WILEY-BLACKWELL

Cladistics (2014) 1–16

Cladistics

10.1111/cla.12082

Computer science and parsimony: a reappraisal, with discussion of methods for poorly structured datasets

Pablo A. Goloboff*

CONICET, INSUE, Fundación Miguel Lillo, 4000, S.M. de Tucumán, Argentina

Accepted 14 April 2014

Abstract

In recent years, several publications in computer science journals have proposed new heuristic methods for parsimony analysis. This contribution discusses those papers, including methods highly praised by their authors, such as Hydra, Sampars and GA + PR + LS. Trees of comparable or better scores can be obtained using the program TNT, but from one to three orders of magnitude faster. In some cases, the search methods are very similar to others long in use in phylogenetics, but the enormous speed differences seem to correspond more to poor implementations than to actual differences in the methods themselves.

Introduction

In recent years there have been a number of papers published in journals devoted to informatics or computer science (CS) that have studied and proposed heuristic methods for searching phylogenetic trees under parsimony. These papers have a common denominator in that, for the most part, they ignore almost completely the contributions by biologists to parsimony analysis. Conversely, the claims and methods in such papers have not been scrutinized by the biological community. Based on citation numbers from actual biologists, the impact those papers have had on phylogenetic practice effectively equates to zero. This is in contrast to well established methods and software used by biologists, such as Hennig86 (Farris, 1988), PAUP* Swofford, 2001), Nona (Goloboff, 1993), or TNT (tree analysis using new technology; Goloboff et al., 2003, 2008), which benefited from years of testing by a significant community of users and to some extent compete with each other (thus being mutually forced to improve). Despite this state of affairs, those unfamiliar with the field of study could easily form the mistaken impression, on reading some of these CS papers of the past decade, that the authors are the first scientists to ever approach the problem of how to find the most parsimonious trees. This does not mean, of course, that the contributions are irrelevant, but the assessment of their relevance is generally hindered by the lack of explicit references to known standards developed by the biological community and by the lack of available software. In some cases, the very datasets used for testing the methods proposed are not made available, thus making the assessment of their relevance even more difficult.

It is probable that the foregoing combination of difficulties have discouraged biologists from examining the methods and claims in such papers. In this contribution, the main results of the papers concerned are discussed, showing that the optimism and conclusions of their authors are often based on arguments and comparisons that are faulty, misleading, or irreproducible – sometimes all three simultaneously.

A number of papers (Goëffon et al., 2006, 2007; Richer et al., 2009, 2012) are exceptions to the majority of CS papers by using publicly available datasets and computer codes, thus making possible more explicit comparisons with other methods (including

*Corresponding author:
E-mail address: pablogolo@yahoo.com.ar

TNT, considered by biologists as the best program for parsimony analysis; Hovenkamp, 2004; Giribet, 2005; Meier and Ali, 2005). They concluded that Sampars (Richer et al., 2012) is superior to Hydra (Goëffon et al., 2007; Richer et al., 2009), LVB (Barker, 2004) and GA + PR + LS (Ribeiro and Vianna, 2009); in their 2007 paper, Goëffon et al. concluded that Hydra outperforms TNT. In the CS literature, therefore, the current state of affairs is that Sampars is the best search strategy ever published for parsimony. One of the co-authors (R. Tello) has posted up lecture slides (http://www.tamps.cinvestav.mx/~ertello/bioinfo/sesion13.pdf) stating very matter-of-factly (on slide 59) that Sampars is the best method for finding most parsimonious trees (but cites only other CS methods as competitors).

Hydra and Sampars indeed appear to outperform other CS methods, but it is shown below that when using the very same methods recommended by Goloboff (1999), TNT can reach (for difficult medium-sized datasets) better parsimony scores, in much shorter times. The same is true of other methods (cyclic perturbation schemes and tree-hybridization) implemented in TNT since 2005.

Methods and background

Test datasets

The set of matrices presented by Andreatta and Ribeiro (2002) and Ribeiro and Vianna (2005) constitutes the test bed used by many of the papers discussed (it is found at http://www2.ic.uff.br/~celso/grupo/instances.zip). From that set of matrices, all the random instances (matrices 1–20) and the real instances, called GOLO (actually a dataset from Zander, 1995), ROPA and SCHU, were reanalyzed here. In addition, the Zilla dataset of Chase et al. (1993) and Roshan et al.'s (2004) dataset with 13 921 bacterial sequences are used for some comparisons (both datasets are available at http://cs.njit.edu/usman/software/recidcm3/public_datasets.tar.gz).

Computers

Most of the testing was carried out on a 32-bit Windows operating platform running on an Intel I5 650 processor at 3.2 GHz. (with 4 GB of RAM, released in early 2010). The only routines that were tested on other machines (a variety of machines of about the same speed, bought in 2011, forming a small Linux cluster, but with each routine run on a single processor) were the extensive ratchet and drifting and the ***pfijo*** routine [XMULT and PF columns in Tables 1–4)].

Background on tree-score calculations

Given the NP-completeness of phylogenetic problems, one of the crucial aspects of any method for searching phylogenetic trees is its speed. Therefore, with a few exceptions – such as Oblong (Goloboff, 2014), designed to privilege memory usage – most of the contributions to parsimony analysis have been aimed at improving speed. The improvements may be both in the form of algorithms for faster score calculations during a search, or in the form of better ways to select the trees to be evaluated (so as to increase the chances of looking at fewer or better trees). As several aspects of tree-score calculations will be discussed in subsequent sections, a brief review of the basic ideas to speed up tree evaluations during searches is given here.

The length of any tree can be calculated from scratch with algorithms such as those described by Farris (1970) or Fitch (1971). However, those algorithms require visiting every internal node, and thus the time needed to evaluate a tree produced by branch-swapping will increase with number of taxa (keep in mind that also the number of SPR or TBR rearrangements increases with number of taxa).

Given that they are so critical, methods to evaluate the trees during searches with a speed that does not strongly depend on the number of taxa were developed relatively early in computational phylogenetics. The first published description of a method for speeding up tree-length calculations during searches was by Goloboff (1994), but the speed of Hennig86 (Farris, 1988) and PAUP (Swofford, 2001) made it evident that they used similar methods. Although neither Farris nor Swofford had published a description, Goloboff (1994, p. 434) noted that a similar logic had been used by Farris (1970) to calculate the length increment of adding a terminal to a growing Wagner tree. Also, Ganapathy et al. (2003) mention that Swofford (1986) described similar methods in his thesis, which is not available online. Additional improvements and extensions to the basic method were published by Goloboff (1996, 1998, 1999), and implemented in TNT (Goloboff et al., 2003, 2008).

Thus, among widely used parsimony programs, the four fastest (Hennig86, PAUP*, Nona, and TNT) all use essentially the same method to speed up searches. Notably, the widely used Phylip (Felsenstein, 2005) does not; it uses instead a full down-pass optimization to calculate tree lengths during branch swapping, which explains the speed difference with the other four programs. The method used in Hennig86, PAUP*, Nona and TNT is based on indirect calculations of tree length, which takes into account that branch swapping consists of successive modifications

to an original tree. Thus, the lengths and character-state reconstructions at each stage can be derived from those in previous stages, instead of calculated from scratch (a technique known as "incremental computing" in CS). Branch swapping consists of clipping the tree in two, creating two subtrees, and then re-joining the two subtrees at different branches. The length increment that will result from inserting one of the subtrees at a specific branch of the other subtree can be calculated by comparing the most parsimonious state-sets for the nodes delimiting the two corresponding branches. In this way, the total length of the new (complete) tree can be calculated at a speed that does not (strongly) depend on the number of taxa – the calculation requires only one comparison between two (pairs of) nodes. Therefore, the time needed for the evaluation of each rearrangement of a tree with t taxa and c characters, approaches $O(c)$ instead of the $O(tc)$ required by a full down-pass.

Evidently, in a full-fledged implementation such as TNT, there are many other steps that need to be carefully considered, including bounding methods (based on unions; Goloboff, 1996, 1999; note that a similar approach was used in POY by Varón and Wheeler (2013), who did not cite Goloboff in that regard). All such other shortcuts, however, only become relevant after the indirect calculation of tree lengths has been implemented – the indirect length calculation is thus the first and most important step towards speeding up searches. Analogous methods are also used in the fastest programs for maximum likelihood (RAxML, Stamatakis et al., 2005; PhyML, Guindon and Gascuel, 2003).

Reinventing wheels

One of the aspects in which computer scientists show their disdain for the work of biologists is in re-describing previously published methods. An example is Andreatta and Ribeiro's (2002) "GRASP" algorithm; it stands for "Greedy Randomized Adaptive Search Procedure", which is just a Wagner tree with a closest-addition sequence. This was first proposed by Farris (1970) and has been available for many years in PAUP* and PAUP. Since the method described by Goloboff (1994) is essential to speeding up tree-length calculations, it is not all that suprising that it has been published again, in different journals, four separate times.

