WILEY-BLACKWELL

Cladistics 24 (2008) 1–13

Cladistics

10.1111/j.1096-0031.2008.00217.x

TNT, a free program for phylogenetic analysis

Pablo A. Goloboff[a,*], James S. Farris[b] and Kevin C. Nixon[c]

[a]CONICET, INSUE, Instituto Miguel Lillo, Miguel Lillo 205, 4000 S.M. de Tucumán, Argentina;
[b]Molekylärsystematiska laboratoriet, Naturhistoriska riksmuseet, Box 50007, S-104 Stockholm, Sweden;
[c]L.H. Bailey Hortorium, Cornell University, Ithaca, NY 14853-4301, USA

Accepted 6 January 2008

Abstract

The main features of the phylogeny program TNT are discussed. Windows versions have a menu interface, while Macintosh and Linux versions are command-driven. The program can analyze data sets with discrete (additive, non-additive, step-matrix) as well as continuous characters (evaluated with Farris optimization). Effective analysis of large data sets can be carried out in reasonable times, and a number of methods to help identifying wildcard taxa in the case of ambiguous data sets are implemented. A variety of methods for diagnosing trees and exploring character evolution is available in TNT, and publication-quality tree-diagrams can be saved as metafiles. Through the use of a number of native commands and a simple but powerful scripting language, TNT allows the user an enormous flexibility in phylogenetic analyses or simulations.

Introduction

Since the first breakthrough in parsimony analysis with the release of Hennig86 (Farris, 1988), parsimony programs have continued to improve, culminating in TNT (Goloboff et al., 2003b), which includes several new methods to facilitate phylogenetic analysis (for reviews see Hovenkamp, 2004; Giribet, 2005; Meier and Ali, 2005). Under an agreement between the Willi Hennig Society and the authors, TNT is now available as a free program. A version of TNT licensed for single-user, academic use can be downloaded from http://www.zmuc.dk/public/phylogeny/TNT. The purpose of the present paper is to provide a general overview of the program and some guidance for beginners.

General interface: menus or commands

TNT is a fully interactive program. All versions (Windows, Linux and Macintosh) can be run by means of commands. In addition, the Windows version has a sophisticated menu interface or GUI (Figs 1–5), which makes TNT the only major phylogeny program that can be menu-driven under Windows. Using emulators or alternative implementations of the Windows APIs (such as WINE or CrossOver Mac), the Windows version of TNT can also be run under Linux or OSX. Note that the menu-driven PAUP* (Swofford, 2002) is a Classic Macintosh program, and as such it is no longer supported on the latest version of OSX or any Intel Macintosh.

Commands can also be read from data files or instruction files, providing an easy way to automate routines. The on-line help (***help*** command) includes a list of all the TNT commands and their options. Throughout, command options (available in all versions) are indicated as italics and bold (e.g. ***procedure***) and menu options (available only in Windows versions) are indicated as bold (e.g. **File/OpenInputFile**). To save space, only the most important and common options are indicated.

A basic analysis

The input data file can be in either TNT or basic Nexus format. The TNT format is derived from

*Corresponding author.
E-mail address: pablogolo@csnat.unt.edu.ar

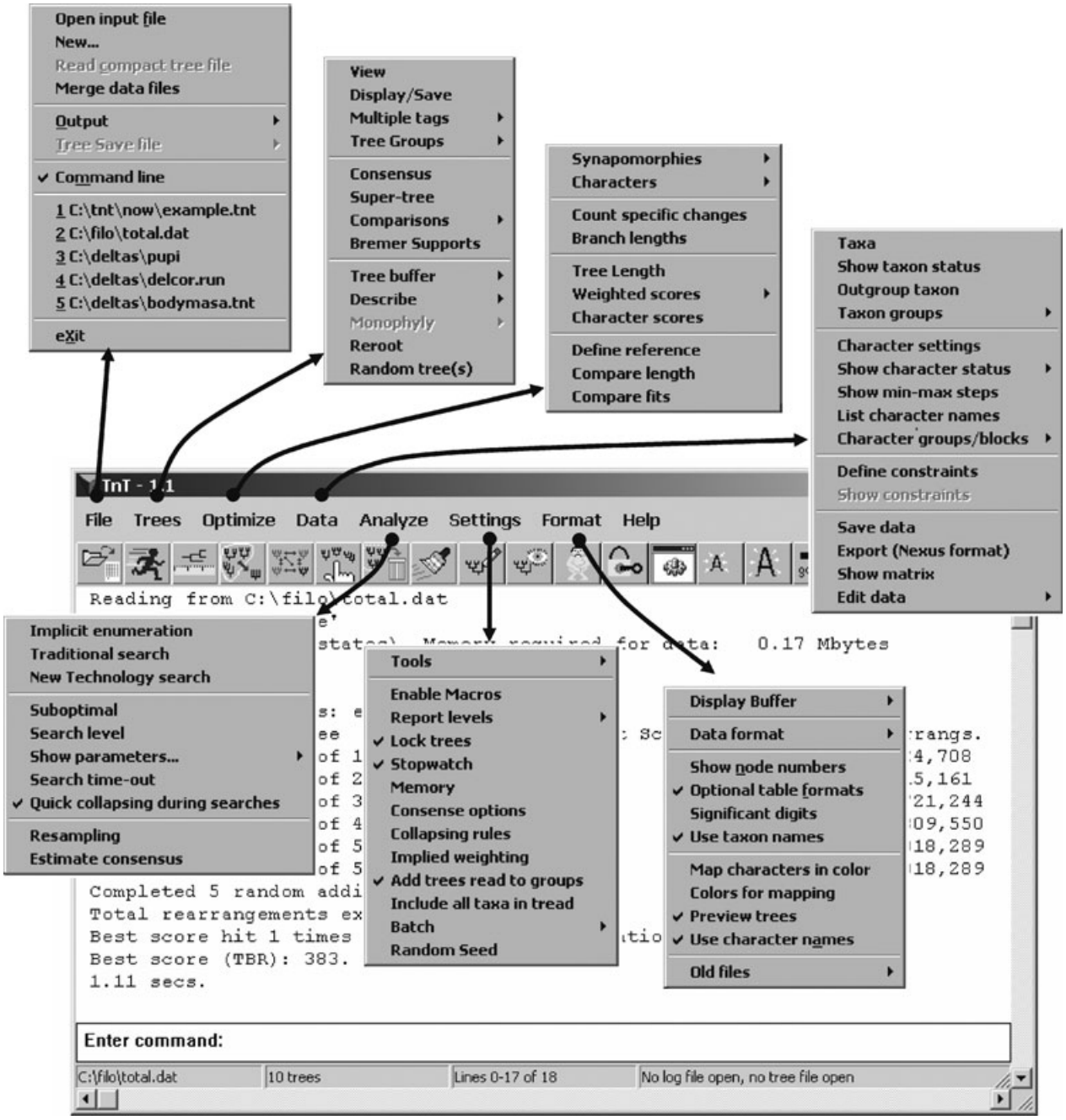
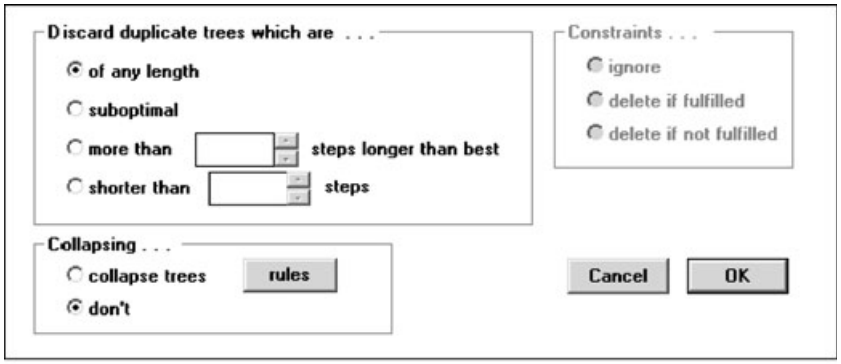


