A Lookahead Branch-and-Bound Algorithm for the Maximum Quartet Consistency Problem

Gang Wu [*] Jia-Huai You [*] Guohui Lin [*] [†]

January 17, 2005

Abstract

A lookahead branch-and-bound algorithm is proposed for solving the Maximum Quartet Consistency Problem where the input is a complete set of quartets on the taxa and the goal is to construct a phylogeny which satisfies the maximum number of given quartets. Such a phylogeny constructed from quartets has many advantages over phylogenies constructed through other ways, one of which is that it is able to overcome the data disparity problem. Nonetheless, the MQC problem has been proven to be NP-hard. The branch-and-bound algorithm integrates a number of previous efforts on exact algorithms, heuristics, and approximation algorithms for the MQC problem, and a few improved search techniques, especially a lookahead scheme, to solve the problem optimally. The theoretical running time analysis of the algorithm is provided, and an extensive simulation study has been well designed to compare the algorithm to previous existing exact algorithms and a best heuristics Hypercleaning. The experimental results on both synthetic and real datasets show that the proposed algorithm outperformed other exact algorithms, and it was competitive to Hypercleaning on many datasets.

1 Introduction

With the availability of more and more genomic data, there are more needs in fast phylogenetic analysis to facilitate biological applications. As a concrete example, the Influenza Sequence Database (`http://www.flu.lanl.gov/`) has a deposit of about 5318 whole genomes for Avian Influenza viruses as of January 16, 2005. The phylogenetic analysis on these viruses at both the chronological level and the territorial level to carry out the evolutionary relationships is crucial to fast understanding of the emergence of new variants and fast vaccination strategy design. There are a number of models as well as algorithms that have been proposed for the phylogenetic analysis, each usually assumes the same amount of biological data associated with every taxon under investigation. In practice, however, the data disparity problem exists which means the availability of biological data for analysis is different for every taxon. This data disparity problem either makes the phylogenetic analysis to use a much less amount of (but common) data for all taxa or limits the phylogenetic analysis to consider a less amount of taxa. Subsequently, it raises the question of how confident we should trust the analytical results that might be obtained using different data for different subsets of taxa. One way of resolving this issue is to use different phylogenetic analysis methods to handle different subsets of taxa where the methods find most applicable, and then to assemble a global phylogenetic pattern out of the achieved sub-patterns for the subsets. The promise of this approach is that, since every piece of phylogenetic analysis on subsets is of high confidence, the global analytical results must also be of high confidence, though there might be needs to resolve potential conflicts among

[*]Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada. Email: wgang,you,ghlin@cs.ualberta.ca.

[†]To whom correspondence should be addressed. Tel: (780) 492-3737, Fax: (780) 492-1071.

the sub-patterns for subsets. The quartet-based phylogeny construction methods can be classified into such efforts to construct a (global) phylogeny for a set of taxa associated with different biological data.

1.1 Quartet-Based Phylogeny Construction

In our discussion of phylogeny construction for a set of taxa, the phylogeny is an unrooted binary tree whose leaves bijectively map to the set of taxa and every internal node in the tree has degree 3. In quartet-based phylogeny construction methods, researchers try to build a phylogeny for every (or most) subset of 4 taxa — called a *quartet topology* or simply a *quartet*, and then assemble a global phylogeny for the whole set of taxa to satisfy all the quartets that have been built, or if not at all possible, to satisfy as many of them as possible. In the weighted version, which is not discussed in this paper, every quartet would be assigned a weight and the global phylogeny is expected to satisfy a maximum weight set of quartets. For a set of 4 taxa, there are 3 possible phylogenies or quartets associated with it. For example, Figure 1 shows the 3 quartets for a 4-taxa set $\{s_1, s_2, s_3, s_4\}$. For simplicity, we use $[s_1, s_2|s_3, s_4]$ to denote the quartet in which the path connecting s_1 and s_2 doesn't intersect the path connecting s_3 and s_4, as shown in Figure 1(a).

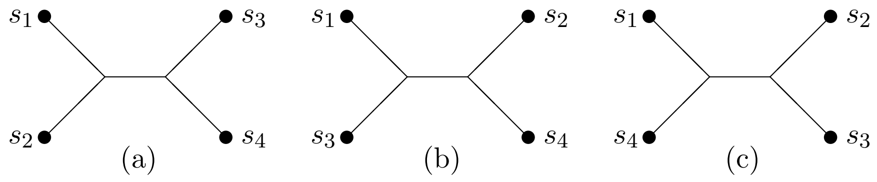

Figure 1: Three possible quartets for 4-taxa set $\{s_1, s_2, s_3, s_4\}$: (a) $[s_1, s_2|s_3, s_4]$, (b) $[s_1, s_3|s_2, s_4]$, and (c) $[s_1, s_4|s_2, s_3]$.

Given a phylogeny T for a set of taxa S, for every subset X of 4 taxa, we can derive a quartet for X by computing the induced subtree of T on X. Such a set of $\binom{n}{4}$ induced quartets is denoted as Q_T. Conversely, given a set Q of quartets (which can be built by various quartet inference approaches), for every subset of 4 taxa, Q contains at most one quartet for it (i.e. no ambiguity). If there exists one global phylogeny T for S such that a quartet $q \in Q$ for a subset of 4 taxa is the same as the one derived from T, then T *satisfies* q or q is *consistent* with T. If there exists one global phylogeny T satisfying all quartets in Q, i.e. $Q = Q_T$, then Q is *compatible* and T is the phylogeny *associated* with Q.

The recognition problem, called the *Quartet Compatibility Problem* (QCP), is to determine whether a given set Q of quartets on a set of taxa S is compatible or not, or equivalently if there is a phylogeny T on S satisfying all the quartets in Q. If Q contains exactly one quartet for every subset of 4 taxa, then Q is *complete*; Otherwise, Q is *incomplete*. It has been known that when Q is complete, the QCP problem can be answered in $O(n^4)$ time, where n is the size of taxa set S; furthermore, if Q is compatible, then the associated phylogeny T is unique and can be constructed within the same time [7]. The situation changes when Q is incomplete, where the recognition problem becomes NP-complete [13].

