JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 3, Number 2, 1996
Mary Ann Liebert, Inc.
Pp. 275–288

Hunting for Trees in Binary Character Sets: Efficient Algorithms for Extraction, Enumeration, and Optimization

DAVID BRYANT

ABSTRACT

We describe a useful and efficient tree building technique that is closely related to the maximum compatible subset method. Given a set of binary characters C and a degree bound d, the algorithm we describe counts the number of trees T that satisfy (1) the edges of T correspond to characters contained in C, and (2) the vertices of T have degree bounded by d. If the characters are weighted then we can efficiently determine which of these trees have maximum summed edge weight, where the weight of an edge equals the weight of the corresponding binary character in C. The complexity of the algorithms equals $O(nk + ndK^{(d-1)})$, where n is the number of taxa, k is the number of characters, and K is the number of distinct characters. A number of new tree consensus methods based on the algorithm are introduced, including one that uses edge weighted trees. As well, we illustrate the applicability of the algorithm for large sequence data sets by analyzing sequences from the "Out of Africa" mtDNA data set.

Key words: character compatibility, phylogenetic trees, phylogenetic inference, consensus methods.

1. INTRODUCTION

COMPATIBILITY METHODS are employed throughout systematics. They appear under a number of guises, in cladistics (Wiley *et al.*, 1991), numerical taxonomy (LeQuesne, 1969), evolutionary systematics (Eldredge and Cracraft, 1980), and molecular biology (Kannan and Warnow, 1994). One common method is to infer phylogenies by finding those phylogenies compatible with the largest number of characters, or, if the characters are weighted, finding the phylogenies compatible with the set of characters of largest combined weight. Unfortunately this general problem is NP-hard,[1] even for binary characters (Day and Sankoff, 1986). Consequently researchers have used the method only on small data sets, or have settled for heuristic algorithms that do not guarantee the best solution. A number of related algorithms and programs are available (see Meacham and Estabrook, 1985). Our approach is to slightly modify the problem, giving a compatibility method that is useful, nontrivial, and has a polynomial time algorithm.

Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand.

[1]A polynomial time solution to an NP-hard problem would give polynomial time solutions to all NP-complete problems. This means, on a practical level, that it is highly unlikely to have an efficient solution (Gary and Johnson, 1979).

We begin with a number of definitions. A *binary character* is a function from the set of taxa to the set $\{0, 1\}$, usually representing some property that distinguishes one group of taxa from the others. Binary characters can be obtained, for example, from genetic data as in Section 5, or from morphological distinctions like vertebrate/invertebrate. Each binary character induces a unique *split* of the set of taxa, that is, a partition into two groups comprising those assigned a 1 and those assigned a 0. We represent a split by an unordered pair $\{A, A'\}$, where A and A' are disjoint sets whose union is the set of taxa. Sets of binary characters or splits are one of the most common starting points for phylogenetic analysis.

An (unrooted) *phylogenetic tree* is a tree with no vertices of degree two, and with leaves labelled bijectively from the set of taxa. None of the internal vertices is labelled. In a *d-tree* every vertex has degree d or less. The split $\{A, A'\}$ *fits* a phylogenetic tree T if we can remove an edge of T to give two subtrees, one with leaf set A and the other with leaf set A'. The *set of splits of a tree* T is the set of splits that fit T and is denoted $\sigma(T)$. A set of splits C is *compatible* if there exists a tree T such that $C \subseteq \sigma(T)$. A set of binary characters is compatible if the induced set of splits is compatible. The *character encoding* $C(T)$ of a tree is the set of binary characters corresponding to $\sigma(T)$.

Let C be a set of binary characters on leaf set L. Put $k = |C|$ and $n = |L|$. Let K be the number of *distinct* characters in C. The main results of this paper are

1. An $O(nk + ndK^{(d-1)})$ algorithm that determines whether there exist d-trees T such that $C(T) \subseteq T$. If so, then one of these trees can then be retrieved in $O(n|C(T)|)$ time.
2. An $O(K^{d-1})$ algorithm that, when the binary characters in C are weighted, determines which d-trees T with $C(T) \subseteq C$ have maximum summed weight.
3. An $O(nk + ndK^{d-1})$ algorithm that determines the smallest d such that a d-tree T with $C(T) \subseteq T$ exists.
4. Efficient algorithms for a variety of consensus methods based on otherwise NP-hard problems. We include a consensus method for trees with weighted edges.
5. A tree construction algorithm that can be used on large sequence data sets.

In contrast to the general trend in the field, our algorithms and associated verifications are simple and nontechnical. Despite their simplicity the algorithms are powerful, as demonstrated by the applications to consensus trees and the analysis of 30 sequences derived from the mtDNA "Out of Africa" data set.

It is important to realize that the optimization method we introduce is an exact method and not a heuristic. We can guarantee that there is no d-tree with greater summed weight and splits all supported by the data set, despite the fact that there can be a very large number of these d-trees. Since the algorithm has only polynomial time complexity it can be applied to large data sets. In such cases an exhaustive search of possible trees or possible subsets of the binary character sets is simply not feasible owing to the astronomically large numbers involved.

Phylogenetic analysis is an intrinsically difficult problem, so we expect every analysis technique to have a weakness. The current technique is no exception, the main difficulty being the restriction to d-trees. Therefore this technique is not suitable for data sets with large numbers of taxa and only short sequences. Note that the problem of determining the smallest d for which a given set of binary characters contains a d-tree is itself NP-hard (Theorem 4). However, as the number of characters grows it is reasonable to expect that every split in the "true" tree is represented in the data. Furthermore if we assume that the "true" tree is binary then choosing a reasonably sized d allows us to cover the majority of cases where some splits are missing from the data set.

