Syst. Biol. 56(3):485–495, 2007

ISSN: 1063-5157 print / 1076-836X online
DOI: 10.1080/10635150701431905

On Divide-and-Conquer Strategies for Parsimony Analysis of Large Data Sets: Rec-I-DCM3 versus TNT

PABLO A. GOLOBOFF[1] AND DIEGO POL[2]

[1]*CONICET, Instituto Superior de Entomología, Instituto Miguel Lillo, Miguel Lillo 205, San Miguel de Tucumán 4000, Tucumán, Argentina; E-mail: pablogolo@csnat.unt.edu.ar*
[2]*CONICET, Museo Paleontológico Egidio Feruglio, Av. Fontana 140, Trelew 9100, Chubut, Argentina; E-mail: dpol@mef.org.ar*

Abstract.—Roshan et al. recently described a "divide-and-conquer" technique for parsimony analysis of large data sets, Rec-I-DCM3, and stated that it compares very favorably to results using the program TNT. Their technique is based on selecting subsets of taxa to create reduced data sets or subproblems, finding most-parsimonious trees for each reduced data set, recombining all parts together, and then performing global TBR swapping on the combined tree. Here, we contrast this approach to sectorial searches, a divide-and-conquer algorithm implemented in TNT. This algorithm also uses a guide tree to create subproblems, with the first-pass state sets of the nodes that join the selected sectors with the rest of the topology; this allows exact length calculations for the entire topology (that is, any solution N steps shorter than the original, for the reduced subproblem, must also be N steps shorter for the entire topology). We show here that, for sectors of similar size analyzed with the same search algorithms, subdividing data sets with sectorial searches produces better results than subdividing with Rec-I-DCM3. Roshan et al.'s claim that Rec-I-DCM3 outperforms the techniques in TNT was caused by a poor experimental design and algorithmic settings used for the runs in TNT. In particular, for finding trees at or very close to the minimum known length of the analyzed data sets, TNT clearly outperforms Rec-I-DCM3. Finally, we show that the performance of Rec-I-DCM3 is bound by the efficiency of TBR implementation for the complete data set, as this method behaves (after some number of iterations) as a technique for cyclic perturbations and improvements more than as a divide-and-conquer strategy. [DCM3; heuristics; large data sets; sectorial search; TNT.]

Advancements in nucleotide sequencing technology have produced a dramatic increase in the amount of comparative data suitable for phylogenetic analyses during recent years. This has resulted in phylogenetic data sets with hundreds or thousands of organisms, the earliest example of which is Chase et al.'s (1993) now classic data set of 500 rbcL sequences. Inclusive phylogenetic data sets have been increasingly common in the last few years (e.g., Källersjö et al., 1998; Soltis et al., 1998; Tehler et al., 2003; Hibbett et al., 2005; McMahon and Sanderson, 2006), demonstrating the interest in large-scale phylogenetic analyses among systematists and in attempting to understand relationships between the major branches in the Tree of Life (Cracraft and Donoghue, 2003).

These large phylogenetic data sets exceeded the capabilities of analytical methods of the 1980s and early 1990s, and therefore parsimony analysis of this kind of large data sets had been considered basically intractable. This situation stimulated research in algorithms and implementations capable of conducting efficient parsimony analysis of large data sets. Early contributions along these lines of research have introduced methods that provide speedups of about 30 to 100 times (Nixon, 1999; Goloboff, 1999) over preexisting methods of parsimony analysis; these methods were implemented in the computer program TNT, written by the senior author and collaborators (Goloboff et al., 2003), and available at http://www.zmuc.dk/public/phylogeny.

Rec-I-DCM3 is a method of heuristic tree search recently introduced by Roshan et al. (2004) within the framework of parsimony phylogenetic analysis. This method was developed with the aim of increasing the capabilities of current heuristic tree search analyses and belongs to a family of methods introduced in phylogeny reconstruction by Huson et al. (1999a, 1999b): the disk covering methods or DCMs. Recently, several works claim that Rec-I-DCM3 allows analysis of much larger data sets than ever before (Roshan, 2004, Roshan et al., 2004, followed by Du et al., 2005a; Dotsenko et al., 2006; Smith and Williams, 2006; Warnow, 2005; Williams and Smith, 2005; Williams et al., 2006;). Roshan (2004) and Roshan et al. (2004) compare the performance of Rec-I-DCM3 with the "default technique" of TNT and claim that "Rec-I-DCM3 convincingly outperforms TNT—the best implemented MP heuristic—often by orders of magnitude" (Roshan et al., 2004:99).

In this paper, we reexamine the relative performance of TNT and Rec-I-DCM3 using the two largest benchmark data sets analyzed in the original study, contrasting the divide-and-conquer strategy of Rec-I-DCM3 to sectorial searches (the divide-and-conquer technique implemented in TNT; Goloboff, 1999).

First, we will briefly introduce and compare (in Background; readers familiar with the procedures may want to skip this section) the algorithms used in TNT and Rec-I-DCM3, followed by a brief discussion (under Previous Comparisons) of the problems with the experimental setup used by Roshan et al. for their original (2004) comparisons. Then, a more efficient use of TNT algorithms is described and the two largest data sets used by Roshan et al. (2004) are reanalyzed (under Comparison of Rec-I-DCM3 and TNT). Finally, a discussion on some characteristics of Rec-I-DCM3 performance, and related problems and prospects, is presented.

The following abbreviations are used throughout the text: CSS, constrained sectorial search (where sectors are selected by reference to polytomies in a constraint tree); DFT, tree-drifting; DCM, disk covering method; DS11K, benchmark data set of 11,361 bacterial sequences; DS14K, benchmark data set of 13,921 bacterial sequences; HTU,

hypothetical taxonomic unit (in the present paper, the state set resulting from the first-pass of the optimization); MKL, minimum known length for a given data set; RAS, Wagner tree obtained with a randomized addition sequence; RAT, ratchet; RSS, random sectorial search (where sectors are selected at random); SS, sectorial search; TF, tree fusing (sometimes also used as a verb); XSS, exclusive sectorial search (where nonoverlapping sectors are selected so that their size is as uniform as possible, and they cover the entire tree). Following Roshan (2004) and Roshan et al. (2004), Rec-I-DCM3$_{(X)}$ stands for method of analysis X, boosted with Rec-I-DCM3 (i.e., method X used to analyze subproblems); analogously, SS$_{(X)}$ stands for method X used to analyze subproblems in a sectorial search.

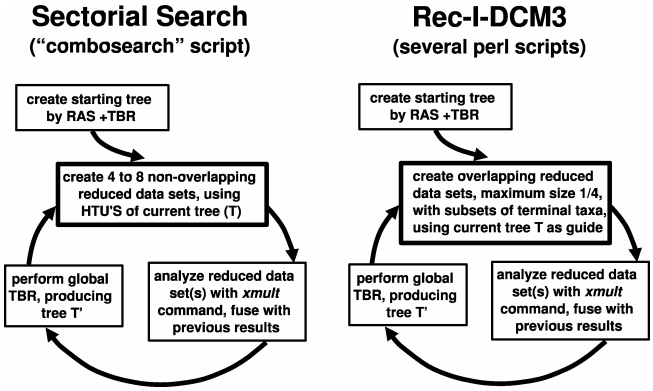

FIGURE 1. General organization of sectorial searches and Rec-I-DCM3. Both sectorial searches and Rec-I-DCM3 have a similar structure, with the only important difference being in how the reduced subproblems are created (by means of firstpass HTUs, in the case of sectorial searches, and selecting disjoint but overlapping subsets of terminals, in the case of Rec-I-DCM3).

