Journal of Classification 2:7-28 (1985)

Journal of
Classification

Optimal Algorithms for Comparing Trees with Labeled Leaves

William H. E. Day

Memorial University of Newfoundland

Abstract: Let R_n denote the set of rooted trees with n leaves in which: the leaves are labeled by the integers in $\{1, \ldots, n\}$; and among interior vertices only the root may have degree two. Associated with each interior vertex v in such a tree is the subset, or *cluster*, of leaf labels in the subtree rooted at v. Cluster $\{1, \ldots, n\}$ is called *trivial*. Clusters are used in quantitative measures of similarity, dissimilarity and consensus among trees. For any k trees in R_n, the *strict consensus tree* $C(T_1, \ldots, T_k)$ is that tree in R_n containing exactly those clusters common to every one of the k trees. Similarity between trees T_1 and T_2 in R_n is measured by the number $S(T_1, T_2)$ of nontrivial clusters in both T_1 and T_2; dissimilarity, by the number $D(T_1, T_2)$ of clusters in T_1 or T_2 but not in both. Algorithms are known to compute $C(T_1, \ldots, T_k)$ in $O(kn^2)$ time, and $S(T_1, T_2)$ and $D(T_1, T_2)$ in $O(n^2)$ time. I propose a special representation of the clusters of any tree T in R_n, one that permits testing in constant time whether a given cluster exists in T. I describe algorithms that exploit this representation to compute $C(T_1, \ldots, T_k)$ in $O(kn)$ time, and $S(T_1, T_2)$ and $D(T_1, T_2)$ in $O(n)$ time. These algorithms are optimal in a technical sense. They enable well-known indices of consensus between two trees to be computed in $O(n)$ time. All these results apply as well to comparable problems involving unrooted trees with labeled leaves.

Keywords: Algorithm complexity; Algorithm design; Comparing hierarchical classifications; Comparing phylogenetic trees; Consensus index; Strict consensus tree.

1. Introduction

In recent years numerical taxonomists have become increasingly interested in the theory and practice of comparing tree-like structures such as cladograms, phenograms, dendrograms and phylogenetic trees. Frequently the essential object can be modeled as an acyclic connected undirected graph (i.e., a *tree*) in which the vertices of degree one (the *leaves*) are each assigned unique labels. Of the vertices of degree greater

The Natural Sciences and Engineering Research Council of Canada partially supported this work with grant A-4142.

Author's Address: William H. E. Day, Department of Computer Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1C 5S7.

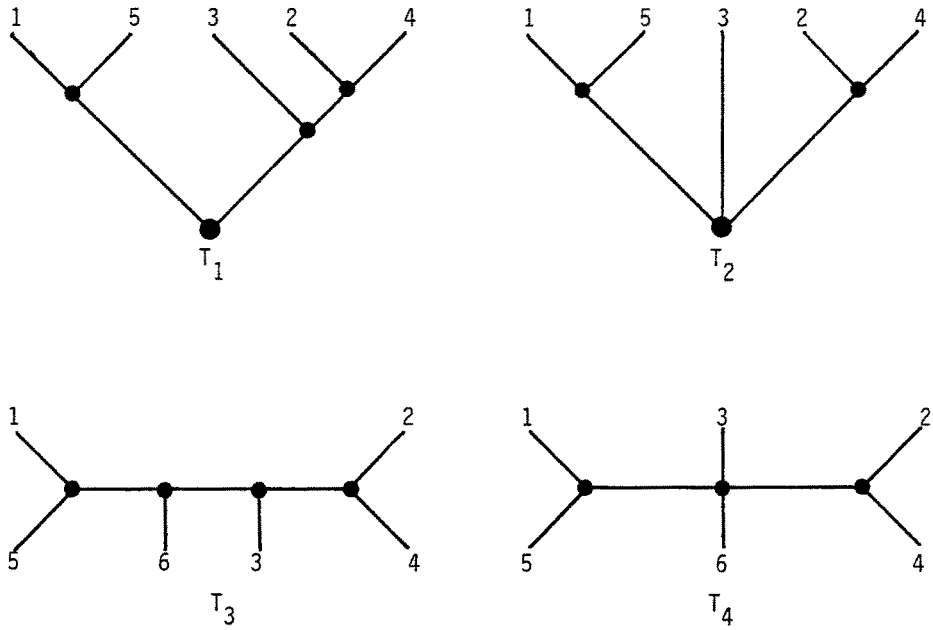


Figure 1. TREES WITH LABELED LEAVES. T_1 and T_2 are rooted trees drawn with the root at the bottom. T_1 and T_3 are binary trees. T_4 is neither rooted nor binary.

than one (the *interior* vertices), one (the *root*) may be distinguished from the others so that the tree becomes *rooted.* Among interior vertices only the root may have degree two; other interior vertices must have degree at least three. A tree is called *binary* if all its vertices have degree at most three. Figure 1 illustrates the various possibilities. Since it is convenient to suppose that leaves are labeled by distinct integers, we denote by R_n (respectively, U_n) the set of all such rooted (respectively, unrooted) trees with leaves labeled by the integers in $\{1, \ldots, n\}$. For further information concerning trees, the reader can consult Harary (1969) or any standard graph theory text.

Much of the interest in tree comparison concerns three central problems: to construct a consensus of a set of trees (Adams 1972; Margush and McMorris 1981; Sokal and Rohlf 1981; McMorris, Meronk and Neumann 1983; Neumann 1983; Stinebrickner 1984); to measure the degree of consensus among trees in a given set (Mickevich 1978; Colless 1980; Nelson and Platnick 1981, pp. 238-257; Schuh and Farris 1981; Rohlf 1982; Stinebrickner 1984); and to measure dissimilarity between two trees (Robinson 1971; Waterman and Smith 1978; Robinson and Foulds 1981; Brown and Day 1984; Hendy, Little and Penny 1984). In some respects these problems are closely related. A consensus tree method may be based on the

optimization of a consensus index. A consensus index method may measure consensus tree characteristics or relationships of consensus tree to original trees. Measuring pairwise dissimilarity between trees may form the basis of consensus tree or index methods. When consensus tree and index methods are restricted to sets of two trees, significant interrelationships among the three problems may exist (Day 1983).

Any tree comparison technique is not likely to be useful unless it is based on analysis of meaningful features in the input data. For trees in R_n, taxonomists associate with each interior vertex v the subset, or *cluster*, of leaf labels in the subtree rooted at v. In Figure 1, for example, T_2 has clusters {1,5}, {2,4} and {1,...,5} associated with its three interior vertices. The set of all such clusters for T is called the *cluster representation* of T and is denoted by T'. For the rooted trees in Figure 1, $T_1'=$ $\{\{1,5\},\{2,4\},\{2,3,4\},\{1,...,5\}\}$ and $T_2'=$ $\{\{1,5\},\{2,4\},\{1,...,5\}\}$. Since $N=\{1,\ldots,n\}$ appears in the cluster representation of every T in R_n, it is called the *trivial* cluster. Clusters can be introduced in several similar contexts. McMorris, Meronk and Neumann (1983) define an *n-tree* on N as a set T of subsets of N satisfying $N \in T, \varnothing \notin T, \{i\} \in T$ for every i in N, and $X \cap Y \in \{\varnothing, X, Y\}$ for every X and Y in T. They call the proper nonsingleton elements of T its *clusters* and consider them the n-tree's important features. Hendy, Little and Penny's hierarchical classifications are n-trees, their directed phylogenetic trees are essentially rooted trees in R_n, and they use clusters to establish the obvious bijection between the sets of hierarchical classifications and directed phylogenetic trees (Hendy, Little and Penny 1984).

