Journal of Algorithms **38,** 237–259 (2001)
doi:10.1006/jagm.2000.1133, available online at http://www.idealibrary.com on IDEAL®

# Constructing Optimal Trees from Quartets

David Bryant

*C.R.M., Université de Montréal, Québec and LIRMM, Montpellier, France*
E-mail: bryant@crm.umontreal.ca

and

Mike Steel

*Biomathematics Research Centre, University of Canterbury, Christchurch, New Zealand*
E-mail: m.steel@math.canterbury.ac.nz

Received June 7, 1999

We present fast new algorithms for constructing phylogenetic trees from quartets (resolved trees on four leaves). The problem is central to divide-and-conquer approaches to phylogenetic analysis and has been receiving considerable attention from the computational biology community. Most formulations of the problem are NP-hard. Here we consider a number of constrained versions that have polynomial time solutions. The main result is an algorithm for determining bounded degree trees with optimal quartet weight, subject to the constraint that the splits in the tree come from a given collection, for example, the splits in the aligned sequence data. The algorithm can search an exponentially large number of phylogenetic trees in polynomial time. We present applications of this algorithm to a number of problems in phylogenetics, including sequence analysis, construction of trees from phylogenetic networks, and consensus methods. 

*Key Words:* quartets; phylogenetic trees; algorithms; consensus; networks.

## 1. INTRODUCTION

The reconstruction of large evolutionary (phylogenetic) trees from smaller subtrees is currently receiving considerable attention in the computational biology community [6, 10, 23, 29, 31, 37, 38].

There is a clear computational advantage to analyzing small subsets of taxa (species). It allows for far more intensive analysis and the application of more complex models to reconstruct trees from the sequence data. Tree criteria such as maximum likelihood, which are computationally horren-



0196-6774/01 $35.00

dous on larger trees, can be solved quickly on four-leaf trees (quartets), where there are just four possible trees to consider.

There are also biological and statistical advantages of considering only small subsets of sequences at a time. In many cases the actual data limit the number of sequences that can be analyzed at one time. The number of sites that can be aligned across four sequences is generally much more than the number of sites that can be aligned across the full set of $n$ sequences, so aligning over the complete set of sequences can result in lost information. Second, a recognized source of error in standard tree building methods such as neighbor joining is that distantly related sequences can mislead tree reconstruction [29]. If only small sets of sequences are considered at one time then those sets containing distantly related sequences can be down-weighted (or even given a zero weighting, as in [23, 29]).

The main difficulty with quarter-based methods is the question of how best to build large trees out of small ones. The general problem—determining a phylogenetic tree that agrees with the largest number of quartets, or maximum weight set of quartets—is NP-hard by a simple reduction from *quartet compatibility* [36]. An exhaustive search is generally infeasible: there are $1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n-5)$ binary trees on $n$ leaves to choose from. When the number of sequences is limited, and the computational time is not, the exact algorithm of [6] can be used: it runs in time $O(n^4 3^n)$ on $n$ sequences and $O(n^4)$ quartets.

The next alternative to exact solutions is the use of heuristic algorithms for quartet optimization. These have been produced by a number of computer scientists, biologists, and mathematicians. The heuristics of Sattath and Tversky [35], Fitch [26], Colonius and Schulze [14], and Bandelt and Dress [1] combine clustering procedures with a pairwise similarity or neighborliness score derived from the quartet sets. An alternative agglomerative algorithm for constructing trees from quartets is provided in [7].

A novel variation on the scoring approach is described by Ben-Dor *et al.* [6]. Instead of constructing a similarity score and then clustering, they embed the $n$ leaves as points in $\Re^n$ using semi-definite programming and then apply a nearest neighbor clustering method.

The tree building, or 'puzzling', part of the Quartet Puzzling heuristic of Strimmer and von Haeseler [37] works by ordering the leaf set arbitrarily, constructing a tree on the first four leaves, and then adding new leaves one at a time, attaching each leaf to the edge that gives optimum quartet score. The same approach is used in [15, 38] to optimize according to different, but related, criteria. These procedures can be seen as analogues of the Wagner tree method [24] because they start with a small tree and insert one leaf at a time.

Dekker [17] proposes a method for constructing trees from quartets and other subtrees using quartet inference rules (see also [11]). The Short

Quartet Method [23] constructs trees using inference rules and greedy selection of quartets.

One important problem with these heuristic approaches is that there has been little systematic analysis of their strengths and weaknesses. Indeed, the quartet approximation problem seems to be resistant to an approximation theoretic approach. A version of the problem was shown to have a PTAS by [30], but the complexity of the approximation algorithm is so astronomical that the result is of theoretical interest only.

Polynomial time exact algorithms have been proposed for a number of constrained versions. The $Q^*$ method of Berry and Gascuel [10] can be applied when there is at most one quartet in the input set for each four leaves and the output tree is constrained to have only quartets from this set. There is always exactly one maximal tree satisfying these conditions. The $Q^*$ method is employed by Kearney [31] to construct trees from quartets selected by an ordinal quartet method.

The $Q^*$ method can be extended by weakening the constraint that all the quartets in the tree come from the input set. The quartet cleaning method [30], $C$-tree construction [8], and hypercleaning method [9] all allow varying degrees of errors in the input set. All run in polynomial time and are well suited for the situation when the quartet set is unweighted and almost tree-like.

In this paper we present a polynomial time algorithm for a constrained version of the quartet optimization problem. The algorithms are fast enough to be applied to moderately large data sets. The constraints are not overly restrictive—the algorithm still searches an exponentially large number of trees—and are general enough to be applied to a wide range of phylogenetic problems. Finally, we note that the algorithms can be applied to weighted sets of quartets, with the possibility of more than one quartet for each set of four leaves.

In the remainder of this section we present basic definitions (Section 1.1), describe the main result (Section 1.2), and outline a number of applications of the algorithm. In Section 2 we present the constrained quartet optimization algorithm. In Section 3 we show how the algorithm can be applied to the extraction of phylogenies from phylogenetic networks and describe the efficiency gains that can be made in this application.

### 1.1. *Basic Definitions*

Hypotheses about the evolutionary relationships between taxa (species) are usually described in terms of a phylogenetic tree. When no ancestral node is given we have an *unrooted* (*phylogenetic*) *tree* which can be formally defined as an acyclic connected graph with no vertices of degree two and all leaves (degree one vertices) labeled uniquely from some leaf set $L$ representing the set of taxa. A phylogenetic tree is *binary* or *resolved* if all internal vertices have degree three.

A phylogenetic tree clearly implies relationships between each subset of its leaf set. This is captured in the notion of induced subtrees. Let $T$ be an unrooted phylogenetic tree, and let $A$ be a subset of its leaf set. Consider the minimal subgraph $T(A)$ of $T$ that connects elements of $A$. Delete all vertices of degree two in $T(A)$ and identify their adjacent edges, thereby obtaining an unrooted phylogenetic tree with leaf set $A$. This tree is called the *subtree of* $T$ *induced by* $A$ and is denoted $T_{|A}$.

The information contained in phylogenetic trees can be coded in a number of ways. Here we consider two encodings: sets of splits and sets of quartets.

A *split* $A|B$ is a partition of the leaf set into two nonempty parts, $A$ and $B$. If we remove an edge $e$ of a phylogenetic tree we divide the tree into two connected components and induce a split of the leaf set of the tree. This split is called the *split associated with* $e$, and the set of all such splits in a tree is denoted $splits(T)$. If $e$ is an external edge then we obtain a split $A \mid B$ with $|A| = 1$ or $|B| = 1$. We call these splits *trivial* splits. Note that $T$ can be reconstructed from the set $splits(T)$.

A *quartet* is a resolved phylogenetic tree on four leaves. There are three possible quartets on a given set of four leaves $\{a, b, c, d\}$. We use $ab \mid cd$ to denote the quartet where $a$ and $b$ are separated from $c$ and $d$ by the internal edge. A phylogenetic tree $T$ *agrees* with a quartet $ab \mid cd$ if $a, b, c, d$ are all leaves of $T$ and the path from $a$ to $b$ does not share any vertices with the path from $c$ to $d$, that is, if $T_{|\{a, b, c, d\}} = ab \mid cd$. Let $q(T)$ denote the set of quartets that $T$ agrees with. We can reconstruct $T$ from $q(T)$.

