BIOINFORMATICS

Vol. 16 no. 2 2000
Pages 140–151

Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees

P. W. Purdom, Jr[1], P. G. Bradford[2], K. Tamura[3] and S. Kumar[4,5]

[1]Computer Science Department, Indiana University, Bloomington, IN 47405-4101, USA, [2]BlackRock Inc., 345 Park Avenue, New York, NY 10154, USA, [3]Department of Biology, Tokoyo Metropolitan University, 1-1 Minami-ohsawa, Hachioji-shi, Tokyo 192-03, Japan, [4]Institute of Molecular Evolutionary Genetics, 311 Mueller Laboratory, University Park, PA 16802, USA and [5]Department of Biology, Arizona State University, Tempe, AZ 85287-1501, USA

Received on June 27, 1998; revised on June 10, 1999; accepted on July 20, 1999

Abstract

Motivation: *In the maximum parsimony (MP) method, the tree requiring the minimum number of changes (discrepancy) to explain the given set of DNA or amino acid sequences is chosen to represent their evolutionary relationships. To find the MP tree, the branch-and-bound algorithm is normally used. For a partial phylogenetic-tree (one that has a subset of the organisms) the traditional algorithm assigns a cost equal to the discrepancy of the partial phylogenetic-tree. We propose a single column discrepancy heuristic which increases this cost by predicting a minimum additional discrepancy needed to attach the sequences yet to be added to the partial phylogenetic-tree. A dynamic Max-mini order of sequence addition is also proposed to quickly terminate branch-and-bound search paths that are guaranteed to lead to suboptimal solutions.*

Results: *We studied the running time of 47 problems generated from 17 data sets. The use of single column discrepancy heuristic speeded up the computation to 2.4-fold for static and 18.2-fold for dynamic search order. The improvement appeared to increase exponentially with the number of sequences. The proposed strategies are also likely to be useful in speeding up the MP tree search using heuristic searches that are based on branch-and-bound-like algorithms.*

Contact: *s.kumar@asu.edu*

Introduction

The maximum parsimony method of phylogenetic inference attempts to determine the evolutionary relationship among a set of organisms based on the fit of the data to the phylogenetic tree that requires the minimum number of changes (Fitch, 1971; Hartigan, 1973). The input is a set of organisms each with an associated string of characters. The string is a DNA sequence or an amino acid sequence. In these sequences, the most common evolutionary change is to substitute a character at a particular place in a sequence with another, but sometimes the change is an insertion or a deletion of characters. Therefore, it is necessary to align the sequences (Taylor, 1996) to ensure that characters at site k are evolutionarily homologous in all sequences. The algorithms in this paper assume that the sequences are already aligned.

A binary tree can be used to represent the evolutionary history of a set of organisms with their associated molecular sequences. Leaves represent the organisms and sequences of the data sets and internal nodes represent ancestral organisms and their sequences. For sake of simplicity, we work with only strictly bifurcating trees in which only two-way splits occur. The root corresponds to the first such split.

The maximum parsimony algorithms under discussion do not find the root of the phylogenetic tree, because the characters (nucleotide or amino acid) are unordered. Therefore, the algorithms represent a possible relationship among organisms with an unrooted binary tree. Each internal node has *three* neighbors. Each leaf corresponds to an organism from the input. For n organisms, there are $(2n-5)!!$ such trees, where $(2n-5)!!$ is the product of the odd numbers from 1 to $2n-5$.

In the parsimony method, each leaf of the candidate phylogenetic tree is labelled with the sequence of its organism. To understand the principle of maximum parsimony, assume that each internal node has also been labelled with some sequence, where all sequences have the same length. For each edge of the phylogenetic tree consider the labels associated with the nodes at each end. For each position k, assign a score of one if the two labels are different and a score of zero if they are the same (Fitch, 1971). The

discrepancy for the particular phylogenetic tree and the particular labelling is the sum of this score over all positions and over all edges. The *discrepancy of a phylogenetic tree* is the minimum discrepancy over all possible labellings of that tree. Under the maximum parsimony criterion, the phylogenetic tree with the least discrepancy is the preferred candidate for how the organisms are actually related to each other.

The best labelling for a particular phylogenetic tree can be determined in linear time (Fitch, 1971; Hartigan, 1973). However, the problem of determining the best phylogenetic tree is NP-complete (Day *et al.*, 1986; Graham and Foulds, 1982). Branch-and-bound algorithms are normally used to find the optimum solution(s) (Felsenstein, 1993; Hendy and Penny, 1982; Kumar *et al.*, 1993; Swofford, 1998). For large problems, heuristic search methods are used (Felsenstein, 1993; Kumar *et al.*, 1993; Swofford, 1998). They provide speed, but do not guarantee finding optimal solutions. The methods proposed can be applied to either type of computation, but this paper concentrates on branch-and-bound algorithms.

Branch-and-bound algorithm

The basic form of branch-and-bound parsimony algorithms is now given. The input consists of a set of organisms, where each organism has a name and a sequence. The algorithm presented here retains this information in global variables. Many bookkeeping and efficiency details are omitted to help bring out the basic ideas. The variables with 'tree' in their name relate to partial phylogenetic trees (evolutionary trees for some subset of the organisms).

The main program is:

Parsimony

1. Select the first three organisms and form *current_tree* by joining those organisms into an unrooted binary tree (there is only one such tree). Set *best_cost* to infinity.
2. Call **Add_organism**(*current_tree*)

The global variable *best_cost* keeps track of the discrepancy of the best solution that has been found.