Yan and Bader (2003) published what they contend is different to Goloboff (1994, 1996), that is, in using down-pass optimizations only, but Goloboff (1996) had been clear that the indirect length calculation requires using the states for a new root located between ancestor and descendant branch of the target tree: it was already well known that, for finding final state-sets, using a down-pass with tree-rerooting is equivalent to doing an up-pass. For example, Swofford and Maddison (1987) had used this property to find final states for internal nodes using tree-rerooting and only down-pass algorithms. Thus, Yan and Bader's method, although presented as different, is basically the same as that described by Goloboff (1994); the only difference is that they find the root states with down-passes only, instead of using down- and up-passes. It is therefore unsurprising that, as stated by Yan and Bader themselves (1994, p. 9), their "new" method has the same time-dependencies as Goloboff's (1994), with time of evaluation $O(c)$ instead of $O(tc)$. Using only preliminary states for internal nodes, as in Yan and Bader (2003), is a disadvantage when zero-length branches are to be collapsed, but their program does not do so. According to fig. 4 of Bader et al. (2006), the branch-and-bound program that uses the "new" method of optimization (Exact-MP) is slower than the exact solution of PAUP* on a single processor, which in turn is slower than TNT's [White and Holland (2011) have compared PAUP* with TNT]. The exact solution of TNT, however, is not especially efficient, as pointed out in the documentation of the program (challenging datasets today are beyond the possibilities of exact solutions, so I have chosen to invest most of my effort into improving heuristic methods). For a faster exact search, see White and Holland's (2011) XMP, although that program has the drawback that it does not eliminate zero-length branches, does not allow for the use of constraints, cannot save suboptimal trees and only processes DNA data with Fitch optimization under equal weights.

Ganapathy et al. (2003) described the same method as Goloboff (1994), but their method was also based on down-passes: they admitted in the paper (as requested by an anonymous reviewer, pers. commun. from the reviewer) that their method for calculating length is "similar to other techniques described in other papers; … what is new is the mathematical analysis" (p. 245–246). They were much more cryptic than Yan and Bader (2003) in the similarities of their method with that of Goloboff (1994).

Keith et al. (2005) published the same method. That is, for the fourth time. In addition to the indirect calculation of tree lengths during branch swapping, their program uses a search method with similarities to simulated annealing and tree-drifting (Goloboff, 1999). It is thus unsurprising that (for Zilla) their program has a speed comparable to that of PAUP* using the ratchet (Nixon, 1999). They (and the reviewers) have missed the obvious implication that PAUP* uses a similar method to calculate tree-lengths during the search, and explicitly stated that "to the best of [their]

knowledge, this is the first time that such an algorithm has been described" (p. 463).

Finally, Ribeiro and Vianna (2005) also used indirect calculation of tree lengths for branch swapping; they did not describe the details, simply stating that (when inserting one subtree into the other) they derive the length increment from the ancestral states of both subtrees. Their only comment on this method ["the investigation of each neighborhood SPR can be implemented in time $O(mn^2)$, which is one order of magnitude faster than the implementation originally proposed in Andreatta and Ribeiro (2002)", p. 330] implied that it had never been used before. This 2005 paper is based on Vianna (2004), where he was more explicit than in the subsequent paper: he incorrectly stated that the algorithm of Goloboff (1994, 1996) evaluates each rearrangement during swapping with times that depend on the number of taxa – that is, ***O(tc)*** instead of ***O(c)***. In Vianna's account, this is because cutting a branch requires ***O(tc)*** to find final states, and reinserting it at another location requires ***O(c)*** to calculate the length increment; the total time for evaluating a rearrangement would then be ***O(tc)*** (Vianna, 2004; p. 26). Two facts, however, are missing from the picture presented by Vianna (2004). First, that clipping using the incremental optimization of Goloboff (1996) can be done much faster than ***O(tc)***. Second, that a search program does not evaluate a single rearrangement; it tries instead all the possible reinsertions for each clipping, rendering the relative time needed for the clipping negligible and making the total time needed to evaluate each rearrangement independent of the number of taxa [which has been perfectly clear for other biologists, such as Ronquist (1998) and Felsenstein (2004), and to other computer scientists, such as Yan and Bader (2003)]: the multiplicity of reinsertions per clipping was explicitly stated by Goloboff (1994, p. 435). Perhaps Vianna's (2004) worst omission on this issue, however, is that [although he used some shortcuts in the vein of "shortcut A" of Goloboff (1996) to lessen the work needed] his own method [very similar to Goloboff's (1994, 1996), but with some adaptations to binary characters, the only ones his program can handle] also requires finding final states when clipping the tree, so that any difference in dependence is purely illusory.

Variable neighbourhoods

One of the reasons computer scientists often put forward for not discussing the work of biologists in detail is that the properties of tree-search algorithms should be studied independently of specific implementations. This was clearly stated by Andreatta and Ribeiro (2002), in one of the earliest contributions of this kind:

> Luckow and Pimentel (1985) have conducted the first evaluation study of algorithms for the phylogeny problem, in which they tested strategies offered by some software packages. Another study was later conducted by Platnick (1987), involving different programs. These studies do not provide a good basis for assessing algorithm efficiency and their results are not conclusive, as far as the algorithms themselves are not precisely described in the references, and completely different techniques and languages have been used for their implementation. (p. 434)

Blazewicz et al. (2010) made a very similar point:

> In order to eliminate the influence of a style of implementation of algorithms on efficiency of applications, a general framework is designed. This work provides an architectural basis for comparison. (p. 77)

On first impression it sounds a reasonable idea, but carries with it the problem that a given implementation can easily turn the conclusions established on the basis of another implementation upside down.

A case where this can happen is in determining the best "neighbourhood" to use during a local search. Biologists are well aware of the differences between SPR and TBR, and that TBR is more thorough and in many cases more time consuming. Probably most of them are also familiar with the idea that applying a different swapping algorithm at different stages of the search may increase the efficiency. For example, Nona started each replication with SPR saving a single tree, and only when it was not possible to find better trees using SPR, did it switch to the more time-consuming TBR. The idea was not to spend so much effort using a thorough algorithm such as TBR if the tree was still sufficiently suboptimal for a more superficial algorithm to find better solutions easily. My recollection (20 years later) is that I implemented the alternating swappers of Nona after being inspired by some feature of PAUP or Hennig86, although I cannot find today the reference (either in the manuals, or programs). In any event, switching from one swapper to another seemed unworthy of a name or much discussion, just a matter of obvious common sense given the implementation then used. Another example of switching between swapping algorithms as the search proceeds is in POY (Varón et al., 2010; for Version 4 and earlier versions as well).

A number of CS papers (e.g. Andreatta and Ribeiro, 2002; Ribeiro and Vianna, 2005; Goëffon et al., 2007; Blazewicz et al., 2010) made a point of using "variable" or "progressive" neighbourhoods. However, there is a very important qualification: which swapping algorithm provides the best trade-off between results and computing time depends on the implementation. If the length of each rearrangement is calculated by using the full down-pass of Fitch's (1971) algorithm,

then the speed with which NNI, SPR or TBR evaluate rearrangements is identical. When each rearrangement, however, is evaluated instead with an indirect length calculation, then the speed with which each rearrangement can be evaluated increases as we move from NNI to SPR, and from SPR to TBR. As discussed by Goloboff (1996, p. 202), this is because the work (=time) needed to recalculate the final states when the tree is divided in two subtrees leads to evaluating more reinsertion points under TBR than under SPR. In fact, in TNT, TBR uses bounding algorithms not used in SPR, with the result that (for large empirical datasets) TBR not only evaluates individual rearrangements faster – *it also needs a shorter absolute time than SPR* to complete a cycle of rearrangements, or to move from a Wagner tree (Farris, 1970) to a locally optimal tree. Of course, the SPR swapper in TNT could be modified so that it uses those shortcuts now implemented only in TBR, but this seems unnecessary, because: (i) the TBR algorithm is already fast enough that it can be applied to many thousands of taxa, and (ii) in those cases where analysis of large datasets becomes impossibly slow due to lack of structure, the shortcut will be equally unproductive for SPR, so that the implementation without using the shortcut should suffice. Thus, although in Nona (where TBR was only slightly more efficient than SPR) it was advantageous to start each replication with SPR, the situation is different in TNT (where TBR is much more efficient than SPR). For that reason, in most searching contexts of TNT, the use of SPR in the initial stages of the search was eliminated (in May 2005, as pointed out in the program documentation and its web site). TNT does not implement NNI, but PAUP* does and, predictably, each rearrangement is evaluated with NNI over 200 times slower than with TBR (for Zilla). In TNT, the speed difference between TBR and SPR increases with number of taxa and dataset structure; for the 13 921-taxon dataset of Roshan et al. (2004), TBR can evaluate rearrangements about 70 times faster than SPR.

In connection with the differences in time needed to evaluate rearrangements by NNI, SPR and TBR, Goëffon et al. (2007, p. 3) stated that:

> the variation cost induced by a NNI transformation can be easily calculated... On the contrary, a large neighborhood relation like TBR requires more computational effort. Exploring all the neighbors of a configuration using TBR is computationally expensive. Indeed, the neighboring trees are subject to important topological modifications. Thus, less information can be conserved for the calculation of the parsimony score of a neighbor tree.

The statement is incorrect: TBR derives more rearrangements per clipping, but because all of them can be traced easily to a single clip, every one can be evaluated with less work than in SPR, and much less work than in NNI.

