Efficient Algorithms for Inferring Evolutionary Trees*

Dan Gusfield
Computer Science Department, University of California, Davis

Sometime in 1991

Abstract

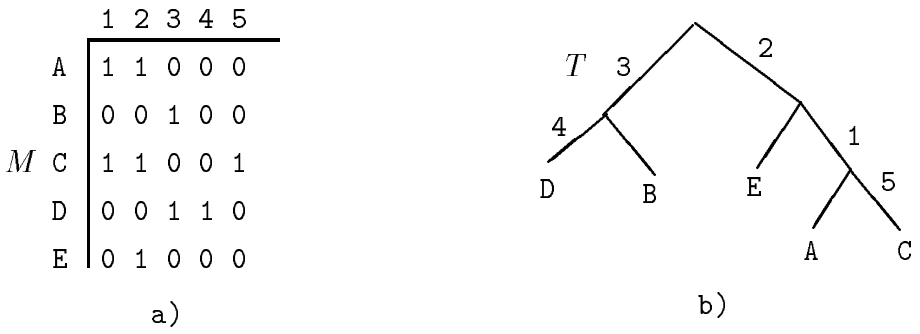
In this paper we examine two related problems of inferring the evolutionary history of n objects, either from present characters of the objects, or from several partial estimates of their evolutionary history. The first problem is called the *Phylogeny* problem, and second is the *Tree Compatibility* problem. Both of these problems are central in algorithmic approaches to the study of evolution [F], [WI], and in other problems of historical reconstruction [HKT], [NA], [NI]. In this paper we show that both of these problems can be solved by graph theoretic methods in linear time, which is time optimal, and which is a significant improvement over existing methods.

1 Perfect Phylogeny

Let M be an n by m 0-1 matrix representing n objects in terms of m characters that describe the objects; cell (i, j) of M has a value of one if and only if object i has character j. A *phylogenetic tree* for M is a rooted tree T where each object is attached to exactly one leaf of T, where each of the m characters is associated with exactly one edge of the tree and where, for any leaf w of T, the characters associated with the edges along the unique path from the root to w exactly specify the character vector of the objects at leaf

*Appeared in Networks, Vol. 21 (1991) p. 19-28

w. In the example in Figure 1, the first matrix M has a phylogenetic tree T, but the second matrix M' does not.



M'	1	2	3	4	5
A	1	1	0	0	0
B	0	0	1	0	1
C	1	1	0	0	1
D	0	0	1	1	0
E	0	1	0	0	1

c)

Figure 1: Matrix M has a phylogenetic tree T. Matrix M' has none.

The interpretation of a phylogenetic tree is that it gives an estimate of the evolutionary history (in terms of branching pattern, but not time) of the objects, based on the following biological assumptions:

1. The root of the tree represents an ancestral object which has none of the present m characters. That is, in the ancestral object, the state of each character is zero.

2. Each of the characters changes from the zero state to the one state exactly once, and never from the one state to the zero state.

The key feature of a phylogenetic tree (without which there would be no interesting problem) is that each character is associated with exactly one edge of the tree. This corresponds to the second assumption above, and represents the point in the evolutionary history of the objects when the character changes from its zero state to its one state. Hence any objects below that edge definitely have that character. We assume that characters satisfying these biological assumptions are known, but point out that finding such characters is a difficult biological task in building a phylogenetic tree.

Phylogeny Problem: Given the n by m 0-1 matrix M, determine whether there is a phylogenetic tree for M, and if so, build one.

We solve this problem with a very simple $O(nm)$ time algorithm, where each comparison operation and each reference to M is counted as one time unit. Existing methods, based on a straightforward implementation of Lemma 1 below, take time $\Theta(m^2n)$ [W], [CS]. Even when M is known to have a phylogeny, existing methods to construct it (based on clustering [H]) take time $\Theta(n^2m)$. Typically, $n << m$, and the phylogeny problem is often at the inner loop of more complex problems, so the improvement in running time in this paper is significant. An easy adversary argument shows that in order to build a phylogenetic tree every element in M must be examined. We will extend this to show that just to *decide* whether M has a phylogenetic tree, every element must be examined, hence our algorithm is time optimal. We first need the following definition and lemma.

Definition: For any column k of M, let O_k be the set of objects with a one in column k, i.e. the objects which have character k.

Lemma 1: M has a phylogenetic tree if and only if for every pair of columns i, j, either O_i and O_j are disjoint or one contains the other.