BACKGROUND

Algorithms in TNT

As recognized in recent reviews (Hovenkamp, 2004; Giribet, 2005; Meier and Ali, 2005), one of the main qualities of TNT is the availability of fast heuristic tree searches. TBR branch swapping uses improved programming techniques in TNT; for the two benchmark data sets analyzed here, the difference in speed with competing software is quite drammatic: PAUP* (Swofford, 1998) takes about 300 times longer than TNT to complete TBR in the case of DS11K and about 900 times longer in the case of DS14K (note that the difference in speed with TNT is much less in smaller data sets). As discussed below, the efficiency of TBR in large trees is key to the success of Rec-I-DCM3. In addition to fast TBR algorithms, TNT also implements the tree search algorithms proposed by Goloboff (1999) and Nixon (1999). These algorithms consist of two different types: algorithms that improve preexisting trees (SS, DFT, RAT) and an algorithm that takes sets of trees and mixes them, producing combined trees (TF). All these algorithms are combined in TNT in what is called a "driven search." We provide a brief overview below.

Sectorial search.—This algorithm (see Fig. 1) is the most relevant for the purpose of this paper, given that it is a divide-and-conquer approach, just like Rec-I-DCM3. SS consists of dividing a tree in "sectors" and creating reduced data sets based on them. A sector comprises a monophyletic group, with some of the internal nodes within the group possibly replacing all the terminal taxa descended from them. Each sector is analyzed separately and the new configuration of the sector is regrafted on the original tree if a better solution is found. The sectors can be chosen at random (RSS) or based on a set of constraints calculated automatically (CSS). A third type of sector selection (XSS, implemented in TNT during 2004) selects large sectors of even size, which cover the entire tree but do not overlap. Regardless of how the sectors are chosen, the creation of the reduced data set is the same. SS creates reduced data sets in which the OTUs are the terminal taxa included in the sector and the HTUs that join the chosen sector with the rest of the tree. The scorings of the HTUs in the reduced data set are represented by the first-pass state sets obtained during optimization of the tree. Obtaining the scorings for the most basal node of the sector requires rerooting the tree before performing the optimization, such that the basal node becomes sister of the sector. When a reduced data set is created in this way, finding a solution N steps shorter than the original subtree topology automatically implies a solution that is N steps shorter for the entire data set. This critical characteristic of SS stems from considering the scorings of the internal nodes (first-pass states) that join the chosen subtree with the rest of the tree and allows regrafting only those changes that actually improve the optimality of the entire tree. A replacement that improves tree score in a sector may also imply that other parts of the tree can now be changed to further improve score, and thus SS uses periodic rounds of global TBR (see Goloboff, 1999). In the implementation provided by TNT, several parameters can be modified, including the size and number of the sectors to be chosen and the type of search to be done for each sector (including the possibility of recursively subdividing the sector with additional SS).

Drifting and ratchet.—These algorithms use cycles of TBR swapping (on a single tree), alternating between perturbation and search phases. As implemented in TNT, the perturbation phase consists of either accepting suboptimal rearrangements with a probability that depends on the tree-score difference (DFT) or accepting the rearrangements that match or improve tree score for a randomly modified set of character weights (RAT).

Tree fusion.—This algorithm evaluates exchanges between two trees for all subtrees with identical composition, accepting those that improve the score; if the first-pass state sets (and local costs) have been calculated for each subtree, the score resulting from an exchange can be calculated quickly visiting only the nodes in the path between the exchange point and the root (Goloboff, 1999).

If the trees to input to TF are suboptimal trees obtained independently (e.g., from different starting points), TF produces score improvements almost immediately.

Driven search.—The main conclusion of early tests of these algorithms (Goloboff, 1999) was that optimal trees for data sets of 500-854 taxa were quickly found when TF was applied to trees resulting from the other algorithms. In order to find the minimum known length (MKL) in the shortest amount of time, a "driven search" seeks to minimize the number of trees obtained in independent replicates (i.e., RASs, followed by TBR, SS, and DFT/RAT) that are input to TF. Thus, the trees produced by the first few replicates are submitted to TF; if these trees fail to produce trees of MKL, some additional replications are done, submitting the new set of trees to TF. This strategy is implemented in TNT (as a driven search, or as an option of the *xmult* command) and creates trees by RAS, followed by TBR, SS, and optionally DFT or RAT. With the default settings, the *xmult* command will perform five RAS, followed by SS (no DFT or RAT), TF those five trees, and stop; only if the user has requested that the search continue until a given score is found and TF on the first five trees fails to produce such a score is a new series of replicates started (which will be subsequently used for TF). In addition to this basic flow control, TNT allows defining a stopping rule by the number of independent hits to MKL, or until the consensus is stabilized, or much more specific routines through its scripting language.

Background on Rec-I-DCM3

Rec-I-DCM3 is the latest and most successful version of DCMs. The method was published by Roshan et al. (2004) but originally developed in Roshan's Ph.D. dissertation (Roshan, 2004). This procedure (see Fig. 1) is based on the recursive and iterative application of the DCM3 decomposition, selecting subsets of terminal taxa based on a given tree topology (i.e., "guide tree"). Each of these subsets of terminal taxa is used to create reduced data sets (subproblems); if the first subdivision of the data set produces subproblems that are too large, the method is recursively applied to the large subproblems, until they are below a (user-specified) maximum size. Based on empirical tests of performance, Roshan (2004) concluded that large data sets should be divided in subproblems no larger than 25% or 12.5% the total number of taxa. Each of the subproblems is submitted to a program for parsimony analysis (Roshan [2004] tried both PAUP* and TNT). The method of analysis employed for solving the subproblems is referred to as the "base method." After the subproblems are solved, the Rec-I-DCM3 program collects the trees resulting from PAUP* or TNT and combines them using a strict consensus supertree method (see Huson et al., 1999a, 1999b). The resulting supertree normally contains multiple polytomies. At this point, the polytomous tree is randomly resolved (i.e., dichotomized) and submitted to TBR-branch swapping (again, using either PAUP* or TNT). We refer to this key step of Rec-I-DCM3 as the global TBR (Roshan et al. [2004] omitted mentioning the global TBR phase, only stating that a "random resolution, to obtain a binary tree" is applied to the polytomized supertree; Roshan's thesis (2004:152) and the Rec-I-DCM3 software make it clear that this phase is part of the method). After global TBR concludes, the first iteration of Rec-I-DCM3 is finished, and the resulting tree is used as the starting point (guide tree) of the next iteration.

Differences between Sectorial Searches and DCM3

At first, it might seem that DCM3 is simply equivalent to SS. The two methods, however, differ in one critical aspect. The subproblems in DCM3 are created by selecting (partly disjoint) sets of terminal taxa and, therefore, finding a better solution for the subproblem does not necessarily imply that the solution will improve the score of the entire tree (i.e., considering all the taxa). Additionally, if conflicting solutions are found for the different subproblems, merging the solutions together will create a tree with polytomies that are randomly dichotomized (i.e., not resolved in their optimal resolution). It is for these two reasons that the round of global TBR swapping is needed after recombining the solutions for the subproblems: the tree resulting from the recombination is usually longer, not shorter. Without global TBR, Rec-I-DCM3 produces very poor results.

As described above, SS divides the tree into subproblems formed by sectors of the tree considering first-pass state sets for internal nodes. This ensures that finding an improved score for the subproblem results in a better score for the entire tree. This simply follows from the algorithms for Fitch/Farris optimization (Farris, 1970; Fitch, 1971), which work by first assigning preliminary state sets in a down pass. Using those same state sets to represent the ancestral nodes for some groups will produce length evaluations identical to those of a Fitch/Farris optimization, for any rearrangement of the relationships between those groups (as long as a constant number of steps, corresponding to the steps occuring within the groups, is added). Therefore, SS differs from DCM3 in that improved scores for the entire tree are generally found even without global TBR (although global TBR may, of course, further improve tree scores).

