers the true tree. It is also important to know how well the source trees reflect the true tree. Otherwise, the measurement of the supertree and the true tree becomes meaningless if the source trees does not contain any information of the true tree. As shown in Figure 4.6, we used average triplet fit and average normalized MAST as metrics to measure the degree of source trees reflect the true tree and supertree recovers the information in the source trees. How good a supertree agrees with the true tree is measured by average normalized MAST.

4.4.2 Results and Analysis

To assess the accuracy of different supertree methods by measuring how well a supertree reflects the true tree in the simulation study, the quality of source trees plays an very important roles. The result shows that the quality of MRP, MRF, MC, and MMC-supertrees measured by average MAST fit or average triplet fit were waved according to the quality of source trees in both fixed sequence length model (Figure 4.7) and proportional model(Figure 4.8. Figure 4.7 and 4.8 also show that the metrics average MAST fit and average triplet fit are coincident in measuring how well supertrees incorporated the information in the source trees.

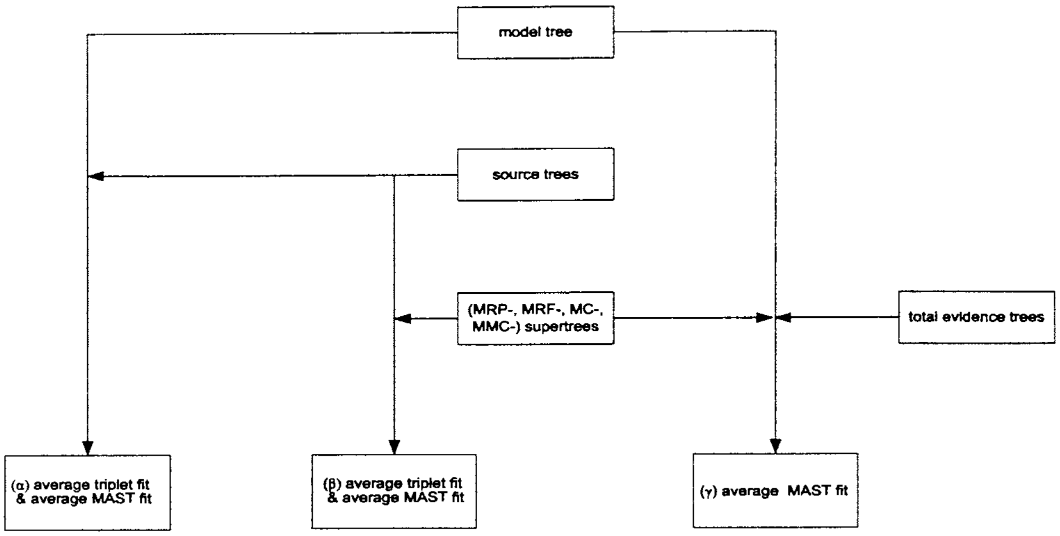

model tree
source trees
(MRP-, MRF-, MC-, MMC-) supertrees
total evidence trees
(α) average triplet fit & average MAST fit
(β) average triplet fit & average MAST fit
(γ) average MAST fit

Figure 4.6 Tree comparison criteria used in heuristic solution comparison experiment. (α) and (β) was used to evaluate how well source trees incorporate the true tree and how well supertrees recovering information in source trees respectively. (γ) was used to measure the degree of supertrees reflecting the true tree.

The degree of overlap between source trees is an important factor to construct supertree. Figure 4.9 show that when the degree of overlap between source trees declines (deletion fraction increases), the MRF, MRP, MC, and MMC supertrees become worse. While the total evidence tree is not affected by this factor since it construed from the total data set not direct or indirect from the source trees.

From Figure 4.7, 4.8, and 4.9, we can see that MRF method is slightly better than MRP method, especially when the deletion probability is high. In other words, MRF is more accurate when the shared taxa among source trees is low. However, the performance of MRF and MRP are not distinguishable as shown in exact solution comparison experiment. Those figures also show that the metric of supertree covering source trees is consistent to the metric of supertree reflecting true tree when the source trees are "good" or source trees incorporate the true tree well. In real supertree reconstruction, we don't know the true tree. However, we can use the metric of supertree covering source trees to measure how well the supertree is.

Figure 4.9 shows that the accuracy of MRF or MRP supertrees are closed to that of the total evidence trees when the source size and the number of source trees is not too small. Thus, estimates of supertree can be obtained with reasonably high accuracy from collections of fragmentary smaller trees but not extremely small with reasonable number of source trees.