Fig. 1. Main menu options of TNT.

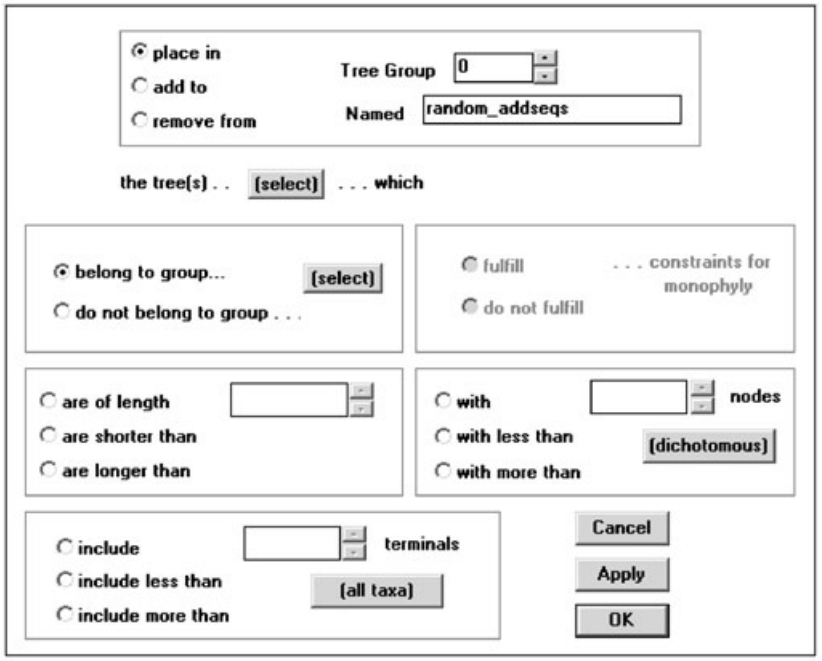
Hennig86/NONA with significant enhancements making it relatively backwards compatible (requiring at the most a little editing to eliminate commands extraneous to TNT). The data file should start with a specification of the data type, which in the case of DNA data is ***nstates dna;***. Next comes the ***xread*** command, followed by the number of characters, the number of taxa, and the data themselves (sequence data must be prealigned). Character states may be IUPAC codes, digits (for morphological characters), ? (for missing data), or - (for gaps).

To read in the data file, select **File/OpenInputFile** or type in ***procedure datafilename***. Usually you will name your data file with a ***.tnt*** extension, such as ***mydata.tnt***.

Before analyzing the data, you should make provision for saving the output to a log file. This can be done by selecting **File/Output/OpenOutput** or by entering ***log logfilename***. Usually you should give a log file the ***.out***



A Trees / Tree Buffer / Filter



B Trees / Tree Groups / Define

Fig. 2. Dialogs for handling internal tree file: A, filtering, and B, creating groups of trees.

extension and the same base name as the data file—something like ***mydata.out***.

All program output is temporarily saved to an internal text-buffer. This text-buffer can be saved to a newly opened log file by selecting **File/Output/SaveDisplayBuffer**. In command-driven versions, the text-buffer can be inspected with the ***view*** command; in the Windows version, the default window displays the text-buffer automatically.

A basic analysis consists of using multiple addition sequences followed by branch swapping. To do this select **Analyze/TraditionalSearch** with default options (just click **Search** when the option dialog appears) or enter ***mult***. This will perform 10 random addition sequences followed by branch-swapping, saving up to 10 trees per replication (roughly equivalent to a "heuristic search" with random addition sequences in PAUP*, or ***hold/10;mult*10;*** in NONA). In the case of small data sets (15–30 taxa), exact solutions using branch-and-bound (which guarantee finding all trees optimal under current settings) can be produced within reasonable times with **Analyze/ImplicitEnumeration** or the ***ienum*** command.

Once calculated, trees may be viewed by selecting **Tree/Display** or by entering ***tplot***. In the Windows version, the tree can be saved as an extended metafile (a publication-quality image file which can be exported to PowerPoint, CorelDraw, etc.), by pressing "M" while viewing the tree-diagram (see below). To save the trees for later reanalysis, create a save file by selecting **File/**

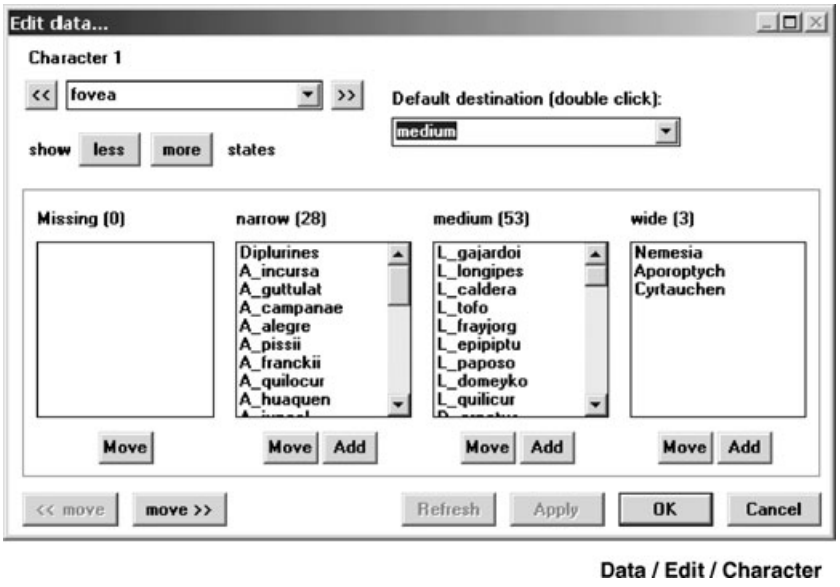


Fig. 3. Dialog for editing data on a character-by-character basis.

