GCS# 150420-004244

Molecular Phylogenetics and Evolution 88 (2015) 93–104

Contents lists available at ScienceDirect

Molecular Phylogenetics and Evolution

journal homepage: www.elsevier.com/locate/ympev

Identifying unstable taxa: Efficient implementation of triplet-based measures of stability, and comparison with Phyutility and RogueNaRok ☆

Pablo A. Goloboff *, Claudia A. Szumik

Consejo Nacional de Investigaciones Científicas y Técnicas, Unidad Ejecutora Lillo, Miguel Lillo 251, 4000 S.M. de Tucumán, Argentina

ARTICLE INFO

Article history:
Received 7 November 2014
Revised 3 April 2015
Accepted 5 April 2015
Available online 10 April 2015

Keywords:
Reduced consensus
Rogue taxa
Unstable taxa
Ambiguity
Phylogenetic analysis

ABSTRACT

This paper describes an efficient implementation of triplet-based measures of stability, in the program TNT. The only available implementations of such measures are much slower than the present one, either because of an inefficient implementation (Phyutility, Thor) or because the stability is evaluated with quartets (RogueNaRok, requiring $O(t^4)$, instead of the $O(t^3)$ possible for triplets). The method to quickly calculate triplets is applied to solving IterPCR (Pol and Escapa, 2009). It is shown that, in some cases, IterPCR or other algorithms in the program TNT (e.g. commands ***prunnelsen***, ***prunmajor***, or ***chkmoves***) produce more informative results than analysis with RogueNaRok.

1. Introduction

As long recognized, taxa that can vary their position in different optimal or near-optimal trees (wildcard, rogue, unstable, or floating taxa) may decrease the resolution and support of otherwise well supported clades. A solution often used for avoiding these problems is pruning the unstable taxa from the trees used to calculate the consensus. It must be noted that a given selection of taxa to include in the consensus is neither correct nor incorrect – each selection simply displays different aspects of the information from the original trees. When some taxa are excluded, the resulting consensus is totally silent regarding the phylogenetic position of the excluded taxa, and the consensus cannot be "wrong" regarding that which it does not say. The difference between alternative taxon prunings – or no pruning at all – lies only in how useful the resulting trees are, for the study in question and from the point of view of the taxonomist.

Wilkinson (1994, 1995, 1996) was among the first to publish some methods to deal with the problem of unstable taxa in phylogenetic analysis. Among many other early examples, Goloboff (1995) presented a consensus excluding a taxon that could be placed in very different positions in the optimal trees. The topic of wildcard and unstable taxa was one of the topics often discussed at meetings in the early nineties. Other early discussions of this problem were also provided by Thorley and Wilkinson (1999), Anderson (2001), Kearney and Clark (2003), as well as many other papers. More recently, the topic was revisited by Pattengale et al. (2011), Aberer and Stamatakis (2011), and Aberer et al. (2013), who proposed several algorithms for improving group frequencies from sets of bootstrap trees.

Given that the need to identify taxa of unstable positions had long been established, the earliest versions of TNT (version 0.2, presented at the 19th meeting of the Willi Hennig Society in Leiden, 2000; and version 0.8, presented at the 22nd meeting in New York, 2003; see Stevenson, 2004, p. 84) already included a facility to find (using brute force) taxa decreasing the resolution of the strict consensus (in the ***prunnelsen*** command). Subsequent versions expanded the repertoire of options for other types of consensus or specific groups, with the commands ***pruncom***, ***prunmajor***, ***chkmoves***, and ***taxonomy*** (see Table 1 for brief description). Some of these options were used by Goloboff et al. (2009) in their analysis of 73,060 taxa, to produce a pruned consensus of 71,181 taxa displaying many additional taxonomic groups. None of the papers by Pattengale et al. (2011), Aberer and Stamatakis (2011), or Aberer et al. (2013), cite or discuss any of these commands and options.

TNT now also implements identification of rogue taxa in a set of trees with Pol and Escapa's (2009) method of IterPCR, in the ***pcrprune*** command. IterPCR is based on checking, for each taxon, the fraction of triplets including it that are resolved identically in all the trees. Pol and Escapa (2009) had implemented their method with a TNT script, which serves as a proof-of-concept but is

☆ This paper has been recommended for acceptance by N. Saitou.

* Corresponding author.
E-mail address: pablogolo@yahoo.com.ar (P.A. Goloboff).

http://dx.doi.org/10.1016/j.ympev.2015.04.003
1055-7903/

Table 1
Commands to help detect unstable taxa in TNT.

Command	Available in TNT since	Description	Published reference
prunnelsen	May 2000	Use brute force (=enumeration) to find taxa decreasing resolution of strict consensus	Goloboff et al. (2008)
pruncom	June 2004	Use brute force (=enumeration) to find taxa decreasing resolution of semistrict consensus	Goloboff et al. (2008)
prunmajor	March 2007	For each group in a reference tree, find the most similar group in one of the trees to be summarized (as explained in Goloboff et al., 2009: 215), keeping note of the taxa to be deleted to make groups identical. Taxa that have to be deleted for more groups/trees have the highest deletion scores. Heuristic, improves majority rule trees	Goloboff et al. (2008)
chkmoves	April 2007	Perform TBR (possibly accepting moves suboptimal by a specified absolute and relative fit difference; Goloboff and Farris, 2001), and list the taxa/clades that can be moved to more (or more distant, optionally) locations. Provides character-based assessments of stability	Goloboff et al. (2008)
taxonomy	December 2011	Finds taxa to prune to match groups in a reference taxonomy	Goloboff and Catalano (2012)
pcrprune	March 2012	Pol and Escapa's (2009) method, IterPCR	This contribution

computationally very inefficient, with the consequence that only relatively small numbers of taxa and/or trees can be analyzed.

The present paper describes how Pol and Escapa's (2009) method can be implemented efficiently, and provides some comparisons with other programs that implement triplet-based measures of stability.

2. EA, IterPCR and LSI

The earliest measure of taxon stability (EA, "explicitly agree") was proposed by Estabrook (1992); it calculated for each terminal the proportion of all quartets including it that are identically resolved in each of the input trees. If a taxon moves around in the trees, then many quartets and triplets including it will be resolved differently in different trees. Thorley and Wilkinson (1999) provided measures derived from Estabrook's, or "leaf stability indices" (LSI), and extended them to triplets (instead of quartets). One of the measures of Thorley and Wilkinson (1999) calculates for each triplet the difference in frequency between the most common and the second most common resolution; for each terminal, the LSI is in this case the average difference in all the triplets including the terminal. Using triplets instead of quartets to calculate the EA or LSI produces *rooted* measures of stability.

EA and LSI require checking all the possible triplets for the $\boldsymbol{t}$ taxa, and the number of triplets grows rapidly with number of taxa, approaching $\boldsymbol{t}^3$. The unrooted version of these measures, using quartets, requires checking a number of quartets that increases much more rapidly, with $\boldsymbol{t}^4$. Using quartets is normally unnecessary for summarizing the results of the analysis of a single data set, which produces trees rooted on the same outgroup taxon – the triplet-based measures are sufficient in that context.

Pol and Escapa (2009) noted that these indices were best calculated by creating a reduced subproblem for each polytomy of degree $\boldsymbol{d}$ in the strict consensus, iteratively recalculating values of stability (PC, positional congruence, similar to EA) after elimination of the node with worst value; they called their method IterPCR. But, although considering a reduced subproblem of the $\boldsymbol{d}$ descendants of a polytomous node requires much less work than for the full taxon set $\boldsymbol{t}$, the number of triplets to check still increases with $\boldsymbol{d}^3$. In Pol and Escapa's implementation of IterPCR, the cost of checking whether a triplet is resolved identically in all input trees increases with number of terminals (linearly or more), so that the cost of calculating IterPCR for a polytomy increases with at least $\boldsymbol{d}^4$. A more efficient implementation is needed for the method to be used in larger instances.

3. Previous work

3.1. Computer programs and scripts

The first program to calculate LSI was RadCon (Thorley and Page, 2000), but only the code for the program is available today – existing binaries are for the PPC Mac, no longer supported. To our knowledge, only two other programs to calculate triplet-based measures of stability are available today. These are Phyutility, published by Smith and Dunn (2008), and the THOR python script by Hill and Wilkinson (available at http://code.google.com/p/phylogenetics/, used by Lahr et al. (2011)). Phyutility has been widely used and cited (e.g. Struck et al., 2011; Nosenko et al., 2013).