The proof of the Lemma is straightforward and appears in a number of places [EJM75] [M]. It is the basis for existing algorithms for the phylogeny problem. These algorithms build T top down, inserting the characters in sorted order (largest to smallest) according to the number of objects that have that character, checking along the way that the conditions of the lemma are satisfied. Note that an algorithm based on a straightforward implementation of Lemma 1, would take $\Omega(nm^2)$ time just to determine if M has a phylogeny.

1.1 An $O(nm)$ Algorithm to Test for a Phylogenetic Tree

1. Considering each *column* of M as a binary number (with the most significant bit in row 1), sort these numbers into decreasing order, placing the largest number in column 1. As a result, all duplicate copies of a column become placed together in a consecutive block of columns.

2. Delete any column that is identical to the column on its right. Call the transformed matrix at this point M'.

3. Let O be the set of all cells in M' with value one. For each cell $(i,j) \in O$, set $L(i,j)$ equal to the largest index $k < j$ such that $M'(i,k) \in O$; set $L(i,j)$ to 0 if there is no such index k. For each column j, set $L(j)$ equal to the largest $L(i,j)$ such that $(i,j) \in O$.

4. Check whether $L(i,j) = L(j)$ for every cell $(i,j) \in O$. If so then M has a phylogenetic tree, otherwise M does not have one.

Theorem 1: Matrix M has a phylogenetic tree if and only if $L(i,j) = L(j)$ for every cell $(i,j) \in O$.

Proof: Let b_k denote the binary number represented by column k of M. If $O_j \subset O_k$ then $b_j < b_k$, so column k will be to the left of column j in M'. Now suppose that for some column j, $L(i,j) = k = L(j)$, and $L(i',j) = k' < k$. Then $O_j \cap O_k \neq \emptyset$, and O_j is not a subset of O_k, since cell (i',j) has value one but cell (i',k) has value zero. So by Lemma 1, M has no phylogeny unless $O_k \subset O_j$. But this is impossible, since $k < j$ in M'. This proves the necessary side of the theorem.

Conversely, suppose that $L(i,j) = L(j)$ for every cell $(i,j) \in O$. First, for any column j, $O_j \cap O_p = \emptyset$ for any p strictly between $L(j)$ and j. So such pairs of columns j and p satisfy the conditions of Lemma 1. Second, if $L(j) \neq 0$ then $O_j \subset O_{L(j)}$.

Now for an arbitrary, but fixed column j, let $L(j) = k > 0$, and let $L(k) = k'$. If $k' > 0$ then it follows that $O_j \subset O_{k'}$, and that $O_j \cap O_p = \emptyset$ for any p strictly between k' and k. If $k' = 0$ then $O_j \cap O_p = \emptyset$ for any p from 1 to $k-1$. Continuing in this way, we see that columns j and p satisfy Lemma 1 for every $p < j$. Since j was arbitrary, all pairs of columns in M' satisfy the conditions of Lemma 1, and M has a phylogenetic tree. □

The algorithm can be implemented in time $O(nm)$ by using radix sort in step 1, with pointers to avoid all but the last column permutations (see [AHU] for details on $O(nm)$ time radix sort). All other operations in the

algorithm are trivially done in that time bound.

Unlike the previous algorithms for this problem, the sorting done in step 1 does not sort the columns according to the number of ones they contain; its purpose is to put column k to the left of column j whenever $O_j \subset O_k$, and to collect together any identical columns.

1.2 Constructing a Phylogeny

The algorithm above decides whether M has a phylogeny. If it does, then the following $O(nm)$ time algorithm constructs one, taking M' as input.

1. Create a node n_j for every column j of M'. For each node n_j such that $L(j) > 0$, direct an edge from $n_{L(j)}$ to n_j, and label the edge with character j and the indexes of all columns identical to column j (which were deleted in step 2). Create a root node r, and for each node n_j such that $L(j) = 0$, direct an edge from r to n_j labeled with character j, and the indexes of all columns identical to j.

2. For each row i, let c_i be the largest index such that cell (i, c_i) has value one in M', and let e be the edge labeled with character c_i. If the head of edge e is a leaf, then attach object i to that leaf. If the head of e is not a leaf, then create a new edge directed from the head of e and attach i to the new leaf created. The resulting directed tree T is a phylogenetic tree for matrix M.

Theorem 2: The above algorithm correctly builds a phylogenetic tree T for M.

