Cladistics **15**, 415–428 (1999)
Article ID clad.1999.0122, available online at http://www.idealibrary.com on IDEAL®

Analyzing Large Data Sets in Reasonable Times: Solutions for Composite Optima

Pablo A. Goloboff

Consejo Nacional de Investigaciones Científicas y Técnicas, Instituto Miguel Lillo, Miguel Lillo 205, 4000 S. M. de Tucumán, Argentina

Accepted October 28, 1999

New methods for parsimony analysis of large data sets are presented. The new methods are sectorial searches, tree-drifting, and tree-fusing. For Chase *et al.*'s 500-taxon data set these methods (on a 266-MHz Pentium II) find a shortest tree in less than 10 min (i.e., over 15,000 times faster than PAUP and 1000 times faster than PAUP*). Making a complete parsimony analysis requires hitting minimum length several times independently, but not necessarily all "islands"; for Chase *et al.*'s data set, this can be done in 4 to 6 h. The new methods also perform well in other cases analyzed (which range from 170 to 854 taxa).

INTRODUCTION

Parsimony analysis is well known as a computationally difficult problem. The most widely used strategy for finding optimal trees is multiple random addition sequences plus tree bisection reconnection branch-swapping (RAS + TBR). That strategy usually works well for data sets of less than 100 taxa. The rationale behind using several independent starting points is that if local optima exist for the data set, the end point of the different searches will eventually fall in a global optimum.

For large data sets, however, this strategy has proved unsuccessful. The most famous example to date is probably Chase *et al.*'s (1993) 500-taxon data set (baptized as Zilla by some authors), which was run for 3.5 months on three Sun Workstations (Rice *et al.*, 1997) using PAUP (Swofford, 1993), without ever finding shortest trees. Soltis *et al.* (1998) considered that analyzing large data sets may be more feasible than suggested by Rice *et al.*'s analysis if one uses combined data sets. However, despite their optimism, none of their PAUP* (Swofford, 1998) searches for two data sets combined (over 2800 bp) could finish in an average time of 530 h—even when they used only 193 taxa and only a few starting points for TBR swapping (they "almost certainly did not find the shortest trees": Soltis *et al.* 1998: 34).

The traditional RAS + TBR analyses (like Rice *et al.*'s) are extremely unlikely to find shortest trees in the case of large data sets. Large data sets require a qualitatively different approach. Using proper methods, it takes less than 10 min to find a shortest tree for Zilla. This is about 15,000 times faster than Rice *et al.*'s analysis with PAUP. The ratio of 15,000 is, however, a gross underestimation of the actual speed ratio of the two methods, since Rice *et al.* never found a shortest tree.

0748-3007/99 $30.00

METHODS

The sources for the data sets tested here are Chase *et al.* (1993) (500 taxa, 1428 characters, minimum known length 16,218), Liebherr (1998) (170 taxa, 206 characters, minimum known length 1653), H. Ochoterena (personal communication) (854 taxa, 937 informative characters, minimum known length 23,005), R. Zander (personal communication) (77 taxa, 75 characters, minimum known length 496), Lipscomb *et al.* (1998) (439 taxa, 4037 characters, minimum known length 42,116, called Mothra), and D. Eernisse (personal communication) (476 taxa, 1008 informative characters, minimum known length 17,765). Subtree pruning regrafting and tree bisection reconnection are abbreviated as SPR and TBR, respectively.[1] All the timings reported are on a 266-MHz Pentium II machine, running under Windows NT, using a single thread of execution, with programs compiled with a Watcom C/C++ compiler (version 11). Branch-swapping in those programs uses Goloboff's (1996) algorithms, but with multi-character optimization performed by storing several characters in a single 32-bit integer (as in Moilanen, 1999; and Ronquist, 1998). For TBR, the "union-construct" shortcut of Goloboff (1996) was modified, producing a slightly more efficient parallel evaluation of destinations (the method of "node clusters"). In the case of Zilla, evaluating the ca. 9.5 million rearrangements to complete TBR on a quasi-optimal tree takes an average time of 5.5 s (i.e., 1.7 million rearrangements/s). For SPR, algorithms from J. S. Farris (personal communication) led to dramatic speed increases (for Zilla, six to seven times) during the initial stages of the search; further speed increases of almost 50% were achieved with "dual evaluation" of destinations during swapping (the two descendants of each internal node are evaluated together, using the union of their state sets, and only when this produces length increments below the current bound are the two nodes evaluated individually). Thus, creating a Wagner tree and completing SPR for Zilla requires an average time of only 15 s. My implementation of some of these algorithms was somewhat experimental; possibly, more careful implementations will produce faster analyses. Since the basic code (e.g., the branch-swappers and optimizers) was the same in all cases, the different results presented here can be directly compared. The methods are incorporated in the program TNT ("Tree Analysis Using New Technology;" Goloboff, Farris, and Nixon, in preparation), beta versions of which can be downloaded from ftp.unt.edu.ar/pub/parsimony or www. cladistics.com).

[1]The terms, introduced by Swofford (1993), are the most commonly used for these methods. The methods themselves, however, had long been in use. Subtree pruning regrafting had been in use in PHYLIP (Felsenstein, 1993) since 1981 as "global rearrangements," and tree bisection reconnection is the same as the branch-breaker method (bb) used in Hennig86 (Farris, 1988).

PROBLEMS WITH LARGE DATA SETS

Rice *et al.* (1997) used PAUP (Swofford, 1993) in their analysis of Zilla. The shortest trees they could find had 16,220 steps (they counted uninformative characters, thus adding 313 steps). The shortest trees for Zilla are 16,218 steps long—2 steps shorter than Rice *et al.*'s trees. However, the impression that months of search on three Sun Workstations were not sufficient to find shortest trees for Zilla when Rice *et al.* published their paper is quite inaccurate. When their paper was submitted, NONA (Goloboff, 1994) was 30 times faster than PAUP[2] and had a better implementation of RAS + TBR.