To illustrate, we give a simple example (Fig. 1). We have selected two internal edges and give the associated splits. We have also selected two sets of four leaves and given the induced quartets.

We need three further definitions in order to be able to summarize our results. First, a set of splits $\mathscr{S}$ is *compatible* if $\mathscr{S} \subseteq splits(T)$ for some tree $T$. Second, a set of splits is *weakly compatible* if for every three splits $A_1 \mid B_1$, $A_2 \mid B_2$, $A_3 \mid B_3$ at least one of the intersections $B_1 \cap B_2 \cap B_3$, $B_1 \cap A_2 \cap A_3$, $A_1 \cap B_2 \cap A_3$, $A_1 \cap A_2 \cap B_3$ is empty [3]. Finally, a set of weakly compatible splits $\mathscr{S}$ on $X$ is *maximum* if $|\mathscr{S}| = n(n-1)/2$ (see [3]).

### 1.2. *Main Results*

Let $w$ be a weighting function defined on the set of all quartets with leaves in $L$. The weights can be negative and do not have to be integers.

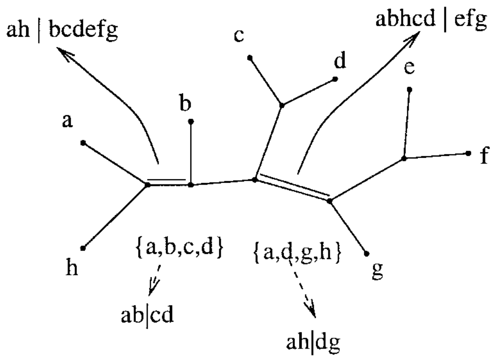


FIG. 1. Splits and quartets of an unrooted phylogenetic tree.

We define the weight of the tree to be

$$w(T) = \sum_{ab|cd \in q(T)} w(ab \mid cd), \tag{1}$$

the sum of all quartet weights in the tree. We will be examining the following problem:

Split Constrained Quartet Optimization

Instance: Weighting $w$ for the quartets on a leaf set $L$. Set $\mathscr{S}$ of splits of $L$. Rational number $\lambda$.

Parameter: Degree bound $d$.

Question: Is there a tree $T$ with vertex degree bounded by $d$ and *splits*$(T) \subseteq \mathscr{S}$ such that $w(T) \geq \lambda$?

Let $n = |L|$ and $k = |\mathscr{S}|$. The computational complexity of this problem can be summarized:

- Polynomial time solvable for bounded $d$, with time complexity $O(n^4 k + n^2 dk^{d-1})$ (Section 2). Can be improved to $O(n^{d+2})$ time when $\mathscr{S}$ is weakly compatible and $d$ equals 3 or 4 (Section 3.1).
- NP-complete without the degree bound $d$, even when the set of splits $\mathscr{S}$ is *weakly compatible* (see Section 3.3).
- Polynomial time solvable *without* degree bound when all quartet weights are nonnegative and $\mathscr{S}$ is *maximum weakly compatible* (Section 3.2).
- NP-complete when the quartet's weights can be negative and $d$ is unbounded, even when $\mathscr{S} = $ *splits*$(T)$ for some tree $T$ (Section 3.3).

The results can be viewed as an extension of the compatibility algorithms of [11]. In [11] the splits, rather than the quartets, are weighted and the criteria of optimization is the sum of the weights of the splits in a tree. The significance of these results is highlighted by the fact that determining an optimal tree with respect to split weights and no degree bound is equivalent to *max clique* [16] and so inherits the depressing complexity attributes of max clique like W[1]-hardness [19] and nonapproximability [5].

### 1.3. *Applications*

We outline a number of possible applications of the split constrained quartet optimization algorithms.

1.3.1. *Analysis of sequence data*. A natural source of splits to serve as a split constraint for the algorithm is the sequence data. We first use an alignment program to determine which positions (or *sites*) in one sequence correspond to which positions in the other sequences. For each position we then obtain a map from the set of sequences to the nucleotide for that sequence at that position. In DNA and RNA there are four nucleotides possible, and so there are eight ways of partitioning the nucleotides into two groups. Each of these partitions can be used to construct a split of the set of sequences.

We have, then, a three step process for inferring phylogenetic trees from aligned sequence data:

(1) Extract the set $\mathcal{S}$ of splits given by the characters at each site in the data.

(2) For each set of four sequences, score the three possible quartet trees using a standard phylogenetic optimization criterion (e.g., parsimony length, likelihood score). The quartet scores can be scaled to indicate relative confidence.

(3) Apply the constrained quartet optimization algorithm to the set of splits constructed in Step (1) with the quartet weighting constructed in Step (2).

1.3.2. *Extracting trees from phylogenetic networks*. An approach to phylogenetic analysis that is growing in popularity is the construction of phylogenetic networks, where the evolutionary relationships are represented by a general graph rather than just a tree.

Phylogenetic networks allow for a more complicated relationship between the different species and can incorporate recombination, hybridization, and horizontal gene transfer. In some cases the data dictate that a tree representation is not suitable, as in the complex evolutionary relations between viruses or in intraspecific data with multiple hybridizations.

Phylogenetic networks can also be employed as an intermediary step in phylogenetic tree reconstruction. Often a network program such as SplitsTree [21] is used to get a general representation of patterns in the data and an indication of how tree-like the data actually are. However, the problem still remains—given a phylogenetic network how does one best extract a phylogenetic tree?

We apply the constrained quartet optimization algorithms to this problem by first converting the network into a collection of splits. In Section 3 we discuss this approach in further detail and show that time complexity gains can be made by exploiting the structure of the network.

1.3.3. *Consensus trees, bootstrapping, and quartet puzzling.* A common problem faced by practitioners in evolutionary biology is the representation of a large collection of trees on the same leaf set by a single *consensus* tree. Tree search criteria such as likelihood can have multiple global optima. Heuristic construction methods (such as quartet puzzling [37]) that involve randomness can construct different trees on different runs, and the user will want to make multiple runs in order to achieve a degree of confidence in the final hypothesis. Bootstrapping, and its close cousin jack-knifing, work on the same principle. The data are randomly sampled and these possibly incomplete samples are used as input for the tree reconstruction criteria. The collection of trees obtained is then used to determine confidence levels for a particular evolutionary hypothesis.

By far the most common consensus technique is the majority rule tree, formed from splits that appear in over half of the input trees. Unfortunately this method will often given uninformative consensus trees, with few internal edges. A rogue taxa that appears in a large number of different places (perhaps because it is only distantly related to the other taxa) can force the consensus tree to collapse completely. A major drawback of the popular Quartet Puzzling method [37] is that the consensus tree it produces tends to be quite poorly resolved.

The constrained quartet optimization algorithm provides a natural solution to the consensus tree problem. We first construct the set of all splits that appear in at least one of the input trees. If the input trees are binary then this set of splits is guaranteed to contain the set of splits of some binary tree, so we can always use a small degree bound. The quartet weighting can be taken from the input data, as in the previous section, or by counting the number of times each quartet appears in an input tree. In this way the consensus technique can be extended to handle weighted trees. Finally, the constrained quartet optimization algorithm can be used to construct, in polynomial time, a consensus tree for the input set of trees.

1.3.4. *Optimal trees with excluded quartets.* Suppose that we are given, for each set of four leaves $\{a, b, c, d\}$, a quartet to *exclude*. We wish to find a tree $T$ of optimal quartet weight such that $q(T)$ contains none of the excluded quartets.