The more interesting computational problem is the optimization problem where Q (either complete or incomplete, but in this paper we consider the complete case) is not compatible and the goal is to construct a phylogeny to satisfy as many quartets as possible. This is the so-called *Maximum Quartet Consistency Problem* (MQC). A dual minimization problem to the MQC problem, the *Minimum Quartet Inconsistency Problem* (MQI), where the input is the same, is to construct a phylogeny to minimize the number of inconsistent quartets. Despite the fact that the MQC and the MQI problems have the same optimal solution(s), their approximabilities differ a lot. The MQC problem is NP-hard [3] and it admits a *Polynomial Time Approximation Scheme* (PTAS) [11]; The MQI problem is NP-hard [3] too but the best approximation ratio so far is $O(n^2)$ where n is the size of taxa set [10].

A few attempts have been made to solve the MQC/MQI problem optimally. Ben-Dor et al. [2] presents a dynamic programming algorithm that evaluates the number of quartets that are consistent with a bipartition of the taxa set and thereby determines a phylogeny to satisfy the maximum number of quartets. The running time of the algorithm is $O(3^n n^4)$. The inconsistent quartets with a bipartition are referred to as *quartet errors* (across the bipartition), which are subject to be determined and changed in order to be compatible to other quartets. If the number of quartet errors is known ahead of time, then the fixed-parameter algorithm proposed by Gramm and Niedermeier [8], which for simplicity is referred to as the GN algorithm subsequently, would be able to detect and correct them, and to return an associated phylogeny. The GN algorithm has a running time $O(4^k n + n^4)$, where k is the number of quartet errors. We note that the algorithm could be modified to solve the general MQC/MQI problem where the number of quartet errors is unknown. In fact, the branch-and-bound algorithm we developed in this paper takes advantage of many technical steps in the GN algorithm, and many technical steps in another exact algorithm for solving the MQC/MQI problem through Constraint Programming, which we developed in a previous work [14].

To get good bounds for speedup in our branch-and-bound algorithm, we review in the following a few heuristics and approximation algorithms that have been designed for the MQC problem and/or the MQI problem. We choose to review only those that are directly useful in our algorithm, while the interested readers may refer to [8] and its reference list for more information. One approach to make Q compatible by changing some quartet topologies is *quartet cleaning* which tries to detect and correct obvious quartet errors according to some rules [4, 3, 6, 9, 11]. There are edge and vertex quartet cleaning, as well as global and local cleaning algorithms. An edge in a phylogeny one-to-one corresponds to a bipartition of the taxa set. Given a bipartition (X, Y) of the taxa set with $|X| \geq 2$ and $|Y| \geq 2$, define $Q_{(X,Y)}$ to be the set of quartets in the form of $[x, x'|y, y']$ where $x, x' \in X$ and $y, y' \in Y$. If the quartet $q \in Q$ for the 4 taxa subset $\{x, x', y, y'\}$ is not $[x, x'|y, y']$, then q is a quartet error *across* the bipartition (X, Y) (or across the edge) [3]. Global quartet cleaning algorithms correct quartet errors only if the number of quartet errors is bounded for every edge (or vertex). Local cleaning algorithms correct quartet errors when the number of quartet errors is bounded (even only) for one edge (or vertex). Della Vedova and Wareham [6] provide an overview of quartet cleaning results. The global edge quartet cleaning algorithm given by Berry et al. [3] computes the optimal phylogeny if, for every edge e inducing a bipartition (A_e, B_e), there are fewer than $(|A_e| - 1)(|B_e| - 1)/2$ quartet errors across edge e. Note that this value is minimal for $|A_e| = 2$ and $|B_e| = n - 2$ and, therefore, the cleaning algorithm computes a guaranteed optimal solution if the total number of quartet errors is smaller than $(n - 3)/2$ [3, 8]. The heuristics *hypercleaning* developed by Bryant et al. [4] is an extension of the quartet cleaning methods. Given a bipartition (X, Y) of the taxa set S, the normalized distance from Q to (X, Y) is defined as $\delta(Q, (X, Y)) = \frac{4|Q_{(X,Y)} - Q|}{|X|(|X|-1)|Y|(|Y|-1)}$, which basically measures the extent of disagreement of Q if edge (X, Y) is included in the output phylogeny. The hypercleaning algorithm computes the following sets of bipartitions in order

$$Best(Q, m) = \left\{(X, Y) \mid \delta(Q, (X, Y)) \leq \frac{2m}{|X||Y|}\right\}, \ m = 0, 1, 2, \ldots$$

It has been shown that bipartitions, corresponding to edges, in $Best(Q, 0)$ and $Best(Q, 1)$ are always compatible to each other, while bipartitions from $Best(Q, m)$ with greater values of m might not be compatible with bipartitions in $Best(Q, 0)$ and $Best(Q, 1)$. The hypercleaning algorithm uses bipartitions in $Best(Q, 0)$ and $Best(Q, 1)$ to form a base set of edges in the final phylogeny and tries to add bipartitions in $Best(Q, 2)$ to it greedily to grow the phylogeny, and if not done then tries to add bipartitions in $Best(Q, 3)$ to it greedily to grow the phylogeny, and so on, till a phylogeny is formed. The overall running time of the hypercleaning algorithm is $O(n^5 f(2m) + n^7 f(m))$ where $f(m) = 4m^2(1 + 2m)^{4m}$ and m denotes the greatest value that the set $Best(Q, m)$ has to be computed in the execution.

1.2 The Organization

In the next section, we present a number of facts associated with the quartets to provide the theoretical foundations for the design of our branch-and-bound algorithm. Some of these facts might be attached to existing algorithms such as the GN algorithm and the hypercleaning algorithm, others might be independent on any algorithm. We present the lookahead branch-and-bound algorithm in Section 3 where the meaning of "lookahead" would become clear. We have implemented our algorithm, as well as some other algorithms mentioned above, and tested them on various synthetic and real datasets. Section 4 summarizes the experimental results and the comparisons we have made. We conclude the paper in Section 5 with some remarks.

2 Theoretical Foundations for the Branch-and-Bound Algorithm

Our branch-and-bound algorithm takes advantage of many technical steps of the GN algorithm [8], which is designed for the MQI problem where the number of quartet errors is known to be exactly k. In the following we include some propositions from [8] that are found useful in the design of our algorithm. The interested readers should refer to [8] for more details. The most important idea in the GN algorithm is to resolve global quartet conflicts through resolving local quartet conflicts [5, 1], referred to as *local conflicts*. A local conflict is a set of 3 incompatible quartets on a subset of exactly 5 taxa. For example, $\{[a, b|c, d], [a, c|b, e], [a, c|d, e]\}$ is a local conflict.