As pointed out by Giribet and Wheeler (in press), the fact that NONA could retain smaller numbers of trees during each replication made a big difference. Saving too many trees per replication is a waste of time, because the trees found by swapping differ too little from the original tree(s) and are unlikely to lead to new optima. That Rice *et al.* collapsed branches with the (then only) option of PAUP that retains branches supported under ambiguous optimizations ("rule 3," Swofford, 1993; Coddington and Scharff, 1994) just made finding shortest trees less likely: collapsing with stricter criteria makes searches more efficient (Goloboff, 1996: 214). Thus, it is better to save fewer trees, collapse more strictly, and complete a larger number

[2]Versions of PAUP* after January 1999 are faster, because Swofford implemented algorithms of Goloboff (1996)—although attributing them to Ronquist (1998)—and Maddison (unpublished). The program, however, still uses RAS + branch-swapping as its main search strategy.

of RAS. Finding every possible tree may be desirable for some evolutionary studies (although even that is doubtful in the case of the possibly hundreds of thousands of most parsimonious trees for Zilla). However, for taxonomic studies there is clearly no point in finding all equally parsimonious trees: if a significant number of trees from the different global optima is sampled, the consensus would be identical to that produced by considering every possible most parsimonious tree. Farris *et al.* (1996) have already shown that, in practice, there is no point in finding all most parsimonious trees for a data set.

While most of the effort in Rice *et al.*'s search was used in needlessly finding (and swapping) thousands of equally parsimonious trees, by doing RAS + TBR saving few trees per replication with NONA, it was already possible in 1997 to find shortest trees for Zilla in 24–48 h, on an ordinary 200-MHz Pentium. This, however, is a very long time. Most of the replications never found shortest trees, and thus most of the search time was effectively being wasted. Better methods are required. Soltis *et al.* (1998) have proposed that using more characters will make parsimony analysis easier, because fewer equally parsimonious trees exist. More characters make exact ties less likely and thus produce fewer trees in a given optimum. However, it is far from clear that adding more characters will also make it easier to find a global optimum; it might make it more difficult (simply multiplying the number of local optima where a search could get stuck), and then the number of independent replications to actually find minimum length may be the same or more.

The problem with RAS + TBR is that it is extremely unlikely that a given replication will find a global optimum in the case of large data sets, because of the existence of *composite optima.* It is well known that beyond 40 or 50 taxa it is common for data sets to exhibit local optima ("islands" of Maddison, 1991). Large trees can be thought of as composed of many sectors. A tree for Zilla could be seen as composed of 10 subtrees of 50 taxa (Fig. 1). Each of those 50-taxon sectors has its own local optima, and whether a given sector is in a globally optimal configuration is (to some extent) independent of whether other sectors are (the lack of independence between optimality of resolutions for different sectors simply makes the problem more difficult). This clarifies why finding a shortest tree is so unlikely for a given RAS + TBR: this requires finding

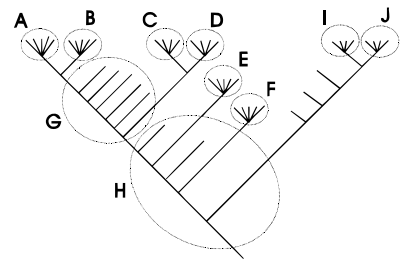

FIG. 1. A large tree, showing sectors. See text for explanation.

a tree where all the sectors are in the proper configurations at the same time. If there is a chance of 0.5 for each sector to be trapped in a local optimum of its own, in this particular case the probability of finding a globally optimal tree in a given RAS + TBR is $0.5^{10} = 0.00098$. This is, on average, less than 1 in 1000 replications. No wonder Rice *et al.* never found a shortest tree—they did only 8 replications.

Large data sets, therefore, require strategies that deal with the problem of composite optima.

Once the problem has been identified, solutions present themselves quite naturally. The solutions are based on analyzing different parts of the tree separately. Consider the tree of Fig. 1; if sectors A, B, C, and D were optimal, just starting a new replication will possibly place other sectors of the tree (say, E, F, and G) in optimal configurations, but it is unlikely that sectors A through D are all again in optimal configurations. Thus, the solution requires that sectors be improved separately, one at a time—that those sectors which are suboptimal are improved without worsening the ones that are already optimal. For this, there are four basic methods: ratchet, tree-fusing, tree-drifting, and sectorial searches. These methods do not attempt to find multiple trees during swapping, but simply concentrate on finding trees as short as possible. Ratchet is to be described elsewhere (Nixon, 1999); I concentrate here on the other methods.

TREE-FUSING (TF)

The basic idea in tree-fusing (TF) is exchanging subgroups between different trees. The subgroups must have identical composition. Moilanen (1999) had used exchanges of subtrees as part of his "natural selection"

algorithm, but he only exchanged one randomly chosen subclade at a time, placing it in a randomly chosen position. Since Gladstein's (1997) "incremental down-pass optimization" can be used to calculate the length obtained by moving a sub-group from one tree to another, all possible exchanges between the two trees can be evaluated quickly, to make only those exchanges which improve the tree. As implemented here, the exchanges comprise all the groups with five or more taxa, present in the consensus of both trees, which are themselves not dichotomously resolved (since the input trees are found by swapping, it is unlikely that a five-taxon group will be in a suboptimal configuration; dichotomously resolved groups need not be analyzed, because the exchange of their respective subgroups would produce the same result or better). For Zilla, this normally requires evaluating the exchange of about 30 subtrees (which can be done in less than a second). The groups to exchange are chosen in a down-pass; if a group within a subtree has been changed, the subtree itself is not changed further.

The best results were obtained when several trees were input to the procedure, and different pairs of trees were fused, producing new trees, as follows:

(1) Randomly select a tree (the "target" tree);

(2) Randomly select one of the remaining trees (the "source" tree). If no trees remain to be fused with the current target tree, do SPR swapping, save the result as a new tree, and go back to step 1.

(3) Evaluate the result of moving each clade in the source tree to the target tree, and go back to step 2. An alternative to the procedure above is doing the exchanges in both directions and changing the trees themselves, but that preserves less of the variability present in the original set of trees. Step 1 is repeated several times or "rounds" (normally three to five). Repeating the procedure on the same original set of trees, but selecting the trees in different orders, may produce different results. The SPR swapping at the end of step 2 on occasion improves the tree.

Tree-fusing, when provided with a set of suboptimal trees, almost invariably produces trees which are much closer to being optimal. For example, doing 10 RAS + TBR on Zilla takes about 5 min; while the best trees found are normally 16,225–16,230 steps, fusing them goes down to 16,220–16,222 within a few additional seconds. It is easy to see why TF works so well. If the 10 RAS + TBR have been obtained independently, each of the sectors will be in an optimal configuration in at least one of the trees. What is needed is putting all the optimal sectors together. Under the ideal conditions described in the legend to Fig. 1 (independence of resolution among different sectors and probability 0.5 for each sector to be in an optimal configuration), 10 trees input to the TF procedure would virtually guarantee that an optimal tree is found. Because those ideal conditions are not met in real cases, TF requires as input either many more trees produced by RAS + TBR or trees which are closer to optimal.