The original implementation of SS analyzed relatively small sectors with simple algorithms, such as six DFT cycles or six RAS+TBR (depending on the number of nodes present in the sector). This early implementation of SS is therefore different from Rec-I-DCM3, which can select sectors comprising large numbers of terminals, and analyzes each of those with rather exhaustive algorithms (i.e., the *xmult* command of TNT, which uses several RAS, SS, and TF), possibly giving an advantage to Rec-I-DCM3.

Those SS settings work efficiently for data sets below 1000 taxa, but for larger data sets it becomes necessary to create very large sectors to cover significant portions of the tree, and (as anticipated by Goloboff, 1999:420) these in turn have to be analyzed with algorithms appropriate for their size. In order to expand the capabilities of SS, the scripting interface of TNT has been modified

(during 2004–2005) for allowing full control of the search algorithms used to resolve each sector. The use of these modifications is discussed below (under Materials and Methods).

Previous Comparisons

The only published comparisons between the performance of TNT and Rec-I-DCM3 have been conducted by Roshan (2004) and Roshan et al. (2004). These studies compared the results obtained by both procedures in the analysis of 10 large data sets, ranging from approximately 1300 to 14,000 taxa. The performance of Rec-I-DCM3 was compared with that of TNT "with default settings" on five runs that extended for 24 h. The basic result of these comparisons was that TNT never matched the score found by Rec-I-DCM3. Furthermore, some of the TNT runs were extended over six more days, which led Roshan et al. (2004) to conclude that Rec-I-DCM3 was able to reach a given score much faster than TNT "with default settings." The largest difference was obtained in the largest data sets, in which the score found by the week-long TNT run was found between 10 and 50 times faster by Rec-I-DCM3.

The higher performance of Rec-I-DCM3 over TNT led Roshan (2004) to suggest that "a DCM3-based divide and conquer strategy is more effective than the one used in TNT-default" (i.e., sectorial search). Given the key differences between DCM3 decomposition and SS described above, this seems an unlikely explanation of the difference in performance. Unfortunately, the published studies did not specify what was meant by running TNT "with default settings." Being fully interactive, TNT does not have a "default" heuristic search. Roshan (personal communication) indicated that the *xmult* command was repeated many times (via a loop TNT command, changing the random seed). Running this routine TNT effectively produces, as previously reported by Roshan et al., much poorer results than obtained with Rec-I-DCM3 (see Figs. 2, 3).

This strategy, however, is an extremely poor use of the algorithms implemented in TNT. As explained above (and in the documentation of TNT), the *xmult* command, unless otherwise specified, will conduct only five replicates of RAS+TBR+SS (with no DFT/RAT), TF the trees obtained by these replicates, and stop. As is clear in the original publication of these algorithms (Goloboff, 1999; see also Goloboff, 2000), the success of combining these search strategies lies in repeatedly inputting to TF gradually increasing sets of trees (resulting from independent replicates of RAS+TBR+SS, and DFT or RAT). Thus, the repeated TF process will gradually produce better and better trees. Roshan et al.'s routine, however, prevents this from happening, since it discards previous results every time a new *xmult* run is begun from scratch (Fig. 2). Whether or not a given instance of *xmult* reaches a low score is independent of the scores obtained in previous instances. TF is used, every time, to fuse the new set of trees obtained by RAS+TBR+SS but ignoring all the trees obtained in previous replicates and rounds of TF.

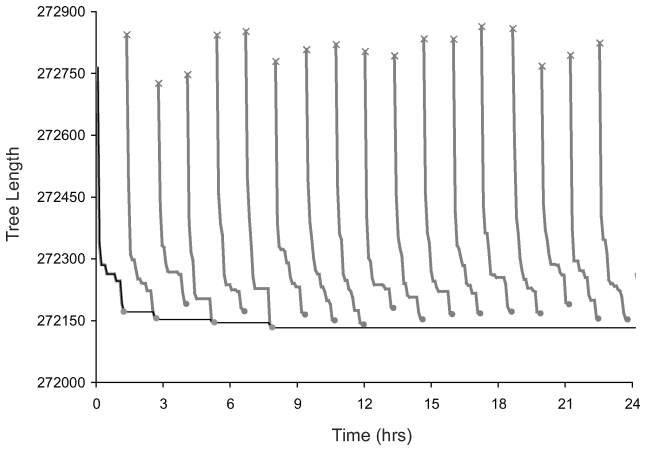

FIGURE 2. Tree score trajectories of TNT searches conducted as in Roshan et al. (2004) for the data set DS-11K (11,361 taxa). Gray lines represent the successive improvements in tree length found during a single execution of the *xmult* command. The starting point of each line is marked with a cross and the final tree obtained after TF is indicated with a solid circle (tree that is subsequently discarded). The thick black line indicates cumulative scores (i.e., the best score found in all previous cycles), equivalent to the "TNTdefault" curve presented in the study of Roshan et al. (2004).

TNT would have performed much better if the previous trees, instead of being discarded, were fused to the new trees obtained (i.e., if the trees in the solid circles of Fig. 2 are pooled and submited to TF). When the repeated fusing procedure is introduced in the settings for running TNT, its performance differs much less markedly from that of Rec-I-DCM3 (Fig. 3), although their relative superiority becomes data set dependent (data not shown).

Regarding the extended TNT runs that Roshan et al. (2004) continued for a week, it must be emphasized that they are not particularly meaningful or informative,

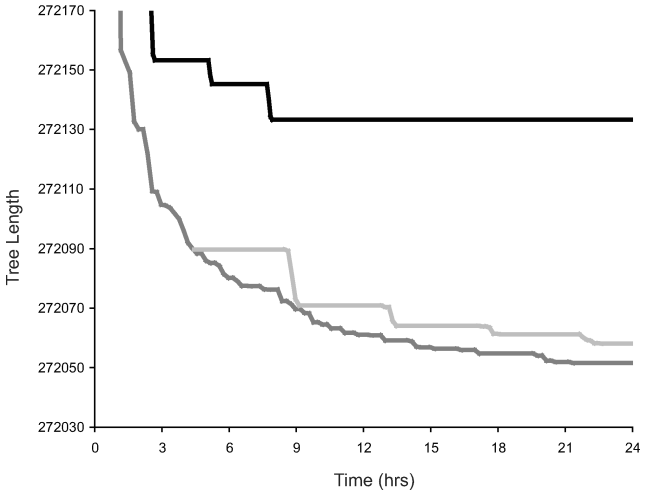

FIGURE 3. Tree score trajectories for the data set DS-11K (11,361 taxa). Scores for TNT search conducted as in Roshan et al. (black line, a single run), for TNT performing repeated fusing of the trees obtained in different replicates (light gray line, average of 10 runs), and for Rec-I-DCM3 (dark gray line; maximum subproblem size 30%, using "xmult"; and "tfuse"; to analyze each subproblem, average of 10 runs).

since the results obtained from them systematically threw away trees that were excellent input candidates for TF and would have gradually improved the best score. Note, for instance, the flattening (Fig. 3) of the TNT run with Roshan's procedure (black line) as opposed to the TNT run with repeated TF that kept improving the score (light gray line) as new trees obtained from additional executions of the *xmult* command were fused with those previously obtained.

Finally, we must note that the importance of repeated TF was clear in the original publication of these algorithms (Goloboff, 1999), and that conducting the repeated TF procedure shown in Figure 3 does not imply elaborate scripts or obscure command options (the *xmultloop* script [see Materials and Methods] was used here to produce those runs; alternatively, executing the command "xmult = replications 12 target 1" would suffice to obtain similar results). Although Roshan et al. did not use repeated TF for the TNT runs, they in fact used it for the analyses of the subproblems in their Rec-I-DCM3 scripts.