2. A SPECIAL CASE: SETS OF CLUSTERS

We first present our algorithm not in terms of collections of binary characters but in terms of clusters and rooted phylogenetic trees, which we now define. A *cluster* is just a subset of the set of taxa. A *rooted phylogenetic tree* is defined in the same way as an (unrooted) phylogenetic tree, except that one internal vertex is distinguished and called the *root,* and this vertex may have degree two. Given two vertices u and v in a rooted phylogenetic tree we say that u is a *descendent* of v if the path from u to the root passes through v. A cluster A *fits* a rooted tree T if there is some internal vertex v such that A equals the set of

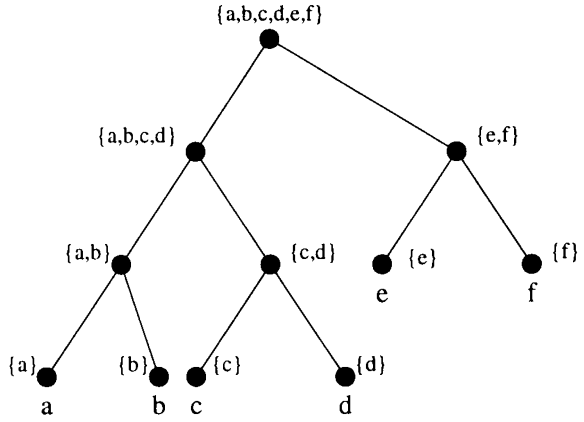

FIG. 1. A rooted binary phylogenetic tree with leaf set $\{a, b, c, d, e, f\}$.

leaves of T that are descendents of v. The *set of clusters of a tree* T equals the set of clusters that fit T. A rooted *d-tree* has no vertices with more than d children.

We can now state the problem formally.

INSTANCE: Collection C of subsets of a leaf set X such that $X \in C$. Number $d \geq 2$.

PROBLEM: Determine if there is a subcollection $C' \subseteq C$ such that C' equals the set of clusters of some rooted phylogenetic d-tree with leaf set X.

Our solution is based on the following:

Observation. *A given collection C of clusters contains a subcollection corresponding to a d-tree with leaf set X if and only if* $X \in C$ *and X equals the union of at most d disjoint clusters in C, which in turn equal the union of at most d disjoint clusters in C and so on, right down to the level of singletons.*

Consider the tree in Figure 1. The internal vertices are labelled with their respective clusters. The leaf set $\{a, b, c, d, e, f\}$ is the union of two disjoint clusters, $\{a, b, c, d\}$ and $\{e, f\}$, which, in turn, equal the union of two disjoint clusters and so on, until we reach the leaves of the tree.

2.1. *Principle algorithm*

Let k be the number of clusters in C and n be the number of taxon in X. We assume that C contains all the single element clusters. In practice these can simply be appended to the input data.

The following procedure creates a table D that is later used to count and optimize the d-trees with clusters in C and leaf set X.

Procedure CREATETABLE (C, d)

1. Represent each cluster in C by an n-digit binary number, with a 1 for the ith digit if the ith taxon is in the cluster, and a 0 otherwise (as in a characteristic vector).
2. Sort the clusters into ascending order of their associated binary numbers.
3. Remove duplicates and label remaining clusters $\{C_1, \ldots, C_K\}$.
4. Create a table D consisting of K empty lists, $D[1], \ldots, D[K]$.
5. Repeat for all subsets $\{p_1, p_2, \ldots, p_j\}$ of $\{1, \ldots, K\}$ with at most d elements:
6. If $C_{p_1}, \ldots, C_{p_j}$ are pairwise disjoint, and $C_{p_1} \cup \cdots \cup C_{p_j} = C_q$ for some $q \in \{1, \ldots, K\}$ then append $\{p_1, p_2, \ldots, p_j\}$ to the list $D[q]$
7. end (repeat)
8. end.

Steps 1, 2, and 3 can be completed in $O(nk)$ time using radix sort (Gusfield, 1991; Aho *et al.*, 1974), where n is the number of taxa and k is the number of clusters. For arbitrary $\{p_1, p_2, \ldots, p_j\}$, $(j \leq d)$, determining whether $C_{p_1}, \ldots, C_{p_j}$ are pairwise disjoint, and calculating $C_{p_1} \cup \cdots \cup C_{p_j}$ takes at most $O(nd)$ time. Since C is well-ordered it takes just $O(\log K)$ time to determine whether $C_{p_1} \cup \cdots \cup C_{p_j}$ is in C, which is just $O(n)$ time since $K \leq 2^n$. Hence steps 4 and 5 can be completed in $O(ndK^d)$ time, where K is the number of *distinct* clusters in C, and the procedure CREATETABLE takes at most $O(nk + ndK^d)$ time. Note that K is often significantly smaller than k.

2.2. Counting the trees

For each i, $1 \leq i \leq K$ we calculate the number $m[i]$ of d-trees with leaf set C_i and clusters contained in C.

```
Procedure CountTrees (D)
1. for i from 1 to K
2.     If C_i is a singleton then m[i]:= 1.
3.     else if D[i] is empty, then m[i]:= 0
4.     else
                m[i]:= Σ_{{p1,p2,...,pj}∈D[i]} m[p1] × m[p2] × ··· × m[pj].
5.     Remove from D[i] all tuples {p1,...,pj} such that one of m[p1],...,m[pj]
       equals 0.
6. end (for)
7. end.
```

Now each tuple $\{p_1, p_2, \ldots, p_j\}$ in $D[i]$ satisfies $C_{p_1} \cup \cdots \cup C_{p_j} = C_i$, so there are only $O(K^{d-1})$ such tuples in each $D[i]$. Hence calculating all the $m[i]$ takes $O(K^d)$ time.

Theorem 1. *After the completion of the above algorithm, $m[i]$ equals the number of distinct rooted d-trees with clusters in C and leaf set C_i.*

Proof. We proceed by induction. The theorem is clearly true when $i = 1$. Suppose that it is true when $i = 1, \ldots, j-1$. Let $\mathcal{T}_j$ be the set of rooted d-trees with leaf set C_j and clusters contained in C. We will show that $m[j] = |\mathcal{T}_j|$.

Let $T \in \mathcal{T}_j$, and let q be the number of children of the root of T. There is a set of q pairwise disjoint clusters $\{C_{p_1}, \ldots, C_{p_q}\}$ in T, and therefore also in C, such that $C_{p_1} \cup \cdots \cup C_{p_q} = C_j$, so the tuple $\{p_1, \ldots, p_q\}$ is contained in $D[j]$. It follows that we can partition $\mathcal{T}_j$ so that each block in the partition corresponds to a different tuple in $D[j]$. Hence

$$|\mathcal{T}_j| = \sum_{\{p_1,\ldots,p_q\} \in D[j]} (\#\text{ trees in } \mathcal{T}_j \text{ containing the clusters } C_{p_1}, \ldots, C_{p_q}).$$

Fix any such tuple $\{p_1, \ldots, p_q\} \in D[j]$. The number of trees in $\mathcal{T}_j$ that contain every cluster $C_{p_1}, \ldots, C_{p_q}$ equals the product $|\mathcal{T}_{p_1}| \times \cdots \times |\mathcal{T}_{p_q}|$. By the induction hypothesis this equals $m[p_1] \times \cdots \times m[p_q]$. Therefore,

$$\begin{aligned} |\mathcal{T}_j| &= \sum_{\{p_1,\ldots,p_q\} \in D[j]} m[p_1] \times m[p_2] \times \cdots \times m[p_q] \\ &= m[j] \end{aligned}$$

as required. □

2.3. Extracting the trees

It is now a simple matter to list the subcollections C' that correspond to d-trees. Given a collection of clusters C, we say that a particular cluster $C_i \in C$ is *unresolved* if there is no cluster $C_j \in C$ such that $C_j \subset C_i, C_j \neq C_i$. The following recursive procedure, GETSUBCOLLECTIONS, builds up a list of clusters.