RogueNaRok is primarily intended to implement maximization of the Relative Bipartition Information Criterion, RBIC (defined as the sum of all support values divided by the maximum possible support in a fully bifurcating tree with the full taxon set; Aberer et al., 2013). However, the RogueNaRok suite (Aberer et al., 2013) also implements the unrooted version of the LSI, based on quartets (no triplet implementation is included). Table 2 shows (second column) the times used by RogueNaRok to calculate the LSI for 100 trees for the classic Zilla rbcL data set (Chase et al., 1993, with 500 taxa), pruned to have varying numbers of taxa (the set of test trees was created by running 100 random addition sequences with no branch swapping, retaining all the trees, using a random seed of 1, with the Windows version of TNT). The implementation of quartets in RogueNaRok is indeed efficient, but the time needed to calculate the unrooted LSI with RogueNaRok increases much more rapidly than what could be done with an efficient implementation of triplets; the curve $\boldsymbol{time} = \boldsymbol{K}\ \boldsymbol{t}^{4.0051}$ (where $\boldsymbol{K}$ is a constant that depends on the processor speed) fits the observed times with $\boldsymbol{r}^2 = 0.9923$.[1] Thus, the program quickly becomes impractical as matrices have increasing numbers of taxa.

Phyutility (third column in Table 2) is significantly slower than RogueNaRok for the numbers of taxa that could be checked, even when it is checking triplets instead of quartets. The empirical results for Phyutility indicate $\boldsymbol{time} = \boldsymbol{K}\ \boldsymbol{t}^{3.8891}$, with $\boldsymbol{r}^2 = 0.9981$. Based on the times needed on the computer used for this paper (an Intel I7-3770 processor running at 3.40 GHz) for increasing numbers of taxa, the time needed to calculate the LSI for 100 trees of 1000 taxa would be about 3.5 days in the case of RogueNaRok, and about 2 weeks in the case of Phyutility. These are extrapolations, because in our experiments RogueNaRok crashes for trees with more than 200 taxa, and Phyutility runs out of memory for trees with more than 350.

Phyutility is in fact so slow that a simple TNT script (as in Fig. 1) has a comparable speed (Table 2, fourth column), even when the script is being interpreted by TNT and it uses no time-saving shortcuts at all. This script is similar to the one used by Pol and Escapa (2009), but calculates the LSI (instead of PC) for all the taxa (instead of the nodes connected to a given polytomy in the

[1] This was estimated with more data than shown in Table 2.

Table 2
Time and memory usage for different programs to calculate the LSI (Thorley and Wilkinson, 1999), both rooted (Phyutility and TNT-script, based on triplets), or unrooted (RogueNaRok, based on quartets). TNT-script used the same routine indicated in Fig. 1. The times are for the trees resulting from 100 random addition sequences for Zilla, retaining all trees, with no branch-swapping, and all the other options as default in the Windows version of TNT, pruned to contain different numbers of taxa. The time to complete Phyutility and TNT-script runs for larger numbers of taxa was extrapolated, from the number of triplets examined in runs interrupted after 5–10 min (this was automatically handled by the scripts used). RogueNaRok crashes for trees with more than 200 taxa.

Taxa	T I M E (s)			RAM (Mbytes)	
	RogueNaRok	Phyutility	TNT-script	RogueNaRok	Phyutility
50	2.7	12.84	5.53	0.9	–
100	32.36	172.7	106.28	33	23.9
150	161.82	843.88	1179.27	74	49
200	507.02	2474.46	2113.98	125	89.5
250	–	5270.9	5519.62	–	154.6
300	–	13711.98	24937.50	–	248.3
350	–	28983.17	25422.43	–	257.4

```
macro=;
sil = all ;
var: mysec tback danod trip[3] dif[root] mydif ;
set danod root + 1 ;
set tback ntrees + 1 ;
macfloat 0 ;
collapse none ;
loop 0 ntax set dif[#1] 0 ; stop
hold + 2 ;
combine root 3 trip
   ttag -;
   ttag = ;
   freqdif /+.-'trip[0-2]' ;
   copytree + ;
   if ( tnodes[ ntrees ] )
      set mydif ( $ttag 'danod' ) ;
      set dif[ 'trip[0]' ] += 'mydif' ;
      set dif[ 'trip[1]' ] += 'mydif' ;
      set dif[ 'trip[2]' ] += 'mydif' ;
      end
   keep 'tback' ;
   endcomb ;
macfloat 10 ;
sil - all ;
loop 0 ntax
   set dif[#1] /= 100 * ( ntax * ( ntax - 1 ) ) / 2 ;
   quote $taxon #1: 'dif[#1]' ;
   stop ;
proc/;
```

Fig. 1. TNT script to calculate leaf stability indices (LSI, Thorley and Wilkinson, 1999). The script enumerates all combinations of triplets and calculates the frequency-difference consensus (Goloboff et al., 2003) for each taxon subset.

consensus); note that the script provides an approximation to the LSI values, because the ***freqdif*** command outputs values of frequency as truncated percentages, with no decimals (thus producing some differences in the third or fourth decimal).

Finally, THOR is significantly slower than Phyutility or RogueNaRok, hardly surprising since it is a Python script interpreted at run-time.

3.2. Theoretical methods for faster calculations

Although the time dependence of Phyutility approaches $O(t^4)$, a number of more theoretically oriented papers have proposed methods to check triplets with cost $O(t^3)$. For significant numbers of taxa, this is orders of magnitude faster than what can be done with Phyutility, or the $O(t^4)$ used by RogueNaRok to check quartets.

It must be noted that the calculation of the triplet distance between two trees can in fact be done faster than $O(t^3)$, with complexities as low as $O(t^2)$ or even $O(t \ln t)$, with methods such as those reviewed by Sand et al. (2013). However, those methods are based on simply calculating the number of triplets not shared by two trees, without identifying them. Examining the individual triplets in a set of trees is required for methods like LSI and especially IterPCR (in this case, because those triplets can be stored and later reused for calculating values of PC after exclusion of some taxa; see Section 4.3).

Mosses (2005) noted that existing methods for identifying most recent common ancestors in constant time (e.g. Schieber and Vishkin, 1988; Gusfield, 1997) could be used to determine the frequency of all triplet resolutions with time $O(t^3)$. However (as he noted), other methods to identify the resolution for each triplet in constant time may have advantages over the identification of most recent common ancestors. Mosses (2005) proposed another method to calculate the triplets supported by the input trees (p. 40) with cost $O(t^3)$. His method was implemented in a program (see Appendix B, p. 93), which was apparently available at the time when Ranwez et al. (2010) published their paper with another supertree method based on triplets. The program, however, is no longer available at the URL given in Mosses (2005), and used the counts of triplet frequencies to subsequently calculate a supertree, not the PC or LSI.

Lin et al. (2009), when describing another method to create triplet supertrees, state that "for each possible triplet over the set of all taxa [n] we count and store the frequency displayed by all the input trees [k] in $O(kn^3)$ time". The paper is concerned with the more computationally demanding aspects of the method (i.e. in finding the tree once the frequency for each triplet has been checked in all the input trees), and then gives no details of how the $O(kn^3)$ calculation is carried out; no program is provided with their paper (nor is available at the "software" page of the Computational Biology Laboratory, http://genome.cs.iastate.edu/CBL/download/).

4. The method implemented in TNT

The implementation in TNT also uses a method which checks each triplet in approximately constant time. The algorithm has some similarities with methods proposed by Mosses (2005), but is simple enough that it can be implemented even in scripts. The Supplementary Material includes a TNT script which calculates (for a single tree) the resolution of each triplet, graphically showing each step of the algorithm, in less than 200 lines of code. The method used in TNT differs from Mosses' (2005) in not checking triplets based on lists of descendants of different internal nodes, but using instead marked intersecting paths. The algorithm is discussed below, using as reference the example of Fig. 2 and the pseudocode of Fig. 3.

4.1. Basic method

Consider as example the tree in Fig. 2. To check all triplets, a triply nested loop is used; the first two loops determine the first two taxa of the triplet, and the third level loop determines the third. The key to accelerated calculation is in a preprocessing done after