We can solve this problem when we also have a degree bound for $T$. Let $Q$ be the set of excluded quartets. We first construct the set of splits

$$S = \{A \mid B : aa' \mid bb' \notin Q, \text{ all } a, a' \in A, b, b' \in B\}. \tag{2}$$

This set is weakly compatible [4] and, furthermore, $q(T) \cap Q = \emptyset$ if and only if $splits(T) \subseteq \mathscr{S}$. Hence the problem of finding an optimal tree $T$ containing no excluded quartets reduces to the Split Constrained Quartet Optimization problem. In Section 3 we give efficient algorithms for constrained quartet optimization when $\mathscr{S}$ is weakly compatible.

Note that if we do not force $Q$ to contain an excluded quartet for every set of four leaves then it becomes NP-hard to determine if there exists a binary tree $T$ such that $q(T) \cap Q = \emptyset$ [12].

1.3.5. *Optimal trees with a given circular order*. A novel approach to phylogenetic tree construction was introduced by Gonet *et al*. in [32]. They first construct a tour $x_1, x_2, \dots, x_n, x_1$ of the set $L$ of leaves using traveling salesman algorithms. They then look for a phylogenetic tree $T$ on $L$ such that $x_1, x_2, \dots, x_n, x_1$ is a *circular order* of $T$; that is, each edge in $T$ lies on exactly two paths connecting adjacent vertices in the tour. There are $2^{n-2}$ possible circular orderings for a binary tree on $n$ leaves [33].

Construct the set

$$\mathscr{S} = \big\{\{x_i, x_{i+1}, \dots, x_j\} \mid L - \{x_i, x_{i+1}, \dots, x_j\} : 1 \le i \le j \le n-1\big\}. \tag{3}$$

This set is maximum weakly compatible. Furthermore, $x_1, \dots, x_n$ is a circular ordering for a tree $T$ if and only if $splits(T) \subseteq \mathscr{S}$. In Section 3.2 we show that the Split Constrained Quartet Optimization problem can be solved in polynomial time *without a degree bound* when $\mathscr{S}$ is maximum weakly compatible and all quartet weights are nonnegative. Hence, given a tour $x_1, \dots, x_n, x_1$ of $L$ and a positive weight for every quartet on $L$, we can determine an optimal weight tree $T$ from among the exponentially many trees that have $x_1, \dots, x_n, x_1$ as a circular order.

## 2. CONSTRAINED QUARTET OPTIMIZATION ALGORITHMS

The key component of the dynamical programming algorithm of [11] is a data structure called the decomposition table, which we describe in Section 2.2. We will also use a decomposition table, though we optimize a different, and more complex, criterion. First, however, we introduce rooted trees and clusters.

## 2.1. *Rooted Trees and Clusters*

A *rooted phylogenetic tree* is defined in the same way as an unrooted phylogenetic tree, except that one vertex, which may have degree two, is distinguished and called the *root*. Given any two vertices $u, v$ in a rooted phylogenetic tree, if the path from $u$ to the root passes through $v$ then we say that $u$ is a *descendent* of $v$. The descendents of a vertex $v$ that are also adjacent to $v$ are called the *children* of $v$. A rooted phylogenetic tree is *binary* or *fully resolved* if every internal vertex has exactly two children.

The rooted analogue of a split is a *cluster*. Given a vertex $v$ in a rooted tree the set of leaves that are descendents of $v$ is called the cluster *associated with* $v$. The set of all clusters associated to vertices in a rooted tree $T$ is denoted *clus*$(T)$.

We will often be converting rooted trees into unrooted trees. Suppose that $T$ is a rooted tree such that $\mathscr{L}(T) \subset L$ and $\mathscr{L}(T) \neq L$. We let UNROOT$(T, L)$ be the unrooted tree given by attaching all leaves in $L - \mathscr{L}(T)$ to the root of $T$ and then taking the underlying unrooted topology. For example the trees in Fig. 2(i) and 2(ii) have unrooted equivalents equal to those trees in Fig. 2(iii) and 2(iv).

## 2.2. *The Decomposition Table*

The key data structure in the algorithm is a *decomposition table* $\mathscr{D} = (\mathscr{C}, D)$. Here $\mathscr{C} = C_1, C_2, \ldots, C_K$ is a collection of clusters of some leaf set

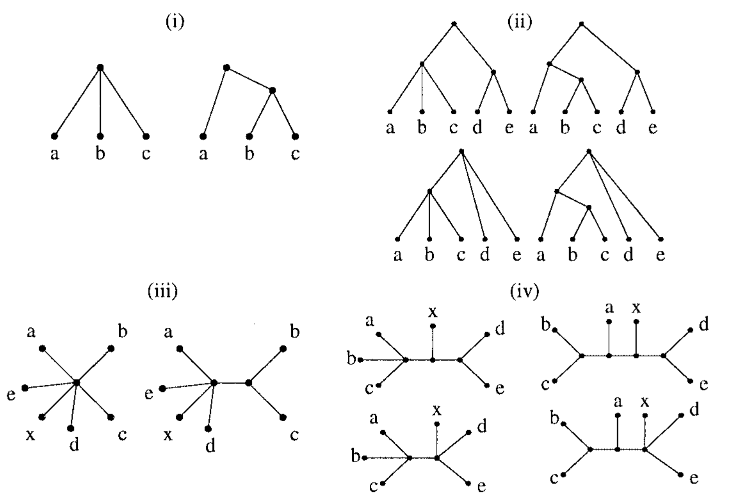


FIG. 2. The trees in (i) $\mathscr{T}(\mathscr{D}, 5)$; (ii) $\mathscr{T}(\mathscr{D}, 9)$; (iii) $\mathscr{T}^*(\mathscr{D}, 5)$; and (iv) $\mathscr{T}^*(\mathscr{D}, 9)$.

$L$, and $D$ is a table with a row $D[i]$ for each cluster $C_i$. Row $D[i]$ contains a list of unordered tuples $[p_1, p_2, \ldots, p_q]$ satisfying

- $q \geq 2$.
- $C_{p_1}, C_{p_2}, \ldots, C_{p_q}$ are pairwise disjoint.
- $C_i = C_{p_1} \cup C_{p_2} \cup \cdots \cup C_{p_q}$.

Note that one row can contain tuples of varying lengths and that we are not concerned with the ordering of the indices within the tuple. Let $\|\mathscr{D}\|$ denote the sum of the lengths of all the tuples in all the rows of $\mathscr{D}$.

To each row $D[i]$ of the decomposition table we associate a set of rooted trees $\mathscr{T}(\mathscr{D}, i)$, all of which have leaf set $C_i$. The set $\mathscr{T}(\mathscr{D}, i)$ is defined recursively:

- if $C_i = \{a\}$ for some leaf $a$ then $\mathscr{T}(\mathscr{D}, i)$ contains the single vertex tree with leaf $a$;
- if $|C_i| \geq 2$ and $D[i] = \emptyset$ then $\mathscr{T}(\mathscr{D}, i) = \emptyset$.
- Otherwise, $\mathscr{T}(\mathscr{D}, i)$ is the set of all possible trees that can be formed by choosing a tuple $[p_1, \ldots, p_r]$ in $D[i]$, choosing subtrees $T_j \in \mathscr{T}(\mathscr{D}, j)$ for each $j = 1, \ldots, r$, and attaching the roots of these subtrees to a new vertex that becomes the root of a rooted tree with leaf set $C_i$.

Since there may be tuples $[p_1, \ldots, p_r]$ in $D[i]$ with $\mathscr{T}(\mathscr{D}, p_j) = \emptyset$ for some $j$, we can have $\mathscr{T}(\mathscr{D}, i) = \emptyset$ even though $D[i] \neq \emptyset$. Note that $\mathscr{T}(\mathscr{D}, i)$ can contain exponentially many trees with respect to the size of $\mathscr{D}$ [11].

We use dynamical programming to enumerate or extract the trees in $\mathscr{T}(\mathscr{D}, i)$. Construct a table $s$ by putting $s[i] = 1$ for all $i$ such that $|C_i| = 1$ and putting

$$s[i] = \sum_{\{p_1, p_2, \ldots, p_q\} \in D[i]} s[p_1] \times s[p_2] \times \cdots \times s[p_q] \tag{4}$$

when $|C_i| > 1$. Then $s[i] = |\mathscr{T}(\mathscr{D}, i)|$ and the values $s[i]$ can be calculated in time $O(\|\mathscr{D}\|)$ [11].