The simple and more reasonable settings for TNT discussed above usually work as well as Rec-I-DCM3 for some data sets, although for others (e.g., the 13,921 taxon data set tested by Roshan and collaborators) there is still a significant difference in favor of Rec-I-DCM3. In the following sections, we discuss better settings for conducting efficient heuristic tree searches in TNT using its scripting language and the same set of basic algorithms (e.g., TBR, SS, DFT, TF).

Comparison of REC-I-DCM3 and Sectorial Searches

Materials and Methods

Data sets.—We focus our comparisons on the two largest data sets used by Roshan (2004) and Roshan et al. (2004), because in their runs these are the data sets for which TNT performed the poorest, in comparison to Rec-I-DCM3. These data sets have 11,361 and 13,921 taxa, respectively, and were downloaded from http://www.cs.njit.edu/usman/software/recidcm3. We refer to these data sets as DS-11K, and DS-14K, respectively. Having only 1360 sites, both data sets are very poorly structured and are probably of not much interest besides their utility for benchmarking search algorithms. MKL for these data sets was obtained with TNT running alone: for DS-11K, 272,007 steps (73 steps shorter than reported by Roshan, 2004), and for DS-14K, 240,860 steps (118 steps shorter than reported by Roshan, 2004). Trees with those scores can be obtained from the first author on request. These scores may not be truly optimal, as they were found only once. Roshan et al. reported a 50-fold speed difference between Rec-I-DCM3 and TNT for another, smaller data set with about 7000 taxa, but this data set is not publicly available and we have not been able to test it.

Platform.—All the runs reported here for both TNT and Rec-I-DCM3 were done on single-processor 3-GHz Pentium IV machines, with 1-Gb RAM, running under Suse Linux 9.1. All the analyses conducted here were run on dedicated processors on which no other jobs were run at the same time.

TNT analyses.—The new analyses conducted on TNT v. 1.0 (November 2005) were performed using two scripts, publicly available at http://www.zmuc.dk/public/phylogeny/TNT/scripts. One of the scripts (*xmultloop.run*) includes the simple example of repeated TF discussed above (Fig. 3), the other (*combosearch.run*) implements a more elaborate strategy developed to better analyze extremely large data sets (such as DS-11K and DS-14K). The latter script makes use of some of the modifications implemented to SS in TNT mentioned above, including the selection of large sectors that can be analyzed using any tree search command. We must note that this particular implementation of SS (although not yet available in TNT when Roshan et al. [2004] conducted their analyses) does not involve any algorithms different from the ones described by Goloboff (1999).

The script used is structured as shown in Fig. 1; the general organization of the script is similar to that of Rec-I-DCM3, highlighting the similarity between SS and Rec-I-DCM3 (the only significant difference being how the subproblems are created). Both Rec-I-DCM3 and SS used the *xmult* command (with identical parameters) to analyze subproblems, so that this is a direct comparison of Rec-I-DCM3$_{(xmult)}$ versus SS$_{(xmult)}$. The basic idea in the script is to repeatedly divide the tree in large nonoverlapping sectors that cover the entire tree (with XSS) and to do a relatively exhaustive search on each sector.

This script requires three search parameters to be specified by the user. The first two parameters are a minimum and a maximum number of sectors in which the tree will be divided. A random number is selected between these limits in each of the iterations of the tree search (so that limits between sectors vary with iterations). The third parameter is the minimum decrease in steps in a given round (or MSD, see below). In this script, each sector is analyzed with the *xmult* command with default parameters (using two rounds of fusing after the multiple RAS for each sector), except for subproblems above 750 taxa, in which the SS search performed by *xmult* subdivides the sector being analyzed into subsectors (with XSS; this subdivision is performed three consecutive times, starting with subsector sizes of 200 nodes and ending with subsectors of 80). After the *xmult* command is executed, TF merges the previous solution of the sector with the new one (ensuring the new solution will be no worse than the previous one). Once the new resolutions are obtained for all the sectors, a round of global TBR is applied to the entire tree. This process is repeated as long as the number of steps saved by analysis of the large sectors equals or exceeds the minimum step difference, MSD (specified by the user as the third parameter for the script). When three rounds of this process fail to match or exceed the minimum step difference MSD, the process of sectorial improvements is interrupted (i.e., the cycle shown in the flowchart of Fig. 1 is broken), the trees obtained in previous rounds (if any) are pooled with the new tree and subjected to TF, and then a new round of the entire script is restarted.

For restarting these successive rounds, the script allows either (a) starting a new build (which takes a long time, initially, but eventually may contribute more diverse trees to use for subsequent TF), or (b) trying to escape local optima by resolving large sectors of the tree as described above, but ignoring the previous resolution. In the runs reported here we have used option (b), which in fact may be convenient for extremely poorly structured data sets, such as DS-14K, because then the trees submitted to TF tend to share more groups (and then the exchange of subgroups is more extensive). The script is a text file (240 lines) that can be easily modified by the user to suit particular needs or to explore more sophisticated strategies.

Rec-I-DCM3 analyses.—The binaries and Perl scripts presented by Roshan et al. (2004), publicly available at http://www.cs.njit.edu/usman/software/recidcm3, were used for the Rec-I-DCM3 runs. For solving each subproblem, Roshan's original scripts call TNT to execute the *xmult* command with default options (i.e., to do five replicates as specified in the previous section). These scripts use TF to fuse the new resolution of the subproblem with the previous one, thus ensuring that the resolution of the subproblem returned by TNT is, for the subset of terminals, not worse than the previous one. Roshan's scripts with the original subproblem tree search settings were used for the Rec-I-DCM3 runs shown in Figure 3 (using maximum subproblem size equal to 30% the total number of taxa), as well as those shown in Figures 8 and 9.

In order to provide comparable performance curves on the divide-and-conquer methods of DCM3 and SS, the Perl scripts to analyze the subproblems created by Rec-I-DCM3 were modified. Each of the DCM3 subproblems was analyzed (in TNT) with the same tree search commands described above for the sectorial searches in the TNT script and the maximum subproblem size was set to 25% the total number of taxa (the size recommended by Roshan [2004]). All other settings of the original Rec-I-DCM3 were left unaltered (including the global TBR search conducted at the end of each iteration, and TF with previous resolution at the end). These settings were employed for the runs shown in Figures 4 through 7.

All the Rec-I-DCM3 analyses on the DS-K11 and DS-K14 data sets were conducted again in order to provide comparable results, specially given the differences in software and hardware between our study and the studies by Roshan (2004) and Roshan et al. (2004). Software differences are due to the version of TNT used here, which has been further improved since 2004. For large data sets, TBR branch swapping is now, due to code improvements, about two times faster than in early 2004. These improvements include faster TBR swapping on optimal or near-optimal trees, faster handling of constraints when finding better trees under TBR (this strongly affects autoconstrained *xmult* searches), and faster transition to a better tree during TBR. The last two affect specially the initial stages of TBR swapping, when better trees are found very often.

The hardware used here is also faster. Roshan et al. used processors of 1.5-GHz or less, whereas we use 3-GHz processors. Therefore, our Rec-I-DCM3 runs perform much better than in Roshan (2004). For each of the data sets, the best score out of the five 24-h runs reported by Roshan (2004) is matched (on average) by our Rec-I-DCM3 runs in 6 h or less. Thus, every 24 h of Rec-I-DCM3 runtime in the present paper is actually equivalent to approximately 4 days with the settings of Roshan (2004). Du et al. (2005a) and Dotsenko et al. (2006) give results for a parallel Rec-I-DCM3, but their reported best scores at 24 h are worse than the best scores reported by Roshan (2004) and much worse than the average score obtained here for Rec-I-DCM3.

