A NEW PHYLOGENETIC TREE MODEL

FOR FUZZY CHARACTERS

By

ANDY (WINGHANG) AUYEUNG

Bachelor of Science

University of Central Oklahoma
Edmond, Oklahoma
2000

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
May, 2005

A NEW PHYLOGENETIC TREE MODEL

FOR FUZZY CHARACTERS

Thesis Approved:

Blayne E. Mayfield

Thesis Adviser

John P. Chandler

G. E. Hedrick

Ulrich Melcher

A. Gordon Emslie

Dean of the Graduate College

PREFACE

Computer Science began as an extension of the study of Mathematics to investigate the mechanisms of mathematical computation. Modern Computer Science often involves the transition from a problem in some domain, to a mathematical model, and subsequently to a computer-based solution. A good transition requires both an effective description and an efficient representation of the application. This dissertation proposes a new phylogenetic tree model, called fuzzy phylogeny. Fuzzy phylogeny is an extension of a classic phylogenetic model, called perfect phylogeny. The motivation and the detail definition of fuzzy phylogeny are first explained. Then, an algorithmic solution that transforms the fuzzy phylogeny problem to its perfect phylogeny counterpart is presented. Next, relaxation problems of fuzzy phylogeny are proposed and the complexities are studied. Finally, the relaxation problems are solved by three approaches and empirical analyses are performed.

ACKNOWLEDGMENTS

First, I thank my advisor, Dr. Blayne Mayfield for his guidance. His patience and comfort have made this academic pursuit pleasant. I also wish to thank the members of my dissertation committee; each provided me many valuable comments and suggestions, Dr. John Chandler, Dr. George Hedrick and Dr. Ulrich Melcher. I want to especially thank Dr. Ulrich Melcher for his mentoring and assistance beyond the dissertation. I thank the Department of Computer Science of Oklahoma State University for the financial support and teaching opportunities I have received. Also, this dissertation is impossible without the finest academic environment the Department offers. I thank Rachel Monn, Tammy Davis and David Monismith for proof reading of my dissertation.

I thank my family, my grandfathers, my grandmothers, my mother, my brother, sister and other family members for their love and support. I also thank my loyal companions, Casa and DD, who have been the most wonderful pets in the world. I thank the many loving people from the First United Methodist Church in Stillwater: Pastor Stan Warfield, Pastor Steven Roach, Ron and Karel Payne, Joe and Glenell Owens, the Davis's (Ed, Tammy, Kelsey and Kyla), Troy and Jana Rosenkranz, and every member of the church. I wish to thank my brother and sister in Christ, Stephen Hobson and Rachel Monn for their friendship and care.

Finally, I thank the Lord God Almighty for making everything possible.

TABLE OF CONTENTS

Chapter Pages

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Chapter 1 Introduction

1.1 Background

The study of phylogenetics involves the identification of evolutionary relationships between species [50,58,59]. Among different phylogenetic models, trees are the most widely used due to their simplicity and effectiveness [18,23,31,43]. When constructing a phylogenetic tree, distances [51] and characters [52,53] are the two most common measurements. In distance-based models, the phylogenetic tree is constructed from a set of (non-negative) pair-wise distances between species. Ultrametric trees [31] and additive trees [13,33] are two well-known examples of this model. In character-based models, the phylogenetic tree is constructed according to the presence or absence of the characters observed from the species. Perfect phylogeny [19,20] is one of the classical character-based models. In this model, there are only two states for each character: zero, where the character is absent; and one, where it is present. The perfect phylogeny problem is to verify whether a given set of species with a set of characters forms a phylogenetic tree and, if it does, to construct the corresponding phylogenetic tree. Dan Gusfield [30] describes an $O(mn)$ time algorithm for this problem where m is the number of species and n is the number of characters. In addition, this algorithm is proven to be asymptotically optimal in time. Extended models from perfect phylogeny have been suggested, such as generalized perfect phylogeny [1,2,12,36,37] and phylogenetic network [56]. In the

generalized perfect phylogeny model, each character has finite, multiple states. Kannan and Warnow [37] describe an $O(2^{2r} m^2 n)$ algorithm for this problem where the number of states of the characters (r) is fixed. However, this problem has been proven to be **NP-hard** in general. In phylogenetic networks, merging sub-trees is allowed. Wang, et al. [56] propose an $O(n^3)$ algorithm for a restricted version of this problem. However, this problem also is proven to be **NP-hard** in general. Models for continuous characters also exist [21,24,42]. Examples of models of this kind include the Brownian Characters Motion model [14,40,41], the Gap Coding method [44,55] and the Manhattan Metric Parsimony model [22,38].

1.2 Motivation

All previous perfect phylogeny based models assume the states of the characters are discrete. However, fuzzy boundaries between species and degrees of character development commonly are found in nature [10,26,27]. Fuzzy boundaries refer to the ambiguous definition between the presence and absence of a character, and degrees of character development refer to the various expression levels of a character. These phenomena show the need for a more relaxed model. This dissertation proposes the fuzzy (perfect) phylogeny model that extends the perfect phylogeny model to allow fuzzy memberships of the characters. The conservation of properties in the new model then is proven. This dissertation also shows how the fuzzy phylogeny problem can be transformed to the perfect phylogeny problem so that it can be solved using previously developed algorithms, such as Gusfield's algorithm [30].

Although the fuzzy phylogeny problem theoretically is equivalent to the perfect phylogeny problem, the introduction of the fuzziness of characters naturally raises the question about uncertainty due to the imprecision of the experimental measurement and/or the range of incompatibility tolerance. The problem of uncertainty can be addressed by permitting adjustments to the experimental character values within a predefined range or window. But this still leaves the adjustment problem for fuzzy phylogeny, that is, how much should the experimental character values be adjusted. The adjustment problem is carefully analyzed and is proven to be **NP-hard**.

1.3 Summary of Results

This dissertation presents both theoretical and empirical results on the fuzzy phylogeny model and its problems. First, the fuzzy phylogeny model is analyzed. It is proven that the perfect phylogeny problem is a subset of the fuzzy phylogeny problem both in terms of decidability and tree construction. Next, the transformation Φ is introduced to solve the fuzzy phylogeny problem. The transformation Φ converts a fuzzy phylogeny problem into its perfect phylogeny counterpart, so that a fuzzy phylogeny problem can be solved using algorithms for the perfect phylogeny problem, such as Gusfield's algorithm [30].

Second, the adjustment problem of fuzzy phylogeny is analyzed. The adjustment problem (as well as the adjustment problem for perfect phylogeny) is proven to be **NP-hard**. Three algorithms then are proposed to solve the adjustment problem, namely the BF-algorithm, the G-algorithm and the H-algorithm. The BF-algorithm and the G-algorithm are both exact algorithms where the optimality of the solution is guaranteed.

The BF-algorithm reduces the unaccountably infinite solution space to a finite solution space of equivalence classes. The BF-algorithm is intuitive; however, it is impractical due to its slow performance. The G-algorithm is an improvement over the BF-algorithm such that invalid solution space is pruned off. The G-algorithm is practical for small instances. However, due to the complexity of the adjustment problem, the G-algorithm is impractical for large instances. On the other hand, the H-algorithm has the aim of fast performance, where solutions are found in seconds, even for large instances. Furthermore, the H-algorithm produces quality solutions for small to medium instances.

1.4 Organization

The remainder of the dissertation is organized as follows. In Chapter 2, the phylogenetic tree models are overviewed, including ultrametric trees, additive trees, perfect phylogeny, generalized perfect phylogeny, phylogenetic network and other models for continuous characters. Within Chapter 3, the fuzzy phylogeny model is introduced and defined. Furthermore, the model is analyzed, and the transformation from a fuzzy phylogeny to a perfect phylogeny is presented. The adjustment problem is proposed and formulated in Chapter 4. In Chapter 5, the BF-algorithm is proposed and explained. The G-algorithm is discussed in Chapter 6, and the H-algorithm is presented in Chapter 7. In Chapter 8, the empirical studies of the proposed algorithms are shown. Chapter 9 discusses future work related this research. Finally, Chapter 10 provides concluding remarks of the dissertation.

Chapter 2 Review of the Literature

2.1 Ultrametric Tree

The ultrametric tree model [31] is based upon the theory of the molecular clock. This theory suggests that the mutation rate in the genome is constant over time. The implication is that all (present) species deviated from their common ancestor at an equal rate. Therefore, an ultrametric tree is a tree in which all leaves in all sub-trees are of equal distance from their (sub-tree) roots. Below is the definition of the ultrametric tree.

Let **X** be a set of objects (species), and each $x \in \mathbf{X}$ is associated with a unique integer from the set $\mathbf{S} = \{1, 2, \ldots, m\}$, where $m = |\mathbf{X}|$. Let M be an $m \times m$ symmetric matrix of positive real numbers with zeros along the diagonal. Each entry $M[i,j]$ is the evolutionary distance between species i and j. An ultrametric tree for M is a rooted tree T that satisfies all of the following conditions:

1. T contains m leaves, each of which is labeled by a unique number $s \in \mathbf{S}$;

2. each internal node of T is labeled using a value $M[i,j]$, where $1 \leq i, j \leq m$ and has at least two children;

3. along any path from the root to a leaf, the numbers labeling internal nodes strictly decrease;

4. for any two leaves i and j in T, $M[i,j]$ is the label of the lowest common ancestor of i and j in T.

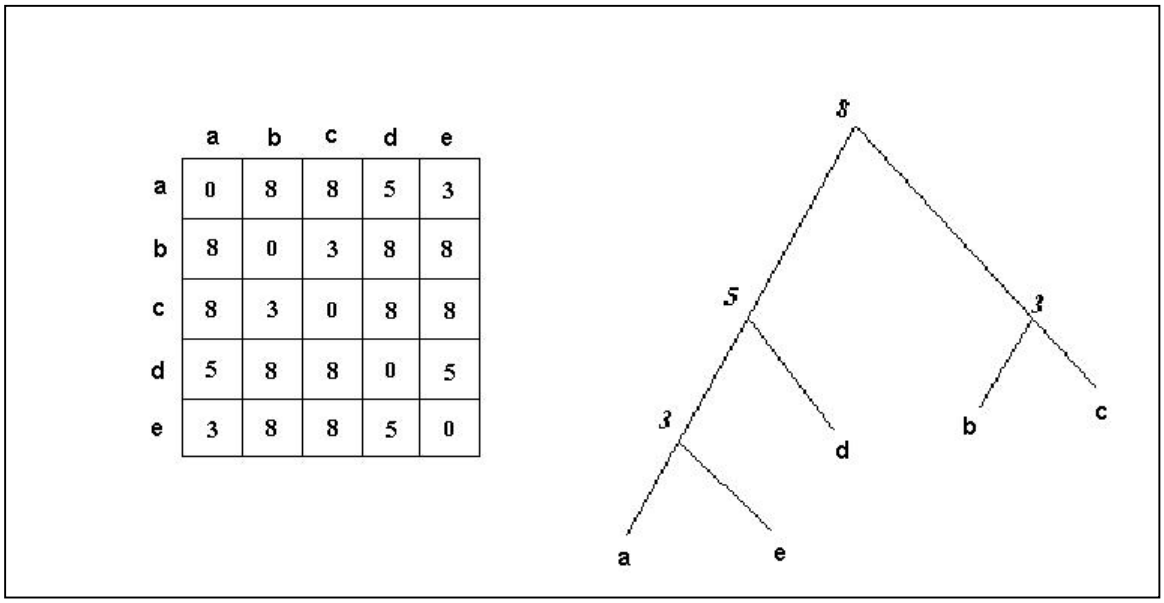


Figure 2.1: A 5×5 matrix *M* and its ultrametric tree *T*.

A matrix is said to be ultrametric if it has an ultrametric tree. Furthermore, the ultrametric tree problem is as follows. Given a matrix *M*, one wants to determine whether there is an ultrametric tree for *M* and to construct *T* if it exists. Figure 2.1 is an example of a matrix *M* and its ultrametric tree *T*. The ultrametric property can be described by Lemma 2.1.

Lemma 2.1:

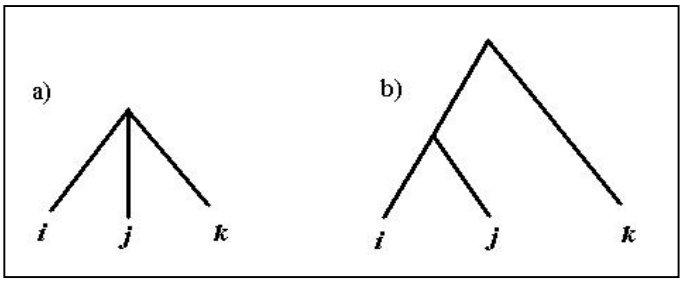
An $m \times m$ symmetric matrix *M* of positive real numbers with zeros along the diagonal is ultrametric if and only if for any three indices *i, j* and *k* ($1 \leq i, j, k \leq m$), the maximum of $M[i,j]$, $M[i,k]$ and $M[j,k]$ is non-unique.

Proof:

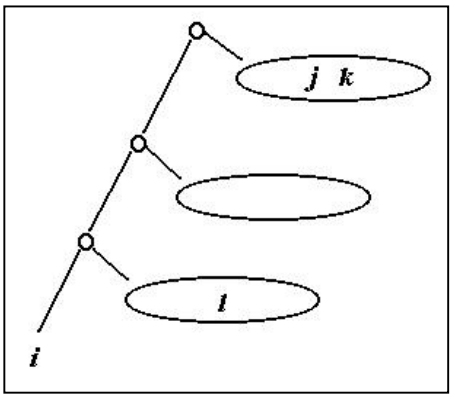
(if) Let *T* be the ultrametic tree for *M*. Then for any three leaves *i, j* and *k*, one of the following is true:

a) if all three leaves have the same lowest common ancestor (that is, $M[i,j] = M[i,k] = M[j,k]$) then the lemma holds.

b) if all three leaves do not have the same lowest common ancestor, then two of the nodes, say i and j, must have a lower common ancestor. Since the tree is ultrametric, $M[i,k] = M[j,k] > M[i,j]$, therefore the lemma holds.



(only-if) Let M be an $m \times m$ ultrametric matrix; then an ultrametric tree T can be constructed as follows. For any $1 \leq i < j \leq m$, i and j are leaves in T, and the entry $M[i,j]$ describes the internal nodes on the path (root, i) (i.e. the path from the root node to the leaf i). In addition, the relative locations of the other n-1 leaves are fixed. Therefore, the ultrametric tree T can be recursively built by constructing the branches of the path (root, i). Furthermore, for any j, k in one sub-tree, $M[i,j] = M[i,k] \geq M[j,k]$ and for any j, l in different sub-tree, $M[i,l] = M[j,l] \geq M[i,j]$; thus a valid ultrametric tree T must exist.



↵

Lemma 2.1 suggests a recursive algorithm that attempts to build an ultrametric tree *T* from a matrix *M*. If *T* can be built successfully, the algorithm accepts *M* and returns *T*. Otherwise, it rejects *M*. This algorithm has a run-time complexity of $O(m^2)$ where *m* is the number of objects.

2.2 Additive Tree

The ultrametric tree model is an idealized model. In practice, most matrices are not ultrametric. Therefore, the additive tree model [18,31,33] was proposed by [13]; this model makes weaker assumptions about distances. If a matrix *M* is ultrametric, then it also is additive. Unlike an ultrametric tree, an additive tree is an unrooted tree. In addition, the additive tree model allows internal nodes to be labeled by objects. That is, some species can be the ancestors of others. The definition of the additive tree is as follows.

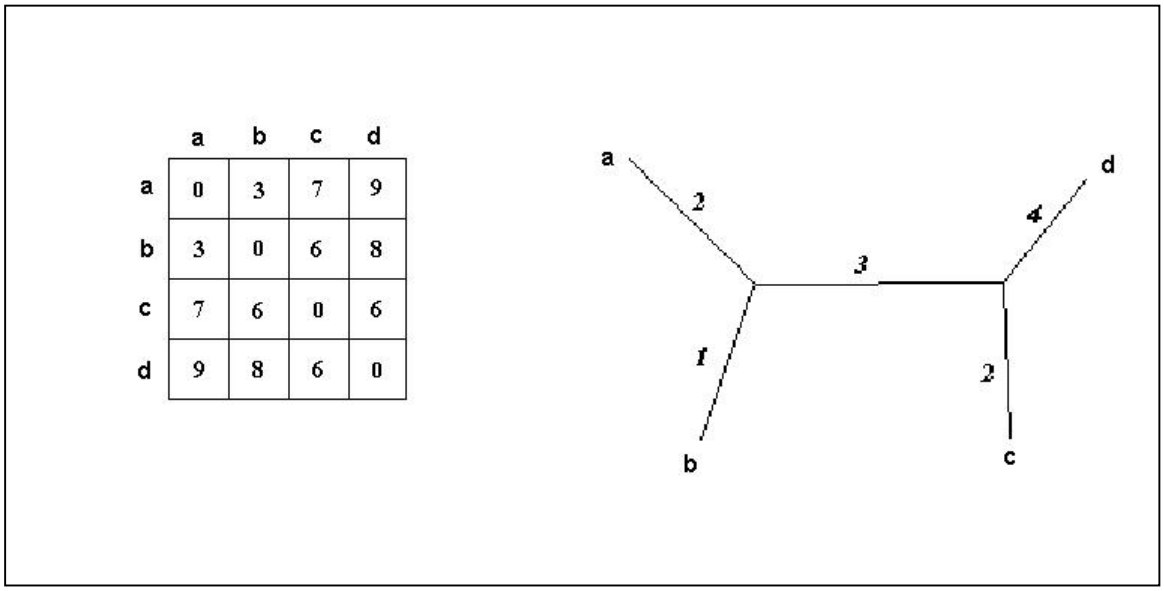


Figure 2.2: A 4×4 matrix M and its additive tree T.

Given an $m \times m$ symmetric matrix M of positive real numbers with zeros along the diagonal, an additive tree T built from M is an edge-weighted tree with at least m nodes, where exactly m distinct nodes are labeled with i ($1 \leq i \leq$ m), such that for every pair of labeled node i, j, the path (i, j) has a total weight of exactly $M[i,j]$.

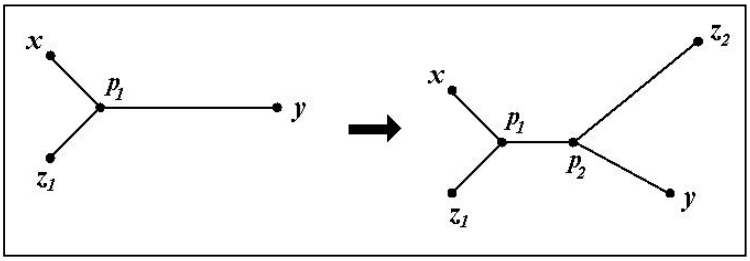
A matrix is additive if it has an additive tree. The additive tree problem is similar to the ultrametric problem, and is as follows. Given a matrix M, one wants to determine whether there is an additive tree T for M and constructs T if it exists. Figure 2.2 is an example of a matrix M and its additive tree T.

A constructive algorithm can be used to solve this problem. First, two arbitrary objects x and y are chosen to form the initial single-edged tree. Then, for $3 \leq z \leq m$, one more object z is inserted into the tree at a point p that satisfies the following linear equations.

$$M[x,y] = M[x,p] + M[y,p]$$

$$M[x,z] = M[x,p] + M[z,p]$$

$$M[y,z] = M[y,p] + M[z,p]$$



The constructive algorithm was introduced by Buneman [13] to solve the additive tree problem. Later, Jotun Hein [33] further improved the algorithm by using the deepest point algorithm in finding insertion points. In conclusion, the additive tree problem can be solved in $O(m^2)$ time. Moreover, the additive tree problem can also be solved in $O(m^2)$ by mapping it to a corresponding ultrametric tree problem [31].

2.3 Perfect Phylogeny

In perfect phylogeny [19,20,23], the phylogenetic tree is constructed according to the divergence of different characters of the objects. Specifically, there are only two states for each characters: zero, where the character is absent; and one, where it is present (i.e. $c(x) \in \{0, 1\}$).

Let M be an $m \times n$ matrix, with $m = |\mathbf{X}|$ and $n = |\mathbf{C}|$ where $\mathbf{X}$ is the set of objects and $\mathbf{C}$ is the set of characters. Each entry where $M[i,j] \in \{0, 1\}$ represents the presence of the character j in object i. A phylogenetic tree for perfect phylogeny is called a PP-tree and a PP-tree T for the matrix M is a rooted tree that satisfies:

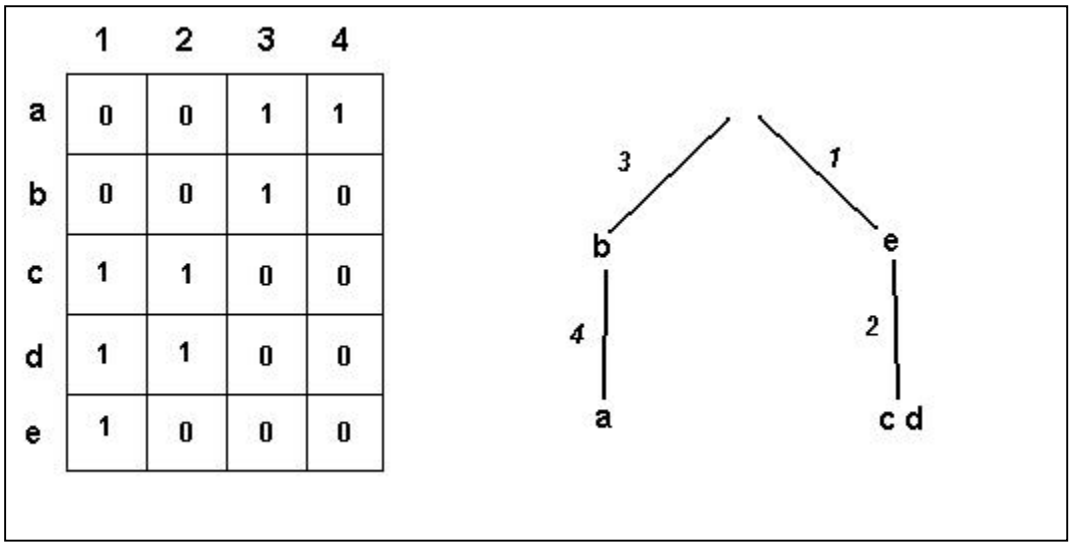


Figure 2.3.1: An example of a 5×4 {0,1} matrix *M* and its PP-tree.

1. each object labels exactly one node in *T*;

2. each edge is labeled by at least one of the *n* characters and each character labels at most one edge;

3. each internal node is either labeled by at least one object or has more than one child;

4. for any object *x* at node *i*, the path (root, *i*) defines exactly all characters *x* has.