At each level it chooses from the list an unresolved cluster C_i and adds to the list pairwise disjoint clusters $C_{p_1}, \ldots, C_{p_j}$ such that $C_{p_1} \cup \cdots \cup C_{p_j} = C_i$.

The first parameter is the list being built up, initially just $\{C_K\}$, and the second parameter is the table D constructed by the procedure CREATETABLE and pruned by procedure COUNTTREES.

```
Procedure GETSUBCOLLECTIONS (C', D)
1. if the only unresolved clusters in C' are singletons then
2.     output the subcollection C'
3. else
4.     choose any unresolved cluster C_i ∈ C' that is not a singleton
5.     for {p_1, ..., p_j} in D[i]
6.         GETSUBCOLLECTIONS (C' ∪ {C_p1, ..., C_pj}, D)
7.     end (for)
8. end (if)
9. end.
```

If C' is the set of clusters of some tree T, then the graph of T can be easily retrieved directly from C' in $O(n|C'|)$ time (Gusfield, 1991). Since it takes only $O(|C'|)$ time to construct C', to extract a single d-tree takes only $O(n|C'|)$ time.

Note that it is not always practical to list all of the subcollections of C that correspond to binary trees, as the following theorem illustrates.

Theorem 2. *Given a set X with n taxa, there exists a collection C with $(n^2+n)/2$ clusters such that the number of subcollections $C' \subseteq C$ that correspond to rooted binary trees ($d = 2$) with leaf set X exceeds 2^{n-2}.*

Proof. Given n, let $X = \{1, 2, \ldots, n\}$ and construct

$$C_{a,b} = \{x \in X : a \leq x \leq b\}$$

$$C = \{C_{a,b} : a \leq b, \qquad \text{where } a, b \in X\}$$

Then $|C| = (n^2+n)/2$. Let $\mathcal{T}_n$ be the set of binary trees with leaf sets X and clusters in C. We claim that $|\mathcal{T}_n| > 2^{n-2}$.

The result is easily verified for $n = 1, 2, 3$. For the induction step suppose that $|\mathcal{T}_k| > 2^{k-2}$. We show that $|\mathcal{T}_{k+1}| > 2^{(k+1)-2}$. Given any tree T in $\mathcal{T}_k$, we construct two new trees.

1. Create a new root with two descendents. Let one descendent be the root of T, and the other descendent be the leaf $(k+1)$.
2. Replace leaf k of T with a new vertex. Let the leaves k and $k+1$ be the two descendents of this new vertex.

Both of these trees are binary and have clusters in C. Hence for every tree in $\mathcal{T}_k$ there are at least two trees in $\mathcal{T}_{k+1}$, all of which are distinct. By the induction hypothesis

$$|\mathcal{T}_{k+1}| \geq 2 \times |\mathcal{T}_k| > 2 \times 2^{k-2} = 2^{(k+1)-2}$$

as required. □

2.4. *Optimization*

We can use the technique for counting trees to solve the following, seemingly more difficult, problem.

INSTANCE: Collection C of subsets of a finite set X. Weighting function $w : C \to \Re$. Number $d \geq 2$.

PROBLEM: Find a subcollection $C' \subseteq C$ that maximizes $\sum_{C_i \in C'} w(C_i)$ such that C' equals the set of clusters of some rooted d-tree.

The following procedure, MAXWEIGHTTREE, takes three parameters: The first parameter is the collection C; the second parameter is the table D constructed by the procedure CREATETABLE and pruned in COUNTTREES; the third is the table m constructed by the procedure COUNTTREES.

The procedure returns three tables, M, D^*, and m^*. For each i, $M[i]$ is the weight of the maximum weight subcollection that corresponds to a binary tree with leaf set C_i. The table D^* is a reduced version of D used to list the maximum weight subcollections, and the table m^* is used to count the number of maximum weight subcollections.

```
Procedure MAXWEIGHTTREE (C, D, m)

 1. for i = 1 to K
 2.     if m[i] ≠ 0 then
 3.         if C_i is a singleton then
 4.             M[i]: = w(C_i)
 5.             D*[i]: = {}
 6.             m*[i]: = 1
 7.         else
 8.             choose {p_1, ..., p_j} in D[i] that maximizes M[p_1] + ··· + M[p_j]
 9.             M[i]: = M[p_1] + ··· + M[p_j] + w(C_i)
10.             D*[i]: = {{q_1, ..., q_r} : M[q_1] + ··· + M[q_r] + w(C_i) = M[i]}
11.             m*[i]: = Σ_{(q_1,...,q_r)∈D*[i]} m*[q_1] × ··· × m*[q_r]
12.         end (if)
13.     end (if)
14. end (for)
15. end.
```

The procedure has complexity $O(K^d)$, where K is the number of distinct clusters in C. The number of optimal subcollections equals $m^*[K]$. To list all the optimal subcollections, execute GETSUBCOLLECTIONS $(\{C_K\}, D^*)$. There are sometimes, however, an exponentially large number of optimal subcollections. Correctness is given by the following theorem.

Theorem 3. *After execution of the procedure* MaxWeightTree, *if* $m[i] \neq 0$ *then* $M[i]$ *equals the weight of the maximum weight d-tree with leaf set* C_i *and clusters in* C.

Proof. Again we let $\mathcal{T}_i$ denote the set of d-trees with leaf set C_i and clusters in C. Let $w(T)$ denote the sum of the weights of the clusters of a phylogenetic tree T. We need to prove two things:

1. $w(T) \leq M[i]$ for all $T \in \mathcal{T}_i$.
2. There is $T \in \mathcal{T}_i$ such that $w(T) = M[i]$.

We proceed by induction. If $i = 1$ then C_i is a leaf, so (1) and (2) hold. Suppose that (1) and (2) are true for $i = 1, \ldots, j-1$.