Once the trees in $\mathscr{T}(\mathscr{D}, i)$ have been enumerated we can extract trees from the collection as follows:

- If $s[i] = 0$ return $\emptyset$,
- else if $|C_i| = 1$ return the single vertex tree labeled by the leaf in $C_i$,
- else choose a tuple $[p_1, \ldots, p_q]$ such that $s[p_1] \times s[p_2] \times \cdots \times s[p_q] \neq 0$.

Extract trees $T_1, \ldots, T_q$ from $\mathscr{T}(\mathscr{D}, 1), \mathscr{T}(\mathscr{D}, 2), \ldots, \mathscr{T}(\mathscr{D}, q)$, respectively. Attach the roots of $T_1, \ldots, T_q$ to a new vertex that becomes the root of a tree $T$ in $\mathscr{T}(\mathscr{D}, i)$. Return $T$.

To illustrate, we give a simple example. Table 1 represents a decomposition table for a collection of clusters of the leaf set $L = \{a, b, c, d, e\}$. Then $\mathscr{T}(\mathscr{D}, 5)$ contains the two trees in Fig. 2(i). Each of the trees in $\mathscr{T}(\mathscr{D}, 5)$ appears as a subtree of two trees in $\mathscr{T}(\mathscr{D}, 9)$, giving a total of four trees in $\mathscr{T}(\mathscr{D}, 9)$ (Fig. 2(ii)). All other collections $\mathscr{T}(\mathscr{D}, i)$ contain single trees.

Since decomposition tables can be used to store rooted trees, we can also use them to store unrooted trees. First fix a leaf $x$, acting as an outgroup, and consider a collection $\mathscr{C}$ of clusters on $L - \{x\}$. Let $\mathscr{D}$ be a decomposition table for $\mathscr{C}$. For each $C_i \in \mathscr{C}$ we define the set of unrooted trees

$$\mathscr{T}^*(\mathscr{D}, i) = \{\textsc{unroot}(T, L) : T \in \mathscr{T}(\mathscr{D}, i)\}. \tag{5}$$

The operation UNROOT is described in Section 2.1.

Returning to our example, suppose that $L' = \{a, b, c, d, e, x\}$. Then $\mathscr{T}^*(\mathscr{D}, 5)$ contains the two unrooted trees in Fig. 2(iii) while $\mathscr{T}^*(\mathscr{D}, 9)$ contains the four trees in Fig. 2(iv).

### 2.3. *Optimal Weight Trees in Decomposition Tables*

Suppose that $\mathscr{D} = (\mathscr{C}, D)$ is a decomposition table for a set of clusters $\mathscr{C}$ of $L - \{x\}$. The collections $\mathscr{T}^*(\mathscr{D}, i)$ can contain exponentially many trees, even when $\mathscr{D}$ is only polynomial in size. Here we show how to locate, from among these exponentially many trees, a tree with maximum summed quartet weight. The algorithm takes $O(n^4|\mathscr{C}| + n^2\|\mathscr{D}\|)$ time.

TABLE 1
Decomposition Table for a Collection of Clusters of the Leaf Set $L = \{a, b, c, d, e\}$

| $i$ | $C_i$ | $D[i]$ | $s[i]$ |
|---|---|---|---|
| 1 | $\{a\}$ | — | 1 |
| 2 | $\{b\}$ | — | 1 |
| 3 | $\{c\}$ | — | 1 |
| 4 | $\{b, c\}$ | [2, 3] | 1 |
| 5 | $\{a, b, c\}$ | [1, 2, 3], [1, 4] | 2 |
| 6 | $\{d\}$ | — | 1 |
| 7 | $\{e\}$ | — | 1 |
| 8 | $\{d, e\}$ | [6, 7] | 1 |
| 9 | $\{a, b, c, d, e\}$ | [5, 8], [5, 6, 7] | 4 |

We use dynamic programming. At each step we optimize with respect to a modification of the quartet weighting criteria of Eq. (1).

As before, let $w$ be a weighting function for the quartets with leaves in $L$. For each $C_i \in \mathscr{C}$ put

$$q_i(T) = \{ab \mid cd \in q(T) : |\{a, b, c, d\} \cap C_i| \geq 3\} \tag{6}$$

and

$$w_i(T) = \sum_{ab|cd \in q_i(T)} w(ab \mid cd). \tag{7}$$

We optimize $w_i(T)$ over all of the trees in the collection $\mathscr{T}^*(\mathscr{D}, i)$ defined in Eq. (5). Put

$$\mathrm{m}[i] = \max\{w_i(T) : T \in \mathscr{T}^*(\mathscr{D}, i)\} \tag{8}$$

and

$$\mathrm{M}[i] = \{T \in \mathscr{T}^*(\mathscr{D}, i) : w_i(T) = \mathrm{m}[i]\}. \tag{9}$$

Finally, for each tuple $[p_1, p_2, \ldots, p_q] \in D[i]$ we define

$$Q(p_1, \ldots, p_q) = \bigcup_{j=1}^{q} \{ab \mid cd : a, b \in C_{p_j}, c \in C_i - C_{p_j}, d \in L - C_{p_j}\} \tag{10}$$

and

$$W(p_1, \ldots, p_q) = \sum_{ab|cd \in Q(p_1, \ldots, p_q)} w(ab \mid cd). \tag{11}$$

We can now state the basis for the dynamical programming algorithm.

THEOREM 2.1. *If* $|C_i| = 1$ *then* $\mathrm{m}[i] = 0$; *otherwise*

$$\mathrm{m}[i] = \max\left\{W(p_1, \ldots, p_q) + \sum_{j=1}^{q} \mathrm{m}[j] : [p_1, p_2, \ldots, p_q] \in D[i]\right\}. \tag{12}$$

*Proof.* We prove the result by induction on the size of $C_i$. If $|C_i| = 1$ then $\mathscr{T}^*(\mathscr{D}, i)$ contains only the trivial unrooted tree with no internal edges. This has an empty quartet set and, consequently, zero weight.

Suppose that the result holds for all $C_j \in \mathscr{C}$ such that $|C_j| < |C_i|$. Suppose that $T^* \in \mathrm{M}[i]$. By the definition of $\mathscr{T}^*(\mathscr{D}, i)$ and $\mathrm{M}[i]$ (Eqs. (5) and (9)) there is $T \in \mathscr{T}(\mathscr{D}, i)$ such that $T^* = \textsc{unroot}(T, L)$. Let $T_1, T_2, \ldots, T_q$ be the maximal subtrees of $T$ rooted at the children of the root of $T$.

By the definition of $\mathscr{T}(\mathscr{D}, i)$ there is a tuple $[p_1, \ldots, p_q]$ such that for each $j \in \{1, \ldots, q\}$ we have $C_{p_j} = \mathscr{L}(T_j)$ and $T_j \in \mathscr{T}(\mathscr{D}, p_j)$.

Consider an arbitrary quartet $ab \mid cd \in q_i(T^*)$. Then $|\{a, b, c, d\} \cap C_i| \geq 3$ and exactly one of the following must hold:

- There is $j \in \{1, 2, \ldots, q\}$ such that $|\{a, b, c, d\} \cap C_{p_j}| \geq 3$, or
- There is $j \in \{1, 2, \ldots, q\}$ such that $\{a, b, c, d\} \cap C_{p_j}$ equals $\{a, b\}$ or $\{c, d\}$. Hence

$$w_i(T^*) = \sum_{j=1}^{q} w_{p_j}(T_j) + W(p_1, \ldots, p_q) \tag{13}$$

$$\leq \mathrm{m}[i] \tag{14}$$

by the induction hypothesis applied to $C_{p_1}, \ldots, C_{p_q}$.