Proof: Note first that there is a directed path in T from a node n_j to node n_k if and only if $O_k \subset O_j$, and hence along that path the size of the characters ($|O_k|$ for column k) strictly decrease. Now for any object i, character c_i is the smallest character that i has (that is $|O_{c_i}| < |O_k|$ for every k such that i has character k), so the path from object i to the root of T encounters all the characters of i. Further, each character is attached to exactly one edge of T, so T is a phylogenetic tree for M. □

Definition: A column of a binary matrix has the *consecutive ones property* if and only if all the ones in the column are in consecutive rows, where rows n and 1 are not considered to be consecutive.

It is known [H] that *if* M has a phylogenetic tree then the rows of M can be permuted so that every column has the consecutive ones property. The consecutive ones property allows a visually nicer presentation of the

matrix. An $O(n^2m)$ algorithm to reorder the rows was given in [H]. Linear time, but fairly complex, algorithms are known for creating, when possible, the consecutive ones property [BL], but for matrices with a phylogenetic tree there is a simpler linear time method.

> Execute line 1 of the algorithm to test for a phylogenetic tree, and then execute the following: 1.5) Considering each *row* of M as a binary number, use *radix* sort to sort these numbers in decreasing order, placing the smallest number in row 1.

Theorem 3: If M has a phylogenetic tree then after steps 1) and 1.5), each column of M has the consecutive ones property.

Proof: The proof is by induction on the number of columns that radix sort has examined in step 1.5. Radix sort (of the rows of M) examines the columns in order m down to 1; when it examines column k, it *stably* permutes the rows of M so that all the rows with a zero in column k are above all the rows with a one in column k. A stable permutation of rows (for column k) does not change the relative order of two rows which have the same element in column k.

Clearly, when a particular column k is examined, the row permutations create the consecutive ones property in column k, so we need to show that the consecutive ones property of any columns to the right is preserved by those permutations. Assume that the columns down to $k+1$ have the consecutive ones property, and consider the affect on column $j > k$ of the row permutations resulting when column k is examined. If $O_k \cap O_j = \emptyset$, then every row with a 1 in column j has a zero in column k, and since the permutations are stable, these consecutive rows will stay together as a consecutive block, so column j will retain the consecutive ones property. If $O_k \cap O_j \neq \emptyset$ then $O_j \subseteq O_k$, so the consecutive rows containing ones in column j also contain ones in column k. Since the row permutations (for column k) are stable, these consecutive rows remain together, and so column j retains the consecutive ones property. □

1.3 $\Omega(nm)$ time is needed for the phylogeny problem

We show that any algorithm to decide if M has a phylogenetic tree must, in worst case, examine every cell in M. The proof is by the so called *adversary*

method, in which we show how an adversary can give consistent answers to the queries of any unknown algorithm, forcing the algorithm to query every cell of M. If the algorithm does not query every cell, then the adversary can fill in unqueried cells, so as to make the algorithm's answer incorrect. A query is specified by the location of a cell, and an answer is either 0 or 1, indicating the value at that cell. The adversary decides which answers to give the algorithm as follows.

1. Answer 1 to each query until a query is made to a cell (i, j) such that all *other* unqueried cells are either in a single column or a single row. Answer 0 for the query to cell (i, j).

2. Suppose that all unqueried cells are in column k. Answer 1 to any remaining query (these are all in column k) as long as column k has at least three unfilled cells. When column k has only two unfilled cells, let (r, k) denote the cell being queried. If $r = i$ (the same i as from step 1) then answer 1, else answer 0. We will prove that in either case the last unfilled cell must be queried.

3. Suppose that all unqueried cells are in row q. Answer 1 to any remaining query (in row q) as long as row q has at least three unfilled cells. When row q has only two unfilled cells, let (q, z) be the cell being queried. If $z = j$ (the same j as from step 1) then answer 1, else answer 0. We will prove that the last unfilled cell must be queried.

Theorem 4: The above adversary forces a query of every cell of M in order to determine whether M has a phylogenetic tree.

Proof: First, if the algorithm stops before step 1 finishes, then all the entries are 1 and there are at least two unqueried cells (i, j) and (s, t) which are not in the same row or column. If we set these cells to 0 and all other cells to 1 then, by Lemma 1, M has no phylogenetic tree, while if all cells are set to 1, then it does. So the algorithm must run until the adversary finishes step 1. In steps 2 and 3 we maintain the condition that the matrix has only one 0 entry until there are only two unfilled cells left. Up to that point the algorithm cannot correctly conclude that there is no phylogeny for, by Lemma 1, there must be at least two 0 entries in a matrix with no phylogeny. Further, one of the unqueried cells is in neither the same row or column as cell (i, j), so it is possible to make that cell 0 and the other 1, and so the

algorithm cannot correctly conclude that a phylogenetic tree must exist for M. So any correct algorithm must make at least these $nm-2$ queries.