Cluster representations play a fundamental role in methods constructing the consensus of a set of trees in R_n. If k is a positive integer and R_n^k denotes the k-fold Cartesian product, the *strict* consensus tree method is a function C: $R_n^k \rightarrow R_n$ such that $C(T_1,\ldots,T_k)' = \cap_{1\leqslant i\leqslant k} T_i'$ for all $T_1,\ldots,T_k$ in R_n. Sokal and Rohlf (1981) call $C(T_1,\ldots,T_k)$ strict because its cluster representation contains just those clusters that are common to all k trees being compared. Figure 2 exhibits the strict consensus $C(T_1,T_2)$ in an example where $C(T_1,T_2)' = T_1' \cap T_2' =$ $\{\{8,11\},\{6,8,11\},\{1,...,14\}\}$. McMorris, Meronk and Neumann (1983) parameterize the strictness concept in a pleasing way: let l be an integer such that $\lfloor k/2 \rfloor + 1 \leqslant l \leqslant k$ ($\lfloor x \rfloor$ denoting the greatest integer not exceeding x), and define the function M_l: $R_n^k \rightarrow R_n$ by placing a cluster in $M_l(T_1,\ldots,T_k)'$ if and only if it is in at least l of $T_1',\ldots,T_k'$. If $l = k$, M_l is strict consensus C; while if $l = \lfloor k/2 \rfloor + 1$, M_l is the *majority rule* consensus tree method of Margush and McMorris (1981). McMorris and Neumann (1983) characterize M_l methods axiomatically.

Cluster representations play a fundamental role in methods measuring similarity or dissimilarity between two trees in R_n. To measure similarity, define the function S: $R_n^2 \rightarrow Z_0^+$ such that $S(T_1,T_2) = |T_1' \cap T_2'| - 1$ for all

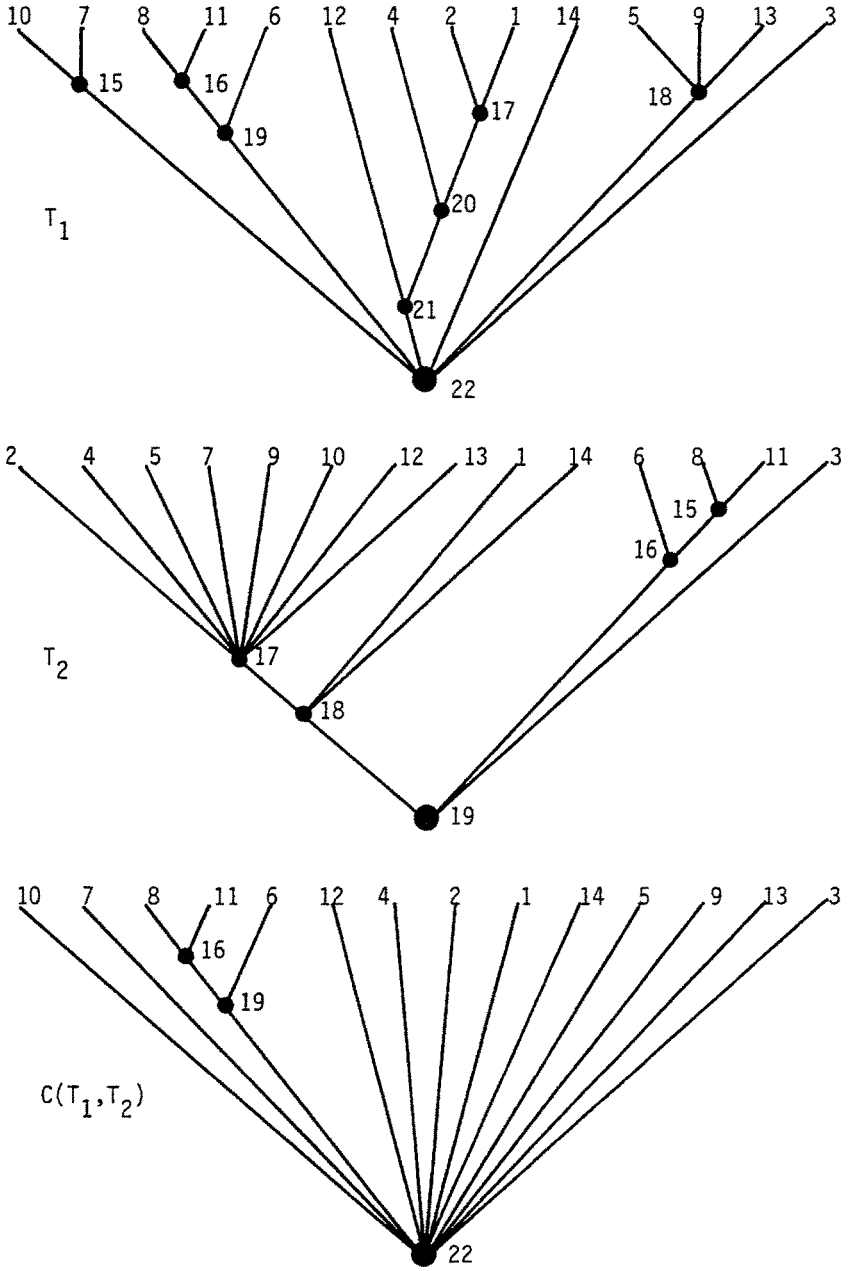


Figure 2. A STRICT CONSENSUS TREE. T_1 and T_2 are modified from phenograms 1.2 and 1.3 in Nelson's Figure 1 (1979). Interior vertices are labeled to facilitate each tree's representation as a postorder sequence with weights.

T_1, T_2 in R_n. Since $S(T_1, T_2)$ counts the nontrivial clusters common to both trees, it has maximum value $(n-2)$ when $T_1 = T_2$ for binary trees, and minimum value (0) when T_1' and T_2' have only the trivial cluster N in common. In terms of strict consensus, $S(T_1, T_2) = |C(T_1, T_2)'| - 1$; equivalently, S counts the nontrivial branch points (*informative components*) in $C(T_1, T_2)$ and is the unnormalized version of Nelson's measure of *component information* (1979). Although Rohlf (1982) describes S, it usually appears in normalized form (Nelson 1979, Colless 1980, Sokal and Rohlf 1981). To measure dissimilarity between trees, let Δ denote the symmetric difference operator on sets and define the function D: $R_n^2 \rightarrow Z_0^+$ such that $D(T_1, T_2) = |T_1' \Delta T_2'|$ for all T_1, T_2 in R_n. Since $D(T_1, T_2)$ counts the clusters in T_1' or T_2' but not in both, it has minimum value (0) when $T_1 = T_2$ and maximum value $(2n-4)$ when T_1 and T_2 are binary and have only the trivial cluster in common. In terms of strict consensus, $D(T_1, T_2) = |T_1'| + |T_2'| - 2 \cdot |C(T_1, T_2)'|$ (Day 1983). D is a metric measure of dissimilarity (Restle 1959; Margush 1982; Day 1983) and has been given axiomatic characterizations (Margush 1982; Day 1983). Tateno, Nei and Tajima (1982) call it the *distortion index*.