Although several authors have examined variable neighbourhoods, in most cases the only changes in neighbourhood are switches between NNI and SPR [or versions of SPR where the clipped subtree is reinserted at a given maximum distance from the original position, as also done in RAxML (Stamatakis et al., 2005)]. The only paper testing variable neighbourhoods that has examined TBR is that of Blazewicz et al. (2010); they used a full down-pass to evaluate scores and explicitly pointed out (p. 87–89) that TBR was so slow as to be simply prohibitive. Andreatta and Ribeiro (2002), Blazewicz et al. (2010) and Ribeiro and Vianna (2005) used neighbourhoods that switch from NNI to SPR as the search proceeds and the preceding swapping method fails to improve results. The latter authors also used what they called "2-SPR" as the final neighbourhood, where two branches of the tree are cut and then reinserted (as done in the "mswap" command of Nona). Goëffon et al. (2007), however, used a decreasing neighbourhood instead – when SPR finds a better tree, then the search continues with a less exhaustive algorithm (either NNI or SPR reinserting at a smaller distance from original position). The rationale could be that there is no point in using an "exhaustive" algorithm such as the full SPR when more superficial algorithms could perhaps lead to better trees now that some change has been done to the tree. Vazquez-Ortiz and Rodriguez-Tello (2011) explored both increasing and decreasing neighbourhoods.

Of all the papers concerned, only Ribeiro and Vianna (2005) were explicit in their use of indirect calculation of tree lengths (and their implementation seems rather inefficient, see below). Thus, the conclusions of all such papers about the relative effectiveness of using NNI, SPR, or TBR, at different stages of the search must be taken with a grain of salt. As careful implementations become progressively faster in evaluating individual rearrangements with neighbourhoods of larger size, there is less and less to be gained by applying an algorithm that examines fewer rearrangements at lower speeds, such as NNI, or even SPR.

The papers by Viana et al. (2007, 2009) and Lin et al. (2007) are interesting in that they used a special form of tree representation, such that the neighbourhoods of a local search can be defined in terms of rearrangements to the tree notation itself. Thus, the rearrangements are not defined in terms of clips and reinsertions of subtrees, which has the unfortunate consequence that (despite the hopes of Lin et al. (2007, p. 1916) in this regard) the indirect length calculation is neither applied nor, apparently, applicable. Thus, even if the types of rearrangements used in these papers eventually prove advantageous in some way, they are doomed to be laden with length evaluations

hundreds or thousands of times slower than the neighbourhoods of standard methods.

In practice, the notion that tree-search algorithms should be studied without regard to a specific implementation only serves as an excuse for the authors not to make embarrassing comparisons between other programs and their own. This is hardly in the interest of actual users of phylogenetic methods.

Comparisons with TNT

There are some CS papers that do make comparisons with the heuristic methods of TNT, but none of these comparisons is very informative. A common approach is to state that the "default" algorithms of TNT were used. Roshan et al. (2004) did this, as discussed by Goloboff and Pol (2007), as did the authors of Hydra and Sampars (Goëffon, 2006; Goëffon et al., 2006; Richer et al., 2009; Vazquez-Ortiz, 2011). Richer et al. (2009) stated:

> TNT ... is known to be highly optimized to be able to evaluate millions of trees per second. TNT integrates a large number of search strategies such as tree drifting, parsimony ratchet, sectorial search and more others. The reference software TNT was used with its default parameters specified in the documentation.

Thus, the authors give the impression that they have compared their algorithm with (among others) the ratchet and tree-drifting. The TNT runs were done by Goëffon (2006) in his thesis, where he gave additional details:

> The time allotted for Hydra is of 300 s (5 min) on the real instances, and 1000 s on the random instances, more difficult. Tree analysis using new technology does not require a specification of duration and it finishes by itself; executed on an AMD Athlon 64 X2 3800, its execution times do not exceed 5 s. (p. 116, translated from French)

The crucial point not mentioned anywhere in those papers is that the default of the ***xmult*** command in TNT does not include drifting or ratchet – only sectorial search and tree-fusing. Goloboff (1999), however, was explicit that

> ... both [sectorial search] and [tree-fusing] require some structure in the data—which is normally the case for real datasets.... For small datasets (i.e., below 100 taxa) neither [sectorial search] nor [tree-fusing] are usually of much help. Small datasets are difficult to analyze only when very poorly structured. That situation is best analyzed by doing multiple RAS + TBR followed by extensive [tree-drifting]. (p. 425)

Goloboff (1999) also stated that the ratchet was as appropriate as tree-drifting for unstructured and random datasets and that his implementation of the ratchet could be improved (speeding it up to work at the same speed as tree-drifting), which I subsequently did when TNT was released in 2003. A similar recommendation was made by Goloboff (2002, p. 77–78):

> Datasets with fewer than 100 taxa will be difficult to analyze only when extremely incongruent. In those cases, the methods of tree-fusing and sectorial searches perform more poorly... Therefore, smaller datasets are best analyzed by means of extensive ratchet and/or tree-drifting, reducing tree-fusing and sectorial searches to a minimum.

Thus, all the papers that examined the defaults of TNT on random datasets have used exactly the algorithms and options Goloboff (1999, 2002) advised not to use.

There is, however, still another detail to be considered: Richer et al. (2009) concluded that Hydra is superior to TNT despite the fact that it finds trees that are only slightly better but using a much longer time [they gave no detailed timings, but the thesis of Goëffon (2006) stated that they used 1000 s for the random datasets, which TNT runs in a second or less]. The alleged reason for the superiority of Hydra is that it reaches the same or similar scores by evaluating *fewer* rearrangements. The reasoning goes that if Hydra was made to evaluate its rearrangements at the same speed of TNT, then it would be clearly superior. Lin et al. (2007) and Lin (2008) presented a similar argument in defence of their own tabu-search methods (which they compared only with PAUP* and Phylip). That, however, is impossible because the speed of rearrangement methods necessarily decreases as fewer rearrangements are derived per clipping. This applies not only to NNI, but also to randomly chosen SPR or TBR moves, as undertaken in some programs; selecting a clipping and a single reinsertion point at random leads to much slower tree evaluations than when all reinsertion points are subsequently tried for each clipping. Thus, the fact that Hydra evaluates fewer rearrangements to achieve similar scores is of secondary interest at best. In an actual phylogenetic analysis, the user's primary concern is not in how many *rearrangements* were made but the *time* needed to finish.

A contribution with some interesting ideas is that of Gregor et al. (2013), who proposed a method for approximating most parsimonious trees without searching, but instead detecting patterns in the data. They compared their PTree program with PAUP* and TNT, but in the case of TNT, *only against SPR*. Their reasons for comparing against SPR, instead of TBR or the more elaborate search strategies implemented in TNT, are not stated anywhere in the paper. Gregor et al. (2013) found that "compared to PAUP* with the SPR or TBR heuristic and TNT with the SPR heuristic, PTree performs worse in all tests in terms of the parsimony costs...." (p. 12). For the two largest datasets tested by Gregor et al. (up to 8000 taxa), the

difference with SPR in TNT (the runs with PAUP* could not be finished for those datasets) was over 1500 steps in favour of TNT. Given that the scores of PTree were vastly inferior, it is difficult to make meaningful speed comparisons, despite which they nonetheless reported that "while PTree was faster than TNT SPR with the HIV dataset with more than 4000 sequences, PTree was slower than TNT SPR with the RAxML dataset" (p. 12). But the TNT times they reported are way too long for a single Wagner tree plus SPR. For 8000 taxa and 1600 bp they reported that TNT needed over 33 h, when building a Wagner tree plus SPR on a single tree for that size of dataset takes only between 5 and 20 min (as opposed to the 23 h of PTree). The only way in which the TNT runs could have used so much time is by using multiple replications and/or saving multiple trees per replication. As the TNT scores were probably better than those of PTree within the first few replications (as suggested by the huge score differences reported in their tables 1 and 3), then the comparison was shifted in favour of PTree by making TNT run for hours and hours – just by adding more replications. Another striking detail of Gregor et al. (2013) is that the "average" tree lengths reported in their tables 1 and 3 (purportedly from 10 independent runs of PTree) contained only integer numbers – no decimals (the text said nothing about rounding to integers, which would be highly unusual anyway). The datasets used by Gregor et al. (2013) were subsets of taxa from other datasets, and they cannot be reconstructed from their description of the experiments, so that their results cannot be checked directly.

