Cladistics **15**, 407–414 (1999)
Article ID clad.1999.0121, available online at http://www.idealibrary.com on IDEAL®

The Parsimony Ratchet, a New Method for Rapid Parsimony Analysis

Kevin C. Nixon

L. H. Bailey Hortorium, Department of Plant Biology, Cornell University, Ithaca, New York 14853

Accepted October 27, 1999

The Parsimony Ratchet[1] is presented as a new method for analysis of large data sets. The method can be easily implemented with existing phylogenetic software by generating batch command files. Such an approach has been implemented in the programs DADA (Nixon, 1998) and Winclada (Nixon, 1999). The Parsimony Ratchet has also been implemented in the most recent versions of NONA (Goloboff, 1998). These implementations of the ratchet use the following steps: (1) Generate a starting tree (e.g., a "Wagner" tree followed by some level of branch swapping or not). (2) Randomly select a subset of characters, each of which is given additional weight (e.g., add 1 to the weight of each selected character). (3) Perform branch swapping (e.g., "branch-breaking" or TBR) on the current tree using the reweighted matrix, keeping only one (or few) tree. (4) Set all weights for the characters to the "original" weights (typically, equal weights). (5) Perform branch swapping (e.g., branch-breaking or TBR) on the current tree (from step 3) holding one (or few) tree. (6) Return to step 2. Steps 2–6 are considered to be one iteration, and typically, 50–200 or more iterations are performed. The number of characters to be sampled for reweighting in step 2 is determined by the user; I have found that between 5 and 25% of the characters provide good results in most cases. The performance of the ratchet for large data sets is outstanding, and the results of analyses of the 500 taxon seed plant rbcL data set (Chase *et al.*, 1993) are presented here. A separate analysis of a three-gene data set for 567 taxa will be presented elsewhere (Soltis *et al.*, in preparation) demonstrating the same extraordinary power. With the 500-taxon data set, shortest trees are typically found within 22 h (four runs of 200 iterations) on a 200-MHz Pentium Pro. These analyses indicate efficiency increases of 20×–80× over "traditional methods" such as varying taxon order randomly and holding few trees, followed by more complete analyses of the best trees found, and thousands of times faster than nonstrategic searches with PAUP. Because the ratchet samples many tree islands with fewer trees from each island, it provides much more accurate estimates of the "true" consensus than collecting many trees from few islands. With the ratchet, Goloboff's NONA, and existing computer hardware, data sets that were previously intractable or required months or years of analysis with PAUP* can now be adequately analyzed in a few hours or days.

[1]This method, the Parsimony Ratchet, was originally presented at the Numerical Cladistics Symposium at the American Museum of Natural History, New York, in May 1998 (see Horovitz, 1999) and at the Meeting of the Willi Hennig Society (Hennig XVII) in September 1998 in Sao Paulo, Brazil.

0748-3007/99 $30.00

INTRODUCTION

The problem of finding most parsimonious trees is NP-complete (in mathematical terms) and the most effective methods use a brute-force approach of performing branch rearrangements on trees, keeping only the most parsimonious trees or a subset of suboptimal trees at each step. Each tree retained is then swapped (the thoroughness of the search determined by the particular algorithm employed). The simplest method of branch swapping has been termed "subtree pruning and regrafting" or SPR by Swofford (1990). During SPR swapping, a branch is clipped from the tree and placed in each of the possible nodes on the remaining tree, measuring the length of the resulting tree at each step. The SPR algorithm has been shown to be of limited value in finding the most parsimonious trees in large and/or highly homoplastic data sets, although it is effective in improving tree length when starting trees are very suboptimal. In contrast to SPR, the most commonly used and thorough branch-swapping algorithms employed in modern phylogenetic software are termed "branch-breaking" (Hennig86; Farris, 1988) or "tree bisection and reconnection," often referred to as TBR (PAUP; Swofford, 1990). Branch breaking first clips a branch from the tree (the "bisection" of the tree into two parts) and then places the clipped branch at every possible remaining branch of the tree; at each possible place, it reroots the clipped branch to each of the segments, thus vastly increasing the number of topologies examined over SPR. Other swapping methods, such as "nearest neighbor" swapping, are too ineffective to be of much use and will not be discussed.