The main program calls the following recursive routine.

Add_organism(*current_tree*)

A1. Set *cost* to **Cost**(*current_tree*). If *cost* $>$ *best_cost* then return.

A2. If all organisms are in *current_tree* then

- A2.1 Set *best_cost* to *cost*. (Use *cost* $-$ 1 if only one example of the best tree is wanted.)
- A2.2 Output *cost* and *current_tree*.
- A2.3 Return.

A3. Select an organism i that is not in *current_tree*.

A4. For each branch j in *current_tree*

- A4.1 Form *tree* by attaching organism i to branch j in a copy of *current_tree* (creating a new internal node).
- A4.2 Call **Add_organism**(*tree*).

The presentation of the algorithm leaves some details unspecified: which three organisms to select in Step A1, which organism to select in Step A3, and which order to try the attachment points in Step A4. The choices made have no effect on correctness, but they can have a major effect on running time. We consider two particular versions of the algorithm: one that uses a static sequence addition order, and the other that uses a dynamic order. The static order program is an improved version of the algorithm included in the first version of the MEGA program, which uses the Max-mini algorithm described in Kumar *et al.* (1993). It is referred to as the static order program from now on. Similar algorithms are also available in the program PAUP* (Swofford, 1998).

At Step A1, both static and dynamic orders select the two sequences that have the largest number of differences in their sequences. They then select a third sequence which has the most positions where it does not agree with either of the first two organisms. This gives a high cost starting phylogenetic tree, which helps speed up the branch-and-bound algorithm.

The most important difference between the static and dynamic orders is how an organism is selected in Step A3. In the static order, sequences are added to the phylogenetic tree in a fixed order. The dynamic search program recomputes the least-cost attachment point for each sequence each time and selects the sequence with the highest such cost. Ties are broken by selecting the organism with the highest second-least-cost, remaining ties are broken using third-least-costs, etc.

At Step A4, each program would like to attach an organism to the point that results in the least increase in the cost, then to the one that results in the second least increase, etc. The dynamic search order program does precisely that. To save time, the static search program does the attachment-order calculation just once (the first time it tries to attach the organism) and then always uses that attachment-order for the organism in each branch of the search tree. Although both programs do calculations for unrooted phylogenetic trees, they use a representation rooted on the internal node of the initial three-node phylogenetic tree. This root has no biological significance, but its use simplifies the programming.

The running time of these algorithms is affected by the number of nodes visited during the search and by the time spent at each node. For the static order, the time per node is determined mainly by the time needed to compute the cost of *current_tree*, which is done using the algorithm based on Fitch (1971). A set of some of the optimum sequences is computed for each internal node. The calculation uses a sequence where each element is a set of characters rather than a single character. An upper bound on the time per node is proportional to the product of the number of positions (*columns*) in a sequence, the number of organisms in *current_tree*, and to the number of different characters that occur in a column. Since only a small number of characters are possible at a position for molecular sequences (four for nucleotide sequences and 20 for amino acid sequences), the complete set of characters of a site at any node in the phylogenetic tree can be packed into one integer. Since the machine instructions (logical and, or, not, etc.) operate on the bits in parallel, this packing eliminated the last factor from the running time. At each call to the cost routine there has been only one change to *current_tree*, so the static order saves time by redoing the calculation only for the nodes above that one place. Note that the computational speed can be enhanced further by using a byte rather than an integer when nucleotide sequences are used. For the purpose of this paper, all comparisons were done on integers since the main purpose of this paper is to explore the relative efficiencies of the fast algorithms for biological problems. (This resulted in the static program running at about half its normal speed on nucleotide sequences.) The dynamic and static search order programs treated the question mark character (used to indicate that a particular character was not known) and the minus character (used to indicate a deletion) differently. In order to ensure that all programs would be solving the same mathematical problem, all columns in the data that contained these character were removed (except for the column with a minus in the cytc data, which had no effect on the comparisons in the paper).

The dynamic algorithm computes for each node the set of *all* optimum sequences using an algorithm based on Fitch (1971) and Maddison and Maddison (1993). From this information the cost of attaching a single organism at any point in the phylogenetic tree can be quickly computed. An upper bound on the time per node is proportional to the product of the length of the sequences, the number of organisms in *current_tree*, and to the number of different characters in a column. Again, word parallelism is used to eliminate the last factor. The running time per node for the dynamic search order is several times larger than the running time for static order because additional information is computed. The dynamic algorithm does these calculations incrementally, so that only positions that might change are recomputed. The steps required to carry out this procedure has a good bit of overhead, but it nearly always saves time.

Dynamic search implementation differs from the static search implementation in one additional way. In the static search implementation, first an upper bound on the cost of the optimal solution is computed by using the Min-mini heuristic search procedure (Kumar *et al.*, 1993). Then the Max-mini principle is used to determine the order in which the sequences will be added to the phylogenetic tree, which yields another upper bound on the cost of the optimal solution. The smaller of the two upper bounds is then used to initialize *best_cost*. This initial upper bound was equal to the true value for 35 of the 47 cases studied. Computing this upper bound takes a negligible amount of time compared with the time taken by the branch-and-bound search that follows. The dynamic search program has two passes. The first pass finds the cost of the optimum solution using the version of **Add_organism** that sets *best_cost* to $cost - 1$. The second pass initializes *best_cost* to the cost of the optimum solution, and then finds all the optimum solutions. When the initial estimate of the cost is correct, using two passes is slower than using one pass; when the initial estimate is high, using two passes is sometimes much faster. Measurements indicated that using two passes was the better approach for the dynamic algorithm (but the dynamic algorithm did not start with a good initial estimate of the cost).