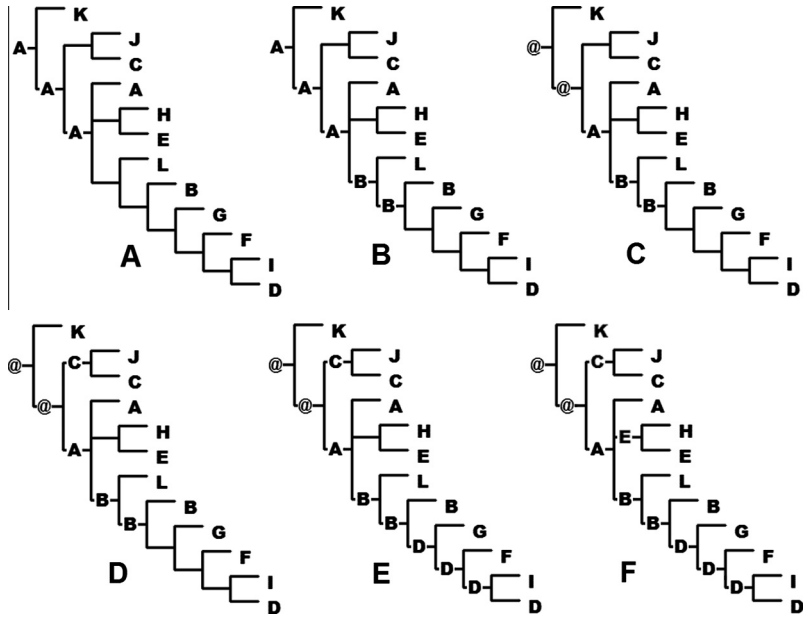

Fig. 2. A tree illustrating the marked intersecting paths used to check triplet resolution in TNT. A, the path below the first taxon (A) is marked; B, the path below the second taxon (B) is marked; C, the path below the intersection node is marked as a non-terminal taxon (@); D, the path below the third taxon (C) is followed until it crosses a marked branch –since a branch marked as non-terminal is found first, then it can be concluded that the triplet is resolved as (C(AB)); E, following the path below taxon D, a branch marked as B is encountered first, indicating that the triplet is resolved as (A(BD)); F, when the first marked node is the intersection point between the first two marked paths, then a polytomy is concluded, (ABE).

the first two taxa of a triplet are selected; this allows using that preprocessing for speeding up the calculations for the triplets formed by those two first taxa and each of the remaining taxa. This preprocessing is done at lines §1–§4 of the pseudocode, and at the top row of Fig. 2. The preprocessing first unmarks all tree nodes, then places three types of marks. First, it marks each node below the first taxon as the taxon (Fig. 2A, or Fig. 3, line §2). Then, it marks the nodes below the second taxon (Fig. 2B, Fig. 3 line §3) until it finds either a marked node, or the tree root; the node at the intersection between the two paths (i.e. the first node marked as A encountered when coming down from taxon B in Fig. 2B) is recorded for future identification of polytomies (using in the pseudocode, Fig. 3 line §3, the variable ***entrypoint***). Last, the preprocessing marks all nodes below the intersection with a code different from each of the terminal taxa (in Fig. 2C, marked as @; in the pseudocode, Fig. 3 line §4, numbered as ***numtaxa*** – keep in mind that terminal taxa are numbered from 0 to number of taxa minus 1).

After this preprocessing, it is possible to check the triplets formed by those first two taxa and each of the remaining taxa with successively less work required. For each of those terminals, the first node marked found when traveling toward the root indicates the resolution of the triplet. If it is a node marked as the first taxon, then the triplet is resolved with the first and third taxa closer to each other; if marked as the second, then the triplet is resolved with the second and third taxa closer; if marked as the number of taxa, then the triplet is resolved with first and second taxa closer. This is shown at line §6 and subsequent lines in the pseudocode. It is also shown in Fig. 2D for terminal taxon C (since a node marked as root is encountered, it can be concluded that taxa A and B are closer than they are to C), and in Fig. 2E for terminal taxon D (since a node marked as B is encountered, it can be concluded that B and D are closer than they are to D). When the first marked node found traveling toward the root is the intersection point (as in Fig. 2F, for ABE), then the triplet is unresolved in the tree.

The nodes visited while traveling down each of the third member of a triplet can also be marked, recording for each third member already used the initial mark it leads to (with variable ***ends_at*** in line §5 of the pseudocode). This is shown in Fig. 2E, where the three internal nodes between terminal taxon D and the first node marked as B are themselves marked as D when traveling down. This makes it possible to check subsequent triplets by traveling down fewer nodes. For example, after marking the nodes between D and the first marked node below as D, and recording that D leads to a B mark, it is possible to travel down F just one node, encountering a node marked as D. Since D had previously been recorded as leading to a B mark, then it can be concluded that B and F are closer to each other than they are to A. In fact, for the tree in Fig. 2, after the node below taxon E has been marked (in Fig. 2F), each of the internal nodes has a mark, and each of the seven remaining taxa (F–L) can be checked by traveling down a single node.

This algorithm can be used for both PC and LSI, but in the case of PC, calculations can be further speeded up. PC calculates the proportion of triplets with a given terminal taxon that are resolved identically in all the trees. Thus, when checking each triplet, as soon as a tree is found where the triplet is unresolved, or resolved differently from previous trees, there is no need to check the triplet in the remaining trees.

4.2. Speeding up the preprocessing

Especially in the case of PC (and IterPCR), a significant proportion of the time may be used only in unmarking every node of every tree (line §1 of pseudocode). In the case of LSI, this time is not so significant, because every triplet must be checked on every tree; in the case of PC, abandoning the checking of a triplet after looking at only the first trees means that the proportion of time used in the inner loops is lower. Thus, saving time in the preprocessing may have an actual impact on the final times.

In the pseudocode of Fig. 3, ***EMPTY*** is defined as a constant. Instead, ***EMPTY*** can be defined as a value lower than any value a marked node could possibly take. This requires marking nodes with values larger than ***numtaxa***. This can be done by adding a value (***curkey***) to each of the marked nodes, increasing ***curkey*** with

```
function fill_LSI ( ) {
 for ( i = 0 ; i < numtaxa - 2 ; ++ i ) {  /** taxon i **/
    for ( j = i + 1 ; j < numtaxa - 1 ; ++ j ) { /** taxon j **/
       §1  set mark for every node x, of every tree a, to EMPTY
       §2  for each tree a, for each node x in the path between i and root, set mark[ a ][ x ] = i
       §3  for each tree a,  for each node x in the path between j and intersection with previous path,
           set mark[ a ][ x ] = j; record, for each tree a, node x in the intersection, as entrypoint[ a ] = x
       §4  for each node x in the path between entrypoint[ a ] and root, set mark[ a ][ x ] = numtaxa
        for ( k = j + 1 ; k < numtaxa ; ++ k ) {  /** taxon k **/
            countout [ 0 ] = countout [ 1 ] = countout [ 2 ] = 0 ;
            for ( a = 0 ; a < numtrees ; ++ a ) {   /**  trees **/
               ends_at = ends_at_space[ a ] ;
               §5  mark path, for tree a, for each node x between k and the first_node below that is non
                   EMPTY, setting mark[ a ] [ x ] = k
               §6  endcode = ends_at [ mark[ a ][ first_node ] ]
               if ( endcode == i || endcode == j || endcode == numtaxa )
                   isout = ends_at [ k ] = endcode ;
               else isout = ends_at [ k ] = ends_at [ endcode ] ;
               if ( first_node == entrypoint[ a ]  )
                  isout = ends_at [ k ] = numtaxa + 1 ;  /** a polytomy! **/
               if ( isout <= numtaxa )
                 if ( isout == i ) ++ countout [ 0 ] ;
                 else if ( isout == j ) ++ countout [ 1 ] ;
                 else ++ countout [ 2 ] ;
               }  /** end trees **/
          §7  use values of countout (frequency of each resolution) to calculate freqdiff = frequency difference
           sumfreqs[  i ] += freqdiff ;
           sumfreqs[  j ] += freqdiff ;
           sumfreqs[  k ] += freqdiff ;
          }  /** end k **/
       }  /** end j **/
    }  /** end i **/
   for ( a = 0 ; a < numtaxa ; ++ a )
    LSI_vals[ a ] =  2 * sumfreqs[ a ] / ( ( numtaxa-1 ) * ( numtaxa-2 ) ) ;
   return ; }
```

Fig. 3. C-style pseudocode implementing the algorithm for fast triplet calculation discussed in the text. The function ***fill_LSI*** uses several arrays which must have been allocated previously. ***Numtrees*** and ***numtaxa*** indicate the number of trees and terminal taxa. The arrays used are ***LSI_vals*** (with ***numtaxa*** cells, used to store the values of LSI for each taxon); ***mark*** (a 2-dimensional array, with ***numtrees*** × ***numtaxa*** cells); ***entrypoint*** (with ***numtrees*** cells, indicating the intersection between the paths below the first two taxa of a triplet, ***i***, ***j***); ***countout*** (with 3 cells, used to indicate how many trees had the first, second, or third member of the triplet as less closely related than the other two); ***sumfreqs*** (with ***numtaxa*** cells, must be initialized to zero before calling ***fill_LSI***, indicating the frequency difference for the different resolutions; see Thorley and Wilkinson, 1999). See text for additional discussion.

every round of preprocessing, so that nodes marked in previous rounds remain set at values small enough to be ignored. ***EMPTY*** must also be defined as a variable, which increases together with ***curkey***. Then, ***EMPTY*** is initialized at the beginning of the function as −1 and ***curkey*** as 0. For every round of preprocessing, both variables are increased in ***numtaxa*** + 1. Each node mark is done with the corresponding value of ***i***, ***j***, or ***k*** plus ***curkey***. Traveling down a terminal taxon continues until a node marked with a value greater than ***EMPTY*** is found. The taxon to which the mark encountered corresponds is calculated by subtracting ***curkey*** from the mark. When after several rounds of preprocessing ***EMPTY*** and ***curkey*** become larger than a certain limit (e.g. more than can be held in a 32-bit number), they are reset to −1 and 0 respectively, and the following round of preprocessing needs to visit each of the nodes of each tree. By defining this limit to be a large enough number (e.g. 10^9), the full visit of all tree nodes for unmarking is done only after thousands of rounds of preprocessing, even for numbers of taxa in the order of 10^5 or 10^6.