Any tree comparison technique is not likely to be useful unless an algorithm for its computation exists that makes efficient use of time and space resources. To evaluate time and space complexities of such algorithms, let n be a measure of problem size and describe an algorithm's *time* (respectively, *space*) *complexity* by a function f expressing for each n the largest amount of time (respectively, space) the algorithm needs to solve any problem instance of size n. In describing the asymptotic behavior of such positive-valued functions, we say that $f(n)$ is $O(g(n))$ whenever there exists a positive constant c such that $f(n) \leqslant c \cdot g(n)$ for all large positive n; $f(n)$ is $\Omega(g(n))$ if $g(n)$ is $O(f(n))$. An algorithm is called *efficient* if its time complexity function is $O(p(n))$ for some polynomial function $p(n)$. An algorithm is called *optimal* if its time complexity is $O(g(n))$ for a lower bound $g(n)$ on the time complexities of all algorithms solving the given problem. When comparing two trees in R_n, it is reasonable to use n itself to measure problem size; but when computing the consensus of k trees in R_n, one must use both k and n. For general information on the analysis of algorithm complexity, the reader can consult the classic textbook by Aho, Hopcroft and Ullman (1974) or the fine survey by Weide (1977).

Very little has been published concerning the design and analysis of algorithms to solve tree comparison problems. In one sentence Robinson and Foulds (1981, p. 146) sketch an algorithm that can be used for D; they conclude that "the calculation of distance therefore presents no difficulties for practical-sized problems." Tateno, Nei and Tajima (1982, p. 391) remark that the computation of D "is straightforward and simple." In a few sentences Hendy, Little and Penny (1984) sketch an algorithm applicable to the computation of D, one apparently requiring $O(n^3)$ time for trees in R_n.

Recently F. J. Rohlf (personal communication) outlined for trees in R_n an algorithm strategy enabling $C(T_1, \ldots, T_k)$ to be computed in $O(kn^2)$ time and $S(T_1, T_2)$ and $D(T_1, T_2)$, in $O(n^2)$ time. I have since investigated whether improved algorithms exist for C, D and S. As a main result I describe in Section 2 an optimal algorithm computing for trees in R_n the strict consensus $C(T_1, \ldots, T_k)$ in $O(kn)$ time. In Section 3 I extend this result to the domain U_n of unrooted trees with labeled leaves. In Section 4 I explain how to compute $D(T_1, T_2)$, $S(T_1, T_2)$ and other consensus indices in $O(n)$ time. In Section 5 I use the new algorithms to examine distributions of dissimilarities for normalized versions of D. I conclude by describing several research problems suggested by this investigation.

2. Computing Strict Consensus Trees in R_n

The time and space complexities of tree comparison algorithms are sensitive to the data structure adopted as the external representation of trees in R_n. To be efficient in its use of space, this representation must occupy $O(n)$ space for a tree in R_n; to foster the design of efficient algorithms, it must provide tree information in a form that is natural and convenient for both users and implementers of tree comparison algorithms. Sequential tree representations incorporating a postorder enumeration of vertices (Standish 1980, pp. 67-73) meet these requirements, and so I adopt the *postorder sequence with weights* (PSW) as the external representation of trees in R_n.

This representation requires for each tree in R_n that interior vertices be assigned arbitrary distinct integer labels and that subtrees above each vertex be ordered in the plane from left to right. The trees in Figure 2 satisfy these requirements. If such a tree has m interior vertices and n leaves, its PSW is a sequence $<u_1, \ldots, u_{m+n}>$ of ordered pairs $u_j = <v_j, w_j>$ in which v_j is a vertex label and w_j is the weight of the subtree rooted at v_j, i.e. the number of vertices in this subtree, excluding v_j. In T_1 of Figure 2, the weight of vertex 19 is 4 since it counts vertices 6, 8, 11 and 16; the weight of every leaf is 0. A tree's vertices appear in its PSW in the order in which they are visited during a postorder traversal (Standish 1980, pp. 60, 72) from its root. Table 1 exhibits the PSWs for the trees in Figure 2.

Since tree comparison algorithms operate on PSWs in standard ways, it is convenient to isolate these elementary operations as separate procedures. Table 2 describes procedures to build a PSW for a tree (PREPARE, ENTER, TEND), to measure a tree's size (M, N), to traverse a tree (TRESET, NVERTEX) and to obtain structural information during traversal (LEFTLEAF). Each procedure can be implemented in Pidgin ALGOL (Aho, Hopcroft and Ullman 1974, Section 1.8) so that it executes in $O(1)$ time; each is straightforward in design and requires no further comment.

Recall that the strict consensus of $T_1, \ldots, T_k$ in R_n is a tree $T = C(T_1, \ldots, T_k)$ with cluster representation $T' = \cap_{1 \leqslant i \leqslant k} T_i'$. Table 3

TABLE 1

Trees Represented as Postorder Sequences With Weights

	T_1		$C(T_1, T_2)$		T_2	
j	v_j	w_j	v_j	w_j	v_j	w_j
1	10	0	10	0	2	0
2	7	0	7	0	4	0
3	15	2	8	0	5	0
4	8	0	11	0	7	0
5	11	0	16	2	9	0
6	16	2	6	0	10	0
7	6	0	19	4	12	0
8	19	4	12	0	13	0
9	12	0	4	0	17	8
10	4	0	2	0	1	0
11	2	0	1	0	14	0
12	1	0	14	0	18	11
13	17	2	5	0	6	0
14	20	4	9	0	8	0
15	21	6	13	0	11	0
16	14	0	3	0	15	2
17	5	0	22	16	16	4
18	9	0			3	0
19	13	0			19	18
20	18	3				
21	3	0				
22	22	21				

exhibits a high-level Pidgin ALGOL procedure, called COMCLUSTER, to compute T' in this case. Initially T_1' is taken as an approximation T' to the final result (line 1). Each remaining T_i' is compared to T' (lines 3-8). Boolean switches associated with clusters in T' are cleared to indicate that no clusters have yet been found in T_i' (lines 3-4). Each cluster in T_i' is tested for membership in T'; if present in T', the corresponding switch is set (lines 5-6). The switches are used to delete from T' clusters that are not in T_i' (lines 7-8). The bottleneck computation occurs at line 6. Any straightforward implementation of the cluster membership test looks at each cluster element and requires $\Omega(n)$ time; since line 6 is executed $\Omega(kn)$ times, straightforward implementations of COMCLUSTER have $\Omega(kn^2)$ time complexity. This asymptotic behavior improves only if one designs a more efficient cluster membership test. The solution to this design problem involves two complementary ideas: to relabel the tree leaves so that each cluster is completely described by two integers; and to encode these descriptions in a table in such a way that the cluster membership test requires $O(1)$

TABLE 2

Procedures to Manipulate Trees Represented as Postorder Sequences with Weights

PREPARE(T)
This procedure prepares to construct a postorder sequence T with weights. Initially T is the empty sequence.

ENTER(T,v,w)
This procedure appends to the end of T an entry <v,w> associating weight w with vertex v.

TEND(T)
This function procedure returns the index of the last entry in T.

M(T)
This function procedure returns as its value the number of interior vertices in T.

N(T)
This function procedure returns as its value the number of leaves in T.

TRESET(T)
This procedure prepares T for an enumeration of its entries, beginning with the first entry.