The time necessary to swap completely through a particular tree increases logarithmically with the number of taxa. This increase is particularly acute with TBR swapping, because the number of trees examined for each bisection is the number of internodes in the main tree (for placement of a clipped branch) multiplied by the number of internodes in the subtree (i.e., the number of terminals in the subtree minus 1). This problem is exacerbated by the tendency in large data sets for there to be extremely large (let us say "vast") numbers of suboptimal trees that are typically very close in both topology and length to the shortest trees (and to any particular tree which is being swapped). This produces a pattern of tree space that is referred to as "islands," e.g., Maddison (1991). Because of the nature of the branch-swapping algorithms, these methods get "bogged down" in large tree islands, collecting trees of equal length that differ by minor rearrangements. If these trees are all kept and swapped themselves, the search stalls and often the user will simply abort the search after a given amount of time has passed (often weeks, months, or even years).

The problem of being trapped in suboptimal islands can be somewhat reduced by implementing strategies which maximize the effort of the tree search algorithms. Several such strategies have been proposed, but the general aspect common to all is to (1) maximize the number of distinct starting trees from which each search is begun (here called a replication); (2) reduce the number of trees kept during each replication, thus minimizing the time spent in any particular island; and (3) collect the results of numerous replications and use a subset of the results as starting points for more thorough (or "complete") analyses, keeping many (or all) trees. I will refer to this approach as the "NONA" strategy since it is most easily implemented in that program.

Even when the NONA strategy is implemented, with large data sets (>500 taxa) considerable time must be expended in order to find solutions in which we have some reasonable confidence. The improvements may be vast over simply running TBR, but the times are still prohibitive with most large data sets. An example is the 500-taxon data set of Chase *et al.* (1993). The original analyses were done with PAUP on a Macintosh Quadra, and after a month of TBR swapping from a single starting point trees of length 16,225 (corrected from the original lengths reported, which were incorrect) were found. Rice *et al.* (1997) reanalyzed this data set, again using the strategy of holding a maximal number of trees and analyzing with TBR, and found trees of length 16,220 (corrected) after 11.6 months of analysis with PAUP on Sun workstations. The Rice *et al.* analysis is flawed for two reasons—first they did not implement any strategy of reducing the number of trees held (the NONA strategy above) and they used software (PAUP) which was orders of magnitude slower than NONA, which was available at the time. Because they allowed the program to hold as many trees as RAM could hold, they were only able to perform eight separate replications (unique starting trees) and ended up collecting more than 8000 trees of length

16,220 (presumably all from one replication, although this was not reported). Nixon and Davis (in preparation) have reanalyzed the 500-taxon data set using NONA on Pentium class computers (no more than twice the speed of the Sun workstations used by Rice *et al.*) and can recover trees of length 16,220 on average every 78 h using the NONA strategy of holding only 2 trees at each replication and then pooling the results of 20 replications and holding 100 trees and performing TBR. By continuing this strategy with the 500-taxon data set, shorter trees (length 16,218) are found ca. every 150 h. Adjusting for the speed of the hardware, this constitutes at least a 50× improvement in speed over the Rice *et al.* analysis.

Based on these and numerous other examples, it is clear that the NONA strategy is much more effective than a nonstrategy such as simply holding all trees and performing TBR (e.g., Rice *et al.*) The reasons for this increase in speed and effectiveness can be explained in numerous ways. More islands are examined because more distinct starting points are used when the NONA strategy is implemented. This is because holding fewer trees decreases the time that any particular replication will spend on an island. The tradeoff is that each replication is less effective on average than a replication that holds more trees. The details of this tradeoff are explored in more detail in Davis *et al.* (in preparation).

THE PARSIMONY RATCHET

The new strategy presented here, the parsimony ratchet, was developed in order to maximize the starting points and reduce the amount of time spent on each search from a particular starting point, while at the same time retaining tree structure from the existing solution at each point. The parsimony ratchet is implemented in the following way:

1. An initial starting tree is generated. Typically, this tree is generated by randomly ordering the taxa, calculating a Wagner tree, and then branch swapping (TBR), holding a few (usually one or two) trees.