Therefore, a matrix is called a perfect phylogeny if it has a PP-tree. The perfect phylogeny problem is as follows. Given a matrix *M*, determine whether *M* has a PP-tree *T*, and if so, construct *T*. An example of a matrix *M* and its PP-tree *T* are shown in Figure 2.3.1. Lemma 2.3 describes the perfect phylogeny property in terms of the matrix *M*.

Lemma 2.3:

Let $v_1, v_2, ..., v_n$ be the column vectors decomposed from *M*. The matrix *M* is a perfect phylogeny if and only if for every pair of column vectors v_p and v_q the set of objects that has character *p* and *q* are either disjoint or one contains the other.

Proof:

(only if) Let v_p and v_q be two column vectors such that the sets of objects that has character *p* and *q* are overlapped. Specifically, there exists three objects *x*, *y* and *z*, such that *M*[*x*,*p*] = 1, *M*[*x*,*q*] = 0, *M*[*y*,*p*] = 1, *M*[*y*,*q*] = 1, *M*[*z*,*p*] = 0 and *M*[*z*,*q*] = 1. Therefore, *z* must be an ancestor of *x*,*y*, and *x* must also be an ancestor of *y*,*z*, which is a contradiction.

(if) Let *M* be a matrix as described above; Gusfield's algorithm [30] constructs a valid PP-tree (Gusfield's algorithm is explained in detail below).

↵

Lemma 2.3 describes the condition that the perfect phylogeny property is violated. That is, there are two characters *p* and *q* and three objects *x*, *y* and *z*, such that *x* has exactly *p*, *y* has exactly *q*, while *z* has both *p* and *q*. Although Lemma 2.3 suggests an algorithm to solve the decision problem of perfect phylogeny, a straightforward implementation requires $O(mn^2)$ time. In fact, faster algorithms exist. Dan Gusfield describes an algorithm that runs in *O(mn)* time. Note that Gusfield's algorithm is asymptotically optimal in time. This statement is true because any algorithm requiring less than *O(mn)* time to solve the decision problem of perfect phylogeny can be proven invalid with a contradictory argument.

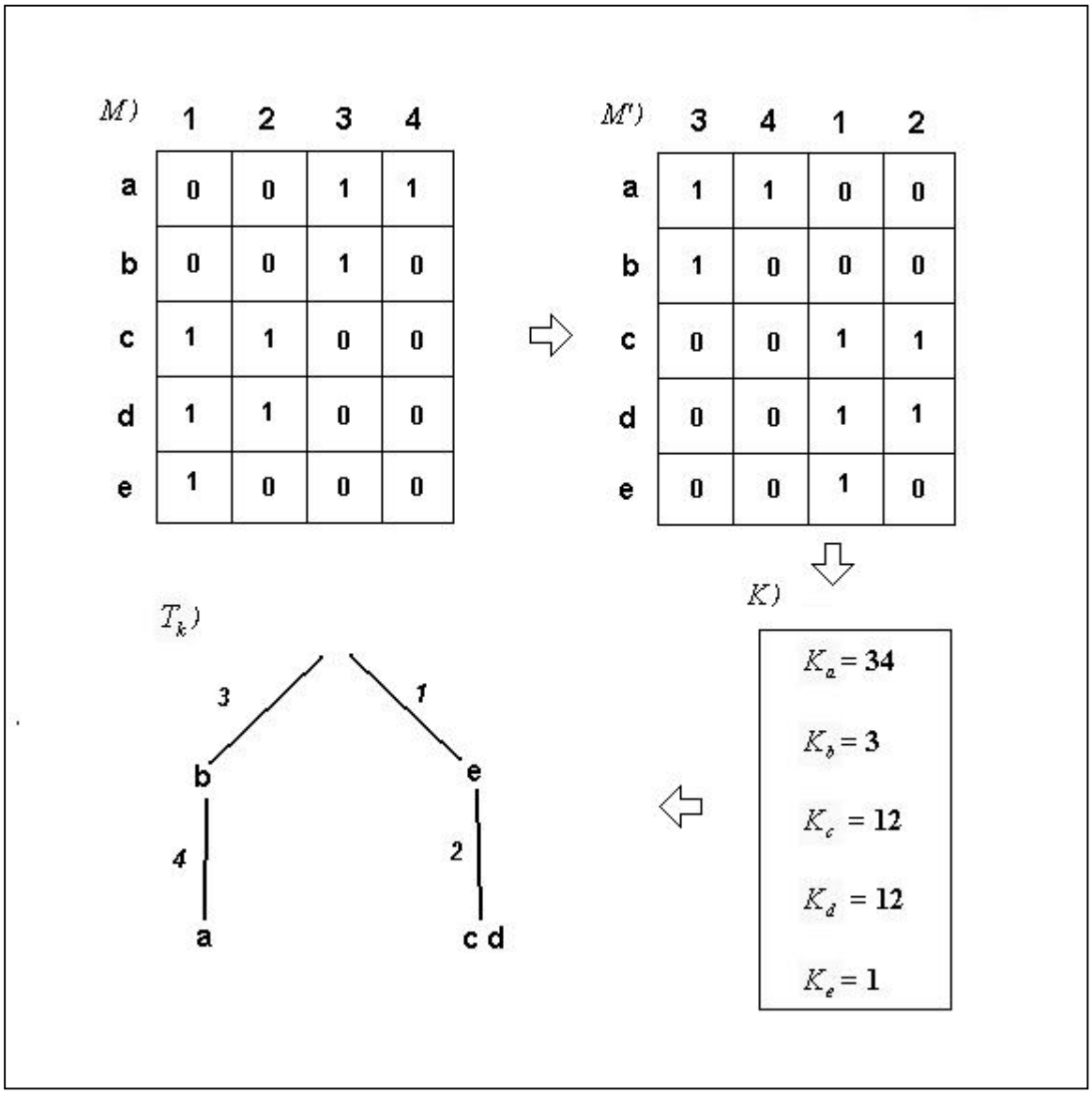


Figure 2.3.2: An example of constructing a PP-tree through the set of keywords.

Gusfield's algorithm is as follows. Since each column vector in *M* can be viewed as a binary number, one can create a matrix *M'* that is the matrix *M* in sorted descending order of the vectors. Then, define the keyword for an object *x* as the string that is comprised of all the characteristics *x* has, in left to right order of *M'*. Furthermore, define

K to be the set of keywords. Then, the PP-tree *T* can be constructed from **K** as follows. Chose the first keyword *k* from **K**, and construct a linear tree *T*. For $2 \le k \le m$, expand *T* by *k* and reject *M* if a cycle is created. Note that since *M'* is sorted, each keyword must follow the order of the current *T* (starting from the root node). In other words, *M* (and *M'*) is a perfect phylogeny if and only if *T* has no cycles. *M'* (and **K**) can be obtained in $O(mn)$ time by a radix sort. In addition, the PP-tree can be constructed from **K** also in $O(mn)$ time. Thus, Gusfield's algorithm requires $O(mn)$ time. Figure 2.3.2 shows an example of a matrix *M*, its sorted image *M'*, the set of keywords **K** and the PP-tree *T*.

2.4 Generalized perfect phylogeny

Generalized perfect phylogeny [1,2,12,29,36,37] extends perfect phylogeny by allowing for multiple state characters. Due to the generalization, a more careful interpretation of the tree is required. In the phylogenetic tree *T*, each root of a sub-tree is the ancestor of its descendent nodes, and each edge represents a state transition of a character. The transition is denoted by a pair $<c, s_i>$ where *c* is the character and s_i is the destination state. Therefore, every object is a descendent of the root (of *T*), which is defined to have zeros in all characters. The model further assumes each transition occurs uniquely. That is, each $<c, s_i>$ can at most label one edge in *T*. Therefore, the characters of a leaf *i* can be obtained by applying the transitions along the path (root, *i*). Notice this interpretation is also true for perfect phylogeny; however the interpretation in the last section is much simpler.

A phylogenetic tree for Generalized Perfect Phylogeny is called a GP-tree, and a GP-tree *T* for a matrix *M* is a rooted tree that satisfies:

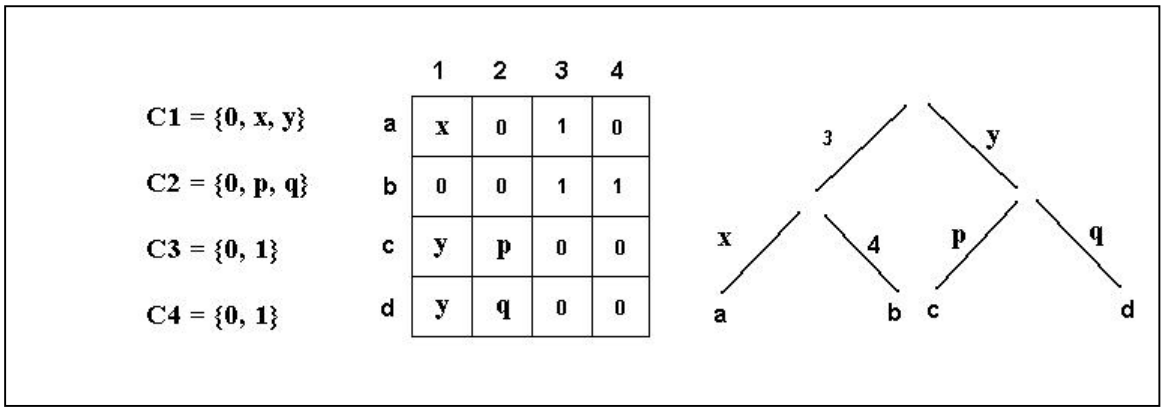


Figure 2.4: An example of generalized perfect phylogeny.

1. each object labels exactly one node in *T*;

2. each non-zero character state in *M* labels exactly one edge;

3. each internal node is either labeled by at least one object or has more than one child;

4. for any node *i*, the path (root, *i*) defines the transitions in characters of *i*.

Thus, the following is the generalized perfect phylogeny problem. Given a matrix *M*, determine whether *M* has a GP-tree *T*, and if so, construct *T*. Figure 2.4 shows an example of the matrix *M*, the characters' states and the GP-tree. This problem is indeed NP-complete in general. Kannan and Warnow [37] describe an algorithm (KW algorithm) that solves this problem when the maximum number of states *r* is fixed. The KW algorithm is based on the following definitions and lemmas.

Definition 2.4.1:

A subset **X'** $\subset$ **X** is called a cluster if for every character *c*, at most one state of *c* is shared by **X'** and **X - X'** in *M*.

Definition 2.4.2:

A cluster **X'** is called a proper cluster if there is some character *c* for which **X'** does not share any state of *c* with **X - X'** in *M*.

↵

Lemma 2.4.1:

Let *T* be a GP-tree for a matrix *M*; then for every sub-tree *T'* of *T*; if **X'** is the set of objects in *T'*, then **X'** is a proper cluster.

↵

Lemma 2.4.2:

A matrix *M* is a generalized perfect phylogeny if and only if there exists a set of proper clusters for *M*.

↵

Therefore, the KW invokes the dynamic programming technique to find the set of proper clusters for *M*. In addition, a preprocessing of sorting the proper clusters is performed to reduced the number of potential clusters. As a result, the KW algorithm requires $O(2^{2r} mn^2)$ time so that if the number of states *r* is fixed as a constant, the problem can be solved in polynomial time of *m* and *n*.

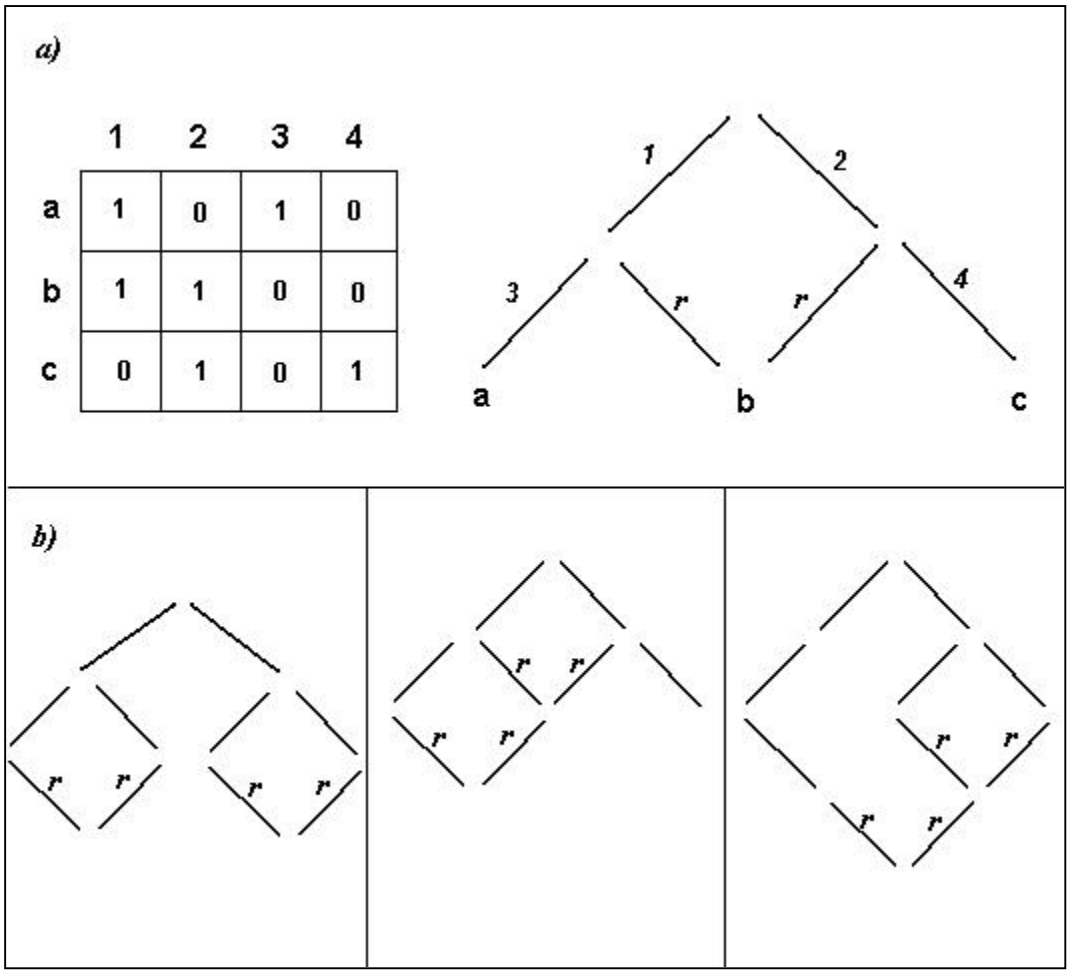


Figure 2.5: (a) An example of phylogenetic network (b) An example of disjointed recombination.

2.5 Phylogenetic Network

The phylogenetic network [56] is motivated by the recombination mechanism often found in viruses and bacteria where genetic material is transferred from one organism to another. Therefore, phylogenetic network extends perfect phylogeny to allow merging between sub-trees. The merging is formulated as a pair of special edges called recombination edges. In this model, each node in the tree either has a regular parent edge

or a pair of recombination parent edges that allow the node to inherit all the 1-state from both parents.

Therefore, the phylogenetic network problem is as follows. Given a matrix *M*, find a phylogenetic network for *M* such that it minimizes the number of recombination edges. An example of a matrix *M* that is not a perfect phylogeny but is a phylogenetic network is shown in Figure 2.5a. Figure 2.5b illustrates the disjoint structure versus the non-disjoint structure. This problem is also **NP-complete** in general. However, Wang et al. [56] propose an $O(n^3)$ algorithm for a restricted version of this problem, where all recombinations are disjoint.

Definition 2.5:

A phylogenetic network is restricted, if in any merged path of a recombination node, there is no node that is in the merged path of a different recombination node.

↵

Lemma 2.5:

Let v_p be the *p*-th column vector of *M*, and O_p is the set of objects that has character *p*. Two sets, O_p and O_q, are conflicted if their objects overlap. A matrix *M* has a restricted phylogenetic network if and only if it satisfies:

a) every O_i and O_j are not conflicted; or

b) if O_i and O_j are conflicted,

b1) for any $k \notin \{i, j\}$, if O_k and O_i are conflicted then $O_k \cap O_i = O_i \cap O_j$ and if O_k and O_j are conflicted then $O_k \cap O_j = O_i \cap O_j$.

b2) if two conflicted pairs (O_i, O_j) and $(O_{k'}, O_{k''})$ have identical intersections, i.e., $O_i \cap O_j = O_{k'} \cap O_{k''}$, then either

b2.1) $(O_i, O_{k'})$ and $(O_j, O_{k''})$ are conflicted pairs and $(O_i, O_{k''})$ and $(O_j, O_{k'})$ are not conflicted pairs; or

b2.2) $(O_i, O_{k''})$ and $(O_j, O_{k'})$ are conflicted pairs and $(O_i, O_{k'})$ and $(O_j, O_{k''})$ are not conflicted pairs.

b3) for any conflicted pair (O_k, O_l), if $O_k \cap O_l \neq O_i \cap O_j$, then either $O_k \cap O_l$ and $O_i \cap O_j$ are disjoint or one contains the other.

b3.1) $O_k \cap O_l$ and $O_i \cap O_j$ are disjoint: then there exists distinct $O_{mi,j}$ and $O_{mk,l}$, i.e. $O_{mi,j} \neq O_{mk,l}$.

b3.2) $O_k \cap O_l \supset O_i \cap O_j$: then there exists a character c_m satisfying,

b3.2.1) c_m is not conflicted with any character and

b3.2.2) $O_i \subseteq O_m$ and $O_j \subseteq O_m$ and

b3.2.3) $O_m \subseteq O_k \cap O_l$.

↵

As a result, if a matrix *M* has a restricted phylogenetic network *T*, then *T* can be constructed as follows.

1. construct PP-trees for those $O_i \cap O_j$'s not contained in any other $O_k \cap O_j$'s.

2. ignore those species used in Step 1 and repeat Step 1 for other $O_i \cap O_j$'s until no conflicted pair exists.

3. connect those PP-trees using recombination edges.

2.6 Phylogenetic Tree Models for Continuous Characters

The use of continuous characters to infer phylogenetic trees was introduced in [21,24,42]. There are three methods where the phylogenetic tree can be constructed from a set of continuous characters. The first method is to estimate the similarity distances between the objects from the set of continuous characters. One popular model of this method is the Brownian Characters Motion model [14,40,41]. The second method is to discretize the continuous characters into ordered, multiple stated characters. For example, the Gap Coding method [44,55] first converts the set of continuous characters into ordered, discrete states according to the discontinuity of the experimental data distribution. The third method is an extension of the parsimony model [31] where the tree is constructed based on the minimum transition of characters. Manhattan Metric Parsimony [22,37] is an example of models of this method.

Chapter 3 Fuzzy Phylogeny

3.1 Fuzzy Phylogeny

Although discrete stated characters provide a simple abstraction of characters, valuable information could be lost during the discretization of the experimental data. In addition, fuzzy boundaries between species and degrees of character development are indeed often found in nature [10,26,27]. Therefore, this dissertation proposes a new phylogenetic model that allows the representation of such fuzziness, called fuzzy (perfect) phylogeny. Fuzzy phylogeny is an extension of perfect phylogeny where the binary membership function [45] (i.e. $c(x) \in \{0,1\}$) is replaced by a fuzzy membership function (i.e. $c(x) \in [0,1]$). In the fuzzy membership function, $c(x) = 0$ represents the total absence of the character, $c(x) = 1$ represents the maximal expression of the character, and $c(x) = z$, for some $0 < z < 1$, represents the partial expression of the character. Furthermore, if $c(x_1) > c(x_2)$, then x_1 has a higher expression on character c than x_2. Now to distinguish the binary membership function and the fuzzy membership function, let F_c denote the fuzzy membership function that is associated with a character c. Then $Val(F_c)$ is defined to be the set of distinct values of $F_c(x)$, for $F_c(x) > 0$ and $\forall x \in X$. Let L be the set of labels c-val, such that $c \in C$ and $val \in Val(F_c)$. In other words, L is the set containing all possible labels where each label is a character c concatenated with a

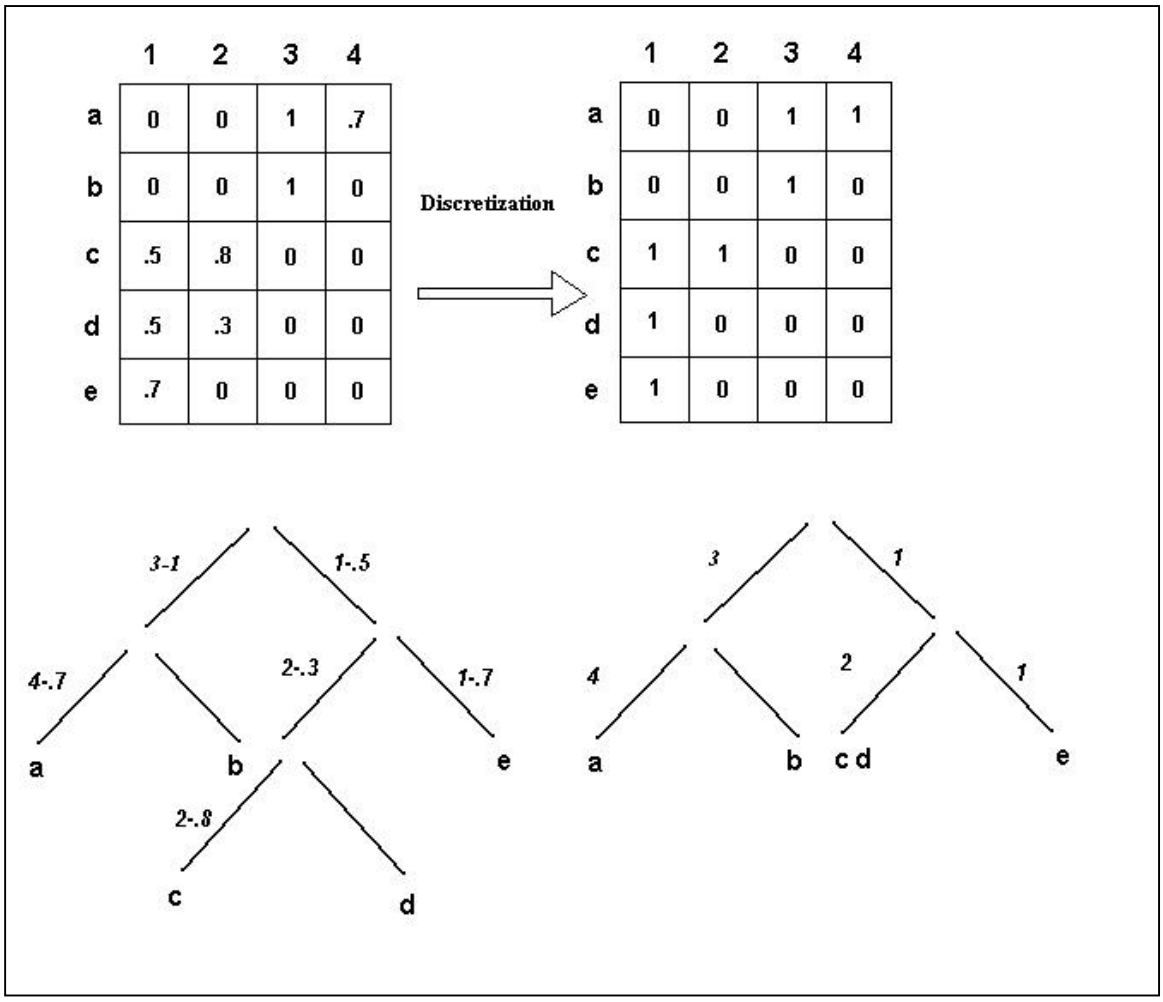


Figure 3.1: An example of a 5×4 [0,1] matrix *M* with its FP-tree and the {0,1}matrix *M'* from *M* with its PP-tree.

distinct value in $Val(F_c)$. The following is the definition of the fuzzy phylogenetic tree (FP-tree).