Moilanen's (1999) method, which he called crossover, although useful in the context of his general strategy, will not produce by itself the dramatic improvements produced by TF. The purpose of crossover in Moilanen's strategy is mostly preserving and creating a diversity of trees for subsequent swapping and selection, while TF addresses the problem of composite optima in a more direct way.

SECTORIAL SEARCHES (SS)

SS is a special form of rearrangement evaluation, needing a tree as starting point. It is based on selecting different sectors of the tree, reanalyzing them separately; if a better configuration is found, it is replaced on the whole tree. The reduced data sets can be analyzed very quickly; they are formed by representing internal nodes by their first-pass state sets (the basal node must be represented by the first-pass state set calculated upward). The sectors can be selected in two ways, randomly or based on a consensus.

Random Sectorial Searches (RSS)

The procedure is as follows:

(1) Select a sector of nodes at random, such that the reduced data set has *S* terminals (see Appendix 1 for an example).

(2) Do *R* replications of RAS + TBR (saving a single tree) for the reduced data set. If the *R* replications produce trees of the same length (thus showing that the selection *S* was in a non-conflictive part of the tree), go to step 3; otherwise, do *r* additional replications, and then go to step 3.

(3) Choose the best among the *R* + *r* replications

and the present resolution for the sector and place it in the whole tree. Go to step 4.

(4) Do a round of global swapping, but only if replacements at step 3 have been made more than X times. Go back to step 1, N times.

Steps 2–4 are skipped if the selection at step 1 produces a data set with only uninformative characters. At step 1, the data are repacked; with this, many 3-state or 4-state characters become binary (which can be optimized faster), and many characters become uninformative.

Good results are produced only for certain parameters. The sectors should not be too small (otherwise, the selection will rarely cover a part of the tree which has local optima) or too big (otherwise, RAS + TBR will never find optimal or quasi-optimal configurations). The best size seems to be 35 to 55 nodes. For that value of S, $R = 3$ and $r = 3$ are enough to make it likely that an optimal tree for the reduced data set is found. A round of global swapping of the entire tree is made every 5 to 10 replacements, as that number makes it likely that (through clade substitution) the tree will have become globally suboptimal under TBR. The number of sector selections needed to produce the best results varied with data set size: the number of selections at which it is likely that new sector selections will overlap with previous ones changes with data set size. For Zilla, 20 to 25 selections seemed to produce the best time:result ratio. At this point, further selections very rarely produced further improvement. For the 854-taxon data set, improvements were often found beyond 50- or 60-sector selections.

This method gets down to trees much shorter than those found by TBR using little additional time. The profiles of tree lengths for 1000 replications of RAS + TBR are compared to those for RAS + TBR + RSS in Table 1 and Fig. 2A. Completing a single RAS + TBR for Zilla takes an average time of 30.0 s, while completing a single RAS + TBR + RSS takes 50.8 s. For RAS + TBR, the fraction of trees which are 16,225 steps or less is only 2.5%, while adding RSS raises the fraction to 18.8%. To find as many trees under 16,226 steps as RAS + TBR + RSS finds in 100 replications, 752 replications of RAS + TBR would be necessary, but this would take 4.4 times longer.

Although far better than TBR alone, RSS rarely finds a shortest tree for a data set like Zilla. The reasons for this are clear: (1) the resolution of the different sectors is not really independent (for example, that we find a resolution optimal for some sector may require in turn that another sector be resolved in a less optimal way); and (2) achieving a configuration optimal for the whole tree may require that some group(s) be moved too far

TABLE 1

Frequency of Different Lengths, in Five Different Methods of Analysis, over 1000 Replications (Ratchet Used Only 446 Replications)

Length	MSS (46.2)	RSS (50.8)	Ratchet (141.8)	TBR (30.0)	MSS + DFT (62.9)
16,218	0.2	0	0	0	1.3
16,219	0.4	0.1	0	0	6.3
16,220	2.5	1.1	5.4	0	12.3
16,221	5.6	1.8	1.1	0	19.3
16,223	7.2	4.1	3.8	0	13.2
16,224	10.7	4.2	6.9	2.5	11.4
16,225	12.0	7.5	9.2	0	9.2
16,226	13.8	8.2	8.3	2.8	4.6
16,227	10.4	8.8	14.3	0.3	3.0
16,228	8.6	9.8	17.0	0.1	1.2
16,229	8.5	8.9	6.9	0.7	0.6
16,230	5.7	7.8	9.2	0.8	0.1
16,231	3.8	7.4	6.5	8.1	0.2
16,232	3.5	4.9	3.8	13.1	0
16,233	2.5	5.4	0.7	1.6	0
16,234	0.8	3.9	4.0	8.4	0
16,235	1.7	2.8	0.5	6.6	0
16,236	0.8	2.9	1.3	6.7	0
16,237	0.6	2.3	0.2	3.6	0
16,238	0.2	2	0.2	1	0
16,239	0.3	1.2	0	8.0	0
16,240	0	1.3	0.2	6.1	0
16,241	0.1	0.4	0.2	5.4	0
16,242	0	0.4	0	3.4	0
16,243	0	0.7	0	5.6	0
16,244	0	0.1	0	7.8	0
16,245	0	0.5	0	2.8	0
16,246	0	0.2	0	3	0
16,247	0	0.7	0	0.5	0
16,248	0	0.2	0	0	0
16,249	0	0.2	0	0.3	0
16,250	0	0	0	0.2	0
16,251	0	0	0	0.1	0
16,252	0	0	0	0	0
16,253	0	0.1	0	0.2	0
16,254	0	0	0	0	0
16,255	0	0.1	0	0.2	0
16,256	0	0	0	0	0
16,257	0	0	0	0	0
16,258	0	0	0	0.1	0

Note. The second line indicates average time per replication (in seconds).