2. The tree found in step 1 is used as the starting point for an iterative search strategy as follows.

3. A random subset of the characters is selected and perturbed. Typically, the characters selected would be increased in weight (e.g., weight doubled) or jackknifed (e.g., weight set to zero). Other weighting schemes, such as weighting by fit or the inverse of fit, could also be implemented (but see discussion below). By experience, a relatively small percentage of characters is selected, usually between 5 and 25% of the total number of informative characters.

4. The current tree is swapped using the perturbed weights to calculate length. Typically, TBR swapping will be used (but other swapping procedures could be used). Only one (or few) tree is kept during the search with the perturbed matrix. This search will end with a single tree retained that is "optimal" in the sense that further swapping using the current method will not result in shorter trees.

5. The weights are reset to the original weights (usually equal, but they could be any weighting scheme desired). Using the current tree as a starting point (i.e., the final tree found in step 4) swapping proceeds (holding one or few trees) until an optimal tree is found for the unperturbed data.

6. Go to 3.

The above general method was originally implemented in the program DADA in March 1998 by generating command files for the program NONA. It has since been implemented directly as the command "nixwts" in recent versions of the program NONA (Goloboff, 1998) and again as a batch driver for NONA in the program Winclada (Nixon, 1999). These implementations allow control over the nature of the search (SPR, TBR), the number of iterations performed, the number of characters sampled, and whether characters are weighted up or jackknifed.

An example batch file for NONA for a data set with 10 characters that would perform three ratchet iterations perturbing 20% of the characters each time follows:

```
Wagner; hold 1; max*; // Generate a Wagner tree and TBR swap holding one tree
Sv rat.tre; // Open a tree file to save trees and save the starting tree
// Iteration 1
ccode/1 .; // set all characters to weight 1
ccode /2 0 5; //set character 0 and 5 to weight 2
hold 1; max*; // TBR swap on the current tree using perturbed weights
ccode /1 .; // set all characters to weight 1
```

```
hold 1; max*; / / TBR swap on the current tree using equal weights
sv; / / save the optimal tree for equal weights found in this iteration
/ / Iteration 2
ccode /1 .; / / set all characters to weight 1
ccode /2 6 8; / / set character 6 and 8 to weight 2
hold 1; max*; / / TBR swap on the current tree using perturbed weights
ccode /1 .; / / set all characters to weight 1
hold 1; max*; / /TBR swap on the current tree using equal weights
sv; / / save the optimal tree for equal weights found in this iteration
/ / Iteration 3
ccode /1 .; / / set all characters to weight 1
ccode /2 5 8; / / set character 5 and 8 to weight 2
hold 1; max*; / / TBR swap on tree using perturbed weights
ccode /1 .;/ / set all characters to weight 1
hold 1; max*; / / TBR swap on the current tree using equal weights
sv; / / save the optimal tree for equal weights found in this iteration
sv /; / / close the tree file, which contains the starting tree and 3 trees from the 3 iterations
Keep 0; / / clear the tree buffer
Proc rat.tre; / / read the trees back into NONA
Best; / / filter trees and keep only the shortest
Hold 100; max*; / / optional step . . . continue swapping on best trees holding more trees
```

To generate the above batch file, a program that randomly selects characters from the matrix is necessary. This is implemented in the "island hopper" function in DADA (Nixon, 1998) and as the parsimony ratchet function in Winclada (Nixon, 1999). Note that the batch file could be simplified by removing the redundant hold statements, since the value for hold was not changed after the iterations began. The extra statements were included here for clarity. The above batch file could be changed to a jackknife ratchet by replacing the ccode /1 .; with ccode [.; to activate all characters and ccode] x y; in place of the ccode /2 x y; statements.

I have experimented with various permutations of the basic ratchet algorithm, as outlined above, including weighting characters according to a measure of fit to the tree, but the performance of such variations is less than the original method of randomly selecting and upweighting. However, this does not preclude the possibility that some modification of the ratchet method using fit as a factor in the weighting process might be effective.

One additional modification of the method that does seem to enhance the effectiveness is to randomly constrain a subset of groups during each iteration. This can be accomplished with versions of NONA that allow the constrain command. An effective strategy is to randomly select between 10 and 20% of the nodes and constrain these during the weighted (or jackknifed) search as well as during the equal-weights search of each iteration. At the beginning of each new iteration, a different randomly selected set of nodes from the current tree is selected and constrained. As outlined below, this increases performance considerably on some data sets.