In the dynamic algorithm, we compare every sequence not in *current_tree* with the union of the sequence sets at the two ends of each branch of *current_tree*. An upper bound on the time for this step is proportional to the product of the number of organisms not in the phylogenetic tree, the number of organisms in the phylogenetic tree, the sequence length, and the number of different characters in a column. (Again, word parallelism eliminates the last factor.) This upper bound depends on the second power of the number of organisms whereas the upper bound for static search order depends only on the first power. However, those steps are performed only when the union of the sequence sets changes. This usually results in skipping the most time consuming step, and the measurements indicate that overall it takes about the same time as the rest of the process (rather than dominating the running time).

Single column discrepancy

The particular cost function used by the branch-and-bound algorithm has a major effect on the running time. For correctness, all that is needed is: (1) that the cost function returns some lower bound for the cost of the best phylogenetic tree that can be obtained by adding organisms to *current_tree*; (2) that the cost function returns the true cost when all the organisms are in *current_tree*. To reduce the number of nodes in the search tree one

wants the cost function to be as large as possible, but to be fast, one must balance the quality of the lower bound against the time required to compute it. If one could quickly compute a perfect lower bound, the number of recursive calls would be bounded by the number of solutions times the number of organisms, and parsimony would be extremely fast.

Traditionally, parsimony algorithms use the discrepancy of *current_tree* as the cost function. *Single column discrepancy* increases that bound by adding a lower bound on the cost associated with those organisms that are not yet in the current phylogenetic tree. For each column one computes a *difference set*, the set of characters that occur among the organisms not in *current_tree* but not among the organisms that are in *current_tree*. The single column discrepancy is the sum over all columns of the number of elements in these difference sets. The cost for *current_tree* is the sum of the single column discrepancy and the discrepancy of *current_tree*.

The reason that single column discrepancy gives a correct lower bound is that each element in the difference set will add at least one to the cost of the phylogenetic tree when the associated organism is added to it. A column in an optimum labelling of *current_tree* does not use any character from the difference set for that column. Once the organism associated with a character from the difference set is added, there must be an edge somewhere in the phylogenetic tree where one end is labelled with the character and the other end is not. Single column discrepancy measures this unavoidable cost.

In cases where there is not much evolution, single column discrepancy may lead to the exact cost of the optimum phylogenetic tree rather than just a lower bound on the cost. Suppose that for an optimum phylogenetic tree the following situation is true for each column of the data. For some character in the difference set, the organisms labelled with that character form a single phylogenetic subtree. When you delete that phylogenetic subtree, the same holds for the remaining characters. In this case, each deleted phylogenetic subtree makes no contribution to the discrepancy of the column. The attachment cost of the deleted phylogenetic subtree is one. Thus, the contribution of the organisms not in *current_tree* to the discrepancy is exactly the same as the amount computed from single column discrepancy. We could ensure that the backtracking algorithm ran fast (in a time proportional to the number of solutions, the length of the data, and a small power of the number of organisms) if the partial phylogenetic trees that lead only to nonoptimum complete phylogenetic trees produced an estimated cost bigger than the optimum cost, but this does not always happen. The improvement from using single column discrepancy is usually large, but the method provides no guarantee of an improvement.

The paper by Foulds *et al.* (1979) contains the idea of single column discrepancy for the special case where *current_tree* is empty. For the empty case one must reduce the result of the above calculation by the number of columns, because a one organism phylogenetic tree has no discrepancy. When *current_tree* is empty, one organism can be added to it at no cost. In Foulds *et al.* (1979) there are also analyses which consider several columns together. To adapt those ideas to speeding up parsimony programs one must find a way to generalize the ideas and then implement them with low overhead.

Increasing the cost estimate never increases the number of nodes in the search so long as the cost is used only to decide which parts of the search space to explore. The dynamic program, however, also uses the cost estimate to help decide which organism to add to *current_tree*. In this case it is possible for the 'improved' cost function to lead to a poor selection and to an increase in the number of nodes, but it is rare for this to happen. (For this paper it happened in 3 of the 47 cases; each of the three cases took less than 1 s.)

In the static order program, the single column discrepancy calculation needs to be done just once. The time needed to compute it is extremely small. In the dynamic program it is done at every node, and the calculation is done incrementally. To help do the calculation quickly, for each column, counts are kept of the number of times that a character occurs in a column and of the number times that it occurs among the organisms not in *current_tree*. The measurements reported below indicate that the time to compute the single column discrepancy for the dynamic program increases the average time per node by 37%. (The quantity that was averaged is the time divided by the product of the number of nodes, the length of the sequences, and the number of organisms.)

Measurements

To determine the effectiveness of single column discrepancy on real data, four programs (static and the dynamic search orders, each without and with single column discrepancy) were run on the 47 data sets described in the Appendix. The number of nodes in the search trees and the total running time (including the time for reading the input and the time for the dynamic program to print answers to a file) are given in Tables 1, 2, 3, and 4. Most of the programs were run on a 200 MHz Pentium Pro processor running the Microsoft Windows 95 operating system. Some cases that needed a lot of memory were run on a computer with the same processor that was running Microsoft Windows NT, but the choice of the version of Windows used had no significant effect on running times.