Given an $m \times n$ matrix M where each entry $M[i, j] \in [0, 1]$, a FP-tree of M is a rooted tree T with m' leaves (where $m' \leq m$) that satisfies the following conditions:

1. each object labels exactly one node in T;

2. each leaf node must be labeled by at least one object;

3. each label in L labels exactly one edge, and each edge has at least one label;

4. each internal node is either labeled by at least one object or has more than one child;

5. for any object *x* at node *i*, the path (root, *i*) contains labels that include but do not exceed the maximum values of $F_c(x)$, *c*;

6. for any two labeled edges e_p and e_q, if e_p and e_q are describing the same character and e_p's value < e_q's value, then e_p must be an ancestor edge of e_q;

Thus, the fuzzy phylogeny problem is that given a [0,1] matrix *M*, one needs to determine whether there is a FP-tree *T* for *M*, and if so, construct *T*. Figure 3.1 shows an example of a [0,1] matrix *M* with its FP-tree and the discretized matrix *M'* from *M* (with threshold = 0.5) with its PP-tree.

3.2 Properties of Fuzzy Phylogeny

Since fuzzy phylogeny is extended from perfect phylogeny, it is important for the following two properties to hold. First, if a matrix *M* is a perfect phylogeny, it must be a fuzzy phylogeny too. (To clarify, *M* itself is a {0,1} matrix, and there is no discretization involved.) Second, if *T* is the PP-tree for matrix *M*, then the FP-tree *T'* for *M* (by the first property, it must exists) must be the same as *T*. These two properties indeed hold in fuzzy phylogeny,

Property 1:

If a matrix *M* is a perfect phylogeny, then *M* is also a fuzzy phylogeny.

Property 2:

Let M be a {0,1} matrix that is a perfect phylogeny, then its PP-tree T is also the FP-tree for M.

Proof:

To show that Property 1 is true, it is sufficient to show that Property 2 is true. Furthermore, Property 2 is true because $\Phi(M) = M$ when M is itself a {0,1} matrix. Therefore, the PP-tree T is also the FP-tree for M.

↵

Uniqueness of the phylogenetic tree is important to a phylogenetic tree model. A phylogenetic tree model is called unique if for any input M, there is only one valid phylogenetic tree T for M. These models have an advantage of providing an unambiguous solution for each input. Note that although uniqueness is a good feature of a phylogenetic tree model, it should be the outcome of a biological theory, not the goal in designing the model. Lemma 3.2 below proves the uniqueness of the fuzzy phylogeny model.

Lemma 3.2

The fuzzy phylogeny model is unique.

Proof:

The proof approaches the lemma by looking at the uniqueness of the sub-trees of T and then by running a recursion that reduces the height of these sub-trees. For instance, T is unique if and only if i) the objects at the root node and the child edges of the root node are unique and ii) every sub-tree rooted at a child of the root node is unique (similarly for every sub-tree). Finally, a sub-tree with height zero must be unique, and that closes the proof. Assuming the converse, then there are two FP-trees T and T' for M,

where $T \neq T'$. For every internal node i in T and the corresponding internal node i' in T' (so, node i and i' are the root nodes of T and T' respectively, for the base case), then:

case 1 (the objects at i differ from the objects at i'): if there is an object x that labels i but x does not label i', then x must be at a descendent node j of i'. Let l be a label that is found on the path (i',j), then x expresses l in T', but not in T. That creates a contradiction to that T and T' are FP-tree for the same matrix.

case 2 (the objects at i the same as the objects at i'): if there exists an edge e_q that is a child edge of i but not of i', then (a) e_q is either not in T' or (b) is below another edge e_p. For (a), an object x below e_q in T has character q, but x does not have character q in T'; that is a contradiction. For (b), there must be a node x_1 that has character q but not p in T, and a node x_2 that has both character q and p in T' however, there is a node x_3 that has character p but not q in T' which must also be in T. This situation violates Lemma 2.3, so there is a contradiction.

↵

3.3 Transformation

The fuzzy phylogeny model provides an effective platform to construct the phylogenetic tree from non-binary data. In fact, its computation is theoretically equivalent to that of perfect phylogeny. Below shows how the fuzzy phylogeny problem can be transformed to the perfect phylogeny problem counterpart.

The number of distinct values d is defined as, $d = \sum_{c_k \in C} |Val(F_k)|$. There is a transformation Φ that converts an $m \times n$ matrix M to an $m \times d$ matrix M', and it is as

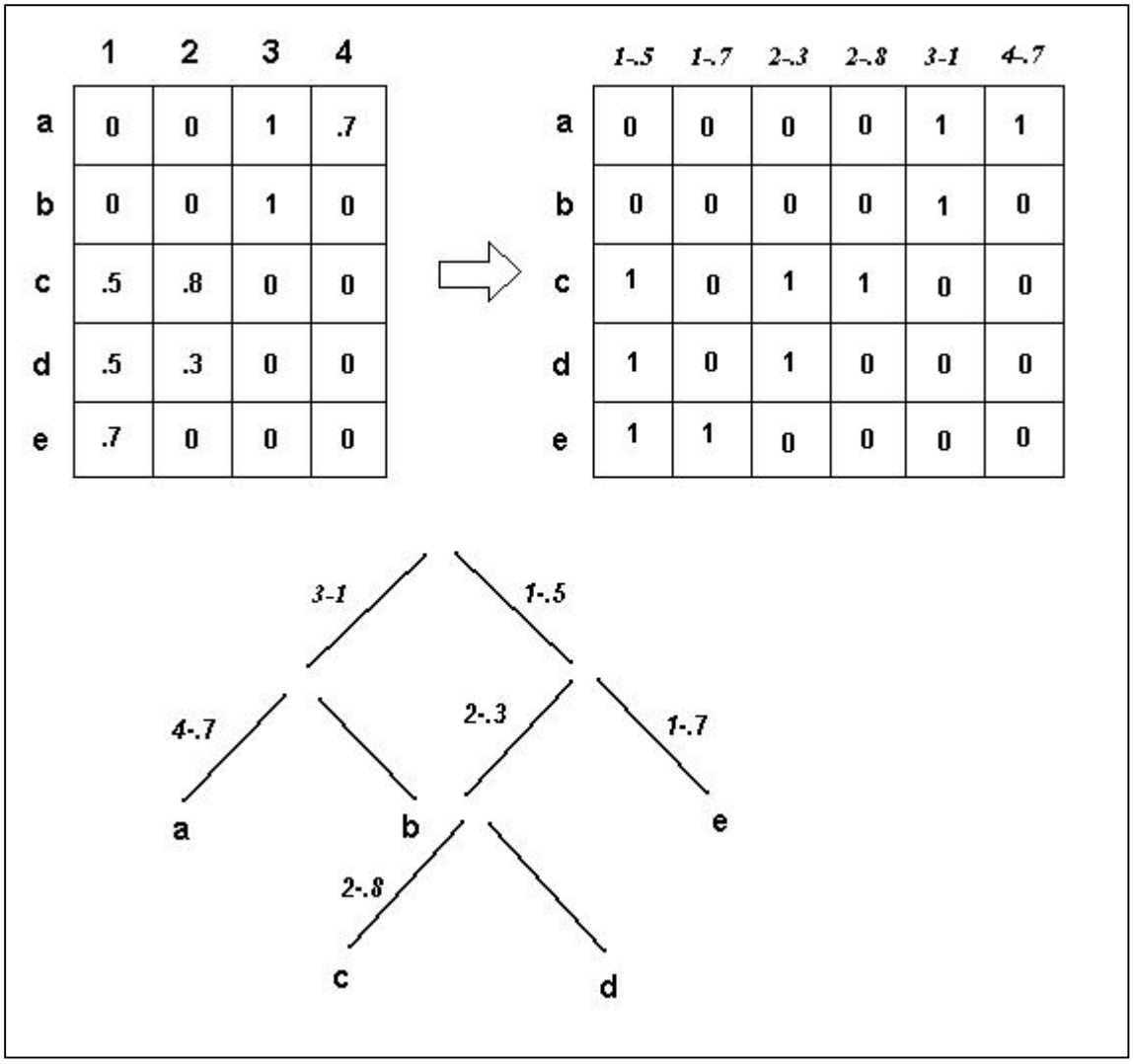


Figure 3.3: An example of a 5×4 [0,1] matrix *M*, $\Phi(M)$ and the PP-tree of

follows. Given *M*, it decomposes *M* into *n* column vectors, $v_1, v_2, ..., v_n$, each with length *m*. Then, it transforms each v_k, where $1 \le k \le n$, to an *m* by $|Val(F_k)|$ matrix M_k'. Finally it concatenates all M_k' to form *M'*. Each column *j* in M_k' corresponds to a distinct value in $Val(F_k)$ and $M_k'[i,j] = 1$ if $F_k(x_i) \ge$ the distinct value of column *j*; it is equal to zero otherwise.

The correctness of the transformation Φ to solve the fuzzy phylogeny problem can be shown with Lemma 3.3.1 and Lemma 3.3.2. Given a matrix *M*, *M'* can be created

in $O(md)$ time. Since the perfect phylogeny problem can be solved in $O(mn)$ time [30], the fuzzy phylogeny problem can then be solved in $O(md)$ time. Furthermore, because $d \leq mn$, it can be described with $O(m^2 n)$ time. An example of the transformation is shown in Figure 3.3.

Lemma 3.3.1:

Let $\Phi(M)=M'$, then M is a fuzzy phylogeny if and only if M' is a perfect phylogeny.

↵

Lemma 3.3.2:

Let $\Phi(M)=M'$; then the PP- tree T' for M' is the FP-tree T for M.

Proof:

If M' is a perfect phylogeny, the PP-tree T' is also the FP-tree for M. For any object x in a node l in T', the path (root, l) has the maximum value of each character of x. In addition, for any l, all labels that have smaller values must be in the path (root, l), and they must be in a decreasing order on the path. Conversely, the "only-if" direction is similar because T can be used for M' also.

↵

Chapter 4 The Relaxation Problems

4.1 Motivation

There are two approaches for building phylogenetic trees, namely constructive [17] and structural [31]. The constructive approach assumes the experimental data fully obey the evolutionary hypothesis and constructs the phylogenetic tree heuristically. This approach is often efficient due to its simplicity. However, a phylogenetic tree would be generated even when the experimental data is invalid (i.e. it does not obey the underlying hypothesis). On the other hand, the structural approach first requires the experimental data to obey the underlying hypothesis and rejects the data if it does not. This approach guarantees the consistency between the phylogenetic tree and the experimental data, yet it often rejects the "almost valid" data that has been perturbed by natural causes.

This dissertation focuses on the structural approach due to its theoretical importance. (Note that the constructive approach is important mainly in applications.) For the structural approach, an "add-on" relaxation is often introduced to repair the data that is perturbed by natural causes. In this chapter, two relaxation problems for perfect phylogeny are presented, namely the Largest Compatible Subset problem [6,7,16] (the LCS problem) and the Adjustment problem for Perfect Phylogeny (the APP problem). After that, the Adjustment problem for Fuzzy Phylogeny (the AFP problem) is proposed,

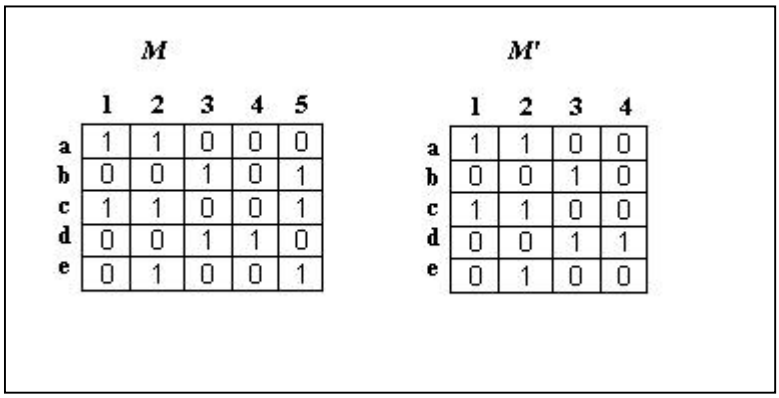


Figure 4.2: An example of a LCS problem instant *M* and a possible solution *M'*.

which is the target problem for the dissertation. Finally, the complexities of the relaxation problems are investigated.

4.2 The Largest Compatible Subset Problem

Recall that the perfect phylogeny problem is that given a matrix *M*, determine whether *M* is a perfect phylogeny (i.e. *M* has a PP-tree). The Largest Compatible Subset [6,7,16] (LCS) problem is a relaxation problem that can be added on to the perfect phylogeny problem. The goal of the LCS problem is to discard the minimum number of characters such that the remaining data is a perfect phylogeny. The formal definition of the LCS problem is as follows. Given an $m \times n$ {0,1} matrix *M*, find an $m \times n'$ matrix *M'* by removing columns from *M*, such that *n'* is maximized and *M'* is a perfect phylogeny. Notice that for any matrix *M*, there exists an *M'* with $n' \geq 1$. Therefore, a variation of the problem may also require $n' \geq k$ for some predefined *k*. Figure 4.2 is an example of *M* and a possible *M'*.

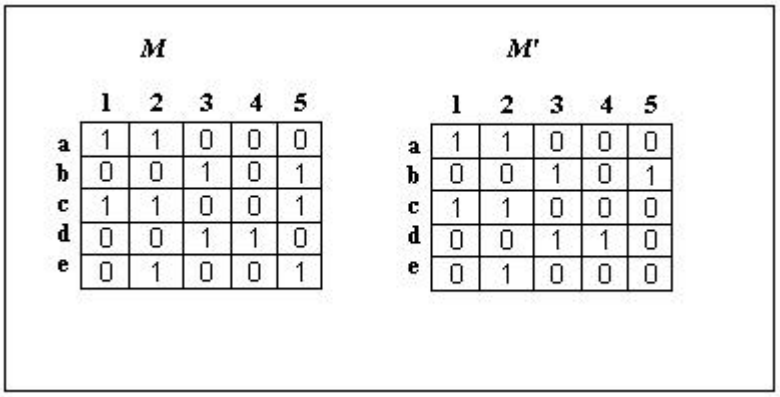


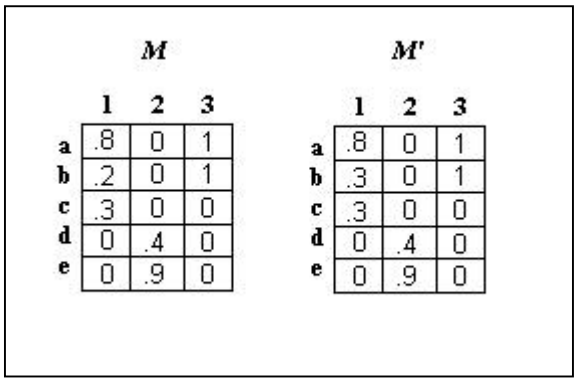
Figure 4.3: An example of an APP problem instant *M* and a possible solution *M'*.

4.3 The Adjustment Problem for Perfect Phylogeny

The Adjustment problem for Perfect Phylogeny (APP problem) is another relaxation problem for perfect phylogeny. Instead of discarding inconsistent characters, the APP problem repairs the inconsistent entries in the matrix (i.e $M[i,j]$). The formal definition of the APP problem is as follows. Given an $m \times n$ {0,1} matrix *M*, find an $m \times n$ matrix *M'* by flipping entries of *M* (i.e. $0 \rightarrow 1$ or $1 \rightarrow 0$), such that the number of flips *f* is minimized and *M'* is a perfect phylogeny. Again, for any matrix *M*, there exists an *M'* since one can turn *M* into any $m \times n$ {0,1} matrix. Therefore, a variation of the problem may limit $f \leq k$, for some predefined *k*. Figure 4.3 is an example of *M* and a possible *M'*.

4.4 The Adjustment Problem for Fuzzy Phylogeny

The Adjustment problem for the Fuzzy Phylogeny (AFP problem) is a proposed relaxation problem for fuzzy phylogeny. In addition, this is the target problem of this dissertation. This problem is an extension of the APP problem. Therefore, the AFP problem is to repair the inconsistent entries in the [0,1] matrix. However, the adjustment



4.4 An example of an optimal AFP problem instant *M* and a possible solution *M'*.

is no longer in binary (i.e. 0→1 or 1→0); rather it is a mapping from [0,1]→[0,1]. The formal definition is as follows. Given an $m \times n$ [0,1] matrix *M*, find an $m \times n$ [0,1] matrix *M'* such that the total adjustment $a = \sum_{i=1}^{m}\sum_{j=1}^{n} | M[i, j] - M'[i, j] |$ is minimized and *M'* is a fuzzy phylogeny.

Similarly, for any matrix *M*, there exists an *M'* since one can turn *M* into any $m \times n$ [0,1] matrix. Therefore, in this dissertation, two variations of the AFP problem are studied. They will be called the AFP problem and the optimal AFP problem in the following text. Figure 4.4 is an example of *M* and a possible *M'* for the optimal AFP problem with $r = 0.1$.

Definition of the AFP problem:

Given an $m \times n$ [0,1] matrix *M* and a value $r \in$ [0,1], find a matrix *M'* such that $|M[i,j]-M'[i,j]| \leq r$ and $M'[i,j] \in$ [0,1], for $1 \leq i \leq m$ and $1 \leq j \leq n$, and *M'* is a fuzzy phylogeny. (Note that the value $2r$ is called the adjustable range.)

↵

Definition of the optimal AFP problem:

Given an $m \times n$ [0,1] matrix M and a value $r \in [0,1]$, find a matrix M' such that $|M[i,j]-M'[i,j]| \leq r$ and $M'[i,j] \in [0,1]$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, M' is a fuzzy phylogeny, and the total adjustment a is minimized.

↵

4.5 The Complexities

The complexity of a problem often plays an important role in an algorithm's development [15,34]. For instance, if problem **A** is an **NP** problem, assuming that **N** ≠ **NP**, then it is unwise to search for a polynomial algorithm that solves **A** exactly. As a result, alternatives might be investigated, such as heuristic algorithms or fast implementations, to solve small problem instants. In this section, the complexities of the three relaxation problems, the LCS problem, the APP problem and the optimal AFP problem, are studied.

In order to investigate the complexities of the relaxation problems, it is critical to understand the transformation technique. The transformation technique provides an easy way to show that a problem is in a complexity class **C**. Let problem **B** be a **C** problem where **C** is a complexity class. Then, problem **A** is also a **C** problem if there exists a pair of transformations, $\alpha()$ $\beta()$, such that $\alpha(b)$ transforms b (an instant of **B**) to a (an instant of **A**) and $\beta(s)$ transforms s (a solution for a) to t (a solution for b). Furthermore, $\alpha()$ and $\beta()$ are both below **C**.

The transformation technique is used to investigate the complexities of the three relaxation problems. Specifically, the question is whether the problems are in **P** or **NP** (again, assuming that **N** ≠ **NP**). Therefore, the following theorem (Theorem 4.5) is used.

Theorem 4.5:

Let problem **B** be an **NP-hard** problem; then problem **A** is also an **NP-hard** problem if there exists a pair of polynomial time transformations, $\alpha()$ $\beta()$, such that $\alpha(b)$ transforms b (an instant of **B**) to a (an instant of **A**) and $\beta(s)$ transforms s (a solution for a) to t (a solution for b).

↵

4.6 The Complexity of the LCS problem

Vertex cover [15,32,45] is a classic problem in graph theory [17,57]. A vertex cover of a graph $G=(V, E)$ is a set $V' \subseteq V$ such that if (u, v) is an edge of G, then either $u \in V'$ or $v \in V'$ (or both). The vertex cover problem (VC problem) is to find a vertex cover that minimizes $|V'|$. In addition, the vertex cover problem is proven to be an **NP-hard** problem.

In order to show the NP-hardness of the LCS problem, a variation of the VC problem is used. Instead of finding a minimized $|V'|$, this variation is to find a maximized $|U'|$ such that if (u, v) is an edge of G, then either $u \notin U'$ or $v \notin U'$ (or both). This variation is equivalent to the original VC problem because $U' = V - V'$. Therefore, this variation is now referred as the VC problem in the following text.

With Lemma 2.3, the $\alpha()$ transformation can be as follows. Given a graph $G=(V,E)$, a $\{0,1\}$ matrix M is created so that for every vertex $u \in V$, there is a corresponding column u' in M and two columns u' and v' have overlapping objects if and only if the two corresponding vertices u and v are adjacent (i.e. $(u,v) \in E$). Therefore, the $\beta()$ transformation is as follows. Let M' be a solution to a LCS problem instant $M = \alpha(G)$. Then, the solution to the VC problem instant G is the set $U' = \{u \in V \mid u' \text{ is in } M'\}$. As a result, Theorem 4.6 is true since $\alpha()$ and $\beta()$ are both within polynomial time.

Theorem 4.6:

The LCS problem is an **NP-hard** problem.

↵

4.7 The Complexity of the APP problem

The NP-hardness of the APP problem can be shown in a similar manner, and the $\alpha()$ transformation is as follows. Given a graph $G=(V,E)$, a $2(n^2-n)+1 \times n$ $\{0,1\}$ matrix M is created where $n = |V|$. Each of the first n^2-n rows is associated with an ordered pair (u,v) of V (note: one row for (u,v) and one row for (v,u)). Furthermore, $M[\text{index}(u,v),u] = 1$ if $(u,v) \in E$ and $M[\text{index}(u,v),w] = 0$, $\forall w \neq u$, where index(u,v) is the row number that is associated with the order pair (u,v). Next, the first n^2-n rows are repeated. Finally, $M[\,2(n^2-n)+1, w] = 1$, for $1 \leq w \leq n$.

As a result, the $\beta()$ transformation is as follows. Let M' be a solution to an APP problem instant $M = \alpha(G)$. The maximum vertex cover for G is the set $U' = \{u \in V \mid$

$M[\,2(n^2-n)+1,u] = 1\}$. Therefore, Theorem 4.7 is true since $\alpha()$ and $\beta()$ are both within polynomial time.