Theorem 2.1 [8] *Given a complete set of quartets Q over a set of taxa S and some taxon $e \in S$, Q is compatible iff there exists no local conflict whose taxa set includes e.*

Notice that when a taxon e is fixed, there are $\binom{n-1}{4}$ subsets of 5 taxa containing e and for each such subset there are $\binom{5}{3} = 10$ potential local conflicts. Therefore, Theorem 2.1 tells us that testing if Q is compatible can be reduced to testing if none of the $O(n^4)$ subsets of 3 quartets is a local conflict. Since testing each set of 3 quartets takes constant time, this implies an $O(n^4)$ algorithm for the QCP problem. When a quartet q is fixed, a similar argument gives that there are at most $6(n-4)$ local conflicts containing q. Subsequently, if we decide to change the quartet topology for a subset of 4 taxa, then at most $6(n-4)$ new local conflicts would be generated and at most $6(n-4)$ previous local conflicts would be resolved. The GN algorithm begins with building a conflict list that contains all the local conflicts (involving a fixed taxon). At every step, it randomly chooses one local conflict and tries to resolve it through changing some quartet in the local conflict. It is proven in [8] that it is sufficient to consider at most four ways of resolving a local conflict and the GN algorithm picks one in some order. The new quartet topology becomes fixed for that specific subset of 4 taxa in the rest of the algorithm execution unless there is no solution along the way and the changing would be undone. The GN algorithm subsequently updates the conflict list (again, containing all the local conflicts involving the fixed taxon). When there is no further local conflict left, the GN algorithm terminates and assembles a phylogeny in another $O(n^4)$ time. While if there have been k quartet changes and there still exist local conflicts, the GN algorithm marks this route of quartet changing unpromising and undoes the last quartet changing in the sequence and tries another way of resolving the local conflict.

It is proven that the main effort at every node in the search tree associated with the GN algorithm is the conflict list updating, which takes $O(n)$ time. Therefore, the overall running time of the algorithm is $O(4^k n + n^4)$. In [8], some efforts have been made to improve the running time in practice through determining the quartets that must be changed, as stated in the following Theorem 2.2; some other efforts have been made to improve the running time in practice but potentially dropping the optimality, through fixing some quartets (and bipartitions) along the computation.

Theorem 2.2 [8] *For a quartet $q \in Q$, if there are more than $3k$ distinct local conflicts that contain q, then q must be changed in the optimal solution.*

We remark that the GN algorithm is designed for a special case of the MQI problem where the number of quartet errors is known to be exactly k. It therefore terminates when it finds the first solution. The algorithm can be modified to return all solutions in $O(4^k n + n^4)$ time. Such a modification would also be able to solve the MQC/MQI problem when the number of quartet errors is only known to be not exceeding some constant. Our branch-and-bound algorithm is designed to solve the general MQC/MQI problem. In order to get a first bound on the number of quartet errors, we use some other phylogeny construction methods to produce a phylogeny. The rest of this section is devoted to some propositions which can fast determine (more) quartets that need to be fixed and (more) quartets that need to be changed. Some of these speedups are novel and make the branch-and-bound algorithm significantly outperform the existing exact algorithms.

Let (X, Y) be a bipartition of the taxa set S and $|X| = \ell$ and $|Y| = n - \ell$. It follows that $|Q_{(X,Y)}| = \binom{\ell}{2}\binom{n-\ell}{2}$. Let $p_1 = |Q_{(X,Y)} - Q|$. Fixing three taxa from Y, a subset of ℓ quartets from Q where each quartet contains these three taxa and one taxon from X is called an ℓ-subset with respect to (X, Y). There are in total $\binom{n-\ell}{3}$ such ℓ-subsets. For an ℓ-subset, if ignoring the difference of the taxa from X gives rise to one unique quartet topology, then the ℓ-subset is *exchangeable* on X; Otherwise, it is *nonexchangeable* on X. Let p_2 and p_3 denote the number of nonexchangeable ℓ-subsets on X and the number of nonexchangeable $(n - \ell)$-subsets on Y, respectively.

Lemma 2.3 *Given $n(> 1)$ positive integers $a_1, a_2, \ldots, a_n$ whose sum is m, then $\sum_{1 \le i < j \le n} a_i a_j \ge \frac{(n-1)(2m-n)}{2}$ and the equality holds when $a_1 = a_2 = \ldots = a_{n-1} = 1$ and $a_n = m - (n - 1)$.*

Proof.

We use induction.

Initial Step:

When $n = 2$, we have $a_1 + a_2 = m$.

$a_1 a_2 = a_1(m - a_1) = -a_1^2 + m a_1$, $1 \le a_1 \le m - 1$

$\Rightarrow a_1 a_2 \ge m - 1$, when $a_1 = 1$ or $a_1 = m - 1$.

Inductive Step:

Our inductive assumption is:

Assume when $n = k$, $\sum_{1 \le i < j \le k} a_i a_j \ge \frac{(k-1)(2m-k)}{2}$, and and the equality holds when $a_1 = a_2 = \ldots = a_{k-1} = 1$ and $a_k = m - (k - 1)$.

We must prove the formula is true for $n = k + 1$.