NVERTEX(T,v,w)
This procedure returns the next entry <v,w> in the current enumeration of T. Valid entries are returned by the first M(T) + N(T) invocations of NVERTEX after initialization by TRESET; thereafter NVERTEX returns the invalid entry <0,0>.

LEFTLEAF(T)
If NVERTEX has returned entry $< v_j, w_j >$ in T, the leftmost leaf in the subtree rooted at v_j has entry $< v_k, w_k >$ where k = j − w_j. This function procedure returns v_k as its value.

time. With this data structure design, line 6 requires $O(1)$ time so that COMCLUSTER has $O(kn)$ time complexity.

Leaf relabeling is accomplished during a postorder traversal of any tree T in R_n. Extract from T's PSW the subsequence $< x_1, \ldots, x_n >$ of leaves in the order in which they are visited during traversal. Define the function e: $N \rightarrow N$ such that for each j in N, $e(j) = i$ when $j = x_i$, i.e. when j is the ith leaf visited during traversal. Clearly e is a bijection on N. Informally, the encoding function e enables one to pass back and forth between leaf labeling conventions convenient to user (external labels) and implementer (internal labels). When any interior vertex v is visited during traversal, the leaves $< x_L, \ldots, x_R >$ in its subtree have been relabeled with integers in the contiguous subsequence $< e(x_L), \ldots, e(x_R) > = < L,$

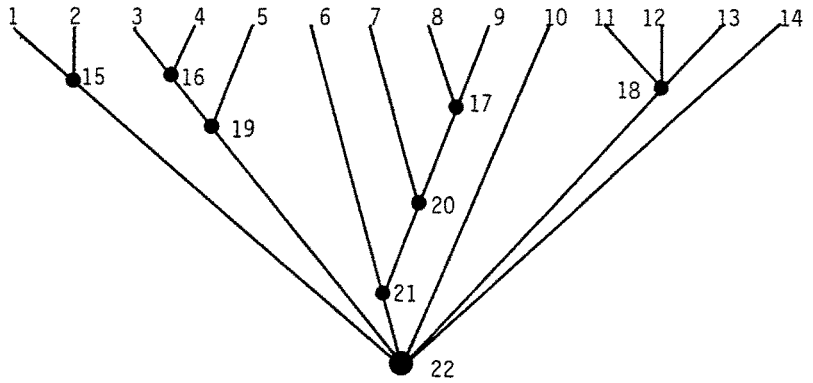
TABLE 3

The COMCLUSTER Procedure

```
     procedure COMCLUSTER(k,T1,...,Tk,T):
     begin
1.       T' ← T'1;
2.       for i ← 2 until k do
             begin
3.               for j ← 1until |T'| do
4.                   switch[j] ← false;
5.               for each cluster Z in T'i do
6.                   if Z = Zj in T' then switch[j] ← true;
7.               for j ← 1 until |T'| do
8.                   if switch[j] = false then T' ← T' − {Zj}
             end
     end
```

$L + 1, \ldots, R>$. Thus in the relabeled tree the cluster associated with v is described concisely by the ordered pair $<L,R>$. By extension, the cluster representation of T can be described concisely by e and the ordered-pair descriptions of T's clusters. Figure 3 exhibits the tree T_1 in Figure 2 with its leaves relabeled after a traversal of vertices in the order shown in Table 1. The encoding function e has mapped the user's leaf labels <1,...,14> into the internal labels <9,8,14,7,11,5,2,3,12,1,4,6,13,10>. After relabeling, the cluster representation of T_1 can be described by the ordered pairs in {<1,2>,<1,14>,<3,4>,<3,5>,<6,9>,<7,9>,<8,9>,<11,13>}.

It remains to design a data structure for e and the encoded cluster representation of T that enables cluster membership to be determined in $O(1)$ time. One solution associates with T an array $X = (X_{ij})$, called the *cluster table* for T, that has n rows and four columns. Column 4 contains cluster switches of the type used by the COMCLUSTER procedure. Column 3 describes the encoding function e. Columns 1 and 2 contain the ordered-pair descriptions of T's clusters. Figure 3 exhibits the cluster table for T_1 in Figure 2. Each cluster $<L,R>$ is assigned a row of X where L is placed in column 1 and R, in column 2. Assignment of cluster to row is specified by a function g defined as follows. Let $<u_1, \ldots, u_{m+n}>$ be T's PSW, where $u_j = <v_j, w_j>$, and extract the subsequence $<y_1, \ldots, y_m>$ of interior vertices in the order in which they are visited by a postorder traversal. Associated with each y_j is the cluster $<L_j,R_j>$. Let f: $\{1, \ldots, m\} \rightarrow \{1, \ldots, m+n\}$ be the function mapping each y_j to its corresponding PSW position, i.e. $y_j = v_{f(j)}$.



	1	2	3	4
1			9	
2	1	2	8	
3	0	0	14	
4	3	4	7	
5	3	5	11	
6	0	0	5	
7	7	9	2	
8	8	9	3	
9	6	9	12	
10	0	0	1	
11	0	0	4	
12	0	0.	6	
13	11	13	13	
14	1	14	10	

Figure 3. A CLUSTER TABLE. T_1 of Figure 2 is shown with leaves relabeled to permit the description of its clusters by ordered pairs of integers. In the cluster table, columns 1 and 2 describe clusters; column 3 describes the leaf relabeling.

Define the function $g : \{1, \ldots, m\} \rightarrow \{2, \ldots, n\}$ such that for each j,

$$g(j) = \begin{cases} L_j \text{ if } j < m \text{ and } w_{f(j)+1} > 0, \\ R_j \text{ if } j < m \text{ and } w_{f(j)+1} = 0, \\ n \text{ if } j = m. \end{cases} \tag{1}$$

Theorem *g is an injection.*

Proof. Let j and k be integers in $\{1, \ldots, m\}$ such that $g(j) = g(k)$. If $g(j) = n$ then immediately $j = k$. Let $g(j) < n$ and suppose without loss of generality that $L_k \leqslant L_j$ and $R_j \leqslant R_k$. Either $g(j) = L_j$ or $g(j) = R_j$. If $g(j) = L_j$ and $j \neq k$, then $w_{f(j)+1} > 0$ so that $L_k < L_j = g(j) = g(k) = L_k$, a contradiction. If $g(j) = R_j$ and $j \neq k$, then $w_{f(j)+1} = 0$ so that $R_k > R_j = g(j) = g(k) = R_k$, a contradication. Thus $j = k$. ●

Corollary *g is a bijection if and only if T is binary.*

Proof. T is binary if and only if $m = n - 1$; the result follows by the pigeon-hole principle. ●

The placement of cluster $<L_j, R_j>$ in X is determined by the rule that $X_{g(j),1} = L_j$ and $X_{g(j),2} = R_j$; the definition of g guarantees that $<L_j, R_j>$ appears in exactly one of two possible rows. Any arbitrary cluster $<L, R>$ may or may not be in X; but if it is, its description must appear in either row L or row R. Thus by using X the cluster membership test can be implemented to execute in $O(1)$ time.

Since tree comparison algorithms operate on cluster tables in standard ways, it is convenient to isolate these operations as separate procedures. Table 4 describes procedures to build a cluster table (BUILD), to establish the correspondence between external and internal labels (ENCODE), to test for cluster membership (ISCLUST), to manipulate switches associated with clusters (CLEAR, SETSW, UPDATE), and to enumerate clusters (XRESET, NCLUS). These procedures can be implemented in Pidgin ALGOL so that BUILD, CLEAR and UPDATE execute in $O(n)$ time and the others, in $O(1)$ time. Of them, only BUILD warrents further discussion.