Carroll et al. (2007, 2009) released a phylogeny package they call PSODA. They did not propose any new methods for tree searching, simply presenting a new implementation of stepwise addition of trees, branch-swapping, and ratchet, which they compared only with Phylip and PAUP*. Although they mentioned the existence of TNT, they provided no comparisons. In a sequel to those papers, Sundberg et al. (2012) proposed a new method, "partial tree mixing" or PTM, based on decomposing the dataset in smaller subsets, analyzing, and then combining the results. They stated that

> ... the performance of PTM was compared to PAUP* using stepwise addition and TBR. TNT and DCM are newer programs which implement a wide variety of heuristic methods. Partial Tree Mixing was implemented in the open source phylogenetics program PSODA. These methods were tested on datasets ranging from 218 to 8780 taxa. PTM was compared against stepwise maximum parsimony where both were followed by a TBR based search until a minima was found. As the step which combines the two final partial trees is equivalent to a standard TBR search, the PTM algorithm was further refined using the Parsimony Ratchet and a sectorial search. (p. 12)

They did not state how many trees were saved in TNT or PAUP* – obviously a large number, given the run times reported. Again, their reasons for improving the results of their own method with ratchet and sectorial search, but without giving TNT such a benefit, were unspecified. They succintly stated (in the caption of their table 2) that "TNT finishes much faster than PTM, but finds less parsimonious trees". For the Zilla dataset, PTM found a tree of the known minimum length (16 218), and TNT found (in a single run) a tree one step longer. This, however, was a single run of TNT, which took 7 s to complete: in contrast, PTM took 2.5 h. If they had tried other random seeds for TNT, one can conjecture that they would have found a most parsimonious tree in longer times – say, 30 s. Even more interesting is the difference in lengths between PTM and TNT in other datasets, for it reveals how carelessly they have made the comparisons. For the dataset "RDPII", PAUP* produced trees 50 steps longer than the best known trees (33 515 steps), while the score reported by TNT was 8651 steps longer. For dataset "U", PAUP* produced trees 911 steps longer than the best known trees (92 195 steps), while TNT reported trees 109 064 steps longer. The only reasonable explanation for such huge length differences is that the "U" and "RDPII" datasets had their gaps coded as a fifth state in TNT (which is the default for DNA sequences), instead of a missing entry as in the other programs. Evidently, Sundberg et al. (2012) neither checked the documentation of TNT for the default treatment of gaps, nor made sure that the datasets had been read under the same conditions by the different programs. No additional checking of their method is possible, because (despite the paper stating that the method has been implemented in that package) current versions of PSODA do not include any options for calculating PTM, and (except for Zilla) the datasets they used are not publicly available.

In the void

The papers in the preceding section include some comparisons with TNT and PAUP*, even if incomplete. In the papers discussed in this section, the only comparisons are with other CS papers. As no program is publicly available for any of the methods discussed in this section, the only possible comparisons are from the timings given by the authors using their own implementation. This is in fact how these CS authors have compared their results against those of others, that is, by reference to the timings and results given in the original papers; none of the comparisons cited here involves new runs of previous programs.

The paper of Ribeiro and Vianna (2009) naturally supersedes Andreatta and Ribeiro (2002), Vianna

(2004) and Ribeiro and Vianna (2005). They summarized their results by means of plots of "time-to-target". These plots show the probability of finding a given target length as the search proceeds in time. For the matrix SCHU (113 taxa), their fig. 5 shows that the probability of finding trees of 760 steps (shortest trees are 759) after a 200 s run with their best method (GA + PR + LS) was about 0.55; having a probability close to unity of finding 760 steps took more than 350 s. Comparable calculations can be made with TNT; with "***xmult = hit 300 giveup 760 noupdate***" the program will hit length 760 independently 300 times (note that these are the defaults of TNT, except for giving up each cycle as soon as the length 760 is found). Dividing the total time needed for the 300 hits to 760 steps (25.57 s) by 300, it can be concluded that TNT needs to run for 0.085 s to have $P = 1.0$ of finding trees of 760 steps. This is 4100 times faster than GA + PR + LS. Ribeiro and Vianna (2009) did not provide time-to-target plots for the length 759 (shortest known trees for that dataset), but TNT needs 0.132 s to have $P = 1.0$ of finding 759 steps – 1500 times shorter than what GA + PR + LS needs to have a $P = 0.55$ of finding trees 1 step longer. The programs used by Ribeiro and Vianna (2009) are completely inadequate for use in actual phylogenetic work. These timings are for GA + PR + LS, which is their best method; the time-to-target plots for their earlier GRASP/VND (Ribeiro and Vianna, 2005; figs 11 and 12) were much worse, over 25 000 times slower than TNT (although some of that difference may be explained by a computer slower than current ones). Some years before Ribeiro and Vianna (2005) used an average of 32 836 s to find the best length (496) for the GOLO dataset (see their table 2), Goloboff (1999) had already reported (p. 425) that trees of 496 steps had been found by TNT 63 times in a total time of 4.31 min (i.e. every 4 s), running on slower machines.

Viana et al. (2007) compared their results only to Vianna (2004). Their method produced results that were far from optimal in many cases, using very long runtimes. The scores reported in a subsequent paper (Viana et al., 2009) were somewhat better, but the runtimes were even worse than in the 2007 paper. For example, for their instance 14 (a "perfect" phylogeny of 35 taxa) their heuristic method took 8 s – which is about 1000 times slower than TNT's exact solution for such a homoplasy-free dataset. Table 2 of Viana et al. (2007) shows that for ROPA and SCHU, the times used (respectively) to reach solutions 5 and 1 steps above minimum known length were 533 and 2832 s: TNT needs, to find better or equal average lengths, average times well below 0.1 s. Thus, Viana et al.'s (2009) program is at least 5000–28 000 times slower than TNT.

Blazewicz et al. (2010) compared their results with those of Ribeiro and Vianna (2005), Viana et al. (2009) and Goëffon et al. (2007). Their runs using NNI did not take much time, but the resulting scores were too far from optimal to be seriously considered; their TBR implementation also found very poor scores, by virtue of being too slow to reasonably explore tree space (Blazewicz et al., 2010; p. 87–89). For the methods based on VNS (variable neighbourhood search with some NNIs and then SPR) the actual times are as long as for the papers just discussed (see their table 11, which summarizes results of 10 runs or more). For the ROPA dataset, the average length found by Blazewicz et al. is 327.27 steps or more, and average time is 107 s or more. For TNT, the command "***xmult = repl 2***" was repeated 300 times with different random seeds (only two replications, instead of the default five, are needed to match or outperform Blazewicz et al.). On this dataset, this takes an average time of 0.031 s, and the average length is 326.96 – that is, shorter trees than Blazewicz et al.'s, but 3400 times faster. For SCHU, Blazewicz et al. (2010) reported an average of 760.7 steps for their best routine, and average times between 1152 and 6047 s. TNT (same commands as for ROPA) takes an average of 0.088 s to run, and the average length is 759.97. Thus, for SCHUH, TNT finds a better average length, but over 10 000 times faster. This huge speed difference is even more significant given that the timings reported by Blazewicz et al. (2010) correspond to runs using 16 processors, instead of just *one* as in the TNT runs.

Hydra, Sampars, and undescribed TNT methods

Richer et al. (2012) described a simulated annealing method, Sampars, based on previous work by Vazquez-Ortiz (2011) and Vazquez-Ortiz and Rodriguez-Tello (2011). This series of papers started out with a humble attitude:

> ...the aim of this work is not the development of a new software able to compete with the best softwares like TNT because they make an intensive use of several heuristics and strategies. Our aim here is to present a new neighborhood which... could be used to improve the efficiency of the existing softwares. (Goëffon et al., 2007, p. 7)

Gradually, however, they became more boastful, and in their most recent paper (Richer et al., 2012), they considered Hydra (Goëffon et al., 2007; Richer et al., 2009) to be well established as the "state-of-the-art", because of the previous demonstration that Hydra examines fewer rearrangements than TNT to achieve comparable lengths. They compared Sampars to Hydra, LVB (Barker, 2004), and GA + PR + LS

(Ribeiro and Vianna, 2009), and found Sampars to be the best of the four methods – and thus, in their view, the best method ever proposed for parsimony. For their comparisons they used a Xeon X5650 (a processor released the same year as the one used for most of the comparisons here, with only minor differences in speed).

Datasets used by Richer et al. 2012

The most significant differences between Sampars and the other programs are in the random datasets. These are 20 datasets with 45–75 taxa, 61–159 binary characters, and significant numbers of missing entries, created by Ribeiro and Vianna (2005), and then considered as a standard by Goëffon (2006), Goëffon et al. (2006, 2007), Viana et al. (2007), Ribeiro and Vianna (2009), Vazquez-Ortiz (2011), Vazquez-Ortiz and Rodriguez-Tello (2011) and Richer et al. (2012). Most of these papers used them without comment, but Goëffon (2007) justified the use of such datasets on the grounds that

> ...the real instances of reasonable size available in the literature (including Zilla) do not offer any real challenge... These datasets are well structured... However, random instances with missing data are difficult; with them, the robustness of the best software is undermined. (p. 117–118, translated from French)

Note that because it is only in the random datasets that these methods outperform the plain ***xmult*** command of TNT, if one were to blindly follow the recommendation of the authors and routinely use their methods, then in the case of real-life datasets one would be adopting inferior methods because these only do well in random datasets (which of course no biological dataset resembles). There is, however, another twist, in the fact that on such random datasets no group is well supported. A plethora of phylogenetic literature exists that is concerned with eliminating weakly supported groups from the final conclusions: several papers (e.g. Farris et al., 1996; Källersjö et al., 1999; Goloboff and Farris, 2001) proposed that well supported groups can be found without much effort and without the need to actually find the trees of minimum length (as rediscovered a few years later by computer scientists: Williams et al., 2004). If groups of negligible support can be safely ignored, the reasonable course of action for those random datasets created by Ribeiro and Vianna (2005) would be "do not even bother finding shortest trees" – showing that

Table 1
Average scores reached by different methods for the 20 random datasets. The data for Hydra and Sampars were taken from the literature (from Goëffon (2006), who did 20-min runs, and from Richer et al. (2012), who give averages for 30 independent runs, respectively). The rest were done with TNT, with 30 independent runs of each routine