$\sum_{1 \le i < j \le k+1} a_i a_j = a_1a_2 + a_1a_3 + \ldots + a_1a_k + a_1a_{k+1} + a_2a_3 + \ldots + a_2a_k + a_2a_{k+1} + \ldots + a_{k-1}a_k + a_ka_{k+1}$

$= (a_1a_2 + a_1a_3 + \ldots + a_1a_k + a_2a_3 + \ldots + a_2a_k + \ldots + a_{k-1}a_k) + (a_1 + a_2 + \ldots + a_k)a_{k+1}$

$\ge \frac{(k-1)[2(m-a_{k+1})-k]}{2} + (m - a_{k+1})a_{k+1}$,

(when $a_1 = a_2 = \ldots = a_{k-1} = 1$ and $a_k = m - a_{k+1} - (k - 1)$)

$= -a_{k+1}^2 + (m - k + 1)a_{k+1} + m(k - 1) - \frac{k(k-1)}{2}$, $(1 \le a_{k+1} \le (m - k))$

$\ge (m - 1)k - \frac{k(k-1)}{2}$, (when $a_{k+1} = 1$ or $a_{k+1} = m - k$)

$= \frac{k[2m-(k+1)]}{2}$

□

Theorem 2.4 *Let Q be a complete set of quartets on an n-taxa set S. For a bipartition (X, Y) of S where $|X| = \ell$, let p_1 be the number of quartet errors in Q across (X, Y), p_2 be the number of nonexchangeable ℓ-subsets on X, and p_3 be the number of nonexchangeable $(n-\ell)$-subsets on Y. If $2p_1 + (\ell-1)p_2 + (n-\ell-1)p_3 \le (\ell - 1)(n - \ell - 1)$, then bipartition (X, Y) must be in the optimal phylogeny.*

Proof.

We consider the unrooted phylogeny.

We divide the quartet topologies set Q into five parts (see Table 1). We group the quartet topologies in the second part into a list of ℓ-subsets and fourth part into a list of $(n-\ell)$-subsets. Each k-subset is either exchangeable or unexchangeable.

Part	Quartet Topologies
1	quartet topologies involve none of species in $X = \{s_{i_1}, s_{i_2}, \ldots, s_{i_k}\}$ $\{s_a, s_b, s_c, s_d\}$
2	quartet topologies involve exactly one species in X, and three species in Y $\{s_{i_1}, s_a, s_b, s_c\}$, $\{s_{i_2}, s_a, s_b, s_c\}$, ...
3	quartet topologies involve exactly two species in X, and two species in Y $\{s_{i_1}, s_{i_2}, s_a, s_b\}$, $\{s_{i_1}, s_{i_3}, s_a, s_b\}$, ...
4	quartet topologies involve three species in X, and one species in Y $\{s_{i_1}, s_{i_2}, s_{i_3}, s_a\}$, ..., $\{s_{i_1}, s_{i_2}, s_{i_3}, s_d\}$, ...
5	quartet topologies involve four species in X $\{s_{i_1}, s_{i_2}, s_{i_3}, s_{i_4}\}$, ..., $\{s_{i_1}, s_{i_2}, s_{i_3}, s_{i_5}\}$, ...

Table 1: Divide the quartet topologies set Q into five parts

Now we will prove that for any phylogeny T_1 in which we cannot find an edge (X, Y), we can construct a new phylogeny T_2 in which there exists an edge (X, Y) and T_2 will satisfy more quartet topologies than T_1.

In T_1, if we trim all the leaf nodes in Y, we can get a sub-phylogeny T_{1X} on X (Figure 2 *b*). Similarly, we can get a sub-phylogeny T_{1Y} on Y (Figure 2 *c*). We can then combine this two sub-phylogenies into a new phylogeny T_2 by a connecting edge(Figure 2 *d*). In other words, we move all the labels in X to a leaf edge of T_1 whose label is in X and keep the phylogeny structure of those leaf edges, and do the same process for the labels in Y.

Figure 2 shows an example of construction of T_2, where $X = \{s_1, s_2, s_3\}$ and $Y = \{s_4, s_5, s_6, s_7, s_8\}$. When constructing T_2, we put T_{1X} at the end of leaf edge s_1, and put T_{1Y} at the end of leaf edge s_6.

For each exchangeable ℓ-subset in the second part, if T_1 satisfies all the quartet topologies, then T_2 also satisfies all of them, no matter where we put T_{1Y}. If T_1 satisfies none of them, T_2 also satisfies none of them. If T_1 satisfies ℓ' $(0 < \ell' < k)$ number quartet topologies, then T_2 satisfies all of them(ℓ) or none of them (0), based on where we put the T_{1Y}. From this observation, we can find a position to put T_{1Y} so that T_2 can satisfy more or equal number of quartet topologies in exchangeable ℓ-subsets in the second part. Similarly, we can find a position to put T_{1X} so that T_2 can satisfy more or equal number of quartet topologies in exchangeable $(n-\ell)$-subsets in the fourth part.

From the above construction of T_2, we can see that T_1 and T_2 satisfy same number of quartet topologies in the first and fifth parts, and T_2 satisfies more or equal number of quartet topologies in the exchangeable ℓ-subsets and $(n-\ell)$-subsets in the second and fourth parts, respectively.

Let the quartet topologies set inferred by T_1 be Q_{T_1}, and the quartet topologies set inferred by T_2 be Q_{T_2}.

First, we consider T_1. Based on the sub-phylogeny T_{1X}, which has ℓ labels and $2\ell - 3$ edges, we recover T_1. Let the numbers of leaf nodes in the branches bifurcated from these edges be $m_1, m_2, \ldots, m_{\ell'}$ (see Figure 3 a). These values have the following properties:

1). $m_i > 0$ for $1 \leq i \leq \ell'$.

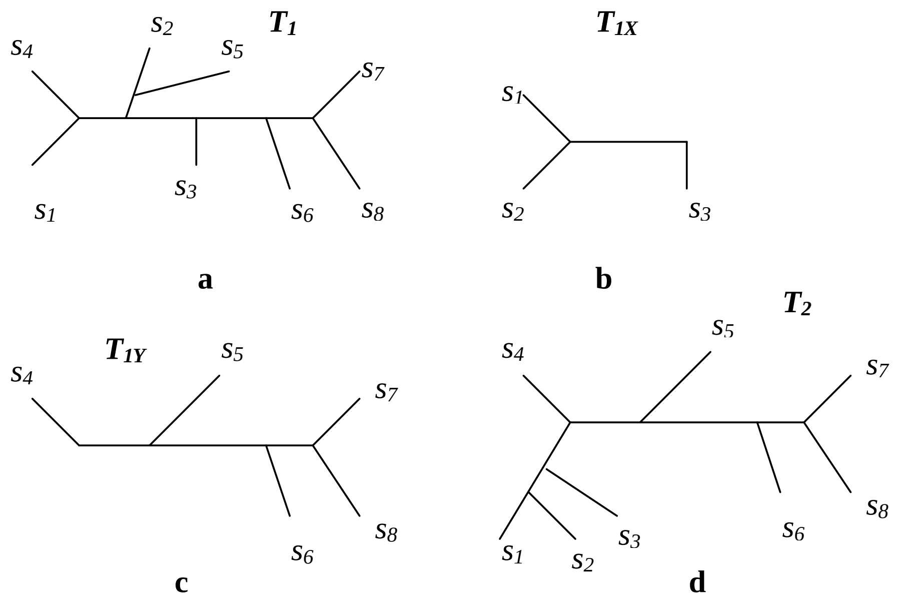

Figure 2: Construction of a new phylogeny T_2 from T_1

2). $\ell' > 1$. If there is only one such branch, then we can find an edge (X, Y), which contradicts our assumption.