If the answer to query $nm-1$ is 1, then there is only one 0 in the matrix and the remaining unfilled cell is neither cell (i,k) nor (q,j). By Lemma 1, M has a phylogeny if and only if this remaining cell is filled with a 1. If the answer to query $nm-1$ is 0, then the last unfilled cell is either (i,k) or (q,j). In either case, by Lemma 1, M has a phylogeny if and only if the remaining cell is filled with a 0. Hence the algorithm must explicitly examine the remaining cell. □

One might object that the adversary above creates a very unnatural matrix since most of the objects are identical. It remains an open question whether the $\Theta(mn)$ lower bound holds when all objects have a distinct set of characters.

1.4 Extension to Undirected Characters

In the phylogeny problem discussed so far, it was assumed that each character has two states, 0 and 1, and that at the root of the phylogeny, each character is in state 0. The second assumption is often too strong. A weaker assumption is that each character is binary, but it is unknown which state of each character is ancestral. However, once choices for the root states are made, the states can be relabeled so that the root states are all zero, and the resulting matrix can be tested to see if it has a phylogenetic tree. So the more general problem is to determine if there is a phylogenetic tree for at least one of the 2^m choices for the root character states. An intuitive choice for the root state of a character j is the majority state: assign the root state of character j to be 1 if and only if $|O_j| \geq n/2$. It was shown [MC] that if there is any choice of root states that leads to a phylogenetic tree, then the majority choice for each character will also. Hence the general problem reduces to one instance of the phylogeny problem already discussed, and so the general problem can also be solved in $O(nm)$ time.

2 Determining the Compatibility of Two Trees

Another extension of the phylogeny problem that has been given considerable attention [F],[EJM75,76a,76b][EM80], is how to determine whether two

phylogenetic trees describe compatible evolutionary history, and if so, how to combine them into a single phylogenetic tree incorporating all the known history. Such a problem arises when the two trees are first constructed separately, using different characters. In [EM80] it was shown that this problem reduces to an a single instance of the phylogeny problem in a table of size at most n by $4n$, and hence the problem can be solved in $O(n^2)$ time, as above. In this section we show how to solve the problem in $O(n)$ time.

2.1 Definitions

Let T_1 and T_2 be two phylogenetic trees for a set of n objects. We will assume that T_1 and T_2 are both in "reduced form", that is, both are binary trees, and no node except the root can have exactly one child. A phylogenetic tree T_3 is a *refinement* of T_1 if T_1 can be obtained by a series of edge contractions of edges of T_3. If T_3 refines T_1, then T_3 agrees with all the evolutionary history displayed in T_1, while displaying additional history not contained in T_1. Trees T_1 and T_2 are *compatible* if there exists a phylogenetic tree T_3 refining both T_1 and T_2 (see Figure 2). Given trees T_1 and T_2, the *tree compatibility* problem is to determine whether the two trees are compatible, and if so, to produce a refinement tree T_3. In a series of papers [EJM75, EJM76] [EM80] the above definition was shown to be equivalent to several other notions of compatibility, and was finally solved as follows [EM80].

Let M_1 be a 0-1 matrix with one row for each object, and one column for each internal node j in T_1. Entry (i, j) of M_1 has value one if and only if object i is found at a leaf of T_1 below node j. That is, column j of M_1 records the objects found in the subtree of T_1 rooted at node j. Matrix M_2 is similarly defined for T_2, and matrix M_3 is the matrix formed by the union of the columns of M_1 and M_2. Then

Theorem 5 [EM80]: T_1 and T_2 are compatible if and only if there is a phylogenetic tree for M_3. Further, a phylogenetic tree T_3 for M_3 is a refinement of both T_1 and T_2.

Given the above theorem, the compatibility problem can be solved in $O(n^2)$ time using the method of section 1. We now show how to solve the problem in $O(n)$ time.