Theorem 4.7:

The APP problem is an **NP-hard** problem.

↵

4.8 The Complexity of the optimal AFP problem

Due to Theorem 4.7, it can be shown that the optimal AFP problem is also an **NP-hard** problem. To show that the optimal AFP problem is **NP-hard**, it is sufficient to show that a subset of the optimal AFP problem is **NP-hard**. Specially, let **sub-AFP** be the set of the optimal AFP problem instants where M is a {0,1} matrix and $r = 1$. Lemma 4.8 shows that if Ψ is an algorithm that solves the APP problem, then Ψ also solves the problem instant of **sub-AFP**. Therefore, the optimal AFP problem is an **NP-hard** problem.

Lemma 4.8:

For any problem instant $M \in$ **sub-AFP**, if there exists a solution M', then there is a {0,1} matrix M'' such that M'' is another solution for M and $a' \geq a''$ where a' is the total adjustment of M' and a'' is the total adjustment of M''.

Proof:

Case 1: if M' is a {0,1} matrix, then $M'' = M'$ and $a' = a''$.

Case 2: if *M'* is not a {0,1} matrix, then there is a {0,1} matrix *M''* such that for every two objects *i* and *j*,

1. if $i > j$ in *u'* (in *M'*), then $i \geq j$ in *u''* (in *M''*);

2. if $i = j$ in *u'*, then $i = j$ in *u''* ;

3. if $i < j$ in *u'*, then $i \leq j$ in *u''*.

In addition, $a' \geq a''$ and *M''* is a solution to *M* (i.e. *M''* is a fuzzy phylogeny and $|M[i,j]-M''[i,j]| \leq r$, but since $r = 1$, it is true for every *M''*). Let $u_i = M[i,u]$, $u'_i = M'[i,u']$ and $u''_i = M''[i,u'']$. Then, $a' = \sum_{u \in M} \sum_{i=1}^{m} |u_i - u'_i|$ and $a'' = \sum_{u \in M} \sum_{i=1}^{m} |u_i - u''_i|$. Then *M''* is created as follows. Let $k \in \{u'_i\}$ for $1 \leq i \leq m$; if $u_i < k$, then $u''_i = 0$, else $u''_i = 1$, so that the total adjustment of *u* is minimized. Without loss of generality, assume $u_i \leq u_j$ if $i < j$. Also, assume that $1 \leq x \leq k \leq y \leq m$ for some *x* and *y*, $u_i = 0$ where $1 \leq i < x$, $u_i = 1$ where $x \leq i < k$, $u_i = 0$ where $k \leq i < y$, $u_i = 1$ where $y \leq i \leq m$. Then the total adjustment of *u'* is $\sum_{i=1}^{m} |u_i - u'_i| = \sum_{i=1}^{x-1} u'_i + \sum_{i=x}^{k-1} 1 - u'_i + \sum_{i=k}^{y-1} u'_i + \sum_{i=y}^{m} 1 - u'_i$ while the total adjustment of *u''* is $k - 1 - i + j - 1 - k = j - i - 2$. As a result, the total adjustment of *u''* ≤ the total adjustment of *u'*; otherwise there must exist a $k' \in \{u'_i\}$ that yields a total adjustment less than that of *k*, which is a contradiction.

Finally, due to the transformation Φ and Lemma 2.3, it can be shown that *M''* is a fuzzy phylogeny, and that closes the proof.

↵

Theorem 4.8:

The optimal AFP problem is an **NP-hard** problem.

↵

Chapter 5 A Brute-force Approach

5.1 The Search Space

Recall from Chapter 3 that given an $m \times n$ matrix M, where each element $M[i,j]$ is in the interval [0,1], one can verify whether M is a fuzzy phylogeny through the transformation $\mathbf{\Phi}$. Let ϕ be a function that verifies whether a matrix is a fuzzy phylogeny (i.e. $\phi(M)$={yes, no}). Then, to solve the (optimal) AFP problems, a naïve brute-force algorithm invokes ϕ on each $\bar{M}$, where $\bar{M}$ is a matrix that is within the adjustable range $2r$ of M (i.e. $|M[i,j]-\bar{M}[i,j]| \le r, \forall\ i,j,\ 1 \le i \le m,\ 1 \le j \le n$). However, this algorithm may never terminate because the number of possible $\bar{M}$ is infinite (except for the trivial cases). However, it is not difficult to see that the infinite set of $\bar{M}$ (i.e. $S_{\bar{M}} = \{\bar{M} \mid |M[i,j]-\bar{M}[i,j]| \le r, \forall\ i,j,\ 1 \le i \le m,\ 1 \le j \le n\}$) can be mapped to a finite set of $\mathbf{\Phi}(\bar{M})$ images (i.e. $S_{\mathbf{\Phi}(\bar{M})} = \{\mathbf{\Phi}(\bar{M}) \mid \bar{M} \in S_{\bar{M}}\}$). In other words, $\mathbf{\Phi}$ partitions $S_{\bar{M}}$ into equivalent classes, each of which can be represented by $\mathbf{\Phi}(\bar{M})$, where $\bar{M}$ is any member of the equivalent class. Furthermore, $I = \mathbf{\Phi}(\bar{M})$ is called the image of $\bar{M}$ (also $S_I = S_{\mathbf{\Phi}(\bar{M})}$). Figure 5.1.1 depicts the concept of the mapping. Therefore, the transformation $\mathbf{\Phi}$ allows brute-force

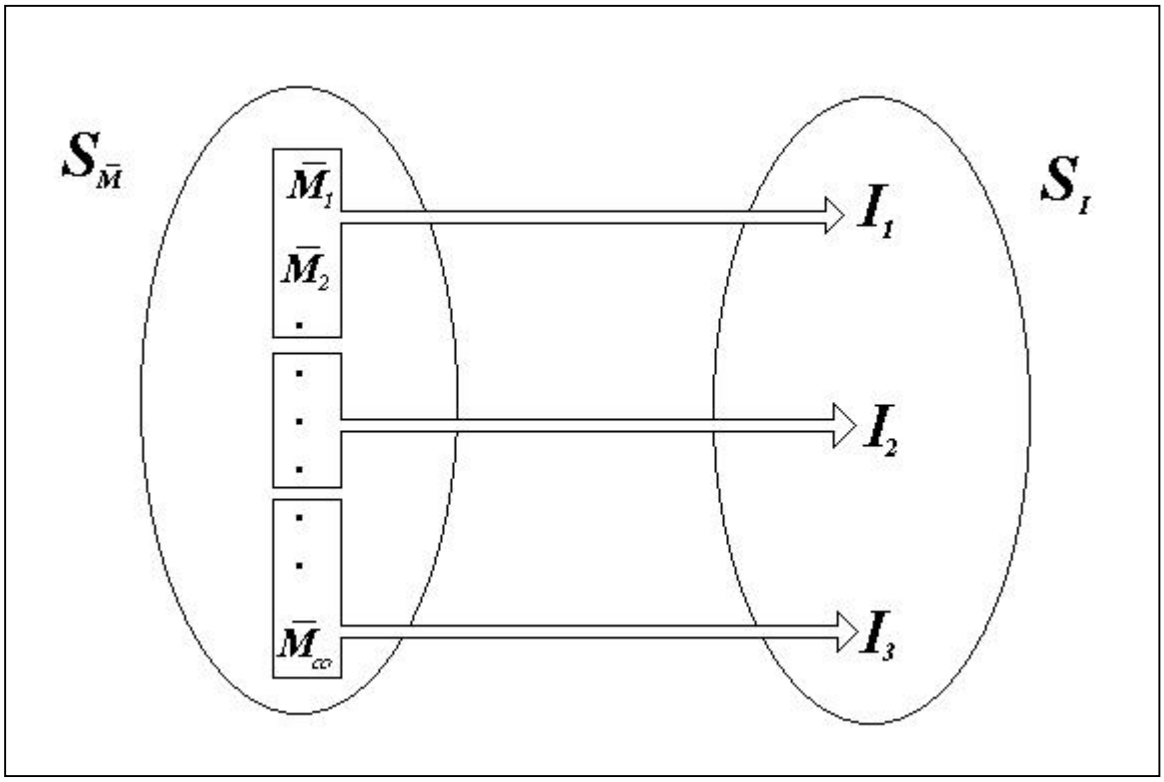


Figure 5.1.1: The concept of mapping the infinite set $S_{\bar{M}}$ to the finite set S_I.

algorithms to search the finite set S_I, instead of the infinite set $S_{\bar{M}}$. As a result, this brute-force approach is guaranteed to terminate and produce the correct answer.

It is important to understand the relationship between the infinite set $S_{\bar{M}}$ and the finite set S_I. In fact, *I* only depends on the orders of the values in $\bar{M}$. Figure 5.1.2 gives an example of three $\bar{M}$ that yield the same *I*. Figure 5.1.2 also shows the orders of each character of $\bar{M}$. In other words, for each column (that specifies a character) in $\bar{M}$, the validity depends on the relative positions of the objects on the development path. In the discussion below, the term *order* is used to mean the order of how the objects lay on the

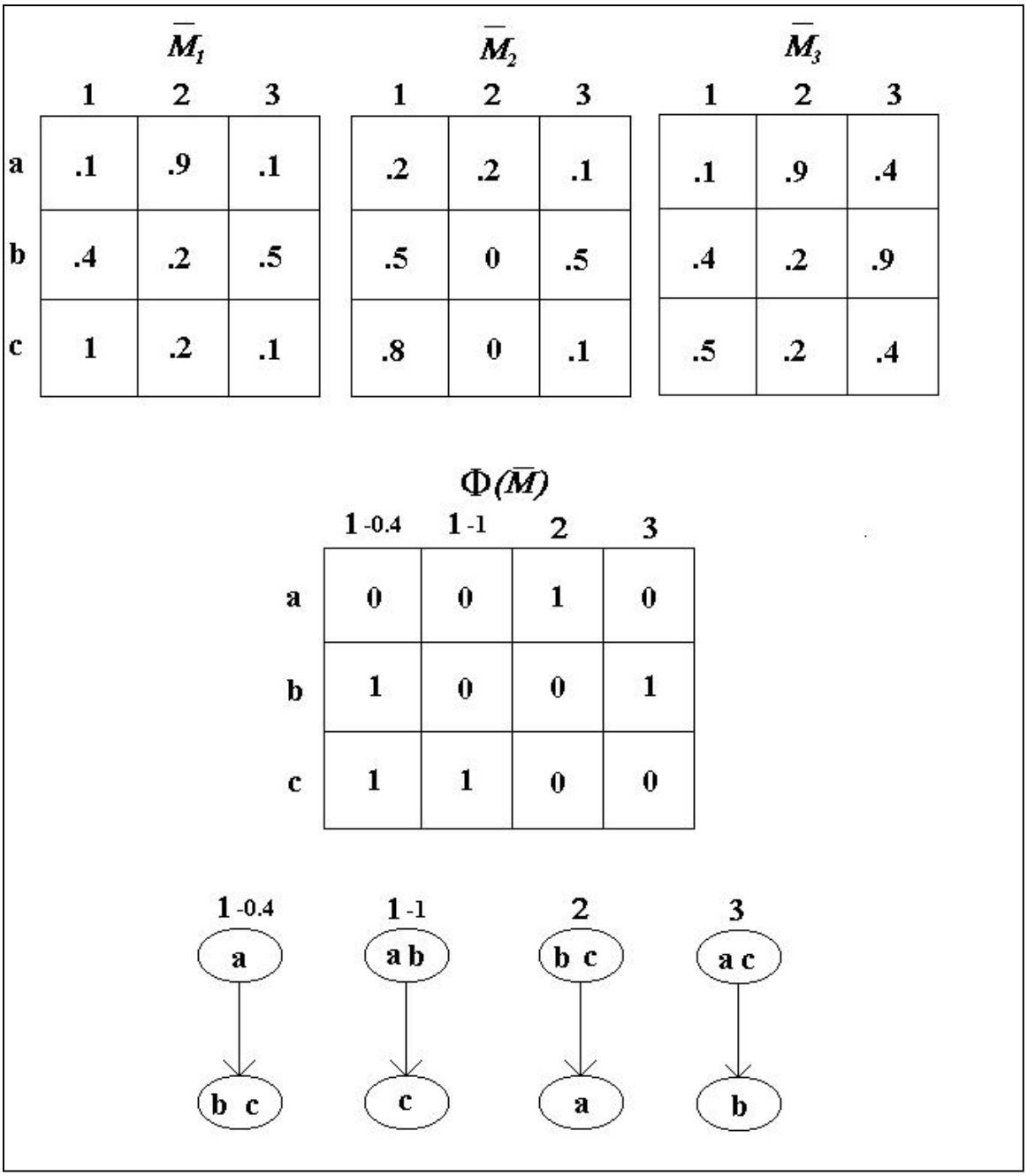


Figure 5.1.2: Three matrices $\bar{M}$, their image *I* and the orders of the characters.

path. In addition, each group in the order is called a *rank*. Therefore, the order of character 3 in Figure 5.1.2 has two ranks, where *a, c* are at rank 1 and *b* is at rank 2.

Therefore, to solve the AFP problem, one invokes ϕ on each I and quits when one valid solution (i.e. an I that is a perfect phylogeny) is found. In addition, one also needs to map the image I back to a [0,1] matrix. To solve the optimal AFP problem, however, one needs to search the entire S_I set by brute-force to find an I that yields the minimum adjusted M'. (See Section 4.4 for the definitions of the problems.)

5.2 The Brute-force Algorithm

The key step of this brute-force method is to enumerate the elements in S_I and to find the [0,1] matrix M' that yields a given I (it is called the reverse mapping). This chapter describes the algorithms for the enumeration and the reverse mapping, when M is an $m \times 1$ matrix (i.e. it is a vector). Then, the enumeration of an arbitrary size M can be obtained by combining results from the columns of M. This algorithm is called the (Brute-Force) BF-algorithm.

In order to enumerate the elements in S_I, a data structure is needed. Let L be a sequence of sets, where each element of L has an associated value. In addition, each member of the set is a 2-tuple <*label*, *count*>, such that *label* is one of the objects' names and *count* is the objects' adjustable range, in terms of the number of elements in the sequence.

The mechanism to translate a matrix M with adjustable range $2r$ to the data structure L is as follows. First, the set of threshold values *Thres* is computed. *Thres* is a finite set of values that is sufficient to yield all possible orders. For each value in M, it produces three threshold values, M, M-r and M+r. However, for any threshold value that is greater than one, it is replaced by one, and similarly if it is less than zero, it is then

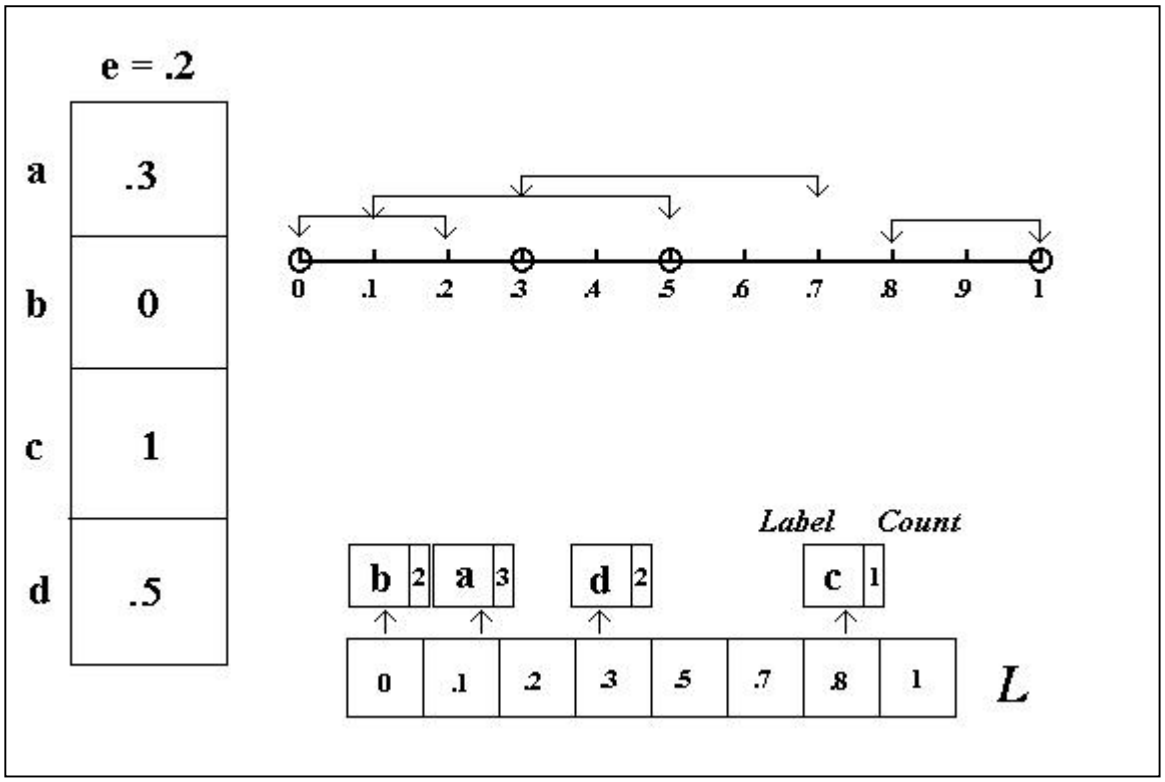


Figure 5.2: An example of a 4×1 matrix M with adjustable range $2r = 0.4$ and

replaced by zero. (The reason that above values are chosen this way will be revealed later by Lemma 5.4.1.) Therefore, *Thres* is the union of these three threshold values of each value in *M*. Then, the length of *L* is |*Thres*|; that is the number of distinct threshold values. In addition, the associated values of *L* are the distinct threshold values arranged in ascending order. Finally, for each object *x*, a 2-tuple is created and placed in the set at where the element (of *L*) is associated with *x*'s leftmost value (i.e. *Max*(0,*M*[*i*]-*r*)), and *count* is the distance from its rightmost value (i.e. *Min*(*M*[*i*]+*r*,1)) in terms of the number of elements in the sequence. Figure 5.2 shows an example of a matrix *M*, an adjustable range 2*r* and the corresponding *L*. Moreover, each variation of this *L* is called a configuration, and the *L* just described is called the initial configuration.

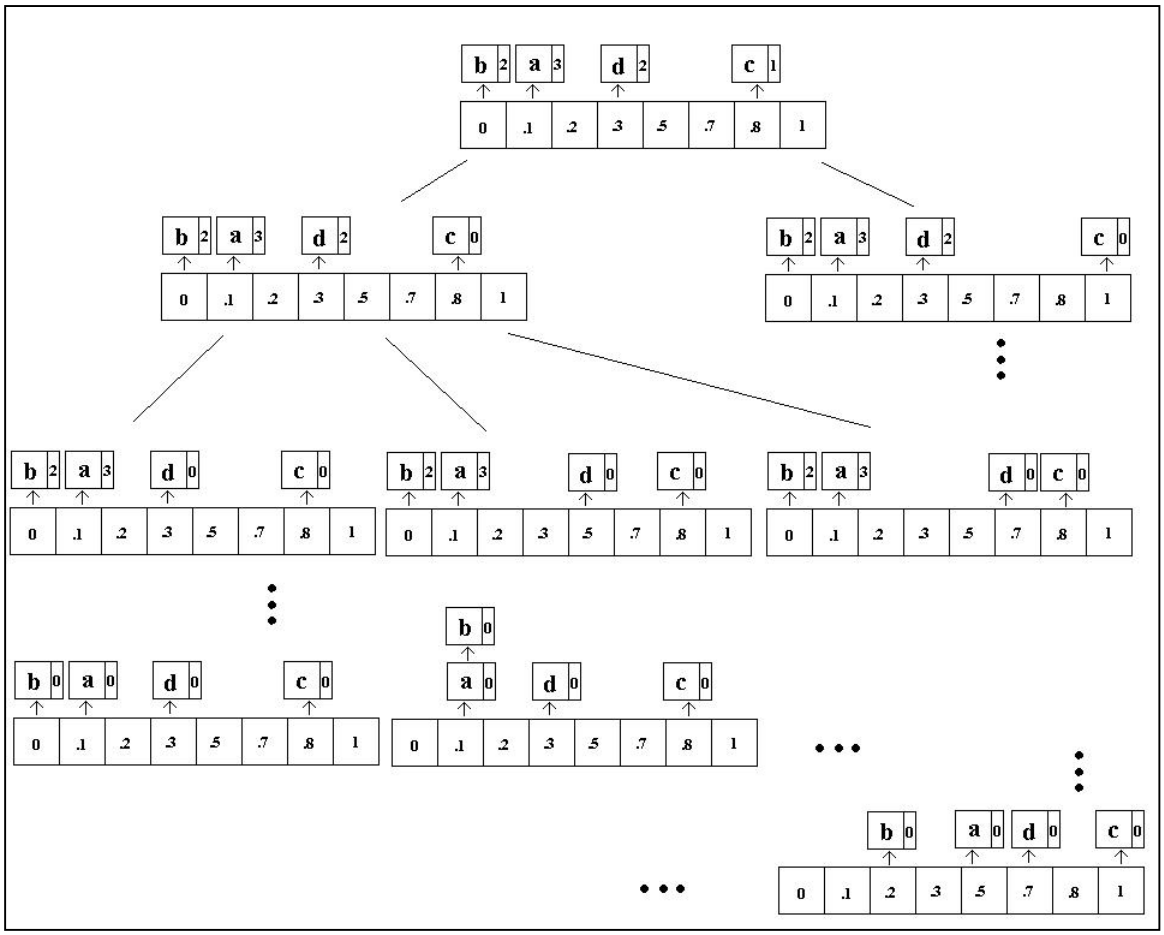


Figure 5.3: The enumerated tree of the example in Figure 5.2.

5.3 The Enumerated Tree

The goal of the enumeration is to generate all possible configurations. It is done in a recursive fashion. A configuration that has at least one 2-tuple which its *count*>0 is called non-fixed (it is called fixed otherwise). A non-fixed configuration that has *n* non-fixed 2-tuple can be expanded to configurations that have *n*-1 non-fixed 2-tuple by fixing one 2-tuple at each possible element of *L*. Figure 5.3 depicts the configuration expansion as an enumerated tree. Actually, the tree is similar to a search tree. The root of the tree is the initial configuration of *L*. The children of an internal node are the different

configurations that are expanded from this internal node. Each level down the tree, the child fixes one more 2-tuple. Therefore, all fixed configurations are at the leaf of the tree and are used to define possible orders.

5.4 Correctness

Now, one needs to show the BF-algorithm indeed generates all possible images *I* and correctly retrieves an *M'* from the image *I*. The enumerated tree has already shown how all possible configurations are generated. Therefore, the question remaining is whether the definition of the data structure *L* correctly captures all possible orders of *M* with the adjustable range 2*r*. Lemma 5.4.1 shows that for any two arbitrary values in *M*, *L* captures all possible orders. As a result, one can show that all possible orders can indeed be generated in general by applying Lemma 5.4.1 recursively.