3). $m_1 + m_2 + \ldots + m_{\ell'} = n - \ell$.

For any two m_i and m_j, let n_1, n_2, n_3 be the number of leaf nodes in X bifurcated from the internal edges connecting m_i and m_j (see Figure 3 b). In this case, the number of quartet topologies in Q_{T_1} that conflict on edge (X, Y) is

$m_i m_j (n_1 n_2 + n_1 n_3 + n_2 n_3) \geq m_i m_j (\ell - 1)$, since $n_1 + n_2 + n_3 = \ell$ and at most one of n_1, n_2, n_3 can be 0.

Therefore, the total number of quartet topologies in Q_{T_1} that conflict on edge (X, Y) is greater or equal to

$(\ell - 1)(m_1 m_2 + m_1 m_3 + \ldots + m_1 m_{\ell'} + \ldots + m_{\ell'-1} m_{\ell'})$
$\geq (\ell - 1) \cdot \frac{(\ell'-1)[2(n-\ell)-\ell']}{2}$
$\geq (\ell - 1)(n - \ell - 1)$.

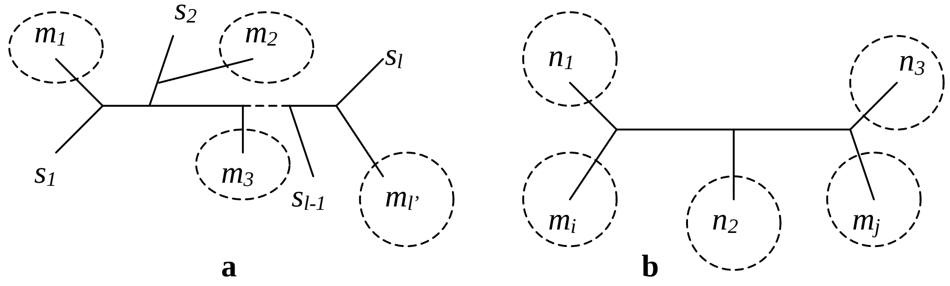

Figure 3:

To achieve T_1, we need change at least $(\ell - 1)(n - \ell - 1) - p_1$ number of quartet topologies in the third part, while we need only change p_1 number of quartet topologies in the third part to achieve T_2.

We know that T_2 satisfies more or equal number of quartet topologies in the exchangeable ℓ-subsets in second part. Since we put T_{1Y} at one end of the T_1, in each ℓ-subset, there exist at least one quartet topology that has same consistency on both T_1 and T_2.

Suppose in the worst case, T_2 doesn't satisfy all the other $\ell-1$ quartet topologies in an unexchangeable ℓ-subset, and T_1 satisfies all of them. That will cause T_1 to satisfy $(\ell-1)p_2$ more quartet topologies than T_2 in the unexchangeable ℓ-subsets in second part.

Similarly, T_1 satisfies at most $(n-\ell-1)p_3$ more quartet topologies than T_2 in the unexchangeable $(n-\ell)$-subsets in fourth part.

We give a summary of the results here:

1). T_1 and T_2 satisfy same number of quartet topologies in the first and fifth parts;

2). In the second part, T_2 satisfies greater or equal number of quartet topologies in the exchangeable ℓ-subset and T_1 satisfies at most $(\ell-1)p_2$ more quartet topologies than T_2 in the unexchangeable ℓ-subsets, which means we need change at most $(\ell-1)p_2$ more quartet topologies to achieve T_2.

3). In the fourth part, T_2 satisfies greater or equal number of quartet topologies in the exchangeable $(n-\ell)$-subsets and T_1 satisfies at most $(n-\ell-1)p_3$ more quartet topologies than T_2 in the unexchangeable $(n-\ell)$-subsets, which means we need change at most $(n-\ell-1)p_3$ more quartet topologies to achieve T_2.

4). In the third part, We need change at least $(\ell-1)(n-\ell-1)-p_1$ number of quartet topologies to achieve T_1, while only p_1 number of quartet topologies should be changed to achieve T_2;

Since $2p_1+(\ell-1)p_2+(n-\ell-1)p_3 \leq (\ell-1)(n-\ell-1)$, in this case T_2 still satisfies more or equal number of quartet topologies than T_1.

$\square$

Note that if $|X|=2$, we call the two taxa in X siblings. In this case, $p_3=0$ and we have the following corollary.

Corollary 2.5 *Let Q be a complete set of quartets on an n-taxa set S. For a pair of taxa a and b, let p_1 be the number of quartet errors in Q across bipartition $(\{a,b\}, S-\{a,b\})$ and p_2 be the number of nonexchangeable pairs on $\{a,b\}$. If $2p_1+p_2 \leq n-3$, then a and b must be siblings in the optimal phylogeny.*

In [14], we have developed a scheme that transforms the computing of a phylogeny to satisfy the maximum number of quartets to the computing of an ultrametric matrix which essentially is about the computing of the least common ancestor for every pair of taxa such that a maximum number of quartets are satisfied. The computing of a desired ultrametric matrix is then formulated into a constraint programming that can be readily solved by calling to Smodels (`http://www.tcs.hut.fi/Software/smodels/`). Though the work is mainly regarded as a reformulation of an old problem, we have imposed in the Smodels a number of deduction rules which speed up the computation. These deduction rules also find useful in determining the quartets that must be fixed according to a set of already fixed quartets. Subsequently, at every node in the branch-and-bound search tree we might be able to shorten the conflict list and identify the illegal ways of resolving a local conflict. The following Theorem 2.6 lists some of the rules that have been implemented into our algorithm.