Conversely, suppose that $[p_1, \ldots, p_q]$ maximizes Eq. (12). There is $T_1, \ldots, T_q$ such that $unroot(T_j, L) \in \mathrm{M}[p_j]$ for all $j = 1, 2, \ldots, q$. Construct a rooted tree $T$ by attaching the roots of $T_1, \ldots, T_q$ to a new root. Then $unroot(T, L) \in \mathscr{T}^*(\mathscr{D}, i)$ and $w_i(T) = \mathrm{m}[i]$. ∎

Theorem 2.1 leads immediately to a compact representation of the optimal trees. We construct a new decomposition table $\mathscr{D}_{opt} = (\mathscr{C}, D_{opt})$ by letting $D_{opt}[i]$ equal the set of tuples $[p_1, \ldots, p_q]$ in $D[i]$ that maximize Eq. (12). It then follows that

COROLLARY 2.1. *For each $C_i \in \mathscr{C}$ we have*

$$\mathrm{M}[i] = \mathscr{T}^*(\mathscr{D}_{opt}, i). \tag{15}$$

To improve the time complexity we precompute the value

$$W[C_i; a, b] = \sum_{x, y \in C_i} w(xy \mid ab) \tag{16}$$

for all $C_i \in \mathscr{C}$ and all $a, b \in L - C_i$. This takes $O(n^4 |\mathscr{C}|)$ time.

It takes a further $O(n^2 \|D\|)$ time to calculate $\mathrm{m}[i]$ for all $i$ and construct the optimal tree decomposition table $\mathscr{D}_{opt}$, where we use $\|D\|$ to denote the sum of the tuple lengths over all tuples in all rows of $D$. The optimal trees can be enumerated using techniques outlined in Section 2.2.

The complete algorithm is summarized in Algorithm 1.

ALGORITHM 1 (OPTIMALD($\mathscr{D}, w$)).

**begin**
Sort $\mathscr{C} = \{C_1, \ldots, C_k\}$ so that $C_i \subset C_j$ implies $i < j$.
**for** $i$ from 1 to $k$ **do**
**for** $a, b \in L - C_i$ **do**
$W[i; a, b] \leftarrow \Sigma_{x, y \in C_i} w(xy \mid ab)$
**end(for)**
**end(for)**
**for** $i$ from 1 to $k$ **do**
**if** $D[i] = \emptyset$ **then**
$M[i] \leftarrow 0$
**else**
$best \leftarrow -\infty$
**for all** $[p_1, p_2, \ldots, p_q] \in D[i]$ **do**
calculate $W([p_1, p_2, \ldots, p_q])$ using $W[\cdot; \cdot; \cdot]$.
$score \leftarrow W(p_1, \ldots, p_q) + \Sigma_{j=1}^{q} M[p_j]$
**if** $score > best$ **then** $D_{opt}[i] \leftarrow \emptyset$
**if** $score \geq best$ **then**
$D_{opt}[i] \leftarrow D_{opt}[i] \cup \{[p_1, \ldots, p_q]\}$
$best \leftarrow score$
**end(if)**
**end(for all)**
**end(if-else)**
**end(for)**
**end.**

## 2.4. *An Algorithm for Split Constrained Quartet Optimization*

We are now in a position to give the main result. Let $w$ be a weighting on the quartets of a leaf set $L$, let $\mathscr{S}$ be a set of splits of $L$, and let $d$ be a degree bound. We will assume that $\mathscr{S}$ contains all of the trivial splits (those that separate a single element from everything else). Let $n = |L|$ and $k = |\mathscr{S}|$.

Let $x$ be an arbitrary leaf in $L$. We construct a collection of clusters

$$\mathscr{C} = \{A : A \mid B \in \mathscr{S}, x \in B\} \tag{17}$$

and order these $C_1, C_2, \ldots, C_k$ so that $C_i, C_j \in \mathscr{C}$ and $C_i \subset C_j$ implies $i < j$. Hence $C_k = L - \{x\}$, the cluster corresponding to the trivial split $\{x\} \mid L - \{x\}$.

We construct a decomposition table $\mathscr{D} = (\mathscr{C}, D)$ as follows:

- If $|C_i| = 1$ then $D[i] \leftarrow \emptyset$.
- If $|C_i| > 1$ then

$$D[i] \leftarrow \left\{ [p_1, p_2, \ldots, p_q] : \begin{array}{l} C_{p_1}, \ldots, C_{p_q} \text{ are pairwise disjoint} \\ C_i = C_{p_1} \cup \cdots \cup C_{p_q} \\ q \le d - 1 \end{array} \right\}. \tag{18}$$

This table is called the *complete decomposition table* for $\mathscr{C}$ with degree bound $d$. It can be constructed in $O(nk^{d-1})$ time, where $k = |\mathscr{C}|$ and $n = |L|$, by considering all tuples of indices of length less than $d$, computing intersections and union, and then determining if they should be included in some row of $D$.

Furthermore, a simple proof by induction gives

LEMMA 2.1 [11]. *If $\mathscr{D}$ is constructed as above, and $C_k = L - \{x\}$, then $T^* \in \mathscr{T}^*(\mathscr{D}, k)$ if and only if $T^*$ has splits in $\mathscr{S}$ and degree bound $d$.*

The decomposition table $\mathscr{D}_{opt}$ containing the optimal trees in $\mathscr{D}$ can be constructed in $O(n^4 k + n^2 dk^{d-1})$ time using Algorithm 1. We have now established:

THEOREM 2.2. *The Split Constrained Quartet Optimization problem can be solved in $O(n^4 k + n^2 dk^{d-1})$ time.*

## 3. OPTIMAL QUARTET TREES IN PHYLOGENETIC NETWORKS

The algorithm for split constrained optimization described in the previous sections makes no assumptions about the structure of the set of splits $\mathscr{S}$. In many cases, prior knowledge of the structure of $\mathscr{S}$ allows us to achieve tighter complexity bounds or even to drop the degree constraint altogether.

One structure that arises in applications (see Sections 1.3.2, 1.3.4, and 1.3.5) is weakly compatible splits. In Section 3.1 we describe gains in efficiency that can be made when the set of input splits $\mathcal{S}$ is weakly compatible. In the case that $\mathcal{S}$ is a maximal collection of weakly compatible splits, and the quartet weights are nonnegative, we can solve the Split Constrained Quartet Optimization problem without having to apply a degree bound (Section 3.2).

We conclude with two complexity results. We show that the results in Section 3.2 for maximal collections of weakly compatible splits cannot be extended to arbitrary collections of weakly compatible splits (unless $P = NP$). Then we prove the rather surprising result that if we allow negative quartet weights then the Split Constrained Quartet Optimization problem (with no degree bound) is NP-hard even when the set of splits $\mathcal{S}$ equals $splits(T)$ of some tree $T$.

### 3.1. *Quartet Optimization with Weakly Compatible Splits*

Let $\mathcal{S}$ be a collection of weakly compatible splits on $L$ and let $d$ be a degree bound. As in Section 2.4 we choose a leaf $x$ and construct

$$\mathcal{C} = \{A : A \mid B \in \mathcal{S}, x \in B\}. \tag{19}$$

Then $\mathcal{C}$ is a *weak hierarchy* [3], which means that for all $U, V, W \in \mathcal{C}$

$$U \cap V \cap W \in \{U \cap V, U \cap W, V \cap W\}. \tag{20}$$

Define a closure operator $\langle \cdot \rangle_{\mathcal{C}}$ on subsets of $L - \{x\}$ by

$$\langle A \rangle_{\mathcal{C}} = \bigcap_{C \in \mathcal{C} : A \subseteq C} C. \tag{21}$$

Weak hierarchies have the property that for every subset $A \subseteq L$, there is $a, a' \in A$ such that $\langle A \rangle_{\mathcal{C}} = \langle \{a, a'\} \rangle_{\mathcal{C}}$ [2]. We write $\langle a, a' \rangle_{\mathcal{C}}$ for $\langle \{a, a'\} \rangle_{\mathcal{C}}$ and construct a table mapping each pair of leaves $a, a'$ to the corresponding subset $\langle a, a' \rangle_{\mathcal{C}}$. The table can be constructed in $O(n^5)$ time using the property that $y \in \langle a, a' \rangle_{\mathcal{C}}$ if and only if there is no cluster $C \in \mathcal{C}$ with $a, a' \in C$ and $y \notin C$.