Unless stated otherwise, the analyses were repeated 10 times (both for TNT and Rec-I-DCM3) in order to take into account the variability of results. The entire set of runs reported here took a total of approximately 6 months of CPU time.

Results

When the divide-and-conquer strategy of TNT described above is used to improve upon trees obtained by a single RAS+TBR+SS, the results obtained in the analyses of the data sets DS-11K and DS-K14 clearly outperform those of Rec-I-DCM3 (Figs. 4 and 5). Recall that for these comparisons, the Rec-I-DCM3 subproblems were analyzed (in TNT) with the same search commands used for the sectors divided by SS in TNT. Therefore, the only important difference between the TNT and Rec-I-DCM3 runs was in how the subproblems were created: by selecting HTUs, in the case of SS, and by selecting subsets of terminal taxa in the case of Rec-I-DCM3. For both data sets, after 50 h, the score of the best Rec-I-DCM3 run was worse than the average TNT score, and the score of the worst TNT run was better than the average Rec-I-DCM3

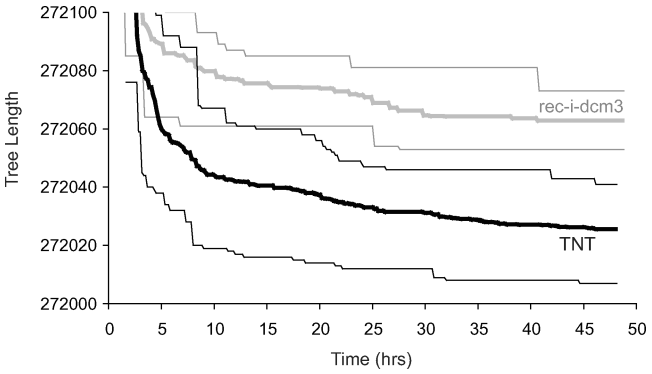

FIGURE 4. Analysis of data set DS-11K (11,361 taxa). Cumulative scores obtained by Rec-I-DCM3 (gray) and TNT (black). Thick curves represent average scores (of 10 independent runs), and thin lines represent minimum and maximum (the best and worst score of all the runs at any given point). For the TNT script the parameter values for the min/max numbers of sectors were set to 4 and 8 and the minimum step difference (MSD) was set to 10 (See Materials and Methods). Rec-I-DCM3 was run with the same search commands for subproblems and a maximum subproblem size of 25%.

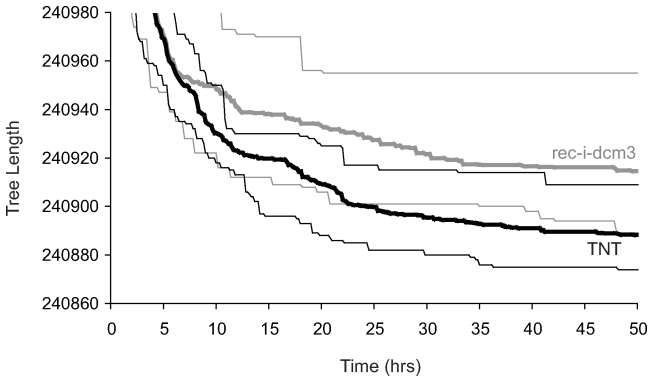

FIGURE 5. Analysis of data set DS-14K (13,921 taxa). Cumulative scores obtained by Rec-I-DCM3 (gray) and TNT (black). Thick curves represent average scores (of 10 independent runs), and thin lines represent minimum and maximum (the best and worst score of all the runs at any given point). For the TNT script the parameter values for the min/max numbers of sectors were set to 4 and 14 and the minimum step difference was set to 5 (See Materials and Methods). Rec-I-DCM3 was run with the same search commands for subproblems and a maximum subproblem size of 25%.

score. In the case of DS-11K (Fig. 4), it takes TNT about 4.5 h to better the average Rec-I-DCM3 score obtained at 24 h (5.3 times faster) and about 5 h to better the average Rec-I-DCM3 score obtained at 48 h (9.6 times). Note that Figure 4 shows strictly comparable runs (using the same commands to resolve subproblems and a subproblem size closer to recommended); the Rec-I-DCM3 run shown in Figure 3 (which used subproblems larger than recommended, of about 30% the total number of taxa) performed better, but even in that case TNT reaches the score obtained by Rec-I-DCM3 at 24 h in only 8 h (3 times faster). In the case of DS-14K, it takes TNT 11 h to better the average Rec-I-DCM3 score obtained at 24 h (2.2 times faster) and 18 h to better the average Rec-I-DCM3 score obtained at 50 h (2.8 times faster).

In sum, in both the DS-11K and DS-14K data sets, TNT's own divide-and-conquer technique (SS) produces better results than the decomposition introduced with Rec-I-DCM3 (applying the same search strategy for the subproblems). This contradicts the main results of Rec-I-DCM3 comparisons published by Roshan et al. (2004) and echoed by several recent contributions (Williams and Smith, 2005; Du et al., 2005a; Dotsenko et al., 2006; Williams et al., 2006). The precise speed difference between these two methods is likely to be dependent on the combination of the particular data set being analyzed and the specific parameter values for the tree search algorithms. Irrespective of these variations, however, it is clear from our results that the "dramatic speedup" reported by Roshan et al. (2004) and their conclusion that Rec-I-DCM3 "convincingly outperforms" the algorithms implemented in TNT were based on an extremely poor design of the TNT runs.

DISCUSSION

As noted above, the fact that SS allows calculating tree lengths exactly while analyzing subproblems is a key advantage that distinguishes it from the Rec-I-DCM3 decomposition. The only exception to this can occur when several SS subproblems are resolved independently at the same time (although during our analyses it was extremely rare that the tree combining all the improved resolutions was longer than the original one). Even in those cases a quick round of global TBR usually saves only a few additional steps (mostly during the initial cycles). In contrast, finding improved solutions for the subproblems in Rec-I-DCM3 normally leads to worsened, instead of improved, global solutions, which require subsequent global TBR swapping. This has several negative implications for the performance of Rec-I-DCM3.

Rec-I-DCM3 Is More "Ratchet" Than "Divide-and-Conquer"

The Rec-I-DCM3 graphs of Figures 3 through 5 are similar to those used by Roshan (2004) and Roshan et al. (2004), with the scores monotonically decreasing with time, because they represent the cumulative best scores of all the preceding Rec-I-DCM3 iterations. Cumulative score curves (as shown for the TNT runs in Fig. 2) show only partial information on the dynamics of an algorithm performance. After some initial cycles, a given iteration of Rec-I-DCM3 ends up producing (*after* global TBR), about 50% of the time, a tree worse than the one resulting from the previous iteration (Fig. 6). After the first few hours, the actual trajectory of Rec-I-DCM3 for DS-11K (Fig. 6a) is clearly not a decreasing one (in two cases the best score was found within the first 15 h, and never again in the subsequent 35 h). In the case of DS-14K (Fig. 6b), the post-TBR trajectories decrease for a longer time than in DS-11K, but that is only an effect of the larger size (and difficulty) of DS-14K; after more hours, the post-TBR trajectory also begins to oscillate around a given score (about 240,930 or 240,940 steps).