Let $T \in \mathcal{T}_j$. There is $\{p_1, \ldots, p_q\} \in D[j]$ such that $C_{p_1}, \ldots, C_{p_q}$ are clusters of T. Now $C_{p_1} \cup \cdots \cup C_{p_q} = C_j$, so the q clusters correspond to the q subtrees branching off the root of T. Using the induction hypothesis applied to the subtrees of T, we obtain

$$w(T) = M[p_1] + \cdots + M[p_q] + w(C_j) \leq M[j]$$

proving (1).

For (2), let $\{p_1, \ldots, p_q\}$ be a tuple in $D[j]$ such that

$$M[p_1] + \cdots + M[p_q] + w(C_j) = M[j]$$

By the induction hypothesis for each $r : 1 \leq s \leq q$ there is a tree T_{p_s} with leaf set C_{p_s} and clusters in C such that $w(T_{p_s}) = M[p_s]$. Construct a new tree T by connecting a new root r to the roots of $T_{p_1}, \ldots, T_{p_q}$. Then $T \in \mathcal{T}_j$ and $w(T) = M[j]$. □

2.5. *Selecting a degree bound*

Given a set of clusters C we wish to determine the smallest value of d such that $C(T) \subseteq C$ for some d-tree T. Once this number is obtained we can guarantee that the above algorithms will return a tree. Unfortunately the following theorem suggests that there is no efficient way to determine this minimal d value.

Theorem 4. *Let C be a set of clusters. The problem of finding the smallest d such that C contains the clusters of a d-tree is NP-hard.*

Proof. We transform EXACT COVER BY 3-SETS (X3C) (Gary and Johnson, 1979) to the minimum d problem. Let $X, |X| = 3q$, be a set and C be a collection of 3-element subsets of X making up an arbitrary instance of X3C. Take X as a set of leaves and construct $C' = C \cup \{\{x\} : x \in X\} \cup \{X\}$. If X has an exact cover by a subcollection of C then the same subsets, together with X and the singleton clusters, determine a q-tree with clusters in C'. Conversely if C'' is a subcollection of C' that determines a q-tree then the nontrivial clusters in C'' give an exact covering of X. Therefore there is an exact covering of X by a subcollection of C if and only if the minimum d such that C' contains a d-tree equals q. It follows that the minimum d problem is NP-hard. □

Because of Theorem 4 we should expect any solution to the problem to have exponentially increasing complexity. We present one algorithm that has running time $O(nk + n\delta K^{\delta})$ where n is the number of leaves, k is the number of clusters, K is the number of distinct clusters, and δ is the minimum value of d such that C contains the splits of a d-tree. It therefore takes polynomial time with respect to the number of leaves and the number of characters, but takes exponentially increasing time with respect to the solution.

```
Procedure MINIMUMd (C)
1. Sort C and remove duplicates. Let K be the number of distinct clusters.
2. for d from 2 to n
3.     Construct D using CREATETABLE (C, d)
4.     Construct m using COUNTTREES (C)
5.     if m[K] > 0 then
6.         put δ = d
7.         exit the procedure.
8.     end (if)
9. end (for)
```

3. SETS OF BINARY CHARACTERS

We now return to considering sets of binary characters. Formally stated, our two main problems are

INSTANCE: Collection C of binary characters on leaf set X. Number $d \geq 3$.

PROBLEM: Determine if there is a subcollection $C' \subseteq C$ such that $C' = C(T)$ for some unrooted phylogenetic d-tree T.

INSTANCE: Collection C of binary characters on leaf set X. Weighting function $w : C \to \Re$. Number $d \geq 3$.

PROBLEM: Find a subcollection $C' \subseteq S$ that maximizes $\sum_{C_i \in C'} w(C_i)$ such that $C' = C(T)$ for some unrooted phylogenetic d-tree T.

Both of these problems can be transformed into cluster problems.
Consider the two trees T_1 and T_2 in Figure 2. Tree T_1 has splits

$$\{\{a, bcde\}, \{b, acde\}, \{c, abde\}, \{d, abce\}, \{e, abcd\}, \{ab, cde\}, \{de, abc\}\}$$

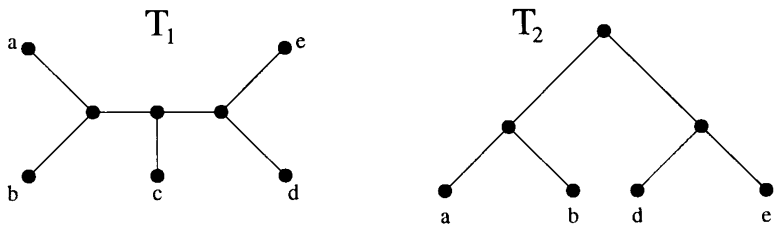

FIG. 2. An unrooted tree T_1 with corresponding rooted tree T_2.

The tree T_2 is obtained from the tree T_1 by rooting T_1 at the internal vertex adjacent to c, and then removing the leaf c together with its adjacent edge. Because of this relationship, it is possible to calculate the clusters of T_2 directly from the splits of T_1. For each split $\{A, A'\}$ of T_1, if $c \in A$, then A' is a cluster of T_2, otherwise A is a cluster of T_2. Hence the clusters of T_2 are $\{\{a\}, \{b\}, \{a, b, d, e\}, \{d\}, \{e\}, \{a, b\}, \{d, e\}\}$. We utilize this relationship to transform the above splits problems into cluster problems.

Theorem 5. *Let X be a set of taxa and $r \in X$. Let f_r be the mapping from binary characters on X to clusters of $X - \{r\}$ given by*

$$f_r(c) = \begin{cases} c^{-1}(1) \text{ when } c(r) = 0 \\ c^{-1}(0) \text{ when } c(r) = 1 \end{cases}$$

Then f_r is a one-to-one mapping with inverse

$$f_r^{-1}(A) = \text{character mapping } A \text{ to } 0 \text{ and } X - A \text{ to } 1$$

Given any collection C of binary characters on X, the set of clusters $f_r(C)$ equals the set of clusters of some rooted d-tree if and only if $C = C(T)$ for some unrooted $(d + 1)$-tree T.

Proof. It is straightforward to verify that f_r has the given inverse. We prove the second result.

Suppose that $C = C(T)$ for an unrooted tree T with leaf set X. Construct the rooted tree T_r from T by (1) making the vertex adjacent to the leaf r the root of T_r and (2) removing the leaf r and its adjacent edge (see Fig. 3).