The summary statistics are calculated on 36 of the data sets. The seq25 data set was excluded because the dynamic

Table 1. Thirteen organism results. This table shows the number of nodes generated and the time (in seconds) used for static and dynamic search programs. For each program, the None column gives the results for the version of the program that does not use single column discrepancy and the Single column gives the time for the version that does use it. The Ratio column gives the value in the none column (n) divided by the value in the single column (s). The ln ratio column gives $100[1 - (\ln n)/(\ln s)]$. The three rows for each data set are: nodes generated during pass 1, nodes generated during pass 2, and total time in seconds

Data	Static				Dynamic			
set	None	Single	Ratio	ln ratio	None	Single	Ratio	ln ratio
atp6					13 919	4 269	3.26	12.4
	17 598	11 146	1.58	4.7	14 978	4 703	3.18	12.0
	18.74	11.77	1.59		59.78	19.44	3.08	
atp8					45 749	4 970	9.21	20.7
	48 184	24 397	1.97	6.3	47 566	7 007	6.79	17.8
	20.46	9.46	2.16		94.19	12.03	7.83	
cox1					70 986	36 238	1.96	6.0
	128 244	86 537	1.48	3.3	68 443	34 758	1.97	6.1
	108.62	68.46	1.59		147.92	81.24	1.82	
cox2					5 826	3 090	1.89	7.3
	8 759	6 404	1.37	3.4	5 996	3 225	1.86	7.1
	5.41	3.97	1.36		15.23	8.68	1.75	
cox3					4 814	865	4.41	20.2
	8 000	4 259	1.88	7.0	5 545	1 055	5.26	19.2
	4.50	2.54	1.77		12.48	2.83	4.41	
cytb					82 988	21 464	3.87	11.9
	172 601	105 039	1.64	4.1	89 052	22 586	3.94	12.0
	217.95	125.34	1.74		405.32	105.32	3.85	
ndh1					96 232	35 688	2.70	8.6
	123 087	87 421	1.41	2.9	103 435	38 896	2.66	8.5
	123.16	97.04	1.27		415.87	146.54	2.84	
ndh2					788 674	182 533	4.32	10.8
	754 600	450 220	1.68	3.8	777 279	187 855	4.14	10.5
	1 788.07	1 055.25	1.69		6 139.39	1 611.59	3.80	
ndh3					47 912	4 810	9.96	21.3
	39 627	17 177	2.31	7.9	53 416	3 314	16.12	25.5
	20.03	8.24	2.43		105.00	9.22	11.39	
ndh4					116 601	25 830	4.51	12.9
	179 751	84 474	2.13	6.2	120 948	25 129	4.81	13.4
	413.02	176.51	2.34		1 099.00	236.61	4.64	
ndh4l					7 770	1 881	4.13	15.8
	10 519	4 896	2.15	8.3	7 879	2 077	3.79	14.9
	4.38	1.97	2.23		18.80	5.59	3.36	
ndh5					1 197 974	235 080	5.10	11.6
	1 380 968	575 844	2.40	6.2	1 237 460	241 519	5.12	11.6
	4 470.87	1 892.63	2.36		13 584.35	2 812.84	4.83	
ndh6					16 160	5 512	2.93	11.1
	16 724	10 120	1.65	5.2	16 625	3 471	4.79	16.1
	10.81	6.09	1.77		57.71	16.28	3.54	
ratite					2 811	1 358	2.07	9.2
	4 747	4 412	1.08	0.9	2 962	1 444	2.05	9.0
	2.41	2.27	1.06		4.49	2.77	1.62	

search algorithm (as it was compiled) could handle no more than 24 organisms. Data sets where some programs needed less than 1 s were excluded because much of this time was for overhead. Including those cases would have

Table 2. The cytb*n* results

Data set	Static				Dynamic			
	None	Single	Ratio	ln ratio	None	Single	Ratio	ln ratio
cytb12					10 917	5 953	1.83	6.5
	12 943	8 682	1.49	4.2	12 103	6 267	1.93	7.0
	9.44	6.92	1.36		34.14	18.04	1.25	
cytb13					20 474	6 671	3.07	11.3
	27 149	18 045	1.51	4.0	21 898	7 514	2.91	10.7
	23.44	16.73	1.40		71.78	24.39	2.94	
cytb14					71 479	27 592	2.59	8.5
	113 680	78 767	1.44	3.2	77 447	28 610	2.71	8.8
	111.28	76.99	1.45		259.76	101.18	2.57	
cytb15					439 687	135 410	3.25	9.1
	786 654	509 150	1.55	3.2	475 194	152 766	3.11	8.7
	914.75	608.11	1.50		1 686.19	505.95	3.33	
cytb16					1 619 538	480 710	3.37	8.5
	1 884 119	1 251 572	1.51	2.8	1 762 040	517 913	3.40	8.5
	2 504.43	1 666.21	1.50		6 348.85	1 883.20	3.37	
cytb17					8 144 105	2 415 660	3.37	7.6
	14 888 639	10 638 244	1.40	2.0	8 907 690	2 607 887	3.42	7.7
	23 913.55	15 869.69	1.46		33 927.50	9475.38	3.58	
cytb18					10 785 421	3 256 511	3.31	7.4
	16 650 996	10 856 675	1.53	2.6	11 787 153	3 519 742	3.35	7.4
	26 775.79	17 103.20	1.57		46 246.42	13 490.15	3.43	
cytb19					77 622 929	19 851 735	3.91	7.5
	158 200 957	109 962 370	1.44	1.3	85 072 694	21 452 051	3.97	7.5
	280 637.01	195 762.09	1.43		350 912.50	88 336.29	3.97	

led to much larger apparent averages for the time per node statistics.