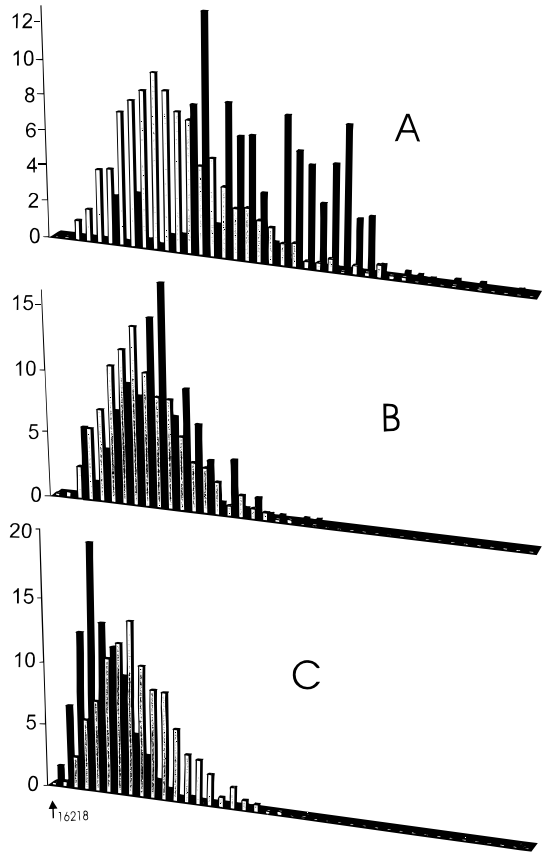

FIG. 2. Frequency of different lengths, for different methods of analysis (data from Table 1). (A) TBR (dark) and RSS (light); (B) 3 ratchet iterations (dark) and MSS (light); (C) MSS (light) and MSS + DFT (dark). All figures drawn to same scale.

away (for example, more than 50 nodes away, when $S = 40$).

Another problem with the method is that the sectors are selected at random; ideally, one would want to select sectors in such a way that they are more likely to represent areas with conflict. The $R + r$ values, to some extent, evaluate this; another possibility is selecting sectors of larger size (which are then more likely to contain areas of conflict) and analyzing them using ratchet or tree-drifting; this proved helpful for very difficult data sets.

Consensus-Based Sectorial Searches (CSS)

This procedure is very similar to RSS, but differs in the way in which the sector selection is done. The sector is selected from a consensus calculated (by some means) previously. The polytomies in the consensus will reflect areas of conflict within the data, and thus polytomies involving S nodes or more can be selected. In practice, it is very unlikely that a group of less than 10 nodes will be resolved in a suboptimal way, so for this method $S = 10$. The rest of the parameters are similar to those in RSS, except that the number of selections is determined by the number of polytomies in the consensus. However, as resolutions of the different polytomies are not completely independent, it is better to repeat (in turn) the selections Y times (three to five).

The results this method produces are as good as the consensus tree provided. A good candidate is the consensus estimated with the method of Goloboff and Farris (in press). For some data sets, however, that method produces polytomies involving too many nodes. The problem then arises that it is rather unlikely that a few replications of RAS + TBR for the reduced data sets will find an optimal configuration for the sector (i.e., the "reduced" data sets are so big that one again faces the problem of having composite optima!). This was the case for the 854-taxon data set. This could be solved by analyzing larger sectors using ratchet or tree-drifting. However, the problem remains that the method of Goloboff and Farris (in press) may tend to show some unsupported nodes as supported for some data sets, and that error would be incorporated into the procedure.

Mixed Sectorial Searches (MSS)

This is a mixture of RSS and CSS. At each replication, it starts with RAS + SPR. Once SPR is completed, the consensus of the SPR-optimal tree with the tree from the previous replication is calculated. That consensus is then used as constraints to complete TBR (which takes less time than unconstrained TBR[3]) and to produce a CSS. That consensus is produced from only two

[3]While completing a round of unconstrained TBR on a quasi-optimal tree for Zilla in TNT takes about 5.5 s, a round of TBR with 250 nodes constrained takes less than 2.5 s. In TNT, the constraints make searches more than twice as fast because the constraints are implemented in such a way that rearrangements violating constraints are not even attempted. PAUP* seems to use a different method, since a round of TBR with the same 250 nodes constrained takes about 80% of the time without constraints (i.e., about 30 s with constraints, 35 to 40 s without). It seems likely that PAUP* checks constraints using group membership variables, a slower method.

trees and is normally more resolved than the estimate used in the previous method; since its only purpose is to detect areas with conflict, it does not matter if some groups not found in shortest trees are present. Once CSS is completed, the set of constraints is abandoned, and RSS is performed. Although part of the search is constrained, the basic structure of the tree (found by RAS + SPR) is not, and the final stage (RSS) is also unconstrained. Furthermore, the constraints used vary from replication to replication. It is then very unlikely that systematic effects (like those introducing bias in Goloboff and Farris' method; see Goloboff and Farris, in press) will cause a problem.

The kind of selections made by RSS and CSS are based on different principles, so that they are combinable. Thus, this mixed method produces better results than the previous two. As part of the search is constrained, the time used is less than for RSS. The profile of tree lengths is slightly better than the profile for 3 ratchet iterations (Fig. 2B). Ratchet, however, takes an average time of 141.8 s per replication, while RAS + MSS takes only 46.2 s. For ratchet, the fraction of trees which are under 16,225 steps is 17.2%, while for RAS + MSS that fraction is 26.6%. To find as many trees under 16,225 steps as 100 RAS + MSS, 154 replications of RAS plus 3 ratchet iterations would be necessary, which would take 4.7 times longer. The comparison is even worse with RAS + TBR, as only 2.5% of the trees are under 16,225; 1064 replications of RAS + TBR would be required to find as many trees under 16,225 steps as RAS + MSS finds in 100 replications, but this would take 6.9 times longer.

TREE-DRIFTING (DFT)

In tree-drifting, suboptimal solutions are accepted during branch-swapping, with a certain probability. The probability of accepting a suboptimal solution depends on both the relative fit difference (RFD; Goloboff and Farris, in press) and the length difference between the new and old solutions. Recall that (given two trees A and B) $RFD_{AB} = (F - C)/F$, where F is the sum of character step differences in the two trees for the characters that fit tree A better (evidence "favorable" to tree A) and C is the sum for the characters that fit tree B better (evidence that "contradicts" tree A). During swapping, a lower bound on the RFD between original and candidate tree can be estimated quickly, by comparing the decrease in length when clipping to the increase in length when joining the clipped clade to a given destination (see Goloboff and Farris, in press). Solutions as good or better than the current one are always accepted.