Let e be the edge adjacent to the leaf r in T. The tree T_r equals the subtree of T rooted at the opposite end of e from r. Therefore a set A is a cluster of T_r if and only if $(A, X - A)$ is a split in T. Hence $f_r(C)$ is the set of clusters of some rooted tree. Moreover, if every vertex in T has valency of d or less, then every vertex in T_r has at most $d - 1$ children.

Conversely, if C equals the set of clusters of some rooted tree T_r, then construct T by attaching a leaf r and an edge to the root of T_r. Then $f_r^{-1}(C)$ will equal $C(T)$. □

It follows that the number of subcollections $C' \subseteq C$ that equal the set of binary characters of a unrooted phylogentic d-tree is the same as the number of subcollections $C' \subseteq f_r(S)$ that equal the set of clusters of a rooted phylogenetic $(d - 1)$-tree.

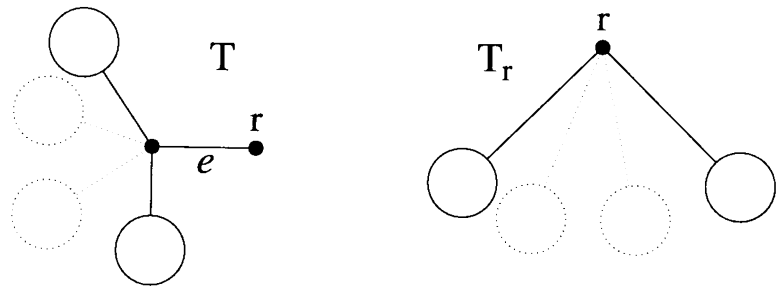

FIG. 3. The transformation from T, on the left, to T_r, on the right. The hollow circles represent subtrees.

Given a set C of k binary characters on a set X, choose arbitrary $r \in X$ and calculate $f_r(C)$. This can be doen in $O(nk)$ time. Apply the above cluster algorithms to $f_r(C)$ to count the total number of subcollections corresponding to $(d-1)$-trees. A particular subcollection $C' \subset f_r(C)$, such as one returned by the algorithm GETSUBCOLLECTIONS, can be transformed into the corresponding set of splits using f_c^{-1}.

If the original set C of binary characters was weighted, then weight the elements of $A \in f_r(C)$ by $w(A) = w[f_r^{-1}(A)]$. If C' is an optimal subcollection of $f_r(C)$ then $f_r^{-1}(C')$ will be an optimal subcollection of C.

4. APPLICATION TO CONSENSUS TREES

Our algorithm can be used as the basis of several consensus methods on profiles of trees. The first is derived from the median consensus tree method.

4.1. Restricted median d-tree

Given two trees T_1 and T_2 define

$$D(T_1, T_2) = |C(T_1) - C(T_2)| + |C(T_2) - C(T_1)|$$

The consensus problem we address is given a profile of trees $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ and a degree bound d, determine if a tree T exists such that

1. T is a d-tree and
2. $C(T) \subseteq \cup_i C(T_i)$

If so, then determine which such tree minimizes

$$D(T, \mathcal{T}) = \sum_i D(T, T_i)$$

This tree is called the *restricted median d-tree*.

Theorem 6. *The restricted median d-tree of a profile of trees can be constructed, or shown not to exist, in $O(n^2dk^{d-1})$ time, where $k = |\cup_i C(T_i)|$ and $n = |L|$.*

Proof. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ be a profile of trees on leaf set L. If T is any tree with leaf set L then

$$\begin{aligned} D(T, T_i) &= |C(T) - C(T_i)| + |C(T_i) - C(T)| \\ &= |C(T)| + |C(T_i)| - 2|C(T) \cap C(T_i)| \end{aligned}$$

It follows that

$$D(T, \mathcal{T}) = K|C(T)| + \sum_{i=1}^{K} |C(T_i)| - 2\sum_{i=1}^{K} |C(T) \cap C(T_i)|$$

Given a character c put $w(c) = 2|\{i : c \in C(T_i)\}| - K$. Then

$$D(T, \mathcal{T}) = \sum_{i=1}^{K} |C(T_i)| - \sum_{c \in C(T)} w(c)$$

Apply our algorithm to $\cup_i C(T_i)$. If the number of d-trees counted equals zero then there is no restricted median d-tree for this profile. If the count is nonzero then the d-tree T with maximum summed weight minimizes $D(T, \mathcal{T})$ and is the restricted median d-tree. The process therefore takes $O(n^2dk^{d-1})$ time. □

Note that if we allow the consensus tree to have splits not in any of the input trees then minimizing $D(T, \mathcal{T})$ is NP-hard (McMorris and Steel, 1994). On the other hand, if we just remove the requirement that

the vertex degree of the consensus tree is bounded then the resulting problem, called the median consensus tree problem, can be solved in polynomial time (Barthelemy and McMorris, 1986). The disadvantage of the standard median tree is that it tends to be relatively uninformative, usually having only a small number of edges (Phillips and Warnow, 1996).

4.2. Restricted assymetric median d-tree

Phillips and Warnow (1996) define an assymetric median tree, closely related to the Nelson consensus tree (Page, 1990), where the function to be minimized is

$$f(T, \mathcal{T}) = \sum_i |C(T_i) - C(T)|$$

Given a degree bound d we can determine the d-tree T such that $C(T) \subseteq \cup_i C(T_i)$, which minimizes $f(T, \mathcal{T})$, provided that such d-trees exist. This is called a *restricted assymetric median tree.*

Theorem 7. *Let C be a set of binary characters. For $c \in C$ put*

$$w(c) = |\{i : c \in C(T_i)\}|$$

If T is the d-tree with $C(T) \subseteq C$ that has maximum summed weight, then T is a restricted assymetric d-tree.

Proof.

$$\begin{aligned} f(T, \mathcal{T}) &= \sum_i |C(T_i) - C(T)| \\ &= \sum_i |C(T_i)| - \sum_i |C(T_i) \cap C(T)| \\ &= \sum_i |C(T_i)| - \sum_{c \in C(T)} w(c) \end{aligned}$$

The result follows. □

Therefore the restricted assymetric median d-tree can also be constructed in $O(n^2 d k^{d-1})$ time, where $k = |\cup_i C(T_i)|$ and $n = |L|$.

4.3. Maximum weight consensus d-tree

At present there are only a few consensus methods that work for trees with weighted edges. If a tree in the input profile has a heavily weighted edge then we would expect this edge to appear in the consensus tree. The following method satisfies this criterion and is efficiently implemented using our algorithm.

INSTANCE: Profile $\mathcal{T}$ of edge-weighted trees on leaf set L, and a degree bound d.