	Average score							
Dataset	Hydra	SAMP	RAS + H	XMUL	PF	RB + H × 3	RB + H × 6	Worst RB + H × 6
1	545.40	545.13	545.33	**545.00**	**545.00**	545.13	**545.00**	**545**
2	1356.10	1355.30	**1355.17**	**1354.84**	**1354.47**	1355.50	**1354.80**	1356
3	833.90	833.43	833.57	**833.22**	**833.28**	**833.37**	**833.07**	834
4	589.40	588.23	588.76	**588.09**	**588.06**	**587.97**	**587.77**	589
5	789.00	789.00	789.03	789.00	789.00	789.00	789.00	789
6	597.30	596.57	597.20	**596.03**	**596.09**	596.83	596.63	598
7	1270.70	1270.83	**1270.77**	**1270.62**	**1269.62**	**1269.40**	**1269.07**	**1270**
8	854.10	853.33	**853.17**	854.12	853.34	**853.00**	**852.97**	**853**
9	1145.20	1144.73	**1144.37**	**1143.25**	**1143.78**	**1144.07**	**1143.37**	1145
10	721.30	720.80	**720.67**	**720.00**	**720.00**	**720.03**	**720.00**	**720**
11	542.60	542.21	542.53	**541.28**	**542.00**	**541.87**	**541.60**	543
12	1213.60	1215.27	1215.07	1215.53	**1212.84**	**1211.65**	**1210.21**	**1214**
13	1518.50	1517.77	**1516.50**	**1516.78**	**1516.62**	**1515.47**	**1515.17**	1518
14	1161.90	1163.03	1163.87	**1161.47**	**1161.62**	**1162.28**	**1161.58**	**1163**
15	754.50	753.90	**753.77**	**753.16**	**752.62**	**752.77**	**752.45**	754
16	530.80	531.00	532.40	531.75	531.19	**529.73**	**529.10**	532
17	2455.20	2456.00	**2454.33**	**2452.97**	**2451.16**	**2450.83**	**2450.42**	**2453**
18	1523.70	1525.67	**1521.23**	**1522.09**	**1521.06**	**1521.08**	**1521.00**	**1521**
19	1016.90	1016.23	**1015.20**	**1015.19**	**1014.81**	**1013.00**	**1012.5**	**1015**
20	663.90	662.82	665.20	664.00	664.41	**662.06**	**661.73**	664

SAMP, Sampars; RAS + H, 100 random addition sequences plus TBR (with mulpars off), retaining the final point of each addition sequence, followed by six cycles of selecting best 40 trees and 500 rounds of hybridization; XMUL, ***xmult*** command, with 500 iterations of tree-drifting and 500 of ratchet (the rest of parameters as defaults), running for 8 min; PF, a single random addition sequence plus TBR, followed by the ***pfijo*** command (with options ***nums 2500 chunksize 20***); RB + H, rebuildit + hybrid script, three to six cycles (see text for details). The last column (Worst RB + H × 6) indicates the worst score obtained in the 30 runs. Bold values indicate those that outperform Sampars, except for the last column, where bold indicates that the worst TNT run produced better results than the average for Sampars.

Table 2
Difference in average score, for each of the 20 datasets, relative to Sampars. All methods run as indicated for Table 1

	Difference with Sampars							
Dataset	Hydra	RAS + H	XMUL	PF	RB + H × 3	RB + H × 6	Worst RB + H × 6	SD Sampars
1	+0.27	+0.20	−0.13	−0.13	0	−0.13	−0.13	0.43
2	+0.80	−0.13	−0.46	−0.83	+0.20	−0.50	+0.70	0.97
3	+0.47	+0.14	−0.21	−0.15	−0.06	−0.36	+0.57	0.56
4	+1.17	+0.53	−0.14	−0.17	−0.26	−0.46	+0.77	0.80
5	0	+0.03	0	0	0	0	0	0.00
6	+0.73	+0.63	−0.54	−0.48	+0.26	+0.06	+1.43	0.56
7	−0.13	−0.06	−0.21	−1.21	−1.43	**−1.76**	−0.83	1.63
8	+0.77	−0.16	−0.79	+0.01	−0.33	−0.36	−0.33	1.27
9	+0.47	−0.36	**−1.43**	−0.95	−0.66	**−1.36**	+0.27	1.09
10	+0.50	−0.13	**−0.80**	**−0.80**	**−0.77**	**−0.80**	−0.80	0.70
11	+0.39	+0.32	**−0.93**	−0.21	−0.34	−0.61	+0.79	0.72
12	−1.67	−0.20	+0.26	−2.43	**−3.62**	**−5.06**	−1.27	2.76
13	+0.73	−1.27	−0.99	−1.15	**−2.30**	**−2.60**	+0.23	1.91
14	−1.13	+0.84	−1.56	−1.41	−0.75	−1.45	−0.03	1.82
15	+0.60	−0.13	−0.74	**−1.28**	**−1.13**	**−1.45**	+0.10	1.11
16	−0.20	+1.40	+0.75	+0.19	**−1.27**	**−1.90**	+1.00	1.23
17	−0.80	−1.67	**−3.03**	**−4.84**	**−5.17**	**−5.58**	−3.00	2.63
18	−1.97	**−4.44**	−3.58	**−4.61**	**−4.59**	**−4.67**	−4.67	3.96
19	+0.67	−1.03	−1.04	−1.42	**−3.23**	**−3.73**	−1.23	2.14
20	+1.08	+2.38	+1.18	+1.59	−0.76	−1.09	+1.18	1.44
∑	**+2.75**	**−3.11**	**−14.39**	**−20.28**	**−26.21**	**−33.81**	**−5.25**	

A negative difference indicates that (for that dataset) the method in question ouperforms Sampars. The last row indicates the sum of differences. Bold values indicate those cases where the magnitude of the difference in favour of TNT is larger than the standard deviation for Sampars scores (reported by Richer et al. (2012), included in the last column for comparison).

those datasets do not strongly support any conclusion is very easy and does not require time-consuming analyses. Thus, methods to actually find the most parsimonious trees in random datasets are much less important than methods to improve the efficiency on empirical well-structured datasets. Random datasets pose peculiar problems for algorithms based on local searches, especially when missing entries are abundant: the differences in tree scores among all possible trees are minimal, and radically different trees can easily have minimum differences in score. Search algorithms therefore have to be adapted to these peculiarities. This is a natural consequence of the "NFL theorem" of Wolpert and Macready (1997), which is well known in the field of computer science: some tree-search algorithms will be more suited to some problems than to others (and no search algorithm will be best for all possible classes of problems).

As discussed above, the peculiarities of unstructured datasets had been recognized by Goloboff (1999, 2002), who pointed out that small but difficult datasets were best analysed by means of numerous starting points, each followed by extensive ratchet and drifting. Ratchet and tree-drifting are based on cyclic perturbations. After its first non-beta release in late 2003, TNT incorporated in 2005 two additional methods that can be used to produce cyclic perturbations (see the list of BugFixes, which is distributed with the TNT package, also available at http://www.zmuc.dk/public/phylogeny/TNT/BugFixes.htm).

The first of those two methods is the command ***pfijo*** (introduced in TNT in May 2005), which borrows some ideas from Roshan et al.'s (2004) rec-i-dcm3. As discussed by Goloboff and Pol (2007), rec-i-dcm3 (although originally presented as a "divide-and-conquer" technique) is best seen as a cyclic perturbation scheme, just like the ratchet. High search efforts for the reduced datasets are therefore counterproductive – the analysis of the reduced datasets best serves to perturbate and provide a new starting point for searching, not to "conquer by division". Thus, the ***pfijo*** command is a sort of ratchet, where the perturbation is given by creating reduced datasets from sectors of the tree (the name ***pfijo*** comes from the spanish *piñón fijo* – fixed-wheel or fixed-gear – as opposed to the ratchet or free-wheel). For the reduced datasets, instead of replacing the HTU's by their own down-pass states (which in a true sectorial search produces exact score evaluations), the states of one of the descendants of the HTU will be used with a probability that depends on a constant K (which changes the parsimony landscape). The higher the value of K, the higher the probability that the basal HTU states will be chosen, so that lower values of K perturbate more strongly (by selecting representatives further away from the root states for each sector; in rec-i-dcm3 only terminals were selected). After searching for each "perturbated" reduced dataset (with only four rounds of RAS + TBR, and tree-fusing with original resolution), then full TBR and a few cycles of tree-drifting from

Table 3
Average times (in seconds) for 30 runs (Sampars times taken from Richer et al., 2009). All methods run as indicated for Table 1