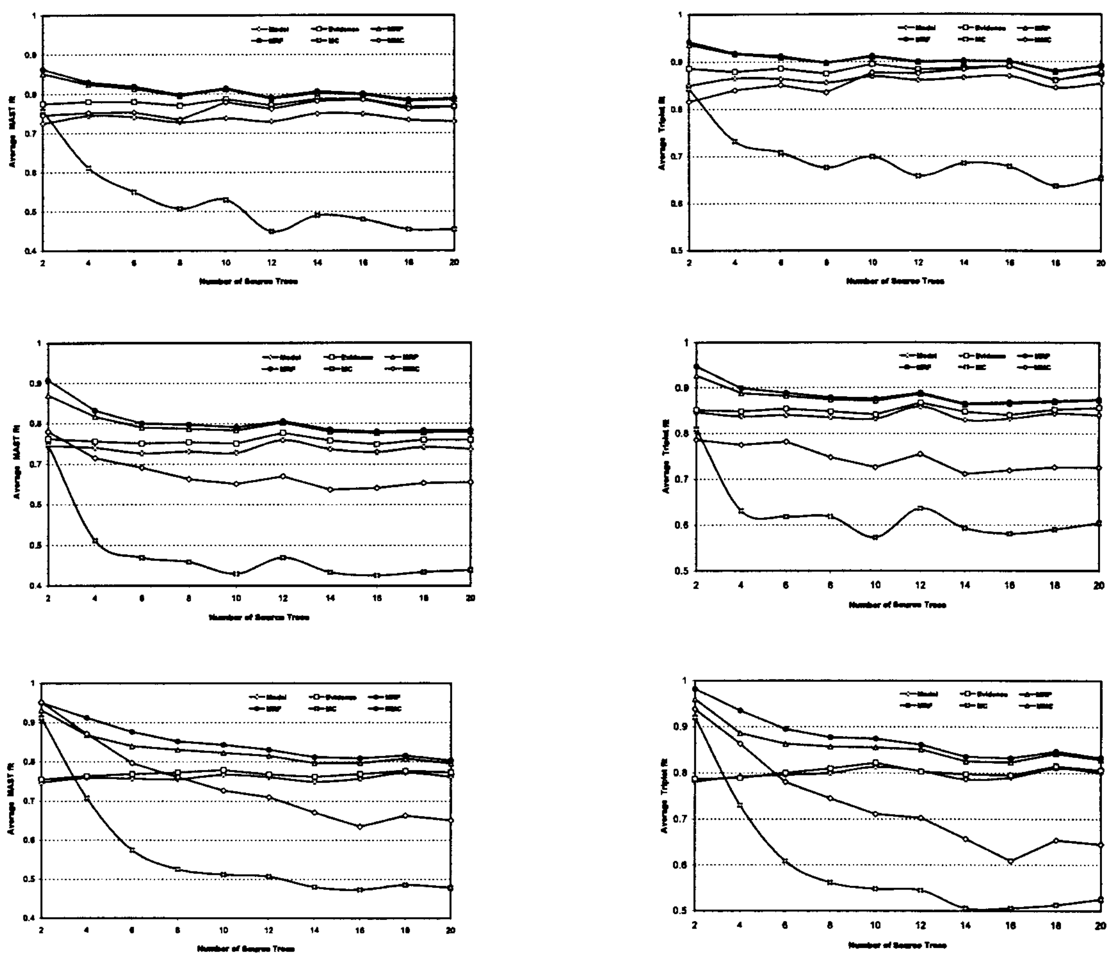


Figure 4.7 Fixed sequence model with 1,000 characters: average $MASTfit$ (first column) and $tripletfit$ (second column) of supertrees and source trees. The figures from top to down represents deletion probabilities of 0.25, 0.50, and 0.75 respectively.