The reported times are total elapsed times. Time entries have been rounded to the nearest hundredth of a second, but (in most cases) the actual values were used for the computations. In most cases nearly all of this time is that required to compute the set of optimum phylogenetic trees. For the problems that took less than a second, the time to input the data was significant. It could vary by a few tenths of a second depending on which order the runs were made. We took care to reduce this variation. For those few problems with a lot of output, the output time was significant. An extreme example is the Cytc22 data with the dynamic program using single column discrepancy. Approximately one solution was produced for each five nodes, and the program used about the same time formatting output as it did backtracking. A few cases were run more than once. Long times were usually repeatable to within 1%. Short times were usually repeatable to within one-hundredth of a second.

The cytochrome *c* data set (Table 3) was used extensively for developing the dynamic program. Only minor changes were made to the program after it was run on the other data sets. Improvements were made to the static program so that it could run larger data sets and to reduce its output time.

The data sets used in these efficiency computations were chosen on the basis of their usefulness in systematics studies in inferring long as well as short term evolutionary histories. For instance, the data sets for Table 2 were selected because they had a large number of organisms and are from widely used mitochondrial cytochrome *b* gene. All but the last data set in Table 1 are from 13 diverse vertebrate species for which complete mitochondrial genomes have been sequenced. The last entry is for 13 birds. Data in Table 4 is for mitochondrial DNA of humans from different geographic regions. Tables 2, 3, and 4 show how the running times vary as a function of the number of organisms selected from the data set.

In most cases the total time was determined primarily by the product of the total number of nodes, the number of organisms, and the length of the sequence. The total time in microseconds divided by the product was

0.79 ± 0.19 (0.58 minimum, 1.45 maximum) for static order without single column discrepancy,

Table 3. The cytc*n* results

Data	Static				Dynamic			
set	None	Single	Ratio	ln ratio	None	Single	Ratio	ln ratio
cytc12					94	49	1.92	14.3
	724	664	1.09	1.3	717	678	1.06	0.9
	0.16	0.15	1.10		0.45	0.43	1.05	
cytc13					258	50	5.16	29.7
	1 152	844	1.36	4.4	1 230	737	1.67	7.2
	0.25	0.20	1.26		0.65	0.50	1.30	
cytc14					366	14	26.14	55.3
	2 351	2 039	1.15	1.8	1418	649	2.18	10.8
	0.54	0.47	1.15		0.76	0.56	1.36	
cytc15					1 183	15	78.87	61.7
	8 158	5 913	1.38	3.6	4 847	1 914	2.53	10.9
	2.36	1.64	1.44		2.09	1.47	1.42	
cytc16					2 312	79	29.27	43.6
	9 033	5 130	1.76	6.2	7 495	4 478	1.67	5.8
	2.67	1.51	1.76		3.91	2.94	1.33	
cytc17					8 274	891	9.29	2.47
	43 986	31 749	1.39	3.0	28 336	22 503	1.26	2.2
	15.45	11.44	1.35		14.53	11.63	1.25	
cytc18					35 978	892	40.33	35.2
	207 178	135 312	1.53	3.5	153 256	52 262	2.93	9.0
	102.65	72.10	1.42		81.04	59.65	1.36	
cytc19					132 034	999	132.17	41.4
	976 645	557 656	1.75	4.1	603 461	93 720	6.44	14.0
	395.77	236.28	1.68		221.46	122.70	1.80	
cytc20					94 807	2 321	40.85	32.4
	2 761 955	2 511 311	1.10	0.6	281 694	9 644	29.21	26.9
	1 124.84	1 049.92	1.07		86.79	13.25	6.55	
cytc21					890 348	10 301	86.43	32.6
	4 500 937	3 610 185	1.25	1.4	815 118	31 812	25.62	23.8
	1 927.16	1 575.76	1.22		385.30	37.83	10.19	
cytc22					1 036 896	16 724	62.00	29.8
	16 602 735	12 704 028	1.31	1.6	2 039 985	47 845	42.64	25.8
	7 768.39	5 887.91	1.32		811.43	44.61	18.19	

0.79 ± 0.20 (0.57 minimum, 1.56 maximum) for static order with single column discrepancy,

1.16 ± 0.35 (0.51 minimum, 1.85 maximum) for dynamic order without single column discrepancy,

1.57 ± 0.68 (0.64 minimum, 3.59 maximum) for dynamic order with single column discrepancy.

When the total time was small, the input time drove up the measured time per node, but this effect is not large since only cases that used at least 1 s were used to compute the averages. When the amount of output was large the output time also drove up the time per node, but this effect was also not important for most of the runs. The last two lines are roughly twice the size of the first two lines. This is associated with the fact that the labelling algorithm for the static algorithm has a single bottom up pass while the labelling algorithm for the dynamic algorithm has a bottom up pass followed by a top down pass, resulting in twice as much work. Greater use of incremental calculation in the dynamic algorithm saves a little time, but makes the program much more complex.

Since the various programs have only moderate differences in their time per node, the running time needed by the various programs is determined primarily by how many nodes each generates when solving a problem.