The first efficiency gain we make is to speed up the calculation of $W[C_i; a, b]$ (Eq. (16)). First a special case. Recall that a *chain* is a collection of clusters $\mathcal{A}$ such that $A, B \in \mathcal{A}$ implies $A \subseteq B$ or $B \subseteq A$.

LEMMA 3.1. *If $\mathcal{C}$ is a chain and $a$ and $b$ are any two leaves then we can compute $W[C_i; a, b]$ for all $C_i \in \mathcal{C}$ in $O(n^2)$ time.*

*Proof.* Suppose that $\mathscr{C} = \{C_1, C_2, \dots, C_k\}$ where $C_i \subseteq C_j$ or all $i \le j$. We calculate $W[C_1; a, b]$ directly. Given $W[C_i; a, b]$ we can calculate $W[C_{i+1}; a, b]$ using

$$W[C_{i+1}; a, b] = W[C_i; a, b] + \sum_{\substack{c, d \in C_{i+1} \\ \{c, d\} \not\subseteq C_i}} w(cd \mid ab). \tag{22}$$

The amortized complexity is then $O(n^2)$. ∎

When $\mathscr{C}$ is not a chain, which is usually the case, we can apply Lemma 3.1 by first partitioning $\mathscr{C}$ into chains. We use Dilworth's theorem [18] to show that when $\mathscr{C}$ is a weak hierarchy we can partition $\mathscr{C}$ into $O(n)$ chains. Recall that an *antichain* is a collection of clusters $\mathscr{A}$ such that $A, B \in \mathscr{A}$ implies $A \not\subseteq B$ and $B \not\subseteq A$.

LEMMA 3.2. *Suppose that $\mathscr{C} = \{A : A \mid B \in \mathscr{S}, x \in B\}$ for some collection $\mathscr{S}$ of weakly compatible splits on a set $X$ of $n$ elements.*

1. *For each element $a \in X - \{x\}$ there are no three elements $a_1, a_2, a_3$ such that $\langle a, a_1 \rangle_{\mathscr{C}}$, $\langle a, a_2 \rangle_{\mathscr{C}}$, and $\langle a, a_3 \rangle_{\mathscr{C}}$ form an antichain in $\mathscr{C}$.*
2. *$\mathscr{C}$ can be partitioned into $n - 1$ chains.*
3. *We can partition $\mathscr{C}$ into $O(n)$ chains in $O(n^3)$ time.*

*Proof.* (1) Suppose there was such an antichain. Then $a_i \notin \langle a, a_j \rangle_{\mathscr{C}}$ for all $i \neq j$. Put $A_i = \langle a, a_i \rangle_{\mathscr{C}}$ for $i = 1, 2, 3$, and $B_i = X - A_i$. Then each $A_i \mid B_i$ is a split in $\mathscr{S}$. Furthermore we have

$$x \in B_1 \cap B_2 \cap B_3 \tag{23}$$

$$a_3 \in B_1 \cap B_2 \cap A_3 \tag{24}$$

$$a_2 \in B_1 \cap A_2 \cap B_3 \tag{25}$$

$$a_1 \in A_1 \cap B_2 \cap B_3 \tag{26}$$

which contradicts the weak compatibility of $\mathscr{S}$ (see Section 1.1).

(2) Let $\mathscr{A} = A_1, A_2, \dots, A_k$ be a maximum cardinality antichain in $\mathscr{C}$. Let $Y$ be the set of elements $y$ such that $\langle y, y \rangle_{\mathscr{C}} \in \mathscr{A}$. Let $Z$ be the set of elements $z$ for which there exists $z'$ such that $\langle z, z' \rangle_{\mathscr{C}} \in \mathscr{A}$. Since $\langle y, y \rangle_{\mathscr{C}} \subseteq \langle y, z \rangle_{\mathscr{C}}$ for all $z$ and $\mathscr{A}$ is an antichain we must have that $Y$ and $Z$ are disjoint.

The number of clusters $A_i \in \mathscr{A}$ such that $A = \langle y, y \rangle_{\mathscr{C}}$ for some $y \in Y$ is bounded above by $|Y|$. For the remaining clusters in $\mathscr{A}$ there are at least two elements $a, b \in Z$ such that $A = \langle a, a' \rangle_{\mathscr{C}}$ for some $a'$ and $A = \langle b, b' \rangle_{\mathscr{C}}$ for some $b'$. By (1) for each element $z$ in $Z$ there are at most two clusters $A_i, A_j \in \mathscr{A}$ such that $A_i = \langle z, z_i \rangle_{\mathscr{C}}$ for some $z_i$ and $A_j = \langle z, z_j \rangle_{\mathscr{C}}$ for some $z_j$. Hence the number of clusters that do not equal $\langle y, y \rangle_{\mathscr{C}}$ for some $y \in Y$ is bounded above by $|Z|$.

Therefore $|\mathscr{A}| \leq |Y| + |Z| \leq |X - \{x\}| = n - 1$. The maximum size antichain contains at most $n - 1$ clusters, so by Dilworth's theorem [18] the collection $\mathscr{C}$ can be covered by $n - 1$ chains.

(3) Since Dilworth's theorem is nonconstructive it does not guarantee an efficient algorithm for constructing the covering. Instead we use (1) again. For each $a \in X - \{x\}$ put

$$\mathscr{C}_a = \{A \in \mathscr{C} : A = \langle a, a' \rangle_{\mathscr{C}} \text{ for some } a'\}. \tag{27}$$

Then by (1) we have that $\mathscr{C}_a$ has a maximal antichain of size two and can hence be decomposed into two chains. It takes $O(n^2)$ time to decompose $\mathscr{C}_a$ into two chains: first construct an incomparability graph for the clusters (noting that $\langle a, a_1 \rangle_{\mathscr{C}} \subseteq \langle a, a_2 \rangle_{\mathscr{C}}$ if and only if $a_1 \in \langle a, a_2 \rangle_{\mathscr{C}}$) and then 2-coloring. Repeating the process for all $a$ gives a partition into at most $2(n - 1)$ chains in $O(n^3)$ time. ∎

Note that the bound of $n - 1$ of Lemma 3.2(2) is obtained in the case that $\mathscr{S}$ is maximum weakly compatible.

We now focus our attention on the complete decomposition table for $\mathscr{C}$.

LEMMA 3.3.

1. *If $[p_1, p_2]$ is a tuple in a decomposition table for $\mathscr{C}$ then there is $a_1, a_2, a_3$ such that $C_{p_1} = \langle a_1, a_2 \rangle_{\mathscr{C}}$ and $C_{p_2} = \langle a_1, a_3 \rangle_{\mathscr{C}} - \langle a_1, a_2 \rangle_{\mathscr{C}}$.*

2. *If $[p_1, p_2, p_3]$ is a tuple in a decomposition table for $\mathscr{C}$ then there is $a_1, a_2, a_3, a_4$ such that $C_{p_1} = \langle a_1, a_2 \rangle_{\mathscr{C}}$, $C_{p_2} = \langle a_3, a_4 \rangle_{\mathscr{C}}$,*

$$C_{p_3} = \langle a_1, a_3 \rangle_{\mathscr{C}} - \langle a_1, a_2 \rangle_{\mathscr{C}} - \langle a_3, a_4 \rangle_{\mathscr{C}}. \tag{28}$$

*Proof.*

(1) There is $y_1, y_1'$ and $y_2, y_2'$ such that $C_{p_1} = \langle y_1, y_1' \rangle_{\mathscr{C}}$ and $C_{p_2} = \langle y_2, y_2' \rangle_{\mathscr{C}}$. Hence $C_{p_1} \cup C_{p_2} = \langle \{y_1, y_1', y_2, y_2' \rangle_{\mathscr{C}}$. Thus there is $a_1 \in \{y_1, y_1'\}$ and $a_3 \in \{y_2, y_2'\}$ such that $C_{p_1} \cup C_{p_2} = \langle a_1, a_3 \rangle_{\mathscr{C}}$. We can then let $a_2$ equal the element in $\{y_1, y_1'\} - \{a_1\}$.