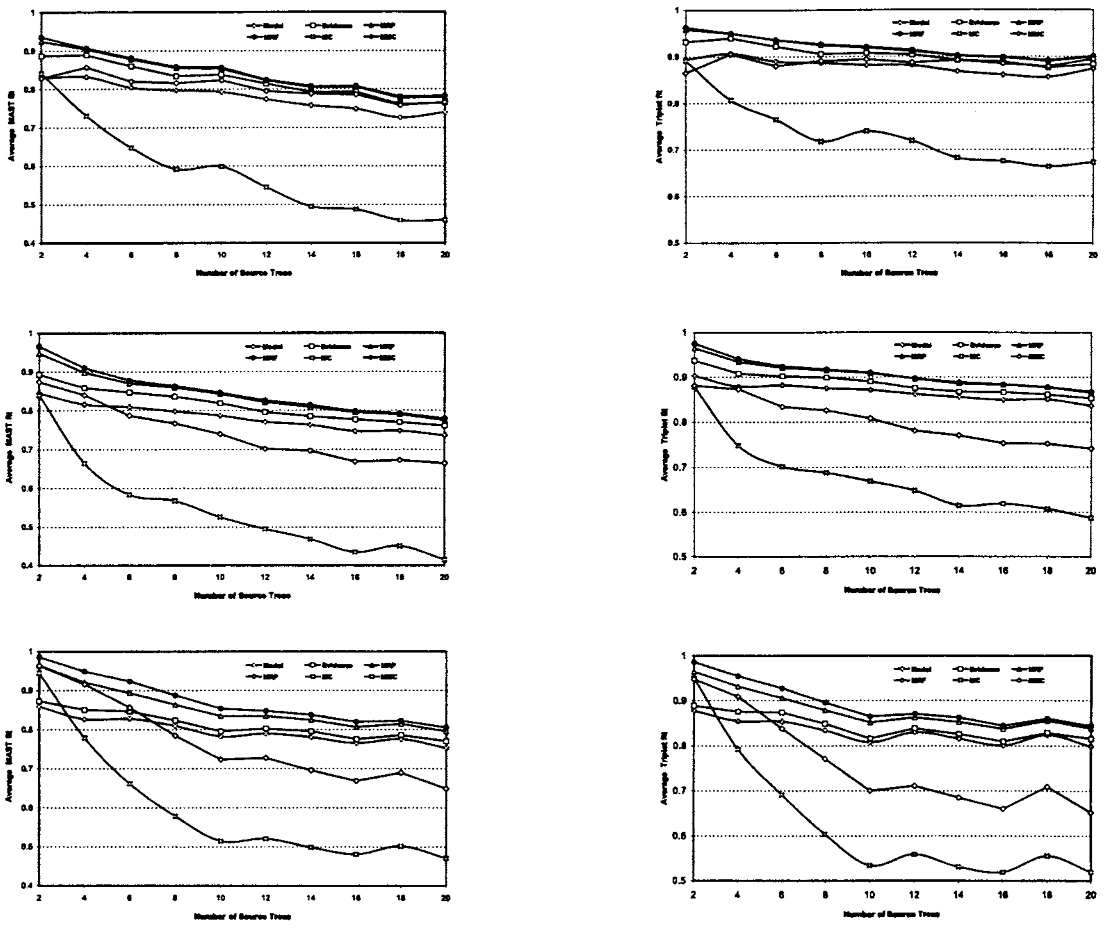


Figure 4.8 Proportional sequence model with a total of 20,000 characters: average *MASTfit* (first column) and *tripletfit* (second column) of supertrees and source trees. The figures from top to down represents deletion probabilities of 0.25, 0.50, and 0.75 respectively.

