

M.Sc. Engg. Thesis

CONSTRUCTING PHYLOGENETIC TREES USING QUARTET-BASED METHODS

By
Rezwana Reaz

Submitted to
Department of Computer Science and Engineering
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology (BUET)
Dhaka-1000

June 26, 2013

The thesis titled “**CONSTRUCTING PHYLOGENETIC TREES USING QUARTET-BASED METHODS**”, submitted by Rezwana Reaz, Roll No. 0411052006P, Session April 2011, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on June 26, 2013.

Board of Examiners

1. _____
Dr. M. Sohel Rahman
Associate Professor
Department of CSE
BUET, Dhaka 1000.
Chairman
(Supervisor)
2. _____
Dr. Abu Sayed Md. Latiful Hoque
Professor & Head
Department of CSE
BUET, Dhaka 1000.
Member
(Ex-officio)
3. _____
Dr. M. Kaykobad
Professor
Department of CSE
BUET, Dhaka 1000.
Member
4. _____
Dr. A.B.M. Alim Al Islam
Assistant Professor
Department of CSE
BUET, Dhaka 1000.
Member
5. _____
Dr. Mohammad Nurul Huda
Professor
Department of CSE
United International University, Bangladesh.
Member
(External)

Candidate's Declaration

This is to certify that the work presented in this thesis entitled “**CONSTRUCTING PHYLOGENETIC TREES USING QUARTET-BASED METHODS**” is the outcome of the investigation carried out by me under the supervision of Associate Professor Dr. M. Sohel Rahman in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology (BUET), Dhaka. It is also declared that neither this thesis nor any part thereof has been submitted or is being currently submitted anywhere else for the award of any degree or diploma.

Rezwana Reaz
Candidate

Contents

<i>Board of Examiners</i>	iii
<i>Candidate's Declaration</i>	iv
Acknowledgements	xiv
Abstract	xv
1 Introduction	1
1.1 Phylogenetic Tree	2
1.2 Applications of Phylogenies	3
1.3 Overview of Phylogenetic Tree Reconstruction Methods	5
1.3.1 Sequence Based Method	6
1.3.2 Distance Based Methods	7
1.3.3 Supertree Method	8
1.4 Motivation Behind Quartet Based Supertree Methods	10
1.5 Literature Review	11
1.6 Objective of This Thesis	13
1.7 Main Contribution and Result Summary	14
1.8 Thesis Organization	15
2 Preliminaries	16
2.1 Phylogenetic Tree Basics	16
2.1.1 Rooted Phylogenetic Tree	16

2.1.2	Unrooted Phylogenetic Tree	17
2.1.3	Rooting and Unrooting Trees	18
2.1.4	Newick Representation	19
2.1.5	Clades and Star	21
2.2	Quartet Trees	22
2.2.1	Quartet Consistency	22
2.2.2	Tree Bipartition	23
2.2.3	Supertrees and Quartet based supertree methods	25
2.3	Algorithms and Complexity	27
2.3.1	Big- <i>O</i> Notation	27
2.3.2	Polynomial Algorithms	28
2.3.3	Complexity Classes	28
2.3.4	Heuristic Algorithms	30
2.3.5	Divide and Conquer Algorithms	30
2.4	Problem Definition	31
2.5	Summary	32
3	Quartet Based Phylogenetic Tree Reconstruction	33
3.1	Algorithm QFM	33
3.1.1	The Divide and Conquer Approach	34
3.1.2	Pseudocode of QFM	35
3.2	Method of Bipartition	36
3.2.1	Algorithmic Components of MFM	36
3.2.2	Algorithm MFM	38
3.3	Variations of Algorithm MFM	40
3.3.1	Algorithm MFM - Ia	40
3.3.2	Algorithm MFM - Ib	41
3.3.3	Algorithm MFM - IIa	42
3.3.4	Algorithm MFM - IIb	44
3.3.5	Algorithm MFM - IIIa	44

3.3.6	Algorithm MFM - IIIb	46
3.4	Time Complexity	47
3.5	Summary	48
4	Experimental Study	57
4.1	Measure of Accuracy	57
4.2	Design of Simulation Study	58
4.3	Experiment on Simulated Data	60
4.3.1	Simulated Data Generation	60
4.3.2	Part I: Comparison of the Performance of Six Approaches of QFM	62
4.3.3	Part II: Comparison of QFM and QMC in Accurate Tree Estimation	73
4.4	Experiment on Biological Data	83
4.5	Summary	84
5	Conclusion & Future Work	86
A	QTREE: A Simulation Tool for Quartet Based Phylogeny	88
A.1	User Manual	89
B	Performance Curves of Six Approaches of QFM vs QMC	95
C	Experimental Running Time	114

List of Figures

1.1	Illustration of a phylogenetic tree.	3
2.1	An example of binary and non-binary rooted trees.	17
2.2	An example of binary and non-binary unrooted trees.	17
2.3	Unrooting a rooted tree.	18
2.4	Generating rooted tree from an unrooted tree.	19
2.5	Illustration of newick expression of a tree.	20
2.6	An example of clades in a tree.	21
2.7	An example of a star tree.	21
2.8	Three possible unrooted trees over four taxa.	22
2.9	Five possible rooting of the unrooted tree $((A, D), (B, C))$ by taking an edge as root.	23
2.10	Two possible rooting of the unrooted tree $((A, D), (B, C))$ by taking an internal node as root.	24
2.11	Different Newick expressions for the same quarter. Arrow denotes the choice of rooting position.	24
2.12	Quartet consistency with a tree T	25
2.13	Example of satisfied, violated and deferred quartets with respect to a partition.	26
3.1	Divide and conquer approach.	50
3.2	QFM (Q uartet F M Algorithm)	51
3.3	An example iteration of the Bipartition Algorithm MFM. The locked taxa are shown in circles.	52

3.4	MFM (M odified F iduccia M attheyses B ipartition A lgorithm)	53
3.5	Pseudocode for Procedure Initial Partition - I	54
3.6	Pseudocode for Procedure Initial Partition - II	55
3.7	Pseudocode for Procedure Initial Partition - III	56
4.1	An example quartet generation from a tree $((A, B), F), (E, (D, C))$	59
4.2	Top: Missing branch rates of QFM - IIIa and QMC. Bottom: Illustration for the proportion of datasets on which the performance of QFM is better than, worse than or equal to QMC.	76
4.3	Missing branch rates of QFM and QMC on different datasets (with different number of taxa, number of quartets and the consistency level). From top to bottom: the number quartets are $n^{1.5}$, n^2 , and $n^{2.8}$, where n is the number of taxa. From left to right: 70%, 80% and 90% of the input quartets are consistent with the model species tree. We did not run our method on more than 100 taxa when the number of taxa is $n^{2.8}$ (since these are computationally intensive and could not be run within a reasonable time limit. Moreover, this model condition is less revealing since both QMC and QFM can reconstruct the true species trees on these datasets.)	81
4.4	Illustration for the proportion of datasets on which the performance of QFM is better than, worse than or equal to QMC. From top to bottom: charts are drawn for Table 4.11, Table 4.12 and Table 4.13.	82
4.5	The 25 species bird phylogeny estimated by QFM using the 227,700 embedded quartets from 18 gene trees.	85
B.1	Comparison of QFM - Ia and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	96
B.2	Comparison of QFM - Ia and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	97
B.3	Comparison of QFM - Ia and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	98

B.4	Comparison of QFM - Ib and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	99
B.5	Comparison of QFM - Ib and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	100
B.6	Comparison of QFM - Ib and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	101
B.7	Comparison of QFM - IIa and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	102
B.8	Comparison of QFM - IIa and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	103
B.9	Comparison of QFM - IIa and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	104
B.10	Comparison of QFM - IIb and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	105
B.11	Comparison of QFM - IIb and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	106
B.12	Comparison of QFM - IIb and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	107
B.13	Comparison of QFM - IIIa and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	108
B.14	Comparison of QFM - IIIa and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	109
B.15	Comparison of QFM - IIIa and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	110
B.16	Comparison of QFM - IIIb and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	111
B.17	Comparison of QFM - IIIb and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)	112

B.18 Comparison of QFM - IIIb and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$
(top), $c = 80\%$ (middle) and $c = 90\%$ (bottom) 113

List of Tables

3.1	The log table corresponding to the iteration shown in Figure 3.3	40
3.2	The log table corresponding to the next iteration of the iteration shown in Figure 3.3	40
4.1	Comparison of QFM (Ia & Ib) to QMC at $c = 90\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).	63
4.2	Comparison of QFM (Ia & Ib) to QMC at $c = 80\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).	64
4.3	Comparison of QFM (Ia & Ib) to QMC at $c = 70\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets(#Quartets).	66
4.4	Comparison of QFM (IIa & IIb) to QMC at $c = 90\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).	67
4.5	Comparison of QFM (IIa & IIb) to QMC at $c = 80\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).	68
4.6	Comparison of QFM (IIa & IIb) to QMC at $c = 70\%$ in number of satisfied quartets. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).	70

4.7	Comparison of QFM (IIIa & IIIb) to QMC at $c = 90\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	71
4.8	Comparison of QFM (IIIa & IIIb) to QMC at $c = 80\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	72
4.9	Comparison of QFM (IIIa & IIIb) to QMC at $c = 70\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	74
4.10	Comparison of QFM - IIIa to QMC in FN rate, FP rate and RF rate at $c = 90\%$ for the dataset used in [32]. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	75
4.11	Comparison of QFM - IIIa to QMC at $c = 70\%$ in FN rate, FP rate and RF rate. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	78
4.12	Comparison of QFM - IIIa to QMC at $c = 80\%$ in FN rate, FP rate and RF rate. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	79
4.13	Comparison of QFM - IIIa to QMC at $c = 90\%$ in FN rate, FP rate and RF rate. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).	80
C.1	Running time of QFM - IIIa for different datasets	116

Acknowledgements

First of all, I would like to declare that all the appraisals belong to the Almighty **ALLAH**.

I would like to express my deep gratitude to my supervisor Associate Professor Dr. M. Sohel Rahman for introducing me to the fascinating and prospective field of phylogenetics. I have learned from him how to carry on a research work, how to write, speak and present well. I thank him for his patience in reviewing my so many inferior drafts, for correcting my proofs and language, suggesting new ways of thinking, leading to the right way, and encouraging me to continue my research work. I again express my indebtedness, sincere gratitude and profound respect to him for his continuous guidance, suggestions and whole hearted supervision throughout the progress of this work. I also express my deepest gratitude to all the members of my defense board for their valuable comments and suggestions.

I convey my heartfelt reverence to my parents and other family members for giving their best support throughout my work to overcome the tedium of repetitive trials to new findings. I am grateful to Md. Shamsuzzoha Bayzid for sharing his knowledge on phylogenetics and providing some computational resources. I also thank Dr. Sagi Snir for providing the code for Quartet MaxCut and the data generation tool. Furthermore, I extend my appreciation to my friends for their tremendous support.

Finally, every honor and every victory on earth is due to Allah, descended from Him and must be ascribed to Him. He has endowed me with good health and with the capability to complete this work. I deeply express my sincere gratitude to the endless kindness of Allah.

Abstract

A phylogenetic tree represents the evolutionary relationship among a group of species. The ‘quartet-based’ phylogenetic tree construction refers to the method of combining many ‘quartets’ (a phylogenetic tree relating 4 species) into a single phylogenetic tree. In this thesis we present a new algorithm for ‘quartet-based’ phylogenetic tree construction and show its superiority in accuracy over the current best method for this problem.

Phylogenetic tree construction methods are inherently computationally very intensive, and usually can be applied to limited number of species. But the ultimate goal of phylogenetic reconstruction is to infer the phylogeny involving all lives on earth, i.e., to infer the ‘Tree of Life’. The ‘supertree’ method (constructing larger tree(s) from many smaller trees) has been identified as a reasonable solution in this regard, as these methods are computationally less intensive compared to other existing phylogenetic tree construction methods (such as maximum likelihood). Over the past decade, supertree construction has become an area of active theoretical and practical research. A ‘quartet’ is the the basic piece of phylogenetic information, so the quartet-based supertree method is responsible for combining many minimal pieces of information into a single, coherent, and more comprehensive piece of information. Since the quartet-based supertree methods are inherently computationally less intensive compared to the other approaches of phylogenetic tree construction, the only challenge of such construction is to achieve the accuracy and scalability.

In this thesis, we have devised a new quartet based supertree method, QFM (Quartet FM), which constructs more accurate trees (in terms of topological accuracy) than the current best quartet based method, QMC (Quartet MaxCut) [32]. The new method is also scalable to large datasets, that is, it performs well on very large datasets without compromising the accuracy.

Also, QFM is found performing same as (even better in some cases) QMC in maximizing the objective function of the underlying optimization problem. In this thesis, we performed an extensive experimental study to evaluate the accuracy and scalability of our algorithm on both simulated and biological datasets. In addition, we have developed a software tool, named, 'QTREE' to simulate and analyze the two methods - QFM and QMC.

Chapter 1

Introduction

The study of evolution is fundamental to the investigation of a wide array of biological questions. For example, estimates of the evolutionary history of sets of molecular sequences are used in biomedical research, including drug and vaccine development [2], in tracking the origins and development of humans over time [15], and even as forensic evidence in the investigation of criminal acts [23]. One of the most ambitious goals in evolution is to discover the relationships among all the species on Earth, the *Tree of Life*. The field responsible for this undertaking is phylogenetics.

Scientific euphoria has recently centered on the reconstruction of evolutionary relationships among different species. It is argued that all life currently on earth is descended from a single common ancestor. Over a period of at least 3.8 billion years, that single original ancestor has split repeatedly into new and independent lineages, i.e., species. On occasions, some of these independent lineages have come back together to form yet other lineages or to exchange genetic information. The evolutionary relationships among these species are referred to as “Phylogeny”, and phylogenetic reconstruction is concerned with inferring the phylogeny of groups of organisms. These relationships are expressed as a tree known as phylogenetic tree.

1.1 Phylogenetic Tree

A phylogeny is the written representation of the evolutionary relationships of a set of organisms. One of the simplest forms of phylogenies are phylogenetic trees, or trees. A phylogenetic tree $T = (V, E)$ is a connected acyclic graph of a set of vertices V and a set of undirected edges E connecting the vertices. A leaf (or node with degree one) in V represents a *taxon* (species, gene etc.), that typically exists in the present day. An internal node (with degree greater than one) in V constitutes an (hypothetical) ancestral taxon from which some descendent taxa evolved. If all internal nodes in a tree have degree at most three, then the tree is referred to as *binary*. Otherwise, the tree is non-binary, and has some node with degree greater than three, also known as a *polytomy*. An edge $e = (u, v) \in E$ represents an evolutionary relationship between the two taxa (plural for taxon) at the vertices u and v connected by e . An edge is synonymously referred to as a *bipartition* since its removal splits the tree T into two connected components (and consequently the set of taxa into two disjoint subsets). The phylogeny problem is to reconstruct the evolutionary history of a set S of taxa where S is the set of leaf nodes. The phylogeny problem can be formally derived as follow.

The Phylogeny Problem

Input: A set S of taxa.

Output: A tree T leaf-labeled by S , such that T represents the evolutionary relationship among the members of S .

Figure 1.1 illustrates an example of a phylogenetic tree (borrowed from a presentation titled *Introduction to Phylogenetic Estimation Algorithms* [39]). The tree is rooted at the most recent common ancestor of the five existing species represented by the five leaves. Other internal nodes represent hypothesized or known ancestors. The common practice today is to use biomolecular sequences as representatives of the species set; which is why the nodes of this tree are labeled by DNA sequences. Other than the molecular sequences, morphological data (e.g., color, size, weight etc.), molecular markers (Single Nucleotide Polymorphism (*SNP*), haplotypes etc.), and gene order and content can be used as the representatives of the species set. These are

commonly known as *character data*.

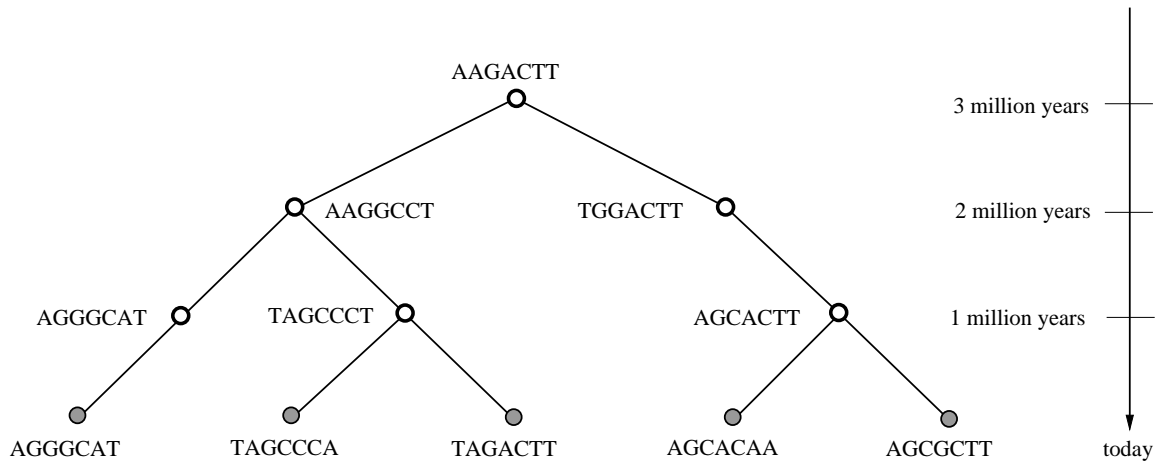


Figure 1.1: A phylogenetic tree (figure borrowed from [39]).

1.2 Applications of Phylogenies

One of the great challenges of science is reconstructing the *Tree of Life*, which is the evolutionary history of all organisms on Earth. The implication of this grand phylogeny - that all living things on Earth today (from bacteria, to seaweeds, to mushrooms, to humans) are related - has forever changed our perception of the world around us. Moreover, the uses of phylogenies, beyond elucidating the evolutionary relationships of biological species, are many and growing. Phylogenies have become an integral part of biological research, including biomedical research, drug design and, areas of bioinformatics (such as protein structure prediction and multiple sequence alignment). In this section, we give a brief overview of applications of phylogeny. The description below are mostly adopted from [19].

- Study of Evolution:** Phylogenies play a major role in the interpretation of information on all characteristics of organisms, from structure and physiology to genomics. Phylogeny reflects the history of transmission of life's genetic information, and hence organizes our knowledge of diverse organisms, genomes, and molecules. At the species level, a phylogeny
 - provides hypotheses about the derivation of traits and the circumstances behind

such derivation. Thus plays a vital role in studies of adaptation and evolutionary constraints [9, 18, 20, 21, 22].

- informs the dynamics of speciation and, to some extent, extinction - the two forces that generate and reduce biodiversity [3, 12].
- **Comparative study:** The most common use of phylogeny is for comparative study [1, 14]. A comparative study is one where a particular question is addressed by comparing how certain biological characters have evolved in different lineages in the context of a phylogeny. This information is used to infer important aspects of the evolution of those characters.
- **Biogeographic hypothesis:** Another common use of phylogenies is to test biogeographic hypothesis. Biogeography is concerned with the geographical distribution of organisms, extant and extinct. For example, a researcher may be interested in whether a particular species have colonized a set of islands a single time or repeatedly. This can be assessed by determining whether all of the species on the island arose from a single most recent mainland common ancestor or whether they are multiple independent mainland species.
- **Inference of amino acids:** One can also use a phylogeny to attempt to infer the amino acid sequence of extinct proteins. This putative extinct proteins can then be synthesized or an artificial gene coding for them can be produced, and the functional characteristics of the proteins that are of interest can be tested.
- **Evolution of diseases:** In a more practical vein, phylogenies can be used to track the evolution of diseases, which can, in turn, be used to design drugs and vaccines that are more likely to be effective against the currently dominant strains. The most prominent example of this use is the flu vaccine, which is altered from year to year as medical experts work to keep track of the influenza types most likely to dominate in a given flu season [2].
- **Biological Investigation:** Finally, phylogenies have even been used in criminal cases

where the phylogenetic evidence plays a prominent role in the trial.

In summary, phylogenies are useful in any endeavor where the historical and hierarchical structure of the evolution of species can be used to infer the history of the point of interest.

1.3 Overview of Phylogenetic Tree Reconstruction Methods

Since the evolutionary history is at best partially known, biologists, mathematicians, and computer scientists have designed a variety of criteria and methods for their accurate reconstruction. Before we discuss the various phylogenetic reconstruction methods, we first briefly describe two types of data that are used to represent the input to those methods, and models of sequence evolution.

Input Data

Early approaches of phylogenetic reconstruction were based on morphological characters of species (such as physical characteristics or biological functions) as data, but with the advancement of molecular biology, researchers developed methods based on molecular (amino acid and nucleotide) sequence data. Molecular sequence data can be used in two ways: character data and distance data. These two are the most common types that are used in phylogenetic analysis.

- **Character Data:** Qualitative characters in biomolecular sequences are the single positions within multiple alignments, and they have a fixed number of states (4 for DNA and RNA, 20 for amino-acids). Phylogenetic reconstruction methods based on character data take as input a matrix $M_{n \times k}$ of n species and k characters, so that each species $s \in S$ is represented by a vector in Z^k . Thus, M_{ij} is the state of character j for species s_i , where $S = \{s_1, \dots, s_n\}$. The output of the method is a phylogeny leaf-labeled by S , and whose internal nodes are also labeled by vectors in Z^k .
- **Distance Data:** Some phylogenetic reconstruction methods take “distance” matrices

as a representation of the input. A distance matrix $M_{n \times n}$ is a symmetric matrix that indicates the pairwise distances between taxa; M_{ij} represents the distance between the two taxa i and j . Distance matrices are usually computed from the character data of the input. Those matrices have $M_{ii} = 0$ for every $1 \leq i \leq n$, but do not necessarily satisfy the triangle inequality.

Phylogenetic tree estimation techniques from molecular sequence can be divided into three high-level types, namely, sequence based, distance based and topology based. We now briefly describe the techniques of these three broad categories.

1.3.1 Sequence Based Method

In sequence based methods, the input is a set of homologous sequences from different species. The method builds a phylogeny that tries to represent the evolutionary history of these sequences. The most promising sequence based methods are maximum parsimony and maximum likelihood methods.

Maximum Parsimony

Maximum Parsimony (MP) [10] is an optimization problem based on the minimum evolution principal¹. A maximum parsimony tree is one that minimizes the number of changes over the edges of the tree needed to “explain” the sequences at its leaves.

Parsimony score of a tree: Let $T = (V, E)$ is a tree with a set of vertices V and a set of undirected edges E . The leaves of T are labeled by a set of sequences S , each of length ℓ . An **extension** f of S on T is a labeling of all vertices of T by sequences of length ℓ that maintains the original leaf-labeling by S . For two sequences x and y , let $H(x, y)$ be the Hamming distance between x and y ². Then the parsimony score of an extension f , denoted by, $score(f, T)$, is $\sum_{(u,v) \in E} H(f(u), f(v))$, where $f(u)$ and $f(v)$ denote the labeling of u and v in f . A **minimal extension** is one that minimizes the parsimony score, and the parsimony score of a minimal

¹The best evolutionary trees are the ones that minimize the number of changes along the edges of the tree.

²For two sequences $x = x_1x_2 \dots x_l$ and $y = y_1y_2 \dots y_l$, $H(x, y) = |\{i : x_i \neq y_i\}|$

extension is the parsimony length of the tree T .

A **maximum parsimony tree** for a given set of sequences S is a tree that has the smallest possible parsimony length of any tree leaf labeled by S . The **parsimony problem** is to find a tree of minimum parsimony cost. This is an NP-hard problem even when the sequences are binary (i.e., the alphabet size is two) [7, 11]. Heuristic searches are used to find a solution of this problem.

Maximum Likelihood

Maximum Likelihood (ML) [8] is also an optimization problem, and seeks the most probable tree according to some specified model of evolution. Given a model of sequence evolution, a maximum likelihood tree is a tree topology, along with a set of branch lengths and parameters for the given model of evolution, that maximizes the conditional probability of observing the sequences at the leaves given the assumed model. Again, let S be a set of sequences of equal length. The maximum likelihood solution for the set S is an edge weighted tree (T, w) , together with a set of model parameters \mathcal{M} , that minimizes $P_r(S | T, w, \mathcal{M})$. As with MP, ML is also NP-hard [4], and heuristic searches are used to compute ML trees.

1.3.2 Distance Based Methods

Distance-matrix methods of phylogenetic analysis explicitly rely on a measure of “genetic distance” between the sequences being classified. Therefore, they require an MSA (multiple sequence alignment) as an input. Distance is often defined as the fraction of mismatches at aligned positions, with gaps either ignored or counted as mismatches [24]. Distance methods attempt to construct an all-to-all matrix from the sequence query set describing the distance between each sequence pair. From this, a phylogenetic tree is constructed that places closely related sequences under the same interior node and whose branch lengths closely reproduce the observed distances between sequences. Distance-matrix methods may produce either rooted or unrooted trees, depending on the algorithm used to compute them.

Neighbor Joining

A popular distance based method is neighbor joining [28]. Neighbor joining takes as input a distance matrix specifying the distance between each pair of taxa (i.e., aligned sequences). The algorithm starts with a completely unresolved tree, whose topology corresponds to that of a star network, and iterates over the following steps until the tree is completely resolved and all branch lengths are known:

- Based on the current distance matrix calculate the matrix Q . Based on a distance matrix relating the n taxa, Q is calculated as follows:

$$Q(i, j) = (n - 2)d(i, j) - \sum_{k=1}^n d(i, k) - \sum_{k=1}^n d(j, k),$$

where $d(i, j)$ is the distance between taxa i and j .

- Find the pair of taxa for which $Q(i, j)$ has its lowest value. Add a new node to the tree, joining these taxa to the rest of the tree.
- Calculate the distance from each of the taxa in the pair to this new node.
- Calculate the distance from each of the taxa outside of this pair to the new node.
- Start the algorithm again, replacing the pair of joined neighbors with the new node and using the distances calculated in the previous step.

1.3.3 Supertree Method

Although the sequence based methods construct quite accurate trees on small to moderate sized datasets, in many cases accuracy decreases as the input size increases or when such a set of homologues³ for the set of species under study does not exist. Topology based methods can perform better than sequence based methods in such cases. The task of the topology-based

³Homology is shared evolutionary history, and genes are called homologous if they descended from a common ancestor gene.

reconstruction method is to build many overlapping small trees with very accurate sequence-based methods and later amalgamate these into a big complete tree that represents each of the input small trees. By the term ‘overlapping’ we mean that the taxa sets corresponding to the small trees are not disjoint. This method is also known as the ‘supertree’ method. The supertree construction technique addresses the problem of ‘Tree Compatibility’ which is an *NP*-hard problem [19].

Tree Compatibility Problem

Input: Set $T = \{T_1, T_2, \dots, T_k\}$ of trees on sets S_1, S_2, \dots, S_k , respectively.

Output: Tree \mathbf{T} , if it exists, such that for each i , $\mathbf{T}|_{S_i}$ refines T_i ⁴.

The construction of a supertree scales exponentially with the number of taxa included; therefore for a tree of any reasonable size it is not possible to examine every possible supertree and weigh its success at combining the input information. Therefore, heuristic methods are essential for supertree reconstruction. Let S be a set of n taxa (i.e., n sequences). The supertree methods follow the steps below.

- **Step 1:** Division of the taxa set S into required number of overlapping subsets.
- **Step 2:** Application of any sequence based or distance based method on each subset to compute small trees.
- **Step 3:** Application of heuristics to combine the small trees to get a single tree.

Matrix Representation Parsimony (MRP) [26]

Matrix representation parsimony (MRP) [26] is the most popular supertree method. MRP is a general supertree method, that is the overlapping small trees are either rooted or unrooted and can have any number of taxa (species). It uses MP to analyze the data matrix created from input trees. As MRP involves solving an NP hard problem (i.e., MP), it is not efficient for large datasets and naturally its accuracy decreases with increased input size.

⁴ \mathbf{T} is said to refine T_i if T_i can be obtained from \mathbf{T} by a sequence of edge-contractions.

Quartet Based Supertree Method

When all the small trees are unrooted and have 4 taxa, a supertree method is called a ‘quartet-based’ supertree method. Here a ‘quartet’ is an unrooted tree relating 4 species (taxa). Since a ‘quartet’ is the most basic piece of phylogenetic information, the quartet-based method is responsible for combining many minimal pieces of information into a single, coherent, and more comprehensive piece of information. An accurate and efficient quartet-based method can overcome the scalability problem of MRP and provide at least as accurate result as MRP. This is why significant attention has been given in the relevant literature for devising efficient and accurate quartet based methods.

In this thesis, we focus on quartet based supertree method. In particular, we concentrate on developing a heuristic approach for combining small trees.

1.4 Motivation Behind Quartet Based Supertree Methods

To realize the usefulness of quartet based supertree method, first we have to realize the advantages of supertree methods. Supertree methods are beneficial over other approaches of phylogenetic tree estimation in the following cases.

- 1) Summarizing the results of available studies on (subsets of) a particular group of interest when access to the sequences is not possible.
- 2) Producing a tree from disparate data-types, such as molecular, morphological, and gene-order data, that require independent types of analysis (and therefore prohibit a combined analysis).
- 3) Analyzing large datasets that would take too long to analyze using other phylogenetic reconstruction methods.

The most crucial role of supertree method is in inferring the *Tree of Life*. In a typical molecular phylogenetic analysis, a tree on a particular set of species is constructed by first

collecting the DNA or protein sequences for a homologous gene in each species, and then using those sequences to construct a tree on that set of species. Using this sort of process to construct the Tree of Life is not feasible, since one particular gene from each species may not provide sufficient information to derive the evolutionary relationship. So collecting the sequences from multiple genes from each species is required. A supertree method provides a solution for such a case by constructing a tree on each gene dataset separately and then combining those trees into a single tree on the entire set of species. Supertrees are presently a necessary tool for many phylogenetic problems. In fact, most, if not all, detailed estimates of the Tree of Life (e.g. the Tree of Life Web Project; <http://tolweb.org/tree/phylogeny.html>) to date have been constructed using a supertree approach. For this reason there is an increasing interest among the research community for studying these methods.

A supertree method can reconstruct a correct tree when the generated input small trees are correct, i.e., when the input small trees represent true evolutionary relations among their leaves. Now let us discuss how the quartet based supertree methods are useful. We need accurate (as much as possible) small trees for accurate reconstruction of the supertree. But sometimes, depending on data, accurate reconstruction of small trees may not be feasible. For example, suppose that the input sequences have some missing information. In such cases, generation of a small tree of a moderate size may not be done accurately due to missing information. But as we have already noted, a quartet carries the basic piece of phylogenetic information. So small trees relating only 4 taxa can still be inferred accurately for such data. When the small trees are accurate then we can expect (reasonably) accurate reconstruction of the supertree depending on the underlying heuristic used to combine the small trees. So the quartet based supertree methods are getting significant research attention compared to other supertree methods.

1.5 Literature Review

Quartet based phylogenetic tree reconstruction has been receiving extensive attention in the literature for more than two decades. Different approaches have been proposed and improved time to time. Among these the most prominent approaches are: quartet quizzing (QP), quartet

joining (QJ) and quartet max-cut (QMC).