With this technique for avoiding superfluous work during preprocessing, the amount of time saved for large numbers of taxa can be in the order of 25–30% in the case of relatively similar trees (trees that are most parsimonious, or close to), to up to 50% in the case of PC for very different trees.

4.3. Storing triplet information with low memory requirements

In the case of IterPCR, Pol and Escapa (2009) noted that the individual PC values need to be recalculated again after each round of elimination, because triplets including one of the eliminated taxa are no longer relevant, thus changing proportions of identically resolved triplets for the taxa remaining. However, whether a given triplet is identically resolved on all trees cannot change with elimination of other taxa not included in the triplet, and storing this information for each triplet avoids the need to recheck all the trees again for each round of elimination. Storing the triplets can thus potentially greatly speed up calculation of IterPCR. It can also speed up the calculation of PC values, by allowing triplets not yet checked to be inferred from triplets already checked (see Section 4.6).

The easiest, most naïve way to store the triplets would be to create a 3-dimensional array, where each cell uses a single byte. Thus, if this variable is named ***ident***, then ***ident[i, j, k]*** = 1 indicates that the triplet ***i***, ***j***, ***k***, is resolved identically in all trees.

However, that naïve method would be a waste of memory. First, many combinations of values of ***i***, ***j***, ***k*** are impossible, because the triplets are checked in such a way (see Fig. 3) that ***j*** can never be equal or greater than ***i***, and ***k*** can never be equal or greater than

j. This also means that, as the values of ***i*** (the first dimension) increase, then the number np_j of possible values for index ***j*** decreases, with $np_j = t - i - 2$. Likewise for index ***k***, so that $np_k = t - j - 1$. In this way, the triplet formed by taxa ***i***, ***j***, ***k*** (the inequality $k > j > i$ is mandatory for this to work) is not stored at ***index[i, j, k]***, but instead at ***index [i, j − i − 1, k − j − 1]***. While access to the array storing the information is more complicated, it is still an insignificant time compared to the time needed to re-check the triplet in each of the input trees, and the number of values that need to be actually stored is 1/6 of the naïve approach. This reduction in memory corresponds to the number of possible orderings of the three members of each triplet.

Another obvious way to save memory is by considering that storing whether the resolution of each triplet is identical in all the trees requires a single bit, not a byte. If ***index*** is made a three-dimensional array, where the last dimension contains as many 32-bit cells as needed to store the bits required, then the information on triplet resolution can be accessed by checking whether the corresponding bit in the array is on or off (this technique, used in RogueNaRok, is also mentioned in Aberer's 2011 thesis, but he makes no mention of the method explained in the previous paragraph; note also that RogueNaRok needs to store resolution of quartets, much more numerous than triplets). Each of the cells in the previous case contained eight bits, but only one is actually needed, so that the memory used reduces to 1/8 of the 1/6, or 1/48 of the memory needed by the naïve approach. The actual number of 32-bit cells to allocate is $np_k + 31/32$. The bit representing the triplet ***i***, ***j***, ***k*** is located at the cell corresponding to ***i***, $j - i - 1$, $k - j - 1/32$, and within that cell, it is the bit identified by the remainder of dividing $k - j - 1$ in 32 (easily done with the module operation, indicated with % in the pseudocode). The pseudocode to allocate memory, store triplets, and subsequently access them is shown in Fig. 4. The actual RAM required to store all triplets for different numbers of taxa is shown in Fig. 5 and the last column of Table 3 (compare with last two columns of Table 2; it is unclear whether Phyutility and RogueNaRok are actually storing the triplets or the memory allocated is being used for other purposes; triplet storage is not really needed for calculation of the LSI).

Although this is in principle not needed for IterPCR, if some method requires storing the individual resolution of a triplet, then a similar approach using 2 bits per triplet could be used (4 values are possible with 2 bits, enough to indicate each of 3 taxa as sister to the others, or unresolved). Storing the individual resolution of each triplet allows further speedups in the case of PC and IterPCR (see Section 4.6) because it makes it possible to infer triplets not yet checked from triplets already checked, at the expense of memory.

4.4. Creating reduced trees

In IterPCR, the resolution of each individual polytomy in the strict consensus tree is treated separately, by creating a reduced subproblem. In TNT, these reduced instances are created automatically, by internally creating reduced trees that represent the nodes connected to the polytomy. That is, for a polytomy where an ancestral node leads to ***n*** descendants, reduced trees with ***n*** leaves (with descendants renumbered from 0 to $n - 1$) are created. The creation

```
void set_mem_bufs ( int numtaxa)
{
   int a , b , i , j ;
   ident = allocate ( (numtaxa – 2 ) * sizeof ( int ** ) ) ;
   for ( i = 0 ; i < numtaxa – 2 ; ++ i ) {
      ident [ i ] = allocate ( (numtaxa – i – 2 ) * sizeof ( int * ) ) ;
      for ( j = i + 1 ; j < numtaxa – 1 ; ++ j ) {
        a = (numtaxa – j – 1 ) ;
        b = 1 + ( ( a + 31 ) / 32 ) ;
        ident [ i ] [ j – i – 1 ] = allocate ( b * sizeof ( int ) ) ; }}
  }

void storem ( int i , int j , int k )
{
    unsigned int the_bit ;
    the_bit = 1 << ( ( k – j – 1 ) % 32 ) ;
    ident [ i ] [ j – i – 1 ] [ ( k – j – 1 ) / 32 ]  |= the_bit ;
   }

int is_triplet_identical ( int i , int j , int k )
{
   unsigned int the_bit;
   the_bit = 1 << ( ( k – j – 1 ) % 32 ) ;
   if ( ( ident [ i ] [ j – i – 1 ] [ ( k – j – 1 ) / 32 ] & the_bit ) ) return 1 ;
   return 0 ;
  }
```

Fig. 4. C-style pseudocode showing how to allocate memory (***set_mem_bufs***), store that a triplet ***i***, ***j***, ***k*** is resolved identically in all trees (***storem***), and to retrieve whether a triplet is resolved identically in all trees (***is_triplet_identical***). Code to store how the triplet is resolved is similar, but using two bits per triplet.

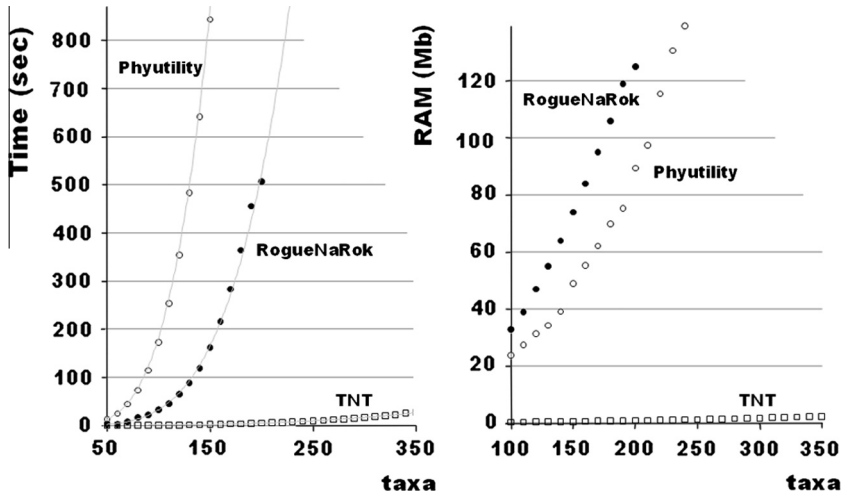

Fig. 5. Plot of time versus number of taxa (left), and RAM used versus number of taxa, for Phyutility, RogueNaRok, and TNT, on 100 trees, calculated as indicated in the text. The RAM for Phyutility is an approximation, recorded from the Windows memory manager, as the amount of RAM used becomes more stable during the initial stages of the run. The RAM used by TNT corresponds to a data set with no characters (thus using memory only to hold the taxon names and trees, just like Phyutility), storing the triplet resolution for subsequent use (as indicated for IterPCR in Section 4.3).