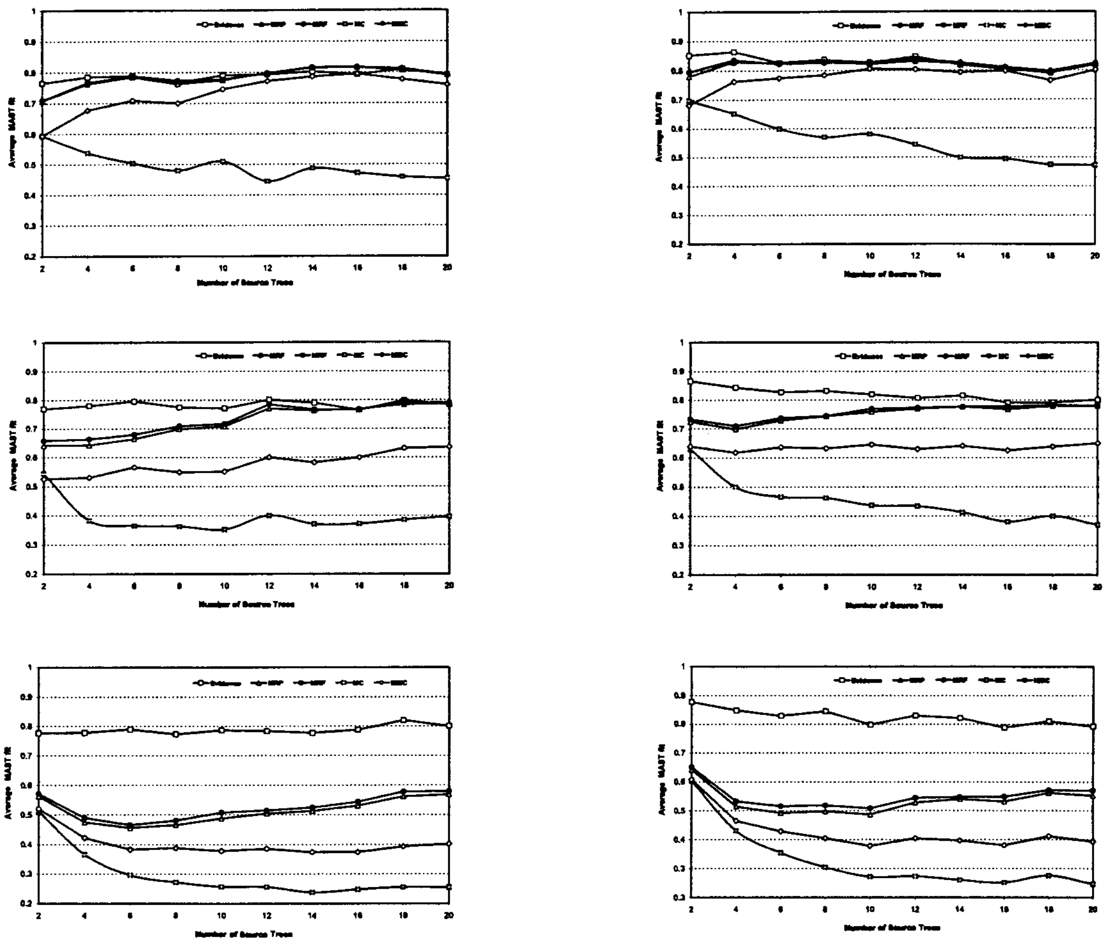


Figure 4.9 Average $MASTfit$ of of supertrees and true tree: The figures of fixed sequence length model is listed in the first column and proportional model in the second column. Figures in both columns from top to down represents deletion probabilities of 0.25, 0.50, and 0.75 respectively.

CHAPTER 5. CONCLUSION AND FUTURE STUDY

Biologists interested in constructing the evolutionary history of life are faced with a large collection of small phylogenetic trees many of which overlap in their label sets. Hence, strong interest has developed in recent years regarding development of methods for constructing supertrees, which combine information from collections of individual trees (24). Disagreement about the precise definition of a supertree, meaning what properties it should exhibit and what information from the source trees it should attempt to preserve (12), has led to a variety of strategies for supertree construction. One key issue is how to resolve the often conflicting information about relationships among the source trees.

In this paper we have outlined one strategy for resolving this conflict between source trees. Starting with the combined matrix of the matrix representations of all source trees, it defines an optimization problem based on minimizing the number of changes between 0 and 1-flips-needed to convert this combined matrix into a *perfect phylogeny*–that is, one perfectly consistent with a single supertree. This problem is $\mathcal{NP}$-complete, as is the MRP parsimony version of this problem, which is used by most biologists working with real data.

Polynomial time algorithms are known for other kinds of supertree construction, such as Semple and Steel's MC and Page's MMC algorithms, but our experimental studies suggest that MRF and MRP usually outperforms MC and MMC in both accurately reconstructing the correct supertree and heuristic supertrees. The exact solution experiment is limited to relatively small trees ($\leq$ 16 taxa), so that an exact MRF and MRP algorithm could be used. The heuristic solution experiment is limit on 100 taxa trees since MRF is pretty slow and many other computational work.

Experiments showed that accuracy of MRF is affected by many of the same factors influencing other supertree strategies (e.g., (19)). Accuracy generally improves with the number of source trees included, with the degree of overlap in their label sets, and with the accuracy of the source trees themselves.

It is very important to measure how well the supertree that we constructed. Does it reflects the "true" tree? Our simulation studies show that we can use the quality of the supertree fitting source trees to measure how well the accuracy of supertree recovering the "true" tree.

These performance analyses are encouraging, because they suggest that estimates of a supertree can be obtained with reasonably high accuracy from collections of fragmentary smaller trees. MRF is one method for extracting the signal from the ever present noise contained in these collections of trees, and it apparently is relatively successful compared to available methods. However, the computational time for MRF is much slower than MRP.