Dataset	SAMP	RAS + H	XMUL	PF	RB + H × 3	RB + H × 6
1	1407.6	16.5	480.0	37.0	58.5	114.1
2	1938.2	37.1	480.0	97.3	91.8	183.0
3	2506.3	28.1	480.0	69.0	78.9	157.5
4	1341.1	21.8	480.0	50.4	69.0	137.9
5	2007.9	24.7	480.0	65.9	72.2	145.4
6	1164.3	24.5	480.0	52.5	72.5	147.3
7	4063.8	43.5	480.0	105.8	116.8	236.2
8	2884.7	39.2	480.0	81.7	101.5	205.6
9	3237.5	38.1	480.0	93.3	104.7	207.7
10	2288.0	32.2	480.0	72.1	100.1	201.8
11	3807.8	29.8	480.0	50.9	88.7	176.4
12	3668.4	65.2	480.0	125.9	137.2	270.6
13	2514.2	52.4	480.0	134.1	121.9	240.1
14	2847.1	53.6	480.0	113.1	127.2	249.3
15	4808.6	42.1	480.0	82.3	110.4	222.2
16	3268.2	36.8	480.0	61.1	92.1	179.5
17	8020.2	98.6	480.0	238.1	193.4	381.2
18	4451.4	64.7	480.0	156.0	153.2	307.6
19	6875.3	57.9	480.0	107.7	133.6	266.3
20	7149.4	44.5	480.0	76.3	109.0	215.7
Speedratio	×1	×82.9	×7.3	×39.7	×31.8	×16.0

Note that the XMUL routine was run with a timeout of 8 min (the times are included to allow comparison with the other routines). The last row indicates the average speed ratio relative to Sampars (calculated for each dataset as T_{SAMP}/T_M and then averaged across datasets for each method M).

the resulting tree are performed. The process is repeated a certain number of times (optionally, the resulting trees can be subject to tree-fusing (Goloboff, 1999) every certain number of cycles). The advantage over other search algorithms is that it can move to trees that are rather different from the starting one, yet near-optimal, in relatively little time. Thus, it is as appropriate as the ratchet for random datasets.

The other TNT command that allows cyclic perturbations is the command ***hybrid*** (introduced in TNT in December 2005). This command is primarily intended for hybridizing trees that are too different for tree-fusing to be useful (see below), but also makes it possible to reinsert in an incomplete tree the missing taxa by using a random addition sequence. Then, with the scripting language of TNT, it is easy to repeatedly create random groups of taxa to be pruned and reinserted in the tree; the following example will prune off and reinsert on tree number 0 random sets of 20 taxa, 100 times:

```
loop 1 100          [routine 1]
   rseed*;
   agroup = 0*20<00;
   pruntaxa 0/{0};
   hybrid * 0;
   stop
```

The first line of the loop makes sure that the random seed is changed every time (so that a different set of taxa is selected every iteration); the second line selects the set of taxa (making sure that the outgroup, taxon 0 by default, is not included); the third line removes the taxa selected from tree 0; the fourth and last line of the loop reinserts the taxa and swaps (with TBR) the tree. This is another sort of ratchet, where the cyclic perturbation is given by removing and reinserting subsets of taxa.

The ***hybrid*** command, however, is intended primarily for hybridizing trees. Much of the success of Goloboff's (1999) strategies was in the use of tree-fusing, which is not very useful for random datasets: tree-fusing only exchanges subgroups with identical taxon composition in both trees, but two independently found near-optimal trees for a random dataset are unlikely to share many groups (the same is true when some wildcard taxa can float around very different positions in the tree). The ***hybrid*** command will take two input trees and identify two partitions in each tree. These partitions will have as similar a taxon composition as possible, and be as evenly sized as possible (i.e. they will preferably have about half the taxa). The taxa not shared by the subtrees will be removed, the two halves of the trees exchanged, and then the trees will be completed (as explained above) and swapped with TBR. Assume there are two trees $\boldsymbol{A}$ and $\boldsymbol{B}$, each with two parts: parts $\boldsymbol{A_1}$ and $\boldsymbol{B_1}$ have a similar but not identical taxon composition, and so do parts $\boldsymbol{A_2}$ and $\boldsymbol{B_2}$. The hybrid command will use the two trees $\boldsymbol{A_1A_2}$ and $\boldsymbol{B_1B_2}$ to create two new trees, $\boldsymbol{A_1B_2}$, and $\boldsymbol{A_2B_1}$. In some cases, this will lead to improvements. This differs from most other attempts at using genetic algorithms

Table 4
Best scores found across all the runs for each method. All methods run as indicated for Table 1

Dataset	Hydra	SAMP	RAS + H	XMUL	PF	RB + H × 3	RB + H × 6
1	545	545	545	545	545	545	545
2	1354	1354	1354	1354	1354	1354	1354
3	833	833	833	833	833	833	833
4	588	587	588	587	588	587	587
5	789	789	789	789	789	789	789
6	596	596	596	596	596	596	596
7	1269	1269	1269	1269	1269	1269	1269
8	852	852	852	852	852	852	852
9	1144	1141	1143	1141	1142	1143	1141
10	721	720	720	720	720	720	720
11	542	541	541	**540**	**540**	**540**	**540**
12	1211	1208	1211	1209	1209	1208	1208
13	1515	1515	1515	1515	1515	1515	1515
14	1160	1160	1161	1160	1160	1160	1160
15	752	752	752	752	752	752	752
16	529	529	530	529	529	**527**	**527**
17	2453	2450	2454	2450	2450	2450	2450
18	1522	1521	1521	1521	1521	1521	1521
19	1013	1012	1012	1013	1012	1012	1012
20	661	**659**	662	661	662	660	660

Bold values indicate those cases where an individual method produced scores better than the others.

in that it exchanges evenly sized partitions. Other methods simply select some subgroup of a tree at random, and randomly place it in some part of the other tree, which is very unlikely to lead to improvements; Ribeiro and Vianna (2009) also attempted to exchange even-sized partitions first; Goëffon et al. (2006) and Richer et al. (2009) used a hybridization that tries to preserve relative distances between taxa, but the exchange of even-sized partitions takes care of that problem more naturally. The hybrid method is not as effective as tree-fusing on well-structured datasets, because tree-fusing attempts many exchanges and can evaluate them quickly using incremental down-pass optimization – so as to effect only those exchanges that improve the tree. The ***hybrid*** command, however, still provides a way to exchange significant portions of the trees in the case of unstructured datasets, in combination with scripts (to facilitate repeated exchanges of tree parts). For example, the following TNT script will effect 100 exchanges between randomly chosen pairs of trees:

```
loop 1 100     [routine 2]
   rseed*;
   tgroup = 0*2;
   hybrid{0};
   stop
```

As shown in the following sections, when these routines are used on the random datasets, the results significantly outperform those of Hydra or Sampars.

Hybridization

This applies cycles of the hybridization (similar to routine 2) to initial populations of trees obtained by RAS + TBR (100 replications, passing onto the hybridizer the best 40 trees available). Each cycle of hybridization consists of 500 exchanges between two randomly chosen trees, and six such cycles (each starting from the best 40 trees available) were repeated. The results are shown in Tables 1–4 (column RAS + H). As can be seen in Table 2, the average results are only 0.16 steps shorter (3.11/20) than those for Sampars [but note that Richer et al. (2012) concluded the superiority of Sampars over Hydra because of an even smaller difference, only 0.14 steps]. In 10 datasets, the average length found by pure hybridization is better than the average of Sampars; on the other 10 it is worse. Thus, the results in terms of score are very similar for both methods, but the difference in time is extremely significant: tree hybridization with TNT takes 80 times less than reported by Richer et al. (2012) for Sampars. If the tree-hybridization is replaced with tree-fusing, the results are [as anticipated by Goloboff (1999)] very poor, with no cycle of fusing producing any significant improvement to the original set of trees. Thus, the tree hybridization implemented in TNT in 2005, although so far undescribed in the literature, is an important tool to apply to random or very poorly structured datasets.

Cyclic perturbations

The specific recommendation Goloboff (1999, 2002) made for this type of dataset is using extensive ratchet

and drifting, easily done with the ***xmult*** command. Thus, the first method based on cyclic perturbations to be tested was 500 iterations of tree drifting and 500 iterations of ratchet per each starting point (initial Wagner tree plus TBR), repeating this until a total of 8 min of search are completed. This was accomplished with the commands "***timeout 8:00; hold 1000; xmult = rep 1000 drift 500 rat 500;***". The 8 min allowed TNT to finish between about 100 ± 50 replications, depending on the dataset. The results (shown in Tables 1–4) are, in terms of score, clearly better than those based on tree hybridization, producing an average score that is 0.72 steps shorter than the average score of Sampars. The difference, however, is concentrated on the larger datasets (11–20): the average diference for datasets 1–10 is 0.47 in favour of ***xmult***, while the average difference with Sampars for datasets 11–20 is 0.97 steps in favour of ***xmult***. For 15 of the 20 datasets, ***xmult*** produces better average results than Sampars, and for the other five, a worse average. While the results are superior to those of Sampars, these searches proceed (on average) seven times faster.

The other cyclic perturbation strategy tested on the 20 random datasets is the ***pfijo*** command, as discussed above. The options used were "***pfijo = nums 2500 chunksize 20;***", starting from a tree created by random stepwise addition (***mult = rep 1 hold 1;***). Note that ***pfijo*** saves the intermediate trees and thus requires that maxtrees be set to more than 2500 (e.g. ***hold 2600***). This produced better results than ratchet and drifting, with the average results for 16 datasets outperforming results of Sampars. The average score difference with Sampars is 1.01 steps in favour of ***pfijo***, but for datasets 1–10 the difference is 0.47 steps, as opposed to 1.56 steps in favour of ***pfijo*** for datasets 11–20. In addition to producing better results, this routine runs almost 40 times faster than Sampars.