TreeSaveFile/Open or by entering ***tsave * savefilename***. Usually you will name your save file with a ***.tre*** extension, so that its name will be something like ***mydata.tre***. If there are multiple trees, their consensus can be found by selecting **Trees/Consensus** or by entering ***nelsen***.

The synapomorphies common to several trees can be plotted on their consensus by selecting **Optimize/Synapomorphies/MapCommon** or by entering ***apo [*** .

Bremer supports can be calculated by using a script (a program written in the scripting language, as explained later) contained in a file called bremer.run, with instructions for TNT to calculate Bremer supports using either searches for suboptimal trees, constraints for non-monophyly, or combinations of both methods. To run this script select **File/OpenInputFile** or enter ***procedure bremer.run***. Resampling (jackknifing, bootstrapping, etc.) can be done by selecting **Analyze/Resampling** or by entering ***resample***.

Quantitative and morphological characters, and data editing

In addition to DNA sequences, data can also consist of morphological characters with up to 32 states (alphanumerical codes), continuous characters (values from 0 to 65, with three decimals), or protein sequences (IUPAC codes), possibly combined (each one must be placed in a different block of data, preceded by indication of the corresponding format). Despite the fact that continuous characters are so common in morphological data sets, all other programs for phylogenetic analysis require that continuous characters be forced into characters with discrete states; TNT instead optimizes continuous characters as such (Goloboff et al., 2006).

In Windows versions, it is possible to edit the data (selecting **Data/Edit**, either on a character-by-character basis, Fig. 3, or on a taxon-by-taxon basis); if characters/states have been named, this facilitates inspecting and proof-checking the data, without the need to look at an alphanumeric matrix. Data editing in command-driven versions is limited, and can only be done one cell at a time.

Groups of taxa, trees, and characters

In commands that require selections of trees, taxa or characters, it is possible to specify them one by one, or by referring to tree, taxon, or character groups (Fig. 2B shows the dialog for defining tree groups), which can be defined by means of ***tgroup*** (for trees), ***agroup*** (for taxa) and ***xgroup*** (for characters). Taxa, characters, and trees are numbered starting from 0, so that for N elements, the last is numbered $N - 1$. When a command expects a list, enclosing the name (possibly truncated) or number of a group in curly braces { } is equivalent to listing all the members of the group. Commands that generate, modify or read trees from files automatically place the trees in tree groups, which makes subsequent manipulations of sets of trees easier. For example, the "***nelsen *;***" option calculates and saves a strict consensus tree to memory (automatically placing it in a group called "***strict_consensus***"), and the ***tnodes*** command counts the number of nodes in subsequently listed trees, so that "***nelsen *; tnodes {strict };***" counts the number of nodes in the strict consensus.

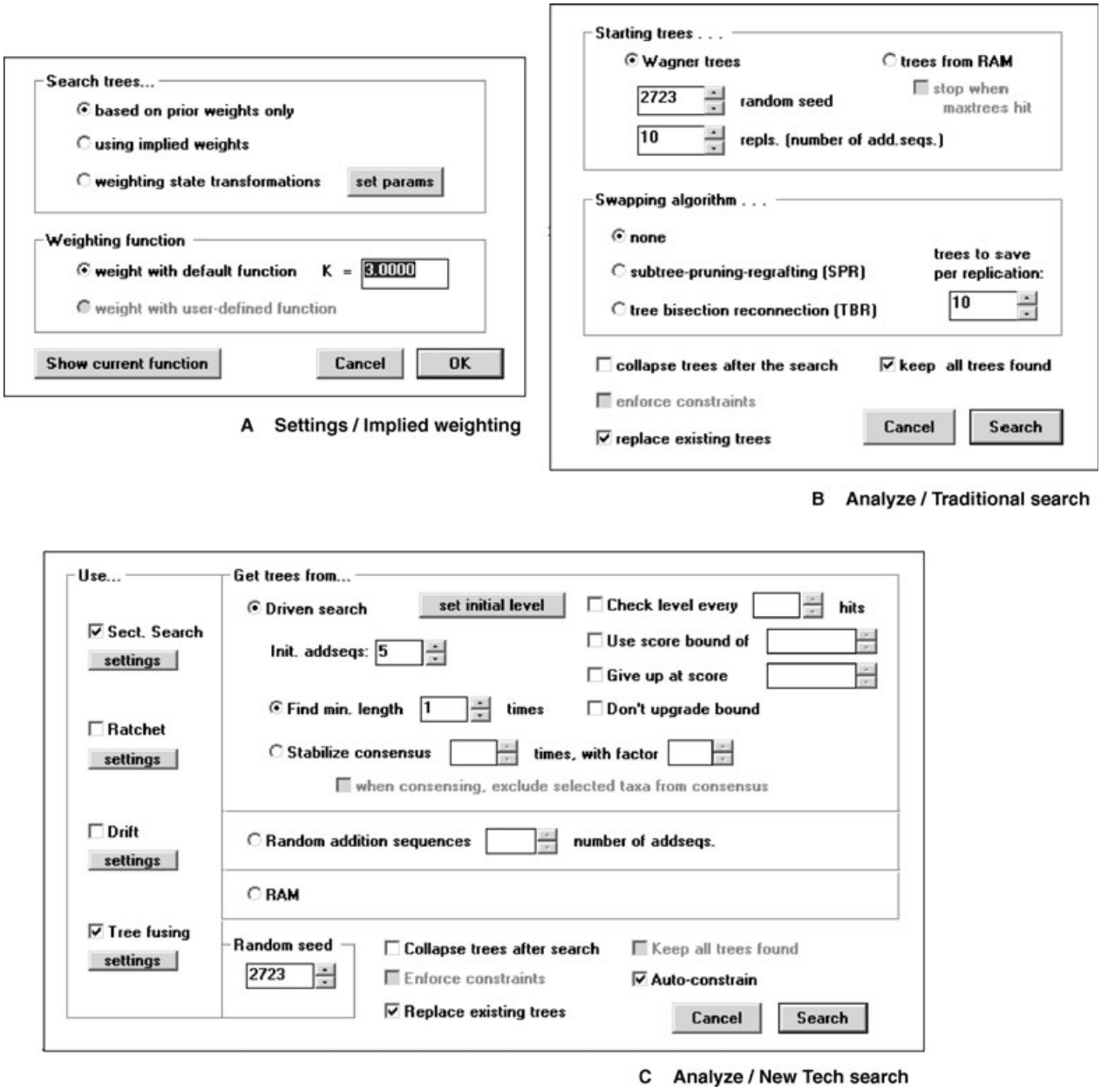


Fig. 4. Dialogs for setting optimality criteria (A), branch-swapping and random addition sequences (B), and new technology searches (C).