PROBLEM: Find a d-tree with maximum summed edge weights, where each edge is weighted by the sum of the corresponding edge weights in trees in $\mathcal{T}$ and each edge appears in at least one tree in $\mathcal{T}$, or show that no such tree exists.

A tree that is a solution to this problem is called a *maximum weight consensus d-tree.*

Theorem 8. *A maximum weight consensus d-tree for a profile of trees can be constructed, or shown not to exist, in $O(n^2 d k^{d-1})$ time, where $n = |L|$ and $k = |\cup_i C(T_i)|$.*

Proof. We apply our algorithms to $\cup_i C(T_i)$ where each character is weighted by

$$w(c) = \sum_{i : c \in C(T_i)} (\text{weight of edge corresponding to } c)$$ □

4.4. *Features common to consensus d-trees*

In general we cannot guarantee the existence of a restricted median d-tree or a maximum weight consensus d-tree. However, *if the input profile contains at least one d-tree then their existence is guaranteed.* For example, if the input profile contains at least one binary tree, as is often the case, then, for all $d > 2$ there exist restricted median d-trees, restricted asymmetric median d-trees, and, if the trees in the input profile have weighted edges, maximum weight consensus d-trees.

Another feature of all these consensus methods is that they return trees containing all splits of the strict consensus tree, which contains those splits common to all trees. They are therefore "Pareto," as described in McMorris and Powers (1993).

Theorem 9. *Given a profile $\mathcal{T}$ of trees on leaf set L, and a degree bound d, all restricted median d-trees for $\mathcal{T}$ and all restricted asymmetric median d-trees for $\mathcal{T}$ contain all the splits of the strict consensus tree of $\mathcal{T}$. Furthermore, if the edges of the trees in $\mathcal{T}$ all have positive weights, then all maximum weight consensus d-trees for $\mathcal{T}$ contain the splits of the strict consensus tree for $\mathcal{T}$.*

Proof. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ be a collection of trees on leaf set L. Choose any T such that $C(T) \in \cup_i C(T_i)$. Put $S = \cap_i C(T_i)$. We claim that $C(T) \cup S$ is compatible.

Consider any two splits $\{A, A'\}$ and $\{B, B'\}$ in $C(T) \cup S$. If $\{A, A'\} \in S$ and $\{B, B'\} \in S$ then $\{A, A'\}$ and $\{B, B'\}$ are clearly compatible. Likewise if both splits are splits of T then they are compatible. If one split is in S and the other split, say $\{B, B'\}$, is not in S then there is a tree $T_i \in \mathcal{T}$ containing $\{B, B'\}$ and also, by definition, the splits of S. Therefore the splits in S' are pairwise compatible and so $C(T) \cup S$ is compatible (Buneman, 1971).

Now let T be a restricted median d-tree for $\mathcal{T}$. If $S \not\subseteq C(T)$ then construct the tree T' containing the splits $C(T) \cup S$. Clearly T' is a d-tree and $D(T', \mathcal{T}) < D(T, \mathcal{T})$, a contradiction. Likewise for a restricted asymmetric consensus d-tree.

Suppose that the edges in the trees in $\mathcal{T}$ have positive weight, and let T be a maximum weight consensus d-tree for $\mathcal{T}$. If $S \not\subseteq C(T)$ then construct the tree T' containing the splits $C(T) \cup S$. Clearly T' is a d-tree and $w(T') > w(T)$. □

5. APPLICATION TO THE ANALYSIS OF LARGE DATA SETS

As an illustration, we apply the above algorithms to the 135 human mitochondrial sequences of Vigilant *et al.* (1991). It was originally argued that the data supported an African origin for *Homo sapiens*. The methodology of Viglant *et al.* (1991) was criticized by a number of people (Templeton, 1991; Hedges *et al.*, 1991; Maddison *et al.*, 1992) although Penny *et al.* (1995) subsequently found further support for the "Out of Africa" hypothesis. We are concerned here not with the position of the root, but with the branching structure of the phylogeny.

The above analyses of the human mitochondrial data all used parsimony criterion to assess trees. In contrast, we assess trees by the number of characters they use from the data set, where the search is restricted to those trees with bounded vertex degree and edges all supported by characters from the input. We make only a preliminary investigation; our main purpose is to demonstrate the potential of the algorithm.

It was observed in Penny *et al.* (1995) (see also Hedges *et al.*, 1991) that the most parsimonious trees found all shared three properties:

1. The 16 !Kung sequences together with the Naron sequence are grouped together.
2. Sequences 1–48, together with the Naron sequence, are grouped together.
3. All major lineages occur in Africa, but only a subset occurs in the rest of the world.

These observations are used in Penny *et al.* (1995) to reject the multiregional model of human development. We wish to examine if these properties hold for trees returned by our algorithm.

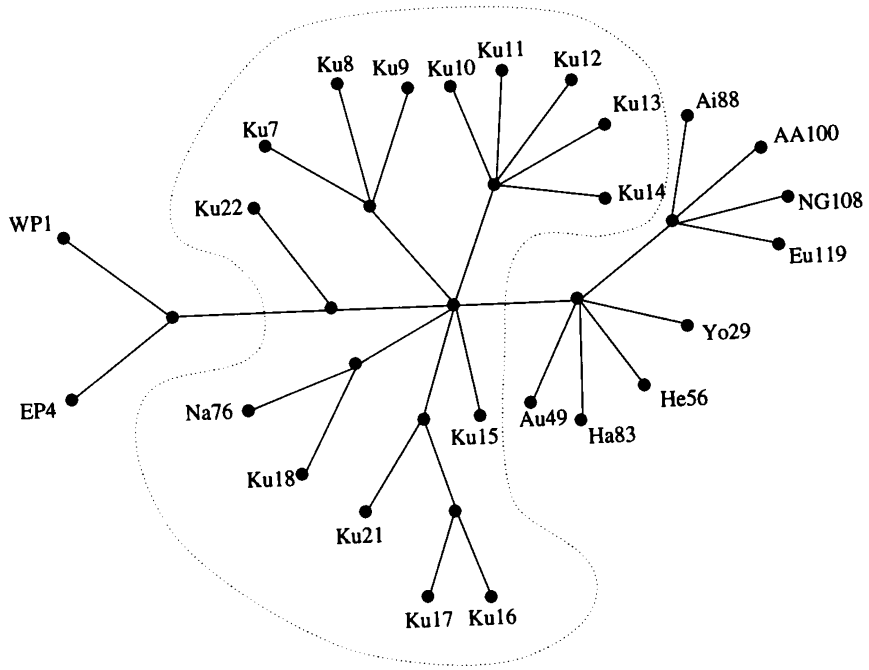

FIG. 4. The maximum weight tree for a set of mtDNA sequences containing all !Kung sequences, the Naron sequence, and one sequence from each of the remaining racial groups. The taxa are numbered as in Vigilant *et al.* (1991). The codes for racial groups are AA (Afroamerican), Ai (Asian), EP (Eastern Pygmy), Eu (European), Ha (Hadza), He (Herero), Ku (!Kung), Na (Naron), NG (New Guinean), WP (Western Pygmy), Yo (Yoruban).