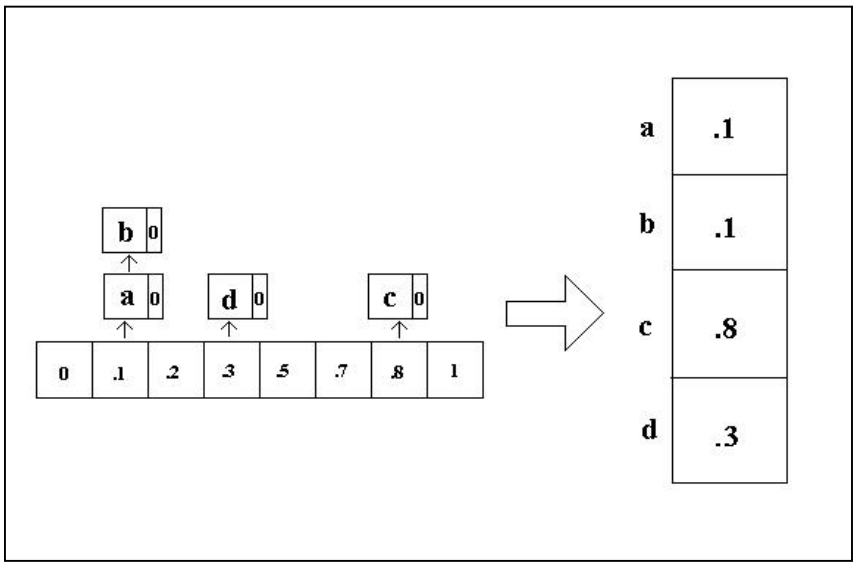
Lemma 5.4.1:

Let *a* and *b* be two values in *M*; *L* captures all possible orders of *a* and *b*.

Proof:

Without loss of generality, we first assume that $a \leq b$. Then, there are three cases of how the two adjustable intervals are arranged (i.e. the intervals $[Max(a-r,0),Min(a+r,1)]$ and $[Max(b-r,0),Min(b+r,1)]$. To complete the proof, one configuration for each order in each case is provided.

Case 1 (disjoint): If the two intervals are disjoint, there is only one order (i.e. $a \rightarrow b$). The initial configuration of *a* and *b* yields this order.

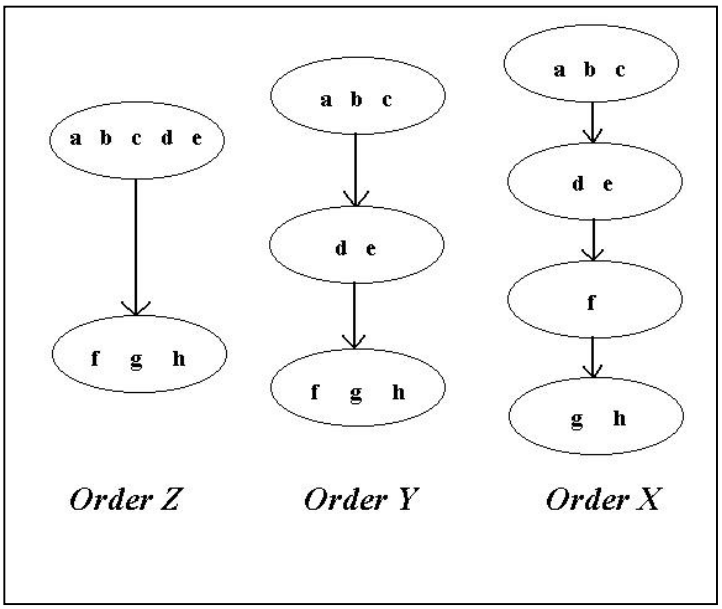


5.4.1 An example of an adjusted matrix $\bar{M}$ from a fixed-configuration by the reverse mapping.

Case 2 (uniquely intersected): If the two intervals have a unique intersection, there are two possible orders (i.e. $a \rightarrow b$ and ab). The initial configuration of a and b yields $a \rightarrow b$ and the configuration that contains $a+r$ and $b-r$ yields ab, since that is the uniquely intersection of the two intervals.

Case 3 (overlapped): If the two intervals are overlapped, there are three orders (i.e. $a \rightarrow b$, $b \rightarrow a$ and ab). The configuration that contains $a-r$ and $b+r$ yields $a \rightarrow b$; and the configuration that contains $b-r$ and $a+r$ yields $b \rightarrow a$. For the order ab, if $a=b$, then the initial configuration yields this order. Otherwise, if $a<b$, then $a+r=b+r'$, for some $-r \leq r' \leq r$, since the two intervals are overlapped.

↵



5.4.2 Order *X* is a refinement of order *Y* and order *Y* is a refinement of order *Z*.

After an image *I* is enumerated in the form of a configuration, one needs to find the corresponding adjusted matrix $\bar{M}$ (i.e. $\Phi(\bar{M}) = I$). The reverse mapping retrieves $\bar{M}$ as follows. $\bar{M}[i]$ is the associated value of the set of where object *i* is located. Figure 5.4.1 gives an example of a fixed configuration and its $\bar{M}$. However, the next question is whether the *M'* produced by the reverse mapping minimizes the total adjustment (i.e. $\sum_{i=0}^{m} |M[i] - \bar{M}[i]|$).

Before discussing the reverse mapping, the concept of refinement should be discussed. An order *X* is called a refinement of an order *Y*, if every rank of *X* is a subset of some rank of *Y*, and for any two objects *a* and *b*, if $a \rightarrow b$ in *Y*, then $a \rightarrow b$ in *X*. (Note that any order is a refinement of itself.) Figure 5.4.2 illustrates the concept of

refinements. The refinement property is as follows. Let *W* and *V* be columns with order *X* and *Y* respectively, where *X* is a refinement of *Y*. If *M* is a matrix that contains column *W* and *M* is a fuzzy phylogeny, then when substituting *W* by *V*, *M* is still a fuzzy phylogeny.

The refinement property assures that the order *X* can be replaced by an order *Y*, for some *Y* in the enumerated tree. The next property, the minimum property (Lemma 5.4.2), says that the total adjustment of *Y* from the reverse mapping is the same as the minimum adjustment of *X*. Therefore, the enumeration and the reverse mapping together guarantee that the BF-algorithm finds the correct answer for the optimal AFP problem (and the AFP problem).

Lemma 5.4.2:

Given an enumerated tree, if *X* is an order (represented by a configuration) in this tree, then there exists an order *Y* also in this tree, such that *X* is a refinement of *Y* and *min(X)=reverse(Y),* where *min(X)* is the minimum adjustment of *X* and *reverse(Y)* is the total adjustment of *Y* from the reverse mapping.

Proof:

The proof is done by an induction on the number of objects in *M*. When there is one object, then there is only one order, and the initial configuration requires zero adjustment. Therefore, *min(X)=reverse(Y)* holds. Assuming *n*-1 objects are on threshold values and their orders are obeyed, the *n*-th element must be on an interval where the two ends are threshold values, say [*p*,*q*] such that there is no object on (*p*,*q*). Since there is no object on (*p*,*q*), placing the *n*-th element on {*p*,*q*} yields the same order. In addition,

placing the *n*-th element on any value on *(p,q)* must result in a larger adjustment. Therefore, it holds for the *n*-th case.

↵

5.5 The Compact (Enumerated) Tree

Further optimization can be done to improve the performance of searching the enumerated tree. A natural implementation of the BF-algorithm may search the enumerated tree in the depth-first order. Therefore, a stack-like structure is required to store the internal nodes during the search. However, this scheme may require very large storage. Actually, a compact representation can be used to reduce the storage requirement. Notice that, for each internal node, there are *count* child nodes. Instead of stacking up all *count* child configurations, one can stack up a compact configuration that can be used to generate these child configurations later. Therefore, each internal node now has two child nodes. One is that it fixed the rightmost 2-tuple to be at its rightmost element, and the other is that it has the same configuration as its parent except that the rightmost 2-tuple only has *count*-1. The semantic is that this 2-tuple can only move to the right at most *count*-1 elements, because the configuration that it moves to the right *count* elements has already been enumerated. In fact, what the optimization does is replace the enumerated tree by an equivalent enumerated binary tree (called compact tree). Figure 5.5 shows the equivalent compact tree of the enumerated tree in Figure 5.3.

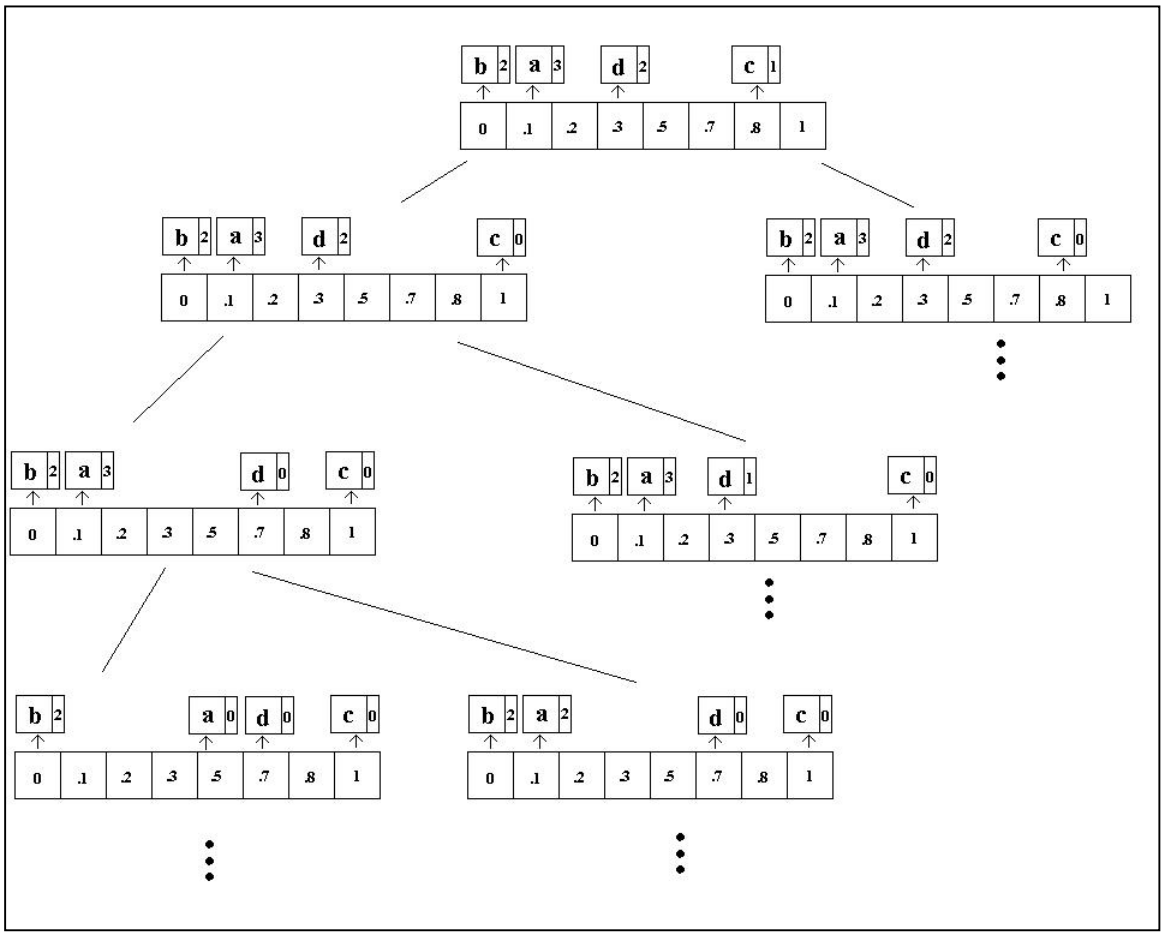


Figure 5.5: The equivalent compact tree of the enumerated tree in Figure 5.3.

5.6 Conclusion

In conclusion, the BF-algorithm provides a way to solve the (optimal) AFP problem by the brute-force approach. It also explains the complexity of the problems and how the data structure *L* can be used to enumerate the images. However, the brute-force approach may be impractical when the problem size is large due to its slow performance. In the next chapter, a new algorithm will be introduced. This algorithm requires further analysis on the structure of the fuzzy phylogenetic tree and actively searches for valid

solutions (i.e. *M'* that is a fuzzy phylogeny). As a result, better performances are expected.

Chapter 6 An Algorithmic Approach

6.1 A Special Case

Before solving the (optimal) AFP problems in general, it is helpful to first look at some special cases. This section discusses a simpler problem, which is a special case of the optimal AFP problem. (See Section 4.4 for the definitions of the problem.) Given an $m \times n$ [0,1] matrix M with an adjustable range $2r$, assume the $m \times n$-1 sub-matrix N (that is the matrix M multiplied by the $n \times n$-1 identity matrix) is a fuzzy phylogeny, and find an $m \times n$ matrix M', such that $M'[i,j]=M[i,j]$, $\forall i,j$, $1 \le i \le m$ *and* $1 \le j \le n$-1, $|M'[k,n]-M[k,n]| \le r$, $\forall 1 \le k \le m$, M' is a fuzzy phylogeny, and the total adjustment is minimized.

In other words, given the first n-1 columns of M form a fuzzy phylogeny, one wants to adjust only the values of the n-th column (within the adjustable range $2r$) such that M is a fuzzy phylogeny. Since this problem is a special case of the optimal AFP problem, the BF-algorithm described in Chapter 5 can be used to solve this problem. However, this special condition (i.e. only one column can be adjusted) not only dramatically reduces the complexity of the problem but also makes it possible to solve the problem more efficiently. The algorithm that solves this problem is explained in detail below and is called the (Special Case) SC-algorithm.

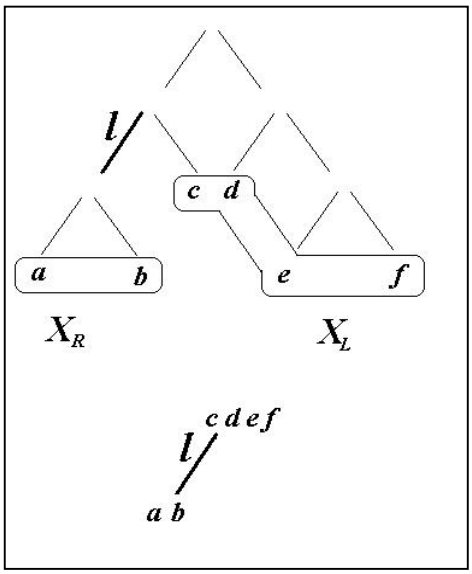


Figure 6.2.1: The two disjoint spanning subsets X_L, X_R, and the labeled edge *l*.

6.2 The Special Case Algorithm

Since the $m \times n$-1 sub-matrix *N* is a fuzzy phylogeny, *N* defines a FP-tree *T*. In order for *M'* to be a fuzzy phylogeny, the *n*-th column must obey the structural information inferred from *N*. In fact, any valid FP-tree *T'* of *M'* must be a refinement of *T*. A tree *T'* is a refinement of a tree *T*, if for every two nodes *u* and *v*, *u* is an ancestor node of *v* in *T*, then *u* is also an ancestor node of *v* in *T'*. In other words, the *n*-th column is compatible with the sub-matrix *N* (i.e. *M'* is a fuzzy phylogeny) if and only if it obeys the structural information from *N*. Therefore, the SC-algorithm attempts to adjust the *n*-th column according to *N*. If there is no valid adjustment, the SC-algorithm rejects *M*. Otherwise, the SC-algorithm returns the minimum adjusted *M'*. In the text below, the structural information is called restrictions.

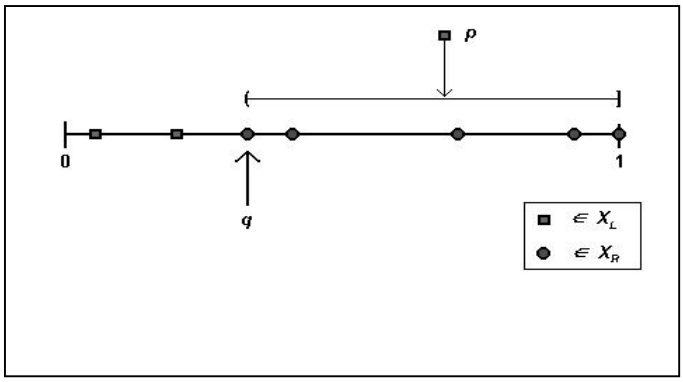


Figure 6.2.2: Viewing the *n*-th column as objects on a [0,1] interval.

The first task is to understand the concept of restrictions. In fact, it is easier to understand restrictions in terms of the FP-tree. Since the $m \times n$-1 sub-matrix *N* is a fuzzy phylogeny, *N* defines a FP-tree *T*. For any valid FP-tree *T'* of *M'*, *T'* must be a refinement of *T*. Furthermore, the *n*-th column refines *T* following exactly one path. (A strict interpretation of the definition of the FP-tree.) Lemma 6.2.1 proves this observation. Let *k* be the number of nodes in *T*; then there are *k* possible paths of where the *n*-th column resides.

Let *l*+ denotes an edge on the path *P* in *T*; then each edge *l*+ separates the set of objects *X* into two disjoint spanning subsets X_L and X_R where X_R contains objects of *X* that is at a descendent node of *l*+ (thus, $X_L = X - X_R$). Figure 6.2.1 is an example of the edge *l*+ on the path and the two disjoint spanning subsets X_L and X_R. Therefore, each edge in *T* enforces one restriction. (The two terms, edge and restriction, may be used interchangeably in the following text if the context is clear.) Lemma 2.3 (see Section 2.3) stays that if a matrix is a (perfect or fuzzy) phylogeny, then no two edges separate *X* in an overlapping manner. Therefore, if *l* is an edge in *T*, then the *n*-th column cannot infer any

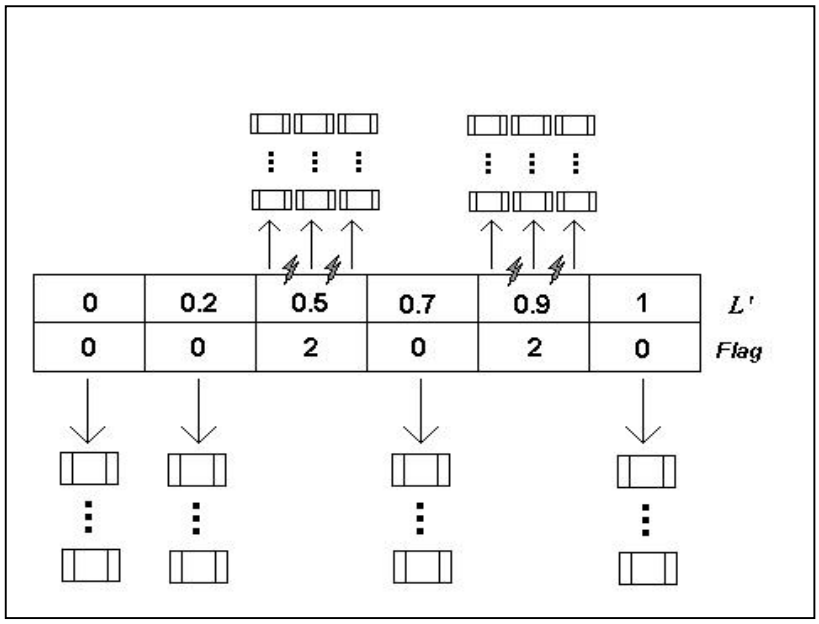


Figure 6.2.3: The improved data structure *L'*.

edge such that it overlaps with *l* in how *X* is to be separated. This observation is proven in Lemma 6.2.2. On the other hand, let *l*- denote an edge that is not on *P* in *T*, then the set of descendent objects of *l*- (*Y*) must be at the same rank. This observation is directly derived from Lemma 2.3 and transformation **Φ**.

Moreover, the relationship between the restrictions and the adjustable range of the *n*-th column is as follows. First, the different values of *M*[*i*,*n*] (1≤*i*≤*m*) can be understood as the positions of the objects on a [0,1] interval. Let *l*+ be an edge in *T* (for *N*) and *l*+ separates *X* into X_L and X_R, then there must be a boundary on this interval that separates X_L and X_R, so that no item can be moved across this boundary. Figure 6.2.2 depicts the overlapping with the point *p* and *q* where *p* represents the rightmost position of the objects in X_R and *q* represents the leftmost position of the objects in X_R. Finally,

let *l*- be an edge that has a set of descendent objects *Y*; then *M*[*y*,*n*] are the same for all *y* $\in$ *Y*.

In the SC-algorithm, the data structure *L* is improved so that it represents the restrictions on the objects. This improved data structure is called *L'* in order to distinguish it from *L*. In *L'*, each 2-tuple is now a 3-tuple < *lcount*, *label*, *rcount*> where *lcount* is the object's left adjustable range; *rcount* is the object's right adjustable range. In addition, each element (representing a point in [0,1]) is associated with a flag that is the number of boundary points at this element. As a result, let *f* be the flag of an element; then there are *f*+1 subset each hold a range of objects between the boundary points. (These subsets are called the boundary subsets in the follow text.) Figure 6.2.3 shows the structure of *L'*.

Using the data structure *L'*, the SC-algorithm iteratively applies the restrictions. For the *l*- restrictions, the set of objects *Y* that are restricted to have the same value is replaced by a new 3-tuple (if valid) where the *lcount* is $\underset{y\in Y}{Max}(lcount)$ and *rcount* is $\underset{y\in Y}{Min}(rcount)$. Then the path *P* is invalid if *lcount* > *rcount*.

For the *l*+ restrictions, the mechanism can be explained as follows. For the first (*k*=1) iteration (i.e. no *l*+ restriction has been enforced), each object in X_L moves to its far left while each object in X_R moves to its far right. Then, The SC-algorithm scans *L'* from left to right to check whether the boundary point *b* exists. If *b* does not exist, return with a rejection. Otherwise, the flag at *q* (the leftmost position of the objects in X_R) is incremented. Furthermore, for any object on *q*, it is placed in the boundary subsets accordingly.

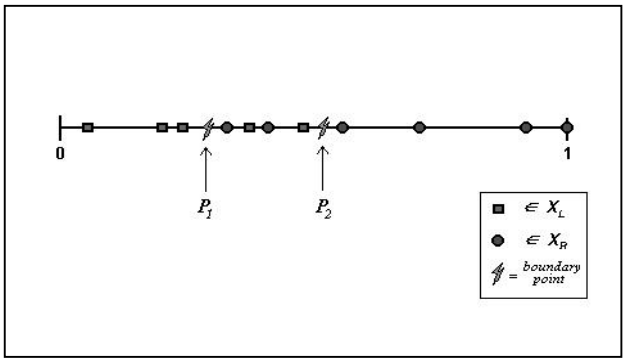


Figure 6.2.4: The two points p_1, p_2 during the *k*-th iteration.