Table 5 exhibits a Pidgin ALGOL version of the BUILD procedure. The cluster table X is initialized (lines 2-3) and tree T in R_n is prepared for a postorder traversal (line 1). The traversal determines whether the vertex v being visited is a leaf (lines 8-10) or an interior vertex (lines 11-15). If v is a leaf, the encoding function value $e(v)$ is stored in column 3 of X (line 9). Otherwise the cluster $<L, R>$ associated with v is identified (lines 9,11), its placement in X is determined (lines 12-13), and it is stored in columns 1 and 2 of X (lines 14-15). In the placement calculation at line 13, variables w and *loc* correspond to $w_{f(j)+1}$ and $g(j)$ in equation (1). Since initialization (lines 1-5) and traversal (lines 6-15) each require $O(n)$ time, BUILD has $O(n)$ time complexity.

Recall that the strict consensus of $T_1, \ldots, T_k$ in R_n is a tree $T = C(T_1, \ldots, T_k)$ with cluster representation $T' = \cap_{1 \leqslant i \leqslant k} T_i'$. Table 6 exhibits a Pidgin ALGOL procedure, called COMCLUST, that computes T'

TABLE 4

Procedures to Manipulate Cluster Tables

BUILD(T,X)

This procedure constructs in X descriptions of the clusters in a rooted tree described by the postorder sequence T with weights. BUILD assigns each leaf an internal label so that every cluster is a set $\{i : L \leqslant i \leqslant R\}$ of internal labels; thus each cluster is simply described by a pair <L,R> of internal labels.

ENCODE(X,v)

This function procedure returns as its value the internal label assigned to leaf v.

ISCLUST(X,L,R)

This function procedure returns value *true* if cluster <L,R> is in X; otherwise it returns value *false*.

CLEAR(X)

Each cluster in X has an associated switch that is either cleared or set. This procedure clears every cluster switch in X.

SETSW(X,L,R)

If <L,R> is a cluster in X, this procedure sets the cluster switch for <L,R>.

UPDATE(X)

This procedure inspects every cluster switch in X. If the switch for cluster <L,R> is cleared, UPDATE deletes <L,R> from X; thereafter ISCLUST(X,L,R) will return the value *false*.

XRESET(X)

This procedure prepares X for an enumeration of its clusters.

NCLUS(X,L,R)

This procedure returns the next cluster <L,R> in the current enumeration of clusters in X. If m clusters are in X, they are returned by the first m invocations of NCLUS after initialization by XRESET; thereafter NCLUS returns the invalid cluster <0,0>.

by incorporating a cluster table in the COMCLUSTER design. That COMCLUST reflects COMCLUSTER's structure is shown by mapping COMCLUSTER statements to comparable COMCLUST statements: 1→1; 2→2; 3–4→4; 5→6–14; 6→15; 7–8→18. COMCLUST builds the cluster table X from T_1 (line 1) and then traverses each remaining T_i to identify and process its clusters (lines 2-18). The increased complexity in COMCLUST's design (lines 6-14) involves constructing for each cluster Z in T_i' a description $<L,R,N>$ where $L = \min\{e(v):v\in Z\}$, $R = \max\{e(v):v\in Z\}$, $N = |Z|$ and e is the encoding function defined in column 3 of X. If $N = R - L + 1$, Z's internal labels form a contiguous sequence

TABLE 5

The BUILD Procedure

```
        procedure BUILD(T,X):
        begin
1.          TRESET(T);
2.          for i ← 2 until N(T) do
3.                X[i,1] ← X[i,2] ← 0;
4.          leafcode ← 0;
5.          NVERTEX(T,v,w);
6.          while v ≠ 0 do
7.                if v is a leaf then
                        begin
8.                          leafcode ← leafcode + 1;
9.                          X[v,3] ← R ← leafcode;
10.                         NVERTEX(T,v,w)
                        end
                  else
                        begin
11.                         L ← X[LEFTLEAF(T),3];
12.                         NVERTEX(T,v,w);
13.                         if w = 0 then loc ← R else loc ← L;
14.                         X[loc,1] ← L;
15.                         X[loc,2] ← R
                        end
        end
```

$<L,L+1,\ldots,R>$ so that membership of Z in X must be tested (line 15); otherwise Z cannot be a cluster in X.

For example, suppose cluster table X in Figure 3 is built from T_1 in Figure 2 and consider clusters of T_2 in Figure 2. Since cluster {6,8,11} has internal labels {5,3,4}, $<L,R,N> = <3,5,3>$; since $N = R-L+1$, the internal labels form a contiguous sequence $<3,4,5>$ so that membership of $<3,5>$ in X must be tested. On the other hand, cluster {2,4,5,7,9,10,12,13} has $<L,R,N> = <1,13,8>$; the cluster cannot be in X since $N \neq R-L+1$.

To derive COMCLUST's time complexity, consider first the tree traversal loop (lines 3-18). Initialization (lines 3-4), leaf processing (lines 6-7) and postprocessing (line 18) each require $O(n)$ time. Since $O(n)$ entries are pushed on stack S (lines 7, 14), $O(n)$ entries are popped (line 10) so that interior vertex processing (lines 8-15) requires $O(n)$ time. Thus each tree T_i, $2 \leqslant i \leqslant k$, is processed in $O(n)$ time; since T_1 processing also requires $O(n)$ time (line 1), COMCLUST has time complexity $O(kn)$.

TABLE 6

The COMCLUST Procedure

```
        procedure COMCLUST(k,T1,...,Tk,X):
        begin
1.          BUILD(T1,X);
2.          for i ← 2 until k do
                begin
3.                  Empty the stack S;
4.                  CLEAR(X); TRESET(Ti); NVERTEX(Ti,v,w);
5.                  repeat
6.                      if v is a leaf then
7.                          push(<ENCODE(v),ENCODE(v),1,1>,S);
                        else
                            begin
8.                              <L,R,N,W> ← <∞,0,0,1>;
9.                              repeat
10.                                 <L*,R*,N*,W*> ← pop(S);
11.                                 <L,R,N,W> ← <min(L,L*),
                                    max(R,R*), N + N*, W + W*>;
12.                                 w ← w - W*
13.                             until w = 0;
14.                             push(<L,R,N,W>,S);
15.                             if N = R − L + 1 then SETSW(X,L,R)
                            end
16.                     NVERTEX(Ti,v,w)
17.                 until v = 0;
18.                 UPDATE(X)
                end
        end
```

NOTE: *push*(r,S) inserts r as the top entry of stack S; *pop*(S) deletes the top entry r from stack S and returns r as the procedure value.