Combining cyclic perturbations with hybridization

The best results for the random datasets were obtained by using the ***hybrid*** command to produce a cyclic perturbation with deletion and reinsertion of some taxa, but saving the trees at each cycle and subjecting them to hybridization (RB + H columns in Tables 1–4). This was implemented by means of a script similar to the two routines shown above, except that:

1 the results for each cycle of perturbation were saved to memory, to be used for the subsequent hybridization;

2 after a certain number of rounds (initially, 1000, then every 250) of reinsertions of randomly pruned taxa producing trees exceeding best trees by five steps, the tree was returned to the last best tree (to avoid drifting too far away from the best tree found);

3 the number of taxa pruned was a random number between 5 and 25 instead of a fixed number;

4 global cycles were repeated, each with 4000 rounds of reinsertions first, followed by 500 rounds of hybridization on the best 33% of the trees found so far. The best overall tree is used as starting point for the subsequent global cycle.

As this was implemented in scripts (rebuildit, globhyb, and multirebuild), it is easy for the user to experiment with additional ideas or fine tuning of the options – the scripts can be changed with any text-editor and run with TNT. Although the run times are somewhat longer than for the previous routines, the results in terms of score for this combined routine are clearly the best. The average scores after three global cycles are already significantly better than for Sampars, with 16 datasets having a better average, two ties, and two worse cases; for eight of the datasets, the difference with Sampars (see Table 2) is larger than the standard deviation of Sampars (reported by Richer et al., 2012; included here in Table 2 for comparison). After six global cycles, the combined routine produces better average scores for 18 datasets, with one tie and a single case (dataset 6) with a worse average (with a minimal difference, however; only 0.06 steps beyond the average of Sampars). In 10 of the datasets, the difference with Sampars (Table 2) is larger than the standard deviation of Sampars, and in nine of the datasets, the *worst* out of 30 runs of the combined routine produced tree scores below the *average* of Sampars, indicating the magnitude of the difference in effectiveness. Completing three cycles of partial building plus hybridization takes 30 times shorter than Sampars, and completing six, about 16 times shorter. This difference in time is well beyond what might be expected from possible minor differences in speed of the computers used.

Conclusions

Although most of the CS papers discussed in this contribution were presented as important advances in parsimony analysis, the methods they proposed are clearly outperformed by routines that were already available in TNT. Some of the methods discussed here had not previously been described in any publication, but they were implemented and documented in TNT during 2005, so they were within the reach of the authors of most of the papers cited here.

The contributions by the Richer laboratory and associates (Goëffon, 2006; Goëffon et al., 2006, 2007; Richer et al., 2009, 2012; Vazquez-Ortiz, 2011; Vazquez-Ortiz and Rodriguez-Tello, 2011) are perhaps the most significant in that they have eventually led to the development of programs (Hydra and

Sampars) that produce results of some quality on random datasets. However, better results can be found with appropriate TNT options, from 30 to 40 times faster.

The laboratories of Viana, Ribeiro and Blazewicz, on the other hand, have proposed (Andreatta and Ribeiro, 2002; Vianna, 2004; Ribeiro and Vianna, 2005, 2009; Viana et al., 2007, 2009; Blazewicz et al., 2010) a series of methods that find trees of acceptable quality in the case of empirical datasets, and strongly suboptimal trees for the random datasets. None of their programs is publicly available, but as the runtimes the authors report are well over 1000 times longer than the runtimes needed by TNT to find comparable scores, the lack of availability of those programs does not seem to represent a serious loss for biologists.

Of course, this is not to say that TNT could not be outperformed by any possible program. Indeed, for exact searches, White and Holland's (2011) program XMP does – and hopefully, for heuristic routines applicable to large and difficult datasets, other programs will in the future. Demonstrating that a new program convincingly outperforms some of the well-established ones, however, will require more than unverifiable claims and a couple casual comparisons. At the very least, the flexibility of a program such as TNT should be used to make it run with options that are most appropriate for the dataset at hand – instead of selecting the least appropriate, as the majority of the papers discussed here seem to do. Alachiotis and Stamatakis (2011) found similar problems with Kasap and Benkrid's (2010) earlier claim of acceleration of parsimony calculations by hundreds or thousands of times: the virtues of their FPGA implementation had been grossly overestimated due to improper comparisons – Kasap and Benkrid compared their method against a full optimization with PAUP*, but tree searches do not use full optimizations and other programs are faster than PAUP*. Block and Maruyama (2013) did compare FPGA against TNT, and they concluded that their FPGA implementation (massively parallel, but using a full down-pass per rearrangement) was slower than TNT.

All of the CS contributions discussed here that purported to present new and useful methods for parsimony analysis failed to provide adequate comparisons against the software and methods currently used by biologists. Some of the researchers involved in those papers are leading figures in CS, and it is obvious from their resumés and achievements outside their work on phylogenetics that they are highly qualified scientists (e.g. Celso Ribeiro, David Bader). Unfortunately the phylogeny programs they produced seem far below the standards now used in biology. I surmise that it is precisely that success in other aspects of their careers that prevented them from investigating the history and current state of the field, and from dedicating more time and energy to produce truly useful software and methods for phylogenetic analysis. Biologists, on the other hand, have been doing research on parsimony searches for several decades now (starting with Farris in the early 1970s), with a stamina and dedication that cannot be easily matched by occasional forays into the field. A similar situation occurs in model-based phylogenetic inference: all the programs that have made a difference to actual users are not those from occasional contributors suddenly struck by genius, but come instead from researchers that have dedicated their entire careers to phylogenetics (e.g. O. Gascuel, J. Huelsenbeck, F. Ronquist, A. Stamatakis, D. Swofford, D. Zwickl). This suggests the general rule that the achievements of researchers in phylogenetic methodology tend to be inversely proportional to their success in fields other than comparative biology.

Implementation

The scripts used for comparing against Hydra and Sampars are available at http://www.lillo.org.ar/phylogeny/published/scripts_CS_and_MP.zip. The methods in those scripts have also been implemented as native in the March 2014 version of TNT (in the ***rebuild*** and ***tfuse:hybrid*** commands). Timings in Table 3 were produced with the scripts, because that is how the methods were available to other researchers prior to 2014 (and it is thus the appropriate comparison against Hydra and Sampars). The new native implementation, however, is 25–30% faster than the scripts.

Acknowledgements

I wish to express my deep gratitude to a number of people who were kind enough to share ideas, discuss problems, and/or criticize earlier versions of the manuscript: Salvador Arias, James Carpenter, Santiago Catalano, Mark Simmons and Claudia Szumik, as well as an anonymous reviewer. Financial support from CONICET (PIP 112 201101 00687) is greatly appreciated.

References

Alachiotis, N., Stamatakis, A. 2011. FPGA acceleration of the phylogenetic parsimony kernel? Proceedings of FPL 2011, Conference on Field Programmable Logic and Applications, Crete, September, pp. 1–6.

Andreatta, A.A., Ribeiro, C.C., 2002. Heuristics for the phylogeny problem. J. Heuristics 8, 429–447.

Bader, D.A., Chandu, V., Yan, M. 2006. ExactMP: an efficient parallel exact solver for phylogenetic tree reconstruction using