Quartet puzzling (QP) [36] infers the phylogeny of n sequences in three steps as follows. First, the maximum-likelihood step uses the maximum-likelihood principle to weight all possible 4-trees. Then, based on these weights, the puzzling step constructs a large number of n -trees. And finally the consensus step computes the consensus tree of these n -trees. TREE-PUZZLE [30] is a program package that implements QP. The *puzzling step* has been modified in [35] by assigning each quartet q_i a weight w_i . In this approach the Bayesian probability p_i of each quartet q_i is computed using the maximum likelihood value of the three associated 4-trees. Strimmer et al. [35] proposed three different ways to use these probabilities to weight 4-trees. In the continuous case, the probabilities are directly used as weights, i.e., $w_i = p_i$. In the binary (unweighted) case, the weight of the 4-tree with highest probability is set to 1, and the weights of the two others are set to 0. In the discrete case, the three w_i 's are discrete, least-squares approximations of the p_i 's. QP with Bayesian weighting scheme was an improvement over the original QP in recovering a true tree [35]. Ranwez and Gascuel [25] proposed weight optimization (WO), a new algorithm which is also based on weighted 4-trees inferred by using the maximum likelihood approach. WO searches for the tree on n taxa such that the sum of the weights of the 4-trees induced by this tree is maximal. WO is faster and offers better theoretical guarantee than QP but is less efficient than traditional phylogenetic reconstruction approaches based on pairwise evolutionary distances or maximum likelihood [25].

Quartet joining (QJ) [40] concentrates on reconstructing reliable phylogenetic trees while tolerating as many quartet errors as possible. This is achieved by carefully selecting two possible neighbor leaves to merge and assigning weights intelligently to the quartets that contain newly merged leaves. Theoretically, if the input quartet set is completely consistent with evolutionary tree T , the quartet-joining algorithm will reconstruct the exact evolutionary tree T . QJ outperforms QP on real datasets. On average, the performance of QJ and NJ [28] are very close but QJ outperforms NJ on quartet sets with low quartet consistency [40].

In 2007, Snir et al. [33] proposed a novel quartet-based method called *short quartet puzzling* (SQP), which gives better estimates of the true tree topology by comparison to both NJ

and MP and QP. It differs from the previous techniques in that it does not consider all possible 4-trees while constructing the output tree and rather considers only a subset of all possible 4-trees as input. This is a two phase technique: the first phase uses the randomized technique for selecting input quartets from all possible 4-trees, and the second phase uses Quartet Max Cut (QMC) [33, 31] technique for amalgamating quartet trees together. The 4-trees are estimated using ML. QMC seeks to find a tree on the full dataset satisfying a maximum number of quartets. It is a divide and conquer approach that repeatedly bipartitions the quartet set by taking the maxcut of the graph constructed based on input quartet sets. Swenson et al. [37] performed an experimental study of QMC and other supertree methods. In this experimental study, source trees were generated using five different encoding techniques. Two QMC-based supertree methods, QMC (All) and QMC (Exp+TSQ), differing only in how the source trees are encoded, produced more accurate supertrees than MRP (in many cases) and the other supertree methods (in many cases) for the smaller (100-taxon and 500-taxon) datasets. But MRP outperforms all QMC methods on the largest (1000-taxon) datasets. In [32], Snir and Rao presented a fast and scalable implementation of QMC.

1.6 Objective of This Thesis

In the study of phylogenetic tree reconstruction, two questions are vital. These are: 1) Is the reconstruction method results in accurate trees? 2) Is the method scalable to (i.e., performs equally on) large datasets (i.e., several hundreds of species)?. The existing best phylogenetic tree reconstruction methods, other than the supertree methods, usually get stuck in the question number 2, since those methods are computationally very intensive to be applied to very large datasets. Supertree methods are usually scalable to large datasets. The best (in terms of accuracy) supertree method to date is MRP [26], though it is not time and space efficient for very large datasets. Quartet based supertree methods can be considered to be a solution to the problem of scalability to large datasets. QMC [33, 31, 32] is the current best quartet best supertree method. It defeats MRP in terms of running time; however it does not always return as accurate tree as MRP does.

In this thesis, our objective is to find out an accurate and scalable quartet based supertree reconstruction method, and also to challenge QMC in terms of accuracy and scalability. In addition, we aim to develop a software interface implementing our method and QMC, so that bioinformaticians can compare and analyze the results of these methods for different datasets easily without having any prior knowledge about how these algorithms work.

1.7 Main Contribution and Result Summary

In this thesis, we address the problem of constructing phylogenetic trees using quartet based method. Our contributions and the main results are summarized below.

1. We propose a new heuristic algorithm (QFM) for reconstructing a supertree from quartets, which follows a divide and conquer based strategy.
2. We compare our method with the best known quartet based method, QMC, by performing an intensive experimental study on simulated datasets.
3. Our experimental results suggest that QFM provides results at least as good as (even better) than QMC in maximizing the optimization criteria (see Section 2.4).
4. To measure the topological accuracy of the estimated trees, both QFM and QMC have been applied to the simulated datasets, generated from a model tree. In most of the cases, the tree estimated by QFM than the tree returned by QMC, is more close to the model tree.
5. QFM, when applied to small to large sized datasets (containing several hundreds of taxa, several lacs of quartets) is found satisfactorily scalable.
6. The accuracy of QFM has also been measured in this thesis by applying it on biological dataset. QFM returns the phylogenetic tree that matches the tree anticipated by biologists for the set of taxa under consideration.

7. Finally as part of this thesis we have developed a software interface ‘**QTREE**’ implementing both QFM and QMC for the use of bioinformatics practitioners interested in phylogenetic analysis and experiments.

1.8 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2 we discuss the relevant ideas and necessary definitions from phylogeny and algorithm theory to understand our research work. Chapter 3 describes our heuristic algorithm QFM for quartet based supertree reconstruction. Chapter 4 deals with our experimental works - data generation, experimental setup, result summary and detailed analysis on results. Finally, We conclude in Chapter 5 with some future directions.

Chapter 2

Preliminaries

In this chapter, we define some basic concepts and terminology related to phylogeny and algorithm theory. Definitions that are not included in this chapter will be introduced as they are needed. We start in Section 2.1, by describing basic concepts related to phylogenetic trees - tree structure, representation etc. Then in Section 2.2 we define a quartet - its structure, representation and other concepts related to quartet based phylogeny. The concepts related to algorithm theory are discussed in Section 2.3. Finally, in Section 2.4 we define the optimization problem that we are addressing in this thesis.

2.1 Phylogenetic Tree Basics

Phylogenetic trees can be either rooted or unrooted with leaves labeled by extant taxa (species). Internal vertices (hypothetical ancestors) are usually unlabeled and have degree at least three.

2.1.1 Rooted Phylogenetic Tree

A *rooted* phylogenetic tree is a *directed tree* with a unique node corresponding to the (usually imputed) most recent common ancestor of all the entities at the leaves of the tree. The node representing the most recent common ancestor is called the root (having degree two or more), from which a unique path leads to any other node. A rooted phylogenetic tree is binary if all internal vertices have degree three, except the root, which has degree two. A rooted

phylogenetic tree is non-binary if any internal node has degree greater than three. Figure 2.1 shows an example of binary (tree at left) and non-binary (tree at right) rooted trees.

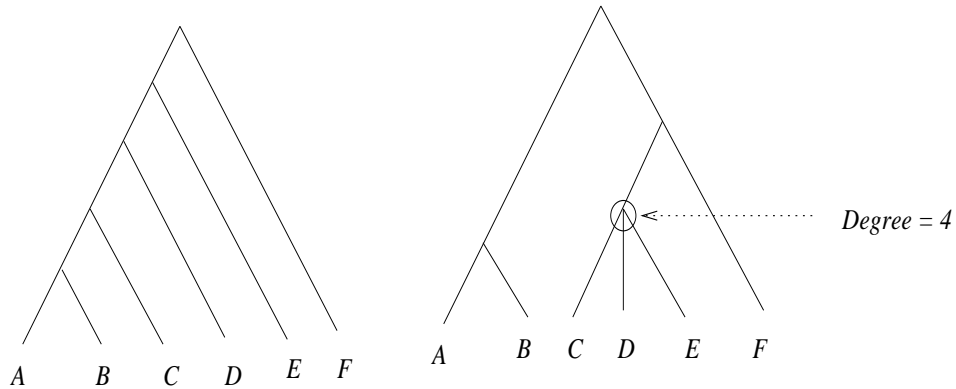


Figure 2.1: An example of binary and non-binary rooted trees.

2.1.2 Unrooted Phylogenetic Tree

An *unrooted* phylogenetic tree illustrates the relatedness of the leaf nodes without making assumptions about ancestry at all. In an unrooted tree the lines represent evolutionary lineages, but unlike a rooted tree, we do not know which way evolution preceded along the lineage. An unrooted phylogenetic tree is binary if all internal vertices have degree three; otherwise the tree is non-binary. Figure 2.2 shows an example of binary and non-binary unrooted trees.

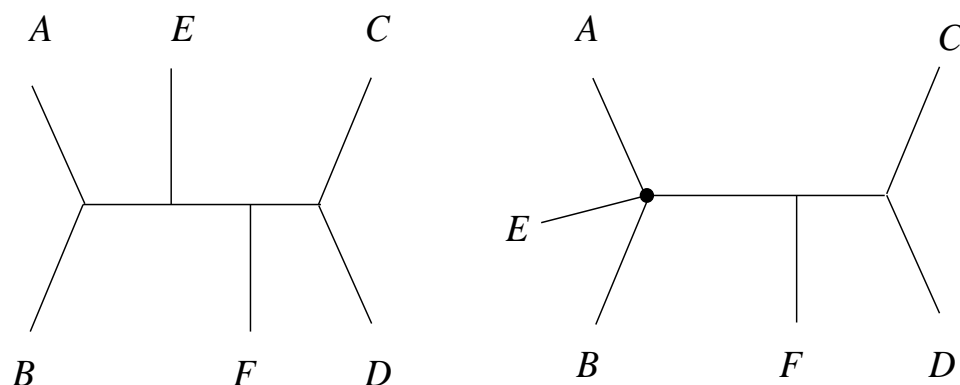


Figure 2.2: An example of binary and non-binary unrooted trees.

2.1.3 Rooting and Unrooting Trees

An unrooted tree can always be generated from a rooted tree by simply omitting the root. In Figure 2.3, T_1 is a rooted tree, an unrooted tree T_2 is generated by discarding the root vertex R and then joining the two children of R by an edge.

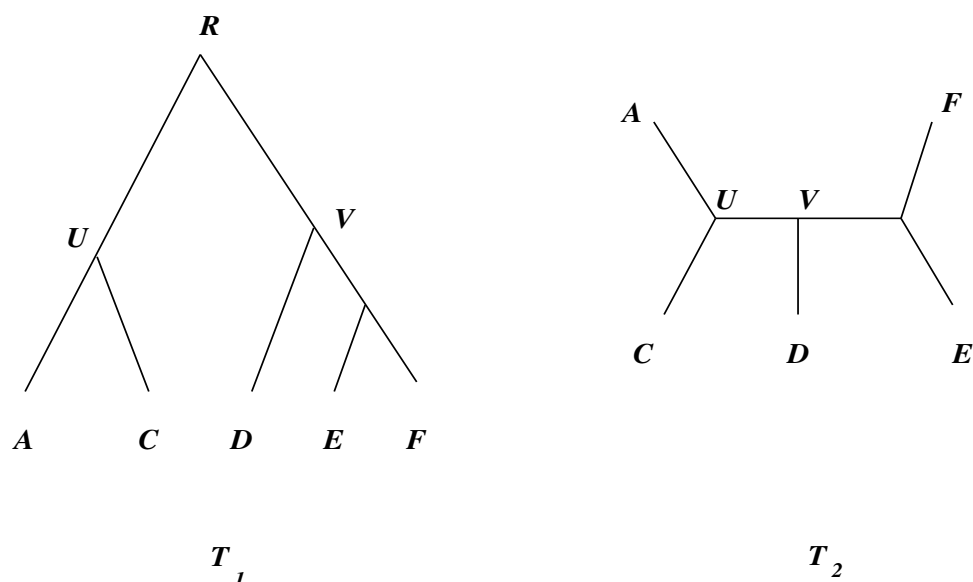


Figure 2.3: Unrooting a rooted tree.

We can generate a rooted tree T_r from an unrooted tree T_u in two ways.

- Rooting at an internal node: We can select any internal node of T_u as a root for T_r . In this way number of possible rooted trees from an unrooted tree equals the number of internal nodes in the unrooted tree.
- Rooting at an edge: We can select any edge e of T_u and replace it by a randomly labeled root node r . Then connect r with the two endpoints of e . In this way number of possible rooted trees from an unrooted tree equals the number of edges in the unrooted tree.

Figure 2.4 shows an example of generating rooted trees T_r from an unrooted tree T_u . However the selection of the internal node or the edge as the root is not done at random. To select a root we need some means of identifying ancestry. Accurately rooting a phylogenetic tree is a

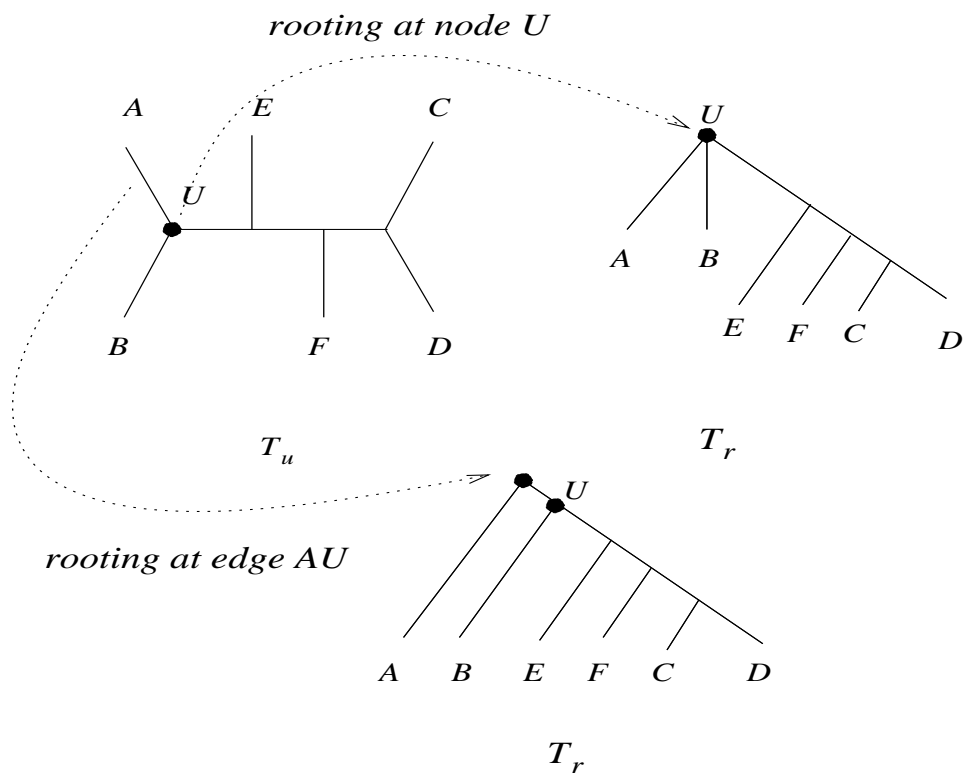


Figure 2.4: Generating rooted tree from an unrooted tree.

complex problem requiring specific knowledge of the set of taxa being studied or the assumption of a molecular clock¹. If the molecular data used to reconstruct a phylogeny are assumed to have evolved at a constant rate over time, then one can root the tree based on estimated leaf-to-leaf distances. This assumption is often violated in real datasets. In mathematical (and computational) phylogenetics, therefore, our goal is to reconstruct only the unrooted version of the rooted tree that represents the evolutionary relationships among the taxa.

2.1.4 Newick Representation

In mathematics, Newick tree format (or Newick notation or New Hampshire tree format) is a way of representing graph-theoretical trees. The Newick Standard for representing trees in

¹The molecular clock is a technique in molecular evolution that uses fossil constraints and rates of molecular change to deduce the time in geologic history when two species or other taxa diverged. It is used to estimate the time of occurrence of events called speciation.

computer-readable form makes use of the correspondence between trees and nested parentheses, noticed in 1857 by the famous English mathematician Arthur Cayley. In this format we express a rooted phylogenetic tree T as “(left(T), right(T))”, where $left(T)$ and $right(T)$ are the children of the root of T . If $left(T)$ and $right(T)$ are internal nodes, then we replace them by “(left(left(T)), right(left(T)))” and “(left(right(T)), right(right(T)))”, respectively. If an internal node has more than two children then they are expressed in left to right order. Figure 2.5 illustrates the formation of the Newick expression of a rooted tree. An important character-

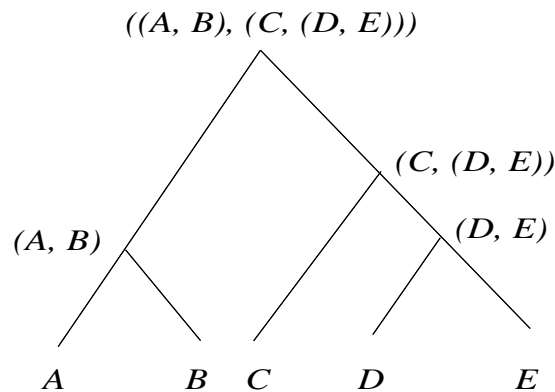


Figure 2.5: Illustration of newick expression of a tree.

istics of this representation is that from a biological point of view it does not make a unique representation of a tree. There are two reasons for this.

- *First*, the left to right order of descendants of a node affects the representation, even though it is biologically not interesting. Thus, to a biologist $(A, (B, C), D)$ is the same tree as $(A, (C, B), D)$.
- *Second*, the standard is for representing a rooted tree. For an unrooted tree we do not have the root. For expressing an unrooted tree, the convention is simply to arbitrarily root the tree and report the resulting rooted tree. So an unrooted tree can have several Newick representations corresponding to its different rooted versions.

2.1.5 Clades and Star

A clade in a tree is a maximal set of leaves that all have the same most recent common ancestor. To generate all the clades in a rooted tree, we need to look at each node in turn, and write down the leaves below that node. Thus, for the tree $T = ((A, B), (C, (D, E)))$ (Figure 2.6), the set of clades of the tree T , denoted $Clades(T)$, is given by $\{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}, \{A, B\}, \{D, E\}, \{C, D, E\}, \{A, B, C, D, E\}\}$.

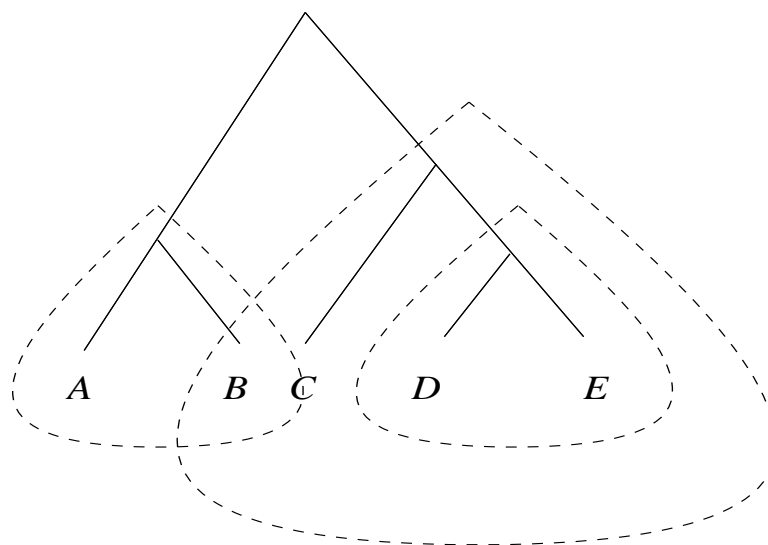


Figure 2.6: An example of clades in a tree.

In graph theory, a *star* S_k is the complete bipartite graph $K_{1,k}$: a tree with one internal node and k leaves (but, no internal nodes and $k + 1$ leaves when $k \leq 1$). A tree T over taxa set P is a *star*, if T has only one internal node and there is an edge from the internal node incident to each taxon $t \in P$. We also refer to such a tree as a *depth one tree*. Figure 2.7 shows an example of a star.

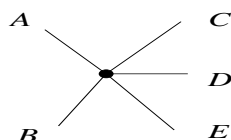


Figure 2.7: An example of a star tree.

2.2 Quartet Trees

A quartet q is an unrooted binary phylogenetic tree over 4 taxa. For a taxa set A, B, C, D of size 4, there are three possible unrooted binary phylogenetic trees, as shown in Figure 2.8.

A quartet has five edges. So a particular quartet can be rooted in five ways, in each case choosing an edge as the root. Figure 2.9 shows all five rootings of the second unrooted tree in Figure 2.8 considering the edges as root. Similarly, Figure 2.10 shows the two possible rooted trees considering the two internal nodes of quartet $((A, D), (B, C))$ as root.

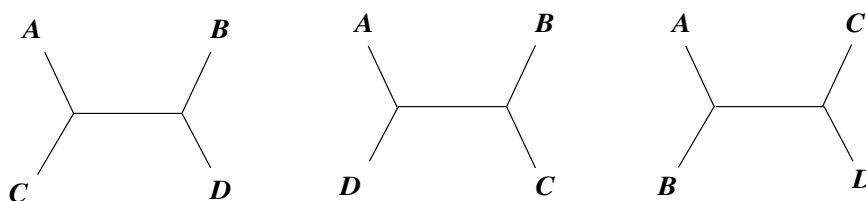


Figure 2.8: Three possible unrooted trees over four taxa.

In Newick format, a quartet is usually represented as $((A, B), (C, D))$ (the rightmost quartet in Figure 2.8). This expression means that there is an edge separating A and B from C and D (note that this does not mean that A is closer to B than to C). The same quartet could have been written as $((B, A), (D, C))$, or $((C, D), (A, B))$, etc, since swapping the siblings does not change the tree topology. Also $(C, D, (A, B))$ and $(A, B, (C, D))$ correspond to the Newick expressions for the rooted trees which are generated from the same unrooted tree $((A, B), (C, D))$ (Figure 2.11) by taking its internal nodes as root respectively. Since an unrooted tree is expressed in Newick format by first considering any rooted version of the tree and then write down the Newick expression for the rooted version, so a single quartet has several Newick representations.

2.2.1 Quartet Consistency

A quartet $((A, B), (C, D))$ is consistent with a tree T if in T , there is an edge (or path in general) separating A and B from C and D . For any four taxa, only one (out of 3 possible quartets) will be consistent with a tree T . In Figure 2.12 among the three quartets, quartet 1

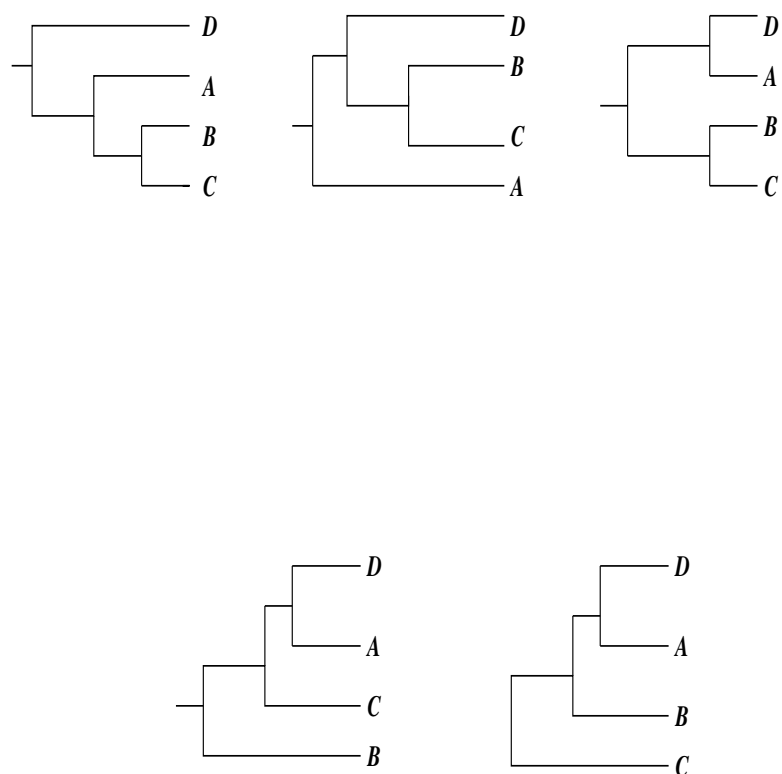


Figure 2.9: Five possible rootings of the unrooted tree $((A, D), (B, C))$ by taking an edge as root.

is consistent with tree T as there exists an edge in T such that it separates A and C from B and D . Other two quartets are inconsistent with T as no such edge exists in T .

2.2.2 Tree Bipartition

A bipartition of an unrooted tree T is formed by taking any edge in T , and writing down the two sets of taxa that would be formed by deleting that edge. Note that when the edge is incident to a taxon, then the bipartition is trivial - it splits the set of taxa into one set with a single taxon, and the other set with the remaining taxa. These bipartitions are present in all trees with any given taxa set. Hence, we will focus just on the non-trivial bipartitions. That is, we will consider only the internal edges (not incident to a taxon) for a bipartition.

Consider T be a tree over the taxa set P . Now, if we take an internal edge e belongs to edge set of T and delete e , then we get two subtrees, namely, T_a and T_b . Let P_a and P_b be the

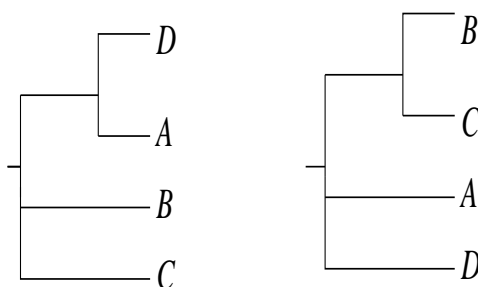


Figure 2.10: Two possible rootings of the unrooted tree $((A, D), (B, C))$ by taking an internal node as root.

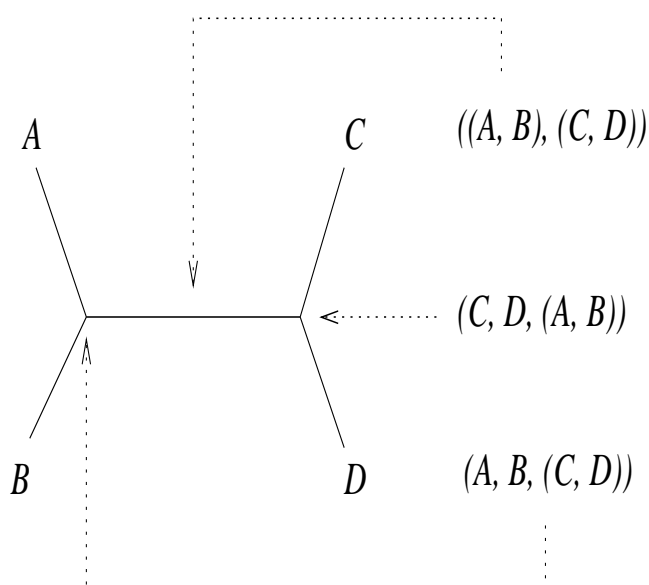
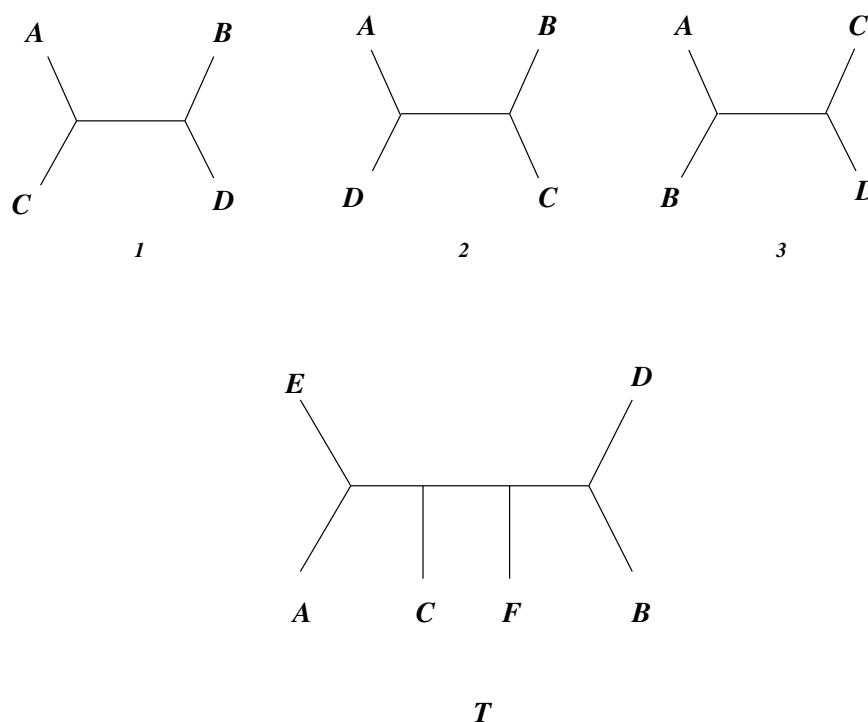


Figure 2.11: Different Newick expressions for the same quartet. Arrow denotes the choice of rooting position.

taxa sets of T_a and T_b respectively. We shall denote a bipartition as (P_a, P_b) . Thus an internal edge in T corresponds to a bipartition of P .

Let $q = ((A, B), (C, D))$ be a quartet, where $A, B, C, D \in P$. If we want to test the consistency of q with respect to T , we need to find out a bipartition (P_a, P_b) such that $A, B \in P_a$ and $C, D \in P_b$ or $A, B \in P_b$ and $C, D \in P_a$. Now, we will define the status of a quartet q with respect to a given bipartition (P_a, P_b) .

A quartet $q = ((A, B), (C, D))$ is **satisfied** with respect to a bipartition (P_a, P_b) if taxa A and B reside in one part and taxa C and D reside in the other. A **satisfied** quartet is

Figure 2.12: Quartet consistency with a tree T .

consistent with T .

A quartet $q = ((A, B), (C, D))$ is **violated** with respect to a bipartition (P_a, P_b) if taxa A and C (or A and D) reside in one part and taxa B and D (or B and C) reside in the other part.

A quartet $q = ((A, B), (C, D))$ is **deferred** with respect to a bipartition (P_a, P_b) if any three of its four taxa reside in one part and the fourth one in the other part.

In Figure 2.13, Q is the input quartet set, P is the taxa set and T is the resultant tree. We consider a bipartition of the taxa as $\{1, 2, 3\}$ and $\{4, 5, 6\}$. This bipartition corresponds to the *dotted* edge in T . With respect to this partition q_2 , q_5 and q_6 are satisfied, q_1 , q_3 and q_4 are deferred and q_7 is violated.

2.2.3 Supertrees and Quartet based supertree methods

Let $\mathbf{T} = \{T_1, T_2, \dots, T_n\}$ be a set of n phylogenetic trees. Consider, P_1, P_2, \dots, P_n be the taxa sets of T_1, T_2, \dots, T_n respectively and $\mathbf{P} = \{P_1 \cup P_2 \cup \dots \cup P_n\}$. A supertree method will be

Q	
$q_1 : ((1, 2), (3, 4))$	$q_2 : ((1, 2), (5, 6))$
$q_3 : ((1, 3), (2, 4))$	$q_4 : ((3, 4), (5, 6))$
$q_5 : ((2, 3), (4, 5))$	$q_6 : ((1, 3), (5, 6))$
$q_7 : ((1, 4), (2, 6))$	
$P = \{1, 2, 3, 4, 5, 6\}$	
$P_a = \{1, 2, 3\}$	$P_b = \{4, 5, 6\}$

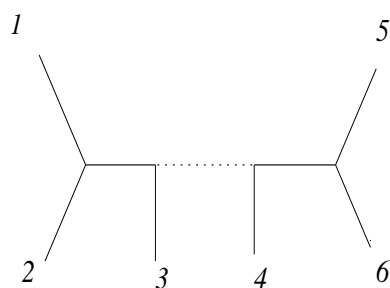
 T

Figure 2.13: Example of satisfied, violated and deferred quartets with respect to a partition.

defined as any function that takes as input a collection \mathbf{T} of phylogenetic trees, and returns a tree, or a collection of trees, with taxa set \mathbf{P} . The input trees to supertree methods are called **source trees**, and a tree in the output is often referred to as a **supertree**. If all the source trees are quartets, then a supertree method is referred to as a *quartet based supertree method*.