It is also possible to specify groups of taxa by referring to specific nodes of a tree (i.e. ***@T N*** refers to all of the taxa descended from node N of tree T), and trees can be edited with the ***edit*** command.

Tree viewing and editing: producing publication-quality output

In Windows versions, commands or menu options that output tree diagrams display the trees in a separate window to "pre-view" the trees. From the previewing window the user can decide whether the tree diagram is to be discarded, written to the text buffer or log file, or saved as a metafile (i.e., a publication-quality image file). If a metafile is opened before displaying the tree (with **File/Output/OpenMeta file**), the tree diagram goes there automatically. The previewing window also allows mapping characters in color, or defining specific legends or colors for tree branches, by a double right-click on a node, when the tree is unlocked.

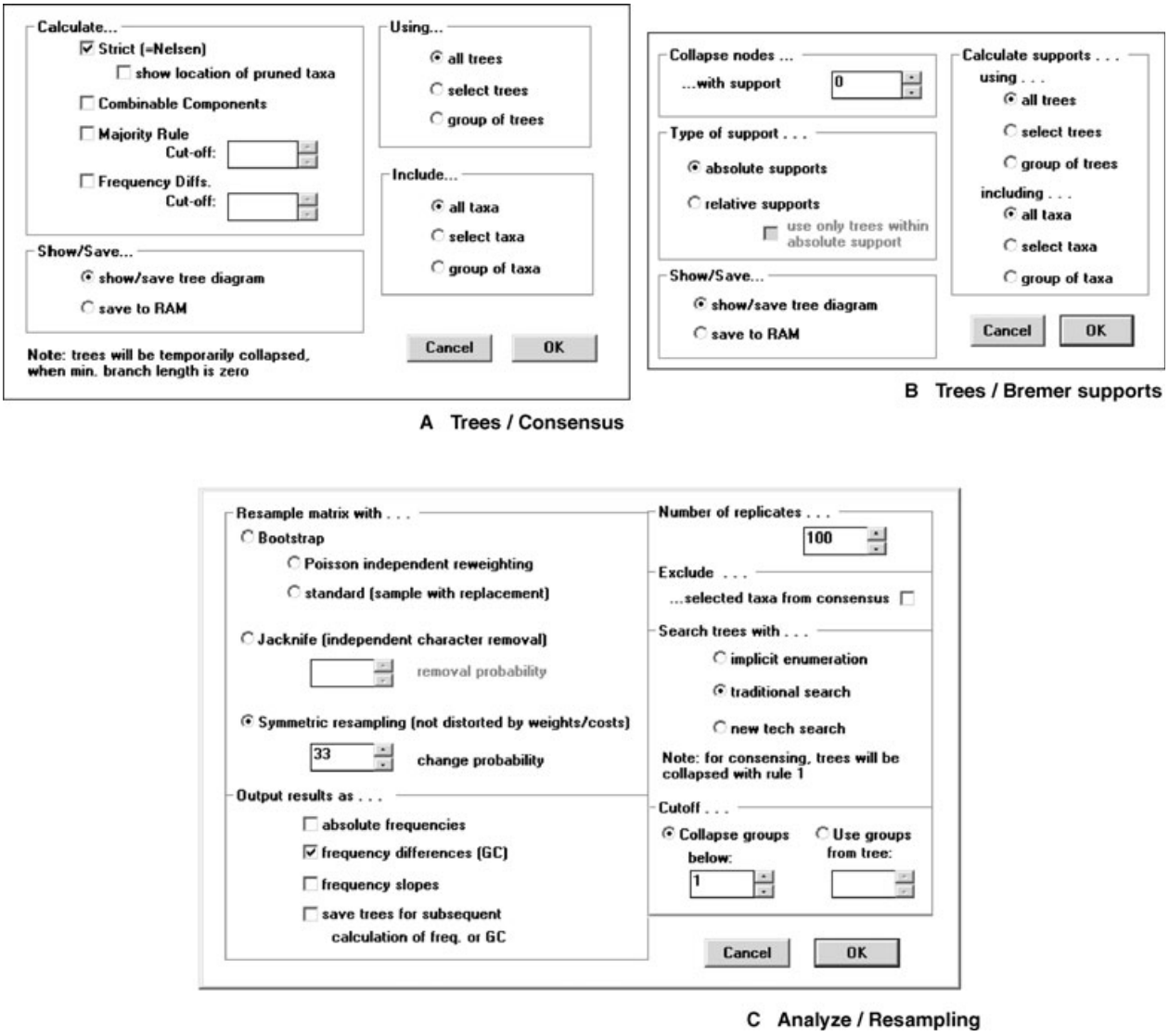


Fig. 5. Dialogs for calculating consensus trees (A), Bremer supports (B), and resampling (C).

Windows versions also allow the graphical editing and selecting of taxa. Selecting **Trees/View** switches to tree-viewing mode. Under tree-viewing mode, a left-click on the node of a tree will select (in red) or deselect (in green) all the taxa descended from the node, so that the dialog for taxon selection (under any menu option which can select some taxa) will initially display that selection. In tree-viewing mode trees may be locked or unlocked. This is controlled by the padlock toggle switch in the toolbar or by selecting **Settings/LockTrees**. If the tree is locked, right-clicking on a node shows a list of synapomorphies for the node (if characters/states have been named, and **Format/UseCharacterNames** is selected, it then uses character names). If the tree is unlocked, it can be edited using the mouse.

Optimality criteria and character types

TNT implements different criteria for parsimony analysis (Fig. 4A). Analyses can be carried out either using equal weights (default), weights predefined by the user, implied weights (Goloboff, 1993; either with standard or with user-defined functions of the homoplasy), or self-weighted optimization (i.e. dynamic weighting of state transformations, using the method of Goloboff, 1997). The scripting language can be used to search iteratively (as in successive weighting, Farris, 1969; dynamic weighting, Williams and Fitch, 1990; or support weighting, Farris, 2001), reassigning weights to either characters or state transformations, in specific ways determined by the user.