PERFORMANCE

For the purpose of this paper, I will only report the results of analyses of the well-known 500-taxon data set "Zilla" that was first analyzed by Chase *et al.* (1993) and reanalyzed by Rice *et al.* (1997).

The results of three "typical" ratchet analyses of the 500-taxon data set are presented in Figs. 1–4. These charts show the length of the tree found during the equally weighted search at each iteration. The first bar represents the length of the starting tree found by beginning with a Wagner tree generated with a randomized taxon order, followed by a TBR search holding two trees. The first of these two trees was then selected as the starting tree for the subsequent ratchet iterations. Time is not shown on these charts, but 200 iterations typically will complete within 6 h on a 266-MHz Pentium. Figures 1, 2, and 3 show that while the number of characters sampled does change search performance somewhat, typical searches are very effective with roughly 10% character sampling. The random aspect of the ratchet prevents a clear relationship from developing, but the Chase *et al.* lengths were discovered in about a quarter-hour or less and the Rice *et al.* lengths in times between a half-hour and an hour. The shortest trees known are obtained in times roughly between one-and-a-quarter and two-and-a-quarter hours. Figure 4, labeled ineffective, is provided to show one of

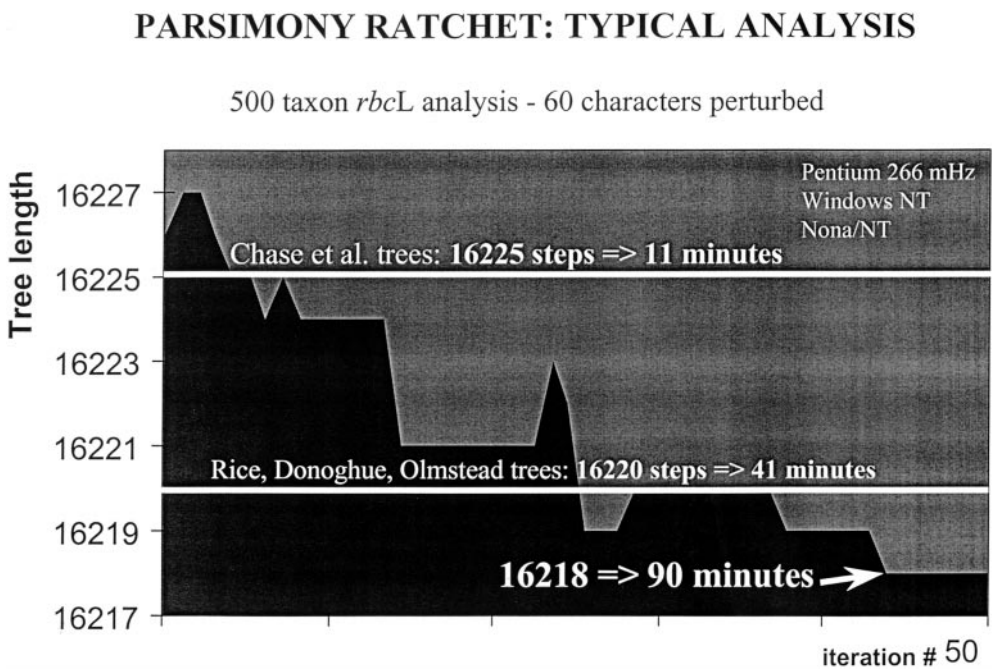

FIG. 1. Performance chart of a typical parsimony ratchet analysis of the 500-taxon rbcL data set. The *X* axis represents successive iterations of the ratchet, from left to right. The first bar represents the length of the starting tree obtained by producing a Wagner tree and performing TBR branch swapping holding two trees. The remaining bars represent tree lengths recovered at each iteration for the equally weighted matrix.