Theorem 2.6 [14] *For a set of* 5 *taxa* $\{a,b,c,d,e\}$,

- *if* $[a,b|c,d]$ *and* $[a,b|c,e]$ *are fixed, then* $[a,b|d,e]$ *must be fixed too;*
- *if* $[a,b|c,d]$ *and* $[a,c|d,e]$ *are fixed, then* $[a,b|c,e]$, $[a,b|d,e]$, *and* $[b,c|d,e]$ *must be fixed too.*

3 A Lookahead Branch-and-Bound Algorithm

Our branch-and-bound algorithm is designed to solve the MQC/MQI problem optimally regardless whether or not the number of quartet errors is known. Typically, in the unknown case, we can use some other phylogeny construction method(s) to infer one phylogeny and determine subsequently its associated quartet set. This provides us a first bound on the number of quartet errors. Note that we consider the general case where the number of quartet errors is only upper bounded rather than known exactly. Therefore, after the first feasible solution is arrived the algorithm does not stop but continues on search till the whole tree has been exhausted, to output the optimal solution.

Our algorithm might be regarded as an improvement over the GN algorithm, with several newly designed speedup techniques. In fact, most of the basic operations in the GN algorithm have been adopted into our algorithm, though they might be put together in different places and in different orders. Some significant differences between the two algorithms are detailed in the following: At the root node of the search tree (produced by our algorithm), a global search is performed to identify quartets need to be fixed using Theorem 2.4 (in $O(n^4)$ time). It then applies Theorem 2.6 to deduce more quartets need to be fixed (in $O(n^4)$ time). Another global search is performed to identify need-to-be-changed quartets according to Theorem 2.2 (in $O(nk)$ time). Similarly as in the GN algorithm, a conflict list at the root node is built (in $O(n^4)$ time). Besides this list, two other lists, one contains the fixed quartets and the other contains the need-to-be-changed quartets, are also maintained. In fact, every node in the search tree is associated with these three lists. At every node, the conflict list is further partitioned into two parts, one consists of local conflicts each contains a need-to-be-changed quartet and the other consists of local conflicts each contains no need-to-be-changed quartet. The algorithm puts priorities on need-to-be-changed quartets. For every need-to-be-changed quartet, 1) change the quartet topology to resolve some (at least one) local conflict; 2) update the fixed quartet list; 3) update the the conflict list; 4) update the need-to-be-changed quartet list. The difference between the size of the new conflict list and the size of the conflict list before quartet changing is defined to be the *contribution* of this need-to-be-changed quartet. The algorithm picks the need-to-be-changed quartet achieving the largest contribution to proceed the search. In the case that there is no need-to-be-changed quartet, similar treatment is done for every way of resolving a local conflict, and then the algorithm picks the way achieving the largest contribution to proceed. The algorithm updates the bound k if a solution is found and in the solution a less number of quartet errors were found.

Notice that our algorithm has a mechanism to look one step forward to identify the best way to resolve a local conflict. For this reason, we call it the lookahead branch-and-bound algorithm. It should be noted that although it appears that at every node our algorithm spends more time than the GN algorithm ($O(n^2k)$ vs. $O(n)$), with a modest value of k our algorithm runs much faster in practice. This is demonstrated true by the experimental results in Section 4. One high-level description of this lookahead branch-and-bound algorithm at every internal node is depicted in Figure 4.

Theorem 3.1 *The lookahead branch-and-bound algorithm described in Figure 4 solves the MQC/MQI problem and runs in $O(4^k n^2 k + n^4)$ time, where n is the size of the taxa set and k is an upper bound on the number of quartet errors.*

4 Experimental Results

4.1 Overview and Synthetic Data Generation

We have designed four experiments to compare the performance of our algorithm to existing algorithms, among which three are on synthetic datasets and the last one is on a real dataset of 30 taxa with about 8.3% quartet errors (i.e. $k = 2272$). Note that the algorithms tested are all quartet-based phylogeny construction algorithms. Therefore, the comparisons are for the purpose of showing the speed of our algorithm (except the comparison made to the hypercleaning algorithm). In all experiments, we say an algorithm could not solve an instance if the algorithm does not terminate in 100 hours. We chose to implement all the involved

```
1.       At every node in the search tree,
1.1.         Use Theorem 2.4 to determine fixed quartets;
1.2.         Use Theorem 2.6 to deduce as many fixed quartets as possible;
1.3.         Use Theorem 2.2 to determine need-to-be-changed quartets;
1.4.         Build a conflict list and partition it into two parts;
1.5.         If there are need-to-be-changed quartets,
1.5.1.           For need-to-be-changed quartet, calculate its contribution;
1.5.2.           Pick the need-to-be-changed quartet achieving the largest contribution;
1.6.         Else,
1.6.1.           For every way of resolving a local conflict, calculate its contribution;
1.6.2.           Pick the resolvement way achieving the largest contribution;
1.7.         If $k$ quartets have been changed but the conflict list is nonempty,
1.7.1.           Kill the node and return to the parent node;
1.8.         If the conflict list is empty,
1.8.1.           Update bound $k$, update the best solution, and return to the parent node;
1.9.         Continue on search at the picked node;
```

Figure 4: A high-level description of the lookahead branch-and-bound algorithm for the MQC/MQI problem.

algorithms in C/C++ (either by ourselves or courtesy to the original authors). All experiments were done on an IBM P690 computer with a 1.7 GHz processor and 32 GB main memory (shared by 16 CPUs).

Similar to [8], to provide a common test bench for all the algorithms we generated artificial datasets. Again we want to emphasis that such datasets are for the purpose of testing the computing power of the algorithms. Therefore, some of our datasets are very hard to all algorithms. For every pair of (n, k), where n refers to the number of taxa and k refers to the upper bound of the quartet errors, we generated a random phylogeny by recursively inserting one taxon onto an arbitrary edge in the existing sub-phylogeny. After all taxa have been added to the phylogeny, which is an unrooted binary tree, we derived the set of quartets induced by the phylogeny. We then arbitrarily picked k out of the $\binom{n}{4}$ quartets and altered the topologies. We remark that this process only guarantees the number of quartet errors is upper bounded by k, but not necessarily equal to k since some combinations of quartet altering might give rise to a new compatible set of quartets. For every pair of (n, k) we generated 10 datasets, and the following reported results are the average for test run on them. Different algorithms have their own computing limits and we have different datasets for them, to be detailed in the following sections.