Table 3
Time and RAM usage for the implementation in TNT, for both PC (Estabrook, 1992) and LSI (Thorley and Wilkinson, 1999), in the case of 100 trees with different numbers of taxa. The times differ for moderately parsimonious trees (*PT*, the same tree sets used for Table 1), or random trees (rand). *PT** indicates the same set of moderately parsimonious trees, but inferring triplets from previously calculated triplet (this requires additional amounts of RAM, to store triplet resolution, as indicated in last column). Nodes/triplet is the average number of nodes that have to be traveled in the innermost loop (at line §5 of the pseudocode, see Fig. 3), for the third member of each triplet (it does not include the number of nodes visited during the preprocessing done for the first two members of each triplet). RAM varies according to whether program stores only whether triplet is resolved identically in all trees (left column, a single bit per triplet needed), or whether it stores as well how the triplet is resolved (last column, two bits per triplet needed).

Taxa	T I M E (s)					Nodes/triplet		RAM (MBytes)	
	PC			LSI		*PT*	Rand	Triplet ident	Triplet resol
	PT	*PT**	Rand	*PT*	Rand				
50	0.03	0.02	0.02	0.03	0.06	1.96	2.26	0.17	0.17
100	0.17	0.11	0.08	0.33	0.53	2.17	2.52	0.36	0.38
150	0.55	0.30	0.17	0.98	1.83	2.09	2.62	0.61	0.67
200	1.22	0.63	0.39	2.45	4.63	2.04	2.68	0.92	1.07
250	2.16	1.22	0.72	4.89	9.19	2.05	2.73	1.32	1.62
300	4.05	2.05	1.16	8.41	16.47	2.03	2.79	1.82	2.34
350	6.19	2.97	1.8	13.28	26.86	2.05	2.83	2.43	3.26

of these reduced trees is straightforward and requires no special discussion.

4.5. Performance analysis

The exact time complexity of the method is hard to calculate. For binary trees, after preprocessing a pair of taxa, then for the third triplet member, every tree node must be visited, on average, about two times (first time to mark it on the way down from the first descendant, second on the way down from the other). This suggests a constant time for each triplet, and a total time $O(t^3)$. But this is the case only when the first pair of taxa can be combined with most of the remaining taxa. Consider the case of 26 taxa named alphabetically from to A to Z; when picking pair X and Y, then the only possible third triplet member is Z, and it is likely that several nodes will have to be traveled until the marked path is intersected. To further complicate matters, the time used in preprocessing is proportionally negligible when picking the first pairs of taxa (e.g. A and B in the example), because then many triplets can be checked efficiently on the basis of that preprocessing (C–Z, in the example), but proportionally significant when picking the last pairs of taxa (such as X and Y, which can only be combined with Z). Which of the two aspects of the calculations dominates the process, and how this dominance changes with numbers of taxa, is difficult to calculate exactly. The precise number of nodes visited (both in the preprocessing, and in the final checking of triplets) also depends on the tree topology, and even on how the taxa are numbered in the matrix.

Thus, the time complexity can be theoretically expected to increase *approximately* with $O(t^3)$, but there may be minor variations relative to this expectation. The method was thus tested empirically, with the same set of trees used to test Phyutility and RogueNaRok (see Table 2). The results are shown in Table 3, and it can be seen that the difference in actual times with those programs is very significant. For (near) most parsimonious trees, the times for the calculation of LSI fit the curve $\boldsymbol{time = K\ t^{2.9872}}$ with $\boldsymbol{r^2} = 0.998$; in the case of random trees, the curve is somewhat steeper, with $\boldsymbol{time = K\ t^{3.12316}}$ ($\boldsymbol{r^2} = 0.99987$) (calculation of PC is significantly faster, as explained in Section 4.2). This difference agrees with the fact that the number of nodes that have to be traveled at the inner loop which identifies triplets remains approximately constant for near most parsimonious trees, while it increases with the taxa in the case of random trees (see Table 3). However, the increase in the number of nodes traveled is very moderate (e.g. for a 4-fold increase in taxa, from 100 to 400, it is necessary to visit 12% more nodes, from 2.515 to 2.822). Therefore, even in the worst case (calculation of LSI on random trees), TNT considerably outperforms Phyutility and RogueNaRok

(see Fig. 5). The work done by RogueNaRok is not strictly comparable to that of TNT, because RogueNaRok checks quartets, which is much more costly than checking triplets. Recall (from Section 3.1) that RogueNaRok can be extrapolated to take 3.5 days on 100 trees of 1000 taxa. When all the trees (as in the analysis of a single data set) are rooted on the same taxon, triplets suffice to identify unstable taxa, and with the method just described, TNT calculates the LSI for 1000 taxa on 100 random trees in 11.3 min.

4.6. Inferring triplets from already checked triplets

The final improvement for calculating PC values can be achieved by considering that some triplets jointly imply others. This dependence has been known for a long time (e.g. Dekker, 1986), and it has been extensively discussed specially in relation to three-item analysis (e.g. De Laet and Smets, 1988). For example, if each of the trees displays ***a***(***bc***) and ***a***(***bd***), it then follows that each of the trees must also display ***a***(***cd***). Table 4 shows all possible combinations for two triplets that share two taxa (***a***, ***b***) and display a unique taxon per triplet (***c***, ***d***).

Inferring a triplet from others avoids the need to check each of the trees for that triplet; the time saved by using this technique is then more significant when checking larger numbers of trees. The time saved is also more significant as the trees are more congruent – it is possible to infer more triplets without having to check them on the trees. Note that, in the case of PC, the tree sets that take longer to check are those comprising more similar trees (e.g. compare the times of parsimonious versus random trees, columns *PT* and *rand* in Table 3), so the method of inferring triplets is best applied in precisely that situation.

A problem with checking triplets is that, as calculations advance, there are many triplets that have been checked already; some of those will allow inferring the triplet to be currently checked, others will not. Finding the triplets previously checked that allow inferring the present one may thus be more costly than just checking the triplet on the trees.

To avoid that problem, the implementation in TNT uses a different approach; as it confirms that a triplet is identically resolved in all the trees, it then compares that triplet to previously resolved triplets comprising the same first and second member and attempts to resolve in advance triplets not yet checked. Recall (from the pseudocode of Fig. 3) that the triplet just checked is composed of taxa ***i***, ***j***, ***k***. Then, previously checked triplets comprising the same first and second member can be enumerated with a loop where ***n*** takes values between ***j*** and ***k*** (exclusive). Inside that loop, it is first necessary to check whether the triplet ***i***, ***j***, ***n*** is resolved identically in all trees (and how). Then, the resolution of the triplets ***i***, ***j***, ***k*** and ***i***, ***j***, ***n*** can be used to infer the resolution of the triplet ***i***, ***n***, ***k*** (as ***j*** < ***n*** < ***k***, it is guaranteed that the triplet ***i***, ***n***, ***k*** has not yet been checked). If, applying the rules of Table 4, the triplet ***i***, ***n***, ***k*** can be resolved from the other two, then its resolution is stored.

Table 4
Rules to combine triplets. Of the 9 possible combinations of resolutions for the triplets ***abc*** and ***abd***, 7 allow inferring an unambiguous resolution for the triplet ***acd*** and two (marked with ?) are compatible with more than one resolution for ***acd***.

abc	abd	acd
a(bc) +	a(bd)	a(dc)
	b(ad)	c(ad)
	d(ab)	d(ac)
b(ac) +	a(bd)	d(ac)
	b(ad)	?
	d(ab)	d(ac)
c(ab) +	a(bd)	c(ad)
	b(ad)	c(ad)
	d(ab)	?

As each triplet is enumerated to be examined on the trees, the program first checks whether the triplet has been resolved in advance from previous triplets. Recall, from Section 4.3, that resolution of a triplet is stored in two bits; value 0 is used to indicate no resolution, and values 1–3 are used to indicate respectively first, second, or third member of the triplet as sister to the rest (therefore, a value different from 0 at the corresponding location indicates that the triplet is resolved). If the triplet has been resolved, the examination of the trees for that triplet is skipped.

The exact time gain by using this technique is very hard to predict. When the examination of a triplet on the trees is not required (thus saving some time at some point of the calculations), this means that the paths below the third member of the triplet are not marked. Therefore, subsequent triplets for the same first two taxa that have to be checked on the trees may require visiting more nodes (thus increasing the time for other parts of the calculation); that subsequent triplets may or may not themselves be skipped simply complicates analysis of performance even more.

Then, the time gain can only be examined empirically. For random trees (data not shown), essentially no time is saved by applying this technique (in some cases, it may even be somewhat slower than calculations without using it). For that reason, using triplet inference in advance is optional in TNT. As can be seen in Table 3, for sets of parsimonious trees, the calculations using triplet inference in advance can proceed (for larger numbers of taxa) at about twice the speed (*PT** column) without using it (*PT* column). As expected, the speed is increased even more significantly when the trees are numerous and very similar; e.g. for a hundred trees of 500 taxa, with 5–10% of the taxa placed at random positions, and the remaining 90–95% related in exactly the same way, calculations proceed about 4 times faster.