When COMCLUST terminates execution, cluster table X describes the cluster representation T' of the strict consensus $C(T_1, \ldots, T_k)$. It remains to construct a PSW T corresponding to T'. Although computing T directly from X seems difficult, computing T from X and T_1 is straightforward. In Table 1, for example, a PSW for $C(T_1, T_2)$ can be obtained from T_1's PSW by removing entries for interior vertices 15, 17, 20, 21 and 18. Table 7 exhibits a Pidgin ALGOL procedure, called CONTREE, that computes a PSW T for $C(T_1, \ldots, T_k)$ using this strategy. COMCLUST is invoked, T is prepared for PSW construction, and T_1 is prepared for postorder traversal (lines 1-2). Entries for T_1's leaves are copied to T (lines 5-7). Entries for T_1's interior vertices are copied to T if their associated clusters are still recorded in X (lines 8-10). Since T_1's traversal (lines 3-12) requires $O(n)$

TABLE 7

The CONTREE Procedure

```
     procedure CONTREE(k,T1,...,Tk,X,T):
     begin
1.       COMCLUST(k,T1,...,Tk,X);
2.       PREPARE(T); TRESET(T1); NVERTEX(T1,v,w);
3.       repeat
4.           if v is a leaf then
                 begin
5.                   R ← ENCODE(X,v);
6.                   ENTER(T,v,w);
7.                   FA[v] ← TEND(T)
                 end
             else
                 begin
8.                   L ← ENCODE(X,LEFTLEAF(T1));
9.                   if ISCLUST(X,L,R) then
10.                      ENTER(T,v,TEND(T) + 1 - FA[LEFTLEAF(T1)])
                 end
11.          NVERTEX(T1,v,w)
12.      until v = 0
     end
```

NOTE: Array FA can be equivalenced to the fourth column of X.

time, COMCLUST dominates execution (line 1) so that CONTREE has $O(kn)$ time complexity.

To establish a lower bound on the time complexities of strict consensus algorithms, recall Cavalli-Sforza and Edwards' result (1967) that R_n contains $\Pi_{2\leqslant i\leqslant n}(2i-3)$ binary trees so that $|R_n| \geqslant 2^n$ for $n \geqslant 5$. Therefore it seems reasonable that general-purpose representations of trees in R_n require $\Omega(n)$ bits. Since strict consensus algorithms then require $\Omega(kn)$ time just to identify the problem, and since CONTREE solves the problem in $O(kn)$ time, CONTREE is an optimal algorithm for the strict consensus tree problem.

3. Computing Strict Consensus Trees in U_n

The strict consensus concept can be developed for U_n as well as for R_n. Let an edge of any tree T in U_n be called *interior* if it is not incident with a leaf. Deleting an interior edge separates T into two components and yields a corresponding bipartition of leaf labels into two subsets. The set of all such partitions is called the *partition representation* of T and is denoted by T'. The partition representation of T_3 in Figure 1 is $T_3' = \{\{\{1,5\},\{2,3,4,6\}\},$

{{2,3,4},{1,5,6}},{{2,4},{1,3,5,6}}}. Partition representations play in U_n the role of cluster representations in R_n. Thus for k a positive integer, the *strict* consensus tree method is a function C: $U_n^k \rightarrow U_n$ such that $C(T_1, \ldots, T_k)' = \cap_{1 \leqslant i \leqslant k} T_i'$ for all $T_1, \ldots, T_k$ in U_n.

Although $C(T_1, \ldots, T_k)$ can be computed using partition representations in U_n, it also can be computed using cluster representations in R_{n-1}. To see this, define function Φ: $U_n \rightarrow R_{n-1}$ such that if T in U_n has edge $\{v,n\}$ incident with leaf n, $\Phi(T)$ is obtained from T by deleting $\{v,n\}$ and n, and taking v as the root. In Figure 1, $\Phi(T_3) = T_1$ and $\Phi(T_4) = T_2$. The function Φ is a bijection (Rohlf 1983; Hendy, Little and Penny 1984) so that $\Phi^{-1}(T_1) = T_3$ and $\Phi^{-1}(T_2) = T_4$ in Figure 1. Using Φ, strict consensus in U_n can be solved as a consensus problem in R_{n-1} by computing

$$C(T_1, \ldots, T_k) = \Phi^{-1}(C(\Phi(T_1), \ldots, \Phi(T_k))) \quad . \tag{2}$$

If each T in U_n is represented by a PSW (the traversal being from leaf n), the PSW for $\Phi(T)$ is simply the PSW for T without its last entry. As a result, the strict consensus of k trees in U_n can be computed in $O(kn)$ time using equation (2) and the CONTREE procedure for trees in R_n.

4. Computing Consensus Indices

Consensus indices measure degree of agreement among objects in a set. Taxonomists often define consensus indices on trees T_1 and T_2 in R_n or in U_n. Table 8 exhibits eleven such indices, each computable in $O(n)$ time from information in the cluster tables for T_1, T_2 and their strict consensus. For further information the reader can consult: Bourque (1978), Robinson and Foulds (1981), Margush (1982) and Hendy, Little and Penny (1984) for D; Nelson (1979), Colless (1980) and Sokal and Rohlf (1981) for CI_C; Mickevich (1978) and Rohlf (1982) for CI_M; Nelson (1979) and Nelson and Platnick (1981) for term information (TERM) and total information (TOTAL); and Schuh and Farris (1981) for levels sum (LSUM). Day (1983) describes a comparison model relevant to CI_C, TERM, TOTAL and LSUM. Shao (1983) investigates interrelationships among many of these consensus indices.

D and S are basic unnormalized indices of dissimilarity and similarity between two trees. D is an unnormalized metric based on the symmetric difference of sets (Restle 1959; Margush 1982). Essentially $S(T,T)$ counts interior vertices in T; taxonomists say that it measures the *resolution* of T. In Table 8 the pairs (d,s) and (d',s') represent alternative strategies for normalizing the pair (D,S), where $d(T_1,T_2) + s(T_1,T_2) = d'(T_1,T_2) + s'(T_1,T_2) = 1$. Notice that $s'(T_1,T_2) = 2s(T_1,T_2) / (1+s(T_1, T_2))$ and $d'(T_1,T_2) = d(T_1,T_2) / (2-d(T_1,T_2))$. The index d is a normalized metric

TABLE 8

Indices of Consensus Between Two Trees

Index	Domain		Expression
$D(T_1,T_2)$	R_n,	U_n	$M(T_1)+M(T_2)-2 \cdot M(T)$
$S(T_1,T_2)$	R_n,	U_n	$M(T)-1$
$d(T_1,T_2)$	R_n,	U_n	$D(T_1,T_2)/\ [D(T_1,T_2)+S(T_1,T_2)]$
$s(T_1,T_2)$	R_n,	U_n	$S(T_1,T_2)/\ [D(T_1,T_2)+S(T_1,T_2)]$
$d'(T_1,T_2)$	R_n,	U_n	$D(T_1,T_2)/\ [D(T_1,T_2)+2 \cdot S(T_1,T_2)]$
$s'(T_1,T_2)$	R_n,	U_n	$2 \cdot S(T_1,T_2)/\ [D(T_1,T_2)+2 \cdot S(T_1,T_2)]$
$CI_C(T_1,T_2)$	R_n		$S(T_1,T_2)/\ [n-2]$
		U_n	$S(T_1,T_2)/\ [n-3]$
$CI_M(T_1,T_2)$	R_n		$\sum_i \min\{n_i-1, n-n_i\}/\ [\lfloor (n-1)/2 \rfloor \cdot \lfloor n/2 \rfloor]$
		U_n	$\sum_i \min\{n_i-1, n-n_i-1\}/\ [\lfloor (n-1)/2 \rfloor \cdot \lfloor (n-2)/2 \rfloor]$
$\mathrm{TERM}(T_1,T_2)$	R_n		$[\sum_i n_i-n-S(T_1,T_2)]/\ [(n-1)(n-2)/2]$
$\mathrm{TOTAL}(T_1,T_2)$	R_n		$[\sum_i n_i-n]/\ [(n+1)(n-2)/2]$
$\mathrm{LSUM}(T_1,T_2)$	R_n		$[\sum_i \binom{n_i}{2}-\binom{n}{2}]/\ [n(n-1)(n-2)/6]$

NOTE: $T=C(T_1,T_2)$. $M(T)$ is the number of interior vertices in T. In R_n each n_i is the cardinality of the ith cluster in T'; in U_n each n_i is the cardinality of a block in the ith bipartition in T'.

based on the symmetric difference of sets (Marczewski and Steinhaus 1958). Since $s'(T_1,T_2) = S(T_1,T_2) / [(S(T_1,T_1) + S(T_2,T_2)) / 2]$, it is the ratio of the strict consensus tree's resolution to the average resolution of T_1 and T_2. For binary trees, the denominator in the d' and s' definitions is independent of T_1 and T_2 and depends only on n; in this case d' is a metric and s' equals CI_C.