For static search, the use of single column discrepancy leads to only a small improvement in the running time, and it never increases the time (by more than a few

Table 4. The seq*n* results

Data set	Static				Dynamic			
	None	Single	Ratio	ln ratio	None	Single	Ratio	ln ratio
seq12					30	10	3.00	32.3
	43	43	1.00	0.0	35	41	0.85	−4.5
	0.03	0.04	0.92		0.26	0.64	0.41	
seq13					33	11	3.00	31.4
	47	47	1.00	0.0	37	52	0.71	−9.4
	0.04	0.04	0.95		0.30	0.45	0.67	
seq14					36	12	3.00	30.7
	70	70	1.00	0.0	79	94	0.84	−4.0
	0.05	0.05	0.98		0.39	0.66	0.59	
seq15					101	59	1.71	11.6
	338	351	1.02	0.3	707	677	1.04	0.7
	0.18	0.18	1.00		1.40	1.54	0.91	
seq16					174	79	2.20	15.3
	273	265	1.03	0.5	481	415	1.16	2.4
	0.15	0.14	1.03		0.94	1.34	0.70	
seq17					209	141	1.48	7.4
	769	739	1.04	0.6	699	555	1.26	3.5
	0.30	0.32	0.94		1.33	1.48	0.90	
seq18					368	220	1.67	8.7
	1 396	1 305	1.07	0.9	973	768	1.27	3.4
	0.58	0.55	1.06		1.42	1.66	0.86	
seq19					696	492	1.41	5.3
	2 789	2 789	1.00	0.0	1 988	1 732	1.15	1.8
	1.22	1.22	1.00		2.81	2.89	0.97	
seq20					4 439	3 431	1.29	3.1
	7 788	7 788	1.00	0.0	5 114	4 637	1.10	1.1
	3.65	3.83	0.95		5.64	5.60	1.01	
seq21					5 568	3 468	1.61	5.5
	9 227	9 227	1.00	0.0	7 362	5 805	1.27	2.7
	4.72	4.80	0.98		8.97	8.71	1.03	
seq22					4 083	1 894	2.16	9.2
	40 571	38 177	1.06	0.6	9 010	5 655	1.59	5.1
	27.77	26.88	1.03		10.06	8.03	1.25	
seq23					47 593	31 421	1.51	3.9
	65 918	61 342	1.07	0.6	69 869	52 933	1.32	2.5
	36.85	34.55	1.07		61.33	51.03	1.20	
seq24					39 668	33 403	1.19	1.6
	70 393	62 965	1.12	1.0	74 941	55 710	1.35	2.6
	40.00	38.24	1.05		64.38	56.28	1.14	
seq25								
	132 335	126 519	1.05	0.4				
	89.62	89.06	1.01					

percent). The median factor of improvement was 1.4 and the maximum was 2.4. The fractional improvement tended to be larger for the larger problems. A measure that did not appear to vary systematically with problem size is ratio of the logarithm of the number of nodes. For static order, the average value of this ratio is 0.966 ± 0.023. Thus, if static search order without single column discrepancy generates N nodes, with single column discrepancy it will typically generate $N^{0.966}$ nodes. The actual improvement will vary greatly depending on the data set.

For dynamic search single column discrepancy leads to a bigger improvement. The median improvement factor was 2.9, the maximum was 18.2. The average ratio of the logarithm of the number of nodes was 0.838 ± 0.136 for the first pass and 0.892 ± 0.070 for the second pass.

It is also interesting to compare the number of nodes that these algorithms produce with that of a naive algorithm, one that generates all $(2n - 5)!!$ candidate phylogenetic trees for n organisms. Since that number increases (slightly faster than) exponentially with n and since the number of nodes generated by the various algorithms also appears to increases exponentially with n, it is useful to compare the ratio of the logarithm of the nodes that the algorithm produces with the logarithm of $(2n - 5)!!$. The results for nodes on pass two are:

> 0.3780 ± 0.1170 (0.1809 minimum, 0.6056 maximum) for static order without single column discrepancy,
>
> 0.3638 ± 0.1092 (0.1794 minimum, 0.5682 maximum) for static order with single column discrepancy,
>
> 0.3656 ± 0.1188 (0.1682 minimum, 0.5682 maximum) for dynamic order without single column discrepancy.
>
> 0.3046 ± 0.1050 (0.1596 minimum, 0.5310 maximum) for dynamic order with single column discrepancy.

The number of nodes generated by these various algorithms appear to increase exponentially with the number of organisms, but the coefficient in the exponent has been greatly reduced.

The sequences in Table 4 (various human sequences) are all closely related, a case where single column discrepancy is expected to be particularly effective. However, all algorithms run fast for this data, so single column discrepancy does not lead to large improvements in the running time. In part the improvement is small because Table 4 is based on DNA sequence data (the other tables are based on amino acid data) for which the alphabet size is just 4, and single column discrepancy is expected to be less helpful with small alphabets.

It is simple to adapt single column discrepancy to the version of parsimony where each column has its own weight. One just needs to use the weight when computing the contribution of the characters in the difference set.

It is more complex to adapt single column discrepancy to the version of parsimony where the cost of an edge depends on the characters that label the two ends of the edge. Basically, one would need to consider for each character in the difference set which character associated with *current_tree* could be used for the cheapest attachment. While the idea is simple enough in principle, carrying it out would require major changes in the algorithms that we used.

Summary

A carefully done backtracking search can find the optimum parsimony trees much more rapidly than a complete search. The key idea needed to obtain a big improvement is to put the organisms into an order such that the most distantly related organisms are added to the phylogenetic tree first, so that the phylogenetic trees with just a few organisms already show most of the optimum cost. Using this idea with n organisms, reduces the running time by a factor of about $e^{0.62n}$ (2.4×10^5 for $n = 20$), but the improvement varies a lot depending on particular data.