Now, for the *k* iteration (*k*>1), since all restrictions are compatible with each other, the boundary point of the *k*-th restriction must occur between two boundary points p_1, p_2 and $p_1 \geq p_2$, such that objects before p_1 are in X_L and objects after p_2 are in X_R. However, objects in $[p_1, p_2]$ can either be in X_L or X_R. Figure 6.2.4 depicts the boundary point for the *k*-th iteration. Furthermore, Lemma 6.2.3 proves this observation. Due to Lemma 6.2.3, the insertion for the *k*-th boundary point is as follows. Since the objects before p_1 and after p_2 are already in order, only objects in $[p_1, p_2]$ are of concern. The SC-algorithm again first moves all objects in X_L to its far left and all objects in X_R to its far right (also moves the boundary points if necessary). Then, a new boundary point is inserted at *q*, and *L'* is again scanned to ensure its validity.

Finally, at the end of the last iteration, if the remained adjustable range is unique, then the solution is just the reverse mapping of *L'*. On the other hand, if the remained adjustable range is non-unique, the SC-algorithm finds the minimum adjusted *M'* as follows. The SC-algorithm scans *L'* from left to right and one object at a time. For the first object (*k*=1), all the boundary points are moved to their far right, and the first object

is placed at the minimum distance from its original position (i.e. $M[i,n]$ where i is the first (left most) object in L'). For the k-th object ($k>1$), since the adjustment of the first k-1 objects is minimized, the position of the k-th object (that minimizes the adjustment of the first k objects) can then be computed. As a result, the SC-algorithm finds and returns the minimum adjusted M' (after the reverse mapping).

Lemma 6.2.1:

Let M be an $m \times n$ [0,1] matrix and N be the $m \times n$-1 [0,1] sub-matrix of M. Then, if the FP-tree T' for M and the FP-tree T for N exist, T' is a refinement of T. Furthermore, T' refines T by exactly one path (i.e. there exists an unique path P, so that every new edge (edges in T'-T) is on P).

Proof:

Assume the contrast, there are two objects x and y such that x is a descendent of y in T while y is a descendent of x in T'. Then $\Phi(M)$ violates Lemma 2.3; that is a contradiction of that T' exists. Furthermore, a direct interpretation of the definition of FP-trees concludes that T' refines T by exactly one path.

↵

Lemma 6.2.2:

The n-th column is compatible with (the restriction of) the edge $l+$ of a FP-tree T (i.e. M' is a fuzzy phylogeny) if and only if there exists a boundary at point $b \in [0,1]$ such that either $[0,b)$ contains only objects in X_L and $(b,1]$ contains only objects in X_R or the opposite (i.e. $[0,b)$ contains only objects in X_R and $(b,1]$ contains only objects in X_L).

Proof:

If the boundary point *b* exists, then no edge inferred from the *n*-th column will overlap with X_L and X_R. Therefore, according to Lemma 2.3, the *n*-th column is compatible with *l*.

On the other hand, if *b* does not exist, then there is an interval $[q,p] \subseteq [0,1]$, with $p>q$, such that $[q,p]$ infers an edge that overlaps with the separation of X_L and X_R. Thus, it is incompatible with the edge *l*.

↵

Lemma 6.2.3:

At iteration $k>1$, assume that there is a valid adjustment, then there exist two points p_1 and p_2, $p_1 \geq p_2$, such that for any object *x* before p_1 (i.e. *x* is in $[0,p)$), *x* must be in X_L; and any object *x* after p_2 (i.e. *x* is in $(p,1]$) must be in X_R. Furthermore, there is no boundary point on the interval (p_1, p_2).

Proof:

Assume the contrast, for every p_1 and p_2, either $[0,p)$ contains object(s) in X_R or $(p,1]$ contains object(s) in X_L (or both). Then, there does not exist a new boundary point that separates *X* into X_L and X_R. However, it contradicts with the assumption. Therefore, p_1 and p_2 must exist.

↵

6.3 The General Case

The BF-algorithm from Chapter 5 solves the (optimal) AFP problems by enumerating and verifying each possible solution. Although the BF-algorithm solves the problems exactly, it is expected to be slow. On the other hand, the SC-algorithm solves the special case of the (optimal) AFP problems where the $m \times n$-1 sub-matrix N is already a fuzzy phylogeny. The SC-algorithm actively searches for a valid solution; thus, fast performances are expected. Although the SC-algorithm requires the sub-matrix N is already a fuzzy phylogeny and cannot be used in general case, it has made invalidation detection possible. In the next section, a new algorithm is introduced. This algorithm combines the techniques used in the BF-algorithm and the SC-algorithm, such that it solves the (optimal) AFP problems exactly in general and actively searches only for valid answers. This algorithm is called the (General) G-algorithm, and it is presented in detail below.

6.4 The General Algorithm

The idea of the G-algorithm is as follows. Similar to the BF-algorithm, the G-algorithm enumerates all possible configurations. However, by using the concepts of the SC-algorithm, for the k-th column, the G-algorithm only enumerates configurations that are compatible to the k-1 previously selected configurations. In order words, the G-algorithm terminates the branch of enumeration once the $m \times k$ sub-matrix is determined to be invalid (i.e. the remained adjustable range is undefined). Due to the technique of invalidation detection, the G-algorithm reduces a significant portion of the search space

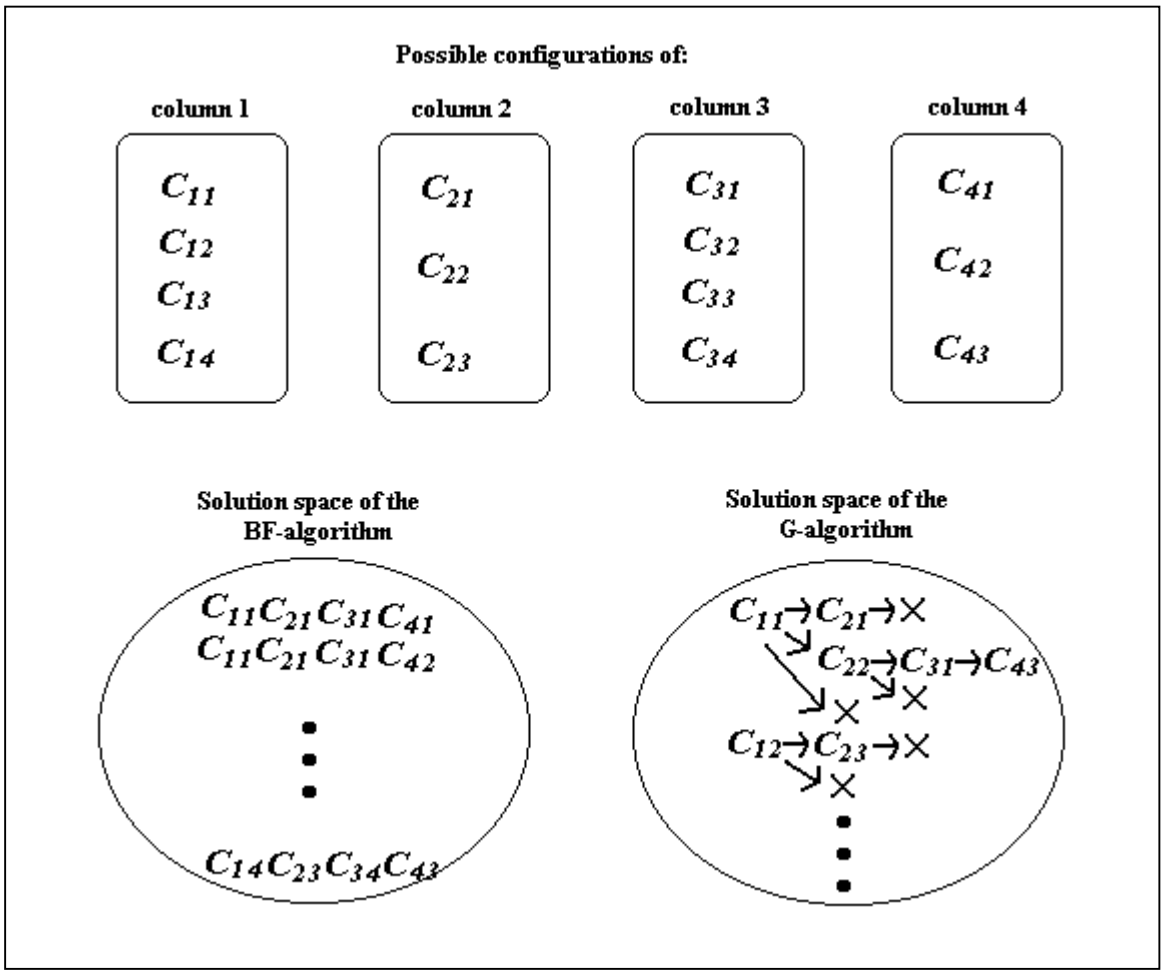


Figure 6.4.1: An illustration of the search space of the BF-algorithm and the G-algorithm.

over the BF-algorithm. As a result, fast performances are expected. Figure 6.4.1 illustrates the search space of the BF-algorithm and of the G-algorithm. Therefore, to solve the AFP problem, the G-algorithm quits when one valid solution is found. To solve the optimal AFP problem; however, the G-algorithm searches the entire solution space and returns the minimum adjusted *M'*.

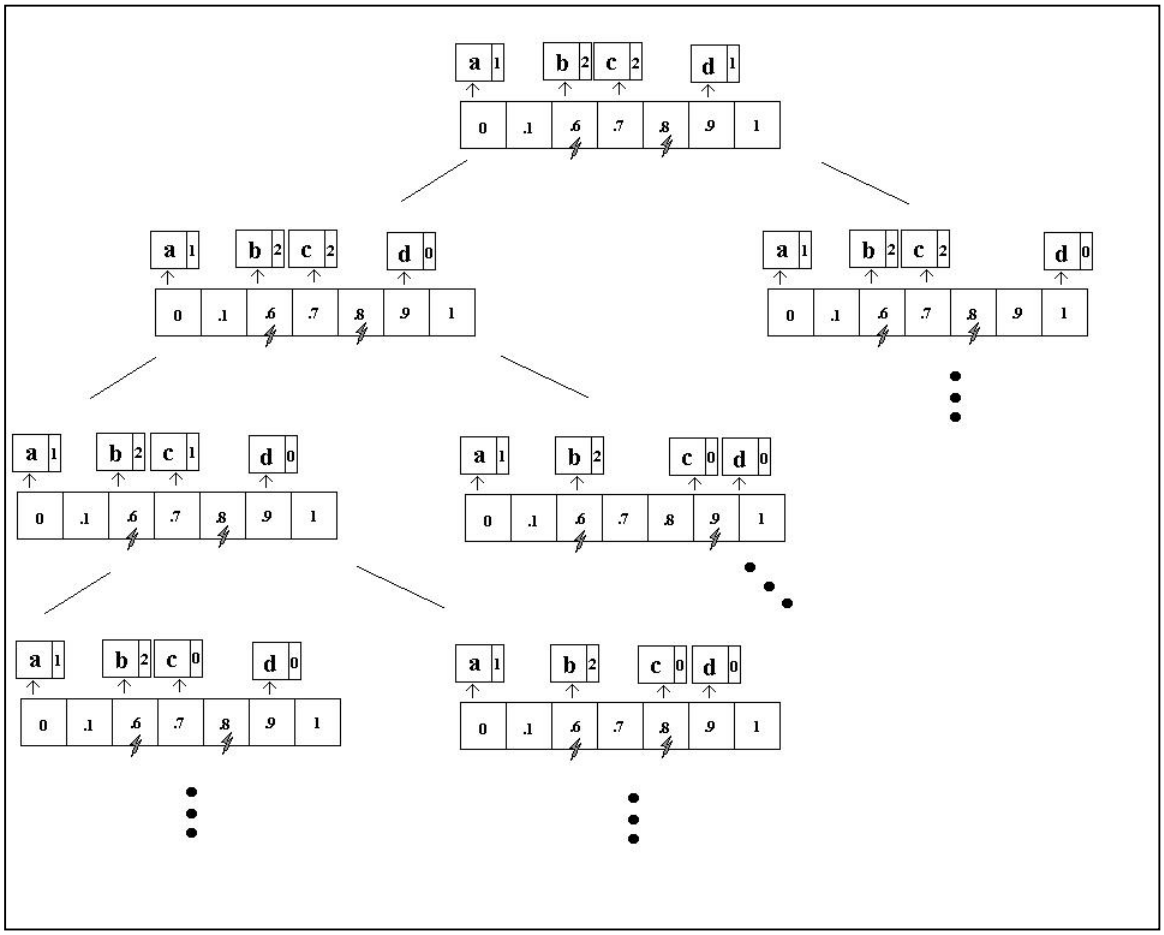


Figure 6.4.2: The enumerated tree of the G-algorithm.

The key for the G-algorithm is to enumerate configurations with restrictions enforced. The SC-algorithm iteratively reduces the adjustable range of the *n*-th column so that it obeys the set of restrictions from *N*. This technique is again used in the G-algorithm. Therefore, the discussion in this section is focused on how the possible configurations are enumerated from the data structure *L'*.

Given an instance of *L'*, the G-algorithm enumerates configurations as follows. First, the root of the enumerated tree is again called the initial configuration and is the

configuration where each 3-tuple moves to its far left. In addition, all boundary points are also moved to their far left accordingly. Then, for each non-fixed configuration, two child nodes are generated. One is that it fixed the rightmost 3-tuple to be at its rightmost element, and the other is that it has the same configuration as its parent except that the count of the rightmost 3-tuple is decremented by 1. The enumeration is indeed very similar to the enumeration in the BF-algorithm with the consideration of also moving the boundary points. Figure 6.4.2 depicts the enumerated tree for the G-algorithm.

6.5 Conclusion

In conclusion, the SC-algorithm actively searches for a valid solution, instead of passively verifying all possible solutions. Therefore, better performances are expected. In fact, unlike the BF-algorithm (that is expected to run in super-polynomial time), the SC-algorithm runs in polynomial time due to the special condition. Although the SC-algorithm does not solve the problems in general, the G-algorithm solves the problems in general by repeatedly applying the SC-algorithm, which results in a better performance (than the BF-algorithm). In fact, the G-algorithm combines the techniques used in the BF-algorithm and the SC-algorithm. It explores the entire solution space by enumeration and at the same time efficiently searches only for valid answers by detection of invalidity. Therefore, the G-algorithm is an improvement over the BF-algorithm. However, due to the complexity inherited from the problems, the performance of the G-algorithm may still be slow for some instances. Therefore, in the next chapter, a heuristic algorithm is proposed. Unlike the BF-algorithm and the G-algorithm that search the solution space systematically, this new algorithm attempts to search the solution space heuristically.

Such approach may result in better performance; however its successfulness and optimality are not guaranteed.

Chapter 7 A Heuristic Approach

7.1 Heuristics

The G-algorithm is an improvement over the BF-algorithm because it reduces the search space by pruning off invalid solutions. However, due to the inherited complexity of the problems, slow performance may still occur for some problem instances. Therefore, in this chapter, a heuristic algorithm called the (Heuristic) H-algorithm is introduced that serves as an alternative of the exact algorithms to the (optimal) AFP problems (i.e. the BF-algorithm and the G-algorithm). Firstly, the H-algorithm extends the definition of valid solutions so that many of the "second class" solutions are now of interest. Secondly, due to the new definition, the H-algorithm searches for solutions by a different approach that offers fast performance even in the case of high complexity problem instances. Thirdly, although the H-algorithm is expected to find a near-optimal solution, it does not guarantee the optimality of the solution. Finally, the H-algorithm is, in fact, an application of a new general purpose searching technique. This new searching technique is called the *Particles Algorithm* and is discussed in detail in this chapter.

7.2 The Particles Algorithm

The Particles Algorithm is a new general purpose searching technique that models after physical properties of particles. Imagine a container that contains various small particles. Due to the random motion of the particles, small particles will collide and

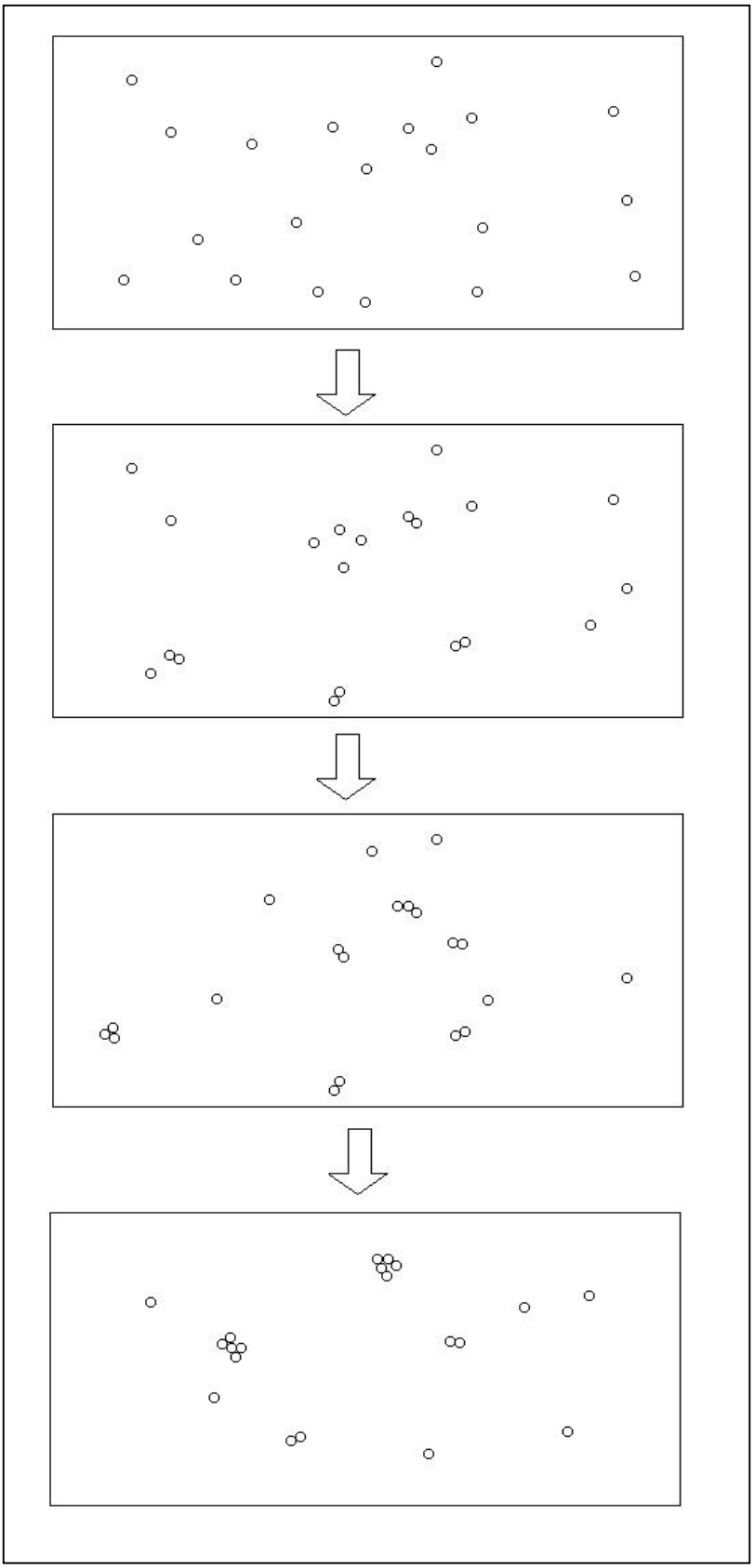


Figure 7.2.1: An illustration of random motions and collisions of particles.

```
Particles Algorithm
{
        initialize the container with particles
        while (terminating condition has not been met)
        {
                simulate collisions and merge particles if it is stable
        }

        return the best particle
}
```

Figure 7.2.2: The pseudo-code of the Particles Algorithm.

combine to form a compound (a big particle) if it is stable. Similarly, big particles will collide to form a bigger particle. After some time, the container is expected to contain a significant amount of big sized particles. The idea of particle collisions is illustrated in Figure 7.2.1.

The analogy between the particle collisions and solution searching is as follows. Often, a solution to a problem consists of multiple elements, A, B, …,N, and each element has various instances, for example, A={ $a_1, a_2, \ldots, a_k$ }. Therefore, an instance of an element can be thought of a particle in a container. Then, a solution can be seen as a compound (a big particle) that consists of exactly *n* particles each from an element. The random motion of the particles creates collisions that combine particles to form a big particle if it is stable. This mechanism is parallel to combining two partial solutions of a problem if the resulted (partial) solution is valid. Under suitable conditions, the small particles are expected to interact and form the desired products. Therefore, with careful design, one can also expect the Particles Algorithm to produce desired solutions to the problem. The Particles Algorithm is presented as pseudo-code in Figure 7.2.2.

The important concepts of the Particles Algorithm are discussed in this chapter. However, the implementation details are beyond the scope of this chapter. Therefore, the discussion is limited to the general functionality and the possible strategy. The Particles Algorithm involves a series of parameters: the number of particles, the initial particles, the collision rate, the terminating condition and the collision mechanism. Furthermore, each parameter is discussed in detail below.

Number of particles

The number of particles is the number of "building blocks" to solutions in the container. The more particles in the container initially, the more big particles are expected at the end. However, more particles also require longer computational time to simulate the collisions. Therefore, the number of particles in the container should be proportional to the complexity of the problem, such that it is sufficient and efficient to search for good solutions.

Initial particles

The initial particles determine the upper bound of the quality of solutions unless injections of new particles or transformations of particles are allowed later in the process. Therefore, the initial particles must cover the good portion of the solution space (if not the entire solution space). In addition, it is possible to optimize the initial particles to some known particles (i.e. known (partial) solutions) so that the prior knowledge can be incorporated in the search.

The collision rate

The collision rate defines the number of collisions in an iteration. Assume that at any moment, one particle can at most collide with one other particle. Then, the maximum

collision rate should define a “total collision”. Let the collision rate be a value ∈ [0,1], then collision rate = 1 means that all particles (except when the number of particles is an odd number) will collide with one another. Similarity, collision rate = 0 means that there will be no collision undertaken. However, the collision rate does not need to be a fixed parameter. In fact, the idea of temperatures can very well cooperate with the Particles Algorithm to vary the collision rate. The idea of temperatures is to vary the collision rate according to the temperature of the environment. When the temperature is high, the random motions of the particles are rapid and therefore result in high collision rate. On the other hand, when the temperature is low, it results in low collision rate. This idea allows the Particles Algorithm to either escape or focus on local optima by varying the collision rate.

The terminating condition

The terminating condition is usually motivated by two concerns, the user-defined condition and the convergence of solutions. The user-defined condition is a limit that the user imposed (e.g. time limit). Therefore, the Particles Algorithm must return a solution when the condition is reached (e.g. when the time is up). The convergence of solutions is the situation where the quality of the solutions remains settled in a significant number of iterations. Then, the Particles Algorithm returns the best solution because it is unlikely that a better solution would be found in further searching.