TABLE 9

The TERM Procedure

```
    procedure TERM(T1,T2,X,T):
    begin
1.      CONTREE(2,T1,T2,X,T);
2.      num ← 1 - M(T) - N(T);
3.      XRESET(X); NCLUS(X,L,R);
4.      repeat
5.          num ← num + R - L + 1;
6.          NCLUS(X,L,R)
7.      until L = 0;
8.      return 2 · num/((N(T) - 1)(N(T) - 2))
    end
```

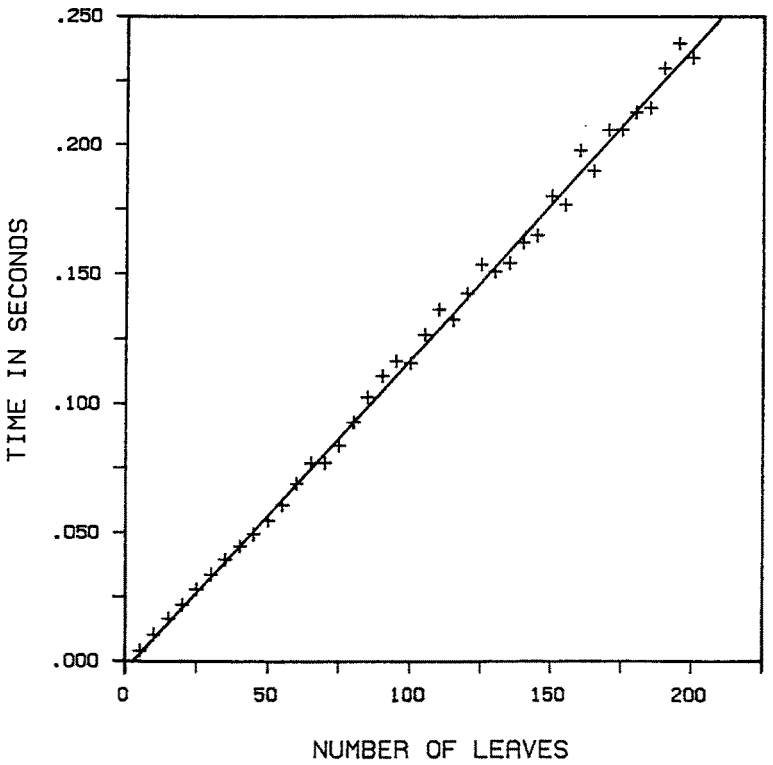


Figure 4. TIME REQUIRED TO CALCULATE D AND S CONSENSUS INDICES BETWEEN RANDOMLY SELECTED UNROOTED BINARY TREES WITH n LABELED LEAVES. Each point is the average of 10,000 computations on a DEC VAX 11/780 computer. The least-squares line is $f(n) = .001202n - .002786$.

TABLE 10

Statistics for Empirical Distributions of 10,000 d and d' Distances Between Randomly Selected Binary Trees with n Leaves

n	Mean	Std. Dev.	Skewness	Kurtosis	5% C.V.	C.N.
			d			
20	.99489	.013	-2.735	12.831	.93750	5
40	.99803	.005	-2.742	11.238	.97222	4
60	.99879	.003	-2.808	11.235	.98214	4
80	.99914	.002	-2.724	10.438	.98684	4
100	.99928	.002	-2.703	10.309	.98958	4
120	.99944	.002	-2.755	10.417	.99138	4
140	.99954	.001	-2.830	10.888	.99265	4
160	.99959	.001	-2.678	9.835	.99359	4
180	.99962	.001	-2.749	10.474	.99432	4
200	.99966	.001	-2.815	11.019	.99490	4
			d'			
20	.99014	.024	-2.596	10.931	.88235	5
40	.99612	.010	-2.701	10.785	.94595	4
60	.99761	.007	-2.786	10.990	.96491	4
80	.99828	.005	-2.710	10.283	.97403	4
100	.99858	.004	-2.692	10.190	.97938	4
120	.99888	.003	-2.747	10.330	.98291	4
140	.99908	.003	-2.823	10.811	.98540	4
160	.99918	.002	-2.673	9.775	.98726	4
180	.99925	.002	-2.743	10.412	.98870	4
200	.99932	.002	-2.809	10.957	.98985	4

NOTE: The 5% critical value can be used to test whether a distance between given trees is statistically less than a distance between randomly generated trees. C.N. = class number.

Table 9 exhibits a Pidgin ALGOL procedure computing in $O(n)$ time the term information (TERM) between two trees. Procedures for other indices are equally straightforward.

5. Empirical Studies

H. T. Wareham has written computer programs (available from Day) to compute consensus indices using the algorithms and ideas described herein. One program accepts trees in U_n as input and is written in PASCAL; the other accepts only binary trees in U_n as input and is written in the C programming language. Both programs run on a DEC VAX 11/780 computer equipped with the UNIX 4.2 BSD operating system. The C program

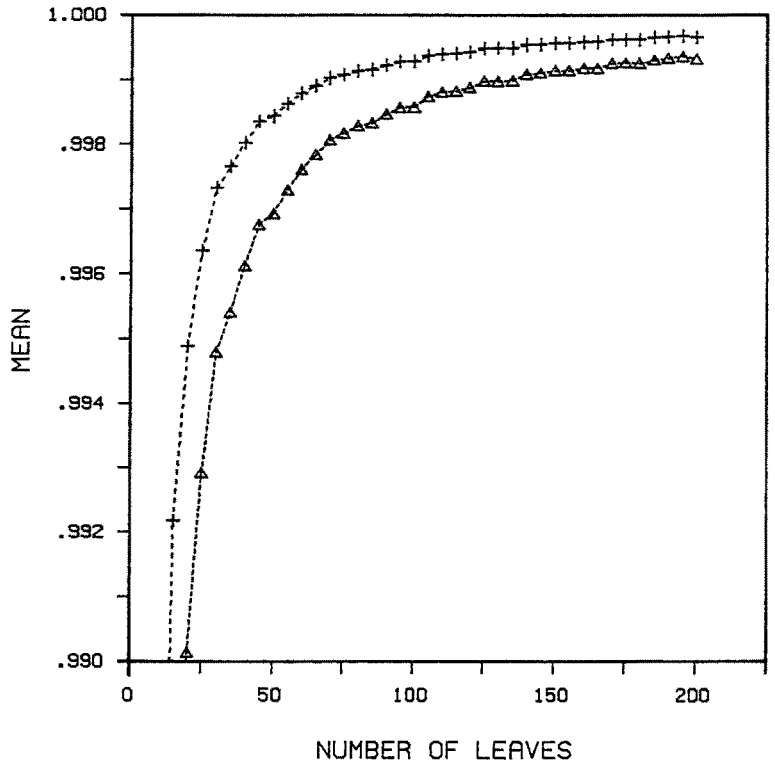


Figure 5. MEANS OF SAMPLES OF 10,000 d AND d' DISTANCES BETWEEN RANDOMLY SELECTED UNROOTED BINARY TREES WITH LABELED LEAVES. Triangles represent d' means.

requires about .24 seconds to compute values of D and S for trees with 200 leaves. Figure 4 displays average times the C program requires to compute these indices between trees of various sizes; these data exhibit the program's linear expected time complexity.