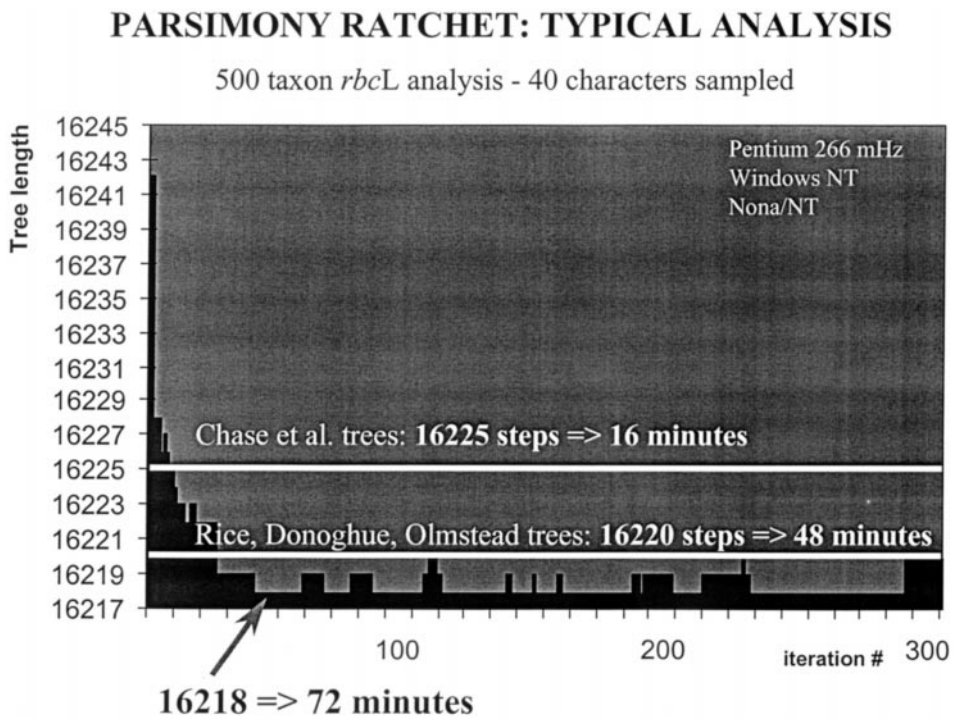

FIG. 2. Another typical analysis of the 500-taxon rbcL data set; legend as for Fig. 1.

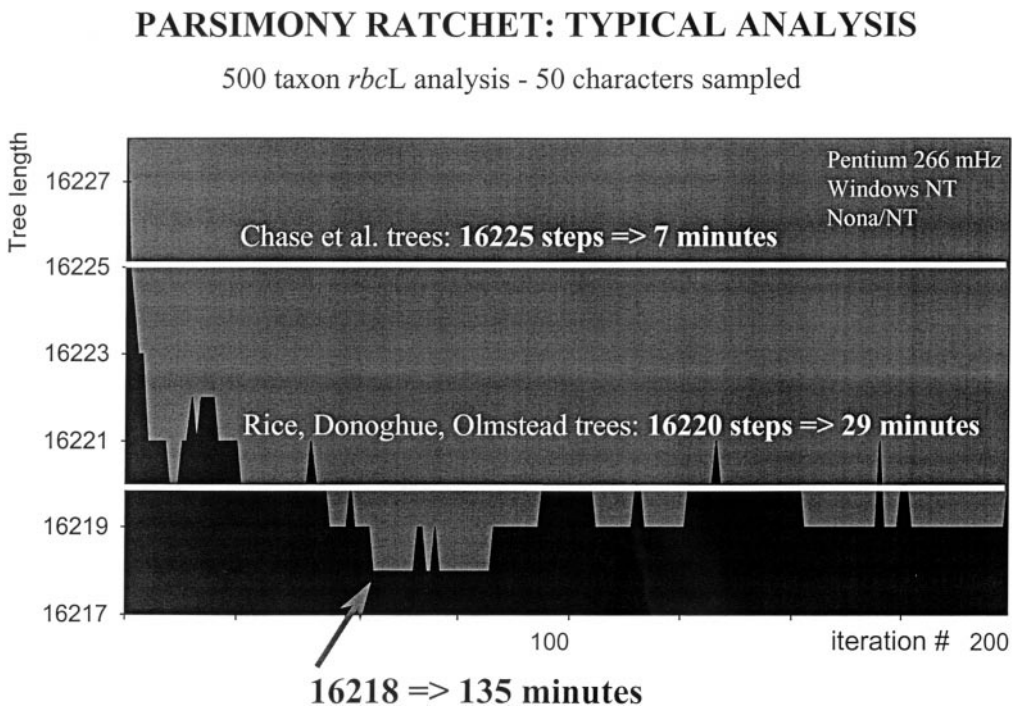

FIG. 3. Another typical analysis of the 500-taxon rbcL data set; legend as for Fig. 1.