The collision mechanism

The collision mechanism perhaps is the most interesting and flexible component of the Particles Algorithm. The collision mechanism should define how particles collide, for example, one-to-one collisions or multi-way collisions; random collisions or

collisions that are based on some molecular attractions; merging only collisions or collisions that breaks up big particles … etc.

The Particles Algorithm itself is a very interesting topic that opens the doors of various researches. The Particles Algorithm, although it is a new technique (as far as the author is aware of), shares various concepts of other search techniques, such as the *Genetic Algorithm* [28] and the *Simulated Annealing* [39]. All three techniques use the natural computing paradigm [49] to solve complex problems where possible solutions to a problem is represented as a natural entity (an individual) and the search for the optimal solution is simulated as a natural event. These techniques navigate the solution space by different principles according to the natural models behind the techniques. Most natural computing techniques require an individual to be a full solution to the problem. In Particle Algorithm, however, a particle is a partial solution to the problem. That in turn allows the algorithm to simultaneously search for good partial solutions to the problem and find for good partial solutions to combine.

7.3 The H-algorithm

The H-algorithm is an application of the Particles Algorithm to the (optimal) AFP problems. (See Section 4.4 for the definitions of the problems.) With the H-algorithm, acceptable solutions can be obtained even in the case where the BF-algorithm and the G-algorithm reject the matrix. Furthermore, the H-algorithm, although considers, but it does not visit the entire solution space. As a result, fast performance is expected.

Before the discussion of the H-algorithm, new definitions for solutions are needed. Throughout the chapters, the term optimal solution has been used to mean the

minimum adjusted matrix *M'* (from *M* and 2*r*) that is a fuzzy phylogeny. In addition, the term valid solution has been used to mean a matrix *M'* (adjusted from *M* and 2*r*) that is a fuzzy phylogeny. Now, define a reduced matrix of an $m \times n$ matrix *M* as the $m \times k$ matrix *K* resulting from the removal of any *n*-*k* columns from *M*. Furthermore, define an adjusted reduced matrix of an $m \times n$ matrix *M* as the [0,1] $m \times k$ matrix that is adjusted from *K* by at most *r* where *K* is a reduced matrix of *M* and 2*r* is the adjustable range of *M*. As a result, a new term, acceptable solution, is introduced to mean a matrix *K'* that is an adjusted reduced matrix of *M'* and is a fuzzy phylogeny. Finally, an acceptable solution *K'* (an $m \times k$ matrix) is called a *k*-acceptable solution where *k* is the number of columns of *K'* (and it is the number of characters it used to infer the fuzzy phylogeny).

All in all, the term (*k*-)acceptable solution is introduced to describe solutions that use less than all *n* characters to infer the fuzzy phylogeny. Some of the experimental data of characters may deviate beyond the expected deviation (which is captured by the concept of the adjustable range). Also, some characters are poorly chosen and are actually not suitable to be used to infer the fuzzy phylogeny. The term (*k*-)acceptable solution allows an algorithm to return a solution under these circumstances. As a result, the H-algorithm solves another variation of the AFP problem, called the relaxed AFP problem.

Definition of the relaxed AFP problem:

Given an $m \times n$ [0,1] matrix *M* and a value $r \in [0,1]$, find an $m \times n'$ matrix *M'* where $n \leq n'$, such that each column *j'* in *M'* is associated with an unique column *j* in *M*,

$|M[i,j]-M'[i,j']| \leq r$ and $M'[i,j'] \in [0,1]$, for $1 \leq i \leq m$ and $1 \leq j' \leq n'$, M' is a fuzzy phylogeny, and the total adjustment a is small.

↵

The H-algorithm views each configuration as one particle, and a particle is stable if and only if no two configurations are of the same type (i.e. they are both configurations of character i) and the matrix K' which that particle corresponds to is a fuzzy phylogeny. Notice that each stable particle is, in fact, an acceptable solution. The H-algorithm initializes the container with randomly generated configurations and simulates one-to-one collisions between the particles. In addition, the collisions are simulated based on random motions. Therefore, if two particles collide and they are compatible, then they will combine and form a big particle; otherwise, the collision is ineffective. At the end of all iterations, the H-algorithm returns the "best" particle in the container. The best particle is the k-acceptable solution that has the maximum k, and in the case of a tie, the best particle is the one that has the minimum total adjustment.

7.4 Conclusion

The three algorithms, the BF-algorithm, the G-algorithm and the H-algorithm, solve the (optimal or relaxed) AFP problems in different ways. The BF-algorithm is the first attempt to solve the problems. It uses Lemma 5.4.1 to divide the infinite solution space into a finite number of equivalent classes. Then by using the brute-force method, the BF-algorithm exhaustively enumerates the possible solutions and verifies their correctness and optimality. The BF-algorithm is intuitive and solves the problems

exactly. However, due to the exhaustive enumeration, the BF-algorithm is expected to have slow performances.

On the other hand, the G-algorithm uses the techniques from the SC-algorithm that prunes off invalid search space once the partial solution is determined to be invalid. The G-algorithm not only reduces the search space over the BF-algorithm but also solves the problems exactly. The G-algorithm is expected to have fast performances in general. However, due to the inherited complexity of the questions, slow performance may still occur in the high complexity problem instances.

Therefore, the H-algorithm is proposed to provide an alternative over the BF-algorithm and the G-algorithm. The H-algorithm is an application of a new general purpose searching technique called the Particles Algorithm. The H-algorithm extents the definition of valid solutions to cover more “second class” solutions. The H-algorithm heuristically searches for good solutions and is expected to have fast performances even in the case of high complexity problem instances.

In the next chapter, an empirical study of the three algorithms is presented. The study compares the algorithms in different dimensions and then discusses and concludes the experimental results.

Chapter 8 The Performance Analysis

8.1 Simulated Biological Data

To analyze the performances of different proposed methods, simulated biological data are used. The performance analyses consist of both positive data and negative data. Positive data is a matrix that is a fuzzy phylogeny. Negative data is a matrix that is not a fuzzy phylogeny. The simulation of negative data is trivial. An $m \times n$ matrix M is generated where each $M[i,j]$ is uniformly chosen from [0,1]. If M is occasionally a fuzzy phylogeny, it will simply be discarded. Notice that the adjustable range $2r$ is not required when generating negative data.

To simulate positive data, an evolutionary model of fuzzy characters is used. The model is compatible with the fuzzy phylogeny model. Over time, it simulates the character development of the objects. Therefore, at the beginning, every object is primitive and does not have any character. This is depicted by the root node of the FP-tree. Then, a divergence occurs and divides the objects into two groups, which is depicted by the two child nodes of the root node. A divergence is an evolutionary event where some objects develop a degree of a character while other objects remained unchanged. These evolutionary events repeat over time and further divide the objects into smaller groups. Figure 8.1 shows an example of a FP-tree and the divergence of objects.

In order to simulate different frequencies and rates of character development, each character is associated with two values, *weight* and *rate*; each of which is uniformly

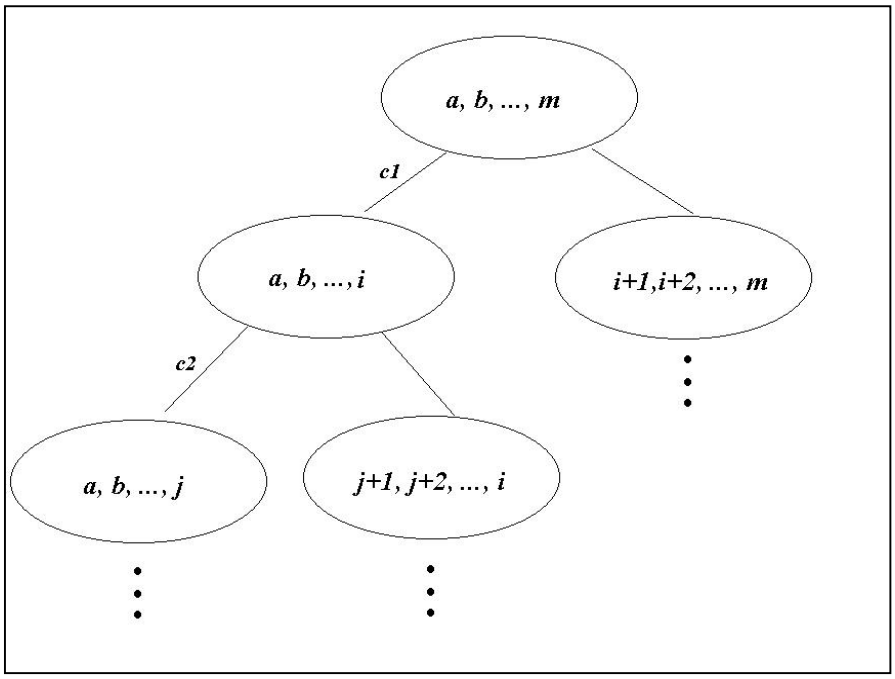


Figure 8.1: A FP-tree and the divergence of the objects.

chosen from [0,1]. For each divergence, a character is chosen randomly according to the *weights*. In addition, the development increment is chosen randomly from a Gaussian [9,25] distribution with mean = *rate* and variance = $(\frac{rate}{2})^2$. Next, the division of the groups is chosen in uniform; each object has equal chance to go to either group.

Furthermore, the occurrence of divergence is chosen randomly among the current leaf nodes. Finally, the frequency of divergence is chosen uniformly from [*n*,*m*). The term *n* is the number of characters, which is an obvious lower bound of the frequency of divergence. The term *m* is the number of objects, so *m*-1 is the number of divisions required to separate the *m* objects. Notice that the *rate* of the character development is chosen randomly. Therefore, normalization is required after the tree is generated. The

normalization replaces the expression value $M[i,j]$ by $\frac{M[i,j]}{\underset{i=1}{\overset{m}{Max}} M[i,j]}$. Thus, all expression values are between [0,1].

Now, positive data is generated. To simulate the experimental errors and other biological noises, the values in M are to be deviated. As a result, for each $M[i,j]$, there is a probability *deviation_pr* that a deviation will be injected, in which the deviation is chosen randomly from a Gaussian distribution with mean = 0 and variance = $(\frac{r}{2})^2$. If the deviation is beyond the adjustable range $2r$, then it is replaced by $M[i,j]+r$ or $M[i,j]-r$ accordingly.

The simulation of positive data ensures the matrix M is adjustable to a fuzzy phylogeny with respect to the adjustable range $2r$. In addition, the original matrix, the matrix before the injection of deviations, provides a referenced matrix M' (and a FP-tree). The advantage of using simulated data is that the performance analyses can be conducted in various parameters, such as the number of objects, the number of characters, and the adjustable range. In fact, it is impractical (or even impossible) to collect real biological data satisfying the dimensions of all parameters.

8.2 Empirical studies

With the simulated data, it is possible to conduct empirical studies on the proposed algorithms. This section explains analyses that are designed to evaluate the performances. Specifically, the time analysis assesses the computational time required for an algorithm and the quality assesses the quality of the solution from an algorithm. (Note that the performance of an algorithm is subjected to the implementation). The time

analysis is performed to all three algorithms (the BF-algorithm, the G-algorithm and the H-algorithm); however the quality analysis is performed only on the H-algorithm (since the BF-algorithm and G-algorithm both solve the (optimal) AFP problems exactly).

All three algorithms are implemented in JAVA [35] using J2SE 1.4.2. In addition, the empirical studies are conducted in a DELL machine with a 1GHz Intel processor and 256MB RAM running Windows XP. The testing data are generated according to the simulation discussed in Section 8.1. In addition, the set of parameters are as follows. Let m be the number of objects, then, the number of characters $n = \left\lfloor \frac{m}{2} \right\rfloor$; the one-sided adjustable range $r = \left\lfloor \frac{1}{4m} + random(\frac{1}{4m}) \right\rfloor$ where $random(k)$ is uniformly distributed over $[0,k]$; the deviation probability $deviation\_pr = 0.8$. Finally, each reported result is an average of ten trials.

8.3 Results

Performance of the Brute-force Algorithm

The time analysis is conducted on the BF-algorithm solving both the AFP problem and the optimal AFP problem. There are three series of tests, “Positive” denotes the AFP problem using positive data, “Positive Optimal” denotes the optimal AFP problem using positive data and “Negative” denotes the AFP problem using negative data. Notice that, there is no “Negative Optimal” because it is expected to be the same as “Negative”. Figure 8.3.1 shows the time analysis of the three tests of the BF-algorithm on a logarithmic scale. In addition, zeros are arbitrarily reset to one in the logarithmic plot. The data of results can also be found in Table 8.3.1 in Appendix A.

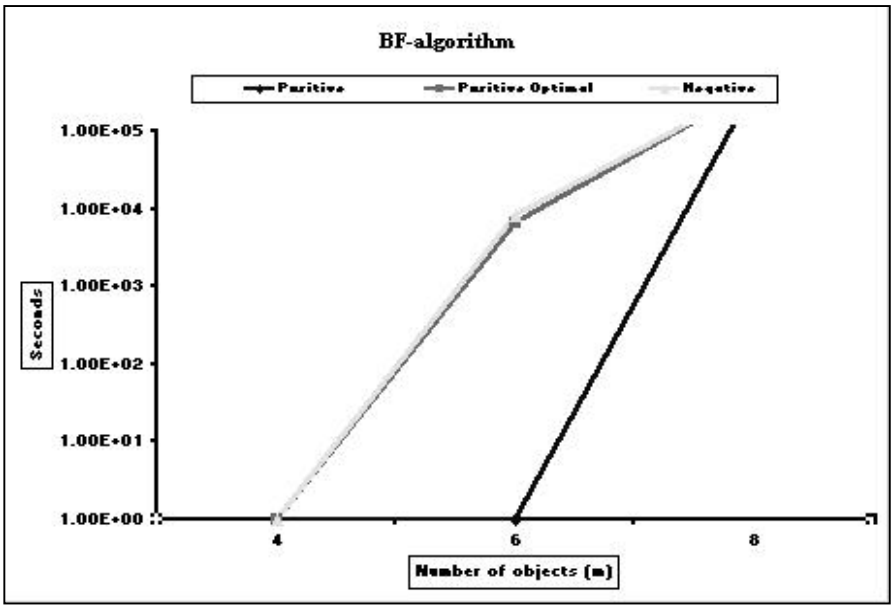


Figure 8.3.1: The time analysis of the BF-algorithm.

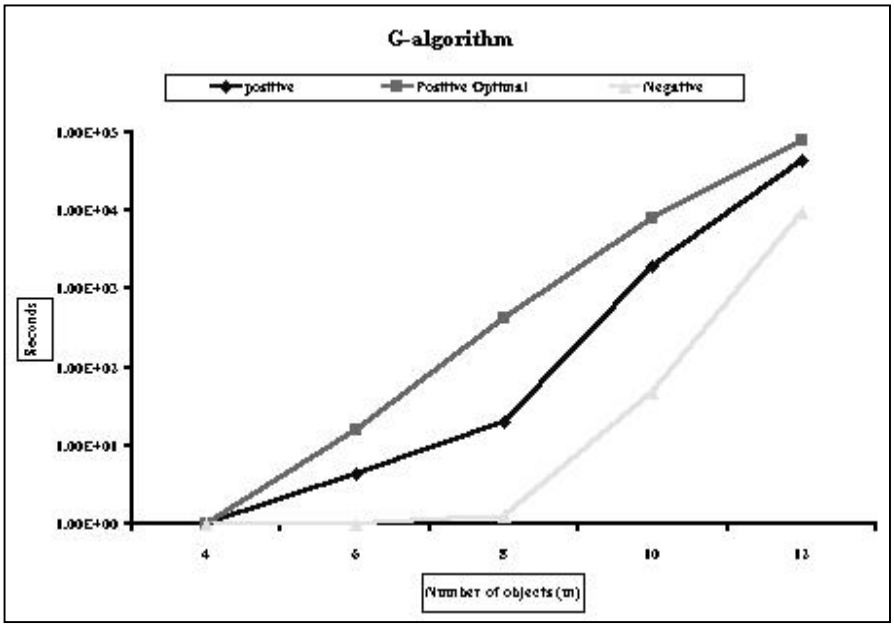


Figure 8.3.2: The time analysis of the G-algorithm.

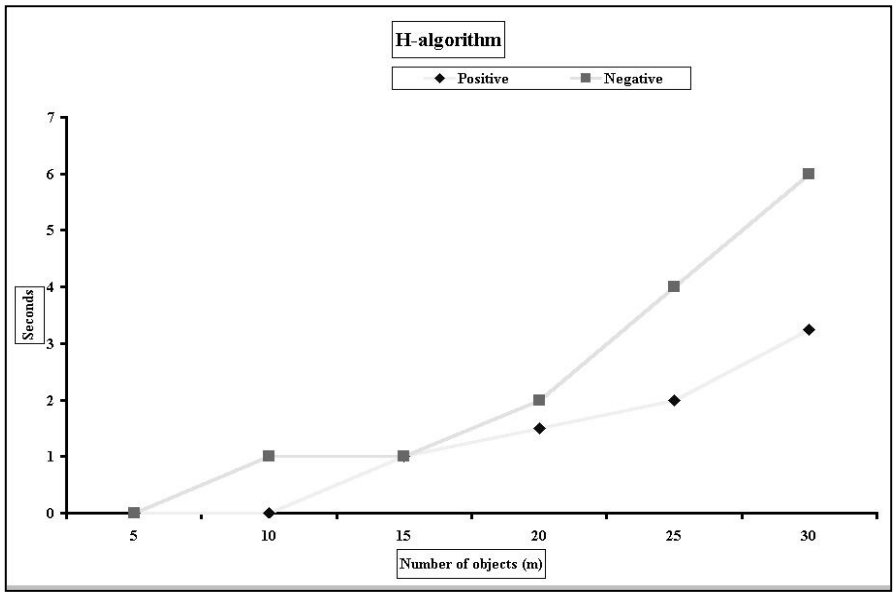


Figure 8.3.3: The time analysis of the H-algorithm.

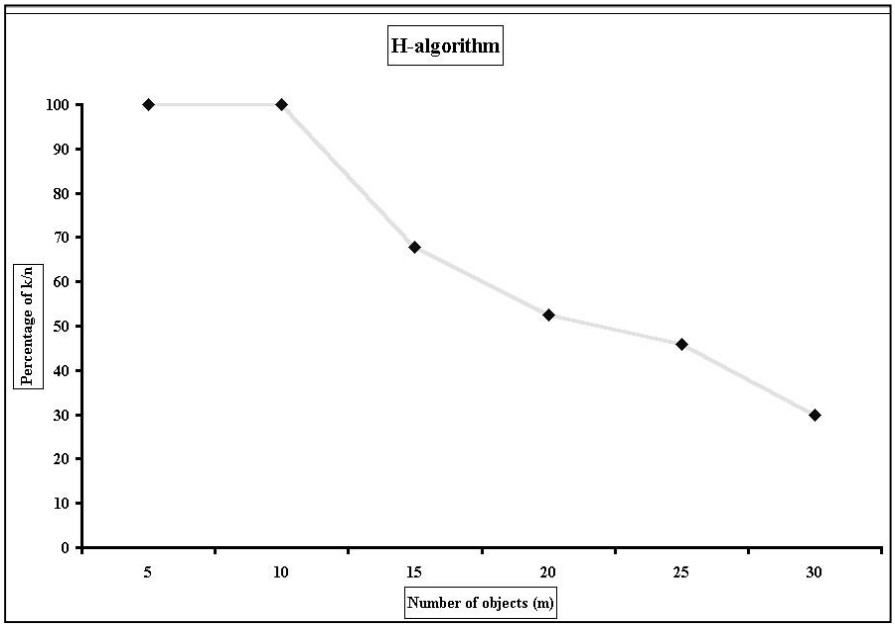


Figure 8.3.4: The quality analysis of the H-algorithm.

Performance of the General Algorithm

The time analysis is conducted on the G-algorithm, solving both the AFP problem and the optimal AFP problem. There are again three series of tests, “Positive”, “Positive Optimal” and “Negative”. Figure 8.3.2 shows the time analysis of the three tests of the G-algorithm on a logarithmic scale. In addition, zeros are arbitrarily reset to one in the logarithmic plot. The data of results can also be found in Table 8.3.2 in Appendix A.

Performance of the Heuristic Algorithm

The parameters to the H-algorithm are selected for the best results. The parameters are as follows. The termination condition *termination* = 30 (i.e. the best solution remains in *termination* iterations); the collision rate *coll_rate* = 0.7; the number of particles *size* = 100 (note that *size* is the number of particles for each kind. i.e. the total number of particle is *n* x *size*).

Both time analysis and quality analysis are conducted on the H-algorithm solving the relaxed AFP problem. There are only two series of tests, “Positive” and “Negative”. Figure 8.3.3 shows the time analysis of the G-algorithm on the two series, and Figure 8.3.4 shows the quality analysis of the G-algorithm on the two series. The data of results can also be found in Table 8.3.3 and Table 8.3.4 in Appendix A.

8.4 Discussion

Discussion on the Brute-force Algorithm

The BF-algorithm is the first attempt to solve the (optimal) AFP problems. The correctness and termination of the algorithm is guaranteed; however, it is expected to be slow due to the complexity of the problems. Figure 8.3.1 shows a consistent result that

the computational time increases rapidly in a small interval. When m = 4, the BF-algorithm (in general) terminates in less than a second. However, it takes almost two hours to terminate when m = 6. Furthermore, the computational time quickly becomes infeasible to measure when m = 8.

Among the three series, “Positive”, “Positive Optimal” and “Negative”, there are only slight differences in their performance. The similar behavior between the two series, “Positive Optimal” and “Negative”, are expected. It is because of that the BF-algorithm would have to brute-force all possible solutions for either series before termination. However, a closer look at the two series reveals that “Positive Optimal” has a slight advantage over “Negative”. The slight advantage is indeed due to the content of the data (i.e. the matrix M). A biologically simulated matrix M is expected to have fewer distinct values than a randomly generated matrix M'; therefore the BF-algorithm takes longer to compute M than M'. On the other hand, when comparing “Positive” with “Positive Optimal”, the BF-algorithm quits when one valid solution is found and examines all possible solutions, respectively. Therefore, “Positive” is expected to yield a better performance than “Positive Optimal”. Due to the scale of the plot, this fact is not obvious in Figure 8.3.1. However, it can be concluded from Table 8.3.1.

In conclusion, although the correctness and termination are guaranteed in the BF-algorithm, the computational time requires is infeasible in practice size. Furthermore, there is little difference in performance regardless of the type of the data and the problem (optimal or not). It indicates that the BF-algorithm is not taking full advantage of the structure of the data. These observations are indeed consistent with theoretical prediction.