Accepting suboptimal solutions with a certain probability is a well-known technique in difficult optimization problems, generally known as simulated annealing (Kirkpatrick *et al.*, 1983). Tree-drifting could be seen as a sort of simulated annealing. In parsimony problems, only Metro (a program formerly included in PHYLIP) has used simulated annealing before. This program performed very poorly (see Platnick, 1987, for benchmarks). Swofford *et al.* (1996) and Rice *et al.* (1997), apparently unaware of Felsenstein's program, have suggested that using simulated annealing could be a way to escape from local optima. However, using just the length to determine the acceptability of a tree (as suggested by Swofford *et al.*) is not an ideal criterion; this will suffer from the same problems as the Bremer Support, discussed by Goloboff and Farris (in press). Determining the acceptability of a tree by using both its raw length difference and the relative fit difference (RFD) is a better criterion, because this takes into account actual conflict between characters (and conflict is in turn what determines local optima). Therefore, in the present method, suboptimal solutions are rejected when RFD is greater than Z:

$$Z = X/(F + J - C),$$

where X is a random number between 0 and 99 and J is the length difference between the tree being swapped and the tree used to start the drift procedure (if a tree shorter than the original input tree is eventually found, J is re-set to 0). Factor J ensures that the tree will not progressively become longer and longer. Note that F and C are calculated between the tree being currently swapped and candidate tree, not between the original input tree and the candidate tree. As implemented here, if the tree is accepted, swapping continues from the candidate tree. Once a given number of changes, C, have been made to the tree, a round of normal swapping is done. The drifting then can be repeated D times.

The performance of the DFT is improved by constraining some nodes during the phase of normal TBR.

This "hard drift" takes the original tree and drifts it, then creates the consensus of the original tree and the tree resulting from the drift and uses that consensus as constraint tree (for each of the D cycles of drifting, a different consensus will be used as constraint). This does not significantly decrease the effectiveness of the method, but it decreases execution times. The advantage of a hard drift is that the round of normal swapping becomes focused on the areas of conflict; the areas which are identical in both trees are unlikely to lead to improvements. To ensure optimality under TBR, every certain number of constrained TBR cycles (5 to 10, depending on the total number of drifts to do), a round of unconstrained TBR is performed. This hard drift method is helpful in very difficult data sets, which require a lot of drifting.

Tree-drifting normally produces further improvements to the results of previous methods, with little additional time. For Zilla, the time for MSS alone is 46.2 s, and adding DFT ($C = 30$, $D = 3$) takes an additional 16.7 s (62.9 s total). Figure 2C and Table 1 show the length profiles for MSS + DFT. To find as many trees within 16,223 steps as 100 replications of MSS + DFT find, 329 replications of MSS alone would be necessary, but this would take 2.4 times longer.

TABLE 2
Summary Results of Test Runs for Different Combined Strategies

Data set	Method	Run time (h)	Times min. length hit	Average time to min. length (min)
	1	16.0	45	21.3
	2	14.9	119	7.5
Zilla	3	15.4	59	15.6
(16,218)	4	4.6	26	10.5
	5	15.3	100	9.2
	6	24.1	140	10.3
Mothra (42,116)	5	20.3	25	49.3
	7	134.1	14	576.0
854 taxa	8	100.2	11	546.0
(23,005)	10	19.0	0	>1140
	11	140.6	25	337.2
476 taxa (17,765)	11	72.9	6	728.7

Note. For each data set, minimum known length is indicated in parentheses.

COMBINED STRATEGIES

The method of TF can produce optimal trees only when fed with large numbers of very suboptimal trees or fewer trees which are close to optimal. SS and/or DFT are the best means to produce quickly nearly-optimal trees, which can then be used for TF. For the most difficult data sets, the best results were obtained when the trees to be fed to TF were also subject to SS and DFT combined (using DFT to analyze sectors of larger size). The strategies were run either by means of a batch file that called the search programs and then called itself again or by a driver program that supervised execution of the search programs. Table 2 shows the summary results of the test runs.

(1) RSS + TF

The searches used 16 replications of RAS + TBR + RSS ($S = 40$, $N = 20$, $R = 3$, $r = 3$, $X = 5$), followed by 3 instances of TF (with three rounds each). The average time per run for Zilla was 16.0 min. This ran 60 times in 16.0 h, finding trees of 16,218 steps 45 times, 16,219 steps 14 times, and 16,220 steps only once. This produced shortest trees every 21.3 min.

(2) CSS + TF

This was tested with Zilla, using as constraint the 250 groups produced in a conservative estimation with Goloboff and Farris' method (the strict consensus of 16 replications of RAS + SPR, collapsing the trees on SPR rearrangements with a relative fit difference of 0.20 or less and an absolute step difference of 2 or less; it takes about 5 min to produce the estimation). The searches used 10 replications of RAS + TBR + CSS, with parameters as in the previous case, except that $S = 10$ and $Y = 4$. The routine ran 154 times in 14.9 h, finding 16,218 steps 119 times, 16,219 steps 27 times, and 16,220 steps 8 times. This produced shortest trees every 7.5 min.

(3) MSS + TF

The searches used 10 replications of RAS + MSS ($R = 3$, $r = 3$, $X = 5$; for CSS, $S = 10$, $Y = 4$; for RSS,

$S = 40$, $N = 20$), tree-fusing the results 3 times (with three rounds each). For Zilla, this routine ran 85 times in 15.4 h, finding 16,218 steps 59 times, 16,219 steps 23 times, and 16,220 steps 3 times. This produced shortest trees every 15.6 min.

(4) MSS + TF under Driver

Since many of the searches in the previous methods often find a shortest tree within the first 5 or 6 min of the search, the rest of the time is really wasted. Therefore, when one already knows the minimum length for a data set, it is possible to make more efficient searches, by searching until that length is found, and then starting over with the next search. A simple driver program was designed, which calls another program that runs a given number of replications of multiple MSS (with other parameters as in previous methods); those trees are input to TF (three times, unless a shortest tree is found first, using three to six rounds each), and if no shortest trees are found, the program that does multiple MSS is called again to do a smaller number of replications (with the resulting trees added to the file containing the trees that failed to produce a shortest tree under TF); this last step is repeated until a shortest tree is found. The driver changes the number of replications according to how often the number currently in use produces shortest trees (if three or more of five main cycles found shortest trees without having to search additional MSS trees, the number of replications is decreased; if fewer than two found shortest trees without additional MSS trees, or additional MSS searches must be repeated several times in a row, the number is increased). The initial value of MSS replications (i.e., trees to be input to TF) was set to 6 (plus 3 more, if the previous ones failed to produce a shortest tree). For Zilla, this program was left running for 4.55 h, finding 26 shortest trees (a shortest tree every 10.5 min).