The oscillating nature of Rec-I-DCM3 runs is more evident when a very good initial tree is taken as the starting point. We let Rec-I-DCM3 run for 520 h on the DS-14K data set (using an initial guide tree of 240,873 steps, found by TNT), repeating this procedure 20 different times with different random seeds. The post-TBR trajectories quickly show a worsening of the trees obtained as more iterations of Rec-I-DCM3 are completed (top curves of Fig. 7) and the starting tree score is never found again. After the first 5 h, the post-TBR trajectories oscillate around 240,930 steps, showing no trend of improvement. This behavior clearly contrasts with the divide-and-conquer SS algorithms of TNT, which quickly (2.5 h, on average) improve the score when the same tree is input as the initial solution (bottom curves of Fig. 7; only best and worst of 10 TNT runs are shown there). The trees resulting from these ten TNT runs were subsequently subjected to TF, producing trees of the MKL for DS-14K: 240,860 steps.

Except for the first few iterations of Rec-I-DCM3 (when the tree is very far from optimal), dividing the tree in subproblems and merging the solutions into a supertree that is randomly dichotomized creates a perturbation to the preceding topology, instead of an improvement.

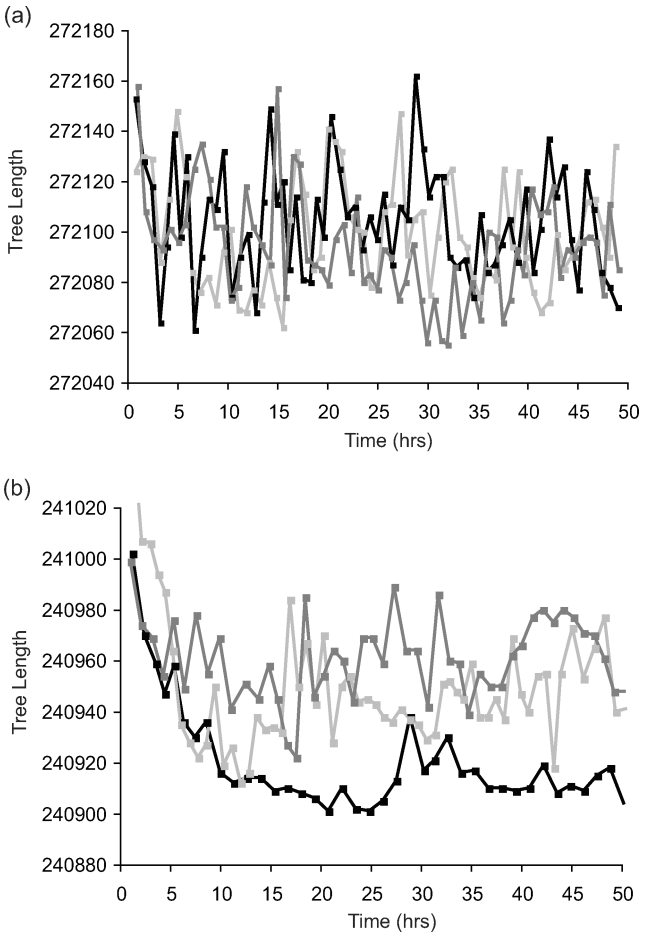

FIGURE 6. Trajectory of tree scores obtained by Rec-I-DCM3. Tree search for subproblems and maximum subproblem size as in Figures 4 and 5 (see Materials and Methods). Each point of the curves represents the tree score of the tree obtained at the end of a Rec-I-DCM3 iteration (after global TBR). (a) Three different runs for the data set DS-11K (11,361 taxa); the general trend (after the first few hours) is clearly not decreasing (e.g., the runs that found the best score at about 89 h resulted in worse trees in the remaining 1516 h). (b) Three different runs for the data set DS-14K (13,921 taxa). The gray and light gray lines show their minimum scores found at 11 and 17 h (respectively) and never again in the remaining 33 or 39 h.

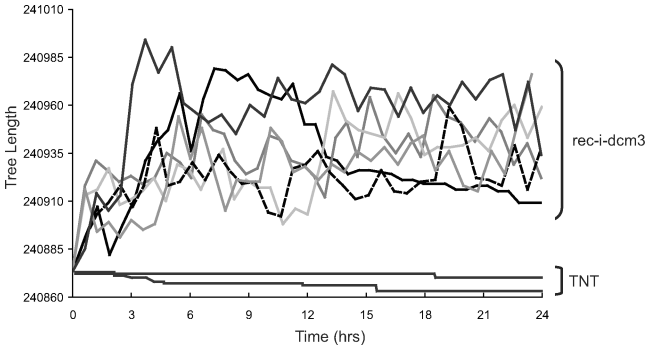

FIGURE 7. Trajectory of tree scores starting from a near-optimal guide tree (240,873 steps) for the data set DS-14K (13,921 taxa). Subproblems of Rec-I-DCM3 analyzed using *xmult* with 3 replications ("xmult = repl 3"; instead of the default 5 to make the process faster; note that more exhaustive search commands for the subproblems produce similar results, with the only difference that each iteration takes longer) and maximum subproblem size equals to 25% the total number of taxa. Only 6 Rec-I-DCM3 curves shown here (each individual run indicated with a different type of line; note that other runs showed the same oscillating pattern). The TNT curves show the best and worst score trajectories (out of 10 independent runs starting from that same tree) using the same search strategy as in Figure 5.

This perturbated topology is almost always less optimal than the original one (see next section) but provides a new starting point for the global TBR swapping. Rec-I-DCM3, after some number of iterations, works as a sort of "ratchet" more than as a strict divide-and-conquer. The perturbation caused by selecting subsets of taxa is obviously a very strong one—possibly too strong, as suggested by the fact that starting from a good tree (240,873) quickly degrades it into trees 60 to 80 steps longer. Instead of gradually improving the tree (which TNT normally does through divide-and-conquer and TF), the oscillations in score of Rec-I-DCM3 get down to really low scores (e.g., MKL), in the case of DS-14K, with an extremely low frequency. This means that (on average) optimal or near-optimal trees will not be found by Rec-I-DCM3 in difficult data sets, unless it is run for very extended periods of time. Even when Rec-I-DCM3 had started from an exceptionally good tree, this never happened within 520 h (3 weeks).

The previous discussion also shows that the difference in performance between TNT and Rec-I-DCM3 increases as the tree scores are closer to minimum length. The trees Rec-I-DCM3 finds after 24 h are not especially good (for DS-14K, 50 to 60 steps above MKL), but Rec-I-DCM3 finds those trees relatively fast: only two times slower than TNT. After 48 h, TNT finds (on average) trees 28 steps above MKL, but Rec-I-DCM3 has a harder time getting down to this length, about 9 to 10 times slower than TNT. Finally, the Rec-I-DCM3 runs starting from trees 13 steps above MKL (i.e., 240,873 steps) never found a better tree in 520 h, which TNT does in an average time of 2.5 h—well over 200 times faster than Rec-I-DCM3.

Rec-I-DCM3 Cannot Efficiently Break Trees into Too Many Parts

The previous section shows that Rec-I-DCM3 cannot be strictly seen as a divide-and-conquer technique. Roshan et al. (2004) optimistically stated that, in the size of data sets that can be analyzed, "Rec-I-DCM3 bridges at least one order [of magnitude], but may be capable of more—biological data sets with 100,000 aligned sequences do not yet exist!" They recognize that, for doing so, it will be necessary to break the trees in more than just a few pieces. But Roshan himself (2004) had noted that decomposing into numerous small subproblems produced inferior results, which would decrease the performance for extremely large data sets. For the largest data sets he tested (DS-11K and DS-14K), Roshan (2004) obtained the best scores using a subproblem size of 25% (that is, "reduced" subproblems of about 2800 taxa).