The original data set contains 135 mtDNA sequences, each sequence being 131 sites long. We discarded those characters with missing entries. Appending the trivial splits, and discarding the nine characters with three or four states, we obtained a set of 192 distinct splits. It would be interesting in the future to employ more sophisticated methods of extracting binary characters from DNA sequences.

A binary tree with 135 leaves contains 267 splits, so the set could not possibly contain a set of characters that defines a binary tree. Indeed we would not even expect the data to contain all the splits of any tree with a small bound on the vertex degree. This was verified when the algorithm was run on the data set. There were no trees found, even when trees with maximum vertex degree seven were searched for, indicating that the data set does not readily support highly resolved trees. Longer sequences are required before our method can be used to analyze all 135 sequences at once. We must, therefore, restrict our attention to smaller subsets of taxa.

To examine whether trees returned by the algorithm satisfied property 1, we used a taxa set containing the 16 !Kung sequences, the Naron sequence, and one sequence from each of the remaining 10 racial groups, the sequences selected randomly within each group. If property 1 holds then we would expect the !Kung sequences to be grouped together in the maximum weight tree.

All of the characters in the reduced data set had two or fewer states. We discarded those characters with missing entries, giving a set of 108 characters, 56 of which were constant. Adding the 27 trivial characters, and removing duplicates, we obtained a set of 52 binary characters. The algorithm quickly found 18 different trees with maximum vertex degree eight and splits contained in this data set. We increased the degree bound to nine and the algorithm found a further 94 trees.

The 52 binary characters were then weighted with the number of times each character appeared in the original set of 108 characters. There was a unique maximum weight tree. When the constant characters were included, this tree contained 91 (out of 108) characters of the original set. We present this tree in Figure 4. The tree is only partially resolved, with a central vertex of degree nine. The !Kung sequences do not form a cluster in the tree but they do make up a central group.

We then chose a set of 32 sequences that were widely spread within each phylogeny to test whether the maximum weight trees satisfy properties 2 and 3. Once again characters with missing sites, or more than

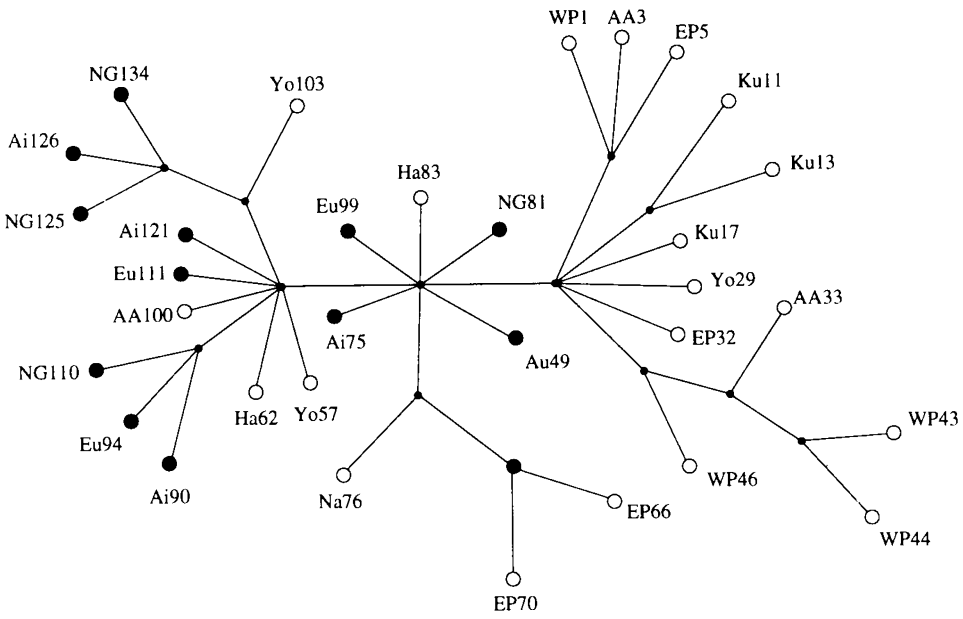

FIG. 5. The maximum weight tree for another set of mtDNA sequences. The taxa are numbered as in Vigilant *et al.* (1991), and african sequences are marked by hollow vertices. The codes for racial groups are AA (Afroamerican), Ai (Asian), EP (Eastern Pygmy), Eu (European), Ha (Hadza), He (Herero), Ku (!Kung), Na (Naron), NG (New Guinean), WP (Western Pygmy), Yo (Yoruban).

three states were discarded, the trivial characters added, and duplicates removed. This time the algorithm found no trees with maximum degree eight and splits in the data set. Examining the decomposition table and list of clusters revealed that trees would be found if the degree bound was increased by one. After several hours of processor time, the algorithm found 516 trees with maximum vertex degree nine and splits in the data set.

Again, there was a unique maximum weight tree. It was supported by 80 (out of 136) characters (Fig. 5). This is a considerably smaller proportion of the data set than for the previous example, indicating a greater level of uncertainty.

The sequences of this set with indices less than 49 formed a cluster in this tree, so the maximum weight tree is consistent with property 2 except for the placement of the Naron sequence (Na76). There is a split in the data set (44th site) separating Ku11, Ku13, Ku17, and Na76 from the other sequences, but there is no tree with maximum vertex degree nine and splits in the data set that contains this split. It is interesting to note that the Naron sequence appears in a variety of different positions in published phylogenies (see Hedges *et al.*, 1991; Templeton, 1991; Vigilant *et al.*, 1991).

We do not have information about where to position the root in the tree so we cannot tell what the major lineages are. However, one consequence of property 3 is that the tree should have a group comprised exclusively of African sequences, a group containing both African and non-African sequences, and no group containing exclusively non-African sequences. This is clearly satisfied by the maximum weight tree in Figure 5. Hence the maximum weight tree is consistent with property 3.