If characters or state-transformations have been assigned differential weights, then these weights or costs are considered during implied weighting or self-weighted optimization. TNT can optimize additive (Farris, 1970) and nonadditive (Fitch, 1971) characters (''ordered'' and ''unordered'' of PAUP*), as well as Sankoff (step-matrix; Sankoff and Rousseau, 1975) characters with any metric costs. Character state-trees can easily be defined in the form of a diagram using ASCII characters (with **Data/CharacterSettings**, selecting ''Character-state-tree'', or with the ***cstree*** command); thus in

```
0 – 1 – 2
     \
5 – 3 – 4
```

the cost of changing between two states is simply the number of lines on the shortest path between them.

Searching for most parsimonious trees: basic methods

In addition to the basic approach described earlier (Fig. 4B), branch-swapping can be applied to any starting tree (with the ***bbreak*** command, or selecting as starting trees ''trees from RAM'' instead of ''wagner trees''). For most medium-sized data sets, the best approach is to run a number of random addition sequences (RAS), each with TBR[1] branch-swapping, until the best score is hit 10 to 20 times independently. This is usually sufficient for all global optima (''islands'', as they were once called) to be found. Note here that if a search is repeated several times, the same random seed will be used unless changed explicitly, either with the ***rseed*** command or from the same dialog box in Windows versions. When the program reports that ''some replications overflowed'' and the goal is to find all most parsimonious trees for the data set at hand, subsequent branch-swapping from the trees produced by RAS + TBR can be used after setting the maximum trees to save to a large number, with **Settings/Memory** or with the ***hold*** command. For data sets that produce very large numbers of equally parsimonious trees, saving all of them is (as noted by Farris et al., 1996) very impractical; in those cases, similar results can be obtained if best scores are found independently a significant number of times, and the results are then strict-consensed.

The branch-swapping algorithms of TNT are very efficient. For medium-sized data sets, RAS + TBR searches typically proceed 5 to 10 times faster than in PAUP* or Nona/Pee-Wee (Goloboff, 1994a,b). For large data sets, the difference in speed is often much larger (e.g. Goloboff and Pol, 2007, for two data sets with *c*. 11 000 and 14 000 taxa, report speed differences in TBR of 300 and 900 times, respectively).

[1]''Tree bisection reconnection'' is a synonym of ''branch-breaking'' (Farris, 1988; see Goloboff, 1999, footnote 1), the swapping method of Hennig86, from which the name of the TNT command, ***bbreak***, is derived.

A stricter collapsing of zero-length branches improves tree-searches

Searches optionally collapse zero-length branches under different criteria or retain all distinct dichotomous trees. The default collapsing rule in TNT is to eliminate any branch for which the minimum possible length (among alternative most parsimonious reconstructions) is zero. As discussed by Goloboff (1996) and Davis et al. (2004), this criterion produces more effective searches than criteria which collapse fewer branches, both in terms of time needed to complete searches, and ability to find shortest trees when doing multiple RAS + TBR saving limited numbers of trees per replication. Under this criterion, the polytomized trees may become longer (see Coddington and Scharff, 1994). Unless explicitly asked to polytomize the trees after a search (either by ticking on the corresponding option, or with the ***collapse auto*** option), TNT will retain the trees as dichotomous, so that re-optimizing them will produce minimum length. Note that even when TNT does not collapse the trees, the program makes sure that all the trees saved would be different *if* they were collapsed. When trees are (by default) retained as binary, consensus calculation re-optimizes the trees, temporarily eliminating zero-length branches. If the trees are collapsed after the search, the consensus calculation should avoid re-collapsing the trees (see below, under Tree Comparisons).

Tree-searches in large and complex data sets: the new technology options

For large and very complex data sets, TNT also implements several algorithms that are much more effective than simple branch-swapping (Fig. 4C). Most of these algorithms were introduced in TNT, and make TNT the only program capable of reliably analyzing data sets with more than a few hundred taxa. Using these new algorithms TNT may require only a hundredth or a thousandth of the time needed by PAUP* to find trees of minimum length. A recent example is Goloboff et al.'s (submitted) reanalysis of McMahon and Sanderson's (2006) 2228-taxon data set: trees of the best length ever found with the ratchet under PAUP* in 1700 h of CPU-time were found by TNT, on average, in 30 min.

The ratchet (Nixon, 1999) and tree-drifting (Goloboff, 1999) use a cyclic scheme of perturbation and search phases. Tree-fusing (Goloboff, 1999) evaluates sub-tree

exchanges between trees, effecting those which improve score. Sectorial searches (Goloboff, 1999) create reduced data sets (using down-pass state-sets for each node), and subject the reduced data set to a search algorithm (in TNT, any specific search command, including further subdivision in sectors, can be used). For extremely large data sets, sectorial searches are the most effective means for quickly finding near-optimal trees (see Goloboff and Pol, 2007). These algorithms can be applied to preexisting trees (with the commands ***ratchet***, ***drift***, ***sectsch*** and ***tfuse***, or selecting **Analyze/NewTechSearch** and "RAM" for "Get trees from", see Fig. 4C), or to trees created de novo with multiple RAS (***xmult*** command, or selecting "Driven search" or "Random addition sequences"). The "driven search" continues until the specified tree-score (or the best score found during the search, if no target score was specified) is hit a given number of times, or until the consensus (re-calculated as new hits to the best score are produced) becomes stable. The latter provides a way to produce accurate consensus trees (especially if the trees are collapsed more strictly, see below) without saving all possible equally parsimonious trees. Note that the consensus stabilization will produce more reliable results, or with fewer hits to minimum length, when the trees are collapsed more strictly (the best choice is TBR-collapsing; see Goloboff, 1999).

In the case of impossibly large data sets (or easier data sets but very impatient users), a conservative estimate of the actual consensus of most parsimonious trees that does not require actually finding them (i.e. a quick consensus estimation; Goloboff and Farris, 2001) can be produced by selecting **Analyze/EstimateConsensus** or with the ***qnelsen*** command. In addition to the built-in routines for tree-searches, the scripting language of TNT allows users to devise their own search strategies.

Constrained searches, suboptimal trees

Tree searches can be performed under either positive or negative constraints, so that only trees either having, or not having, certain specified groups are acceptable. This can be useful for calculating Bremer supports. The Windows version has a uniquely simple way to define constraints, by just clicking on tree nodes, with **Data/DefineConstraints**. In command-driven versions, constraints can be defined by reference to trees or taxon groups in the ***force*** command. Once defined, constraints must be explicitly applied to searches, either by ticking on the corresponding box, or with the ***constrain*** command. In the case of PAUP*, searches can use either positive or negative constraints, but not both at the same time; TNT can use both positive and negative constraints.