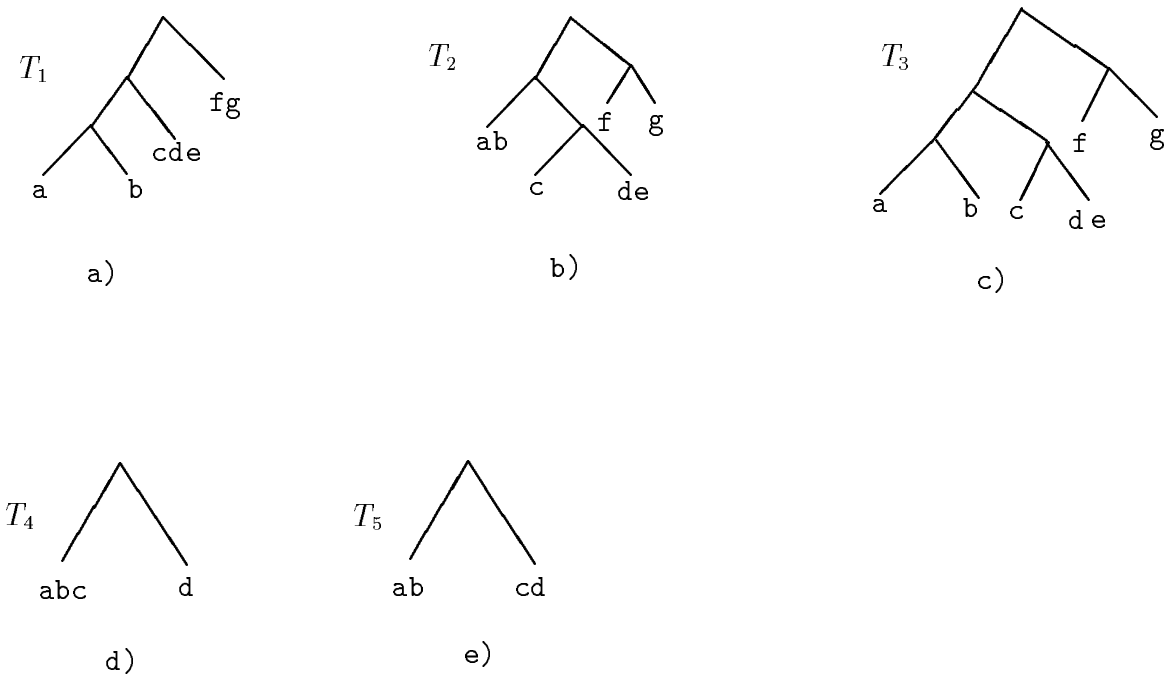


Figure 2: T_1 and T_2 are compatible; they are refined by T_3. T_4 and T_5 are not compatible.

2.2 An $O(n)$ Time Compatibility Algorithm

1. With a depth first traversal of T_1 do the following: set a pointer from each object i to the leaf, $V_1(i)$, that i is attached to; for each object i, set $N_1(i)$ to be the number of objects in the leaf i is attached to; record for each internal node x, the number of objects at the leaves below x. Do the similar traversal of T_2. Declare all objects to be active.

2. For $j = 1, 2$

   begin

   If j = 1, then set k = 2, else set k = 1.

   While there is an active object i such that $N_j(i) > N_k(i)$, do the following:

   Follow the path from $V_k(i)$ to the root of T_k until a node x is reached that has at least $N_j(i)$ leaves in its subtree $T(x)$. Check that the objects at the leaves of $T(x)$ are the same objects attached to $V_j(i)$. If not, then stop: T_1 and T_2 are not compatible. Otherwise, replace node $V_j(i)$ in T_j with $T(x)$, update the pointers $V_j(i)$, and declare each object in $T(x)$ to be inactive.

   end.

3. Considering the set of objects at a leaf to be its label, check whether the labeled trees T_1 and T_2 are isomorphic, and if so, then $T_3 = T_1 = T_2$ is the desired refinement.

The proof of correctness is straightforward and left for the reader.