5. Pruning trees or matrices?

The ***pcrprune*** command in TNT includes the option of saving to a taxon-group the taxa to prune for improving the consensus, so this group can be subsequently used to easily produce a pruned consensus. It must be stressed that the aim of the analysis is to find taxa which, when pruned from the *trees*, will improve resolution. This is different from eliminating the taxa from the *matrix*: pruning some taxa (rogue or not) may alter the relationships between the *other* taxa. Unless there are independent grounds to exclude those taxa from the analysis (e.g. the suspicion of taxonomic misidentification or errors in sequencing), then the character information provided by the rogue taxa must be considered. Of course, the condition of being a rogue may help detect chimaeric or missequenced taxa, but this needs to be independently verified – being a rogue can provide in itself no grounds for exclusion from the matrix.

That unstable taxa should not be excluded from the matrix has long been recognized. Kearney and Clark (2003), who seem otherwise skeptical of pruned consensus methods for dealing with the problem of missing entries, admit that "the clear advantage these methods have over omitting taxa is that they include all taxa that may affect the topology of the tree". Even earlier, Wilkinson (1995) had made it clear that taxa which alter the relationships between the other taxa should not be excluded from the matrix, and devised a method to detect those taxa which (by virtue of their character–state combinations) could not possibly affect the relationships between the other taxa (he called this method safe taxonomic reduction or STR; an improved heuristic was recently proposed by Siu-Ting et al., 2015). In the reduced consensus presented by Goloboff (1995, Fig. 63), the single rogue taxon had numerous missing entries (its alternative locations required the same number of steps for each character; Goloboff, 1995: 138), and it did not

affect the relationships between the other taxa. Of course, in many cases, "taxonomically safe" eliminations will not be sufficient to improve the resolution of the consensus, and then the taxa responsible for the decrease in resolution must instead be ignored (pruned) when calculating consensus. Excluding from the matrix the rogue taxa, when they affect the relationships between the other taxa, amounts to ignoring relevant information – the character–state combinations provided by the rogues are relevant precisely *because* they affect the relationships between the other taxa. Examples of some taxa affecting the relationships between the others have been known since the earliest days of cladistics. This is the very reason why Wagner trees may fail to find a most parsimonious tree (the relationships for the initially added taxa are not the same as when taxa not yet added are considered), and why analyses with a comprehensive representation of taxa are now preferred over "chicken–monkey–mouse" analyses (once acceptable, now seen as overly simplistic). A simple example of a taxon affecting the relationships between the other taxa is in Fig. 6A. The optimal tree displays ((**ab**)(**cd**)), but if taxon **d** is excluded from the matrix, then the optimal tree displays (**a**(**bc**)) – i.e. with **b** closer to **c** than it is to **a**, which is contrary to the relationships supported by all the evidence taken together. Bootstrapping the full matrix, and ignoring the position of **d** (i.e. pruning **d** from the *trees*, not from the *matrix*), produces a well supported ((**ab**)**c**). In contrast, bootstrapping with **d** excluded from the matrix produces a well supported (**a**(**bc**)). In the example, **d** is not a "rogue" taxon in the sense that it does not move among alternative positions in optimal or near-optimal trees, but the same effect can also be produced when taxon **d** *is* a rogue. This is the case for the modified matrix of Fig. 6B; calculating most parsimonious trees with TNT, then finding the rogue and displaying the strict consensus with indication of its possible locations (with "***pcrprune*** />0; ***nelsen*** / / {0};") produces the results shown in Fig. 6B. Running the matrix without **d** displays **a** closer to the group **c**–**h**, instead of closer to **b** as supported by all the evidence considered simultaneously.

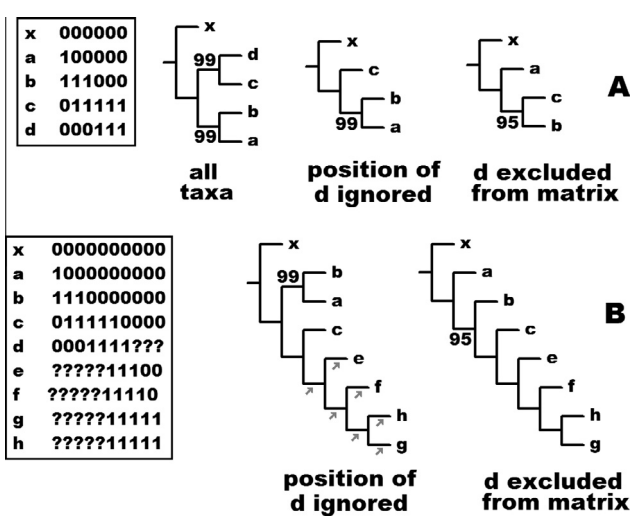

Fig. 6. Matrices for which inclusion/exclusion of a taxon (***d***) modifies the relationships between the other taxa, when analyzed under parsimony with equal weights; bootstrap frequencies are for every character represented by 10 identical copies. In the top matrix, ***d*** is not a rogue taxon. In the bottom matrix, ***d*** can change its position to be sister group of different taxa/clades in the rest of the tree, producing 7 equally parsimonious trees. The examples illustrate the difference between excluding taxa from the *trees* (which displays only selected aspects of the same input trees) and excluding taxa from the *matrix* (which ignores part of the evidence provided by the observations and can thus produce different trees).

Although Aberer had stated in his thesis (Aberer, 2011) that there is no need to exclude rogue taxa from the matrix, Aberer and Stamatakis (2011) recommend excluding them from the matrix and re-running the analysis. Aberer and Stamatakis demonstrate that excluding the rogues often produces a different set of relationships for the remaining taxa, but never explain why that is a reason to exclude the rogues and reanalyze the pruned matrix. It is in fact a reason to *not* exclude the rogues. The recommendation of Aberer and Stamatakis (2011) is against the most basic principles of empirical inference: as much relevant evidence as possible should be taken into account. Perhaps realizing this, in their subsequent paper Aberer et al. (2013) refer only to pruning taxa from trees.

6. Discussion and conclusions

The problem of improving group supports or consensus trees by pruning taxa is a complex and multifaceted one. Deepak et al. (2012) demonstrate that finding optimal reduced consensus trees (sensu Wilkinson, 1994) is an NP-hard problem. The RogueNaRok algorithms attempt to find taxa to prune maximizing two criteria: either the number of groups at a given cutoff frequency for the consensus (with the –***b*** flag), or the sum of group supports. Ideally, other factors should be considered as well: for example, pruning a taxon to gain a bipartition of 2, may be deemed less desirable than pruning a taxon to gain a bipartition setting apart a large group. The second case, at least in most situations, will be considered as more informative; it certainly allows for easier subsequent improvement by pruning additional taxa (as it produces two polytomies of lower-order than the big polytomy left in the former case). It is also necessary to consider the cost of pruning taxa, so that a trade-off is made between the number of consensus nodes gained and taxa pruned. The cost of pruning, furthermore, may not be uniform for all taxa (the researcher may be more interested in determining the relationships of some taxa, including taxa that cannot be pruned at all).

But the problem is not just computational: it is also necessary to consider whether the criteria themselves always produce the desired result. In the case of TNT, most commands, instead of calculating overall sums of supports or number of nodes in the whole tree, strive to improve the resolution or support of polytomous or poorly supported nodes, one at a time. In this regard, the PC values of the reduced problem representing a polytomy in the strict consensus are used only as a guide to try taxon (or group) prunings, but the actual effect of pruning is assessed by recalculating the consensus. Those prunings that make the polytomy more resolved can be displayed (or saved). This, unlike RogueNaRok, does not use an optimality criterion over all the tree. In part, the approach in TNT was chosen because actual taxonomic practice requires considering aspects other than sheer number of groups or sums of group frequencies, and it is not obvious how – or even whether – all these aspects can be captured in a single optimality criterion. To further complicate matters, as noted by the online help of RogueNaRok itself, using different cutoffs for the nodes to be optimized (i.e. the –***c*** option) can produce substantially different sets of rogue taxa, thus blurring the benefits of using an optimality criterion (only the user can choose the result of one cutoff over another, and the resulting trees are the only possible basis to make this decision). What is obvious is that, in some contexts, the optimality criterion used by RogueNaRok produces results that are less than ideal, even exploring different cutoffs.