For calculation of Bremer supports, or for other purposes, the program can search for suboptimal trees (based on either fit difference and/or relative fit difference, of Goloboff and Farris, 2001). The maximum acceptable difference in score is set with **Analyze/Suboptimal** or the ***subopt*** command.

Character optimization: diagnosis and mapping

Character mapping and the diagnosis of the trees obtained are one of the main components of a cladistic analysis. TNT can either produce lists of synapomorphies or character changes, or display those results on tree diagrams (with the menu options **Optimize/Synapomorphies** and **Optimize/Characters**, or with the commands ***apo*** and ***map***). When multiple most parsimonious trees are used to calculate consensus trees, the consensus is longer, so that optimizing it for producing synapomorphy lists or studying character evolution is (while possible) inappropriate. In this case TNT allows an optimization of the multiple most parsimonious trees individually, displaying on the consensus tree a summary of the changes that are common to all the trees used to produce the consensus. This is done with the **Common Synapomorphies** or **Common Changes** options, which are also available as options of the ***apo*** and ***map*** commands.

TNT also implements options for counting specific transformations (e.g., are gains more common than losses?), and enumerating all possible most parsimonious reconstructions, in the case of ambiguity. The scripting language can access these options as well as handling individual reconstructions (or finding the best state assignments with a fixed state at one or more nodes, for a given tree; see the documentation for the ***iterrecs*** command).

Finding wildcard taxa, tree comparisons, consensus trees

The options for comparing trees and summarizing results are one of the most important aspects of TNT. The standard methods for consensus (strict, semi-strict or combinable components, majority, and frequency differences) are accessed with **Trees/Consensus** (Fig. 5A), or with the commands ***nelsen***, ***comcomp***, ***majority*** and ***freqdifs***. As noted above, the trees are (by default) temporarily collapsed as the consensus is calculated (needed if the trees are retained as binary after searches, which is the default); whether trees are temporarily collapsed is changed with **Settings/ConsenseOptions**, or the ***collapse notemp*** command. If the trees have different taxon sets, then the default action (changed with **Settings/ConsenseOptions** or the ***unshared*** command) is pruning all the trees so that they have

identical taxon sets. Poorly resolved consensus trees (or some groups with low frequencies, in the case of majority rule or frequency difference trees, often used to measure group supports; see below) may be caused by just one or a few taxa moving among alternative distant positions in the input trees.

TNT implements several options that facilitate identifying the taxa responsible for the lack of resolution, or responsible for the low group supports:

(a) automatic evaluation of alternative taxon prunings, counting the numbers of nodes gained in either the strict or semistrict consensus, and reporting those prunings which improve the results (**Trees/Comparisons/PrunedTrees**, or ***prunnelsen*** and ***pruncom*** commands); since the alternative prunings are tried combinatorially, this option may (depending on the resolution of the consensus trees) become very time-consuming beyond five or six taxa (or clades) eliminated at the same time;

(b) agreement subtrees, which identify the largest subset of taxa which are identically related in all input trees (**Trees/Comparisons/AgreementSubtrees**, or ***prunnelsen***). This is often used also as a measure of similarity between the input trees;

(c) a heuristic command which, given the groups in a reference tree, attempts to identify the taxa to prune to increase group frequencies (this is the method used by Goloboff et al. (submitted), implemented with the ***prunmajor*** command);

(d) TBR-tracking (***chkmoves*** command), which submits a tree to TBR branch-swapping, and records (for each clade) the number of possible positions, maximum distance and depth of rerooting, for all the moves that produce equally optimal trees (or trees within a specified score difference); and

(e) calculation of approximate frequencies of group recovery with the ***rfreqs*** command, which is a sort of majority rule, but calculates a similarity index between partitions (based on the composition of the taxa inside and outside the group). The score is 1 for two identical (=monophyletic) groups, and decreases to the extent that there are more differences. In this way, a group which is "almost" monophyletic (i.e. which could be made monophyletic by ignoring the position of just a few terminals) in most of the trees will receive scores approaching 100%. Groups that have scores near 100% are probably amenable to have their frequencies improved by ignoring the position of few taxa (while groups with very low scores can, probably, be improvable only by pruning large numbers of taxa).

Most of these commands allow identifying the taxa either visually or (perhaps by interacting with the scripting language) by placing them in taxon groups, so that automation of routines to improve consensus trees by using a combination of these basic approaches is possible.

TNT also natively supports semistrict consensus supertrees (Goloboff and Pol, 2002), as well as easy creation of MRP matrices (for subsequent supertree analysis under either parsimony or cliques).

For simple comparisons between tree topologies, TNT allows checking (and reporting) of duplicate trees, checking the groups present in one tree but not in another (a sort of "anticonsensus", either with ***tcomp*** or **Trees/Comparisons/CompareGroups**), and (if constraints have been defined) checking whether each of the groups defined in the constraints is (or is not) monophyletic (***mono***, or **Trees/Monophyly**).

Another important component of the tree-comparison routines is found in the options which calculate different coefficients of tree similarity. The natively implemented options for topological comparison are the retention index (Farris, 1989) of the MRP of one tree mapped onto the other (***tcomp*** command), which is a variation of Farris (1973) "distortion coefficient", and SPR-distances between trees (using the heuristic method of Goloboff (2008), with the **Trees/Comparisons/SPR-Distances**, or ***sprdiff*** command). In conjunction with the scripting language, it is also possible to implement Robinson–Foulds distances (Robinson and Foulds, 1981), triplet dissimilarity, number of steps in the MRP, number of "flippings" (changes to the MRP matrix), etc.

Bootstrapping, jackknifing and Bremer support

For measuring group supports, TNT implements both Bremer supports and measures based on resampling (jackknifing, bootstrapping or symmetric resampling).

For resampling (**Analyze/Resampling**, see Fig. 5C, or ***resample***), the user may define any search routines (or use the instructions in a file or script) to analyze each resampled data set. After analyzing each resampled data set, TNT will automatically compute the strict consensus tree (collapsing for which is done according to the criterion in effect; as with consensus stabilization, the best choice is TBR-collapsing; see Goloboff and Farris, 2001; and Goloboff et al., 2003a). A summary of results is calculated at the end, and it is optionally possible to save the strict consensus for each replication for subsequent manipulations and consensing.