It is interesting to note the differences in structure between this tree and the phylogeny of Vigilant *et al.* (1991). The tree in Figure 5 bears closer resemblence to the more parsimonious phylogenies given in Hedges *et al.* (1991) and Penny *et al.* (1995).

There is considerable potential for further analysis of this data set using these methods. In particular, different character weightings and different techniques for extracting binary characters could be used. Unfortunately all analyses will be limited by the lack of resolution in the human mtDNA data set. It was this lack of resolution that forced us to use only some of sequences at a time. However, this problem is not unique to our approach, but rather is faced by all studies tackling the human mtDNA data set.

ACKNOWLEDGMENTS

I wish to thank Mike Steel and Tandy Warnow for their helpful suggestions and proofreading.

REFERENCES

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

Barthélemy, J.P., and McMorris, F.R. 1986. The median procedure for n-trees. *J. Classification* 3, 329–334.

Buneman, P. 1971. The recovery of trees from measures of dissimilarity, 387–395. *In* Hodson, F.R., Kendall, D.G., and Tautu, P., eds., *Mathematics in the Archeological and Historical Sciences,* Edinburgh University Press, Edinburgh.

Day, W.H.E., and Sankoff, D. 1986. Computational complexity of inferring phylogenies by compatibility. *Syst. Zool.* 35(2), 224–229.

Eldredge, N., and Cracraft, J. 1980. *Phylogenetic Patterns and the Evolutionary Process.* Columbia University Press, New York.

Gary, M.R., and Johnson, D.S. 1979. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman, San Francisco. CA.

Gusfield, D. 1991. Efficient algorithms for inferring evolutionary trees. *Networks* 21, 19–28.

Hedges, S.B., Kumar, S., and Tamura, K. 1991. Human origins and analysis of mitochondrial DNA sequences. *Science* 255, 737–739.

Kannan, S., and Warnow, T. 1994. Inferring evolutionary history from DNA sequences. *SIAM J. Comput.* 23(4), 713–737.

Le Quesne, W.J. 1969. A method of selection of characters in numerical taxonomy. *Syst. Zool.* 18, 201–205.

Maddison, D.R., Ruvolo, M., and Swofford, D.L. 1992. Geographic origins of human mitochondrial DNA: Phylogenetic evidence from control region sequences. *Syst. Biol.* 41(1), 111–124.

McMorris, F.R., and Powers, R.C. 1993. Consensus functions on trees that satisfy an independence axiom. *Discrete Appl. Math.* 47, 47–55.

McMorris, F.R., and Steel, M.A. 1994. The complexity of the median procedure for binary trees, 136–140. *In* Diday, E., et al., eds., *New Approaches in Classification and Data Analysis. Proc. 4th Conf. Internat. Federation of Classification Societies, Paris, 1993.* Springer-Verlag, Berlin.

Meacham, C.A., and Estabrook, G.F. 1985. Compatibility methods in systematics. *Annu. Rev. Ecol. Syst.* 16, 431–446.

Nei, M. 1991. Relative efficiencies of different tree-making methods for molecular data, 90–128. *In* Miyamoto, M.M., and Cracraft, J., eds., *Phylogenetic Analysis of DNA Sequences.* Oxford University Press, New York.

Page, R.D.M. 1990. Tracks and trees in the antipodes: a reply to Humphries and Seberg. *Syst. Zool.* 39, 288–299.

Penny, D., Steel, M., Waddell, P.J., and Hendy, M.D. 1995. Improved analysis of human mtDNA sequences support a recent African origin for Homo sapiens. *Mol. Biol. Evol.* 12(5), 863–882.

Phillips, C., and Warnow, T.J. 1996. The asymmetric median tree—a new model for building consensus trees. *Proceedings of the Combinatorial Matching conference, Laguna Beach, CA* (to appear).

Templeton, A.R. 1991. Human origins and analysis of mitochondrial DNA sequences. *Science* 255, 737–737.

Vigilant, L., Stoneking, M., Harpending, H., Hawkes, K., and Wilson, A.C. 1991. African populations and the evolution of human mitochondrial DNA. *Science* 253, 1503–1507.

Wiley, E.O., Siegel-Causey, D., Brooks, D.R., and Funk, V.A. 1991. *The Compleat Cladist: A Primer of Phylogenetic Procedures.* Museum of Natural History, University of Kansas, Lawrence, Kansas.

Address reprint requests to:
David Bryant
Department of Mathematics and Statistics
University of Canterbury
Christchurch, New Zealand

Received for publication April 13, 1995; accepted as revised March 5, 1996.

This article has been cited by:

1. Alexey Markin, Oliver Eulenstein. 2019. Computing Manhattan Path-Difference Median Trees: A Practical Local Search Approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **16**:4, 1063-1076. [Crossref]
2. Alexey Markin, Oliver Eulenstein. 2019. Efficient Local Search for Euclidean Path-Difference Median Trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **16**:4, 1374-1385. [Crossref]
3. Atif Rahman, Lior Pachter. Transcript Abundance Estimation and the Laminar Packing Problem 203-211. [Crossref]
4. Alexey Markin, Oliver Eulenstein. Path-Difference Median Trees 211-223. [Crossref]
5. Alexey Markin, Oliver Eulenstein. Manhattan Path-Difference Median Trees 404-413. [Crossref]
6. Sarah Baskowski, Vincent Moulton, Andreas Spillner, Taoyang Wu. 2015. Neighborhoods of Trees in Circular Orderings. *Bulletin of Mathematical Biology* **77**:1, 46-70. [Crossref]
7. Sarah Bastkowski, Andreas Spillner, Vincent Moulton. 2014. Fishing for minimum evolution trees with Neighbor-Nets. *Information Processing Letters* **114**:1-2, 13-18. [Crossref]
8. David Bryant, Mike Steel. 2012. 'Bureaucratic' set systems, and their role in phylogenetics. *Applied Mathematics Letters* **25**:8, 1148-1152. [Crossref]
9. Sebastian Böcker, Andreas W.M. Dress. 2001. Patchworks. *Advances in Mathematics* **157**:1, 1-21. [Crossref]
10. David Bryant, Mike Steel. 2001. Constructing Optimal Trees from Quartets. *Journal of Algorithms* **38**:1, 237-259. [Crossref]
11. David Bryant. Optimal Agreement Supertrees 24-31. [Crossref]
12. Cynthia Phillips, Tandy J. Warnow. 1996. The asymmetric median tree — A new model for building consensus trees. *Discrete Applied Mathematics* **71**:1-3, 311-335. [Crossref]