4.2 Experiment 1

This experiment is intended to make comparisons among the exact algorithms proposed for the general MQC/MQI problem, including the dynamic programming algorithm by Ben-Dor et al. [2] (denoted as DP), the constraint programming approach in our previous work [14] (denoted as CP), the modified GN algorithm (denoted as GN-Opt), and our lookahead branch-and-bound algorithm (denoted as LBnB-Opt). Since DP and CP do not take k as an input and in fact their running times are independent on k, we generated datasets defined by a pair (n, p) where for each dataset p records the percentage of quartet errors in the given complete quartet set. For GN-Opt and LBnB-Opt, we used $k = \binom{n}{4} \times p$ as the first upper bound on the number of quartet errors. We used quartet error percentage $p = 1\%, 5\%, 10\%, 15\%, 20\%, 30\%$. The largest value set for n was 25, since LBnB-Opt, which appeared to run the fastest, failed to terminate in 100 hours for pair $(30, 30\%)$. Some of the running time results for these four algorithms are summarized in Table 2, where a '$-$' indicates that an algorithm didn't terminate in 100 hours. It is surprising to see that GN-Opt performed inferior to DP and CP on most of the datasets. It is very encouraging to see that LBnB-Opt outperformed the other three algorithms on all datasets.

Problem Size		DP	CP	GN-Opt	LBnB-Opt
$n = 10$	$p = 1\%$	2 secs	1 sec	1 secs	1 sec
	$p = 5\%$	2 secs	1 sec	1 secs	1 sec
	$p = 10\%$	2 secs	1 sec	5 secs	1 sec
	$p = 15\%$	2 secs	1 sec	16 secs	1 sec
	$p = 20\%$	2 secs	1 sec	35 secs	1 sec
	$p = 30\%$	2 secs	1 sec	2 mins	1 sec
$n = 15$	$p = 1\%$	2 mins	1 sec	20 secs	1 sec
	$p = 5\%$	2 mins	1 sec	10 mins	1 sec
	$p = 10\%$	2 mins	1 sec	5 hours	10 secs
	$p = 15\%$	2 mins	1 sec	$-$	30 secs
	$p = 20\%$	2 mins	1 sec	$-$	1 min
	$p = 30\%$	2 mins	1 sec	$-$	2 mins
$n = 20$	$p = 1\%$	40 hrs	10 mins	20 mins	1 sec
	$p = 5\%$	40 hrs	40 mins	$-$	2 mins
	$p = 10\%$	40 hrs	6 hrs	$-$	10 mins
	$p = 15\%$	40 hrs	6 hrs	$-$	40 mins
	$p = 20\%$	40 hrs	8 hrs	$-$	3 hrs
	$p = 30\%$	40 hrs	10 hrs	$-$	10 hrs
$n = 25$	$p = 1\%$	$-$	20 mins	9 hrs	5 secs
	$p = 5\%$	$-$	10 hrs	$-$	10 mins
	$p = 10\%$	$-$	$-$	$-$	1 hr
	$p = 15\%$	$-$	$-$	$-$	3 hrs
	$p = 20\%$	$-$	$-$	$-$	18 hrs

Table 2: The running times for four exact algorithms for the MQC/MQI problem: Dynamic Programming (DP), Constraint Programming (CP), Modified GN Algorithm (GN-Opt), and Lookahead Branch-and-Bound Algorithm (LBnB-Opt). A '$-$' indicates that an algorithm didn't terminate in 48 hours.

4.3 Experiment 2

This experiment is designed to make sole comparison between the GN algorithm and our lookahead branch-and-bound algorithm. As we have seen in Experiment 1 that in terms of finding the optimal solution, LBnB-Opt outperformed GN-Opt significantly. Therefore, in this experiment we only compare them in finding the first solution, in which the number of quartet errors is at most k. We denote these two algorithms as GN-1st and LBnB-1st, respectively. We generated 10 datasets for each pair (n, k), where $n = 10, 20, 30, 40, 50$ and $k = 5, 10, 20, 30, 40, 50, 100, 200$. For every value of n, we found both GN-1st and LBnB-1st terminated within a second on the datasets corresponding to some small values of k; and only when $k \geq 40$ their running times start to depart. Another observation is that for each dataset in the experiment, surprisingly, the number of quartet errors in the solution by GN-1st is equal to the number of quartet errors in the solution by LBnB-1st, and is equal to k. For these reasons, we chose to report their running times only, and only for datasets with $k \geq 40$, in Table 3. We set the time limit to be 24 hours in this experiment because of time constraint. From the table, we see that LBnB-1st outperformed GN-1st on all datasets, slightly for small values of k but significantly for large values of k.

Problem Size		GN-1st	LBnB-1st
$n = 10$	$k = 40$	1 sec	1 sec
	$k = 50$	15 secs	1 sec
$n = 20$	$k = 40$	3 secs	1 sec
	$k = 50$	10 secs	1 sec
	$k = 100$	35 mins	20 secs
	$k = 200$	–	2 mins
$n = 30$	$k = 40$	1 sec	1 sec
	$k = 50$	6 secs	2 secs
	$k = 100$	21 mins	35 secs
	$k = 200$	–	5 mins
$n = 40$	$k = 40$	1 sec	1 sec
	$k = 50$	2 secs	2 secs
	$k = 100$	15 mins	1 min
	$k = 200$	–	10 mins
$n = 50$	$k = 40$	1 sec	1 sec
	$k = 50$	3 secs	3 secs
	$k = 100$	6 mins	2 mins
	$k = 200$	–	20 mins

Table 3: The running time comparison between two algorithms for the MQC/MQI problem where the number of quartet errors is bounded: the Fixed-Parameter Algorithm (GN-1st) and the Lookahead Branch-and-Bound Algorithm (LBnB-1st). A '–' indicates that an algorithm didn't terminate in 24 hours.