(2) This time we choose $y_i, y_i'$ for $i = 1, 2, 3$ such that $C_{p_i} = \langle y_i, y_i' \rangle_{\mathscr{C}}$. Then $C_{p_1} \cup C_{p_2} \cup C_{p_3} = \langle y_1, y_2, y_3, y_1', y_2', y_3' \rangle_{\mathscr{C}}$. There is $a_1, a_3 \in \{y_1, y_2, y_3, y_1', y_2', y_3'\}$ such that $\langle a_1, a_3 \rangle = C_{p_1} \cup C_{p_2} \cup C_{p_3}$. We then choose $a_2$ and $a_4$ from $\{y_1, y_2, y_3, y_1', y_2', y_3'\} - \{a_1, a_3\}$ so that $\langle a_1, a_2 \rangle_{\mathscr{C}}$ and $\langle a_3, a_4 \rangle_{\mathscr{C}}$ equal two of $C_{p_1}, C_{p_2}, C_{p_3}$. The result follows. ∎

We now have the tools we need to derive the efficient algorithm.

THEOREM 3.1. *If $\mathcal{S}$ is a set of weakly compatible splits and $d$ equals 3 or 4 then split constrained quartet optimization can be solved in $O(n^{d+2})$ time.*

*Proof.* Fix $x$ and construct $\mathcal{C} = \{A : A \mid B \in \mathcal{S}, x \in B\}$ and the table containing $\langle a, a' \rangle_{\mathcal{C}}$ for each $a, a'$. This takes $O(n^5)$ time. By Lemma 3.2 we can partition $\mathcal{C}$ into $O(n)$ chains in $O(n^3)$ time. Applying Lemma 3.1 for each chain and each pair of leaves $a, b$ we can calculate the values $W[C_i; a, b]$ for all $a, b$ and all $C_i \in \mathcal{C}$ in $O(n^5)$ time.

By Lemma 3.3, since $d$ equals 3 or 4 there are at most $O(n^d)$ tuples in the decomposition table, so $\|D\|$ is $O(n^d)$ and the complete decomposition table can be constructed in $O(n^{d+1})$ time. We can now apply Algorithm 1 to obtain the result. ∎

We conjecture that Theorem 3.1 can be extended for larger values of $d$, though we suspect that a different proof technique is required. In any case, the complexity of $O(n^6)$ when $d = 4$ is about the limit of a practical algorithm.

### 3.2. *Maximum Weakly Compatible Splits*

The maximum cardinality of a collection of weakly compatible splits on a set of $n$ elements is $\binom{n}{2}$. Those collections $\mathcal{S}$ for which $|\mathcal{S}| = \binom{n}{2}$ are called *maximum weakly compatible*. These collections have a special structure, allowing them to be represented in terms of cuts in a circle [3] or as a planar *splitsgraph* [21]. For every tree $T$ there is a maximum weakly compatible set containing *splits*$(T)$. In many ways, these collections of splits fall between trees and weakly compatible splits in terms of generality and complexity.

Here we show that the structural properties of maximum weakly compatible splits allows us to solve split constrained quartet optimization in polynomial time *without* a degree bound. The key result is

LEMMA 3.4. *Let $\mathcal{S}$ be a maximum weakly compatible set of splits and let $T$ be a tree such that splits$(T) \subseteq \mathcal{S}$. Then there is a binary tree $T'$ such that splits$(T) \subseteq$ splits$(T') \subseteq \mathcal{S}$.*

*Proof.* By Theorem 5 of [3] we can order the leaf set as $x_0, x_1, \ldots, x_{n-1}$ such that for every split $A \mid B$ with $x_0 \in B$ we have $A \mid B \in \mathcal{S}$ if and only if there is $i, j$ for which

$$A = \{x_i, x_{i+1}, \ldots, x_j\}. \tag{29}$$

That is, if and only if $A$ is an *interval* with respect to the ordering $x_1, \ldots, x_{n-1}$.

We proceed by induction. Let $d$ be the maximum degree of any vertex in $T$. The result holds, trivially, if $d = 3$. Suppose that the result holds for all $T$ with maximum degree less than $d$ and that $T$ is a tree with maximum degree $d$.

Let $T_0$ be the rooted tree with leaf set $L - \{x_0\}$ such that UNROOT$(T_0, L) = T$. Let $v$ be a vertex of $T$ with degree $d$. The corresponding vertex $v_0$ in $T_0$ has $d - 1$ children.

There is $a_1 < a_2 < \cdots < a_d$ such that the cluster sets corresponding to the children to $v_0$ equal

$$\left\{\{x_i : a_j \leq i < a_{j+1}\} : j = 1, 2, \ldots, d - 1\right\}. \tag{30}$$

If we insert the cluster $\{x_i : a_i \leq i \leq a_3\}$ into $T_0$, and the corresponding split into $T$, then $v$ will have degree $d - 1$ and $T$ will still have splits contained in $\mathcal{S}$. We repeat the process to obtain a tree that contains all the splits of the original tree, has splits contained in $\mathcal{S}$, and has maximum degree $d - 1$. The result follows from the inclusion hypothesis. ∎

Suppose now that all quartet weights are nonnegative. If $\mathcal{S}$ is a maximum weakly compatible collection of splits and $T$ is a nonbinary tree such that $splits(T) \subseteq \mathcal{S}$ then by Lemma 3.4 there is binary $T'$ such that $splits(T) \subseteq splits(T')$. Furthermore, $q(T) \subseteq q(T')$ and since all quartets have nonnegative weight we have $w(T) \leq w(T')$. Hence we can find a tree with optimal weight and splits in $\mathcal{S}$ by searching through the just binary trees with splits in $\mathcal{S}$. By Theorem 3.1 we now have

THEOREM 3.2. *Let $\mathcal{S}$ be a maximum weakly compatible set of splits of L and let w be a nonnegative weighting for quartets of L. We can find a tree T with splits $(T) \subseteq \mathcal{S}$ and maximum quartet weight in $O(n^5)$ time.*

Note that if we drop the nonnegativity constraint then the problem becomes NP-hard (Theorem 3.4).

### 3.3. *Complexity Results*

We conclude with two complexity results, showing that the polynomial time results are, in a sense, tight. First we consider the case when $\mathcal{S}$ is weakly compatible and all quartets have nonnegative weight.

THEOREM 3.3. *Split constrained quartet optimization is NP-complete when d is unbounded, even when all quartet weights are nonnegative and $\mathcal{S}$ is weakly compatible.*

*Proof.* The problem is clearly in NP.

We provide a reduction from the problem of determining a maximum compatible subset of a set of weakly compatible splits, which was shown to be NP-complete in [12].

For every split $A_i \mid B_i \in \mathscr{S}$ there exists a quartet $a_i a_i' \mid b_i b_i' \in q(A_i \mid B_i)$ such that $a_i a_i' \mid b_i b_i' \notin q(A_j \mid B_j)$ for all other splits $A_j \mid B_j \in \mathscr{S}$ [3, 4]. Choose one of these quartets for each split and give it weight one. Give all other quartets weight zero. Then for any tree $T$ with leaf set $L$ and $q(T)$ we have

$$\sum_{ab|cd \in q(T)} w(ab \mid cd) = |\,splits(T)\,|. \tag{31}$$

Hence the weight of the optimal weight tree equals the size of the maximum compatible subset of $\mathscr{S}$. ∎

Our second complexity result rules out the possibility of an extension of Theorem 3.2 to include negative quartet weights.

THEOREM 3.4. *Split constrained quartet optimization is NP-complete when $d$ is unbounded and some quartet weights are negative, even when $\mathscr{S}$-splits$(T)$ for some tree $T$.*

*Proof.* The problem is clearly in NP.

We provide a reduction from Vertex cover. Let $G$ be a graph with vertex set $V$ and edge set $E$. Put $M = 2|V|$. Put $L = \{v', v'' : v \in V\}$ and let $T$ be the tree only containing clusters $\{v', v''\}$ and one central vertex $x$ of degree $|V|$.