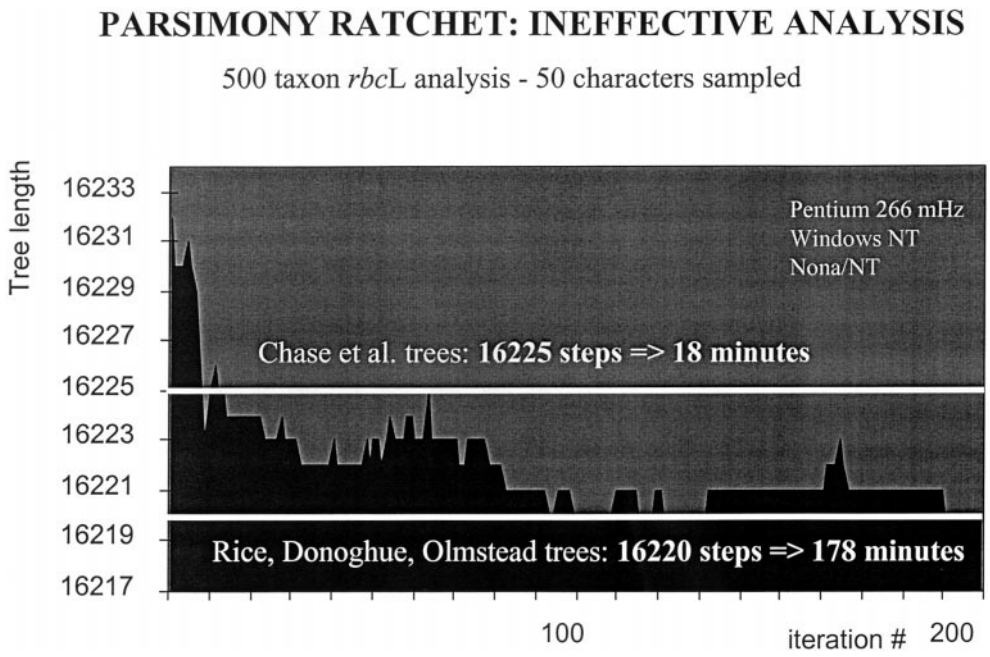

FIG. 4. A "poor" analysis of the 500-taxon rbcL data set; legend as for Fig. 1.

the poorest levels of performance attained with the ratchet with this data set. In this trial, the ratchet found the Chase *et al.* lengths in 18 min and Rice *et al.* lengths in 178 min, but was unable to find the shortest known trees in 200 iterations. Thus, even the ratchet can get stuck on suboptimal islands, and it would be better to implement five trials of 200 iterations than one trial of 1000 iterations.

On a 200-MHz Pentium Pro computer running NONA, a run of 200 iterations holding one tree at each iteration and randomly sampling 70 characters requires approximately 6–8 h to complete. Each iteration takes less than 2 min. The amount of time required is reduced by approximately 30% when nodes are randomly selected and constrained at a level of about 20%; the constraints also generally improve the rate of finding the shortest trees known for this data set. Because approximately three of four runs of 200 iterations attain a length of 16,218, the shortest trees known for this data set, there is a greater than 95% probability that a user would find shortest trees within 22 h. This should be considered in light of the fact that Rice *et al.* failed to find trees of this length in 11.6 months of computer analysis of the same data set. Even accounting for a 2× difference in processor speed, a typical ratchet analysis performs several thousand times better than the Rice *et al.* analysis.

DISCUSSION

There are several ways to explain the effectiveness of the parsimony ratchet in breaking islands and finding optimal trees much more rapidly than previous methods. Probably the most intuitive explanation is that each iteration generates a new "starting tree" that retains much of the information in the last tree found, but is sufficiently different to allow breaking the island within which that previous tree was bound. This new starting tree is found, in the case of the 500-taxon tree, in a matter of seconds (usually less than 1 min) and will generally have a length (when mapped with equally weighted characters) that is within a few steps (or even the same length) as the previous tree. To accomplish the same thing with a traditional method would require on average a minimum of 30–60 min (ca. 15 min to generate a much longer starting tree followed by some significant amount of branch swapping to get within range of the previously found optimal tree). Further, with such an approach, all of the progress made (e.g., the general structure of the tree) must be reconstructed. Thus, the parsimony ratchet can be viewed as a much more effective way to generate new starting trees that retain a significant amount of structure already attained, thus allowing most of the computing time to be spent improving on the current tree and breaking islands.