For Bremer supports, finding suboptimal trees where the different groups are non-monophyletic is up to the user. In simple data sets, just using the optimal trees as a starting point for searches, saving successively larger sets of more suboptimal trees (make sure you select "trees from RAM" as starting trees, see Fig. 4B, and tick on "stop when maxtrees hit", or use the ***bbreak = fillonly*** command, so that the suboptimal trees are not needlessly swapped). Increasing the value of suboptimal in several steps is important, because otherwise the values of Bremer support will probably be overestimated—the

search for suboptimal trees is very likely to fill the tree buffer with very suboptimal trees, thus missing most of the slightly suboptimal trees (which are needed to identify the least supported groups). Once the optimal and suboptimal trees are stored in memory, the program checks minimum score differences to lose each group (with **Trees/BremerSupports**, Fig. 5B, or the ***bsupport*** command) and plots them on a tree diagram.

For better supported groups or very noisy data sets, finding a tree not displaying a given group by just accepting suboptimal trees may require saving enormous numbers of suboptimal trees—and it may be impossible in the case of large data sets with hundreds of thousands of optimal trees. The best method is then searching for trees lacking the group of interest, using negative constraints (see above). Creating constraints and searching for each of the groups in the tree may be very tedious, but a simple script distributed with TNT, Bremer.run, automates this task (creating the constraints and searching for every group). The script automatically writes to the corresponding branches the difference, and plots the tree.

The Bremer.run script also implements an alternative means of calculating Bremer supports, which is probably the only way to estimate Bremer supports for very large data sets. For each group for which the support is to be calculated this method consists of calculating first the average tree-length for each of a number of simple searches (e.g. 5 RAS + TBR saving a single tree) with the group constrained to be monophyletic. This is then followed by a similar calculation, but with the group constrained to not be monophyletic. The difference between the two averages constitutes an estimation of the Bremer supports. This uses a reasoning similar to that in Farris et al.'s (1994) congruence test: if the positively and negatively constrained searches are (on average) off by the same numbers of steps, then their difference in length equals the Bremer supports.

Trees may be plotted with multiple support measures (Bremer support, bootstrap frequency, jackknife frequency, etc.) attached to each branch. This may be done by selecting **Trees/MultipleTags/Store** or the command **ttag =** before running the support calculation, then plotting the trees with **Trees/MultipleTags/ShowSave** or the command **ttag**. The multiple tags can also be saved, in the Windows version, in extended metafile formats.

Randomization

For specific tests or simulations, TNT allows generation of random trees (**Trees/RandomTrees** or ***randtrees***), character permutation (***xperm***), or generation of data (random or simulated under Jukes-Cantor, ***xread***). The results of these can be used, with suitable scripts, to implement specific tests. An example is Goloboff et al.'s (2006) analysis, in which the number of SPR moves corresponding to the consensus trees for discrete-only and continuous-only data sets were compared to those for pairs of random trees, by means of the script in Fig. 6. This script takes as arguments two numbers (the numbers of trees to compare), and outputs the proportion of pairs of random trees which differ in as few (or fewer) SPR-moves (for further explanation of scripting in TNT, see next section). Using scripts, a wide variety of tests can easily be implemented.

Extending TNT's capabilities with scripts

The scripting language allows the making of decisions and the automation of processes, so that TNT

```
macro = ;
var : obs_moves  matches ;              /* declare variables */
set obs_moves sprdiff[ %1 %2 300x5 ] ;  /* observed moves    */
set matches 0 ;
loop 1 1000                        /* repeat 1000 times         */
   rseed * ;                       /* change random seed        */
   keep 0 ;                        /* trash previous trees ...  */
   randtrees 2 ;                   /* and generate two new ones */
   if (  sprdiff[ 0 1 300x5 ] <= 'obs_moves' )
     set matches ++ ;              /* count matches ...         */
     end
   stop
/*** Now, calculate proportion and output results *********/
set matches 'matches' / 1000 ;
quote Observed: 'obs_moves' moves - P(match) = 'matches' ;
proc/;
```

Fig. 6. Example of a simple script, to test whether the number of SPR-moves to interconvert two trees is greater than the number of moves to interconvert two random trees.

can be programmed to perform specific tasks. This extends the capabilities of the program enormously. An example is Pickett et al.'s (2005) analysis, in which they automated tens of thousands of tree searches using TNT scripts.

Only the basic aspects of the scripting language will be explained here; a more detailed description is provided in the documentation that comes with the program.

If a script is placed in a file called ***xxx.run***, in current directory, then just typing ***xxx*** at the TNT command prompt will automatically execute file ***xxx.run***, passing to the file any arguments given to ***xxx*** (argument number ***n*** is identified, within the file, as ***%n***; argument 0 is the name of the file itself).

The basic components of the scripting language are:

(a) commands for flow control and decision-making: decisions are made with ***if***, and repetitive actions are executed with the ***loop*** command (with the symbol # replaced by the current value of the loop's index variable, in every cycle).

(b) internal variables (or values calculated by TNT), accessible only within the context of scripting commands; for example, the expression ***ntrees*** is equivalent to the number of trees in memory minus 1 (recall that TNT numbers everything, except data blocks, starting from 0). Thus,

```
loop 0 ntrees
     tplot #1 ;
     stop
```

will sequentially plot the tree diagram, for each of the trees in memory.

(c) variables defined by the user: these establish the connection between internal variables and the regular commands; they can also be used to store values of calculations. Writing the name (or number) of a user variable within quotes, is equivalent to writing the value of the user variable.

(d) expressions: in the context of the scripting language, when a number is expected, a parenthetical expression (with operations between numbers, or logical comparisons) can be used instead of the number.

Flow-control and decision making

The ***if*** command has the following syntax:

```
if(expression)
  action(s)A;
else
  action(s)B;
  end
```

If the expression following the ***if*** is different from 0, then ***action(s) A*** are executed; otherwise, ***action(s) B*** are executed. The ***else*** and ***action(s) B*** can be omitted. There can be any number of nested ***if***s/***else***'s.

The syntax for a ***loop*** is:

```
loop = name i + j k (action(s)) stop
```

the loop will start at value i and end at value k, every time increasing by the value of j (the "$+j$" can be omitted if the increment is 1; if $k < i$, then the loop is decreasing). Within the specified action(s), the expression "#name" is equivalent to writing the current value of the loop index value (if the equal sign and the name are omitted, then the loop is not named, and the value of the loop is accessed with "#N", where N is the nesting level of the loop to be accessed; nesting level is exclusive of each input file).

Within the loop, the command ***setloop N*** will re-set the loop value to N, ***continue*** will move to the next cycle (skipping all actions between ***continue*** and ***stop***), and ***endloop*** will interrupt the execution of the loop.

Other commands that execute loops are ***sprit*** and ***tbrit*** (execute subsequent actions for each of the SPR or TBR rearrangements of the specified tree), ***travtree*** (executes subsequent actions for each of the nodes in a tree, visited according to a down-pass, up-pass, or path-travelling sequence), and ***iterrecs*** (executes subsequent actions for each of the equally parsimonious reconstructions of the specified character, with the possibility of forcing a specific state at any given node). TNT's on-line help explains the usage of each of these commands in detail.