It is simple to add single column discrepancy to the basic backtracking algorithm. This improves the running time by a factor of about $e^{0.01n}$ (1.2 for $n = 20$).

Adding single column discrepancy to a dynamic search order program improves the running time by a factor of about $e^{0.04n}$ (2.2 for $n = 20$).

The question of whether or not it is worth using a dynamic search order is complicated. Using dynamic search along with single column discrepancy improves the asymptotic number of nodes by a factor of about $e^{0.05n}$ (2.7 for $n = 20$) compared with static search without single column discrepancy. This is important for large n. On the other hand, the greater complexity of dynamic search increases time per node by about a factor of 2. The dynamic algorithm was the fastest for only 10 of the cases tested. These were many (but not all) of the cases that needed large amounts of time.

These results suggest that for even greater speed, one should consider algorithms that use a dynamic search order near the root of the search tree, but use a fixed order for the deeper levels. This approach should lead to a noticeable reduction in the number of nodes while not significantly increasing the time per node. Experiments are needed to verify that this approach works well in practice.

Saving time is not particularly important except when you are using a lot of it. The algorithms in this paper are particularly useful when n is large. They are also useful for smaller problems when doing a computationally intensive bootstrap analysis to assess the robustness of the inferred phylogenetic tree (Felsenstein, 1993).

Acknowledgements

This research was partially supported by funds from Arizona State University to SK, and research grants to PWP (NSF), SK (NIH) and Masatoshi Nei (NSF, NIH).

Table A. Some characteristics of the data sets used

Data set	Number of organisms	Sequence length	Useful sequence length	Maxium characters in column	Optimum discrepancy	Extra discrepancy	Solutions
atp6	13	214	120	8	459	59	1
atp8	13	50	43	9	201	1	2
cox1	13	510	89	7	268	117	2
cox2	13	224	80	7	260	68	1
cox3	13	258	72	7	215	72	1
cytb	13	337	85	7	507	85	1
ndh1	13	311	124	7	475	99	2
ndh2	13	340	222	9	1033	119	2
ndh3	13	112	56	8	230	19	1
ndh4	13	452	233	9	965	158	1
ndh4l	13	50	211	7	211	37	2
ndh5	13	565	294	10	1368	205	1
ndh6	13	116	81	8	325	31	1
ratite	13	370	63	4	132	28	1
cytb12	12	376	96	7	313	89	1
cytb13	13	376	106	7	348	74	1
cytb14	14	376	108	7	369	74	1
cytb15	15	376	110	8	398	69	2
cytb16	16	376	112	8	420	72	1
cytb17	17	376	114	8	441	70	1
cytb18	18	376	115	8	446	68	2
cytb19	19	376	118	8	480	67	1
cytc12	12	23	15	5	24	3	222
cytc13	13	23	15	6	28	5	222
cytc14	14	23	18	6	33	1	222
cytc15	15	23	18	6	34	1	666
cytc16	16	23	18	6	36	1	1 202
cytc17	17	23	19	6	38	0	4 938
cytc18	18	23	19	6	39	0	31 350
cytc19	19	23	19	6	43	1	61 428
cytc20	20	23	20	6	48	1	4 662
cytc21	21	23	21	6	56	2	13 986
cytc22	22	23	23	6	61	0	13 986
seq12	12	637	21	3	28	5	14
seq13	13	637	21	3	28	9	14
seq14	14	637	21	3	28	9	42
seq15	15	637	24	3	36	9	420
seq16	16	637	25	3	39	16	210
seq17	17	637	30	3	47	12	210
seq18	18	637	32	3	50	12	210
seq19	19	637	32	3	51	13	420
seq20	20	637	33	3	53	12	630
seq21	21	637	33	3	53	13	1 470
seq22	22	637	35	4	60	15	1 260
seq23	23	637	37	4	67	19	3 780
seq24	24	637	41	4	71	15	3 780
seq25	25	637		4	73		7 560

Appendix

Table A gives a brief description of the data used for timing the various algorithms. The columns are: (1) the data set name, (2) the number of organisms in the data set, (3) the length of each sequence in the data set, (4) the number of informative columns in the data set, (5) the maximum number of distinct characters in any column of the data set, (6) the discrepancy of the optimum solutions for the data set (using only the informative columns), (7) the additional discrepancy that results from using all columns, and (8) the number of optimum solutions.

The following gives a reference, a brief description, and a list of organisms for the data sources for the measurements. Refer to the reference for more details. We use the same organism names that the cited source uses.