(5) MSS + DFT + TF under Driver

This was similar to the previous one, but added DFT to each MSS replication. The initial number of MSS + DFT replications was set to 5 (with $C = 30$, $D = 3$, for DFT). The driver ran Zilla for 2.9 h, finding minimum length 18 times (a shortest tree every 9.7 min). The addition of DFT after MSS increased speed, but only by a small factor; possibly, the improvement in length by DFT would have been easily achieved with the subsequent TF. Running the same routine with less drifting ($C = 25$, $D = 2$) produced apparently better results (minimum length hit 100 times in 15.3 h, a shortest tree every 9.2 min), but the difference may be non-significant. This strategy (with MSS followed by DFT with $C = 30$, $D = 3$) was also used in the 439-taxon data set, hitting 42,116 steps 25 times in 20.5 h (a shortest tree every 49.3 min). The 439-taxon data set takes much longer than Zilla (in part because it has many more characters: 4037 instead of 1398), but it still runs reasonably fast.

(6) RAS + TBR + DFT + TF under Driver

This strategy places more emphasis on the drift, rather than on SS. For CSS, $N = 1$ and $S = 10$; for RSS, $N = 2$ *and* $S = 40$; the DFT stage used $C = 35$ and $D = 3$. When run with the driver program, this routine hits minimum length for Zilla 140 times in 24.1 h (a shortest tree every 10.3 min). Thus (in the case of Zilla, at least), MSS followed by TF produces about the same results as DFT followed by TF; MSS + DFT followed by TF produces slightly better results (10% faster).

(7) MSS + RSS Using Ratchet + TF

This strategy was used in the 854-taxon data set, which is much more difficult than Zilla. Reasonably good results were obtained by running 50 replications of RAS + MSS ($S = 45$, $N = 60$) followed by analysis of randomly chosen sectors ($S = 80$, $N = 10$) with 15 iterations of jackknife-ratchet. Being larger, the sectors to be analyzed with ratchet were more likely to cover conflictive areas, and since they covered larger parts of the tree, a lower number of them was selected. The extra ratchet analysis of larger sectors normally produced further improvements on the tree, without taking as long as ratchet on the entire tree. A round of global TBR swapping was done every 10 replacements to the tree ($X = 10$). The routine ran 26 times in 134.10 h, finding 23,005 steps 14 times (every 9.6 h), 23,006 steps 7 times, and 23,007 steps 5 times.

(8) MSS + RSS Using Ratchet + DFT + TF under Driver

The addition of DFT ($C = 30$, $D = 3$) did not significantly improve results over the previous method. The

driver program ran for 100.2 h, hitting 23,005 steps 11 times (every 9.1 h).

(9) MSS + Ratchet + RSS

This strategy does some number of ratchet iterations to the tree produced by MSS; when the character weights are re-set to the original ones, RSS (instead of just TBR) is used for further improvement before the next ratchet iteration. It never uses TF. The results of this strategy for Zilla were highly variable, with minimum length sometimes hit very fast (within the first 2 or 3 min) and sometimes never in 15 min or more. The average results seemed inferior to those using TF, although no detailed timings were done.

(10) Multiple MSS + Ratchet + RSS + TF under Driver

For the 854-taxon data set, this strategy seems inferior to the others tried. Minimum length was never found in 19 h, using the driver program, with initial number of MSS replications set to 20 (for CSS, $S = 10$, $N = 4$; for RSS, $S = 50$, $N = 50$), each followed by eight ratchet iterations and RSS (same parameters).

(11) Multiple MSS + RSS Using DFT + Hard DFT + TF under Driver

This method produced the best results for the 854-taxon data set. The strategy is similar to strategy 8, but used more exhaustive drift and selected a larger number of sectors to be analyzed with DFT (instead of ratchet). Therefore, the number of MSS replications was lower, initially set to 8 (changed by the driver during runs to 10–12). The other parameters used were for CSS, $S = 10$, $N = 3$; for RSS, $S = 50$, $N = 50$, $X = 15$; for RSS analyzed with DFT, $S = 100$, $N = 15$, $X = 15$, analyzed with eight rounds of hard DFT; and for the global DFT, $C = 40$, $D = 25$, doing unconstrained TBR every 7 cycles of drifting. The resulting trees were subject to TF. This method hit 23,005 steps 25 times in 140.6 h (minimum length hit every 5.6 h). A similar strategy was also the one that produced the best results on the 476-taxon data set from Doug Eernisse, the most difficult data set analyzed here (the parameters were similar to those for the 854-taxon data set, except initial number of replications set to 10, changed by the driver to 22; for RSS, $S = 45$, $N = 25$, $X = 10$; for RSS analyzed with DFT, $N = 10$; and for the global DFT, $D = 80$, doing unconstrained TBR every 15 cycles of drifting). For the 476-taxon data set, this strategy found minimum known length 6 times in 72.9 h (minimum length every 12.1 h).[4]

DISCUSSION

Aside from SS, TF, and DFT, the best existing method for complex data sets is the parsimony ratchet (Nixon, 1999). Ratchet does partial changes to a tree, without changing all the tree structure; it has the advantage, over SS, that the changes made to the tree are based on character conflict—which is in turn what determines the existence of local optima. Thus, ratchet is better than SS at identifying areas of conflict. However, ratchet must complete TBR twice in every iteration, for the entire tree. The method of SS makes several faster searches. For Zilla, the results with SS + DFT + TF were superior to the best results obtained with ratchet alone (possibly, minimum length every 2 or 4 h) and superior to the best results obtained with ratchet plus TF (minimum length twice per hour, but using constraints). Ratchet eventually reaches minimum length in almost every case (artificial cases where it never does can be constructed). SS alone (or with only a few cycles of DFT) usually gets stuck at some non-minimal length, but it gets down to that point faster than ratchet, and then it is an ideal method to use in combination with TF.

Zilla, despite its size, seems a rather "clean" data set. In other words, Zilla seems to be a very good case of composite optima, with little interaction between the resolution for the different sectors in the tree. Other data sets take almost as much work to find shortest trees, despite being smaller. The (highly islandic) 170-taxon data set was also run with the driver program; MSS + DFT + TF found minimum length trees every

[4]If 12.1 h seems a long time, consider that the best trees PAUP* found for this data set after 86 h (with 274 RAS + TBR, saving up to 10 trees per replication, collapsing the trees with "amb −") were 17,778 steps—13 steps away from minimum known length. For the present strategy, trees of 17,778 steps are almost always found within the first 15 min of each individual replication.