One of the cases in which the RogueNaRok results are unsatisfactory is when a polytomy can be improved by pruning several taxa that belong to a single group, which amounts – from the point of view of the taxonomist – to a single pruning. This is

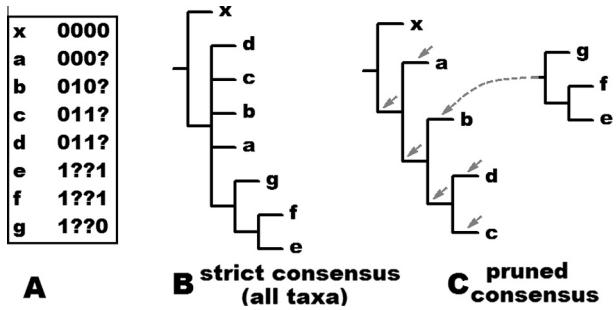

Fig. 7. A matrix which produces 7 most parsimonious trees, for which the number of resolved groups as optimality criterion for deciding taxon prunings finds no possible improvement. The pruning of the group ***efg*** (present in the strict consensus tree) with indication of the branches where this group can join the other subtree, however, gives much more information of the groupings and topology in the input trees.

especially clear in the case of strict consensus trees, but perhaps an analogous criterion could be applied in the case of majority rule consensus trees or frequencies of individual groups. As example, consider the matrix of Fig. 7A, which produces seven equally parsimonious trees and the strict consensus of Fig. 7B. Taxa **e**, **f** and **g** form a monophyletic group in all the optimal trees, but this group can move around the skeleton formed by the other taxa (Fig. 7C shows the possible locations of the group **efg**, with the TNT commands "***pcrprune*** />0; ***nelsen*** / / {0};"). Pruning only one terminal taxon at a time, it is not possible to improve the consensus shown in Fig. 7B – the three must be removed at the same time. Since RogueNaRok seeks a tree where the support/number of resolved groups over all the tree is maximum, the two additional nodes gained when pruning the three taxa together are tied with the two nodes lost when deleting **e**, **f** and **g** (**efg** itself, plus the group **ef**), and RogueNaRok does not find any way to improve the polytomy in Fig. 7B. These results are obtained even if using a cutoff frequency of 100, and allowing three taxa to be pruned simultaneously at each step, with the flags –***s*** 3 –***c*** 100. When using a lower cutoff (e.g. with the default –***c*** 50), the situation is worsened and the results of RogueNaRok fail to indicate any floating group even when the group is much smaller, relative to the region of the tree where the group can move. Aberer (2011:15) does discuss the possibility that two taxa may have different possible locations in the trees but always moving together as sister taxa; however, the optimality criterion used by RogueNaRok does not properly take this into account for examples like Fig. 7. It is not that the RogueNaRok result is "incorrect" – the problem is that the optimality criterion maximized by RogueNaRok is not too useful in this case. In contrast, the TNT commands ***prunnelsen*** and ***pcrprune*** resolve polytomies in the consensus one at a time, by considering the relationships of the terminals or groups connected to the node in question. Thus, for a particular polytomy, TNT considers the pruning of a group connected to the node examined as a single prune – and then it reports that the consensus can be improved by pruning the group **efg**, producing a summary as in Fig. 7C, more useful than the RogueNaRok result. Aberer et al. (2013:162) state that they "have already demonstrated" that their methods "consistently identify rogues with a more harmful effect on consensus tree support than rogues identified by triplet frequency"; they provide no citation for the demonstration, but probably refer to the thesis of Aberer (2011). However, the thesis of Aberer (2011) does not consider IterPCR (Pol and Escapa,

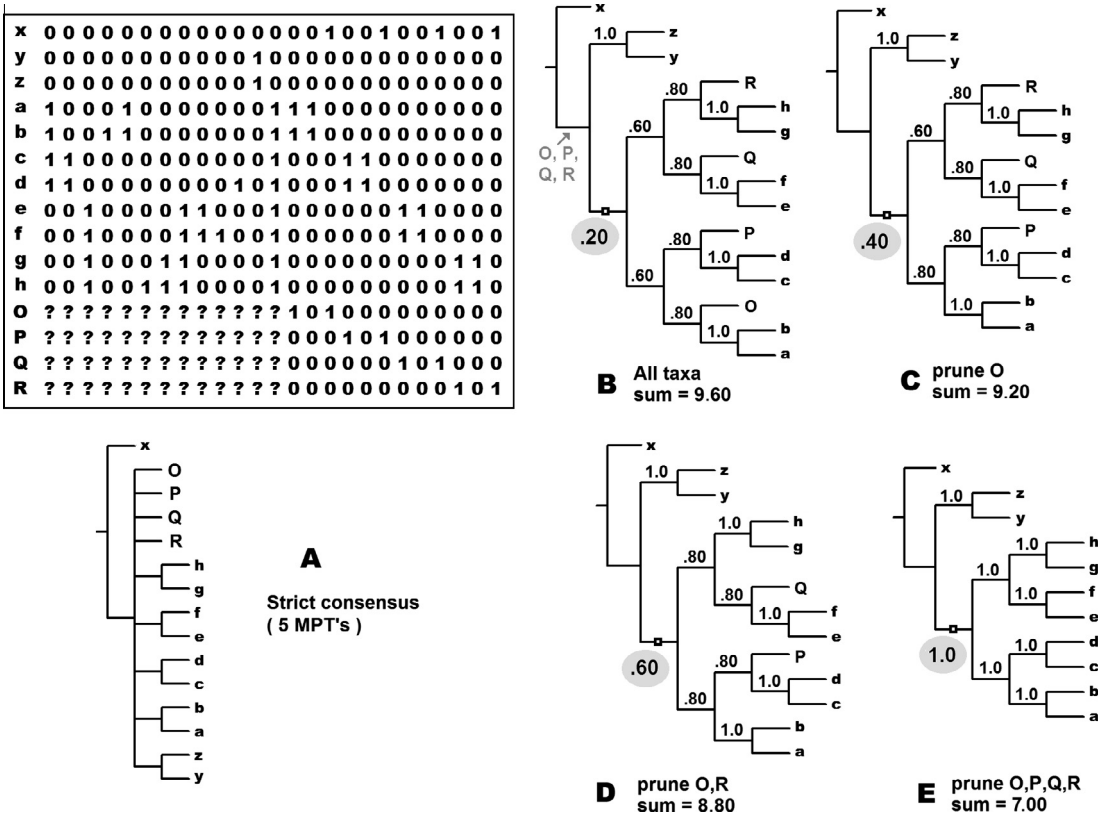

Fig. 8. A matrix which produces 5 most parsimonious trees, with 4 taxa (O–R) of ambiguous placement, which can each be placed either at the base of the tree or as sister to 4 different groups. A, Strict consensus for all taxa. B, the most parsimonious tree having none of the taxa of ambiguous placement at the base; C–E, consensus and group frequencies for different subsets of the taxa of ambiguous placement removed from the trees; in every case, sum of group frequencies is decreased, but the frequency of the highlighted group increases. See text for additional discussion.

2009) or any of the other methods in TNT described by Goloboff et al. (2008). Aberer (2011) only evaluates LSI, only for terminal taxa, only in the unrooted $O(t^4)$ version of RogueNaRok, and only relative to the sum of frequencies for all the groups in the tree. As observed by Wilkinson (2006) and Pol and Escapa (2009:517), triplet measures like LSI or PC when calculated only for terminal taxa can be misleading. For Fig. 7, taxon **b** has the lowest LSI, and taxa **efg** the highest, when all terminal taxa are considered (because **efg** covary); the situation is reversed when **efg** is counted as a single unit, case in which that unit has the lowest value of LSI.

Finally, the pruning of some taxa may decrease the sum of supports over all tree groups, but increase the support for a specific group. If the group in question is the target of the analysis, the supports calculated without those taxa may be more useful, despite the decreased overall supports. Consider the matrix in Fig. 8; it produces 5 equally parsimonious trees and the strict consensus shown in Fig. 8A. The 5 trees result from different placements of 4 taxa, O–R; each of this taxa can be placed (one at a time) at the base of the tree (as indicated in Fig. 8B with gray arrows), or as sister to 4 different groups in the other 4 trees; the fifth tree has none of these rogues at the base, and is displayed in Fig. 8B (with group frequencies shown on the branches). Assume that the taxonomist is interested in resolving the middle part of the tree; when all the taxa are considered, this group has a frequency of 0.20. Pruning O (Fig. 8C) increases the support of the group from 0.20 to 0.40, but it *decreases* the sum of supports from 9.60 to 9.20. Since the sum of supports is decreased, the criterion used by RogueNaRok does not consider the pruning of O to be an "improvement" even when it actually is. The same happens for pruning additional taxa, P–R, which continue increasing the support of the group and decreasing the total sum of supports (Figs. 8D–E). RogueNaRok may take the groups in a reference tree (with the −***t*** option), and report the prunings needed to improve the sum of supports for that tree; one might thus think that giving RogueNaRok a tree with a single group – the group of interest – would allow recognizing O–R as the taxa responsible for the low support. However, this is not the case: the reference tree given to RogueNaRok apparently cannot have polytomies (any attempt to give RogueNaRok a less than fully resolved reference tree triggered error messages or crashes). In contrast, IterPCR immediately recognizes O–R as the problematic taxa, and so do the TNT commands ***chkmoves***, ***prunmajor*** (when using the full tree of Fig. 8B as reference), and ***prunnelsen***. If each character is represented by several copies, then these same results are mimicked by the trees resulting from bootstrapping or jackknifing.