2.2.1 Implementation and Analysis

Step 1 takes $O(n)$ time, since all the tasks are done while doing a depth first search traversal [AHO]. We will implement step 2 so that each iteration takes time $O(N_j(i))$, and hence the total time of step 2 is $O(n)$. To check if the objects in $T(x)$ are the same as those in $V_j(i)$, first check that the number of objects at the leaves of $T(x)$ is exactly $N_j(i)$. If so, traverse $T(x)$ to identify the leaves of $T(x)$. Then for every object p attached to $V_j(i)$ check, in unit time, that $V_k(p)$ is a leaf in $T(x)$. If all $V_j(i)$ objects are in $T(x)$, then the two sets of objects are the same, since their numbers are equal. Tree $T(x)$ has size

$O(N_j(i))$ since T_j is in reduced form, so step 2 takes only $O(n)$ time in total. For step 3, it is known [AHU] that isomorphism testing of general (labeled or unlabeled) trees can be done in $O(n)$ time, although the algorithm is a little involved. However, there are very simple isomorphism tests for step 3, due to the fact that the trees are binary, and that the objects at the leaves are distinct. The following is one of several simple ways to implement step 3:

3a. Traverse the trees to assign the distance of each node from the root, and to identify the leaves L of T_1 whose siblings are also leaves.

3b. Repeat step 3c until only the root node of T_1 remains.

3c. Choose a node v in L and remove it from L. Find the leaf x in T_2 with the same label as v, check that its sibling is a leaf, and that it has the same label as the sibling of v. If there is no such x, or any of the checks fail, then the trees are not isomorphic. Otherwise, remove these four nodes from their respective trees, making the respective parents of v and x (denoted p_1 and p_2) leaf nodes. Label p_1 and p_2 with the same unused number from $n+1$ to $2n$. Check whether the sibling of p_1 is a leaf (using the distances), and if so, add p_1 to L.

Implementation detail for step 3c

To find x given v we first check if v is an original leaf of T_1. If so, then pick any object i attached to v (hence $v = V_1(i)$); the only candidate for x is $V_2(i)$, and we can check the label of x in $O(|V_2(i)|)$ time as in the implementation of step 2. If v is not an original leaf of T_1, then it is labeled with a single number, so an index vector can locate x from v in unit time.

Acknowledgement

Thanks to Chip Martel for suggesting the particular isomorphism test given above.

References

[AHO] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[BL] K. Booth, G. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms", JCSS, vol. 13, 1976 335-379.

[CS] J. Camin, R. Sokal, "A method for deducing branching sequences in phylogeny". Evolution 19, 311-326, 1965.

[EJM] G. Estabrook, C. Johnson, F. McMorris, "A Mathematical Foundation for the Analysis of Cladistic Character Compatibility". Math. Bioscience, 29, 1976.

[EJM] G. Estabrook, C. Johnson, F. McMorris (1975), "An idealized concept of the true cladistic character". Math Biosci, 23:263-272.

[EJM] G. Estabrook, C. Johnson, F. McMorris (1976), "An algebraic analysis of cladistic characters", Discrete Math, 16:141-147.

[EM] G. Estabrook, F. McMorris (1977), "When are two qualitative taxonomic characters compatible?" J. Math Biosci, 4:195-200.

[EM] G. Estabrook, F. McMorris (1980) "When is one estimate of evolutionary relationships a refinement of another?", J. Math Biosci, 10:367-373.

[F] J. Felsenstein, "Numerical Methods for Inferring Evolutionary Trees". The Quarterly Review of Biology, Vol. 57, No. 4, Dec. 1982.

[F2] J. S. Farris, "Inferring Phylogenetic Trees from Chromosome Inversion Data", Systematic Zoology 28, 1967.

[FM] W. M. Fitch and E. Margoliash, "The Construction of Phylogenetic Trees", Science, 155, 1967.

[G] D. Gusfield, "The Steiner Tree Problem in Phylogeny", Technical Report 332 - Yale University, Department of Computer Science, September 1984.

[H] J. Hartigan. Clustering Algorithms. Wiley, 1975.

[HKT] Hodson, Kendall, and Tautu. Mathematics in the Archeological and Historical Sciences. University Press, Edinburgh, 1970.

[M] C. Meacham, "Theoretical and computational considerations of the compatibility of qualitative taxonomic characters", Nato ASI series vol. G1 on Numerical Taxonomy, Springer-Verlag 1983.

[MC] F. McMorris, "On the compatibility of binary qualitative taxonomic characters", Bull. Math Biol., 39: 133-138, 1977.

[NA] D. Najok, "Principles and Modifications of Local Genealogical Algorithms in Textual History", Computers and the Humanities 14 (1980).

[NI] S. C. Nita, "Establishing the Linkage of Different Variants of a Romanian Chronical" in [HKT].

[WI] E. O. Wiley, Phylogenetics: The Theory and Practice of Phylogenetic Systematics. Wiley-Interscience 1981.

[W] E. O. Wilson, "A consistency test for phylogenies based on contemporaneous species", Systematic Zoology, 14: 214-220.