maximum parsimony. Proceedings of the International Conference on Parallel Processing, 14–18 August, Columbus, OH, pp. 65–73.
Barker, D., 2004. LVB: parsimony and simulated annealing in the search for phylogenetic trees. Bioinformatics 20, 274–275.
Blazewicz, J., Formanowicz, P., Kedziora, P., Marciniak, P., Taron, P., 2010. Adaptive memory programming: local search parallel algorithms for phylogenetic tree construction. Ann. Oper. Res. 183, 75–94.
Block, H., Maruyama, T. 2013. A hardware acceleration of a phylogenetic tree reconstruction with maximum parsimony algorithm using FPGA. 2013 International Conference on Field-Programmable Technology, FPT 2013, Kyoto, Japan, 9–11 December, pp. 318–321.
Carroll, H., Ebbert, M., Clement, M., Snell, Q. 2007. PSODA: better tasting and less filling than PAUP. Proceedings of the 4th Biotechnology and Bioinformatics Symposium, 19 and 20 October, Colorado Springs, CO, pp. 74–78.
Carroll, H.D., Teichert, A.R., Krein, J.L., Sundberg, K., Snell, Q.O., Clement, M.J., 2009. An open source phylogenetic search and alignment package. Int. J. Bioinform. Res. Appl. 5, 349–364.
Chase, M.W., Soltis, D.E., Olmstead, R.G., Morgan, D., et al., 1993. Phylogenetics of seed plants: an analysis of nucleic sequences from the plastid gene rbcL. Ann. Mo. Bot. Gard. 80, 528–580.
Farris, J.S., 1970. Methods for computing Wagner trees. Syst. Zool. 34, 21–24.
Farris, J.S. 1988. Hennig86, Program and Documentation. Port Jefferson, New York.
Farris, J.S., Albert, V., Källersjö, M., Lipscomb, D., Kluge, A., 1996. Parsimony jackknifing outperforms neighbor-joining. Cladistics 12, 99–124.
Felsenstein, J. 2004. Inferring Phylogenies. Sinauer Associates, Sunderland, MA, pp. 580.
Felsenstein, J. 2005. PHYLIP (Phylogeny Inference Package) Version 3.6. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle, http://evolution.genetics.washington.edu/phylip.htm.
Fitch, W., 1971. Toward defining the course of evolution: minimal change for a specific tree topology. Syst. Zool. 20, 406–416.
Ganapathy, G., Ramachandran, V., Warnow, T. 2003. Better hill-climbing searches for parsimony. Proceedings of the Third International Workshop on Algorithms in Bioinformatics (WABI 2003), 15–20 September, Budapest, Hungary, pp. 245–258.
Giribet, G., 2005. A review of "TNT: tree analysis using new technology.". Syst. Biol. 54, 176–178.
Goëffon, A. 2006. Nouvelles heuristiques de voisinage et mémétiques pour le problème Maximum de Parcimonie. Thesis de Doctorat, École Doctorale d'Angers, pp. 1–56.
Goëffon, A., Richer, J.M., Hao, J.K., 2006. A distance-based information preservation tree crossover for the maximum parsimony problem. Lect. Notes Comp. Sci. 4193, 761–770.
Goëffon, A., Richer, J.M., Hao, J.K., 2007. Progressive tree neighborhood applied to the maximum parsimony problem. IEEE/ACM Trans. Comput. Biol. Bioinform. 5, 1–10.
Goloboff, P. 1993. Nona, a tree-searching program. Program and documentation. Available at: http://www.lillo.org.ar/phylogeny/Nona-PeeWee.
Goloboff, P., 1994. Character optimization and calculation of tree lengths. Cladistics 9, 433–436.
Goloboff, P., 1996. Methods for faster parsimony analysis. Cladistics 12, 199–220.
Goloboff, P., 1998. Tree searches under Sankoff parsimony. Cladistics 14, 229–237.
Goloboff, P., 1999. Analyzing large data sets in reasonable times: solutions for composite optima. Cladistics 15, 415–428.
Goloboff, P. 2002. Techniques for analyzing large data sets. In DeSalle, R., Giribet, G., Wheeler, W. (Eds), Methods and Tools in Biosciences and Medicine, Techniques in Molecular Systematics and Evolution. Birkhäuser Verlag, Basel, pp. 70–79.
Goloboff, P., 2014. Oblong, a program to analyse phylogenomic data sets with millions of characters, requiring negligible amounts of RAM. Cladistics, 30, 273–281.
Goloboff, P., Farris, J.S., 2001. Methods for quick consensus estimation. Cladistics 17, S26–S34.
Goloboff, P., Pol, D., 2007. On divide-and-conquer strategies for parsimony analysis of large data sets: Rec-I-DCM3 versus TNT. Syst. Biol. 56, 485–495.
Goloboff, P., Farris, J.S., Nixon, K.C. 2003. TNT: Tree Analysis using New Technology. Program and Documentation. http://www.zmuc.dk/public/phylogeny/TNT.
Goloboff, P., Farris, J.S., Nixon, K.C., 2008. TNT, a free program for phylogenetic analysis. Cladistics 24, 774–786.
Gregor, I., Steinbrück, L., McHardy, A., 2013. PTree: pattern-based, stochastic search for maximum parsimony phylogenies. PeerJ 1, e89. pp. 1–12.
Guindon, S., Gascuel, O., 2003. A simple, fast and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst. Biol. 52, 696–704.
Hovenkamp, P. 2004. Review of T.N.T. – tree analysis using new technology, Version 1.0, by P. Goloboff, J. S. Farris, K. Nixon. Cladistics 20, 378–383.
Källersjö, M., Albert, V., Farris, J.S., 1999. Homoplasy increases phylogenetic structure. Cladistics 15, 91–93.
Kasap, S., Benkrid, K., 2010. High performance phylogenetic analysis with maximum parsimony on reconfigurable hardware. IEEE Trans.Very Large Scale Integrat. (VLSI) Syst. 99, 1–13.
Keith, J., Adams, P., Ragan, M., Bryant, D., 2005. Sampling phylogenetic tree space with the generalized Gibbs sampler. Mol. Phylogenet. Evol. 34, 459–468.
Lin, Y.M. 2008. Tabu search and genetic algorithms for phylogeny inference. PhD thesis, North Carolina University, Faculty of Operations Research, Raleigh, NC, pp. 119.
Lin, Y.M., Fang, S.C., Thorne, J.L., 2007. Tabu search algorithm for maximum parsimony phylogeny inference. Eur. J. Oper. Res. 176, 1908–1917.
Luckow, M., Pimentel, R., 1985. An empirical comparison of numerical Wagner computer programs. Cladistics. 1, 47–66.
Meier, R., Ali, F.B., 2005. The newest kid on the parsimony block: TNT (tree analysis using new technology). Syst. Entomol. 30, 179–182.
Nixon, K., 1999. The parsimony ratchet, a new method for rapid parsimony analysis. Cladistics 15, 407–414.
Platnick, M., 1987. An empirical comparison of microcomputer parsimony programs. Cladistics. 3, 121–144.
Ribeiro, C.C., Vianna, D.S., 2005. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. Int. Trans. Oper. Res. 12, 325–338.
Ribeiro, C.C., Vianna, D.S., 2009. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. Int. Trans. Oper. Res. 16, 641–657.
Richer, J.M., Goëffon, A., Hao, J.K., 2009. A memetic algorithm for phylogenetic reconstruction with maximum parsimony. Lect. Notes Comp. Sci. 5483, 164–175.
Richer, J.M., Rodriguez-Tello, E., Vazquez-Ortiz, K.E., 2012. Maximum parsimony phylogenetic inference using simulated annealing. Proceedings of the EVOLVE 2012, Mexico City, Mexico. Adv. Intell. Soft Comp. 175, 189–203.
Ronquist, F., 1998. Fast Fitch-parsimony algorithms for large data sets. Cladistics 14, 387–400.
Roshan, U.W., Moret, B.M.E., Williams, T.L., Warnow, T. 2004. Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. Proceedings of the 3rd IEEE Computational Systems Bioinformatics Conference (CSB 2004), 16–19 August, Stanford, CA, pp. 98–109.
Stamatakis, A., Ludwig, T., Meier, H., 2005. RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. Bioinformatics 21, 456–463.
Sundberg, K., Clement, M., Snell, Q., Whiting, M., Crandall, K. 2012. Phylogenetic search through partial tree mixing. BMC Bioinformatics 13 (Suppl.), S8–S18.

Swofford, D.L. 1986. Studies in numerical cladistics: phylogenetic inference under the principle of maximum parsimony. PhD thesis, University of Illinois at Urbana, Champaign, pp. 289.
Swofford, D.L. 2001. PAUP*: Phylogenetic Analysis using Parsimony (* and Other Methods). Sinauer Associates, Sunderland, MA.
Swofford, D.L., Maddison, W.P., 1987. Reconstructing ancestral character states under Wagner parsimony. Math. Biosci. 87, 199–229.
Varón, A., Wheeler, W., 2013. Local search for the generalized tree alignment problem. BMC Bioinformatics 14, 66–78.
Varón, A., Vinh, L.S., Wheeler, W.C., 2010. POY version 4: phylogenetic analysis using dynamic homologies. Cladistics 26, 72–85.
Vazquez-Ortiz, K.E. 2011. Metaheurísticas para la resolución del problema de máxima parsimonia. Tesis de Maestría, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Laboratorio de Tecnologías de la Información, pp. 116.
Vazquez-Ortiz, K.E., Rodriguez-Tello, E. 2011. Metaheuristics for the maximum parsimony problem. Proceedings of the Sixth IASTED CIB 2011, November, Pittsburgh, PA. ACTA Press, Calgary, pp. 105–113.
Viana, G.V., Gomes, F.A., Ferreira, C.E., Meneses, C.N. 2007. Uma implementação eficiente de uma heurística de busca local em multi-vizinhanças para um problema de filogenia. Proceedings of the XXXIX Simpósio Brasileiro de Pesquisa Operacional, Fortaleza, Brasil, pp. 2045–2056.
Viana, G.V., Gomes, F.A., Meneses, C.N., Ferreira, C.E., 2009. Parallelization of a multineighborhood local search heuristic for a phylogeny problem. Int. J. Bioinform. Res. Appl. 5, 163–177.
Vianna, D.S. 2004. Heurísticas híbridas para o problema da filogenia. Tese de Doutorado, Pontificia Universidad Católica de Río de Janeiro, pp. 101.
White, W.T., Holland, B.R., 2011. Faster exact maximum parsimony search with XMP. Bioinformatics 27, 1359–1367.
Williams, T., Berger-Wolf, B.M.T., Roshan, U., Warnow, T. 2004. The relationship between maximum parsimony scores and phylogenetic tree topologies. Technical Report TR-CS-2004-04, Department of Computer Science, The University of New Mexico.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comp. 1, 67–82.
Yan, M., Bader, D. 2003. Fast character optimization in parsimony phylogeny reconstruction. Technical Report, August 27, University of New Mexico at Albuquerque.
Zander, R., 1995. Phylogenetic relationships of *Hyophiladelphus* gen. nov. (Pottiaceae, Musci) and a perspective on the cladistic method. Bryologist 98, 363–374.