Internal variables

TNT provides easy access to more than 120 internal variables to be used in decision making (e.g. number of taxa, trees, tree-scores, degree of resolution, branch lengths, tree-distances, hits to minimum length or number of rearrangements in the last search, most parsimonious states at internal nodes, etc.). Internal variables cannot be accessed from regular TNT commands, but only from the scripting commands.

User variables

The user can define variables (arrays with up to five dimensions). Once named and defined (with the ***var*** command), their values can be set with the ***set*** command. For example, the following script will randomly select 10 characters (between 0 and nchar, the number of characters minus 1) and place them (with the ***xgroup*** command) in a character-group named "selected":

```
var : i;
xgroup = 0 (selected) ;
loop 1 10
   set i getrandom [ 0 nchar ] ;
   xgroup 0 'i' ;
   stop
proc/;
```

Note that ***getrandom***, being an internal variable, could not be accessed from the ***xgroup*** command, so that the value has to be transferred first to a user variable called ***i***. The script is merely an example, since the ***xgroup*** command has a built-in option to select characters at random, with which the equivalent result can be achieved more simply: ***xgroup =0 (selected) *10;***.

Other features of the scripting language

The scripting language also allows:

(a) formatted input (***hifile*** command) and output (***quote*** and ***lquote*** commands).

(b) In Windows versions, customizable dialogs (***opendlg*** command).

(c) access to taxon names, character names, branch-legends, etc., for use in formatted output, or for setting internal variables. Within almost any context, a dollar sign followed by the specification of the type and number is automatically converted (e.g. ***$taxon n*** is equivalent to typing the name of taxon number ***n***, and ***$ttag i*** is equivalent to typing the legend of branch number ***i***). Conversely, values can be written to tree-tags for subsequent display on a tree-diagram (***ttag*** command).

(d) input redirection to specific parts of a script (with the ***goto*** and ***label*** commands). Instructions in that section are effectively executed as functions, making it possible to write well structured scripts.

(e) report of progress and interruption (***progress*** command), for time consuming scripts. This is best used with the output silenced (with the ***silent*** command, so that the program does not produce tons of unnecessary output).

Running TNT in parallel

In Linux and Mac versions, it is possible to run TNT in parallel (this requires installation of the PVM system, available at http://www.netlib.org/pvm3). With the ***ptnt*** command, the program can launch, monitor (with information retrievable from the slaves at any time), and control slave tasks (possibly interrupting them, making them return results and continue running, or substituting the instructions for the slaves for a new set of instructions). For details see the documentation of the ***ptnt*** command.

Acknowledgements

The authors wish to express their gratitude to all the users of TNT who have helped improve it, with bug reports and suggestions of new features: Lone Aagesen, James Carpenter, Santiago Catalano, Jonathan Coddington, Julian Faivovich, Gonzalo Giribet, Peter Hovenkamp, Daniel Janies, Diana Lipscomb, Rudolf Meier, Marcos Mirande, Diego Pol, Martin Ramirez, Claudia Szumik, among others. Rudolf Meier and Kurt Pickett offered useful comments and advice on the manuscript.

References

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

Davis, J., Nixon, K., Little, D., 2004. The limits of conventional cladistic analysis. In: Albert, V. (Ed.), Parsimony, Phylogeny, and Genomics. Oxford University Press, Oxford, pp. 119–147.

Farris, J., 1969. A successive approximmations approach to character weighting. Syst. Zool. 18, 374–385.

Farris, J., 1970. Methods for computing wagner trees. Syst. Zool. 19, 83–92.

Farris, J., 1973. On comparing the shapes of taxonomic trees. Syst. Zool. 22, 50–54.

Farris, J., 1988. Hennig86, ver. 1.5. Program and Documentation, Distributed by the Author. Port Jefferson Station, NY.

Farris, J., 1989. The retention index and the rescaled consistency index. Cladistics 5, 417–419.

Farris, J.S., 2001. Support weighting. Cladistics 17, 389–394.

Farris, J., Källersjö, M., Kluge, A., Bult, C., 1994. Testing significance of incongruence. Cladistics 10, 315–319.

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

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

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

Goloboff, P., 1993. Estimating character weights during tree search. Cladistics 9, 83–91.

Goloboff, P., 1994a. Nona: A Tree-searching Program. Program and documentation, available at http://www.zmuc.dk/public/phylogeny/Nona-PeeWee.

Goloboff, P., 1994b. Pee-Wee: Parsimony and Implied Weights. Program and documentation, available at http://www.zmuc.dk/public/phylogeny/Nona-PeeWee.

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

Goloboff, P., 1997. Self-weighted optimization: character state reconstructions and tree searches under implied transformation costs. Cladistics 13, 225–245.

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

Goloboff, P., 2008. Calculating SPR-distances between trees. Cladistics (in press). doi: 10.1111/j.1096-0031.2007.00189.x

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

Goloboff, P., Pol, D., 2002. Semi-strict supertrees. Cladistics 18, 514–525.

Goloboff, P., Pol, D., 2007. On divide-and-conquer strategies for parsimony analysis of large data sets: rec-i-dcm3 vs TNT. Syst. Biol. 56, 485–495.

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

Goloboff, P., Farris, J., Nixon, K.. 2003b. T.N.T.: Tree Analysis Using New Technology. Program and documentation, available at http://www.zmuc.dk/public/phylogeny/tnt.

Goloboff, P., Mattoni, C., Quinteros, Y.S., 2006. Continuous characters analyzed as such. Cladistics 22, 589–601.

Hovenkamp, P., 2004. Review of TNT—Tree analysis using New Technology, ver. 1.0. Cladistics 20, 378–383.

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

Meier, R., Ali, F., 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.

Pickett, K.M., Tolman, G.L., Wheeler, W.C., Wenzel, J.W., 2005. Parsimony overcomes statistical inconsistency with the addition of more data from the same gene. Cladistics 21, 438–445.

Robinson, D.F., Foulds, L.R., 1981. Comparison of phylogenetic trees. Math. Biosci. 53, 131–147. 58, 825–841.

Sankoff, D., Rousseau, P., 1975. Locating the vertices of a Steiner tree in an arbitrary space. Math. Program. 9, 240–246.

Swofford, D., 2002. PAUP*: Phylogenetic Analysis Using Parsimony (* and other methods), ver. 4. Sinauer Associates, Sunderland, MA.

Williams, P., Fitch, W., 1990. Phylogeny determination using dynamically weighted parsimony method. Meth. Enzymol. 183, 615–626.