4.4 Experiment 3

The hypercleaning algorithm is known as a best heuristics for the MQC/MQI problem (at the writing of this paper). The largest instance reported in [15, 4] contains 18 taxa. This third experiment is designed to compare the performances of our algorithm and the hypercleaning algorithm in terms of both running time and the quality of the returned solution. For this purpose, we generated datasets for $n = 30, 40, 50$ with various bounds on the number of quartet errors $k = 100, 200, 300$. Note that smaller values of n and larger values of k do not make sense since they correspond to either too easy datasets or too difficult datasets. Our algorithm was run to return both the first solution and the optimal solution (that is, by LBnB-1st and LBnB-Opt, respectively). It is again interesting to note that all three solutions for a dataset contain a same number of quartet errors.[1] Subsequently, we chose to report the running time only in Table 4. From the statistics, it is not surprising to see that LBnB-Opt requires much more time to search a whole tree than LBnB-1st which terminates at the time one solution is found. Nonetheless, it is encouraging to see that LBnB-1st beats Hypercleaning in some "easy" datasets.

[1]The value of m in Hypercleaning on these datasets ranges from 1 to 3. This might provide a reason that its solutions are optimal.

Problem Size		Hypercleaning	LBnB-1st	LBnB-Opt
$n = 30$	$k = 100$	40 secs	35 secs	6 mins
	$k = 200$	55 secs	5 mins	36 mins
	$k = 300$	80 secs	45 mins	2 hrs
$n = 40$	$k = 100$	10 mins	1 min	20 mins
	$k = 200$	12 mins	10 mins	3 hrs
	$k = 300$	18 mins	135 mins	18 hrs
$n = 50$	$k = 100$	50 mins	2 mins	2 hrs
	$k = 200$	55 mins	20 mins	12 hrs
	$k = 300$	62 mins	3 hrs	–

Table 4: The running time comparison between three algorithms for the MQC/MQI problem: the Hypercleaning Algorithm and two versions of the Lookahead Branch-and-Bound Algorithm (LBnB-1st and LBnB-Opt). A '–' indicates that an algorithm didn't terminate in 24 hours.

4.5 A Real Dataset

We have obtained a real dataset which contains 30 Avian Influenza viruses with their pairwise evolutionary distances measured by the Complete Composition Vector defined on the whole set of proteins for the viruses [?]. We applied the Four-Point Method [7] on this 30×30 distance matrix to infer a quartet for every subset of 4 taxa. We ran the Neighbor-Joining algorithm [12] on the distance matrix too to produce a phylogeny on these 30 taxa and use the set of induced quartet to estimate an upper bound of 2272. Unfortunately GN-1st did not terminate in 100 hours. Hypercleaning finished in 1 hour and 2 minutes and the returned phylogeny resolves 1922 quartet errors. Our algorithms LBnB-1st and LBnB-Opt finished in 21 hours and 29 hours, respectively, and the returned phylogenies both resolve 1871 quartet errors.

5 Concluding Remarks

Among the exact algorithms for solving the MQC/MQI problem, we found the dynamic programming (DP) the least powerful and our lookahead branch-and-bound algorithm (LBnB-Opt) seemingly the most powerful. Nonetheless, the performance of DP (and CP) is not affected by the number of quartet errors and thus DP might be superior for datasets containing a small number of taxa.

The fixed-parameter algorithm solves the MQC/MQI problem in the special case where the number of quartet errors is known exactly. This special case is unlikely to happen, rather than that, the more often we find it easy to obtain an upper bound on the number of quartet errors. The fixed-parameter algorithm can be modified to solve the general problem, however, our experiments showed that the running time grows too fast to be feasible. Our lookahead branch-and-bound algorithm might be regarded as an improvement over the fixed-parameter algorithm. It outperforms in general significantly in both finding the first solution and in finding the optimal solution. One of our future work is to design more techniques to speedup the search, including faster determination of need-to-be-changed quartets and need-to-be-fixed quartets. Our expectation is that the algorithm has a competitive running time to hypercleaning on datasets of modest size, say 50 taxa.

Lastly, we remark that all experiments were done for the purpose of testing the speed of the lookahead branch-and-bound algorithm. Though on all the synthetic datasets every algorithm found the optimal solution, we didn't compare the quality of the phylogenies for the real dataset.

References

[1] H. Bandelt and A. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advance in Applied Mathematics*, 7:309–343, 1986.

[2] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg. From four-taxon trees to phylogenies: the case of mammalian evolution. In *Proceedings of the RECOMB*, pages 9–19, 1998.

[3] V. Berry, T. Jiang, P. E. Kearney, M. Li, and H. T. Wareham. Quartet cleaning: Improved algorithms and simulations. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA 99)*, pages 313–324, 1999.

[4] D. Bryant, V. Berry, T. Jiang, P. Kearney, M. Li, T. Wareham, and H. Zhang. A practical algorithm for recovering the best supported edges of an evolutionary tree. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 287–296, 2000.

[5] H. Colonius and H. H. Schultze. Tree structure for proximity data. *British Journal of Mathematical and Statisticcal Psychology*, 34:167–180, 1981.

[6] G. Della Vedova and H. T. Wareham. Optimal algorithms for local vertex quartet cleaning. *Bioinformatics*, 18:1297–1304, 2002.

[7] P. Erdos, M. Steel, L. Szekely, and T. Warnow. A few logs suffice to build (almost) all trees (Part 1). *Random Structures and Algorithms*, 14:153–184, 1999.

[8] J. Gramm and R. Niedermeier. A fixed-parameter algorithm for minimum quartet inconsistency. *Journal of Computer and System Science*, 67:723–741, 2003.

[9] T. Jiang, P. E. Kearney, and M. Li. Orchestrating quartets: approximation and data correction. In *Proceedings of the 39th FOCS*, pages 416–425, 1998.

[10] T. Jiang, P. E. Kearney, and M. Li. Some open problems in computational molecular biology. *Journal of Algorithms*, 34:194–201, 2000.

[11] T. Jiang, P. E. Kearney, and M. Li. A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application. *SIAM Journal on Computing*, 30:1942–1961, 2001.

[12] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[13] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal on Classification*, 9:91–116, 1992.

[14] G. Wu, G.-H. Lin, and J. You. Quartet based phylogeny reconstruction with answer set programming. In *Proceedings of the 16th ICTAI*, pages 612–619, 2004.

[15] H. Zhang. Design, implementation, and analysis of a novel quartet-based phylogenetic reconstruction method. Master's thesis, Department of Computer Science, University of Waterloo, 2000.