Other explanations of the effectiveness of the ratchet involve the nature of the tree space of suboptimal trees and the relationships of tree island structures to each other. The "weighted" (or jackknifed) search during each iteration will typically result in between 5 and 10 rearrangements (based on observation of program behavior during runs). These rearrangements, while favored by the particular weighting, move the topology away from the previous topologies with branch moves that would have produced longer trees and thus not be accepted under equal weights. However, these are not random moves, since they produce trees that are more optimal for only slightly perturbed data. By allowing rearrangements to accumulate that are more than a single TBR rearrangement away from the previous topology, the search in essence moves into new tree space and potentially into new islands that may allow access to shorter trees.

At first, there may seem to be a close similarity between the parsimony ratchet and methods such as simulated annealing that utilize a time-dependent probability function to relax periodically the optimality criterion and search longer trees. There are important differences. First, in simulated annealing, in general more trees would be held and thus more time spent on each island. Only after some search effort is expended within a certain island is the criterion relaxed to collect longer trees. However, because these trees are optimal in terms of the same data matrix, they generally will be much more similar to the trees already being swapped. The need to hold more trees, and the fact that merely collecting suboptimal trees is less effective at breaking islands, means that simulating annealing cannot perform as well as the parsimony ratchet. However, one might improve simulated annealing by implementing a periodic reweighting or jackknifing as

the means of relaxing the optimality criterion. It seems unlikely that even with this improvement a simulated annealing approach could reach the efficiency levels of the parsimony ratchet.

Because the parsimony ratchet tends to sample many different tree islands, another advantage is that even with fewer trees collected these typically represent a much broader sample of tree space than can be attained by accumulating many trees from a single island (e.g., as in the case of the Rice *et al.* analysis). This can be shown with an example from a relatively small data set (149 rbcL sequences from Rubiaceae; Ochoterena, in preparation). In this case, the consensus of 50,000 trees collected from 1000 distinct starting points using traditional methods is identical to the consensus of the best trees from less than 200 trees collected from four ratchet runs. Thus, collecting ratchet trees in much smaller numbers is a more effective approach in estimating the "true" consensus than is collecting many more trees from a single island, as is typically done.

In summary, the parsimony ratchet is a novel method that searches tree space more effectively by reducing the search effort spent on generating new starting points and retaining more information from existing results of tree searches. Speed is further enhanced because the method requires holding only one (or few) trees during searching, so time is not spent swapping through almost identical trees. By sequentially generating new trees that differ within a limited number of rearrangements and are optimal for slight perturbations of the data, islands are effectively broken and the efficiency of tree searches is vastly improved. For a given amount of search time, the parsimony ratchet is more likely to encounter shortest trees and collects a broader sample of trees of any given length than previously used search strategies. Such an increase in performance will permit analysis of very large data sets without sacrificing rigor.

ACKNOWLEDGMENTS

I thank D. Lipscomb and J. W. Wenzel for improving the manuscript. I also thank J. Davis, P. Goloboff, J. Carpenter, H. Ochoterena, S. Borgardt, D. Little, and L. Vazquez for help and discussions.

REFERENCES

Chase *et al.* (1993). Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene *rbc*L. *Ann. Missouri Bot. Gard.* **80,** 528–580.

Farris, S. J. (1988). Hennig86. Software and manual. Published by the author, Port Jefferson, NY.

Goloboff, P. (1998). Nona. Computer program and software. Published by the author, Tücuman, Argentina.

Horovitz, I. (1999). A report on "One Day Symposium on Numerical Cladistics." *Cladistics* **15,** 177–182.

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

Nixon, K. C. (1998). Dada ver. 1.9. Software and manual. Published by the author, Trumansburg, NY.

Nixon, K C. (1999). Winclada (beta) ver. 0.9. Published by the author, Ithaca, NY. [Available at http://www.cladistics.com]

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

Swofford, D. L. (1990). PAUP: Phylogenetic Analysis Using Parsimony, ver. 3.0. Illinois Natl. Hist. Surv., Champaign, IL.