Discussion on the General Algorithm

The G-algorithm is the second attempt to solve the (optimal) AFP problems. The correctness and termination of the algorithm is again guaranteed. Figure 8.3.2 shows a significant improvement of the G-algorithm over the BF-algorithm. For instance, the G-algorithm takes less than one second when $m = 8$ while the time required in the BF-algorithm is already infeasible. However, due to the complexity of the (optimal) AFP problems, the computational time still quickly increases with m.

Among the three series, "Positive", "Positive Optimal" and "Negative", there is much difference in their performance. There are clear distinctions in performance with the order, "Positive Optimal" > "Positive" > "Negative". Since the G-algorithm prunes off invalid solutions once they are detected, performance distinctions are very evident. Most noticeably, "Negative" terminates quickly because all solutions are invalid. Again, "Positive" terminates quicker than "Positive Optimal" because of the early termination.

In conclusion, the G-algorithm not only guarantees the correctness and termination, but it also has a significant improvement on the computational time over the BF-algorithm. Furthermore, the G-algorithm takes full advantage of the structure of the data and reduces computational time when possible. Finally, these observations are also consistent with theoretical prediction.

Discussion on the Heuristic Algorithm

The H-algorithm is the third attempt to solve the (optimal) AFP problems. Actually, it solves the relaxed AFP problem, which is a variation of the optimal AFP problem. The correctness and termination of the algorithm are not guaranteed nor required in the H-algorithm. The goal is to retrieve a "good" solution quickly. Figure 8.3.3 shows that the G-algorithm only takes seconds to retrieve the solution. For

instance, it takes about three to six seconds to retrieve solution when m = 30. Similarly, due to the number of distinct values, “Positive” again has a slight advantage over “Negative”.

Figure 8.3.4 shows how the quality of the solutions decreases with increasing m, where quality = ($\frac{n'}{n}$ x 100%). The H-algorithm retrieves solution with quality = 100% when $m \leq 10$. Notice that these solutions are also solutions to the AFP problem since $n' = n$. In addition, the H-algorithm retrieves a solution with quality ≥ 50% when $m \leq 22$. Then the quality decreases steadily when $m > 22$.

In conclusion, the H-algorithm does not guarantee the optimality of the solution; however, it offers a virtually instant retrieval of solution with reasonably good quality. When $m \leq 10$, the H-algorithm actually solves the AFP problem. However, it only takes less than one second, compared to about half an hour for the G-algorithm. The H-algorithm is a good alternative to solve the (optimal) AFP problem. Furthermore, it is possible that the quality of the solutions can be increased by compromising the computational time (which is virtually instant currently), by adjusting the parameters to the H-algorithm.

Chapter 9 Future Work

9.1 The Enhanced Algorithm

By using the data structure L (and L'), configurations are enumerated in a tree fashion. However, the G-algorithm reveals that two configurations c_1 and c_2 impose the same phylogenetic information (structurally) if they have the same order of objects. Furthermore, the G-algorithm also shows how to obtain the minimum adjustment given a particular order of objects. Therefore, it is possible to further reduce the solution space by partitioning configurations that impose the same phylogenetic information into equivalence classes. In other words, instead of enumerating all possible combinations of configurations, one wants to enumerate all possible permutations of configurations. This algorithm is temporarily called the (Enhanced) E-algorithm.

The E-algorithm should only enumerate configurations with unique order. Therefore, the enumeration scheme needs to be modified. In fact, the modification is simple. Recall from Chapter 6 that a non-fixed configuration in the enumerated tree expands into two children. One is that the rightmost element is moved to its far right position, and the other one is that the rightmost element has *count*-1. It is not difficult to see that both children might have the same order of objects. Therefore, in the E-algorithm, instead of moving the rightmost element, the first (from right to left) element that creates an order change is moved. Figure 9.1 depicts the enumerated tree for the E-algorithm. This way, all leaf configurations of the enumerated tree would have unique

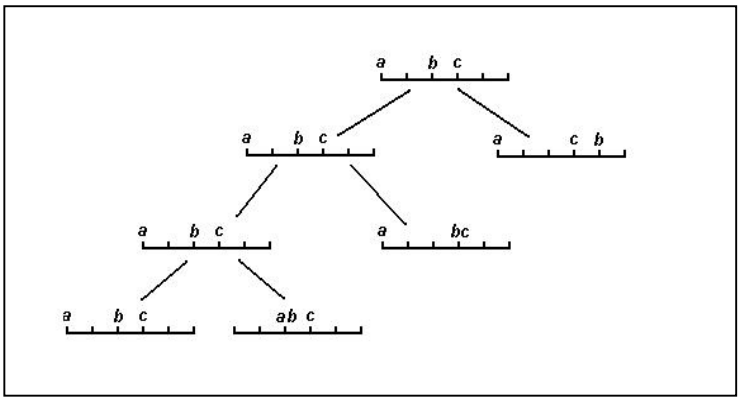


Figure 9.1: An illustration of the enumerated tree for the E-algorithm.

order of objects. As a result, the solution space is further reduced, and the optimality of the algorithm is reserved. Finally, given a particular order of objects, the minimum adjusted configuration (that has the same order) can be obtained using similar scheme as the G-algorithm (see Section 6.2). The E-algorithm is expected to further improve the performance from the G-algorithm. With a good choice of implementation platform, the author also expects that the (optimal) AFP problem could be solved within a good practical range.

9.2 More Relaxation Problems

The (optimal) AFP problems suggest a new angle to phylogenetic tree constructions. Recall from Chapter 4 that there are two main approaches for building phylogenetic trees: one heuristically constructs the phylogenetic tree without a validity check on the experimental data, and the other one requires the experimental data to follow a tight definition. The (optimal) AFP problems offer a medium of the two, in

which it requires a validity check on the experimental data and also provides a relaxed definition of the phylogenetic model.

For instance, relaxation problems can easily be formulated for the ultrametric tree and additive tree models. Here two possible formulations are suggested. Furthermore, these two formulations are called the AUP problem and the AAP problem respectively.

Definition of the AUP problem:

Given an $m \times n$ symmetric matrix M where each entry $M[i,j] \in [0,\infty]$ and a value $r \in [0, \infty]$, find a matrix M' such that $|M[i,j]-M'[i,j]| \leq r$ and $M'[i,j] \in [0, \infty]$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, and M' is ultrametric.

↵

Definition of the AAP problem:

Given an $m \times n$ symmetric matrix M where each entry $M[i,j] \in [0,\infty]$ and a value $r \in [0, \infty]$, find a matrix M' such that $|M[i,j]-M'[i,j]| \leq r$ and $M'[i,j] \in [0, \infty]$, for $1 \leq i \leq m$ and $1 \leq j \leq n$, and M' is additive.

↵

9.3 Confidences of the Solutions

The concept of the adjustable range r allows us to construct the phylogenetic tree with imperfect data values. In addition, the concept of the total adjustment provides an elementary method to assess how confidently M' infers the true evolutionary relationship. A better measurement could be introduced that assesses the confidence of T, not M'. Below are three possible formulations of the problems.

1. Let *AM* be the set of *M'* that is adjustable to a fuzzy phylogeny, and let *TM* be the set of FP-tree topologies induced from *AM*. Given *M* and *r*, compute the size of *TM* (i.e. |*TM*|).

2. Let *S* be the space where *M* is within the adjustable range, and let *s* be the subspace of *S* that yields *T*. The confidence of *T* is that, *conf(T) = sizeOf(s)/sizeOf(S)*. Given *M*, *r* and a FP-tree topology *T*, find *conf(T)*.

3. Given *M* and *r*, find a FP-tree topology *T*, such that *conf(T)* is maximized.

Chapter 10 Conclusions

The study of phylogenetics involves the identification of evolutionary relationships between species. Perfect phylogeny is one of the classical character-based models. In this model, there are only two states for each character, zero, where the character is absent, and one, where it is present. The perfect phylogeny problem is to verify whether a given set of species with a set of characters form a phylogenetic tree, and if it does, construct the corresponding phylogenetic tree. Dan Gusfield describes an $O(mn)$ time algorithm for this problem where m is the number of species and n is the number of characters. In addition, this algorithm is proven to be asymptotically optimal in time.

All previous perfect phylogeny based models assume the states of the characters are discrete. However, fuzzy boundaries between species and degrees of character development are commonly found in nature. Fuzzy boundaries refer to the ambiguous definition between presence and absence of a character and degrees of character development refer to the various expression levels of a character. These phenomena show the need for a more relaxed model. This dissertation proposes the fuzzy (perfect) phylogeny model that extends the perfect phylogeny model to allow fuzzy memberships of the characters. The conversation of properties in the new model is then proven. This dissertation also shows how the fuzzy phylogeny problem can be transformed to the perfect phylogeny problem so that it can be solved by previously developed algorithms, such as Gusfield's algorithm. Due to the fuzziness of characters, two relaxation problems

for the fuzzy phylogeny are proposed, namely the AFP problem and the optimal AFP problem. However, the problems are proven to be **NP-hard**.

Three algorithms are then proposed to solve the adjustment problem, namely the BF-algorithm, the G-algorithm and the H-algorithm. The BF-algorithm and the G-algorithm are both exact algorithms where the optimality of the solution is guaranteed. The BF-algorithm reduces the unaccountably infinite solution space to a finite solution space of equivalent classes. The BF-algorithm is intuitive; however, it is impractical due to its slow performance. The G-algorithm improved from the BF-algorithm so that invalid solution space is pruned off. The G-algorithm is practical for small instances. However, due to the complexity of the problem, the G-algorithm is impractical for large instances. On the other hand, the H-algorithm is proposed with the aim for fast performances where solutions are found in seconds even for large instances. Furthermore, the H-algorithm produces quality solutions for small to medium instances.

Finally, future works for this research is discussed. The E-algorithm further improves the G-algorithm by preventing redundant configurations that impose the same phylogenetic information. Relaxation problems to other phylogenetic models are also suggested for future investigation. Moreover, possible problems that assess the confidence of a phylogenetic tree are introduced.

REFERENCES

[1] Agarwala, R. and Fernandez-Baca, D., "A Polynomial-Time Algorithm for Perfect Phylogeny Problem when the Number of Character States is Fixed," SIAM Journal of Computing, 23(6): 1216-1224, 1994.

[2] Agarwala, R. and Fernandez-Baca, D., "Simple Algorithms for Perfect Phylogeny and Triangulating Colored Graphs," International Journal of Foundation of Computer Science, 7(1): 11-22, 1996.

[3] Altschul, S. F., The Statistic of Sequence Similarity Score, http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-html.

[4] Auyeung, A., "A New Phylogenetic Tree Model for Fuzzy Characters," In Proceedings of The International Conference on Information Technology, IEEE Computer Society, Las Vegas, Nevada, 2005.

[5] Auyeung, A. and Abraham, A., "Estimating Genome Reversal Distance by Genetic Algorithm," The IEEE Congress on Evolutionary Computation, pp. 1157-1161, 2003.

[6] Auyeung, A. and Abraham, A., "The Largest Compatible Subset Problem for Phylogenetic Data," The Genetic and Evolutionary Computation Conference, 2004.

[7] Auyeung, A. and Abraham, A., "The Role of Simulated Evolutions in Bioinformatics," ISE Book Series on Evolutionary Machine Design: Methodology and Applications, Nova Science Publishers, New York, 2004.

[8] Auyeung, A. and Melcher, U., "Evaluation of Protein Sequence Alignment using Structural Information," In Proceedings of The International Conference on Information Technology, IEEE Computer Society, Las Vegas, Nevada, 2005.

[9] Bain, L. J. and Engelhardt, M., Introduction to Probability and Mathematical Statistics 2nd Edition, Duxbury Thomson Learning, 1992.

[10] Barlow-Irick, P., Barriers to Making Order out of Chaos: Species with Fuzzy Boundaries, http://www.largocanyon.org/science/species/intro.htm.

[11] Berg, J. M., Tymoczko, J. L. and Stryer, L., Biochemistry 5th Edition, W.H. Freeman and Company New York, 2002.

[12] Bonet, M., Phillips, C., Warnow, T. J. and Yooseph, S., "Constructing Evolutionary Trees in the Presence of Polymorphic Characters," SIAM Journal of Computing, 29(1): 103-131, 1999.

[13] Buneman, P., "The Recovery of Trees from Measures of Dissimilarity ," In Mathematics in the Archaeological and Historical Sciences, F. R. Hodson, D. G. Kendall and P. Tautu (Eds), pp. 387-395. Edinburgh University Press, 1971.

[14] Chakraborty, R. and Nei, M., "Genetic Differentiation of Quantitative Characters between Populations or Species. I. Mutation and Random Drift," Genetical Research, 39: 303-314, 1982.

[15] Cormen, T. H., Leiserson, C. E. and Rivest, R. L., Introduction to Algorithms, McGraw-Hill Book Company, 2000.

[16] Day, W. and Sankoff, D., "Computational Complexity of Inferring phylogenies by Compatibility," Systematic Zoology, 35(2): 224-229, 1986.

[17] Diestel, R., Graph Theory, Springer Verlag, 2000.

[18] Durbin, R., Eddy, S., Krogh, A. and Mitchison, G., Biological Sequence Analysis – Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, 2002.

[19] Estabrook, G. F., Johnson, C. and McMorris, F. R., "A Mathematical Foundation for Analysis of Cladistic Character Compatibility," Math Bioscience, 29: 181-187, 1976.

[20] Estabrook, G. F. and McMorris, F. R., "When are Two Qualitative Taxonomic Characters Compatible," Journal of Mathematical Biology, 4: 195-200, 1977.

[21] Falconer, D. S. and Mackay, T. F., Introduction to Quantitative Genetics 4th Ed., Longman Scienctific and Technical, Harlow, U.K., 1996.

[22] Farris, J. S., "Methods for Computing Wagner Trees," Systematic Zoology, 19: 83-92, 1970.

[23] Felsenstein, J., Inferring Phylogenies, Sinauer Associates, Sunderland, Massachusetts, 2004.

[24] Fisher, R. A., "The Correlation between Relatives on the Supposition of Mendelian Inheritance," Transactions of the Royal Society of Edinburgh, 52: 399-433, 1918.

[25] Frank, H. and Althoen, S. C., Statistics Concepts and Applications, Cambridge University Press, 1994.

[26] Giannattasio, R. B. and Spooner, D. M., "A Reexamination of Species Boundaries between Solanum megistracolobum and S. toralapanum (Solanum sect. Petota, series Megistacroloba): Morphological Data," Systematic Botany, 19(1):89-105, 1994.

[27] Giannattasio, R. B. and Spooner, D. M., "A Reexamination of Species Boundaries and Hypothesis of Hybridization Concerning Solanum megistracolobum and S. toralapanum (Solanum sect. Petota, series Megistacroloba): Molecular Data," Systematic Botany, 19(1):106-115, 1994.

[28] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Professional, 1989.

[29] Goldberg, L. A., Goldberg, P. W., Phillips, C. A., Sweedyk, E. and Warnow, T., "Minimizing Phylogenetic Number to Find Good Evolutionary Tree," Discrete Applied Mathematics, 71: 111-136, 1996.

[30] Gusfield, D., "Efficient Algorithm for Inferring Evolutionary History," Networks, 21:19-28, 1991.

[31] Gusfield, D., Algorithm on Strings Trees, and Sequences – Computer Science and Computational Biology, Cambridge University Press, 1997.

[32] Halperin, E., "Improved Approximation Algorithm for the Vertex Cover Problem in Graphs and Hypergraphs," SIAM Journal of Computing, 31(5): 1608-1623, 2002.

[33] Hein, J. J., "An Optimal Algorithm to Reconstruct Trees from Additive Data," Bulletin of Mathematical Biology, 51(5): 597-603, 1989.

[34] Hopcroft, J. E., Motwani, R. and Ullman, J. D., Introduction to Automata Theory, Languages, and Computation, 2nd Edition, Addison Wesley, 2001.

[35] Horstmann, C. S. and Cornell, G., Core JAVA 2. Sun Microsystems, 1999.

[36] Kannan, S., Lawler, E. and Warnow, T., "Determining the Evolutionary Tree ," Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 475-484, 1990.

[37] Kannan, S. and Warnow, T., "A Fast Algorithm for the Computation and Enumeration of Perfect Phylogenies when the Number of Character States is Fixed," Proceeding of the Sixth Annual ACM of Symposium on Discrete Algorithms, 595-603, 1995.

[38] Kluge, A. G. and Farris, J. S., "Quantitative Phyletics and the Evolution of anurans," Systematic Zoology, 18: 1-32, 1969.

[39] Laarhoven, P. J. M. V. and Aarts, E. H. L. Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, 1987.

[40] Lande, R. 1976., "Natural Selection and Random Genetic Drift in Phenotypic Evolution," Evolution, 31: 442-336, 1976.

[41] Lynch, M. and Hill, W. G., "Phenotypic Evolution by Neutral Mutation," Evolution, 40: 915-935, 1986.

[42] Lynch, M. and Walsh, B., Genetics and Analysis of Quantitative Traits. Sinauer Associates, Sunderland, Massachusetts, 1998.

[43] Maddison, D. R., Schulz, K. S., Wheeler, T., Frumkin, J., Roy, R. and Maddison, W. P., The Tree of Life Web Project, http://tolweb.org/tree/.

[44] Mickevich, M. F. and Johnson, M. S., "Congruence between Morphological and Allozyme Data in Evolutionary Inference and Character Evolution," Systematic Zoology, 25: 260-270, 1976.

[45] Paschos, V. T., "A Survery of Approximately Optimal Solution to Some Covering and Packing Problems," ACM Computing Surveys, 29(2): pp. 171-209, 1997.

[46] Pedrycz, W. and Gomide, F., An Introduction to Fuzzy Sets: Analysis and Design, MIT Press, 1998.

[47] Pevzner, P. A., Computational Molecular Biology An Algorithmic Approach, MIT Press, 2001.

[48] Russell, P. J., iGenetics, Benjamin Cummings, 2002.

[49] Russell, S. J. and Norvig, P., Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall, 2002.

[50] Saccone, C. and Pesole, G., Handbook of Comparative Genomes, Principles and Methodology, Wiley-Liss, 2003.

[51] Sankoff, D., "Edit Distance for Genome Comparison based on Non-local Operations," The Third Annual Symposium on Combinatorial Pattern Matching, 644:121-135, 1992.

[52] Sankoff, D., Cedergren, R. and Abel, Y., "Genomic Divergence through Gene Rearrangement," Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences, 26:428-438, 1990.

[53] Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B. and Cedergren, R., "Gene Order Comparisons for Phylogenetic Inference: Evolution of the Mitochondrial Genome," Proceedings of the National Academy of Science USA, 89:6575-6579, 1992.

[54] Tao, J., Ying, X. and Zhang, M. Q., Current Topics in Computational Molecular Biology, MIT Press, 2002.

[55] Thiele, K. The Holy Grail of the perfect character: The Cladistics Treatment of morphometric data. Cladistics, 9:275-304. 1993.

[56] Wang, L., Zhang, K. and Zhang, L., "Perfectly Phylogenetic Networks with Recombination," Proceeding of the Sixteenth Annual ACM Symposium on Applied Computing, 46-50, 2001.

[57] West, D. B., Introduction to Graph Theory, Prentice Hall, 2000.

[58] Wheeler, Q. D. and Meier, R., Species Concepts and Phylogenetic Theory, Columbia University Press, 2000.

[59] Wiens, J. J., Phylogenetic Analysis of Morphological Data, Smithsonian Institution Press, 2000.

APPENDIX A

Table 8.3.1: Experimental results of the time analysis of the BF-algorithm. *m* is the number of objects and the time is recorded in seconds.

BF-algorithm	Positive	Positive Optimal	Negative
m = 4	0s	0s	0s
m = 6	0s	6629.75 s	8108.5 s
m = 8	345600+s	345600+s	345600+s

Table 8.3.2: Experimental results of the time analysis of the G-algorithm. *m* is the number of objects and the time is recorded in seconds.

G-algorithm	Positive	Positive Optimal	Negative
m = 4	0s	0s	0s
m = 6	4.25s	15.5 s	0s
m = 8	19.75s	415.33 s	1.25s
m =10	1913.25s	8000 s	49.25 s
m =12	44176 s	80000 s	9318 s

Table 8.3.3: Experimental results of the time analysis of the H-algorithm. *m* is the number of objects and the time is recorded in seconds.

H-algorithm	Positive	Negative
m = 5	0 s	0 s
m = 10	0 s	1 s
m = 15	1 s	1 s
m = 20	1.5 s	2 s
m = 25	2 s	4 s
m = 30	3.25 s	6 s

Table 8.3.4: Experimental results of the quality analysis of the H-algorithm. *m* is the number of objects and the quality is measured in percent *n'*/*n*.

H-algorithm	Positive
m = 5	100 %
m = 10	100 %
m = 15	67.86 %
m = 20	52.5 %
m = 25	45.83 %
m = 30	30 %

VITA

Andy (Winghang) Auyeung

Candidate for the Degree of

Doctor of Philosophy

Dissertation: A NEW PHYLOGENETIC TREE MODEL FOR FUZZY CHARACTERS

Major Field: Computer Science

Biographical:

Personal Data: born in Hong Kong, China on October 3, 1977.

Education: Received Bachelor of Science in Computer Science from the University of Central Oklahoma in May 2000. Completed the Requirements for the Doctor of Philosophy with a major in Computer Science at Oklahoma State University in May 2005.

Experience: Employed by Oklahoma State University, Department of Computer Science as a graduate assistant, 2000-2004.

Name: Andy (Winghang) Auyeung Date of Degree: May, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: A NEW PHYLOGENETIC TREE MODEL FOR FUZZY CHARACTERS

Pages in Study: 95 Candidate for the Degree of Doctor of Philosophy

Major Field: Computer Science

Abstract: The study of phylogenetics involves the identification of evolutionary relationships between species. Perfect phylogeny is one of the classical character-based models. In this model, there are only two states for each character: zero, where the character is absent; and one, where it is present. All previous perfect phylogeny based models assume the states of the characters are discrete. However, fuzzy boundaries between species and degrees of character development are commonly found in nature. These phenomena show the need for a more relaxed model. This dissertation proposes the fuzzy (perfect) phylogeny model that extends the perfect phylogeny model to allow fuzzy memberships of the characters. In this dissertation, the fuzzy phylogeny model is studied, including its properties and algorithmic solution. Due to the fuzziness of characters, two relaxation problems for the fuzzy phylogeny are proposed, namely the Adjustment problem for Fuzzy Phylogeny (AFP problem) and the optimal AFP problem. However, the problems are proven to be **NP-hard**. So, two super-polynomial algorithms, namely, the BF-algorithm and the G-algorithm, are introduced to provide exact solutions to the (optimal) AFP problem. Moreover, to provide a practical solution to the (optimal) AFP problem, a sub-optimal algorithm called the H-algorithm is also introduced. Finally, future work for this research is discussed. That includes improvement on algorithms, other relaxation problems, and formulations of problems that assess the confidence of a phylogenetic tree.

ADVISER'S APPROVAL: Blayne E. Mayfield