7.6 min, while ratchet alone found them every 20 min. The difference in time between ratchet and MSS + DFT + TF for this data set was smaller than for Zilla. Apparently, both SS and TF require some structure in the data—which is normally the case for real data sets. For the (very poorly structured) 77-taxon data set ratchet was about three times faster than MSS + TF: the driver found a shortest tree every 30–40 s, while ratchet (numerous replications of RAS + 35 ratchet iterations) found them every 10 s. For large random data sets (150 taxa and 200 characters), ratchet was also far better than MSS + TF. In each iteration, ratchet can freely change most of the tree structure, if necessary, which is not the case for either SS or TF.

Through the use of the relative fit difference, the rearrangements tried by DFT are determined by character conflict much more than those tried by SS or TF, and DFT can make radical changes to the tree structure. Therefore, DFT performs much better than SS + TF in the case of random or poorly structured data sets, outperforming even ratchet. The difference from ratchet arises in part because ratchet actually tries to find an optimal tree for the re-weighted data; DFT simply finds a series of (possibly suboptimal) solutions, without completing TBR. If ratchet is modified such that the re-weighted search is not completed (but simply finishes when some number of rearrangements have been done), perhaps run times (and results) would be more comparable to those for DFT.

For small data sets (i.e., below 100 taxa) neither SS nor TF are usually of much help. Small data sets are difficult to analyze only when very poorly structured. That situation is best analyzed by doing multiple RAS + TBR followed by extensive DFT. For the 77-taxon data set, 100 such replications (with $C = 30$, $D = 25$) found minimum length 63 times, in a total time of 4.31 min (minimum length every 4.1 s).

For large data sets, TF has two important advantages over other methods. First, it uses suboptimal trees, as long as they have some sectors in optimal configurations. Therefore, one is not forced to throw effort away when a series of replications did not succeed in finding shortest trees. If a set of trees did not produce optimal trees under TF, it is possible that adding a few more trees will. Second, TF provides an additional way to test for global optimality. If the best length found several times independently does not produce better trees under TF, it is likely that length is indeed the minimum possible for the data set. In the case of Zilla, all methods used here show an asymptotic approach to optimality; without the driver, 16,218 is found in 50 to 75% of the cases. During the development of these methods, length 16,218 has been hit more than 1000 times. Possibly more significant is that 300 trees of minimum length (from about 100 independent hits) were used as input for TF, but TF did not produce any improvements. Therefore, it is very unlikely that shorter trees exist for Zilla.

The best results for the two most difficult data sets analyzed here (strategy 11, above) were obtained when increasing the exhaustiveness of each replication to use as input for TF, rather than greatly increasing the number of replications. This suggests that, possibly, the most significant improvement to the driver will be having it change during the run not only the number of replications, but also the exhaustiveness of each replication (i.e., the number of sector selections to be analyzed with DFT and the number of global DFT cycles).

The methods described here do not attempt to find multiple equally parsimonious trees during swapping (they can indirectly find multiple trees, of course). As discussed above, finding all equally parsimonious trees for large data sets is entirely unnecessary. In fact, it is unnecessary to hit all islands of most parsimonious trees. If each of the 10 sectors in the tree of Fig. 1 has two global optima (and they are independent), then there are 2^{10} or 1024 global optima not connected through TBR rearrangements. As islands were defined by Maddison (1991), all 1024 combinations of the different optima for each sector are in different islands. However, it is possible to produce a consensus identical to that from the 1024 trees by using only 2 trees, as long as each sector is in a different global optimum in both trees and the trees are collapsed with the TBR algorithm (as in Goloboff and Farris, in press). Therefore, an analysis does not require that all islands are hit, only that minimum length is hit independently a given number of times, to make sure that all optima for each sector are sampled at least once. The real problem is that some optima are more difficult to sample, simply because there are few trees in those optima. This is the same problem encountered by Goloboff and Farris (in press), and it will affect any search method. Moilanen (1999) suggested that his method might find some most parsimonious trees not easily found by other methods, but it seems most likely that his method—just like

any other—will find optima with many trees more commonly than optima with few trees. The best solution for this problem is using a number of hits to minimum length such that even uncommon optima are sampled. Whether minimum length has been hit enough times to produce an accurate consensus can be determined by means of the same criterion that has long been used to determine whether a search is likely to have found minimum length: when a number of additional hits to minimum length does not further de-resolve the consensus, the consensus is stable, and the search is finished. For Zilla, the driver program was used to supervise such a search, re-calculating the consensus every 3 hits to minimum length, until it became stable (for this data set, stability was usually achieved with 12 to 18 hits to minimum length). To make it less likely that errors were produced, the consensus was re-calculated again after having become stable (also every 3 hits to minimum length), until it was stable again or until it had the same number of nodes in the consensus calculated the first time. The final result was calculated as the strict consensus of both consensus trees. The average run time for this procedure was 4.04 h and it produced the exact result (i.e., no spurious nodes at all, of 498) in 9 of 11 cases. The two exceptions had 2 and 1 spurious nodes,[5] thus producing an average error rate (sensu Goloboff and Farris, in press: number of spurious nodes divided by number of nodes recovered) of 0.068%. Note that this is about the same error rate in Goloboff and Farris' method, when high cutoff frequencies are used (the present method uses a strict consensus), and it obviously depends on the structure of Zilla. If the consensus was re-calculated until stability is reached three times instead of two (which would probably take about 6 h instead of 4), the error rate would essentially disappear.

[5]Rice *et al.*'s consensus (based on 8975 trees from a single hit to length 16,220) has 46 spurious nodes (of 434) or an error rate of 10.6%.