Steps of Quartet Based Supertree Reconstruction

Given a set S of n aligned molecular sequences of n species. Let, P be the taxa set containing these n species. The quartet method has following three steps:

- **Step 1 [Selection of Subsets]:** The method first selects a set P' of subsets of size 4 from P . The subsets can be overlapping, i.e., one taxon may exist in multiple 4-element subsets. But union of all subsets in P' must return P .

- **Step 2 [Estimation of Source Trees]:** For each subset in P' , a quartet is estimated by analyzing the 4 sequences (taken from S) of the corresponding 4 species. The analysis is done using any sequence based method (for example, maximum likelihood method). All these estimated quartets makes the quartet set Q .
- **Step 3 [Inference of Supertree]:** An amalgamation method is applied to combine the collection of the quartet trees.

2.3 Algorithms and Complexity

In this section, we briefly introduce some terminologies related to the theory of algorithms and complexity. For a more in-depth knowledge, the readers are referred to refer to [13, 6, 16].

2.3.1 Big- O Notation

The most widely accepted complexity measure for an algorithm is the *running time* which is expressed by the number of operations it performs before producing the final answer. Instead of reporting that an algorithm takes, say, $5n^3 + 4n + 3$ steps on an input of size n , it is much simpler to leave out lower-order terms such as $4n$ and 3 (which become insignificant as n grows), and even the detail of the coefficient 5 in the leading term (computers will be five times faster in a few years anyway), and just say that the algorithm takes time $O(n^3)$ (pronounced “big oh of n^3 ”). The reason behind such simplification is that we are often interested only in the “asymptotic behavior”, that is, the behavior of the algorithm, when applied to very large inputs, which is insensitive to constant factors and low order terms. We now define this notation precisely. Let $f(n)$ and $g(n)$ are the functions from the positive integers to the positive reals, then we write $f(n) = O(g(n))$ (which means that $f(n)$ grows no faster than $g(n)$) if there exists positive constants c_1 and c_2 such that $f(n) \leq c_1g(n) + c_2$ for all n .

This cavalier attitude toward constants in case of big- O notation may seem very rude since programmers and algorithm developers are very interested in constants and give tremendous effort in order to make an algorithm run faster even by a factor of 2. But understanding and

analyzing algorithms at theoretical level would be impossible without the simplicity afforded by big- O notation.

2.3.2 Polynomial Algorithms

An algorithm is said to be *polynomially bounded* (or simply *polynomial*) if its complexity is bounded by a polynomial of the size of a problem instance. Examples of such complexities are $O(n)$, $O(n \log n)$, $O(n^{100})$, etc. An algorithm that is guaranteed to terminate within a number of steps which is an exponential function of the size of the problem, is called *exponential or nonpolynomial* algorithm. Suppose, we need to check every binary number of n digits to find a solution of a problem, so the complexity is $O(2^n)$. Now if we add an extra digit, we must check two times as many numbers. In contrast, when the running time of an algorithm is bounded by $O(n)$, we call it a *linear-time* algorithm or simply a *linear* algorithm.

2.3.3 Complexity Classes

We first introduce some important concepts. *Decision problems* refer to the algorithmic questions that can be answered by ‘yes’ or ‘no’.

A *deterministic algorithm* is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states. All computers that exist exist now, run deterministically. Deterministic algorithms can be defined in terms of a state machine: a state describes what a machine is doing at a particular instant in time. Let, just after we enter the input, the machine is in its initial state or start state. A deterministic algorithm uniquely determines at most one next state from all possible choices of next states. In contrast, a *nondeterministic algorithm* is one which determines many states as the next state simultaneously. We may regard a nondeterministic algorithm as having the capability of branching off into many copies of itself, one for the each next state. Thus, while a deterministic algorithm must explore a set of alternatives one at a time, a nondeterministic algorithm examines all alternatives at the same time.

A problem P_1 is *polynomially reducible* to problem P_2 ($P_1 \leq_p P_2$) if there exists a polynomial

time algorithm that transforms every instance I_1 of P_1 to an instance I_2 of P_2 such that the answer to I_1 is “yes” ($I_1 \in P_1$) if and only if the answer to I_2 is “yes” ($I_2 \in P_2$).

The Class P

P is the class of problems that can be solved by deterministic polynomial time algorithm. This implies that there is a deterministic algorithm that takes as input an instance I and has a running time polynomial in I such that if I has a solution, the algorithm returns such a solution; and if I has no solution, the algorithm correctly reports so. Clearly $P \subseteq NP$. But the question, “ $P = NP$?” is still unresolved. It is widely believed that $P \neq NP$. However, proving this has turned out to be extremely difficult, one of the deepest and most important unsolved puzzles of mathematics.

The Class NP

NP is the class of problems – solutions of which can be verified deterministically in polynomial time. This means that there is an efficient (low-order polynomial) deterministic checking algorithm C that takes as input the given instance I (the data specifying the problem to be solved), as well as the proposed solution S , and outputs true if and only if S really is a solution to instance I . Moreover the running time of $C(I, S)$ is bounded by a polynomial in $|I|$. We can also define NP as the class of decision problems that can be solved nondeterministically in polynomial time, which is why NP stands for “nondeterministic polynomial time.”

The class NP -complete and NP -hard

A problem p is NP -complete if it satisfies the following two conditions.

1. $p \in NP$.
2. For every problem $p' \in NP$, $p' \leq_p p$.

A problem satisfying condition 2 is said to be NP -hard, whether or not it satisfies condition 1. NP -complete problems are considered to be the hardest problems in NP . These problems have the following interesting properties.

- (a) No *NP*-complete problem can be solved by any known polynomial algorithm.
- (b) If there is a polynomial algorithm for any *NP*-complete problem, then there are polynomial algorithms for all *NP*-complete problems.

2.3.4 Heuristic Algorithms

In computer science, a heuristic algorithm, or simply a heuristic, is an algorithm that is able to produce an acceptable solution to a problem in many practical scenarios, but for which there is no formal proof of its correctness. Alternatively, it may be correct, but may not be proven to produce an optimal solution, or to use reasonable resources. Heuristics are typically used when there is no known method to find an optimal solution, under the given constraints (of time, space etc.) or at all. These algorithms, usually find a solution close to the best one and they find it fast and easily. Sometimes these algorithms can be accurate, that is they actually find the best solution, but the algorithm is still called heuristic until this best solution is proven to be the best. The method used for a heuristic algorithm is one of the known methods, such as greediness, but in order to be easy and fast the algorithm ignores or even suppresses some of the problem's demands. Heuristics rely on ingenuity, intuition, a good understanding of the application and meticulous experimentation to attack a problem [6].

2.3.5 Divide and Conquer Algorithms

In computer science, divide and conquer is an important algorithm design paradigm based on multi-branched recursion. The divide-and-conquer strategy solves a problem by: 1) breaking it into subproblems that are themselves smaller instances of the same type of problem, 2) recursively solving these subproblems, and 3) appropriately combining their answers. The real work is done piecemeal, in three different places: in the partitioning of problems into subproblems; at the very tail end of the recursion, when the subproblems are so small that they are solved outright; and in the gluing together of partial answers. These are held together and coordinated by the algorithm's core recursive structure [6]. This technique is the basis of

efficient algorithms for numerous problems, such as sorting (e.g., quicksort, merge sort).

2.4 Problem Definition

In this thesis, we focus on the inference of a supertree from a given set of quartets. In this section, we will define the optimization problem that we address in reconstructing supertree from a set of quartets. Before we formally define the problem, we first discuss two important concepts, *quartet compatibility* and *resolved quartet*.

Consider a set $Q = \{q_1, q_2, \dots, q_n\}$, where q_i is a quartet over a set, p_i , of 4 taxa. Q is said to be *compatible* if there is a phylogenetic tree T on $P = p_1 \cup p_2 \cup \dots \cup p_n$, such that each $q_i \in Q$ is consistent with T . In this case, the quartet set Q and the tree T are said to be *consistent*. An quartet is called *resolved* if every internal node has degree three [34].

Suppose we are given a set Q of resolved quartets on P , where P is a set of n taxa. For given a tree T , consistency of T with Q can be verified by checking each resolved quartet in Q against T . The *Quartet Compatibility* Problem is the problem of determining the existence of a phylogeny T on P that satisfies all the quartets. This is an *NP*-complete [34] problem.

Quartet-based supertree reconstruction addresses the problem of *Maximum Quartet Consistency* (MQC), which is a natural optimization problem. This problem takes a quartet set Q as input and finds a phylogenetic tree T such that *maximum* number of quartets in Q become consistent with T (or T satisfies maximum number of quartets). Now we formally define the optimization problem that we address in this thesis.

Problem 1 *Maximum Quartet Consistency.*

Input: A set of quartets Q on a taxa set P .

Output: A phylogenetic tree T on P such that T satisfies the maximum number of quartets of Q .

The Maximum Quartet Consistency (MQC) problem represents an NP-hard optimization problem [34]. Existing approaches for the MQC problem can be categorized as either heuristic or

exact. Exact methods such as exhaustive searches or branch and bound algorithms are prohibitively time consuming for large datasets. They are applicable in inferring phylogenetic tree for a limited number of taxa. Therefore, for any substantial problem (hundreds of taxa) we are forced to rely on heuristics to guide a limited search of tree space in the hope of finding good (optimal or near optimal) trees. The focus of the thesis is on heuristic solutions for the MQC problem as we aim to build the phylogenetic tree for several hundreds of taxa.

2.5 Summary

In this chapter, we have introduced some basic concepts and terminology related to phylogeny and algorithm theory. In Section 2.1, we described the basic concepts related to phylogenetic trees - tree structure, representation etc. Then in Section 2.2 we defined a quartet - its structure, representation and other concepts related to quartet based phylogeny. The concepts related to algorithm theory were discussed in Section 2.3. Finally, in Section 2.4 we have defined the optimization problem that we are addressing in this thesis.

Chapter 3

Quartet Based Phylogenetic Tree Reconstruction

In this chapter we give our heuristic algorithm of constructing a phylogenetic tree from a set of input quartets. Our quartet based reconstruction technique follows a divide and conquer approach. The divide and conquer approach uses a bipartition technique which is inspired by the famous Fiduccia Mattheyses (FM) bipartition technique [5]. So we call our algorithm **Quartet FM (QFM)** algorithm. In Section 3.1 we present the overall divide and conquer approach. Then we describe our bipartition technique in Section 3.2 and its time complexity in Section 3.4.

3.1 Algorithm QFM

Let, Q be a set of quartets over a set of taxa, P . Algorithm QFM aims to construct a tree T , satisfying the largest number of input quartets possible. The divide and conquer approach recursively creates bipartition of the taxa set p , where each bipartition corresponds to an internal edge in the tree under construction. We shall describe the bipartition algorithm in Section 3.2.

3.1.1 The Divide and Conquer Approach

Divide Step: At each recursive step, we partition the taxa set P into two sets P_a and P_b . After the algorithm partitions the taxa set, it augments each part with a uniquely denoted dummy (artificial) taxon. This taxon will play a role while returning from recursion. After the addition of the dummy taxon to the sets P_a and P_b , we subdivide the quartet set Q into two sets, Q_a and Q_b . A quartet set Q_i takes those quartets $((a, b), (c, d))$ from Q such that either all four taxa a, b, c and d or any three of the four taxa belong to P_i . That is, a quartet $q \in Q$, such that q is satisfied or violated, is not considered to be included in Q_a or Q_b . Moreover, in every deferred quartet (three taxa are in the same part), the other taxon is renamed by the name of the dummy taxon and the quartet continues to the next step (each such taxon in all deferred quartets is renamed to the same name). Thus we get, two (Q_i, P_i) pairs corresponding to each bipartition. We then recurse on both pairs (Q_a, P_a) and (Q_b, P_b) . The recursion ends when quartet set Q is empty or $|P| < 3$. We then return a depth 1 tree (star) over the taxa set P .

Conquer Step: On return from the recursion, at each step, we have two trees, T_1 (corresponding to (Q_a, P_a)) and T_2 (corresponding to (Q_b, P_b)). These two trees are rerooted such that the dummy taxon serves as root in each tree. Then the dummy taxon is removed from each tree and the two roots are joined by an internal edge. Thus we get the merged tree T where each bipartition corresponds to an internal edge.

Figure 3.1 describes the high level divide and conquer algorithm. Let Q be the input quartet set and P be the corresponding taxa set. $Q = \{((1, 2), (3, 4)), ((1, 3), (2, 4)), ((2, 3), (4, 5)), ((1, 2), (5, 6)), ((3, 4), (5, 6)), ((1, 3), (5, 6))\}$. Hence, $P = \{1, 2, 3, 4, 5, 6\}$. P is bipartitioned into two sets, $P_a = \{1, 2, 3, X\}$ and $P_b = \{4, 5, 6, X\}$. Here, X is a dummy taxon. The way this bipartition is returned will be illustrated for this pair of (P, Q) in Section 3.2. The bipartition (P_a, P_b) satisfies quartets $q_3 : ((2, 3), (4, 5))$, $q_4 : ((1, 2), (5, 6))$ and $q_6 : ((1, 3), (5, 6))$ from Q . So these quartets will not be considered in the next level. Q_a takes $q_1 : ((1, 2), (3, 4))$ and $q_2 : ((1, 3), (2, 4))$ as three of the taxa of q_1 and q_2 reside in P_a . We replace the taxon which does not belong to P_a with the dummy taxon X . Hence we get $Q_a = \{((1, 2), (3, X)), ((1, 3), (2, X))\}$.

Similarly we get $Q_b = \{((X, 4), (5, 6))\}$. Both P_a and P_b have been partitioned further into (P_{a_a}, P_{a_b}) and (P_{b_a}, P_{b_b}) . The partition (P_{a_a}, P_{a_b}) satisfies $((1, 2), (3, X))$ and violates $((1, 3), (2, X))$ of Q_a and (P_{b_a}, P_{b_b}) satisfies the only quartet of Q_b . So the quartet sets for the next level are empty. In such case, no more recursion is required. We return a *star* for each of the taxa sets $P_{a_a}, P_{a_b}, P_{b_a}$ and P_{b_b} . The returned trees are merged by removing the dummy taxon of that level and joining the branches of dummy taxa. In Figure 3.1, the upper half shows the *divide* steps. The *depth one trees* are returned when no more recursion is required. The lower half of Figure 3.1 shows how the trees are returned and merged upon return from recursion. Thus we get the final merged tree $((((1, 2), 3), (4, (5, 6))))$ (at bottom) satisfying 5 quartets in total. The satisfied quartets are q_1, q_3, q_4, q_5 and q_6 (Figure 3.1).

3.1.2 Pseudocode of QFM

The pseudocode of Algorithm QFM is given in Figure 3.2. This is a recursive algorithm which takes a taxaset P and a quartet set Q as input. The recursion terminates when either of the following conditions gets true.

- Cardinality of P is less than or equal to 3
- Q is empty.

The algorithm uses five procedures, namely, DEPTH_ONE_TREE, MFM, POPULATE, RE-ROOT and MERGE. These procedures work as follows:

- DEPTH_ONE_TREE(P): It returns a star T over the taxa set P .
- MFM(P, Q): It takes taxaset P and quartet set Q as input and returns a bipartition of the taxa set. The underlying algorithm is described in Section 3.2.
- POPULATE($P_i, Q, dummy$): It takes one part (let, P_i) of the bipartitioned taxa set as input. Also it takes the quartet set Q and an artificial taxon *dummy* as input. The function of this procedure is to populate the quartet set Q_i corresponding to P_i . It checks

each quartet $q \in Q$ and decides whether to insert it on Q_i based on the following conditions.

- If all four taxa of q belong to P_i , then add q to Q_i .
- If any three of the four taxa of q belong to P_i , then replace the taxon which does not belong to P_i with *dummy* and add the resultant quartet to Q_i .
- RERoot($T_i, dummy$): The tree T_i is rerooted such that the artificial taxon *dummy* serves as the root of the resultant tree T'_i .
- MERGE($T_i, T_j, dummy$): It takes two trees T_i and T_j as input which are already rooted at taxon *dummy*. Then, it removes the taxon *dummy* from each tree and joins the trees by an internal edge connecting the root nodes. Finally, it returns the merged tree T .

3.2 Method of Bipartition

The most crucial part of our algorithm is the bipartition (divide step) technique. Here, we adopt a new bipartition technique inspired from the famous Fiduccia and Mattheyses (FM) algorithm for bipartitioning a hyper graph minimizing the cut size [5]. This is why our algorithm is referred to as the MFM (Modified FM) Bipartition Algorithm. Before we give our bipartition algorithm we shall discuss about the components of the algorithm. First, in Section 3.2.1 we describe the algorithmic components of Algorithm MFM. Next, in Section 3.2.2 we describe the generic MFM algorithm. Then in Section 3.3 we describe six different versions of the generic MFM algorithm.

3.2.1 Algorithmic Components of MFM

Algorithm MFM takes a pair of taxa set and a quartet set (P, Q) as input. It partitions P into two sets, namely, P_a and P_b with an objective that (P_a, P_b) satisfies maximum number of quartets from Q . The algorithm starts with an initial partition and iteratively searches for a

better partition. We will use a heuristic search to find the best partition. Before we describe the steps of the algorithm, we describe the algorithmic components.

Partition Score:

We assess the quality of a partition by assigning a *partition score*. We use a scoring function, $Score(P_a, P_b, Q)$, such that higher score will indicate a better partition. This function checks each $q \in Q$ against the partition (P_a, P_b) and determines whether q is *satisfied*, *violated* or *deferred*. We will define the score function in terms of the number of satisfied and violated quartets. Let s and v denote the number of satisfied and violated quartets. Then, two natural ways of defining the score function are: 1) taking the difference between the number of satisfied and violated quartets ($s - v$), and 2) taking the ratio of the number of satisfied and violated quartets (s/v). We can also use some other complicated score functions defined in terms of the number of satisfied, violated and deferred quartets (i.e., $Score(P_a, P_b, Q) = f(s, v, d)$, where d denotes the number of deferred quartets).

Gain Measure:

Let (P_a, P_b) be a partition of set of taxa P . Let $t \in P$ be a taxon and without loss of generality we assume that $t \in P_a$. Let (P'_a, P'_b) be the partition after moving the taxa t from P_a to P_b . That means, $P'_a = P_a - t$, and $P'_b = P_b \cup t$. Then we define the *gain* of the transfer of the taxon t with respect to (P_a, P_b) , denoted by $Gain(t, (P_a, P_b))$, as $Score(P'_a, P'_b, Q) - Score(P_a, P_b, Q)$.

Singleton Bipartition:

A singleton bipartition is a bipartition with a single taxon in one (or both) of the partitions. We do not allow our bipartition algorithm to return a singleton bipartition to avoid the risk of infinite loop.

3.2.2 Algorithm MFM

Now we describe the steps of the generic MFM (**M**odified **F**M) Bipartition Algorithm. Let, (P, Q) be the input to the bipartition algorithm, where P be a set of taxa and Q be a set of quartets over the taxa set P .

We start with an initial bipartition (P_{a_0}, P_{b_0}) of P . Obtaining (P_{a_0}, P_{b_0}) , we search for a better partition iteratively. At each iteration, we perform a series of transfers of taxa from one partition set to the other to maximize the number of satisfied quartets. At the beginning of an iteration, we set the status of all the taxa as *free*. Then, for each *free* taxon $t \in P$, we calculate $Gain(t, (P_{a_0}, P_{b_0}))$, and find the taxon t_1 with the maximum gain. There can be more than one taxa with the maximum gain where we need to break the tie. We will discuss this issue later. Next we transfer t_1 and set the status of this taxon as *locked* in the new partition that indicates that it will not be considered to be transferred again in this current iteration. This transfer creates the first intermediate bipartition (P_{a_1}, P_{b_1}) . The algorithm then find the next free taxon t_2 with maximum gain with respect to (P_{a_1}, P_{b_1}) , and transfer and lock that taxon to create another intermediate bipartition (P_{a_2}, P_{b_2}) . Thus we transfer all the free taxon one by one. Let Q be the input quartet set and P be the corresponding taxa set. $Q = \{((1, 2), (3, 4)), ((1, 2), (5, 6)), ((1, 3), (2, 4)), ((3, 4), (5, 6)), ((2, 3), (4, 5)), ((1, 3), (5, 6))\}$ (same as used in Figure 3.1). In this example, we used $s - v$ as the partition score. Hence, $P = \{1, 2, 3, 4, 5, 6\}$. Assume that following the steps of the initial bipartition, we get the initial bipartition $P_{a_0} = \{1, 2\}$ and $P_{b_0} = \{3, 4, 5, 6\}$. Figure 3.3 shows the first iteration of the bipartition algorithm.

Let $\{t_1, t_2, \dots, t_n\}$ be the ordering of locking. That is, t_1 has been locked first, then t_2, t_3 and so on. Let, the gain values of the corresponding partitions are:

$$Gain(t_1, (P_{a_0}, P_{b_0})), Gain(t_2, (P_{a_1}, P_{b_1})), \dots, Gain(t_n, (P_{a_{n-1}}, P_{b_{n-1}})).$$

Now we define the cumulative gain up to k th transfer as

$$CGain(k) = \sum_{i=1}^k Gain(t_i, (P_{a_{i-1}}, P_{b_{i-1}})).$$

The maximum cumulative gain, $MCGain(t_1, t_2, \dots, t_n)$ is defined as

$$MCGain(\{t_1, t_2, \dots, t_n\}) = \max_{1 \leq i \leq n} CGain(i).$$

In each iteration the algorithm finds the current ordering $(\{t_1, t_2, \dots, t_n\})$ of the transfers and save this order in a log table along with the cumulative gains (see Table 2 for example). Let t_m be the taxon in the log table corresponding to the $MCGain(\{t_1, t_2, \dots, t_n\})$, that means we obtain the maximum cumulative gain after moving the m th taxon (with respect to the order stored in the log table). Then we rollback the transfers of the taxa (t_{m+1}, \dots, t_n) that were moved after t_m . Let the resultant partition after these rollbacks is (P_a, P_b) . This partition will be the initial partition for the next iteration. In this way, the algorithm continues as long as the maximum cumulative gains is greater than zero and return the resultant bipartition. Table 3.1 lists the ordering of locking, corresponding gain and cumulative gain with respect to the iteration illustrated in Figure 3.3. From Table 3.1 we note that we get maximum cumulative gain 2 after moving taxon 3. Here, we also get maximum value of cumulative gain after moving taxon 4. We break the tie arbitrarily. We consider the taxon for which we get maximum cumulative gain for the first time. For this example, we get maximum cumulative gain 2 at taxon 3 for the first time. So we rollback all the moves after that move. The resultant partition after this rollback is $(\{1, 2, 3\}, \{4, 5, 6\})$ (partition (P_{a_1}, P_{b_1}) in Table 1). Similarly, Table 3.2 lists the ordering of locking, corresponding gain and cumulative gain with respect to the iteration next to the iteration illustrated in Figure 3.3. From Table 3.2 we get that the maximum cumulative gain 0, so the moves are rolled back and we get the final resultant partition $(\{1, 2, 3\}, \{4, 5, 6\})$.

The bipartitioning of a taxa set P is done at each divide step (see Section 3.1.1). At each divide step we have a (P, Q) pair as input. The bipartition algorithm returns a bipartition (P_a, P_b) of the taxa set P . We then divide Q into Q_a and Q_b and obtain (P_a, Q_a) and (P_b, Q_b) pairs. P_a and P_b will be further bipartitioned in subsequent divide steps. The pseudo-code of the bipartition method MFM is given in Figure 3.4.

k	$Taxon$	$Gain$	$CGain(k)$
1	3	2	2
2	4	0	2
3	2	-2	0
4	1	-1	-1
5	5	-1	-2
6	6	2	0

Table 3.1: The log table corresponding to the iteration shown in Figure 3.3

k	$Taxon$	$Gain$	$CGain(k)$
1	4	0	0
2	2	-2	-2
3	1	-1	-3
4	5	-1	-4
5	6	2	-2
6	3	2	0

Table 3.2: The log table corresponding to the next iteration of the iteration shown in Figure 3.3

3.3 Variations of Algorithm MFM

In this section we shall discuss different versions of MFM algorithm, where these versions differ from one another in i) initial partitioning, ii) partition score, and iii) selection of free taxon. We have implemented Algorithm QFM using six different approaches of Algorithm MFM. Thus we get six different versions of QFM. Performance of these different approaches are summarized in Chapter 4.

3.3.1 Algorithm MFM - Ia

This version of the MFM algorithm has the following characteristics:

- **Random Initial Partition:** Let P be the input taxa set and Q be input quartet set. Initially P_{a_0} and P_{b_0} is empty. We select each taxon from P and insert it into P_{a_0} and P_{b_0} alternative manner. Finally, we get P_{a_0} and P_{b_0} which are balanced, that is, the difference of their cardinality is either 0 or 1.
- **Partition Score:** Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$Score(P_a, P_b, Q) = s - v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

- **Selection of free taxon:** At each iteration, all the free taxa are moved to the other partition one by one. Among multiple choices of free taxa, we select the taxon with maximum gain. If there are multiple choices of such a taxon, then we select the taxon among those choices, transfer of which satisfies maximum number of quartet. If there is a tie again, we break the tie at random. The most important point is, here we do not keep any check whether the transfer of the selected taxon creates a singleton partition or not.

3.3.2 Algorithm MFM - Ib

This version of the MFM algorithm is almost similar to the previous version (Section 3.3.1). It differs from the previous version in the following characteristic:

Partition Score

Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$Score(P_a, P_b, Q) = (s - v) + s/v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

The pseudocode for initial partition for Algorithm MFM - Ia and MFM - Ib is given in Figure 3.5. We call this version Initial Partition - I.

3.3.3 Algorithm MFM - IIa

MFM - II noticeably differs from MFM - I in the way of initial partition and the choice of free taxon. The characteristics of this version are summarized below.

Initial Bipartition

Let $Q = q_1, q_2, \dots, q_m$, where $m = |Q|$ be the input quartet set. Initially both P_{a_0} and P_{b_0} are empty. Now we consider the quartets of Q one by one and do the following.

Let $q = ((t_1, t_2), (t_3, t_4))$ be a quartet in Q . If none of the 4 taxa belongs to either P_{a_0} or P_{b_0} , then we insert t_1 and t_2 in P_{a_0} and t_3 and t_4 in P_{b_0} . Otherwise, if any of the 4 taxa exists in either P_{a_0} or P_{b_0} we take the following actions to insert a taxon which does not exist in P_{a_0} or P_{b_0} . We maintain an insertion order. We consider t_1, t_2, t_3 and t_4 respectively.

- To insert t_1 , we look for the partition of t_2 (if t_2 exists in any part) and inset t_1 into that partition. But if t_2 does not exist in either of the partition, then we look for the partition of either t_3 or t_4 (either of these two must exist in P_{a_0} or P_{b_0}) and inset t_1 into the other partition.
- To insert t_2 , we look for the partition of t_1 and inset t_2 into that partition.
- To insert t_3 , we look for the partition of t_4 (if t_4 exists in any part) and inset t_3 into that partition. But if t_4 does not exist in either of the partition, then we look for the partition of either t_1 and inset t_3 into the other partition.
- To insert t_4 , we look for the partition of t_3 and inset t_4 into that partition.

When we insert a taxon t to any part, we remove it from P . After considering each $q \in Q$ and inserting taxa accordingly, if P remains non-empty, we insert the remaining taxa to either part randomly.

Partition Score

Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$Score(P_a, P_b, Q) = s - v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

Selection of free taxon - check for singleton partition

The way follow for initial partition may result in imbalanced partition (majority of taxa in one part and a few taxa in the other part) which may result in a singleton partition in the course of transferring free taxon from one partition to the other. We should not allow any singleton bipartition to avoid infinite loop. Therefore, we need to add some additional conditions in selecting the free taxon so that its transfer does not result in a singleton partition. Also, there could be more than one taxa with maximum gain, where we need to decide which one to transfer. We consider the following cases to address these issues. Let, M be a set of taxa with maximum gain.

- Case 1: $|M| = 1$ and transfer of $t \in M$ will not result in a singleton partition. Then, we transfer taxon t .
- Case 2: $|M| > 1$ and all the corresponding partitions are singleton. In that case, we look for a taxon with second highest maximum gain. Now, assume that M' contains the taxa with second highest maximum. Then we select one taxon considering these cases for M' as we did for M . And we continue in this way further if appropriate.
- Case 3: $|M| > 1$ and at least one corresponding partition is not singleton. If there is only one taxon, transfer of which will not result in a singleton partition then that taxon is selected to be transferred. In case of a tie, we find out the taxon (taxa), for which the corresponding partition(s) satisfies (satisfy) the maximum number of quartets. If one

such taxon is available, then we choose that taxon for the transfer. Otherwise, if we have multiple such taxa available, we choose one taxon to transfer at random.

3.3.4 Algorithm MFM - IIb

This version of the MFM algorithm is almost similar to the previous version (Section 3.3.3) in the way of initial partition and selection of free taxon. It differs from the previous version in the following characteristic:

Partition Score

Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$Score(P_a, P_b, Q) = (s - v) + s/v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

The pseudocode for initial partition for Algorithm MFM - IIa and MFM - IIb is given in Figure 3.6. We call this version Initial Partition - II.

3.3.5 Algorithm MFM - IIIa

MFM - III noticeably differs from MFM - II in the way of initial partition. The characteristics of this version are summarized below.

Initial Bipartition

The initial bipartitioning is done in four steps.

- Step 1: We count the frequency of each distinct quartet in Q .
- Step 2: Then sort Q by the frequency count of the quartets in non increasing order.

- Step 3: Suppose after sorting $Q = q_1, q_2, \dots, q_m$, where $m = |Q|$. Now we consider the quartets one by one in the sorted order. Initially both P_{a_0} and P_{b_0} are empty.

Let $q = ((t_1, t_2), (t_3, t_4))$ be a quartet in Q . If none of the 4 taxa belongs to either P_{a_0} or P_{b_0} , then we insert t_1 and t_2 in P_{a_0} and t_3 and t_4 in P_{b_0} . Otherwise, if any of the 4 taxa exists in either P_{a_0} or P_{b_0} we take the following actions to insert a taxon which does not exist in P_{a_0} or P_{b_0} . We maintain an insertion order. We consider t_1, t_2, t_3 and t_4 respectively.

- To insert t_1 , we look for the partition of t_2 (if t_2 exists in any part) and inset t_1 into that partition. But if t_2 does not exist in either of the partition, then we look for the partition of either t_3 or t_4 (either of these two must exist in P_{a_0} or P_{b_0}) and inset t_1 into the other partition.
 - To insert t_2 , we look for the partition of t_1 and inset t_2 into that partition.
 - To insert t_3 , we look for the partition of t_4 (if t_4 exists in any part) and inset t_3 into that partition. But if t_4 does not exist in either of the partition, then we look for the partition of either t_1 and inset t_3 into the other partition.
 - To insert t_4 , we look for the partition of t_3 and inset t_4 into that partition.
- Step 4: When we insert a taxon t to any part, we remove it from P . After considering each $q \in Q$ and inserting taxa accordingly, if P remains non-empty, we insert the remaining taxa to either part randomly.