Thus, there are situations in which IterPCR or other methods in TNT will produce more useful results than RogueNaRok. There is little doubt, however, that the methods in TNT can be improved as well (e.g. the algorithms only deal with rooted trees, but the lack of resolution within a group may correspond to different rootings of the subclade). Some of the methods used in TNT resemble (although they are not identical to) maximum agreement subtrees or agreement forests, and perhaps some of those methods could be combined with the approach described here. And, of course, there are many cases where the criterion of maximizing global group frequencies produces more useful results than the methods in TNT. One of the strengths Aberer et al. (2013) demonstrated for RogueNaRok is that it can be used on trees with hundreds of thousands of taxa; the algorithms of TNT cannot reach those limits (neither in terms of memory, or speed), but still can easily handle trees with a few thousand taxa.

In conclusion, it seems clear that although RogueNaRok is a big step forward in methods for identification of rogues in very large trees, it is still far from being the ultimate solution, and that more flexible methods are needed. The very multiplicity of possible contexts in which phylogenetic trees are to be summarized, and the fact that users may be more interested in displaying some aspect of the input trees than others, make it unlikely that the most useful summary of a set of trees will be always be achievable by using only one global optimality criterion and a simple command-line program – a more interactive implementation, allowing users to take active part in exploring different representations of the relationships embodied in the trees, will probably be necessary.

Acknowledgments

We thank comments from Salvador Arias and Santiago Catalano. We also appreciated support from CONICET (PIP 112 201101 00687), and a grant (to J. Cracraft and L. Lohman, entitled "Assembly and evolution of the Amazonian biota and its environment: an integrated approach", from NSF, NASA, and Fundaçao de Amparo à Pesquisa do Estado de São Paulo). Two reviewers (Mark Wilkinson, and Anonymous) provided very helpful criticisms of the manuscript.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.ympev.2015.04.003.

References

Aberer, A., 2011. Advanced Methods for Phylogenetic Post-Analysis. Master's Thesis. Technische Universität München/Heidelberg Institute for Theoretical Studies, Germany, January 2011, 77 pp.

Aberer, A., Stamatakis, A., 2011. A Simple and Accurate Method for Rogue Taxon Identification. IEEE BIBM 2011, Atlanta, Georgia, USA, November 2011.

Aberer, A., Krompas, D., Stamatakis, A., 2013. Pruning rogue taxa improves phylogenetic accuracy: an efficient algorithm and Webservice. Syst. Biol. 62, 162–166.

Anderson, J., 2001. The phylogenetic trunk: maximal inclusion of taxa with missing data in an analysis of the Lepospondyli (Vertebrata, Tetrapoda). Syst. Biol. 50, 170–193.

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.

De Laet, J., Smets, E., 1988. On the three-taxon approach to parsimony analysis. Cladistics 14, 363–381.

Deepak, A., Dong, J., Fernández-Baca, D., 2012. Identifying rogue taxa through reduced consensus: NP-Hardness and exact algorithms. In: ISBRA'12 Proceedings of the 8th International Conference on Bioinformatics Research and Applications. Springer-Verlag, Heidelberg, pp. 87–89.

Dekker, M.C., 1986. Reconstruction Methods for Derivation Trees. Thesis submitted to the Department of Mathematics and Computer Science, Vrije Universiteit, Amsterdam.

Estabrook, G.F., 1992. Evaluating undirected positional congruence of individual taxa between two estimates of the phylogenetic tree for a group of taxa. Syst. Biol. 41, 172–177.

Goloboff, P., 1995. A revision of the South American spiders of the family Nemesiidae (Araneae, Mygalomorphae). Part I: species from Peru, Chile, Argentina, and Uruguay. Bull. Am. Mus. Nat. Hist. 224, 1–189.

Goloboff, P., Catalano, S., 2012. GB-to-TNT: facilitating creation of matrices from GenBank and diagnosis of results in TNT. Cladistics 28, 503–513.

Goloboff, P., Farris, J., 2001. Methods for quick consensus estimation. Cladistics 17, S26–S34.

Goloboff, P., Farris, J., Nixon, K., 2008. TNT, a free program for phylogenetic analysis. Cladistics 24, 774–786.

Goloboff, P., Farris, J., Källersjö, M., Oxelmann, B., Ramírez, M., Szumik, C., 2003. Improvements to resampling measures of group support. Cladistics 19, 324–332.

Goloboff, P., Catalano, S., Mirande, M., Szumik, C., Arias, J.S., Källersjö, M., Farris, J.S., 2009. Phylogenetic analysis of 73,060 taxa supports major evolutionary groups. Cladistics 25, 211–230.

Gusfield, D., 1997. Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, 534 pp.

Kearney, M., Clark, J., 2003. Problems due to missing data in phylogenetic analyses including fossils: a critical review. J. Vertebr. Paleontol. 23, 263–274.

Lahr, D., Grant, J., Nguyen, T., Lin, J.-H., Katz, L., 2011. Comprehensive phylogenetic reconstruction of amoebozoa based on concatenated analyses of SSU-rDNA and actin genes. PLoS ONE 6 (7), e22780. http://dx.doi.org/10.1371/journal.pone.0022780.

Lin, H.T., Gordon Burleigh, J., Eulenstein, O., 2009. Triplet supertree heuristics for the tree of life. BMC Bioinformatics 10 (Suppl. 1), S8. http://dx.doi.org/10.1186/1471-2105-10-S1-S8.

Mosses, C., 2005. Triplet Supertrees. Thesis presented to the Department of Computer Science, University of Aarhus, Denmark, 6 June 2005, 94 pp.

Nosenko, T., Schreiber, F., Adamska, M., Adamski, M., Eitel, M., Hammel, J., Maldonado, M., Müller, W.E.G., Nickel, M., Schierwater, B., Vacelet, J., Wiens, M., Wörheide, G., 2013. Deep metazoan phylogeny: when different genes tell different stories. Mol. Phylogenet. Evol. 67 (2013), 223–233.

Pattengale, N., Aberer, A., Swenson, K., Stamatakis, A., Moret, B., 2011. Uncovering hidden phylogenetic consensus in large datasets. IEEE/ACM Trans. Comput. Biol. Bioinform. (TCBB) 8, 902–911.

Pol, D., Escapa, I.H., 2009. Unstable taxa in cladistic analysis: identification and the assessment of relevant characters. Cladistics 25, 515–527.

Ranwez, V., Criscuolo, A., Douzery, E.J.P., 2010. SuperTriplets: a triplet-based supertree approach to phylogenomics. Bioinformatics 26, 115–123.

Sand, A., Holt, M.K., Johansen, J., Fagerberg, R., Brodal, G.S., Pedersen, C.N., Mailund, T., 2013. Algorithms for computing the triplet and quartet distances for binary and general trees. Biology – Spec. Iss. Dev. Bioinform. Algorithms 2 (4), 1189–1209.

Schieber, B., Vishkin, U., 1988. On finding lowest common ancestors: simplifications and parallelizations. SIAM J. Comput. 17, 1253–1262.

Siu-Ting, K., Pisani, D., Creevey, C.J., Wilkinson, M., 2015. Concatabominations: identifying unstable taxa in morphological phylogenetics using a heuristic extension to safe taxonomic reduction. Syst. Biol. 64, 137–143.

Smith, S.A., Dunn, C., 2008. Phyutility: a phyloinformatics utility for trees, alignments, and molecular data. Bioinformatics 24, 715–716.

Stevenson, D.Wm., 2004. Abstracts of the 22nd annual meeting of the Willi Hennig Society. Cladistics 20, 76–100.

Struck, T.H., Paul, C., Hill, N., Hartmann, S., Hösel, C., Kube, M., Lieb, B., Meyer, A., Tiedemann, R., Purschke, G., Bleidorn, C., 2011. Phylogenomic analyses unravel annelid evolution. Nature 471, 95–98. http://dx.doi.org/10.1038/nature09864.

Thorley, J., Page, R.M., 2000. RadCon: phylogenetic tree comparison and consensus. Bioinformatics 16, 486–487.

Thorley, J., Wilkinson, M., 1999. Testing the phylogenetic stability of early tetrapods. J. Theoret. Biol. 200, 343–344.

Wilkinson, M., 1994. Common cladistic information and its consensus representation: reduced Adams and cladistic consensus trees and profiles. Syst. Biol. 43, 343–368.

Wilkinson, M., 1995. Coping with abundant missing entries in phylogenetic inference using parsimony. Syst. Biol. 44, 501–514.

Wilkinson, M., 1996. Majority rule reduced consensus and their use in bootstrapping. Mol. Biol. Evol. 13, 437–444.

Wilkinson, M., 2006. Identifying stable reference taxa for phylogenetic nomenclature. Zoolog. Scr. 35, 109–112.