APPENDIX 1

C function to select, at random, a sector of the tree having no more than "sector_sz" nodes as terminals and no less than "min_sz." Once the nodes are selected, the function create_matrix( ) re-packs the data (returning 0 if no informative characters are present). The number of terminals in the entire tree is ntax (terminals are numbered from 0 to ntax-1, internal nodes from ntax to root = 2*nt-2). The root node is never selected. Other global variables used are anc[i] (ancestor of node i), list[i] (the ith node in the list of nodes descended from the selected node), clad_sz[i] (the number of terminals included in node i), marker[i] (takes value 2 if node i is selected as terminal for the reduced data set, 0 if node is within the sector chosen but not selected, 1 if node is outside), lefdes[i] and rigdes[i] (the left and right descendants of node i), and inlist[i] (a list for an internal loop).

```
int selectem ( void )
{ int a, x , items, nod, min_sz, cur_sz ;
  min_sz = ( sector_sz * 80 ) / 100 ;
  for ( nod = rand ( ) % root ; clad_sz[ nod ] < min_sz ; ) nod = anc [ nod ] ;
  if ( nod == root ) nod = lefdes [ root ] ;
  if ( !nod ) nod = rigdes [ root ] ;
  items = marknodes ( list, nod, 0, marker ) ;
  for ( a = items ; a-- ; ) if ( list[ a ] < ntax ) marker[ list[ a ] ] = 2 ;
  if ( clad_sz [ nod ] >= sector_sz ) {
    cur_sz = clad_sz[ nod ] + 1 ;
    for ( a = 0 ; a < items ; ++a ) {
      x = list[ a ] ;
      if ( !marker[ x ] && ( cur_sz - clad_sz[ x ] ) + 1 >= min_sz ) {
        marknodes ( inlist, x, 1, marker ) ;
```

```
      marker[ x ] = 2 ;
      if ( ( cur_sz -= ( clad_sz[ x ] - 1 ) ) <= sector_sz ) break ; }}}
  marker [ nod ] = 2 ;
  return ( create_matrix ( list, items ) ) ; }

int marknodes ( int * lst, int from, int val, int * where )
{   int a = 0, b = 1, x ;
    int * dede [ 2 ] ;
    int side ;
    dede [ 0 ] = lefdes ;
    dede [ 1 ] = rigdes ;
    lst [ 0 ] = from ;
    while ( a < b )
         { where [ x = lst [ a++ ] ] = val ;
           if ( x >= ntax ) {
                 side = 1 & rand (  ) ;
                 lst [ b++ ] = dede [ side ] [ x ] ;
                 lst [ b++ ] = dede [ 1 - side ] [ x ] ; }}
    return ( b ) ; }
```

ACKNOWLEDGMENTS

I thank Jim Carpenter, Doug Eernisse, Steve Farris, Kevin Nixon, Claudia Szumik, Ward Wheeler, and Mike Whiting for discussion, comments, and encouragement. Support from PEI 0324/97 and PICT 01-04347/98 was deeply appreciated.

REFERENCES

Chase, M. W., Soltis, D. E., Olmstead, R. G., Morgan, D., Les, D. H., Mishler, B. D., Duvall, M. R., Price, R. A., Hills, H. G., Qiu, Y.-L., Kron, K. A., Rettig, J. H., Conti, E., Palmer, J. D., Manhart, J. R., Sytsma, K. J., Michaels, H. J., Kress, W. J., Karol, K. G., Clark, W. D., Hedren, M., Gaut, B. S., Jansen, R. K., Kim, K.-J., Wimpee, C. F., Smith, J. F., Furnier, G. R., Strauss, S. H., Xiang, Q.-Y., Plunkett, G. M., Soltis, P. S., Swensen, S. M., Willimas, S. E., Gadek, P. A., Quinn, C. J., Eguiarte, L. E., Golenberg, E., Learn, G. H., Jr., Graham, S. W., Barret, S. C. H., Dayanandan, S., and Albert, V. A. (1993). Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcL. *Ann. Mol. Bot. Gard.* **80,** 528–580.

Coddington, J., and Scharff, N. (1994). Problems with zero-length branches. *Cladistics* **10,** 415–423.

Farris, J., Albert, V., Källersjö, M., Lipscomb, D., and Kluge, A. (1996). Parsimony jackknifing outperforms neighbor-joining. *Cladistics* **12,** 99–124.

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

Giribet, G., and Wheeler, W. The position of the arthropods in the animal kingdom: Ecdisozoa, islands, and the parsimony ratchet. *Mol. Phylogenet. Evol.*, in press.

Gladstein, D. (1997). Efficient incremental character optimization. *Cladistics* **13,** 21–26.

Goloboff, P. (1994). NONA: A Tree Searching Program. Program and documentation. Available at ftp.unt.edu.ar/pub/parsimony.

Goloboff, P. (1996). Methods for faster parsimony analysis. *Cladistics* **12,** 199–220.

Kirkpatrick, S., Gellat, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science* **220,** 671–680.

Liebherr, J. K., and Zimmerman, E. C. (1998). Cladistic analysis, phylogeny, and biogeography of the Hawaiian Platynini (Coleoptera: Carabidae). *Syst. Entomol.* **23,** 101–136.

Lipscomb, D., Farris, J., Kallersjo, M., and Tehler, A. (1998). Support, ribosomal sequences and the phylogeny of the eukaryotes. *Cladistics* **14,** 303–338.

Maddison, D. (1991). The discovery and importance of multiple islands of most parsimonious trees. *Syst. Zool.* **40,** 315–328.

Moilanen, A. (1999). Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics* **15,** 39–50.

Nixon, K. C. (1999). The parsimony ratchet a new method for rapid parsimony analysis. *Cladistics* **15,** 407–414.

Platnick, N. (1987). An empirical comparison of microcomputer parsimony programs. *Cladistics* **3,** 121–144.

Rice, K., Donoghue, M., and Olmstead, R. (1997). Analyzing large data sets: rbcL 500 revisited. *Syst. Biol.* **46,** 554–563.

Ronquist, F. (1998). Fast Fitch-parsimony algorithms for large data sets. *Cladistics* **14,** 387–400.

Soltis, D., Soltis, P., Mort, M., Chase, M., Savolainen, V., Hoot, S., and Morton, C. (1998). Inferring complex phylogenies using parsimony: An empirical approach using three large DNA data sets for angiosperms. *Syst. Biol.* **47,** 32–42.

Swofford, D. (1993). PAUP: Phylogenetic Analysis Using Parsimony, version 3.1. Program and documentation. Laboratory of Molecular Systematics, Smithsonian Institution, Washington, DC.

Swofford, D. (1998). PAUP*: Phylogenetic Analysis Using Parsimony (* and Other Methods), version 4. Sinauer Associates, Sunderland, MA.

Swofford, D., Olsen, G., Waddell, P., and Hillis, D. (1996). Phylogenetic inference. *In* "Molecular Systematics" (D. Hillis, C. Moritz, and B. Mable, Eds.), 2nd ed. pp. 407–514. Sinauer Associates, Sunderland, MA.