A fast heuristic of MRF is needed for future study. In addition, the accuracy of supertree is highly related to the quality of source trees. How to discover the "bad" trees among the source trees using statistics is also need to study.

BIBLIOGRAPHY

[1] R.D.M. Page (2002). Modified mincut supertrees. *Lecture Notes in Computer Science, 2452*, 537–551

[2] J. Felsenstein (1981). Evolutionary trees from gene frequencies and quantitative characters: finding maximum likelihood estimates. *Evolution, 35*, 1229–1242.

[3] E. Adams III (1986). N-trees as nestings: complexity, similarity and consensus. *Journal of Classification, 3*, 299–317.

[4] G.F. Estabrook, C.S. Johnson and F.R. McMorris (1975). An idealized concept of the true cladistic character. *Mathematical Biosciences, 23*, 263–272.

[5] C. Korostensky, G. H. Gonnet (2000). Using traveling salesman problem algorithms for evolutionary tree construction. *Bioinformatics, 16*(2), 101–103.

[6] I. Pe'er , R. Shamir, and R. Sharan (2000). Incomplete directed perfect phylogeny. *Lecture Notes in Computer Science, 1848*, 143–153.

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

[8] S. Kannan and T. Warnow (1997). A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing, 26*(6), 1749–1763.

[9] W.H.E. Day, and D. Sankoff (1986). Computational Complexity of Inferring Phylogenies by Compatibility. *Systematic Zoology, 35*(2), 224–229.

[10] J. Felsenstein (1982). Numerical methods for inferring evolutionary trees. *Quarterly Review of Biology*, *57*, 379–404.

[11] M. Constantinescu and D. Sankoff (1995). An efficient algorithm for supertrees. *Journal of Classification*, *12*, 101–112.

[12] C. Semple, M. Steel(2000). A supertree method for rooted trees. *Discrete Applied Mathematics*, *105*, 147–158.

[13] A.V. Aho, S. Yehoshua, T.G. Szymanske, J.D. Ullman (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, *10*(3), 405–421.

[14] M. R. Garey and D. S. Johnson (1979). *Computers and Intractability: A guide to the theory of NP-completeness.* New York: W.H. Freeman.

[15] P. Kearney, M. Li, J. Tsang, and T. Jiang (1999). Recovering branches on the tree of life: an approximation algorithm. *SODA*, 537–5465.

[16] M. J. Sanderson (2001). *r8s software.* http://ginger.ucdavis.edu/r8s/r8s.html (date accessed : Dec. 11, 2001).

[17] A. Rambaut and N. C. Grassly (1997). Seq-Gen: an application for the Monte-Carlo simulation of DNA sequence evolution along phylogenetic trees, *Comput. Appl. Biosci.*, *13*, 235–238.

[18] D. L. Swofford (1999). *PAUP*: Phylogenetic analysis using parsimony (*and other methods), Version 4.*

[19] O. R. P. Bininda-Emonds, and M. J. Sanderson (2001). An assessment of the accuracy of MRP supertree construction, *Systematic Biology*, *50*, 565–579.

[20] S. M. Ross (1996). *Stochastic processes.* New York: Wiley Press.

[21] D. L. Swofford, G. J. Olsen, P. J. Waddel, and D. M Hillis (1996). Phylogenetic inference. In (D. M. Hillis, C. Moritz, and B. K. Mable, eds.), *Molecular Systematics(2nd ed.)*, 407–514. Sinauer Associates, inc., Sunderland, Ma.

[22] M. Farach, T. Przytycka, and M. Thorup (1995). Agreement of many bounded degree evolutionary trees. *Information Processing Letters*, *55*,279–301.

[23] R.D.M. Page (2002). *Supertree Software.* http://darwin.zoology.gla.ac.uk/ rpage/supertree (date accessed : Oct. 8, 2002)

[24] M. J. Sanderson, A. Purvis, and C. Henze (1998). Phylogenetic supertrees: assembling the trees of life. *Trends Ecol. Evol.*, *13*, 105–109.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. David Fernández-Baca for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Oliver Eulenstein and Dr. Gavin Naylor. I am also grateful the for guidance and many advice given by Dr. Mike Sanderson and Dr. John G Burleigh. Thanks them for helping me to design experiment and analyze the results. I would additionally like to thank my officemates Wen-Chiek Chang and Justin Schonfeld for their many helpful discussions on research.