Partition Score

Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$Score(P_a, P_b, Q) = s - v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

Selection of free taxon - check for singleton partition

The way follow for initial partition may result in imbalanced partition (majority of taxa in one part and a few taxa in the other part) which may result in a singleton partition in the course of transferring free taxon from one partition to the other. We should not allow any singleton bipartition to avoid infinite loop. Therefore, we need to add some additional conditions in selecting the free taxon so that its transfer does not result in a singleton partition. Also, there could be more than one taxa with maximum gain, where we need to decide which one to transfer. We consider the following cases to address these issues. Let, M be a set of taxa with maximum gain.

- Case 1: $|M| = 1$ and transfer of $t \in M$ will not result in a singleton partition. Then, we transfer taxon t .
- Case 2: $|M| > 1$ and all the corresponding partitions are singleton. In that case, we look for a taxon with second highest maximum gain. Now, M' contains taxa with second highest maximum. Then we select one taxon considering these cases for M' as we did for M . And we continue in this way further if appropriate.
- Case 3: $|M| > 1$ and at least one corresponding partition is not singleton. If there is only one taxon, transfer of which will not result in a singleton partition then that taxon is selected to be transferred. In case of a tie, we find out the taxon (taxa), for which the corresponding partition(s) satisfies(satisfy) the maximum number of quartets. If one such taxon is available, then we choose that taxon for the transfer. Otherwise, if we have multiple such taxa available, we choose one taxon to transfer at random.

3.3.6 Algorithm MFM - IIIb

This version of the MFM algorithm is almost similar to the previous version (Section 3.3.5) in the way of initial partition and selection of free taxon. It differs from the previous version in the following characteristic:

Partition Score

Let P_a and P_b be the current partition and Q be the quartet set. The partition score is computed using the following formula:

$$\text{Score}(P_a, P_b, Q) = s - v + s/v$$

where s and v are the *number of satisfied* and the *number of violated* quartets (respectively) in Q with respect to the partition (P_a, P_b) .

The pseudocode for initial partition for Algorithm MFM - IIIa and MFM - IIIb is given in Figure 3.7. We call this version Initial Partition - III.

3.4 Time Complexity

In this section, we shall derive the theoretical running time of Algorithm MFM (P, Q) , where P is a set of taxa and Q is a set of quartets over the taxa set P . Let, n and m be the cardinality of taxa set P and the quartet set Q respectively. We first derive the running time for the *Initial Partition*.

- Initial Partition - I: It inserts each taxa $t \in P$ alternatively either in P_a or in P_b . So the time complexity is $\mathbf{O}(n)$.
- Initial Partition - II: It checks each quartet $q \in Q$ and inserts each of its 4 taxa either in P_a or in P_b by checking the existing elements of P_a and P_b . The length of P_a or P_b is bounded by $\mathbf{O}(n)$, so the time required to insert taxa of each quartet is $\mathbf{O}(n)$. Hence, the total running time is $\mathbf{O}(nm)$.
- Initial Partition - III: In this approach, before we perform the steps of Initial Partition - II, we first count the frequency of the distinct quartets in Q and sort Q by frequency count. Each of the counting and the sorting step requires $\mathbf{O}(m^2)$ running time. So the total complexity is $\mathbf{O}(m^2) + \mathbf{O}(nm)$.

Now we explain the time required for the remaining part of Algorithm MFM which is accomplished in several iterations. Let, the maximum cumulative gain becomes 0 in k iterations. The time complexity per iteration is described below.

- **Gain Measure of a Partition:** The gain of a new partition is the difference between its score and the score of initial partition. The difference is measured in $\mathbf{O}(1)$ time. We need to find out the time required to calculate partition score of a partition (P_a, P_b) . To calculate score, each $q \in Q$ is checked against the partition (P_a, P_b) which takes $\mathbf{O}(n)$ since the length of P_a or P_b is bounded by $\mathbf{O}(n)$. Hence to check m quartets, that is, to calculate partition score $\mathbf{O}(nm)$ time is required.
- **SELECT_FREE_TAXON(P):** One taxon is selected among the free taxa. For each free taxon *Gain* is measured and the taxon with maximum gain is selected. There are n free taxa initially, so this step requires $n \times \mathbf{O}(nm) = \mathbf{O}(n^2m)$ time. The selected taxon is made *locked*.
- There are n free taxon initially. Each taxon is selected and locked one after another. So the total time complexity to lock all the taxa = $n \times \mathbf{O}(n^2m) = \mathbf{O}(n^3m)$.
- Each locked taxon has a gain associated with it. When all taxa are locked, cumulative gain and maximum cumulative gain are calculated. These take $\mathbf{O}(n)$ time.

Overall the running time for one iteration is $\mathbf{O}(n^3m) + \mathbf{O}(n) = \mathbf{O}(n^3m)$. For k iterations, the time complexity becomes $\mathbf{O}(n^3mk)$. Depending on the initial partition technique used, the time complexity of initial partition will be added with this value to give the total time complexity of Algorithm MFM.

3.5 Summary

In this chapter we discussed our quartet based phylogenetic tree reconstruction method. In Section 3.1 we described our algorithm QFM. Then in Section 3.2 we presented the bipartition algorithm that we use in Algorithm QFM. We described the generic approach of our bipartition

algorithm MFM in Section 3.2.2. We discussed six different versions of Algorithm MFM in Section 3.3. Finally, in Section 3.4 we derived the time complexity of Algorithm MFM (the main contributing part of time complexity of QFM).

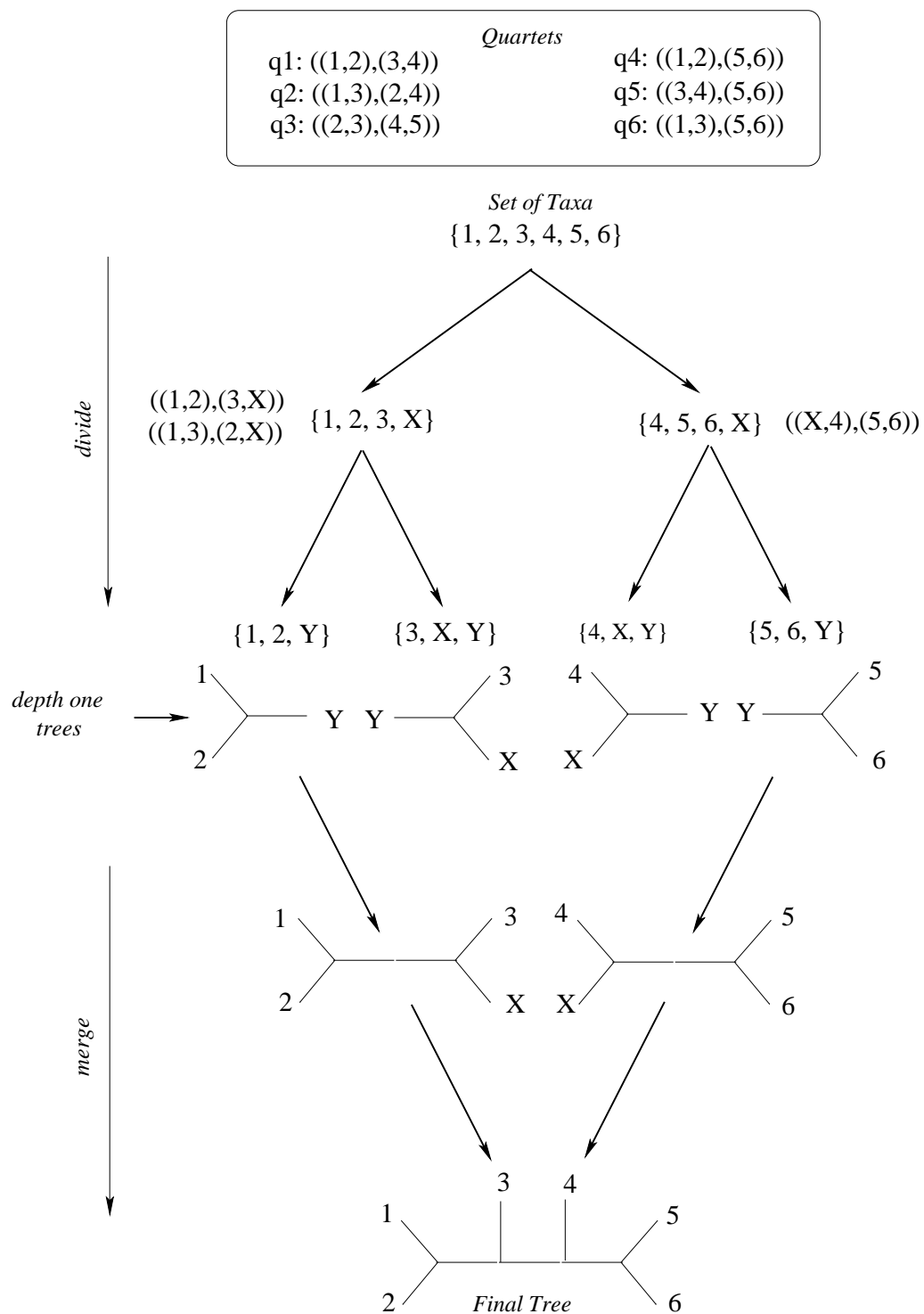


Figure 3.1: Divide and conquer approach.

Algorithm QFM(P, Q)

if $|P| \leq 3$ or Q is *empty*

then $T = \text{DEPTH_ONE_TREE}(P)$

return T

else

$(P_a, P_b) \leftarrow \text{MFM}(P, Q)$

$dummy \leftarrow$ uniquely generated artificial taxon

$P_a \leftarrow \{P_a \cup dummy\}$

$P_b \leftarrow \{P_b \cup dummy\}$

$Q_a \leftarrow \text{POPULATE}(P_a, Q, dummy)$

$Q_b \leftarrow \text{POPULATE}(P_b, Q, dummy)$

$T_1 = \text{QFM}(P_a, Q_a)$

$T_2 = \text{QFM}(P_b, Q_b)$

$T'_1 = \text{REROOT}(T_1, dummy)$

$T'_2 = \text{REROOT}(T_2, dummy)$

$T = \text{MERGE}(T'_1, T'_2)$

return T

Figure 3.2: QFM (Quartet FM Algorithm)

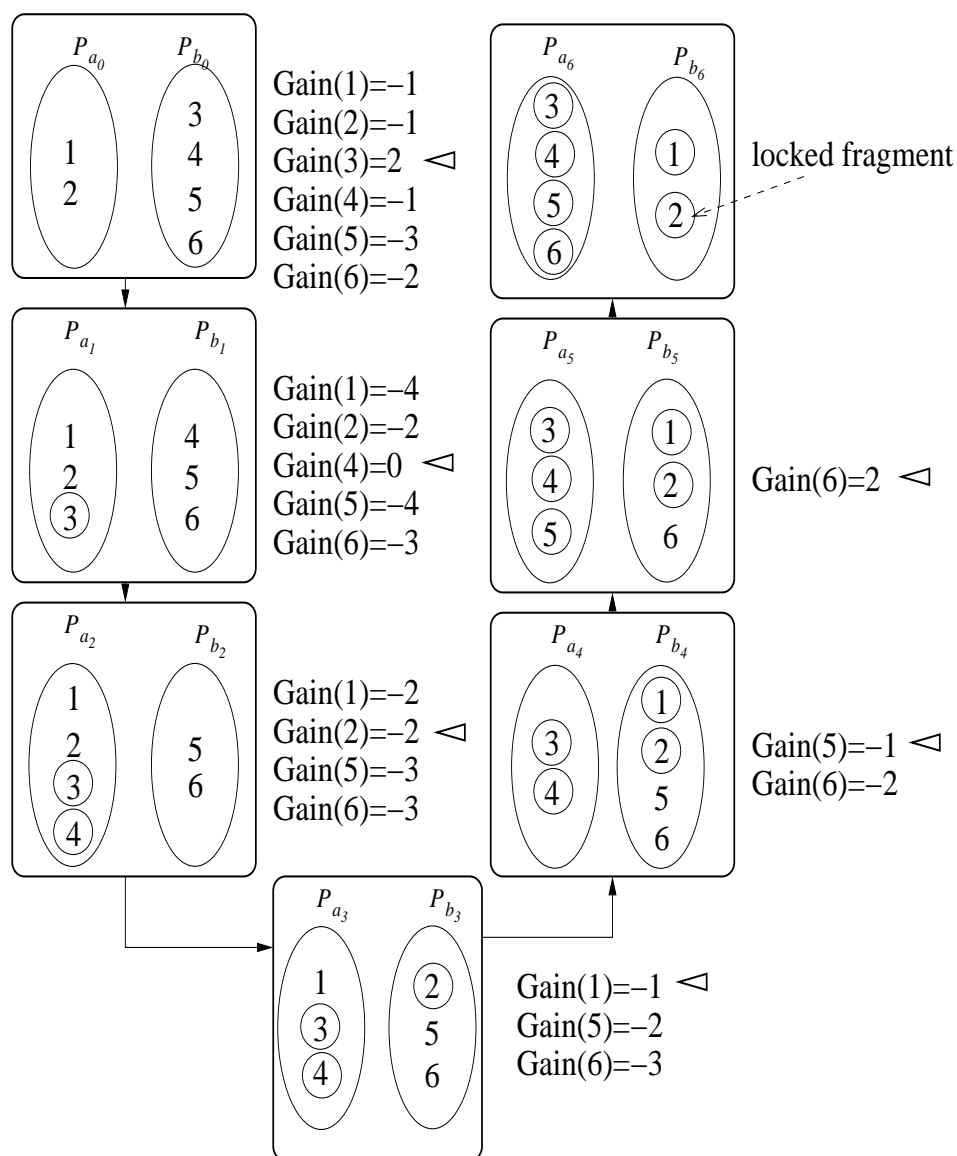


Figure 3.3: An example iteration of the Bipartition Algorithm MFM. The locked taxa are shown in circles.

```

Algorithm MFM( $P, Q$ )
( $P_{a_0}, P_{b_0}$ )  $\leftarrow$  INITIAL_PARTITION( $P, Q$ )
repeat always
  FREE_LOCKS( $P$ )    //set the status of each taxon free
  CLEAR_LOG()      //maintain a log file, initially blank
   $i \leftarrow 1$ 
  while there is a free taxon do
    begin
       $t_i \leftarrow$  SELECT_FREE_TAXON( $P$ )    //find a free taxon to
      transfer next
      transfer  $t_i$  to the other partition
      update ( $P_{a_{i-1}}, P_{b_{i-1}}$ ) to ( $P_{a_i}, P_{b_i}$ )
      LOCK( $t_i$ )    //set the status of taxon  $t_i$  locked
      LOG_RECORD( $t_i, \text{Gain}(t_i, (P_{a_{i-1}}, P_{b_{i-1}}))$ )    //write on log file
      increment  $i$ 
    end do

  FREE_LOCKS()
  check the log file and find  $MCGain(t_1, t_2, \dots, t_n)$  and  $t_m$     //cumu-
  lative gain is maximum at the  $m$ -th transfer
  if  $MCGain(t_1, t_2, \dots, t_n) > 0$ 
    then
      set new ( $P_{a_0}, P_{b_0}$ ) by rolling back the transfers that occurred
      after the transfer of  $t_m$ 
      CLEAR_LOG()
      continue with the the loop
    else
      terminate the algorithm and output current partition
  end repeat

```

Figure 3.4: MFM (**M**odified **F**iduccia **M**attheyses **B**ipartition **A**lgorithm)

```
Procedure INITIAL_PARTITION-I ( $P, Q$ )  
 $P_{a_0} \leftarrow \phi$   
 $P_{b_0} \leftarrow \phi$   
 $flag \leftarrow 0$   
while  $P$  is not empty do  
  begin  
    Select any taxon  $t$  from  $P$   
    if  $flag$  is zero  
      then Insert  $t$  to  $P_{a_0}$   
      else Insert  $t$  to  $P_{b_0}$   
    Toggle  $flag$  and remove  $t$  from  $P$   
  end do  
return ( $P_{a_0}, P_{b_0}$ )
```

Figure 3.5: Pseudocode for Procedure Initial Partition - I

Procedure INITIAL_PARTITION-II (P, Q)

$P_{a_0} \leftarrow \phi$

$P_{b_0} \leftarrow \phi$

for each $q = ((t_1, t_2), (t_3, t_4)) \in Q$ **do**

if t_1, t_2, t_3, t_4 non existent in P_{a_0} or in P_{b_0}

then Insert t_1 and t_2 to P_{a_0}

 Insert t_3 and t_4 to P_{b_0}

else

if t_1 non existent in P_{a_0} or in P_{b_0}

then if t_2 exists in P_{a_0} or in P_{b_0}

then Insert t_1 to that partition

else if t_3 exists in P_{a_0} or in P_{b_0}

then Insert t_1 to the other partition

else if t_4 exists in P_{a_0} or in P_{b_0}

then Insert t_1 to the other partition

if t_2 non existent in P_{a_0} or in P_{b_0}

then Insert t_2 to the partition where t_1 exists

if t_3 non existent in P_{a_0} or in P_{b_0}

then if t_4 exists in P_{a_0} or in P_{b_0}

then Insert t_3 to that partition

else Insert t_3 to the partition where t_1 does not exist

if t_4 non existent in P_{a_0} or in P_{b_0}

then Insert t_4 to the partition where t_3 exists

return (P_{a_0}, P_{b_0})

Figure 3.6: Pseudocode for Procedure Initial Partition - II

Procedure INITIAL_PARTITION-III (P, Q) $P_{a_0} \leftarrow \phi$ $P_{b_0} \leftarrow \phi$ COUNT_FREQUENCY(Q) //count the frequency of
each distinct quartet in Q $Q \leftarrow$ SORT_BY_FREQUENCY(Q) //sort quartets by
their frequency count**for** each $q = ((t_1, t_2), (t_3, t_4)) \in Q$ **do** **if** t_1, t_2, t_3, t_4 non existent in P_{a_0} or in P_{b_0} **then** Insert t_1 and t_2 to P_{a_0} Insert t_3 and t_4 to P_{b_0} **else** **if** t_1 non existent in P_{a_0} or in P_{b_0} **then if** t_2 exists in P_{a_0} or in P_{b_0} **then** Insert t_1 to that partition **else if** t_3 exists in P_{a_0} or in P_{b_0} **then** Insert t_1 to the other partition **else if** t_4 exists in P_{a_0} or in P_{b_0} **then** Insert t_1 to the other partition **if** t_2 non existent in P_{a_0} or in P_{b_0} **then** Insert t_2 to the partition where t_1 exists **if** t_3 non existent in P_{a_0} or in P_{b_0} **then if** t_4 exists in P_{a_0} or in P_{b_0} **then** Insert t_3 to that partition **else** Insert t_3 to the partition where t_1 does not exist **if** t_4 non existent in P_{a_0} or in P_{b_0} **then** Insert t_4 to the partition where t_3 exists **return** (P_{a_0}, P_{b_0})

Figure 3.7: Pseudocode for Procedure Initial Partition - III

Chapter 4

Experimental Study

In this chapter we will present our experimental setup, data generation method, results and analysis thereof. Before that, in Section 4.1 we will describe the accuracy measures for simulation study and in Section 4.2 we will describe a general strategy of simulation study. Then in Section 4.3 we will describe how we have performed our experiment on simulated data. We will then highlight the data generation method (Section 4.3.1), result summary and comparison of the 6 approaches of QFM (Section 4.3.2). Then in Section 4.3.3 we will analyze the accuracy of QFM over QMC. Finally, in Section 4.4 we will explain our experimental results on biological datasets.

4.1 Measure of Accuracy

Since the true tree is unknown, measuring the accuracy of an estimated tree is a big challenge. If the true tree is known, then topological measures are used to determine how close the estimated tree is with the true tree. Let us assume that the true tree is known; now we will discuss some measures to determine the topological accuracy of the estimated tree.

Let T_0 on taxa set S is the true tree, and T is an estimated tree for the same taxa set. Let $C(T_0)$ and $C(T)$ are the sets of non-trivial bipartitions of T_0 and T respectively. There are several techniques that have been used to quantify errors in T with respect to T_0 , of which the dominant ones are the followings:

1. False Negatives (FN): The false negatives are those edges in T_0 inducing bipartitions that do not appear in $C(T)$. Such a branch is also called a “missing branch” or a “missing edge”. The false negative rate is the fraction of the total number of non-trivial bipartitions that are missing, or $\frac{|C(T_0)-C(T)|}{|C(T_0)|}$ [19].
2. False Positives (FP): The false positives in a tree T with respect to the tree T_0 are those edges in T that induce bipartitions that do not appear in $C(T_0)$. The false positive rate is the fraction of the total number of non-trivial bipartitions in T that are false positives, or $\frac{|C(T)-C(T_0)|}{|C(T)|}$ [19].
3. Robinson-Foulds (RF) [27]: The most typically used error metric is the sum of the number of false positives and false negatives, and is called the Robinson-Foulds distance. The sum of the number of (internal) edges in both the model and the output trees is $2n - 6$ (where n is the number of taxa in each tree) when both trees are binary. The sum of the number of false positives and false negatives ranges from 0 (so the trees are identical) to at most $2n - 6$. To turn this into an error rate, that number is divided by $2n - 6$.

4.2 Design of Simulation Study

Simulation studies are widely used to evaluate the performance of phylogenetic reconstruction methods, as well as that of supertree methods. Simulation studies are important for two reasons:

- When performing a phylogenetic analysis of real data, we do not typically know the true tree for the taxa under study. In simulation studies we know the true tree because we explicitly construct it, and can use this tree to measure the topological accuracy of the tree(s) returned by a given tree inference method.
- Furthermore, the theoretical guarantees on performance for phylogenetic methods are usually very loose. Simulations help overcome this problem by providing actual figures on the topological accuracy of the trees returned by phylogenetic reconstruction methods, as long as the model space is adequately explored, and the models are biologically relevant.

The simulation study of a quartet based method involves the following four steps:

- **Generate a model tree:** Generate either a random or a biologically-based model tree T over the taxa set under consideration. A biological model tree is one which is reconstructed from a thorough analysis of a biological data set, and believed by experts to be the most accurate estimate of the history of that set of taxa. A random model tree can be picked uniformly at random from all possible tree topologies or by some random process.
- **Generate quartets:** Generate all possible induced quartets. An induced quartet is generated by deleting all but the 4 leaf nodes of interest with the edges incident to those and suppressing all degree two nodes. Figure 4.1 shows an example. In this example, we want to generate a quartet over 4 leaves B , F , E and C . So we delete the leaf nodes A and D with their incident edges and obtain quartet $((B, F), (E, C))$.

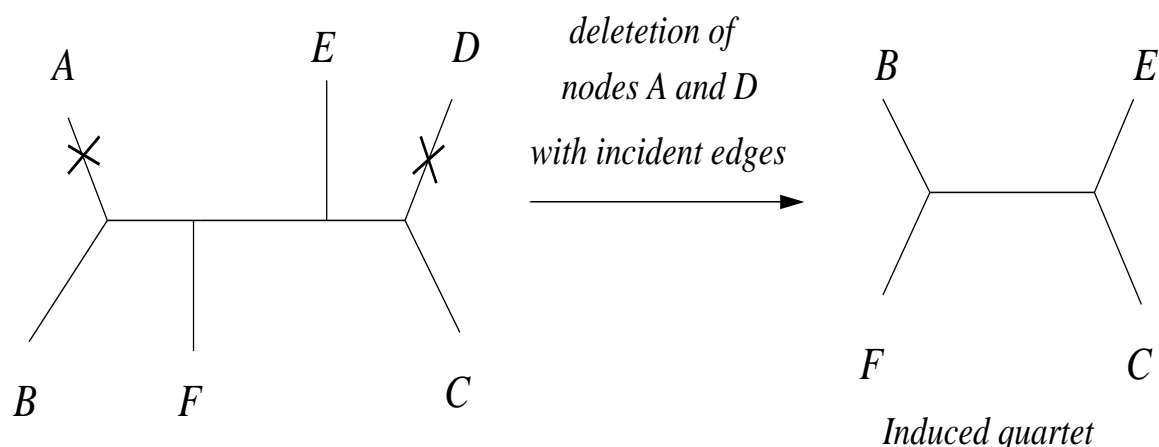


Figure 4.1: An example quartet generation from a tree $((A, B), F), (E, (D, C))$.

- **Sample input quartets:** Select quartets that will make the input quartet set Q . Sampling is done either at random or following different encoding techniques [38, 32].
- **Apply tree reconstruction method:** Apply the quartet based method on Q and obtain a supertree T' .
- **Compare with the model tree:** The inferred phylogeny T' is then compared against the

model phylogeny T , and error (usually topological) is quantified, using some measure (see Section 4.1).

4.3 Experiment on Simulated Data

In this section, we will describe the way we performed our experiment on simulated data. First, we have generated a model tree at random, which is our *true tree*. Then, we have generated sets of induced quartets from the model tree. We have applied both QFM and QMC on the generated input sets and then compared the results of QFM with that of QMC. We will analyze our results in two parts.

- Part I: We will compare and analyze the results of the 6 approaches of QFM with that of QMC (in terms of number of satisfied quartets) and identify one version of QFM as the best among the 6 approaches.
- Part II: We will compare the topological accuracy of the trees estimated by the best approach of QFM and QMC and show that QFM returns more accurate trees than QMC does.

Section 4.3.1 describes our simulation data sets and Section 4.3.2 summarizes results of Part I and Section 4.3.3 describes the findings of Part II of our simulation study.

4.3.1 Simulated Data Generation

We have generated seven model trees over 25, 50, 100, 200, 300, 400 and 500 taxa respectively. Let n denote the number of taxa of a model tree. From each model tree over n taxa we have generated quartet sets by varying the two following parameters:

- **Size of the input quartet set:** From all possible induced quartets, we take only a subset of induced quartets to make our input quartet set. We sample input quartets from all possible induced quartets uniformly at random. For each tree of n taxa, we generate quartet sets of size $n^{1.5}$, n^2 and $n^{2.8}$. In [32], QMC is applied to the quartet sets of size n^2

and $n^{2.8}$ and QMC has been found performing better for $n^{2.8}$ sized quartet sets. So in this thesis, we have chosen these two sizes for quartet sets. Also we have chosen a size $n^{1.5}$ at random to test the performance of both methods on a comparatively smaller quartet set.

- **Percentage of consistent quartets:** All induced quartets are consistent with the model tree. After we take a required number of induced quartets in the input set, we flip a fixed percentage of quartets so that they become inconsistent with the model tree. We denote the percentage of consistent quartets in the input set by c . From each input quartet set of a certain size (e.g., $n^{1.5}$, n^2 or $n^{2.8}$), we make three different input quartet sets taking $c = 70\%$, 80% and 90% .

We have used the tool developed in used in [32] to generate the model tree and the input quartet sets. The tool takes the number of taxa, size of quartet set and percentage of consistency as input and returns the quartet set accordingly. For 25, 50 and 100-taxon model trees, we have generated quartet sets of size 125 ($= 25^{1.5}$), 625 ($= 25^2$), 8208 ($= 25^{2.8}$), 354 ($= 50^{1.5}$), 2500 ($= 50^2$), 57164 ($= 50^{2.8}$), 1000 ($= 100^{1.5}$), 10000 ($= 100^2$), and 398108 ($= 100^{2.8}$) respectively. For each size, we varied the the percentage of consistency c by making it 70%, 80% and 90%. Thus for model trees of 25, 50 and 100 taxa, we get $(3 \times 3) \times 3 = 27$ input quartet sets. Similarly, for 200, 300, 400 and 500-taxon model trees, we varied quartet set size by $n^{1.5}$ and n^2 , where $n = 200, 300, 400$ and 500 . For each size, we again varied the the percentage of consistency c by making it 70%, 80% and 90%. Thus we get $(4 \times 2) \times 3 = 24$. In total we have generated $27 + 24 = 51$ input quartet sets.

We have also analyzed the datasets (uniformly sampled quartets) used in [32], where the number of taxa (n) takes values 25, 50, 100, 200, 300, 400 and 500. For these cases, we have n^2 quartets with $c = 90\%$ in each input set. So including these additional 7 input sets, total 58 input quartet sets make our simulation datasets.

4.3.2 Part I: Comparison of the Performance of Six Approaches of QFM

In Section 3.3 we have described six different approaches of our bipartition algorithm MFM. We have implemented all the six versions of our bipartition algorithm, which resulted in six different approaches for our quartet based supertree method QFM. We have applied simulation datasets over all these approaches. Also, we have applied our datasets to QMC. For QMC, we have used their implementation developed in [32]. The experiments have been done in an Intel dual core machine of 2 GB RAM with Linux OS. We now summarize results (i.e., number of satisfied quartets) of the all six versions of QFM and compare the results with that of QMC.

Algorithm QFM - Ia & Ib

QFM - Ia is implemented with MFM - Ia (Section 3.3.1). Similarly, MFM - Ib (Section 3.3.2) is implemented in QFM - Ib. Table 4.1 shows the results of QFM - Ia & Ib and the results of QMC at $c = 90\%$. Similarly, Table 4.2 and Table 4.3 show the results when 20% ($c = 80\%$) and 30% ($c = 70\%$) quartets are flipped respectively.

Table 4.1: **Comparison of QFM (Ia & Ib) to QMC at $c = 90\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

#Taxa	#Quartets	#Satisfied		
		QFM		QMC
		approach-Ia	approach-Ib	
25	125	114	112	114
25	625	557	558	559
25	8208	7266	7233	7388
50	354	316	309	319
50	2500	2248	2248	2252
50	57164	51280	51277	51448
100	1000	895	895	904
100	10000	8990	8979	9004
100	398108	358001	357995	358298
200	2829	2539	2542	2244
200	40000	36002	35984	36012
300	5197	4671	4688	4689
300	90000	80990	80957	81015
400	8000	7187	7201	7206
400	160000	143988	143981	144017
500	11181	10075	10051	10091
500	250000	224978	224948	225017

Table 4.2: **Comparison of QFM (Ia & Ib) to QMC at $c = 80\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

#Taxa	#Quartets	#Satisfied		
		QFM		QMC
		approach-Ia	approach-Ib	
25	125	99	100	106
25	625	491	491	501
25	8208	6485	6485	6567
50	354	286	288	289
50	2500	1987	1986	2004
50	57164	45600	45602	45732
100	1000	800	791	803
100	10000	7998	7992	8009
100	398108	318228	318228	318487
200	2829	2260	2245	2270
200	40000	32003	32009	32042
300	5197	4166	4127	4186
300	90000	72012	71986	72025
400	8000	6435	6427	6437
400	160000	128010	128000	128046
500	11181	8957	8948	8982
500	250000	199995	199989	200041

The results obtained from Table 4.1 - Table 4.3 show that QFM - Ia and QFM - Ib perform as closely as QMC. Though QFM - Ia and QFM - Ib satisfy less number of input quartets than QMC, but the difference is nominal with respect to the total number of input quartets. To compare the performance of these two approaches with respect to QMC, we have drawn the performance curves which are given in Appendix B. The performance curves obtained for QFM and QMC are almost the same.