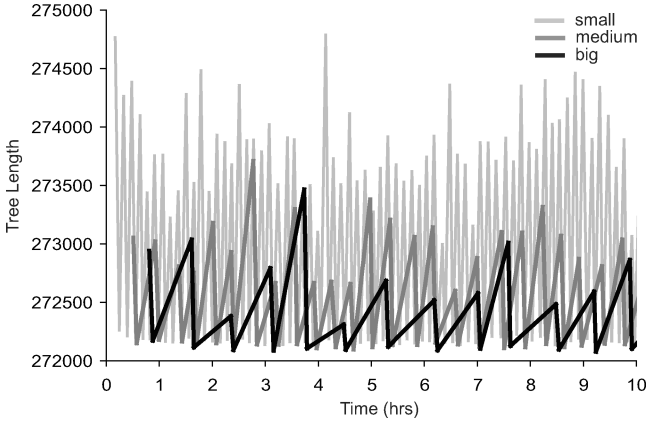

FIGURE 8. Trajectory of tree scores of Rec-I-DCM3 showing the tree length obtained after merging of the subproblems and after the global TBR for each of the iterations (data set DS-11K; 11,361 taxa). The three curves represent results using different subproblem sizes: maximum size 30% (black), 15% (dark gray), and 7.5% (light gray) of the total number of taxa. The peaks correspond to the tree length after recombination of the subproblems, and the valleys correspond to the tree length obtained after global TBR. Search parameters for the DCM3 subproblems as in Figure 3 (i.e., the original Rec-I-DCM3 scripts).

Examining the trajectories of tree scores obtained by Rec-I-DCM3 iterations before and after the global TBR branch swapping reveals the magnitude of this effect. This is exemplified using the DS-11K data set, performing Rec-I-DCM3 runs with subproblems of maximum size 30%, 15%, and 7.5% of the total number of taxa (Fig. 8). In each curve, the peaks (poor scores) correspond to the score obtained after subproblems are solved and recombined in the supertree, and the valleys (good scores) represent the score obtained after performing the postrecombination global TBR. This figure shows that tree lengths obtained after recombination (i.e., the peaks) are notably high, much higher for the smaller subproblems. When subproblems are relatively large (e.g., 30% the total number of taxa), half the iterations obtain a recombined tree at least 500 steps longer than the guide tree upon which the iteration had started. For smaller subproblems (15% the total number of taxa), in most iterations this difference grows to 1000 or more steps (and the smallest step increase is of approximately 450 steps). In the curve for the smallest subproblems (7.5% the total number of taxa), most of the iterations produce recombined trees about 1600 steps longer than the preceding tree (and the smallest step increases are of about 800 steps).

The global TBR phase drastically improves the poor scores obtained after recombination, reaching relatively good scores (i.e., valleys) at the end of each Rec-I-DCM3 iteration. However, as the subproblem size decreases, relative to the total number of taxa in the data set, the peaks of these Rec-I-DCM3 curves get higher and the global TBR tends to obtain longer trees. In other words, when subproblems are relatively large (30%), global TBR can recover from the perturbation caused by DCM3 decomposition and recombination, but for smaller subproblems global TBR has a much harder time recovering from the severely suboptimal solution produced by DCM3 decomposition and recombination.

The solution produced by smaller subproblems is more suboptimal, for two connected reasons. One reason is that reducing subproblem size increases the chances of different subproblems having conflicting resolutions (which create polytomies when the subproblems are recombined). Note, however, that the polytomies can only be produced by what (given the subsets of terminals for the subproblems) appear to be improvements, because TF at the end of the analysis of each subproblem guarantees that the solution returned by TNT will be (for the taxon subset) not worse than the original one. However, the problem with recombination is more than just the polytomies caused by conflicting resolution of subproblems, as Roshan (personal communication), when developing Rec-I-DCM3, had attempted to simply do TBRswapping using the polytomized tree as constraint tree, but this produced poor results. Recombination, therefore, not only creates polytomies but also some actual resolution that is farther away from optimal. Therefore, the ultimate reason for the poor performance of small subproblems is that they increase the chances of an improved solution for the subproblem actually worsening the global score (given that the subproblem is less representative of the whole data set). This is in fact one of the possible explanations Roshan (2004) offered (with a different wording) for the inferior performance of smaller subproblems. This problem is clearly reflected in the cumulative Rec-I-DCM3 curves (Fig. 9); as the subproblems represent a smaller percentage of the total number of taxa, the performance of this method decreases. It must be noted that this problem is most evident in very large data sets and the difference between dividing in 1/2, 1/4, or 1/8 may be less dramatic in matrices with fewer taxa (in that case TBR is capable of better recovering from the disastrous effect of subproblem recombination). But as

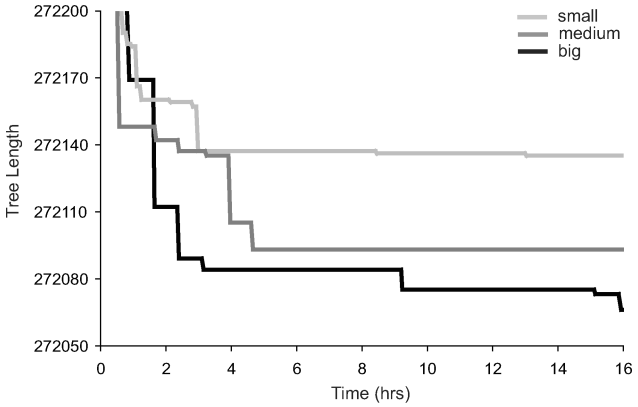

FIGURE 9. Cumulative scores for the Rec-I-DCM3 runs presented in Figure 8 for the data set DS-11K using three different subproblems size. The general effect of dividing in numerous smaller subproblems is obtaining worse overall scores.

far as large data sets are concerned, this poses a serious problem for Rec-I-DCM3 analyses. In contrast, and by virtue of using exact length calculations when analyzing sectors of the tree, SS poses no such limit on the number of subproblems in which a tree can be broken.

In sum, the global TBR round is the phase of Rec-I-DCM3 in which the actual score improvement is made. Since this phase is simply performing TBR branch swapping on the entire tree, the performance of Rec-I-DCM3 is bound by the efficiency of TBR implementation for large trees. This means that (unless one is willing to accept strongly suboptimal results) Rec-I-DCM3 can be applied to very large data sets only with a TBR branch swapper as fast as the one in TNT. Roshan et al. (2004) write as if their method, by virtue of breaking down the computational complexity, could be used with PAUP* just as well as with TNT, which is plainly not true for these large data sets: using PAUP* to perform global TBR, not a single iteration of Rec-I-DCM3 could be completed in 24 h.

CONCLUSIONS

Our experiments show that (contra Roshan et al., 2004) Rec-I-DCM3 does not provide any speedup relative to the SS algorithms described by Goloboff (1999) and implemented in TNT. Furthermore, for parsimony analysis of the extremely large data sets tested here, Rec-I-DCM3$_{(xmult)}$ is well below the performance of SS$_{(xmult)}$. This results from the exact tree length calculations possible for SS subproblems, suggesting that Rec-I-DCM3$_{(X)}$ is very likely to produce poorer results than SS$_{(X)}$ for any search routine X.

Our experiments also show that (after some number of cycles) Rec-I-DCM3 behaves more like a type of ratchet than as a true divide-and-conquer strategy (with the decomposition in subproblems acting as a perturbation more than as a "conquest"). Its oscillating behavior causes optimal or near optimal scores to be found very infrequently for difficult and large data sets (in comparison with tree search strategies implemented in TNT, which allow steady and gradual improvements). Furthermore, the analysis of different subproblem sizes shows that Rec-I-DCM3 has serious limitations when the data sets are divided in many subproblems (which will be needed for analyzing extremely large data sets).