For binary trees T_1 and T_2 in U_n, Hendy, Little and Penny (1984) conjecture that the probability that $d'(T_1, T_2) = 1$ approaches 1 as $n \to \infty$. To investigate this conjecture, Wareham computed d and d' distances between 10,000 pairs of randomly selected binary trees in U_n for each $n = 5(5)200$. Table 10 exhibits summary statistics for the empirical distributions. Figure 5 displays d and d' means for the distributions; these data support the conjecture.

6. Conclusion

I described an optimal algorithm to compute in $O(kn)$ time the strict consensus of k trees with n labeled leaves. It can be used to compute in $O(n)$ time well-known indices of consensus between trees with labeled

leaves. It can be employed to investigate sampling distributions of consensus indices based on the strict consensus tree. It is based on two complementary design ideas: to relabel tree leaves so that every tree cluster has a concise description; and to encode these descriptions in a cluster table so that cluster membership can be tested in constant time. This design suggests several problems for research. Can the PSW representation of a tree be constructed in $O(n)$ time from its cluster table? If so, it may be possible to compute M_l consensus trees in $O(k^2n)$ time; is this result then optimal? Can the two design ideas lead to improved algorithms for consensus tree or index methods other than those already mentioned? The leaf relabeling idea may lead to an efficient algorithm for a *quartet* distance measure recently proposed by G. F. Estabrook, F. R. McMorris and C. A. Meacham (personal communication).

References

ADAMS, E. N., III (1972), "Consensus Techniques and the Comparison of Taxonomic Trees," *Systematic Zoology, 21*, 390-397.

AHO, A. V., HOPCROFT, J. E., and ULLMAN, J. D. (1974), *The Design and Analysis of Computer Algorithms*, Reading, Massachusetts: Addison-Wesley.

BOURQUE, M. (1978), "Arbres de Steiner et Réseaux dont Certains Sommets sont à Localisation Variable," Ph.D. dissertation, Université de Montréal, Quebec, Canada.

BROWN, E. K., and DAY, W. H. E. (1984), "A Computationally Efficient Approximation to the Nearest Neighbor Interchange Metric," *Journal of Classification, 1*, 93-124.

CAVALLI-SFORZA, L. L., and EDWARDS, A. W. F. (1967), "Phylogenetic Analysis Models and Estimation Procedures," *American Journal of Human Genetics, 19*, 233-257.

COLLESS, D. H. (1980), "Congruence between Morphometric and Allozyme Data for *Menidia* Species: A Reappraisal," *Systematic Zoology, 29*, 288-299.

DAY, W. H. E. (1983), "The Role of Complexity in Comparing Classifications," *Mathematical Biosciences, 66*, 97-114.

HARARY, F. (1969), *Graph Theory*, Reading, Massachusetts: Addison-Wesley.

HENDY, M. D., LITTLE, C. H. C., and PENNY, D. (1984), "Comparing Trees with Pendant Vertices Labelled," *SIAM Journal on Applied Mathematics Theory, 44*, 1054-1065.

MARCZEWSKI, E., and STEINHAUS, H. (1958), "On a Certain Distance of Sets and the Corresponding Distance of Functions," *Colloquium Mathematicum, 6*, 319-327.

MARGUSH, T. (1982), "Distances Between Trees," *Discrete Applied Mathematics, 4*, 281-290.

MARGUSH, T., and McMORRIS, F.R. (1981), "Consensus n-Trees," *Bulletin of Mathematical Biology, 43*, 239-244.

McMORRIS, F.R., MERONK, D.B., and NEUMANN, D.A. (1983), "A View of some Consensus Methods for Trees," in *Numerical Taxonomy: Proceedings of a NATO Advanced Study Institute*, ed. J. Felsenstein, Berlin: Springer-Verlag, 122-126.

McMORRIS, F.R., and NEUMANN, D. (1983), "Consensus Functions Defined on Trees," *Mathematical Social Sciences, 4*, 131-136.

MICKEVICH, M.F. (1978), "Taxonomic Congruence," *Systematic Zoology, 27*, 143-158.

NELSON, G. (1979), "Cladistic Analysis and Synthesis: Principles and Definitions, with a Historical Note on Adanson's *Familles des Plantes* (1763-1764)," *Systematic Zoology, 28*, 1-21.

NELSON, G., and PLATNICK, N. (1981), *Systematics and Biogeography: Cladistics and Vicariance,* New York: Columbia University Press.

NEUMANN, D.A. (1983), "Faithful Consensus Methods for n-Trees," *Mathematical Biosciences, 63,* 271-287.

RESTLE, F. (1959), "A Metric and an Ordering on Sets," *Psychometrika, 24,* 207-220.

ROBINSON, D.F. (1971), "Comparison of Labeled Trees with Valency Three," *Journal of Combinatorial Theory, 11,* 105-119.

ROBINSON, D.F., and FOULDS, L.R. (1981), "Comparison of Phylogenetic Trees," *Mathematical Biosciences, 53,* 131-147.

ROHLF, F.J. (1982), "Consensus Indices for Comparing Classifications," *Mathematical Biosciences, 59,* 131-144.

ROHLF, F.J. (1983), "Numbering Binary Trees with Labeled Terminal Vertices," *Bulletin of Mathematical Biology, 45,* 33-40.

SCHUH, R.T., and FARRIS, J.S. (1981), "Methods for Investigating Taxonomic Congruence and Their Application to the Leptopodomorpha," *Systematic Zoology, 30,* 331-351.

SHAO, K. (1983), "Consensus Methods in Numerical Taxonomy," Ph.D. dissertation, State University of New York, Stony Brook, New York.

SOKAL, R.R., and ROHLF, F.J. (1981), "Taxonomic Congruence in the Leptopodomorpha Re-examined," *Systematic Zoology, 30,* 309-325.

STANDISH, T.A. (1980), *Data Structure Techniques,* Reading, Massachusetts: Addison-Wesley.

STINEBRICKNER, R. (1984), "s-Consensus Trees and Indices," *Bulletin of Mathematical Biology, 46,* 923-935.

TATENO, Y., NEI, M., and TAJIMA, F. (1982), "Accuracy of Estimated Phylogenetic Trees from Molecular Data I. Distantly Related Species," *Journal of Molecular Evolution, 18,* 387-404.

WATERMAN, M.S., and SMITH, T.F. (1978), "On the Similarity of Dendrograms," *Journal of Theoretical Biology, 73,* 789-800.

WEIDE, B. (1977), "A Survey of Analysis Techniques for Discrete Algorithms," *Computing Surveys, 9,* 291-313.