Algorithm QFM - IIa & IIb

Though the results obtained from QFM - Ia and QFM -Ib are quite satisfactory, to get even better results we moved towards QFM - IIa and QFM - IIb. The algorithmic improvements are described in Section 3.3.3 and Section 3.3.4. The results of QFM - IIa & IIb and the results of QMC at $c = 90\%$ are shown in Table 4.4. Similarly, Table 4.5 and Table 4.6 show the results when 20% ($c = 80\%$) and 30% ($c = 70\%$) quartets are flipped respectively. QFM - IIa is implemented with MFM - IIa (Section 3.3.3). Similarly, MFM - IIb (Section 3.3.4) is implemented in QFM - IIb.

Table 4.3: **Comparison of QFM (Ia & Ib) to QMC at $c = 70\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets($\#Quartets$).

#Taxa	#Quartets	#Satisfied		
		QFM		QMC
		approach-Ia	approach-Ib	
25	125	88	87	92
25	625	426	427	429
25	8208	5668	5668	5746
50	354	252	244	260
50	2500	1734	1738	1754
50	57164	40015	39861	40015
100	1000	706	699	712
100	10000	7007	6994	7016
100	398108	278483	278483	278676
200	2829	2007	1983	2025
200	40000	28024	27998	28044
300	5197	3635	3637	3683
300	90000	63020	63012	63057
400	8000	5590	5557	5663
400	160000	111951	111965	112065
500	11181	7809	7741	7894
500	250000	175028	175011	175073

Table 4.4: **Comparison of QFM (IIa & IIb) to QMC at $c = 90\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

#Taxa	#Quartets	#Satisfied		
		QFM		QMC
		approach-IIa	approach-IIb	
25	125	114	116	114
25	625	563	563	559
25	8208	7388	7388	7388
50	354	318	312	319
50	2500	2251	2251	2252
50	57164	51448	51448	51448
100	1000	904	899	904
100	10000	9004	9001	9004
100	398108	358298	358298	358298
200	2829	2549	2547	2244
200	40000	36002	36000	36012
300	5197	4691	4687	4689
300	90000	81012	81005	81015
400	8000	7208	7202	7206
400	160000	144016	144007	144017
500	11181	10075	10060	10091
500	250000	225016	225008	225017

Table 4.5: **Comparison of QFM (IIa & IIb) to QMC at $c = 80\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	$\#Satisfied$		
		QFM		QMC
		approach-IIa	approach-IIb	
25	125	106	103	106
25	625	501	501	501
25	8208	6567	6567	6567
50	354	289	288	289
50	2500	2000	2000	2004
50	57164	45600	45732	45732
100	1000	801	799	803
100	10000	8007	8005	8009
100	398108	318487	318487	318487
200	2829	2267	2262	2270
200	40000	32037	32037	32042
300	5197	4173	4182	4186
300	90000	72029	72012	72025
400	8000	6448	6446	6437
400	160000	128038	128031	128046
500	11181	8969	8969	8982
500	250000	200042	200037	200041

With a close observation of the results, we find that we get more satisfactory results this time. Our result, in most of the cases matches with the result of QMC. In some cases QFM outperforms QMC and in a few cases our result is less than that of QMC. But the difference is nominal with respect to the total number of input quartets. We have drawn the performance curves for these results which are included in Appendix B. Again we get almost the same curves for QFM and QMC.

Algorithm QFM - IIIa & IIIb

To get even more satisfactory results we make some algorithmic improvements in our bipartition algorithm. QFM - IIIa is implemented with MFM - IIIa (Section 3.3.5). Similarly, MFM - IIIb (Section 3.3.6) is implemented in QFM - IIIb. Table 4.7 shows the results of QFM - IIIa & IIIb and the results of QMC at $c = 90\%$. Similarly, Table 4.8 and Table 4.9 show the results when 20% ($c = 80\%$) and 30% ($c = 70\%$) quartets are flipped respectively.

Table 4.6: **Comparison of QFM (IIa & IIb) to QMC at $c = 70\%$ in number of satisfied quartets.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

#Taxa	#Quartets	#Satisfied		
		QFM		QMC
		approach-IIa	approach-IIb	
25	125	87	90	92
25	625	426	434	429
25	8208	5668	5746	5746
50	354	260	253	260
50	2500	1750	1749	1754
50	57164	40015	40015	40015
100	1000	704	706	712
100	10000	7015	7010	7016
100	398108	278676	278676	278676
200	2829	2030	2016	2025
200	40000	28044	28043	28044
300	5197	3664	3650	3683
300	90000	63045	63044	63057
400	8000	5647	5615	5663
400	160000	112054	112054	112065
500	11181	7837	7837	7894
500	250000	175064	175044	175073

Table 4.7: Comparison of QFM (IIIa & IIIb) to QMC at $c = 90\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	$\#Satisfied$		
		QFM		QMC
		approach-IIIa	approach-IIIb	
25	125	114	116	114
25	625	563	563	559
25	8208	7388	7388	7388
50	354	318	315	319
50	2500	2252	2252	2252
50	57164	51448	51448	51448
100	1000	903	895	904
100	10000	9004	9003	9004
100	398108	358298	358298	358298
200	2829	2251	2536	2244
200	40000	36011	36003	36012
300	5197	4689	4682	4689
300	90000	81012	81003	81015
400	8000	7206	7206	7206
400	160000	144018	144008	144017
500	11181	10088	10074	10091
500	250000	225017	225017	225017

Table 4.8: Comparison of QFM (IIIa & IIIb) to QMC at $c = 80\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	$\#Satisfied$		
		QFM		QMC
		approach-IIIa	approach-IIIb	
25	125	106	103	106
25	625	501	501	501
25	8208	6567	6567	6567
50	354	289	286	289
50	2500	2002	2002	2004
50	57164	45732	45732	45732
100	1000	804	806	803
100	10000	8008	8006	8009
100	398108	318487	318487	318487
200	2829	2273	2264	2270
200	40000	32038	32037	32042
300	5197	4181	4174	4186
300	90000	72034	72020	72025
400	8000	6451	6446	6437
400	160000	128048	128043	128046
500	11181	8991	8983	8982
500	250000	200044	200044	200041

With a close observation of the values, we see that in most of the cases our result is greater than or equal to that of QMC. In a very few cases, our result is less than that of QMC with a nominal difference. The performance curves are shown in Appendix B. Moreover, QFM - IIIa performs better than QFM - IIIb. So we can identify QFM - IIIa as the best among the 6 approaches of QFM.

4.3.3 Part II: Comparison of QFM and QMC in Accurate Tree Estimation

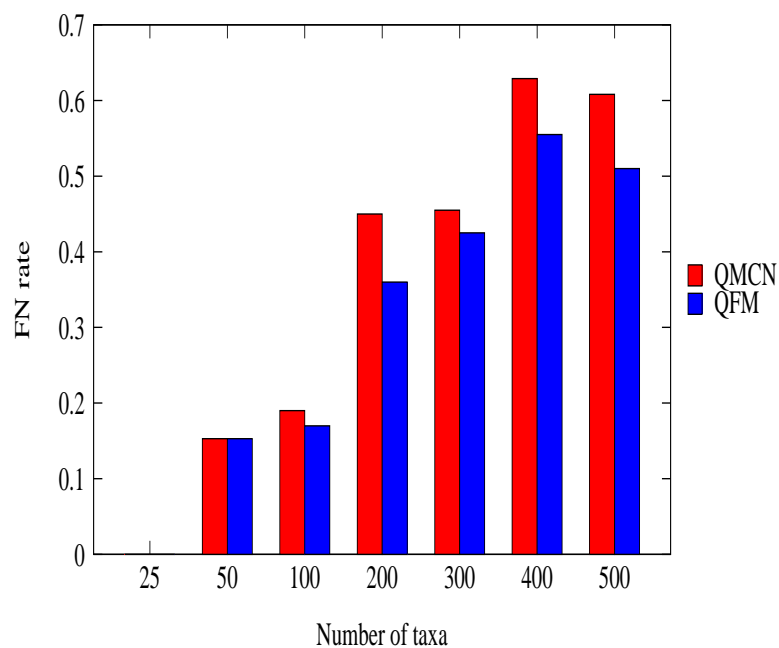
In this section, we will analyze and compare accuracy of the trees estimated by QFM and QMC. Among the six approaches of QFM algorithm, we find QFM - IIIa performing best. We measure the topological accuracy of the trees estimated by QFM - IIIa with the trees returned by QMC. FN (False Negative) rate, FP (False Positive) rate and RF (Robinson Foulds) rate are the three standards for measuring the accuracy of the estimated trees. In most of the studies, FN rate and RF rate have been considered as more reliable to measure the accuracy. To test the topological accuracy of QFM - IIIa against QMC, first we have applied our method over the dataset used in [32]. The results are shown in Table 4.10, which show that QFM clearly outperforms QMC. Figure 4.2 shows two charts based on Table 4.10. These charts are drawn for FN rates (missing branch rates). In Figure 4.2, the chart at top compares the missing branch rates of QFM - IIIa and QMC. The chart at bottom (Figure 4.2) illustrates the percentage of datasets on which the performance of QFM is better than, worse than or equal to QMC. The chart reveals that QFM is better than QMC in 71% of the cases.

Table 4.9: Comparison of QFM (IIIa & IIIb) to QMC at $c = 70\%$ in number of satisfied quartets. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	$\#Satisfied$		
		QFM		QMC
		approach-IIIa	approach-IIIb	
25	125	87	92	92
25	625	434	434	429
25	8208	5746	5746	5746
50	354	260	263	260
50	2500	1753	1754	1754
50	57164	40015	40015	40015
100	1000	722	721	712
100	10000	7018	7014	7016
100	398108	278676	278676	278676
200	2829	2026	2020	2025
200	40000	28043	28044	28044
300	5197	3672	3675	3683
300	90000	63057	63055	63057
400	8000	5648	5631	5663
400	160000	112066	112047	112065
500	11181	7814	7826	7894
500	250000	175078	175078	175073

Table 4.10: Comparison of QFM - IIIa to QMC in FN rate, FP rate and RF rate at $c = 90\%$ for the dataset used in [32]. The left two columns represent the size of model tree (#Taxa), and the number of input quartets (#Quartets).

#Taxa	#Quartets	#Satisfied		FN Rate		FP Rate		RF Rate	
		QFM	QMC	QFM	QMC	QFM	QMC	QFM	QMC
25	625	563	563	0.0	0.0	0.136	0.136	0.068	0.068
50	2500	2252	2252	0.154	0.154	0.298	0.298	0.226	0.226
100	10000	9000	9002	0.171	0.190	0.402	0.340	0.287	0.265
200	40000	36001	35984	0.363	0.450	0.599	0.609	0.481	0.530
300	90000	80980	80980	0.426	0.455	0.687	0.653	0.556	0.554
400	160000	144002	143997	0.555	0.629	0.776	0.758	0.665	0.694
500	250000	224998	224996	0.510	0.609	0.759	0.746	0.643	0.678



■ QFM<QMC ■ QFM=QMC ■ QFM>QMC

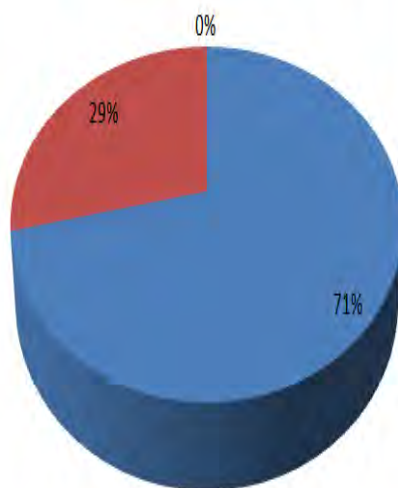


Figure 4.2: Top: Missing branch rates of QFM - IIIa and QMC. Bottom: Illustration for the proportion of datasets on which the performance of QFM is better than, worse than or equal to QMC.

We have also measured the topological accuracy for the results summarized in Table 4.3.2 to Table 4.9. We have compared FN rate, FP rate and RF rate for trees returned by QFM-IIa and QMC. The results are summarized from Table 4.11 to Table 4.13. The results have been analyzed by drawing graphs for FN rates, which are shown in Figure 4.3 and Figure 4.4. Figure 4.3 shows the bar charts comparing the values of missing branch rates (FN rate). Figure 4.4 shows the percentage of cases where QFM is better than, worse than or equal to QMC. These charts reveal that QFM is better in 81% of the cases for Table 4.11 and Table 4.13. For Table 4.12 QFM is better in 47% of the cases.

Table 4.11: **Comparison of QFM - IIIa to QMC at $c = 70\%$ in FN rate, FP rate and RF rate.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	FN Rate		FP Rate		RF Rate	
		QFM	QMC	QFM	QMC	QFM	QMC
25	125	1.0	0.75	1.0	0.818	1.0	0.784
25	625	0.222	0.263	0.363	0.363	0.293	0.313
25	8208	0.0	0.0	0.0	0.0	0.0	0.0
50	354	1.0	0.938	1.0	0.957	1.0	0.947
50	2500	0.436	0.474	0.531	0.574	0.434	0.524
50	57164	0.0	0.0	0.0	0.0	0.0	0.0
100	1000	1.0	0.964	1.0	0.979	1.0	0.972
100	10000	0.467	0.517	0.588	0.567	0.527	0.542
100	398108	0.010	0.021	0.010	0.021	0.010	0.021
200	2829	0.985	0.976	0.995	0.989	0.989	0.984
200	40000	0.652	0.616	0.751	0.731	0.702	0.673
300	5197	0.989	0.984	0.997	0.993	0.993	0.989
300	90000	0.694	0.742	0.825	0.832	0.759	0.787
400	8000	1.0	1.0	1.0	1.0	1.0	1.0
400	160000	0.783	0.812	0.879	0.872	0.831	0.842
500	11181	0.982	0.989	0.996	0.996	0.989	0.993
500	250000	0.827	0.836	0.897	0.893	0.862	0.865

Table 4.12: Comparison of QFM - IIIa to QMC at $c = 80\%$ in FN rate, FP rate and RF rate. The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	FN Rate		FP Rate		RF Rate	
		QFM	QMC	QFM	QMC	QFM	QMC
25	125	0.857	0.867	0.909	0.909	0.883	0.888
25	625	0.050	0.100	0.136	0.181	0.093	0.141
25	8208	0.0	0.0	0.0	0.0	0.0	0.0
50	354	0.591	0.759	0.809	0.851	0.699	0.805
50	2500	0.268	0.275	0.362	0.383	0.315	0.329
50	57164	0.0	0.0	0.0	0.0	0.0	0.0
100	1000	0.824	0.941	0.938	0.969	0.881	0.955
100	10000	0.289	0.341	0.495	0.443	0.392	0.392
100	398108	0.0	0.0	0.0	0.0	0.0	0.0
200	2829	0.912	0.946	0.975	0.975	0.943	0.960
200	40000	0.649	0.680	0.766	0.756	0.708	0.718
300	5197	0.941	0.965	0.976	0.983	0.959	0.974
300	90000	0.663	0.698	0.815	0.785	0.739	0.741
400	8000	0.952	0.959	0.987	0.985	0.969	0.972
400	160000	0.679	0.712	0.834	0.809	0.757	0.760
500	11181	0.939	0.953	0.986	0.984	0.962	0.969

Table 4.13: **Comparison of QFM - IIIa to QMC at $c = 90\%$ in FN rate, FP rate and RF rate.** The left two columns represent the size of model tree ($\#Taxa$), and the number of input quartets ($\#Quartets$).

$\#Taxa$	$\#Quartets$	FN Rate		FP Rate		RF Rate	
		QFM	QMC	QFM	QMC	QFM	QMC
25	125	0.846	0.800	0.909	0.837	0.878	0.832
25	625	0.0	0.167	0.0	0.318	0.0	0.242
25	8208	0.0	0.0	0.0	0.0	0.0	0.0
50	354	0.590	0.576	0.809	0.702	0.699	0.639
50	2500	0.147	0.111	0.383	0.319	0.265	0.215
50	57164	0.0	0.0	0.0	0.0	0.0	0.0
100	1000	0.769	0.800	0.907	0.897	0.838	0.848
100	10000	0.169	0.325	0.392	0.443	0.280	0.384
100	398108	0.0	0.0	0.0	0.0	0.0	0.0
200	2829	0.833	0.905	0.954	0.959	0.894	0.932
200	40000	0.557	0.464	0.706	0.619	0.631	0.542
300	5197	0.989	0.921	0.997	0.967	0.993	0.944
300	90000	0.506	0.599	0.737	0.704	0.622	0.652
400	8000	0.836	0.923	0.969	0.967	0.903	0.945
400	160000	0.593	0.592	0.791	0.725	0.692	0.659
500	11181	0.900	0.937	0.982	0.978	0.941	0.957
500	250000	0.536	0.587	0.779	0.751	0.657	0.668

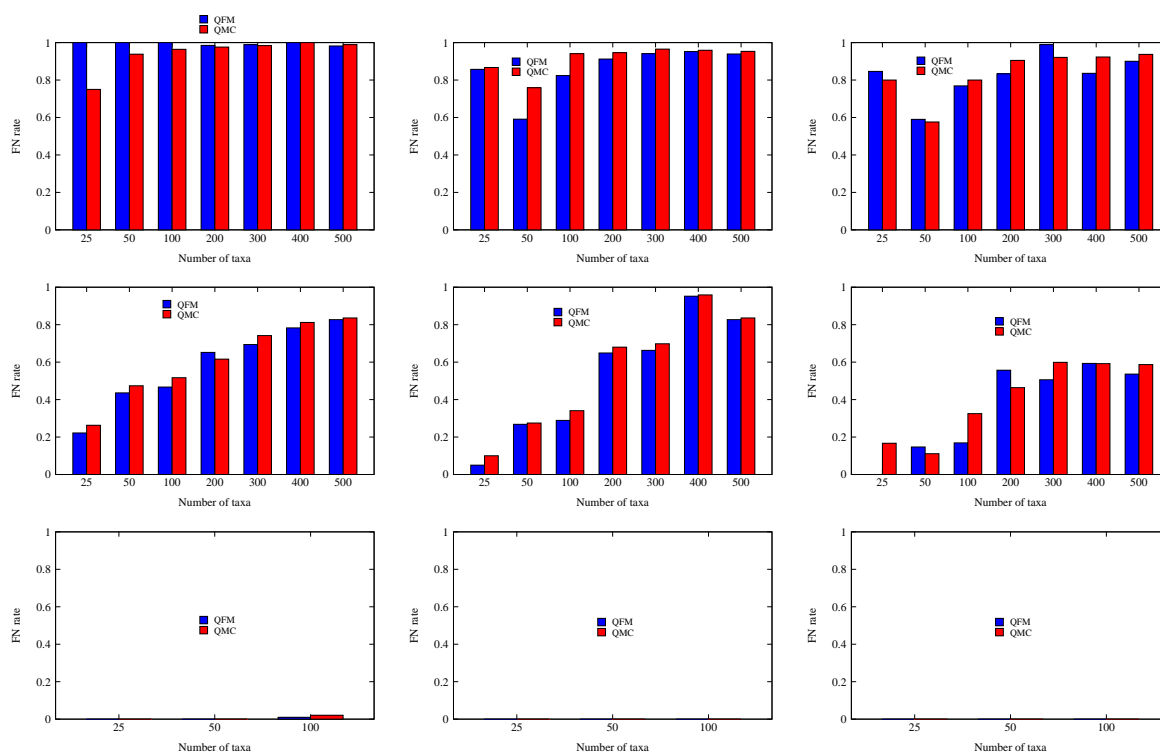


Figure 4.3: Missing branch rates of QFM and QMC on different datasets (with different number of taxa, number of quartets and the consistency level). From top to bottom: the number quartets are $n^{1.5}$, n^2 , and $n^{2.8}$, where n is the number of taxa. From left to right: 70%, 80% and 90% of the input quartets are consistent with the model species tree. We did not run our method on more than 100 taxa when the number of taxa is $n^{2.8}$ (since these are computationally intensive and could not be run within a reasonable time limit. Moreover, this model condition is less revealing since both QMC and QFM can reconstruct the true species trees on these datasets.)

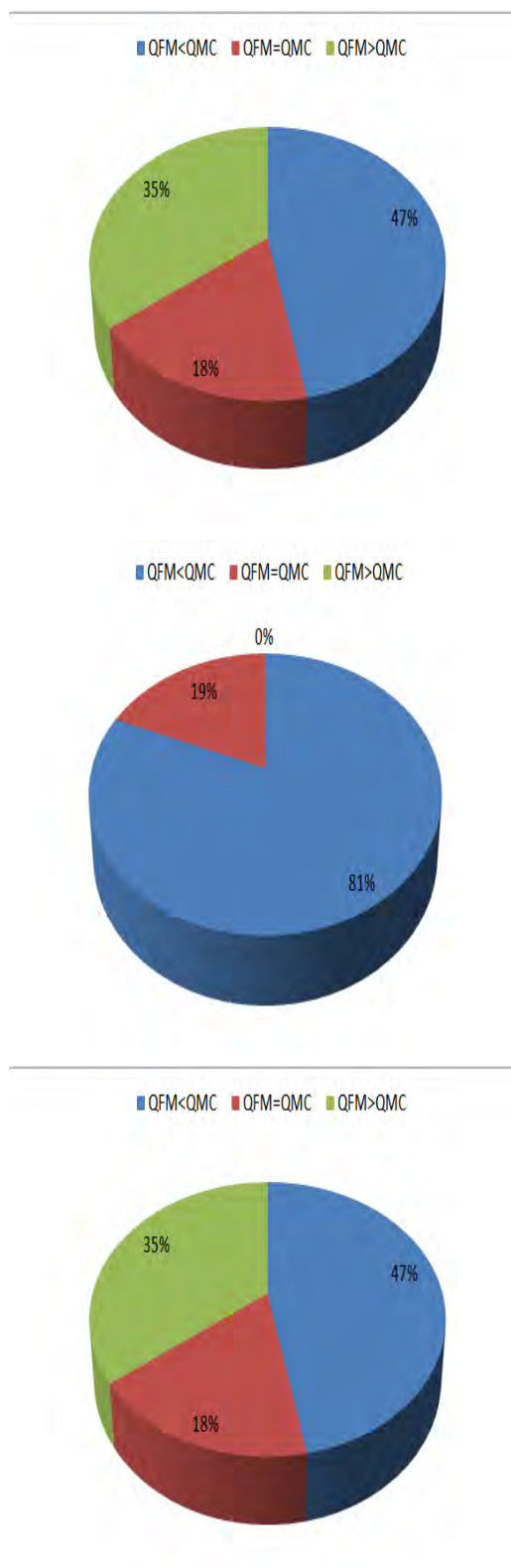


Figure 4.4: Illustration for the proportion of datasets on which the performance of QFM is better than, worse than or equal to QMC. From top to bottom: charts are drawn for Table 4.11, Table 4.12 and Table 4.13.

A close observation of the results reveals the following facts:

- QFM outperforms QMC in terms of FN rate in almost all cases. And in many cases, we observe substantial improvement for QFM.
- In terms of RF rate our method outperforms QMC in most of the cases.
- At lower consistency rate ($c = 70\%$ or 80%) our method performs better than QMC.
- Our method performs better than QMC for large quartet sets.

QFM is a new promising divide and conquer supertree method having an algorithmic appeal which is able to return more accurate tree than QMC does. Another advantage of QFM is that it is flexible in choosing the partition score function. QFM can be customized to take different scoring functions (i.e., $s - v$, s/v , etc.) without making any change in the algorithmic construct. We have tested that QFM may not give the same result for different scoring functions for the same dataset. In other words, maximizing the ratio of number of satisfied and violated quartets (as done by QMC) or maximizing the difference between number of satisfied and violated quartets may not give best result for all datasets. So for different datasets we may need to adapt different scoring function. QFM provides us with the flexibility to change the scoring function as needed. In future we shall try to make our algorithm self adaptable to the appropriate scoring function.

4.4 Experiment on Biological Data

We tested our algorithm on a biological dataset too. In this case, there is no model tree to compare with. So we have compare our estimated tree against a biological model tree. This model tree is hypothetical which is obtained by thorough analysis of a biological data set, and believed by experts to be the most accurate estimate of the history of that set of taxa.

Our biological dataset consists of 18 gene trees on 25 taxa (species), obtained from TreeBASE [29]. This dataset has originally been used to study the efficacy of species tree methods at the family level in birds, using the Australo-Papuan Fairy-wrens (Passeriformes: Maluridae)

clade [17]. This dataset considers 25 taxa (species) from 4 families of birds. The families are: *Amytornis*, *Stipiturus*, *Malurus* and *Clytomias*.

We took 18 gene trees over 25 birds from these 4 families. We decomposed every gene tree into its induced quartets which called *embedded quartets* [32, 41]. Then we take the union of all these quartets (multiple copies of a quartet is retained). In this way we have 227,700 quartets. We used these quartets to estimate a species tree using our method QFM (approach IIIa). The tree estimated by QFM is shown in Figure 4.5.

Since we have applied our method for biological data, we are interested to analyze the estimated tree with respect to the cluster(s) (group of species under a common ancestor) identified. The estimated tree shows the following characteristics.

- The clusters identified by this tree are consistent with prior studies ([17]). From Figure 4.5 we see that all the *Amytornis* birds have been grouped together. Similarly, *Stipiturus*, *Malurus* and *Clytomias* birds have been grouped separately. So clearly, the clusters have been identified correctly.
- Also, as suggested in an earlier work [17], QFM placed the grasswrens (*Amytornis*) as the sister to the rest of the family, and the emu-wrens (*Stipiturus*) as the sister to fairywrens (*Malurus*, *Clytomias*).

These characteristics show that the tree estimated by QFM is same to the biological tree believed by experts for this set of species. Therefore, QFM performs well on biological dataset which indicates the qualitative significance of our algorithm.

4.5 Summary

In this chapter we have explained our experimental works and presented the results obtained. The chapter starts with a description of accuracy measure in Section 4.1. In Section 4.2 we briefly described the technique of a simulation study. Then, in Section 4.3 we described the way we performed our simulation on simulated data - data generation (Section 4.3.1), result summary (Section 4.3.2) of 6 approaches of QFM and analysis of accuracy of QFM over QMC

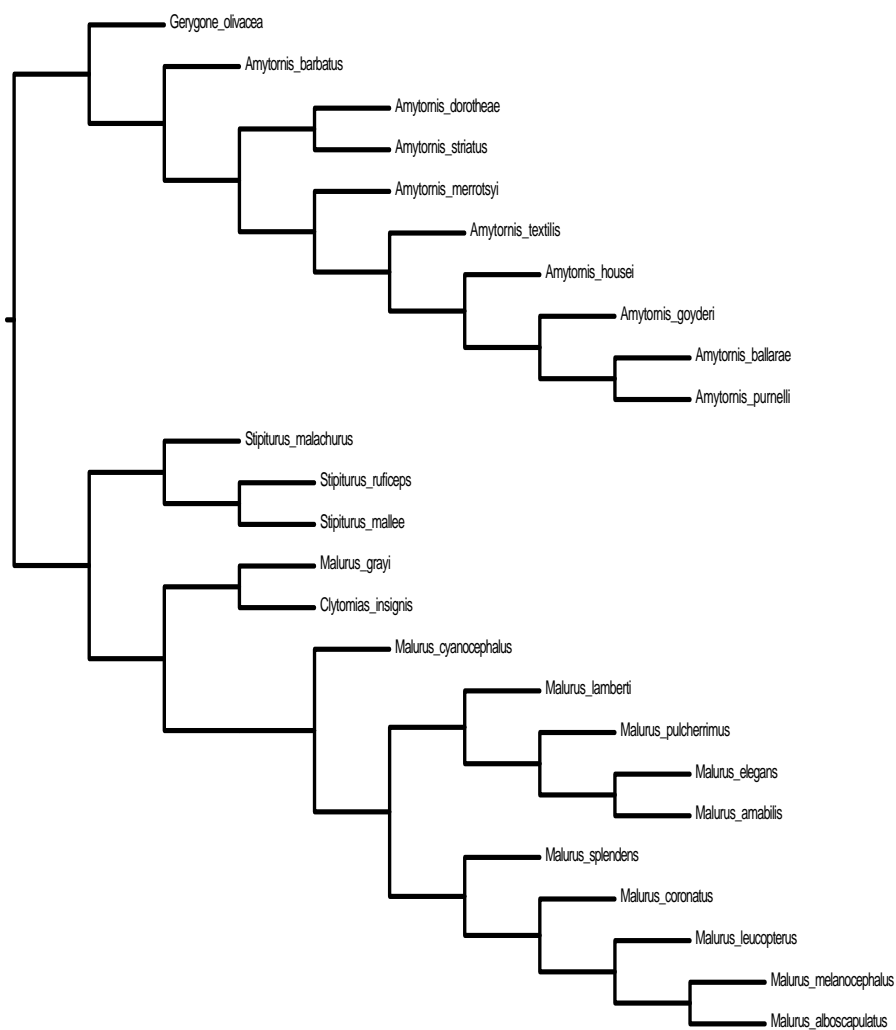


Figure 4.5: The 25 species bird phylogeny estimated by QFM using the 227,700 embedded quartets from 18 gene trees.

(Section 4.3.3). Finally, in Section 4.4 we explained our experimental result on biological dataset.

Chapter 5

Conclusion & Future Work

In this thesis we have presented a new quartet based phylogenetic approach. Also we showed superiority of our method over the approach which is known to be the best to date in quartet based phylogeny. In addition, we have developed a software tool which is an all in one tool to simulate and analyze our method and the method we compared with.

We have started with an introductory overview on phylogeny Chapter 1. In that chapter we have described different approaches of phylogenetic tree construction and discussed various practical applications of phylogeny. We also gave motivation and literature review of quartet based phylogeny; and described our objective and main results of this thesis.

In Chapter 2 we have introduced some basic concepts and terminology related to phylogeny and algorithm theory. In particular we focused on the concepts related to quartet based phylogeny.

In Chapter 3 we have presented our algorithm QFM. We have described the six different versions of our algorithm. In Chapter 4 we have presented our simulation results for simulated and biological datasets. We have also showed the comparative analysis of our method and QMC (the method we compared with).

In Chapter A we have described our software tool ‘QTREE’ and provided an user manual. In Chapter B we have shown some performance curves as part of the experiments described in Chapter 4. We listed the experimental running times in Chapter C.

This thesis goes a long a way to devise a more accurate quartet based phylogenetic approach

compared to the existing approaches. However, we have figured out some more tasks to be done to make our approach more challenging which could not be done in this thesis due to time limitation. We plan to accomplish those tasks in future.

1. In this thesis, we have generated simulation data using uniform distribution. We plan to generate input quartet sets using geometric distribution as done in [32].
2. We indicated that our algorithm can be modified to take different scoring functions. But we have not yet identified yet which scoring function will give better result in what type of input data. Our plan is to make our algorithm self adaptable to choose appropriate scoring function.
3. The current implementation of our algorithm is not very efficient. By efficient coding our algorithm will run much faster. We will focus on this immediately.
4. At present, our software tool 'QTREE' can be used to simulate and analyze the two methods - QFM and QMC. We want to incorporate other approaches of quartet based phylogeny in this tool so that the researchers developing new methods find this software very helpful to compare the new method with existing ones.

Appendix A

QTREE: A Simulation Tool for Quartet Based Phylogeny

QTREE

QTREE is a software tool to simulate and analyze two quartet based supertree methods, namely, *QFM* (Quartet FM) and *QMC* (Quartet MaxCut).