On the positive side, Rec-I-DCM3 has two main advantages. First, the method is fairly simple and it does not require much programming effort. Second, selecting subsets of taxa can also provide an interesting perturbation to escape from local optima. If properly tuned and combined with other algorithms, this procedure may provide a useful approach under some circumstances. For example, within the driven search strategy of TNT, instead of periodically beginning new random addition sequences from scratch, perhaps a single round of Rec-I-DCM3 (or a similar method) could quickly provide a sufficiently different tree that is not as far from optimal as a Wagner tree. Rec-I-DCM3 also has the advantage that it is very general and can (in principle) be used to run analyses under criteria other than parsimony without extensive modifications. For criteria like maximum likelihood (e.g., Du et al., 2005b) or for the analysis of unaligned sequences (e.g., Wheeler, 2005), where heuristic searches are computationally more expensive than in parsimony, and no implementation of sectorial searches exists, Rec-I-DCM3 may provide a real improvement.

ACKNOWLEDGMENTS

For encouragement, discussion, and enlightenment over the years, PAG thanks James Carpenter, Kevin Nixon, and Steve Farris. The cluster used to make our runs was funded by NSF (AToL grant 0228699 to W. Wheeler). DP acknowledges funding from the Mathematical Biosciences Institute supported by NSF under Agreeement No. 112050. We also thank Bernard Moret for discussions on the subject and Usman Roshan for discussion and useful comments on an early version of the manuscript (despite his continued disagreement with some of our conclusions). We thank editor Rod Page, associate editor Olivier Gascuel, and several anonymous reviewers for their helpful comments and advice on different versions of the manuscript. This paper was presented (by PAG) at the symposium on "The problems of phylogenetic analysis of large data sets" (December 12, 2005) held in the Mathematical Biosciences Institute (Ohio State University); we thank many of the participants for discussion and input.

REFERENCES

Chase, M., et al. 1993. Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcL. Ann. Mo. Bot. Gard. 80:528–580.

Cracraft, J., and M. J. Donoghue. 2003. Assembling the Tree of Life. Oxford University Press, Oxford, UK.

Dotsenko, Y., C. Coarfa, L. Nakhleh, J. MellorCrummey, and U. W. Roshan. 2006. PRec-I-DCM3: A parallel framework for fast and accurate large-scale phylogeny reconstruction. Int. J. Bioinformatics Res. App. 2:407–419.

Du, Z., F. Lin, and U. W. Roshan. 2005a. Reconstruction of large phylogenetic trees: A parallel approach. Comput. Biol. Chem. 29:273–280.

Du, Z., A. Stamatakis, F. Lin, U. W. Roshan, and L. Nakhleh. 2005b. Parallel divide-and-conquer phylogeny reconstruction by maximum likelihood. Proceedings of the 2005 International Conference on High Performance Computing and Communications HPCC05. Lecture Notes in Computer Science 3726:776–785.

Farris, J. S. 1970. Methods for computing Wagner Trees. Syst. Zool. 19:83–92.

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

Giribet, G. 2005. A review of "TNT: Tree Analysis Using New Technology." Syst Biol. 54:176–178.

Goloboff, P. A. 1999. Analyzing large data sets in reasonable times: Solutions for composite optima. Cladistics 15:415–428.

Goloboff, P. 2000. Techniques for analysis of large data sets. Pages 70–79 *in* Techniques in molecular systematics and evolution (R. De Salle, W. Wheeler, and G. Giribet, eds.). Birkhauser Verlag, Basel.

Goloboff, P. A., J. S. Farris, and K. C. Nixon. 2003. T.N.T.: Tree analysis using new technology. Version 1.0. Program and documentation, available at http://www.zmuc.dk/public/Phylogeny/TNT.

Hibbett, D. S., R. H. Nilsson, M. Snyder, M. Fonseca, J. Constanzo, and M. Shonfeld. 2005. Automated phylogenetic taxonomy: An example in Homobasidiomycetes (mushroom forming fungi). Syst. Biol. 54:660–668.

Hovenkamp, P. 2004. Review of T.N.T.—Tree Analysis Using New Technology, Version 1.0, by P. Goloboff, J. S. Farris, and K. Nixon. Cladistics 20:378.

Huson, D., S. Nettles, and T. Warnow. 1999a. Diskcovering, a fast converging method for phylogenetic tree reconstruction. J. Comput. Biol. 6:369–386.

Huson, D., L. Vawter, and T. Warnow. 1999b. Solving large scale phylogenetic problems using DCM2. Proceedings 7th International

Conference on Intelligent Systems for Molecular Biology (ISMB '99):118–129.

Källersjö, M., J. S. Farris, M. W. Chase, B. Bremer, M. F. Fay, C. J. Humphries, G. Petersen, O. Seberg, and K. Bremer. 1998. Simultaneous parsimony jackknife analysis of 2538 rbcL DNA sequences reveals support for major clades of green plants, land plants, seed plants and flowering plants. Plant Syst. Evol. 213:259–287.

McMahon, M., and M. Sanderson. 2006. Phylogenetic supermatrix analysis of GenBank sequences from 2228 papilionoid legumes. Syst. Biol. 55: 818–836.

Meier, R., and F. B. Ali. 2005. The newest kid on the parsimony block: TNT (Tree analysis using new technology). Syst. Entomol. 30:179.

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

Roshan, U. W. 2004. Algorithmic techniques for improving the speed and accuracy of phylogenetic methods. Ph.D. dissertation, University of Texas, 229 pp.

Roshan, U. W., B. M. E. Moret, T. L. Williams, and T. Warnow. 2004. Rec-I-DCM3: A fast algorithmic technique for reconstructing large phylogenetic trees. Proceedings 3rd IEEE Computational Systems Bioinformatics Conference (CSB 2004):98–109.

Smith, M. L., and T. L. Williams. 2006. Phylospaces: Reconstructing evolutionary trees in tuple space. Fifth IEEE International Workshop on High Performance Computational Biology (HiCOMB '06): 18.

Soltis, D. E., P. S. Soltis, M. E. Mort, M. W. Chase, V. Savolainen, S. B. Hoot, and C. M. Morton. 1998. Inferring complex phylogenies using parsimony: An empirical approach using three large DNA data sets for angiosperms. Syst. Biol. 47:324.

Swofford, D. 1998. PAUP*: Phylogenetic analysis using parsimony (*and other methods). Version 4.0. Sinauer Associates, Sunderland, Massachusetts.

Tehler, A., D. P. Little, and J. S. Farris. 2003. The full-length phylogenetic tree from 1551 ribosomal sequences of chitinous fungi, Fungi*. Mycol. Res. 107:901–916.

Warnow, T. 2005. Disk covering methods: improving the accuracy and speed of large scale phylogenetic analyses. Chapter 21, pp. 1–24 *in* Handbook of computational biology (S. Aluru, ed.). Chapmann & Hall, CRC Computer and Information Science Series.

Williams, T. L., D. A. Bader, M. Yan, and B. M. E. Moret. 2006. High-performance phylogeny reconstruction under maximum parsimony. Pages 369–394 *in* Parallel computing for bioinformatics and computational biology: Models, enabling technologies and case studies (A.Y. Zomaya, ed.). John Wiley & Sons, New York.

Williams, T. L., and M. L. Smith. 2005. Cooperative Rec-I-DCM3: A population based approach for reconstructing phylogenies. 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB '05):18.

Wheeler, W. C. 2005. Alignment, dynamic homology, and optimization. Pages 71–80 *in* Parsimony, phylogeny, and genomics (V. A. Albert ed.). Oxford University Press, Oxford, UK.

First submitted 15 November 2006; reviews returned 23 January 2007; final acceptance 9 March 2007

Associate Editor: Rod Page