When the data set name ends with a two digit number, various numbers of organisms where used. The *n*th set consists of the first *n* of the listed organisms. The first 13 data sets (Kumar, 1996; Nei, 1996; Russo *et al.*, 1996) have the same list of organisms, so the list is given just once (under ndh6).

atp6: ATP subunit 6, mitochondrial (mt) amino acid sequence.

atp8: ATP subunit 8, mt amino acid sequence.

cox1: Cytochrome c oxidase subunit 1, mt amino acid sequence.

cox2: Cytochrome c oxidase subunit 2, mt amino acid sequence.

cox3: Cytochrome c oxidase subunit 3, mt amino acid sequence.

cytb: Cytochrome b, mt amino acid sequence.

ndh1: NADH subunit 1, mt amino acid sequence.

ndh2: NADH subunit 2, mt amino acid sequence.

ndh3: NADH subunit 3, mt amino acid sequence.

ndh4: NADH subunit 4, mt amino acid sequence.

ndh4l: NADH subunit 4L, mt amino acid sequence.

ndh5: NADH subunit 5, mt amino acid sequence.

ndh6: NADH subunit 6, mt amino acid sequence. Fin-back whale, Blue Whale, Cow, Rat, Mouse, Opossum, Chicken, African clawed frog, Rainbow trout, Loach, Carp, Lamprey, Sea Urchin. See (Kumar, 1996; Nei, 1996; Russo *et al.*, 1996) for species names and the GenBank accession numbers for the complete mitochondrial genomes.

ratites (Cooper *et al.*, 1992): Nucleotide sequences of the mitochondrial 12S ribosomal RNA sequences from flightless birds. Three moas, two kiwis, emu, cassowary, ostrich, rhea, and Tinamou.

cytb*n* (Yoder *et al.*, 1996) ($12 \leq n \leq 19$): Cytochrome b, mt amino acid sequence. Organisms used: Human, Common Chimpanzee, Gorilla, Orangutan, Common Gibbon, Agile Gibbon, Rhesus Monkey, Guereza Monkey, Proboscis Monkey, Squirrel Monkey, Aye-aye, Fat-tailed Lemur, Coquerel Lemur, Mouse Lemur, Golden Crown Sifaka, Ruffed Lemur, Collared Lemur, Red Crowned Lemur, Greater Bush Baby.

cytc*n* (Foulds *et al.*, 1979) ($12 \leq n \leq 22$): Cytochrome c mitochondrial amino acid sequence (only variable positions included). Organsims used: Human, Monkey, Horse, Dog, Pig, Whale, Rabbit, Bat, Seal, Mouse, Zebra, Kangaroo, Chicken, Penguin, Duck, Pigeon, Emu, Ostrich, Turtle, Frog, Tuna, Carp.

seq*n* (Vigilant *et al.*, 1992) ($12 \leq n \leq 25$): Mitochondrial D-loop DNA sequences from different humans: W. Pygmy(1), W. Pygmy(2), W. Pygmy(4), !Kung(7), !Kung(8), !Kung(9), !Kung(10), !Kung(11), !Kung(13), !Kung(15), !Kung(16), !Kung(17), !Kung(18), !Kung(19), !Kung(22), Asian(23), Yoruban(24), Yoruban(25), Yoruban(26), Asian(28), Yoruban(29), E. Pygmy(32), W. Pygmy(37), W. Pygmy(38), W. Pygmy(39).

References

Cooper,A., Mourer-Chauvire,C., Chambers,G.K., von-Haeseler,A., Wilson,A.C. and Paabo,S. (1992) Independent origins of New Zealand Moas and Kiwis. *Proc. Natl Acad. Sci. USA*, **89**, 8741–8744.

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

Felsenstein,J. (1993) PHYLIP: Phylogeny Inference Package, version 3.5. University of Washington, Seattle, WA.

Fitch,W.M. (1971) Toward defining the course of evolution: minimum change for a specific tree topology. *Syst. Zool.*, **20**, 406–416.

Foulds,L.R., Hendy,M.D. and Penny,D. (1979) A general approach to proving the minimality of phylogenetic trees illustrated by an example with a set of 23 vertebrates. *J. Mol. Evol.*, **13**, 151–166.

Graham,R.L. and Foulds,L.R. (1982) Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Math. Biosci.*, **60**, 133–142.

Hartigan,J.A. (1973) Minimum evolution fits to a given tree. *Biometrics*, **29**, 53–65.

Hendy,M.D. and Penny,D. (1982) Branch and bound algorithms to determine minimal evolutionary trees. *Math. Biosci.*, **59**, 277–290.

Kumar,S. (1996) Patterns of nucleotide substitution in mitochondrial protein coding. *Genetics*, **143**, 537–548.

Kumar,S., Tamura,K. and Nei,M. (1993) MEGA: Molecular Evolutionary Genetics Analysis, version 1.0. Pennsylvania State University, University Park, PA.

Maddison,W.P. and Maddison,D.R. (1993) *MacClade: Analysis of Phylogeny and Character Evolution*. Sinauer Associates, Sunderland, MA.

Nei,M. (1996) Phylogenetic analysis in molecular evolutionary genetics. *Ann. Rev. Genet.*, **30**, 371–403.

Russo,C.A.M., Takezaki,N. and Nei,M. (1996) Efficiencies of different tree-building methods in recovering a known vertebrate phylogeny. *Mol. Biol. Evol.*, **13**, 525–536.

Swofford,D.L. (1998) PAUP*: Phylogenetic Analysis Using Parsimony and other methods, University of Illinois, Champaign, IL.

Taylor,W.R. (1996) Multiple protein sequence alignment: algorithms and gap insertion. In Doolittle,R.F. (ed.), *Methods in Enzymology,* vol 266. Academic Press, San Diego.

Vigilant,L., Stoneking,M., Harpending,H., Hawkes,K. and Wilson,A.C. (1992) African populations and the evolution of human mitochondrial DNA. *Proc. Natl Acad. Sci. USA*, **89**, 8741–8744.

Yoder,A.D., Cartmill,M., Ruvolo,M., Smith,K. and Vilgalys,R. (1996) Ancient single origin for Malagasy primates. *Proc. Natl Acad. Sci. USA*, **93**, 5122–5126.