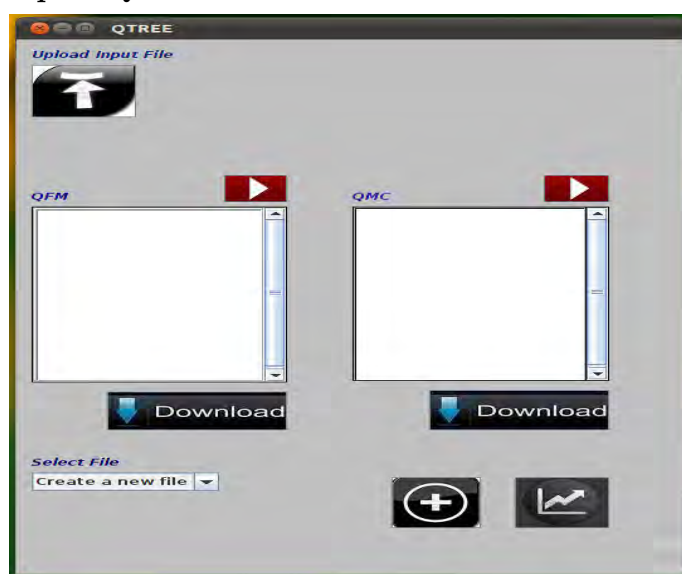
Functions

- User can **upload** an input file (in '*.txt' format) and **simulate** both QFM and QMC on the input.
- QTREE shows the output of QFM and QMC in two different displays.
- User can **download** the output file (in '*.txt' format) for each individual run.
- User can store the simulation results by creating a **graph dataset** (i.e., by creating a file).
- User can **create** multiple files to create multiple **graph datasets** and select one from multiple files to store result of an individual run.
- User can **select** a file (i.e, a graph dataset) to **draw graph**.

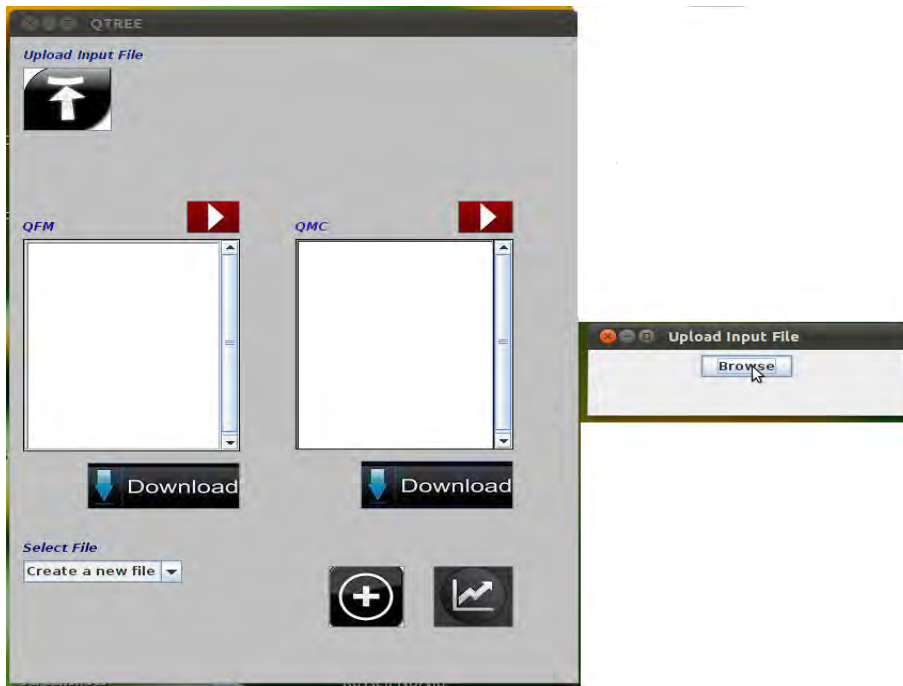
Requirements	Linux OS, Basic Perl installation, JDK 1.5 or above
Input File Format	The input file contains a number of quartets in Newick format (e.g., ‘((A, B), (C, D))’). Each quartet must be ended with a semicolon ‘;’ and there must be only one quartet per line.
Output File Contents	The outfile file contains the estimated supertree in Newick format.
Graph Output	QTREE provides the facility to draw the performance curve of QFM vs QMC based on the results of multiple runs. The ‘X’-axis of the output curve denotes the ‘Number of Taxa’ and the ‘Y’-axis denotes the ‘Number of Satisfied Quartets’. When user chooses a file to store the result of a simulation, the three values - taxa count, number of satisfied quartets by QFM and number of satisfied quartets by QMC are being inserted into the chosen file.

A.1 User Manual

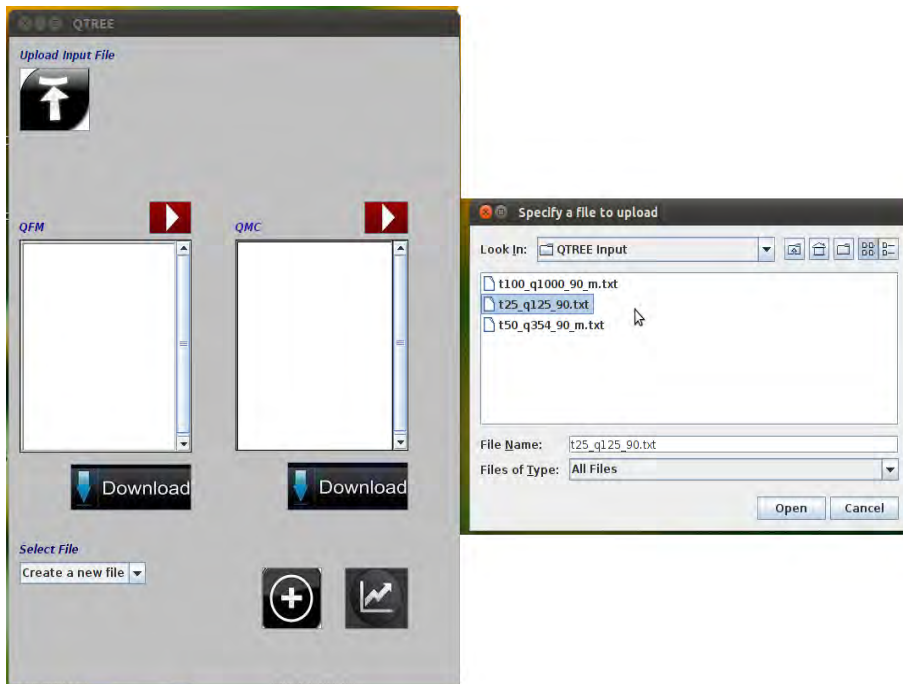
Open QTREE



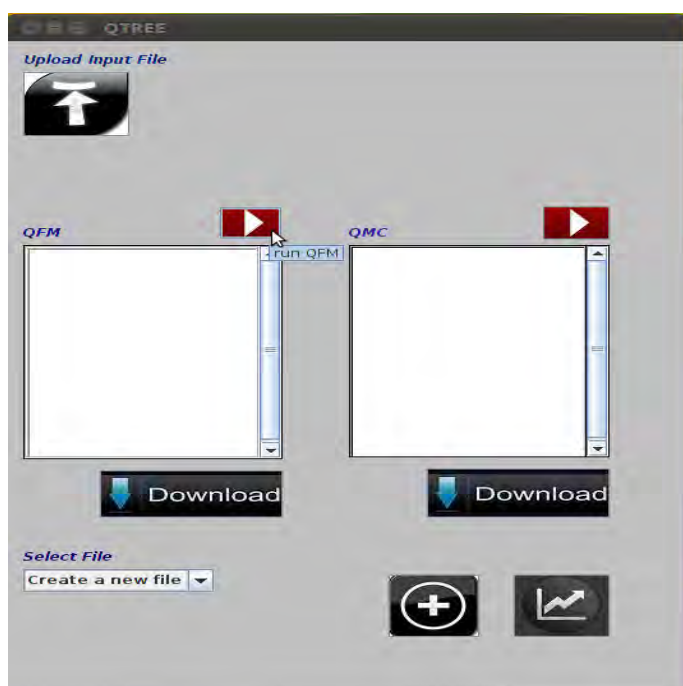
Click ‘Upload input file’ button. A ‘browse’ dialog box will appear.



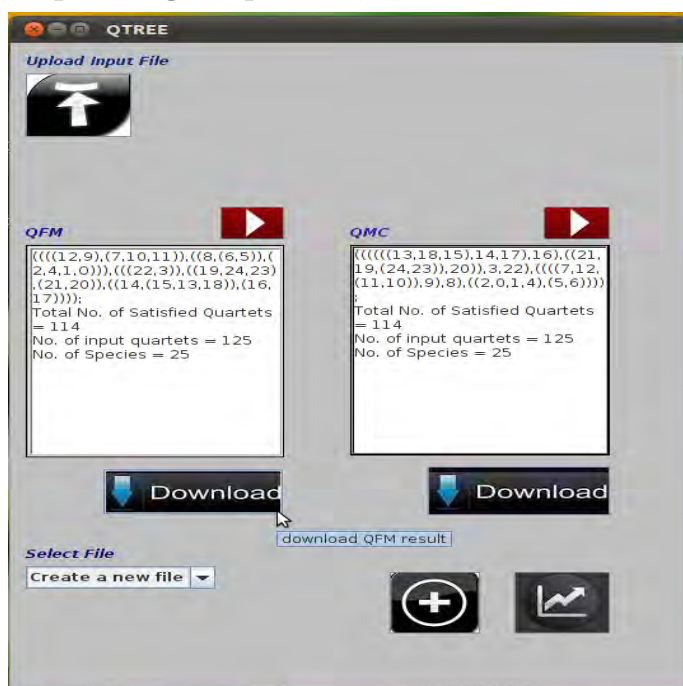
Select input file location and double click the selected file.



Click 'run QFM' button to start simulation of QFM. Similarly click 'run QMC' button to start simulation of QMC

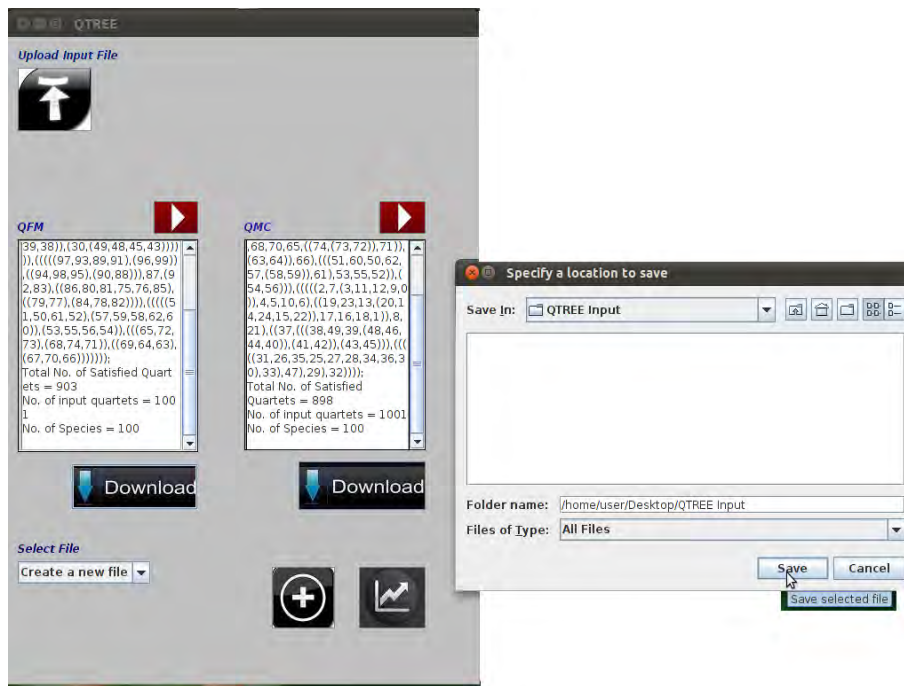


Outputs will be displayed in associated scroll panels. To download corresponding output file, Click 'download' button.

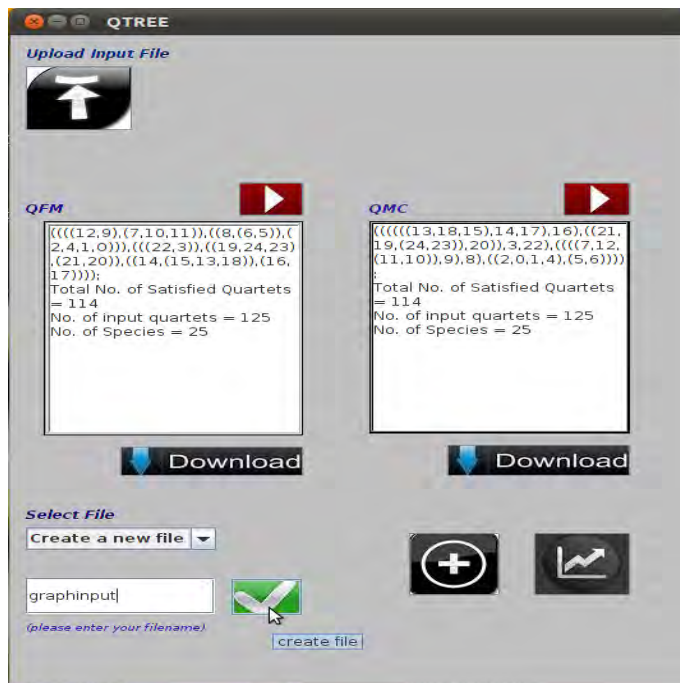


A save dialog box will appear. Choose a location to save outputfile.

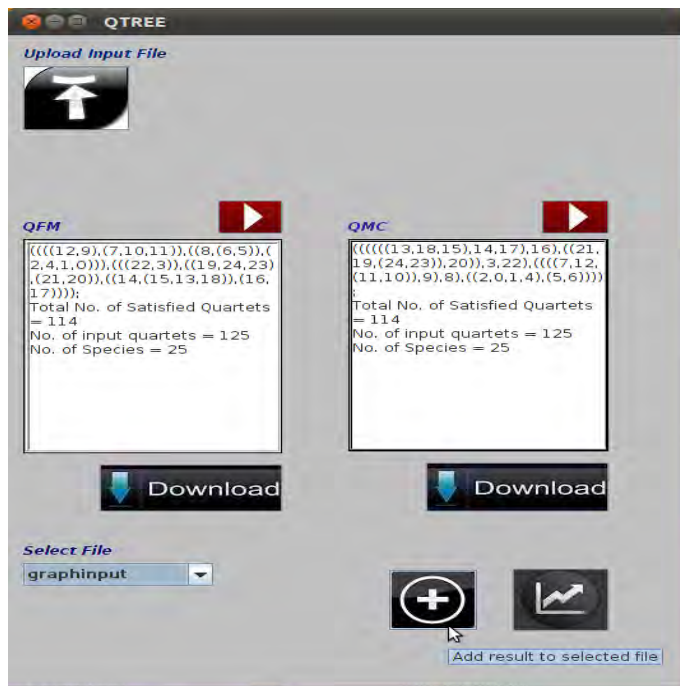
92APPENDIX A. QTREE: A SIMULATION TOOL FOR QUARTET BASED PHYLOGENY



Select or Create a file to store result as graph input. Selecting ‘Create a new file’ option will ask for a file name. Enter file name and Click ‘Create file’ button.



Click ‘Add result to selected file’ button to append the result in selected file.



Click 'Draw curve for selected file' button to draw graph on selected file.




Another window will appear showing the graph.

94 APPENDIX A. QTREE: A SIMULATION TOOL FOR QUARTET BASED PHYLOGENY

QTREE

Upload Input File



QFM

```

(39,38)),(30,(49,48,45,43))))
),((((((97,93,89,91),(96,99))
),(94,98,95),(90,88))),87,(9
2,83),((86,80,81,75,76,85),
((79,77),(84,78,82))),((((5
1,50,61,52),(57,59,58,62,6
0)),(53,55,56,54)),((65,72,
73),(68,74,71)),((69,64,63),
(67,70,66))))));
Total No. of Satisfied Quartets = 903
No. of input quartets = 100
No. of Species = 100
                
```

Download

QMC



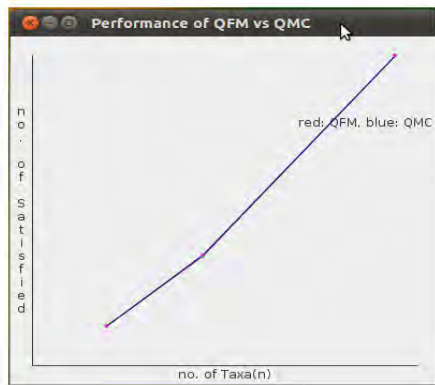
```

(68,70,65,((74,(73,72)),71)),
(63,64)),66),(((51,60,50,62,
57,(58,59)),61),53,55,52)),(
54,56))),((((2,7,(3,11,12,9,0
)),4,5,10,6),((19,23,13,(20,1
4,24,15,22)),17,16,18,1)),8,
21),((37,((38,49,39),(48,46,
44,40)),(41,42)),(43,45))),((
(31,26,35,25,27,28,34,36,3
0),33,47),29),32)))));
Total No. of Satisfied Quartets = 898
No. of input quartets = 1001
No. of Species = 100
                
```

Download

Select File

graphinput

Appendix B

Performance Curves of Six Approaches of QFM vs QMC

The performance curves have been drawn for the results summarized in Table 4.1 to Table 4.9.

Performance Curves for Approach Ia with $n^{1.5}$ input quartets

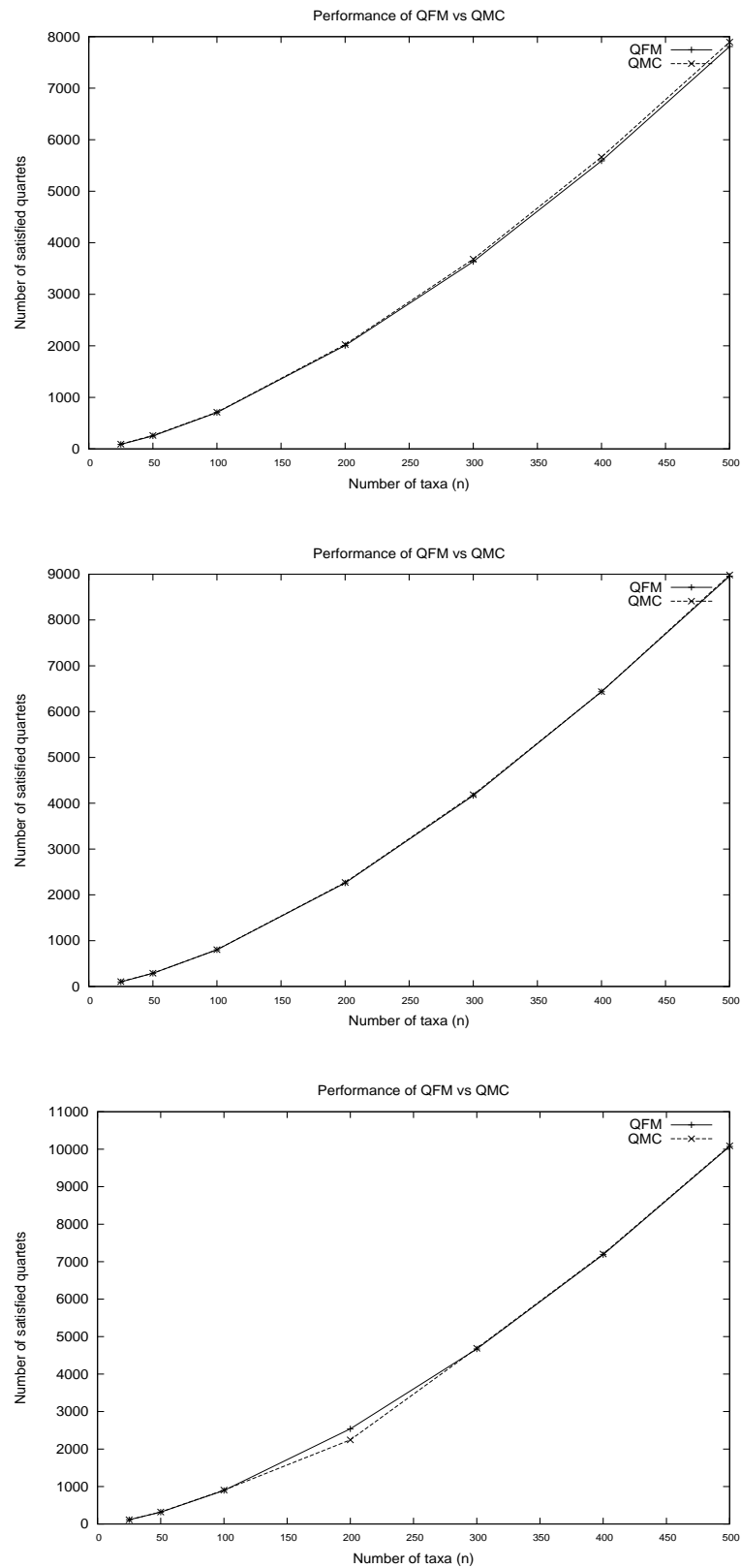


Figure B.1: Comparison of QFM - Ia and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - Ia with n^2 input quartets

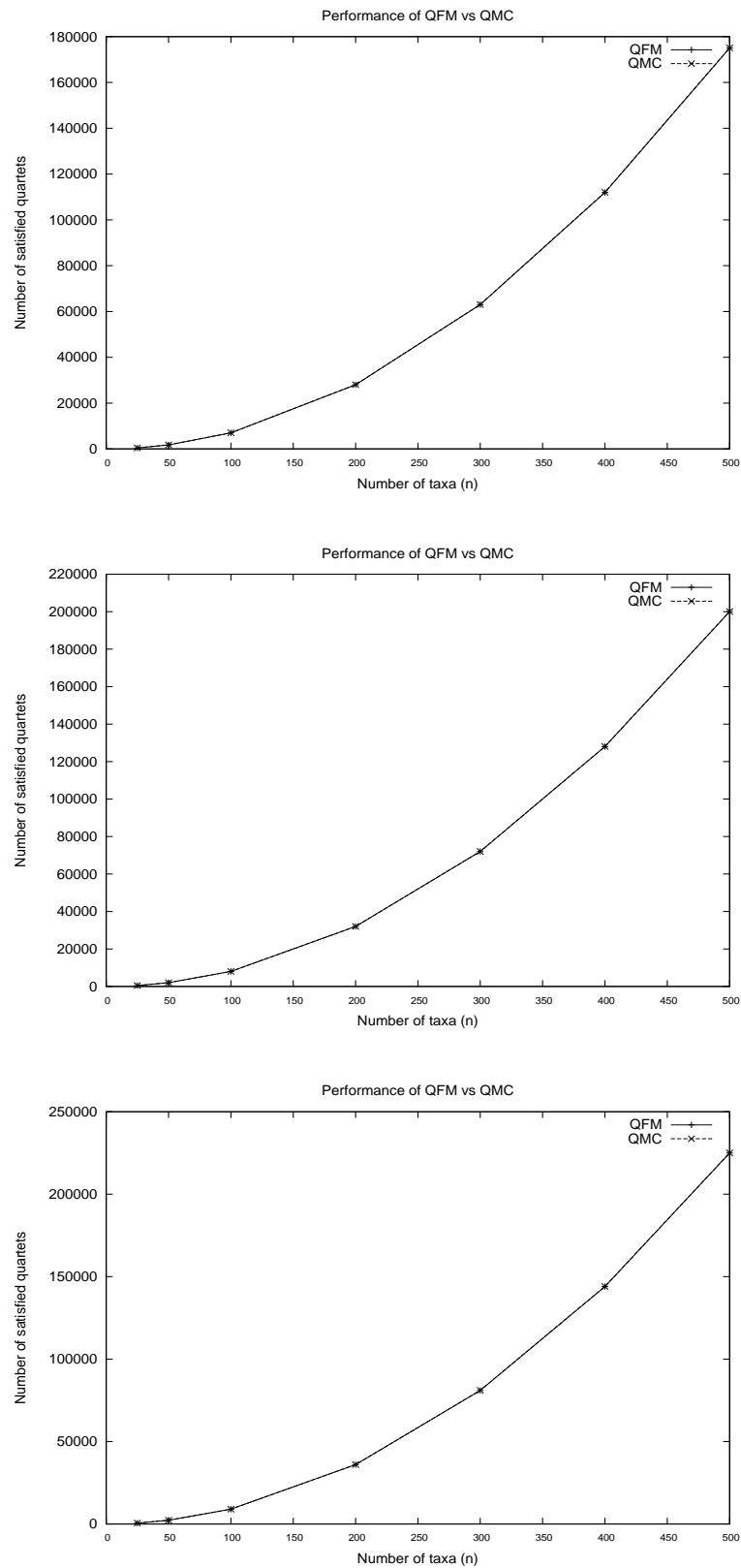


Figure B.2: Comparison of QFM - Ia and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - Ia with $n^{2.8}$ input quartets

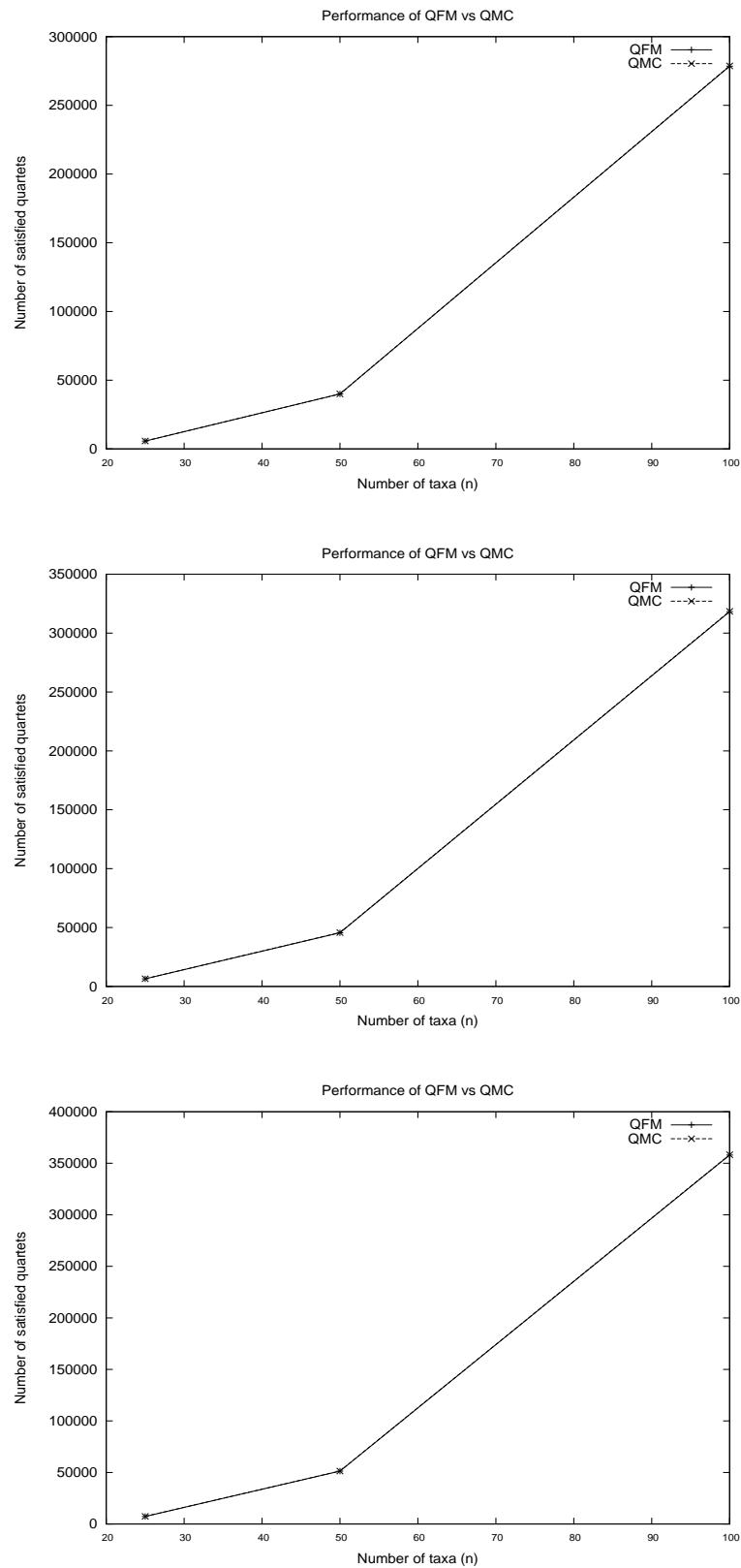


Figure B.3: Comparison of QFM - Ia and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - Ib with $n^{1.5}$ input quartets

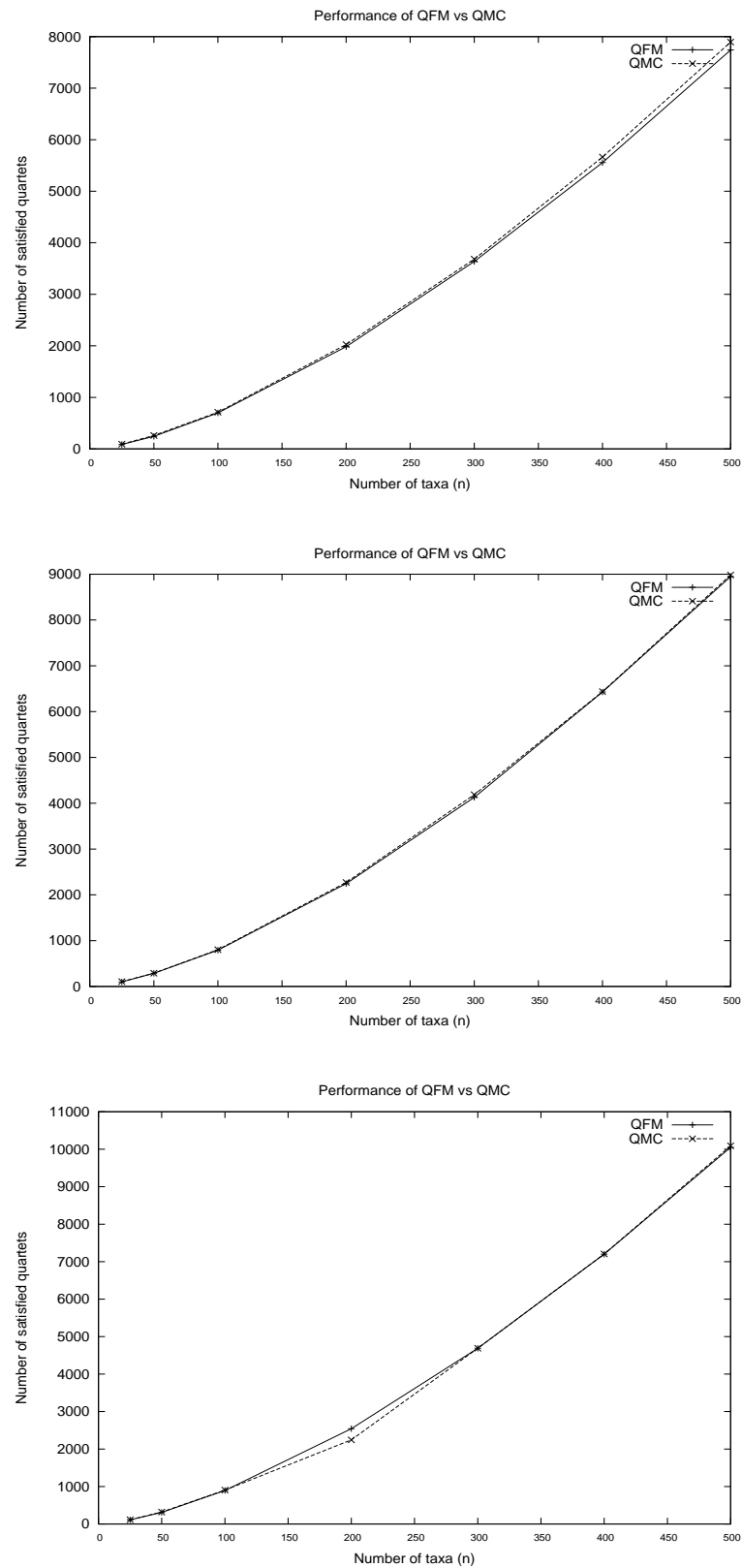


Figure B.4: Comparison of QFM - Ib and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