Label the internal vertex adjacent to $v'$ and $v''$ by $v$. Let $Q_E$ be the set of quartets

$$Q_E = \{u'u'' \mid v'v'' : \{u, v\} \in E\} \tag{32}$$

and let $Q_V$ be the set of quartets

$$Q_V = \{v'v'' \mid w'x' : v, w, x \in V\}. \tag{33}$$

We give each quartet in $Q_E$ weight $M$ and each quartet $Q_V$ weight $\frac{-2}{(n-2)(n-3)}$, where $n = |V|$.

Suppose that $V'$ is a vertex cover for $G$ with size $k$. Construct $T'$ with split set $\{\{v', v''\} \mid (L - \{v', v''\}) : v \in V'\}$. Then $q(T')$ contains all quartets in $Q_1$ and exactly those quartets in $Q_V$ of the form $v'v'' \mid w'x'$ for some $u \in V$. It follows that $T'$ has summed quartet weight $kM - k$.

Conversely, if $T'$ has summed quartet weight $kM - k$ then $T'$ must contain all quartets in $Q_E$ and only $k\frac{(n-2)(n-3)}{2}$ quartets in $Q_V$. We can then construct a vertex cover

$$V' = \{v : \{v', v''\} \mid (L - \{v', v''\}) \in splits(T')\} \tag{34}$$

for $G$ of size $k$. ∎

## ACKNOWLEDGMENTS

This work was carried out while D. Bryant held a Bioinformatics Postdoctoral Fellowship from the Canadian Institute for Advanced Research, Evolutionary Biology Program. Research supported in part by the Natural Sciences and Engineering Research Council of Canada and the Canadian Genome Analysis and Technology grants to D. Sankoff. The second author thanks the New Zealand Marsden Fund. We also thank C. Semple and V. Berry for help with proofreading.

## REFERENCES

1. H.-J. Bandelt and A. Dress, Reconstructing the shape of a tree from observed dissimilarity data, *Adv. Appl. Math.* **7** (1986), 309–343.
2. H.-J. Bandelt and A. Dress, Weak hierarchies associated with similarity measures—an additive clustering technique, *Bull. Math. Biol.* **51** (1989), 133–166.
3. H.-J. Bandelt and A. Dress, A canonical decomposition theory for metrics on a finite set, *Adv. Math.* **92** (1992), 47–105.
4. H.-J. Bandelt and A. Dress, "A Relational Approach to Split Decomposition," Technical report, Universität Bielefeld, 1994.
5. M. Bellare, O. Goldreich, and M. Sudan, Free bits, PCPs, and non-approximability—towards tight results, *in* "Proc. 36th Ann. Symp. Found. Comp. Sci. (FOCS)," pp. 422–431, 1995.
6. A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg, From four-taxon trees to phylogenies: the case of mammalian evolution, *in* "Proc. 2nd Annual Int. Conf. Comp. Mol. Biol. (RECOMB)," pp. 9–19, 1998.
7. V. Berry, "Méthodes et algorithmes pour reconstruire les arbres de l'Évolution," Ph.D. thesis, Université Montpellier II, 1997.
8. V. Berry and D. Bryant, Faster reliable phylogenetic analysis, *in* "Proc. 3rd Annual Int. Conf. Comp. Comp. Mol. Biol. (RECOMB)," pp. 59–68, 1999.
9. V. Berry, D. Bryant, P. Kearney, M. Li, T. Jiang, T. Wareham and H. Zhang, A practical algorithm for recovering the best supported edges in an evolutionary tree. Manuscript (1999).
10. V. Berry and O. Gascuel, Inferring evolutionary trees with strong combinatorial evidence, *Theoret. Comput. Sci.* **240**(2) (2000), 271–298.
11. D. Bryant, Hunting for trees in binary character sets: efficient algorithms for extraction, enumeration, and optimization, *J. Comput. Biol.* **3** (1995), 275–288.
12. D. Bryant, "Building Trees, Hunting for Trees and Comparing Trees," Ph.D. thesis, University of Canterbury, NZ, 1997.
13. D. Bryant and M. Steel, Extension operations on sets of leaf-labelled trees, *Adv. Appl. Math.* **16** (1995), 425–453.
14. N. Colonius and H. H. Schulze, Tree structures for proximity data, *British J. Math. Statist. Psych.* **34** (1981), 167–180.
15. M. Csüros and M. Kao, Recovering evolutionary trees through harmonic greedy triplets, *in* "Proc. SODA," pp. 261–270, 1999.
16. W. H. E. Day and D. Sankoff, Computational complexity of inferring phylogenies by compatibility, *Systems Zool.* **35** (1986), 224–229.
17. M. C. H. Dekker, "Reconstruction Methods for Derivation Trees," Master's thesis, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1986.

18. R. Dilworth, A decomposition theorem for partially ordered sets, *Ann. of Math.* **51** (1950), 161–165.
19. R. Downey and M. Fellows, Fixed-parameter tractability and completeness II, On completeness for W [1], *Theoret. Comput. Sci.* **141** (1996), 109–131.
20. A. Dress, Towards a theory of holistic clustering, *in* "Mathematical Hierarchies and Biology," pp. 271–289, Amer. Math. Soc., Providence, 1997.
21. A. Dress and D. Huson, Computing phylogenetic networks from split systems, manuscript, 1998.
22. A. Dress and M. Steel, Convex tree realizations of partitions, *Math. Lett.* **5** (1992), 3–6.
23. P. L. Erdös, M. Steel, L. Székeley, and T. Warnow, A few logs suffice to build (almost) all trees (1), *Random Structures Algorithms* **14** (1999), 153–184.
24. J. Farris, Estimating phylogenetic trees from distance matrices, *Amer. Natur.* **106** (951) (1972), 645–667.
25. M. Fellows, personal communication, 1997.
26. W. M. Fitch, A non-sequential method of constructing trees and hierarchical classifications, *J. Mol. Evol.* **18** (1981), 30–37.
27. M. Golumbic, "Algorithmic Graph Theory and Perfect Graphs," Academic Press, New York, 1980.
28. A. Gordon, Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labelled leaves, *J. Classification* **3** (1986), 335–348.
29. D. H. Huson, S. Nettles, and T. Warnow, Obtaining highly accurate topology estimates of evolutionary trees from very short sequences, *in* "Proc. 3rd Annual Int. Conf. Comp. Mol. Biol. (RECOMB)," pp. 198–209, 1999.
30. T. Jiang, P. Kearney, and M. Li, Orchestrating quartets: approximation and data correction, *in* "Proc. 39th FOCS," pp. 416–425, 1998.
31. P. Kearney, The ordinal quartet method, *in* "Proc. 2nd Annual Int. Conf. Comp. Mol. Biol. (RECOMB)," pp. 125–134, 1998.
32. C. Korostensky and G. Gonnet, "Using Travelling Salesman Problem Algorithms for Evolutionary Tree Construction," Technical Report 322, ETH, Zurich, 1999.
33. V. Makarenkov and B. Leclerc, Circular orders of tree metrics, and their uses for the reconstruction and fitting of phylogenetic trees, *in* "Mathematical Hierarchies and Biology," pp. 183–208, Amer. Math. Soc., Providence, 1997.
34. F. R. McMorris, D. B. Meronk, and D. A. Neumann, A view of some consensus methods for trees, *in* "Numerical Taxonomy" (J. Felsenstein, Ed.), pp. 122–125, Springer-Verlag, Berlin/New York, 1983.
35. S. Sattath and A. Tversky, Additive similarity trees, *Psychometrika* **42** (1977), 319–345.
36. M. Steel, The complexity of reconstructing trees from qualitative characters and subtrees, *J. Classification* **9** (1992), 91–116.
37. K. Strimmer and A. von Haeseler, Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies, *Mol. Biol. Evol.* **13** (1996), 964–969.
38. S. Willson, Measuring inconsistency in phylogenetic trees, *J. Theoret. Biol.* **190** (1998), 15–36.