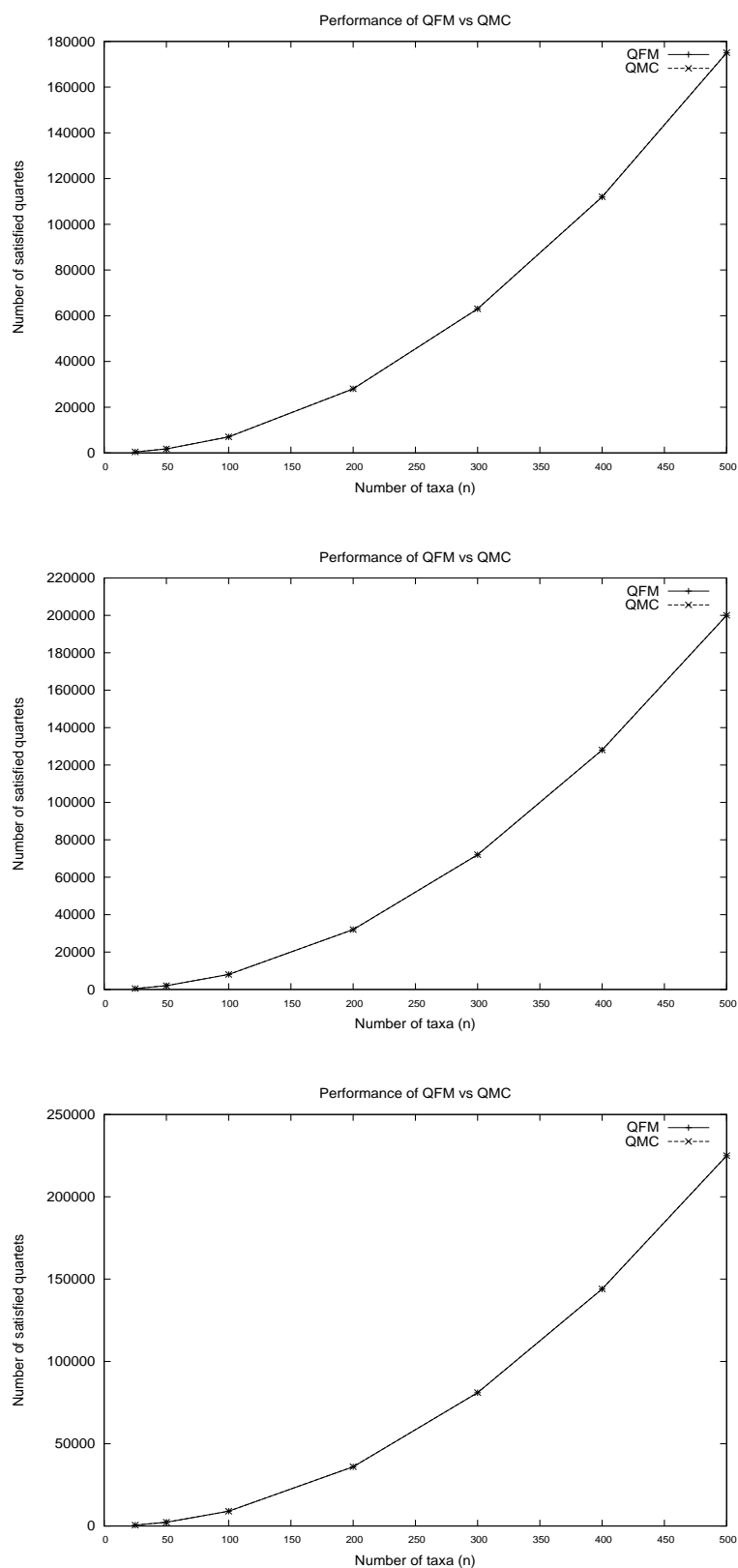
Performance Curves for QFM - Ib with n^2 input quartets

Figure B.5: Comparison of QFM - Ib and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - Ib with $n^{2.8}$ input quartets

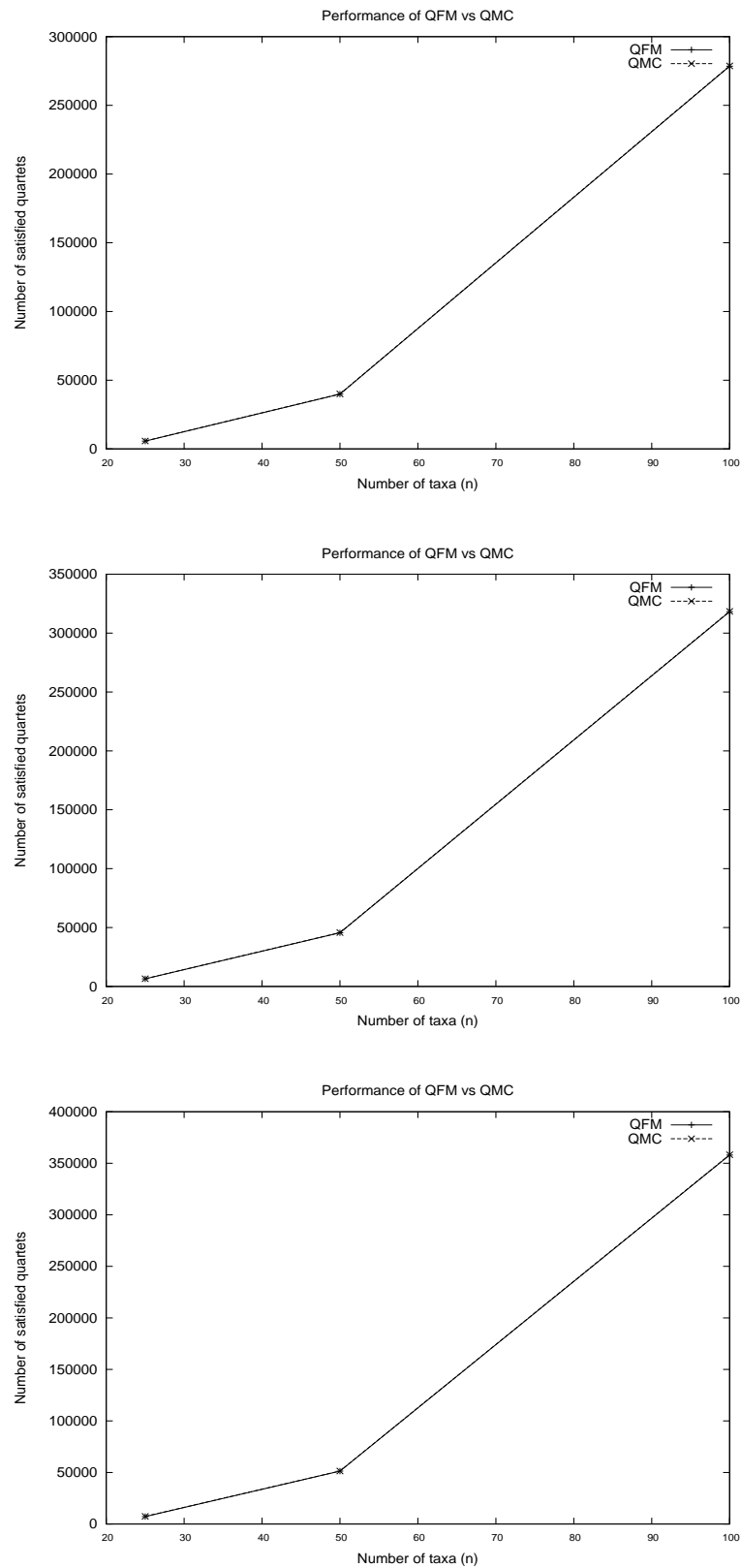


Figure B.6: Comparison of QFM - Ib and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

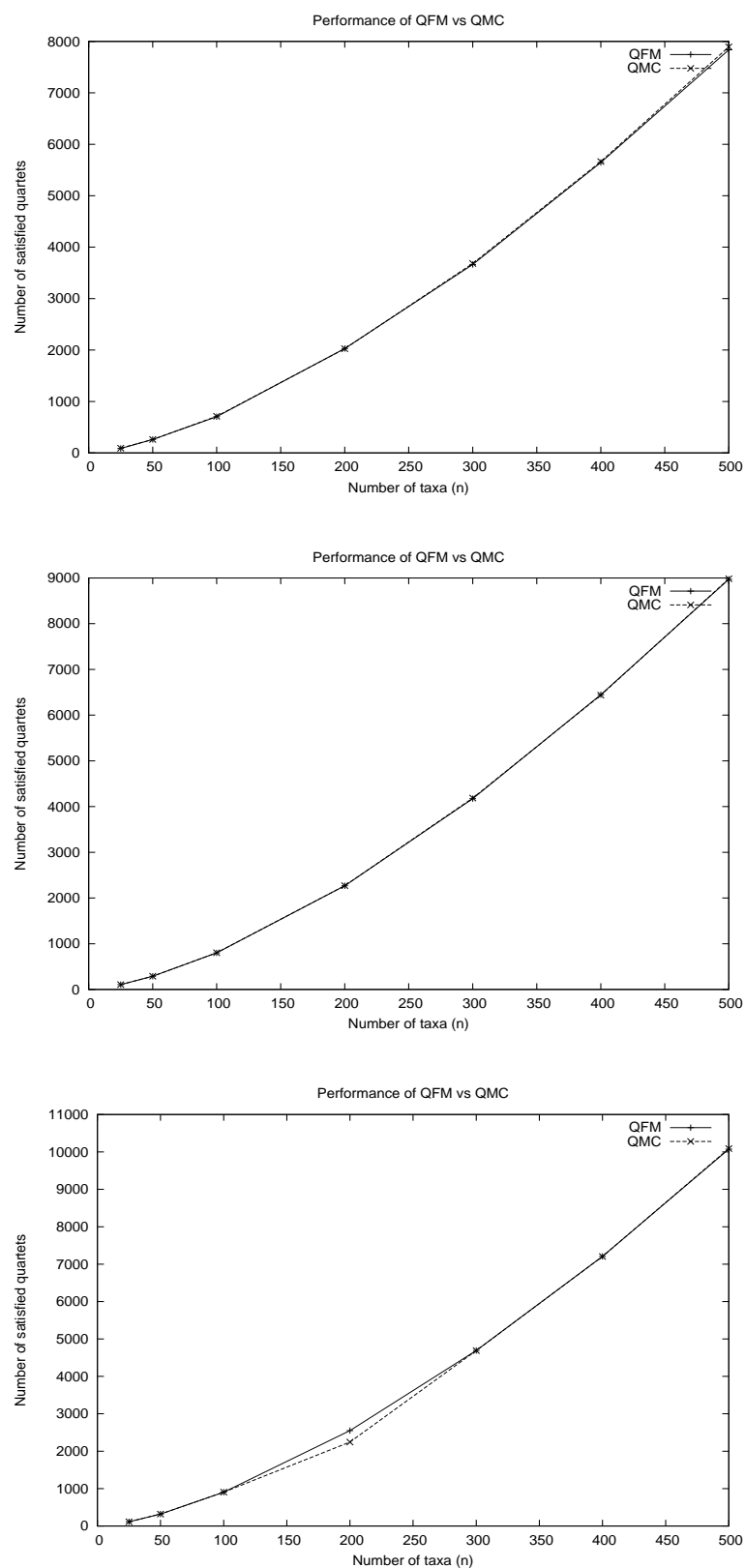
Performance Curves for QFM - IIa with $n^{1.5}$ input quartets

Figure B.7: Comparison of QFM - IIa and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - IIa with n^2 input quartets

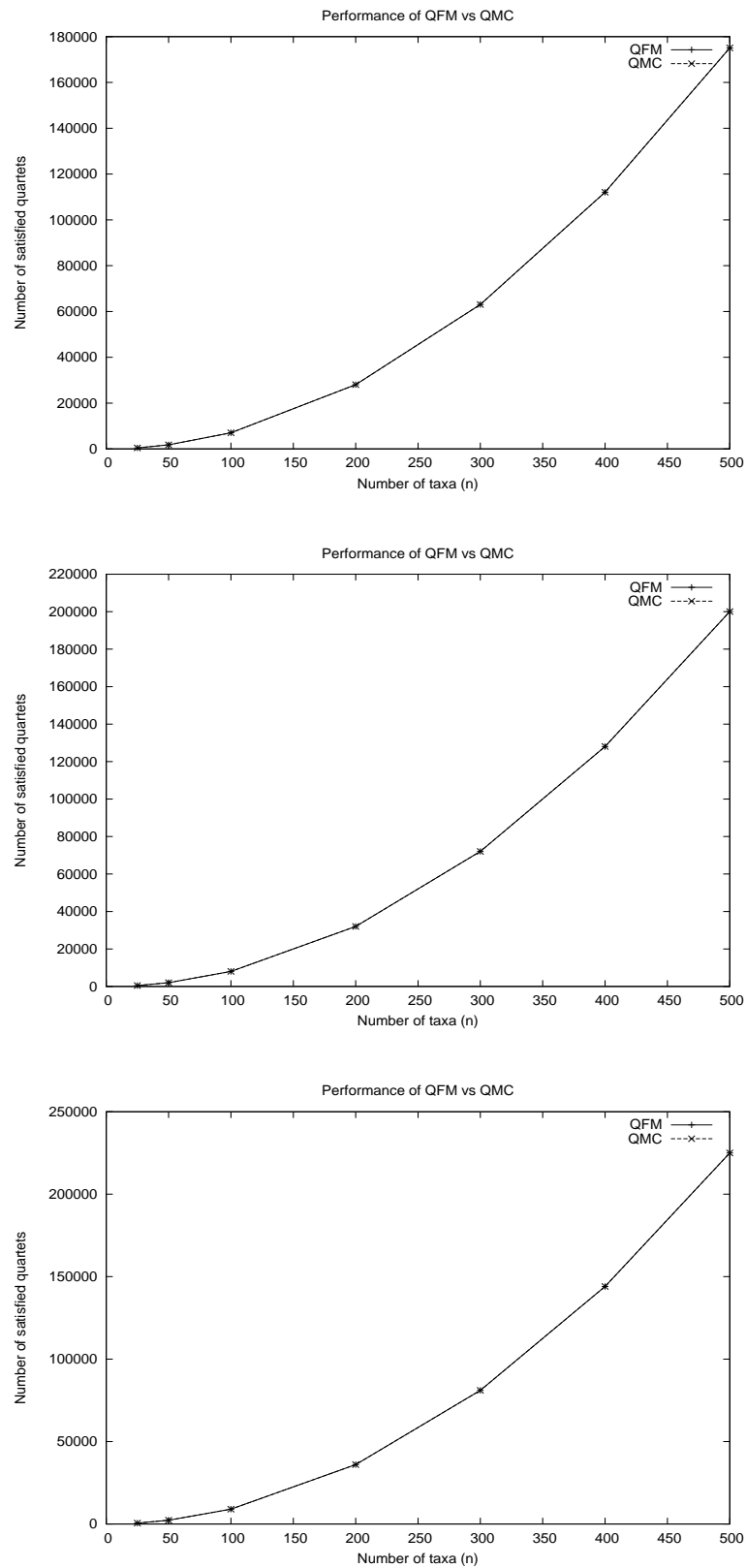


Figure B.8: Comparison of QFM - IIa and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

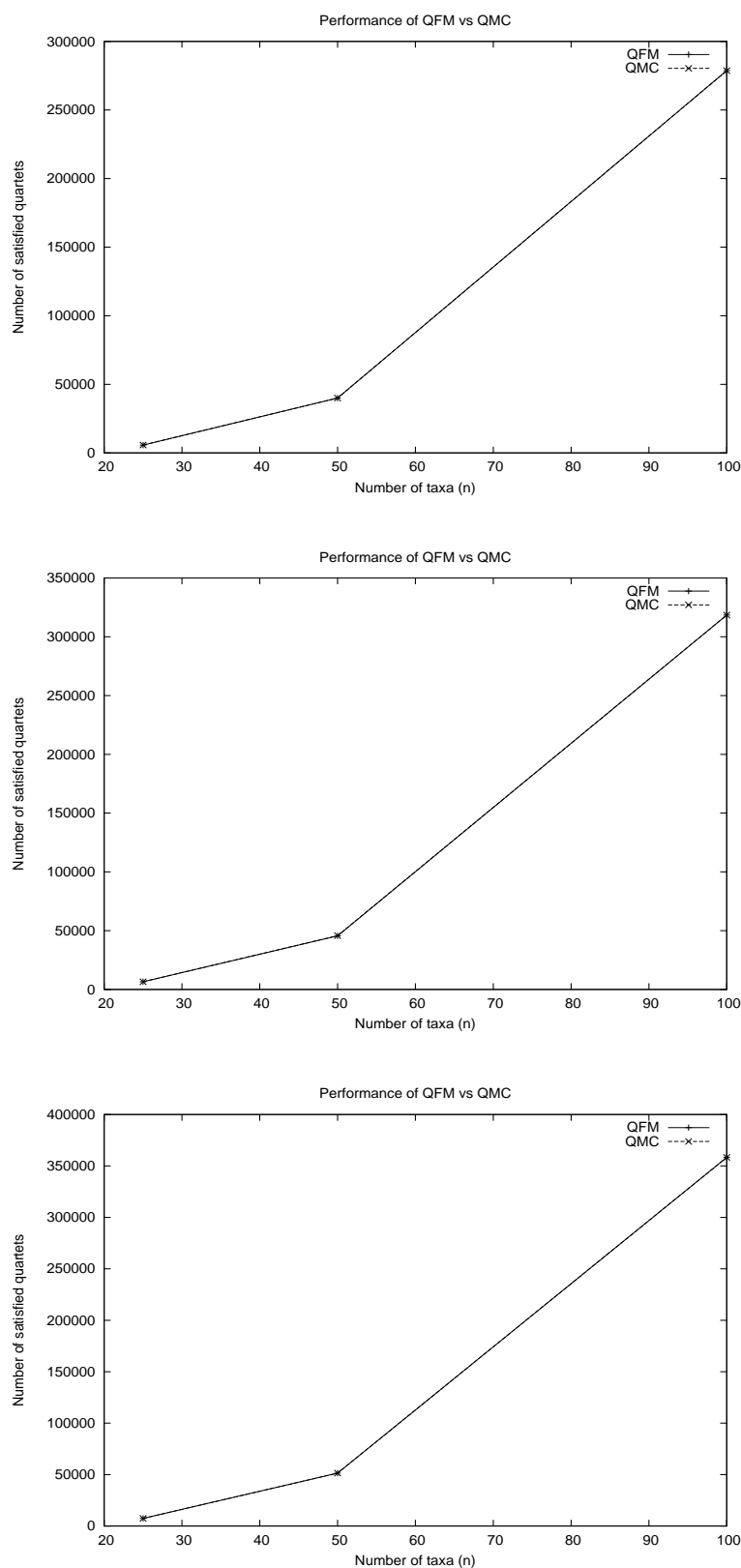
Performance Curves for QFM - IIa with $n^{2.8}$ input quartets

Figure B.9: Comparison of QFM - IIa and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - I Ib with $n^{1.5}$ input quartets

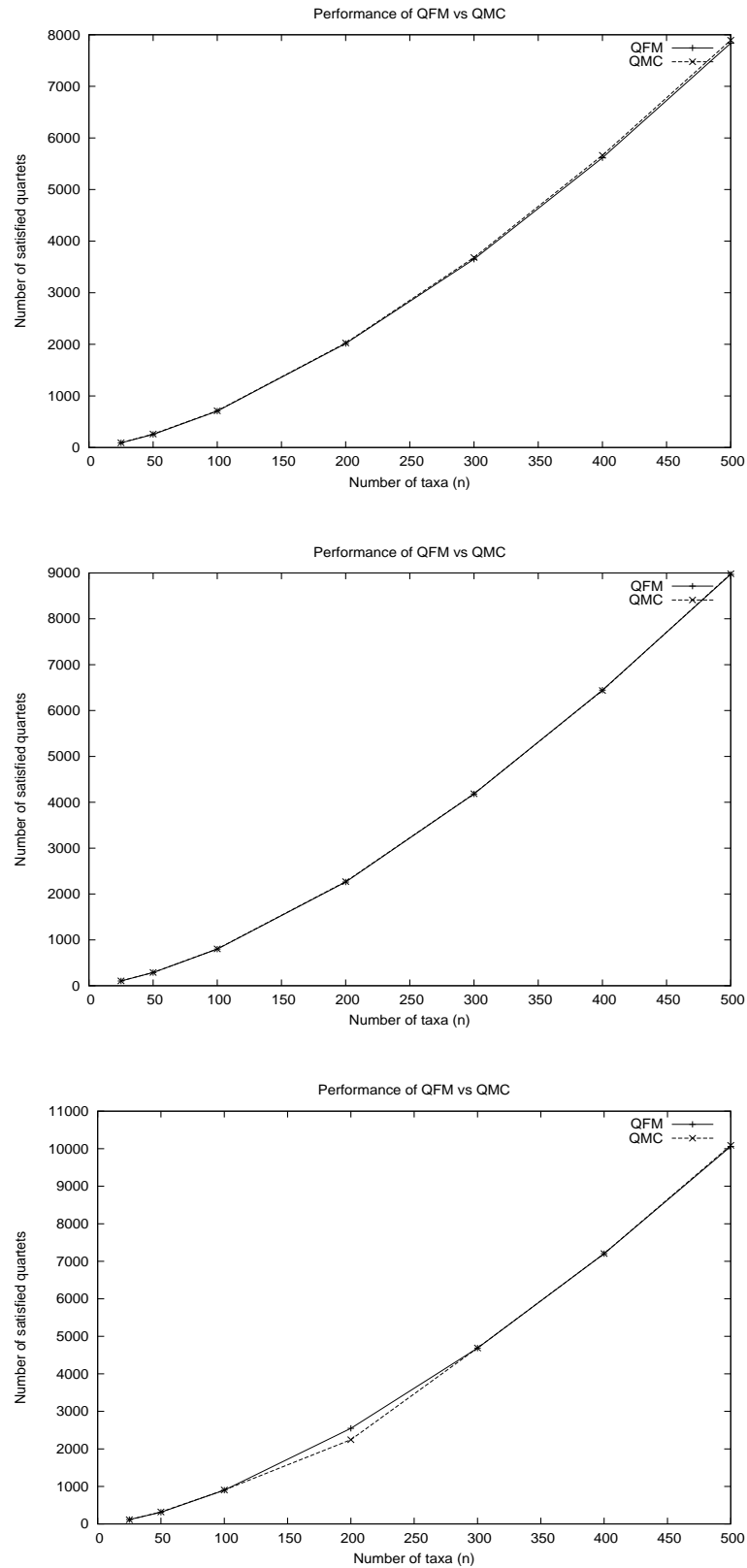


Figure B.10: Comparison of QFM - I Ib and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

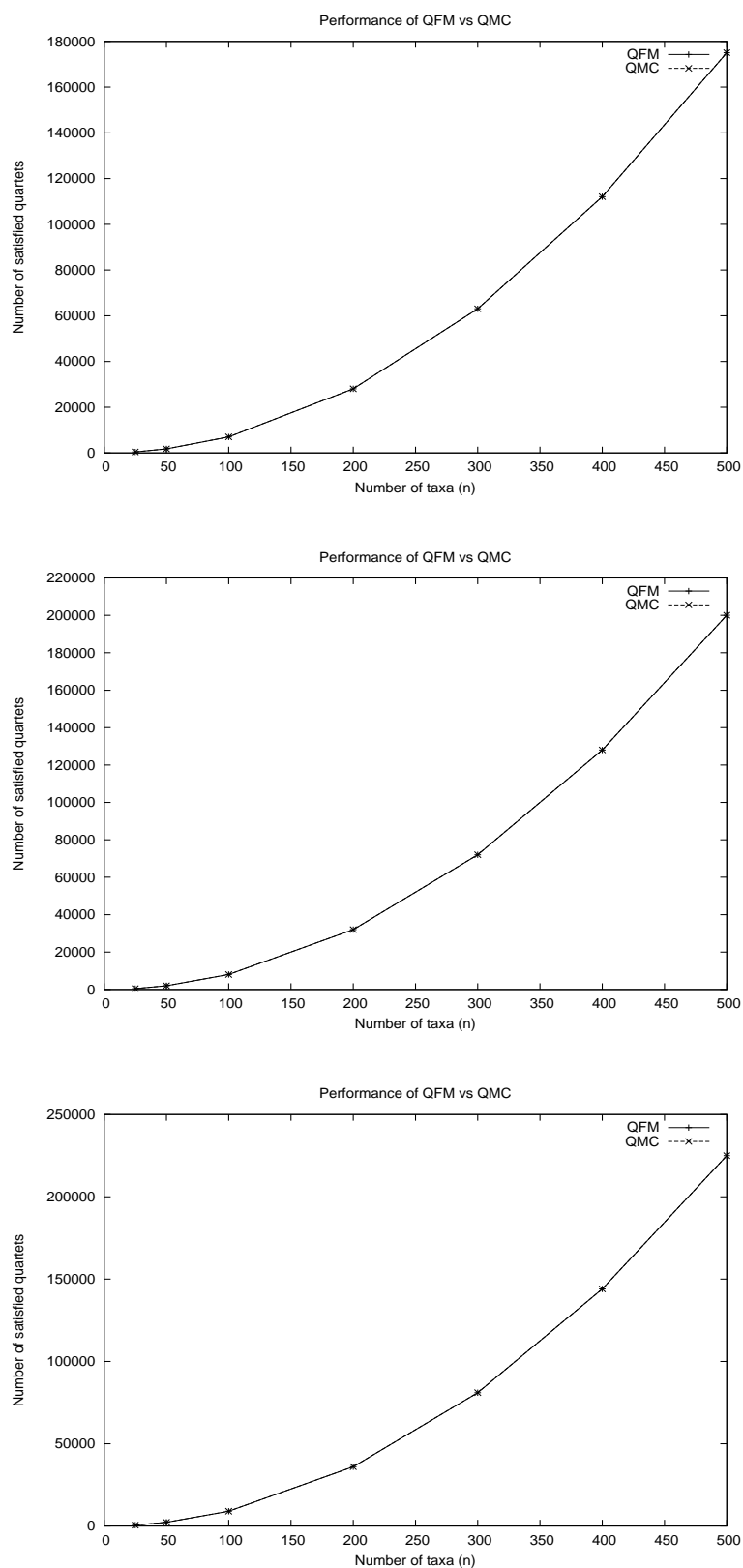
Performance Curves for QFM - IIb with n^2 input quartets

Figure B.11: Comparison of QFM - IIb and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - I Ib with $n^{2.8}$ input quartets

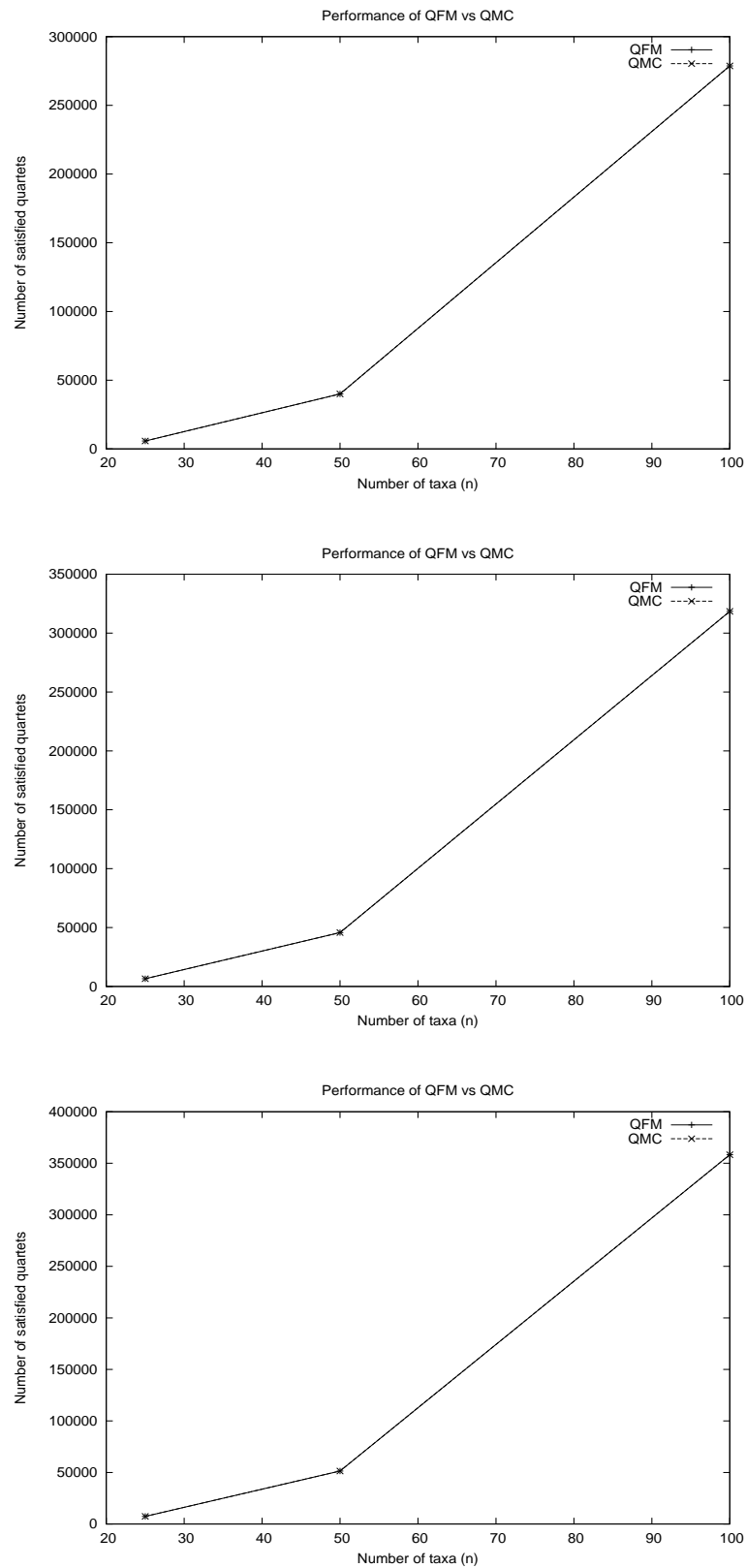


Figure B.12: Comparison of QFM - I Ib and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

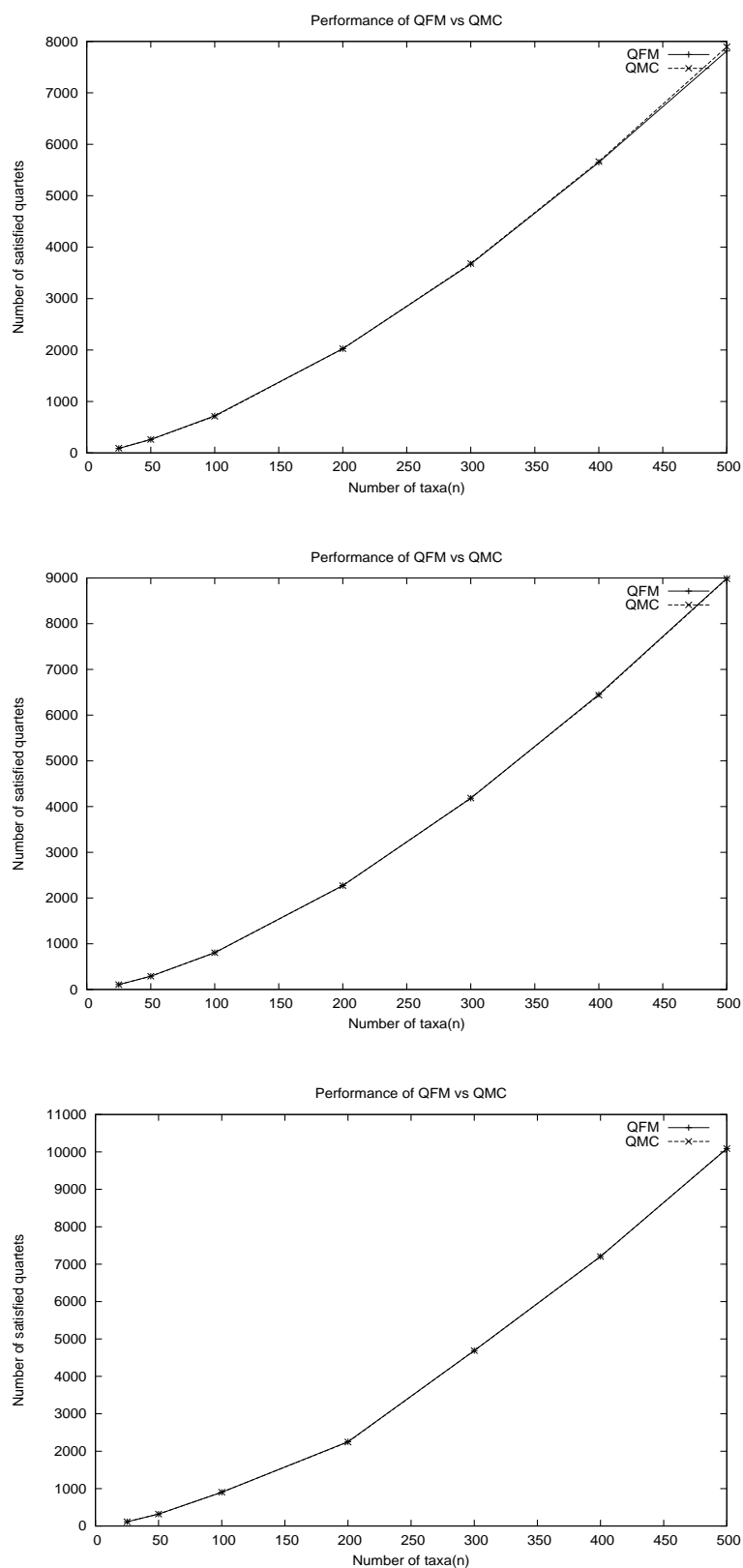
Performance Curves for QFM - IIIa with $n^{1.5}$ input quartets

Figure B.13: Comparison of QFM - IIIa and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - IIIa with n^2 input quartets

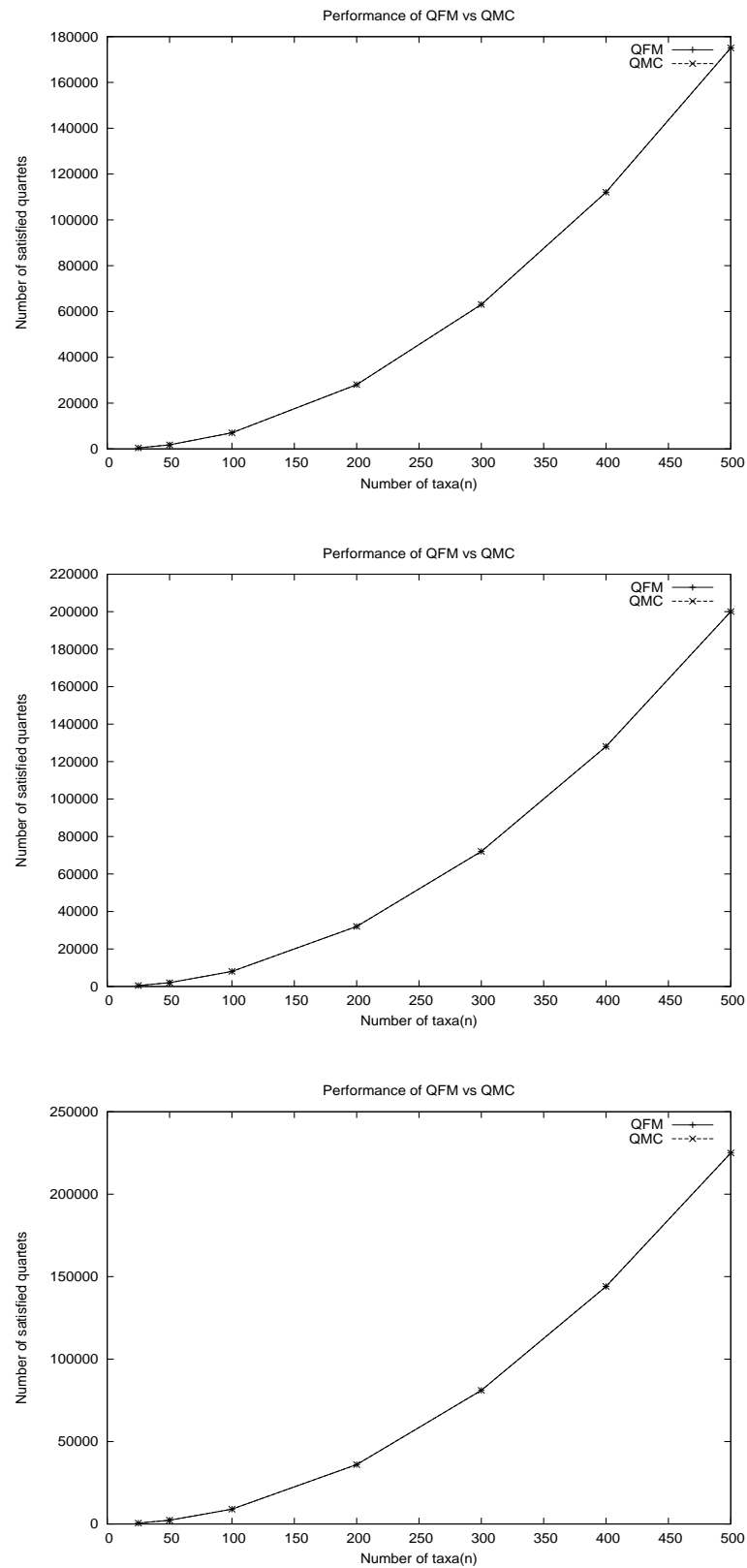


Figure B.14: Comparison of QFM - IIIa and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - IIIa with $n^{2.8}$ input quartets

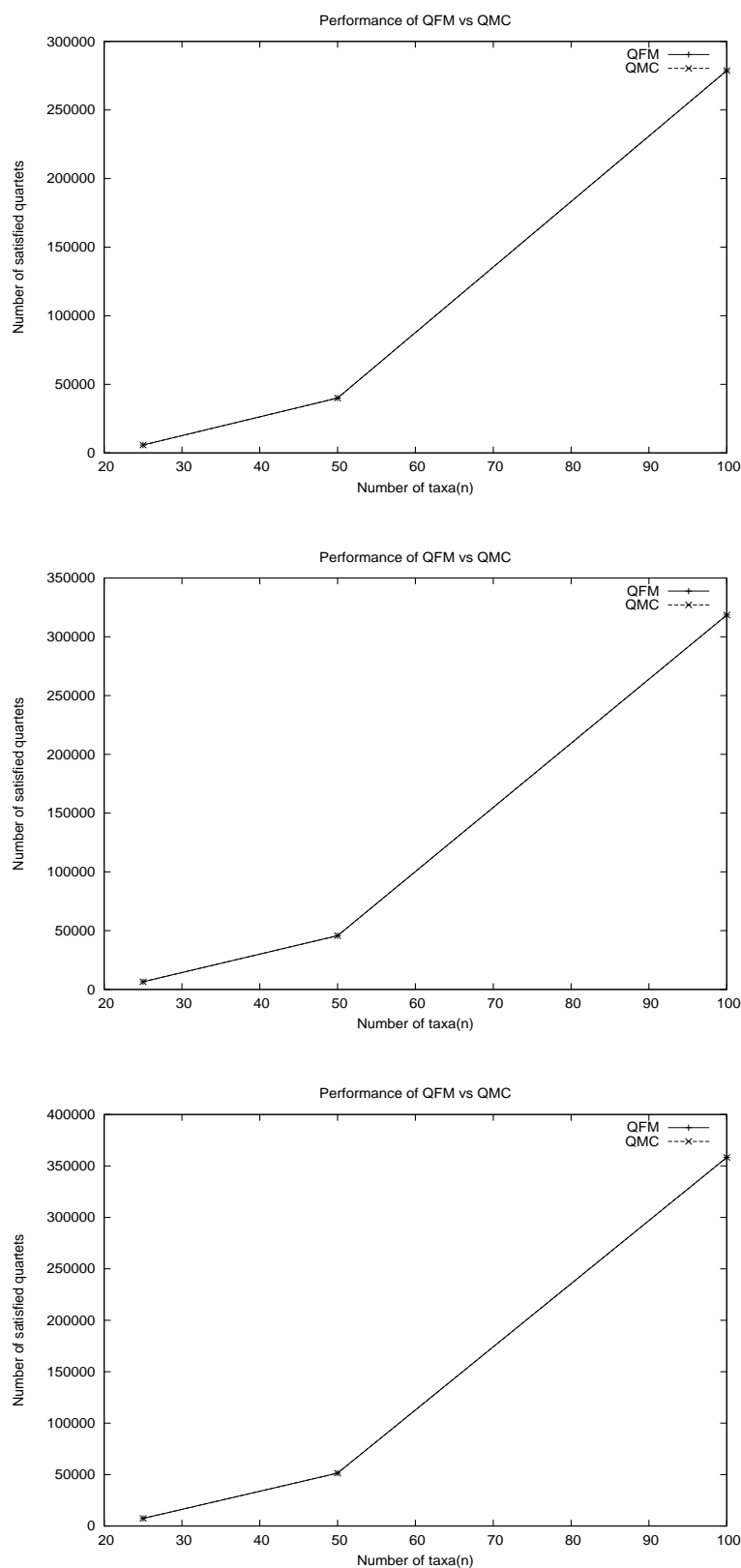


Figure B.15: Comparison of QFM - IIIa and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - IIIb with $n^{1.5}$ input quartets

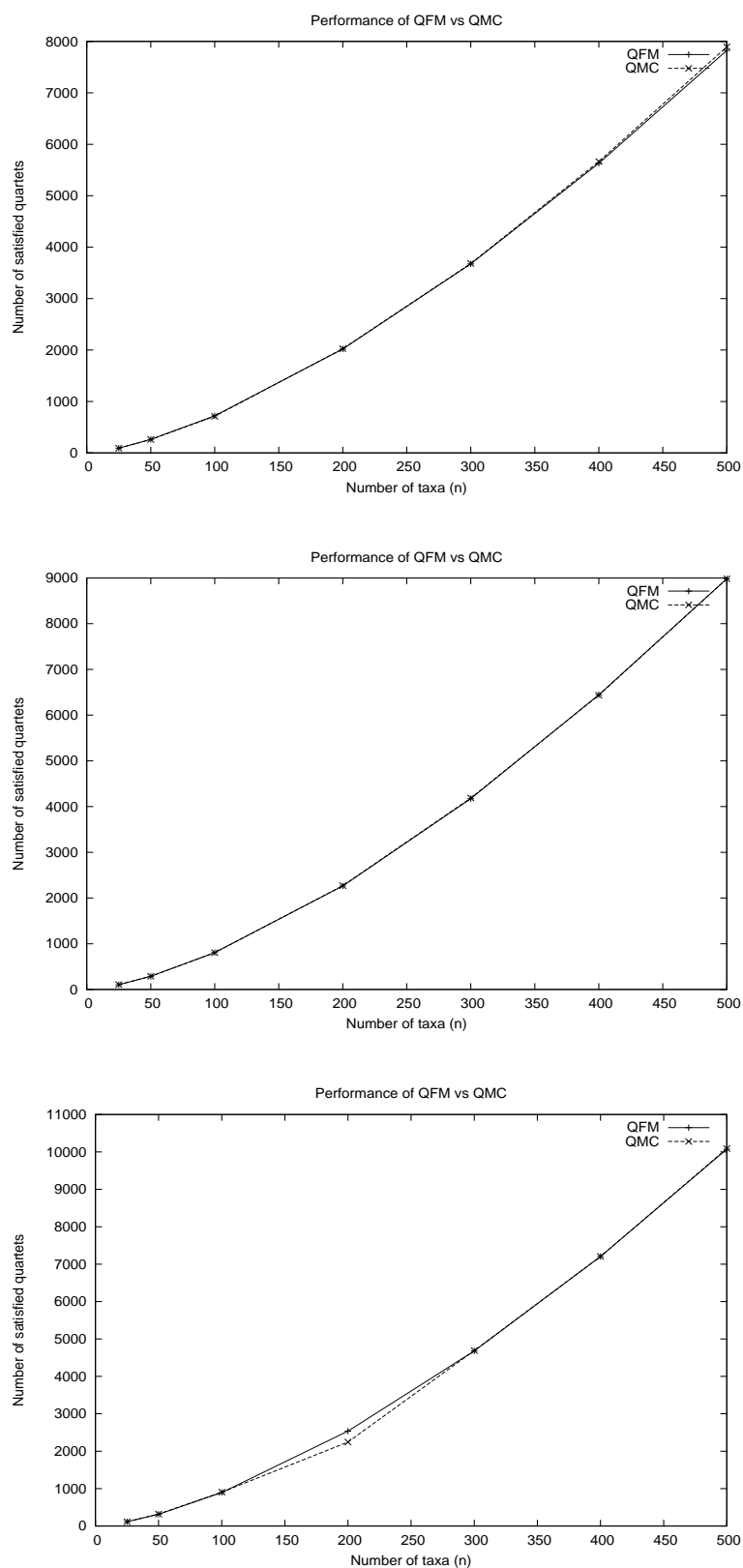


Figure B.16: Comparison of QFM - IIIb and QMC when size of quartet set = $n^{1.5}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

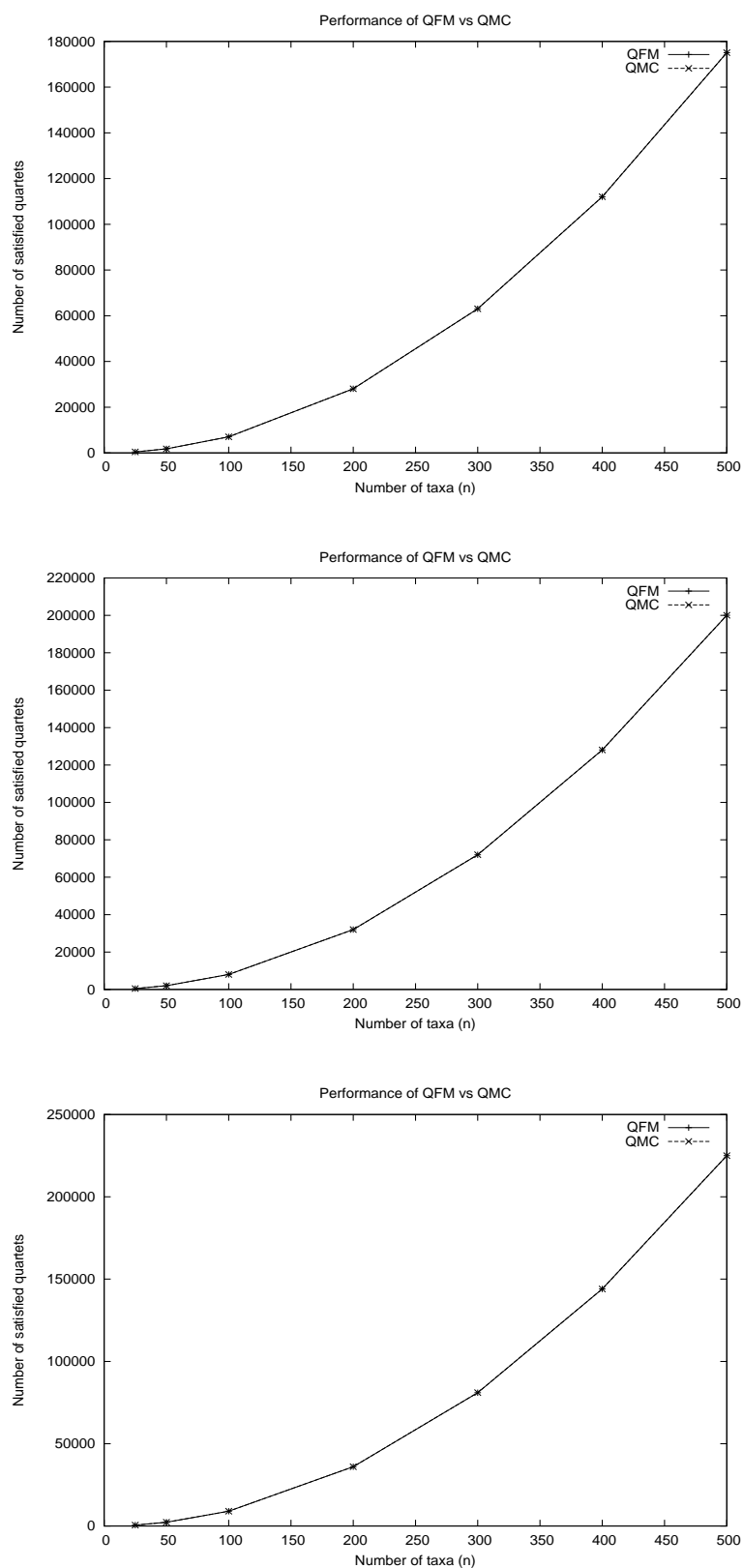
Performance Curves for QFM - IIIb with n^2 input quartets

Figure B.17: Comparison of QFM - IIIb and QMC when size of quartet set = n^2 at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Performance Curves for QFM - IIIb with $n^{2.8}$ input quartets

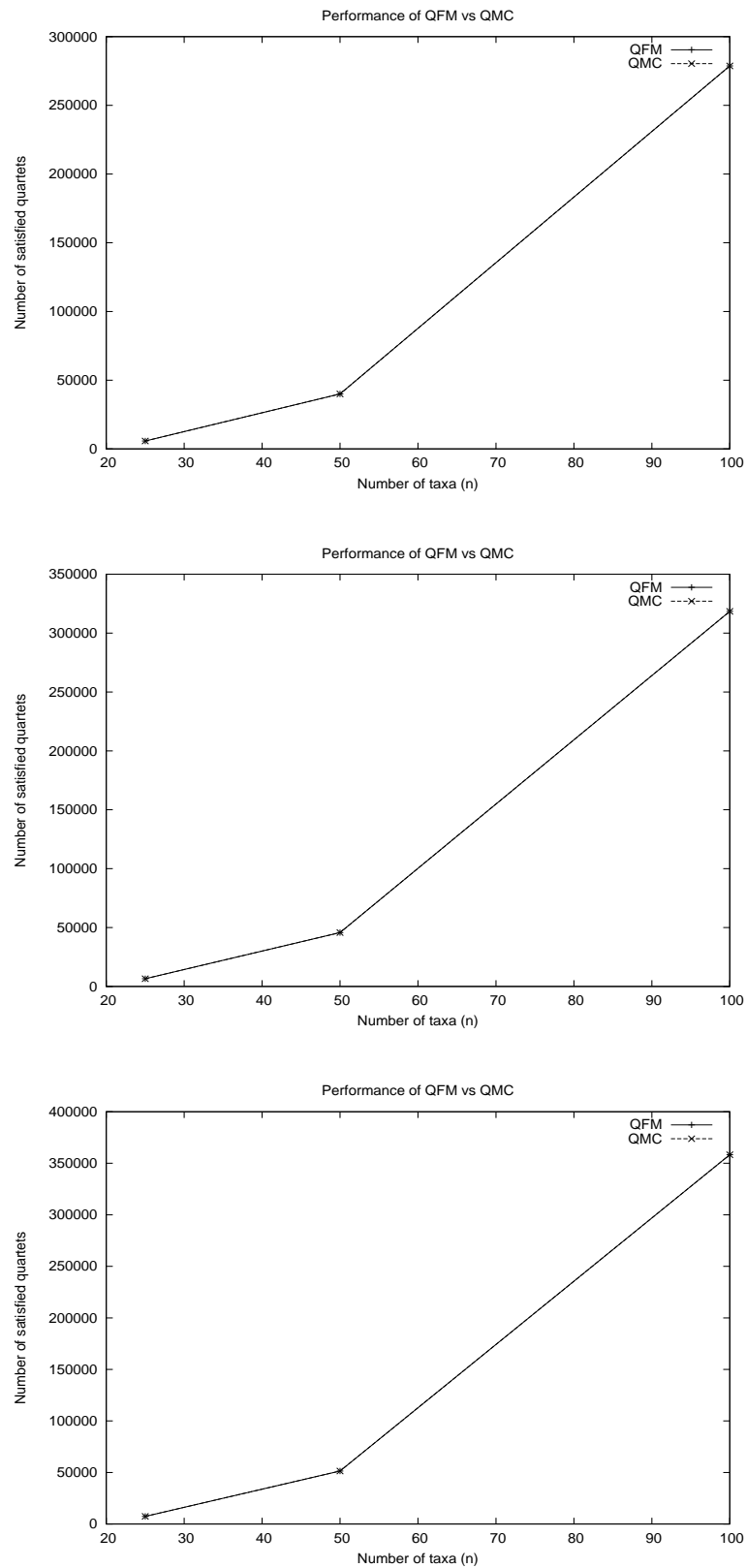


Figure B.18: Comparison of QFM - IIIb and QMC when size of quartet set = $n^{2.8}$ at $c = 70\%$ (top), $c = 80\%$ (middle) and $c = 90\%$ (bottom)

Appendix C

Experimental Running Time

#Taxa	#Quartets	%Consistency	Time(s)
25	125	70	3
25	125	80	3
25	125	90	3
25	625	70	5
25	625	80	4
25	625	90	4
25	8208	70	18
25	8208	80	19
25	8208	90	18
50	2500	70	12
50	2500	80	12
50	2500	90	11
50	354	70	7
50	354	80	6
50	354	90	5

#Taxa	#Quartets	%Consistency	Time(s)
50	57164	70	1161
50	57164	80	1138
50	57164	90	1178
100	10000	70	72
100	10000	80	55
100	10000	90	61
100	1000	70	13
100	1000	80	12
100	1000	90	13
100	398108	70	58137
100	398108	80	59957
100	398108	90	59123
200	2829	70	66
200	2829	80	64
200	2829	90	48
200	40000	70	969
200	40000	80	1153
200	40000	90	1036
300	5197	70	247
300	5197	80	252
300	5197	90	240
300	90000	70	5999
300	90000	80	6227
300	90000	90	6694

#Taxa	#Quartets	%Consistency	Time(s)
400	160000	70	17927
400	160000	80	22582
400	160000	90	23118
400	8000	70	654
400	8000	80	610
400	8000	90	569
500	11181	70	1574
500	11181	80	1393
500	11181	90	1387
500	250000	70	58003
500	250000	80	48870
500	250000	90	47952

Table C.1: Running time of QFM - IIIa for different datasets

Index

- algorithm
 - complexity, 27, 47
 - deterministic, 28
 - divide and conquer, 30, 34
 - heuristic algorithm, 30
 - linear algorithm, 28
 - nondeterministic, 28
 - polynomial algorithm, 28
- amino acid, 4
- asymptotic behavior, 27
- biogeographic hypothesis, 4
- biological data, 83
- bipartition, 23, 35
 - gain measure, 37
 - MFM, 36, 38
 - score, 37
 - singleton, 37
- clade, 21
- compatibility problem, 9
- evolution, 1, 3, 4, 6
 - tree of life, 10
- FN rate, FR rate, RF rate, 57
- maximum quartet consistency, 31
- newick representation, 19, 22
- phylogenetic methods
 - distance based, 7
 - ML, 7
 - MP, 6
 - NJ, 8
 - sequence based, 6
- phylogeny, 2
 - application, 3
 - problem, 2
 - reconstruction, 5
 - taxon, 2
 - tree, 1
 - tree estimation, 6
- protein, 4
- QFM, QMC, 57, 60, 62, 65, 69, 73, 83
- QTREE, 14, 88
- quartet, 22
 - consistency, 23
 - deferred, 25
 - quartet compatibility, 31
 - rooted, 22

- satisfied, 24

- unrooted, 22

- violated, 25

Quartet FM Algorithm, 33

running time, 27

star, 21

supertree, 8, 11

- construction, 9

- MRP, 9

- quartet based method, 10

 - QJ, 12

 - QMC, 13

 - QP, 12

 - SQP, 12

tree

- phylogenetic, 16, 32, 33

- rooted, 16, 18

- unrooted, 17

References

- [1] Daniel R Brooks and Deborah A McLennan. *Phylogeny, ecology, and behavior: a research program in comparative biology*. University of Chicago Press, 1991.
- [2] Robin M Bush, Catherine A Bender, Kanta Subbarao, Nancy J Cox, and Walter M Fitch. Predicting the evolution of human influenza a. *Science*, 286(5446):1921–1925, 1999.
- [3] SB Carroll, JK Grenier, and SD Weatherbee. *From dna to diversity: Blackwell science. Inc, Malden, Massachusetts*, 2001.
- [4] Benny Chor and Tamir Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(suppl 1):i97–i106, 2005.
- [5] Fiduccia CM and Mattheyses RM. A linear time heuristics for improving network partitions. *Proc. The 19th Design Automation Conference*, pages 175–181, 1982.
- [6] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Virkumar Vazirani. *Algorithms*. 2006.
- [7] William HE Day. Computationally difficult parsimony problems in phylogenetic systematics. *Journal of theoretical biology*, 103(3):429–438, 1983.
- [8] Joseph Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.
- [9] Joseph Felsenstein. Phylogenies and the comparative method. *American Naturalist*, pages 1–15, 1985.

- [10] Walter M Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, 1971.
- [11] Les R Foulds and Ronald L Graham. The steiner problem in phylogeny is np-complete. *Advances in Applied Mathematics*, 3(1):43–49, 1982.
- [12] Douglas J Futuyma. The uses of evolutionary biology. *Science-AAAS-Weekly Paper Edition*, 267(5194):41–42, 1995.
- [13] Michael R. Garey and David S. Johnson. Computers and intractability a guide to the theory of np-completeness. 1979.
- [14] Paul H Harvey and Mark Pagel. Comparative method in evolutionary biology (pod). 1991.
- [15] S Blair Hedges, Sudhir Kumar, Koichiro Tamura, Mark Stoneking, et al. Human origins and analysis of mitochondrial dna sequences. *Science*, 255(5045):737–739, 1992.
- [16] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.
- [17] June Yong Lee, Leo Joseph, and Scott V. Edwards. A species tree for the australo-papuan fairy-wrens and allies (aves: Maluridae). *Systematic Biology*, 2011.
- [18] David A Liberles, David R Schreiber, Sridhar Govindarajan, Stephen G Chamberlin, Steven A Benner, et al. The adaptive evolution database (taed). *Genome Biol*, 2(8):1–0028, 2001.
- [19] C Randal Linder and Tandy Warnow. An overview of phylogeny reconstruction. *Handbook of Computational Molecular Biology*, 2005.
- [20] Wayne P Maddison. A method for testing the correlated evolution of two binary characters: are gains or losses concentrated on certain branches of a phylogenetic tree? *Evolution*, pages 539–557, 1990.

- [21] EP Martins. Phylogenies and comparative data, a microevolutionary perspective. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 349(1327):85–91, 1995.
- [22] TJS Merritt and JM Quattro. Evidence for a period of directional selection following gene duplication in a neurally expressed locus of triosephosphate isomerase. *Genetics*, 159(2):689–697, 2001.
- [23] Michael L Metzker, David P Mindell, Xiao-Mei Liu, Roger G Ptak, Richard A Gibbs, and David M Hillis. Molecular evidence of hiv-1 transmission in a criminal case. *Proceedings of the National Academy of Sciences*, 99(22):14292–14297, 2002.
- [24] David W Mount. Sequence and genome analysis. *Bioinformatics: Cold Spring Harbour Laboratory Press: Cold Spring Harbour*, 2, 2004.
- [25] Vincent Ranwez and Olivier Gascuel. Quartet-based phylogenetic inference: improvement and limits. *Mol. Biol. Evol.*, 18(6):1103–1116, 2001.
- [26] Mark A Regan. Matrix representation in reconstructing phylogenetic relationships among the eukaryotes. *Biosystems*, 28(1):47–55, 1992.
- [27] DF Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981.
- [28] Naruya Saitou and Masatoshi Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.
- [29] M. J. Sanderson, M. J. Donoghue, W. Piel, and T. Eriksson. Treebase: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *Amer Jour Bot*, 81:183, 1994.
- [30] Heiko A. Schmidt, Korbinian Strimmer, Martin Vingron, and Arndt von Haeseler. Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18(3):502–504, 2002.

- [31] Sagi Snir and Satish Rao. Quartets maxcut: A divide and conquer quartets algorithm. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 7(4):704–718, 2010.
- [32] Sagi Snir and Satish Rao. Quartet maxcut: A fast algorithm for amalgamating quartet trees. *Journal of Molecular Phylogenetics and Evolution*, 62:1–8, 2012.
- [33] Sagi Snir, Tandy Warnow, and Satish Rao. Short quartet puzzling: A new quartet-based phylogeny reconstruction algorithm. *Journal of Computational Biology*, 15(1):91–103, 2008.
- [34] Michael Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of classification*, 9(1):91–116, 1992.
- [35] Korbinian Strimmer, Nick Goldman, and Arndt von Haeseler. Bayesian probabilities and quartet puzzling. *Mol. Biol. Evol.*, 14:210–211, 1997.
- [36] Korbinian Strimmer and Arndt von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13:964–969, 1996.
- [37] M Shel Swenson, Rahul Suri, C Randal Linder, and Tandy Warnow. An experimental study of quartets maxcut and other supertree methods. *Algorithms for Molecular Biology*, 6(1):7, 2011.
- [38] M Shel Swenson, Rahul Suri, C Randal Linder, Tandy Warnow, et al. An experimental study of quartets maxcut and other supertree methods. *Algorithms for Molecular Biology*, 6(1):7, 2011.
- [39] Tandy Warnow. Introduction to phylogenetic estimation algorithms. Presentation, University of Texas at Austin, 26 February, 2009. (<http://www.cs.utexas.edu/phylo/research/resources.html>), last accessed on May 17, 2013.

- [40] Lei Xin, Bin Ma, and Kaizhong Zhang. A new quartet approach for reconstructing phylogenetic trees: Quartet joining method. LNCS 4598, pages 40–50. Springer, 2007.
- [41] O. Zhaxybayeva, J.P. Gogarten, R.L Charlebois, W.F. Doolittle, and R.T. Papke. Phylogenetic analyses of cyanobacterial genomes: quantification of horizontal gene transfer events. *Genomre Research*, 16(9):1099–1108, 2006.