Bioinformatics, 36(13), 2020, 3988–3995
doi: 10.1093/bioinformatics/btaa214
Advance Access Publication Date: 28 March 2020
Original Paper

Phylogenetics

A phylogenetic C interpreter for TNT

Pablo A. Goloboff* and Martín E. Morales

Unidad Ejecutora Lillo (Fundación Miguel Lillo – Consejo Nacional de Investigaciones Científicas y Técnicas), S. M. de Tucumán 4000, Argentina

*To whom correspondence should be addressed.

Associate Editor: Russell Schwartz

Received on February 5, 2020; revised on March 17, 2020; editorial decision on March 20, 2020; accepted on March 24, 2020

Abstract

Motivation: TNT (a widely used program for phylogenetic analysis) includes an interpreter for a scripting language, but that implementation is nonstandard and uses several conventions of its own. This article describes the implementation and basic usage of a C interpreter (with all the ISO essentials) now included in TNT. A phylogenetic library includes functions that can be used for manipulating trees and data, as well as other phylogeny-specific tasks. This greatly extends the capabilities of TNT.

Availability and implementation: Versions of TNT including the C interpreter for scripts can be downloaded from http://www.lillo.org.ar/phylogeny/tnt/.

Contact: pablogolo@yahoo.com.ar

1 Introduction

TNT (Goloboff and Catalano 2016; Goloboff *et al.*, 2008), a parsimony-based program for phylogenetics, can be used for a number of comparisons, simulations and analyses beyond parsimony searches. The program implements numerous metrics for tree comparisons (e.g. Goloboff, 2008; Goloboff *et al.*, 2017), routines for consensus and supertree calculations (e.g. Goloboff and Pol, 2002; Goloboff *et al.*, 2003), identification of rogue taxa (e.g. Goloboff and Szumik, 2015) and other facilities.

In addition to the built-in commands, TNT also implements a scripting language of its own. This language is specifically designed for ease of use within TNT instruction files, making it possible to intersperse scripting instructions (e.g. decision making, storing values of calculations, flow control) with regular TNT commands (i.e. commands for calculating and handling trees). The scripting language of TNT allows programming rather elaborate routines, such as creation of matrices with only four homoplasy-free multistate characters determining any given tree topology (based on Steel and Penny, 2005; see Supplementary Material of Goloboff and Wilkinson, 2018), creation of matrices with minimum number of homoplastic binary characters determining known fully symmetrical or pectinate trees (based on Chai and Housworth, 2011; see Supplementary Material of Goloboff, 2014), or enumeration and manipulation of alternative most parsimonious reconstructions of continuous characters (Giannini and Goloboff, 2010, see script at http://www.lillo.org.ar/phylogeny/tnt/scripts/xdelcor.run).

Several other phylogeny programs include a scripting language, often with its own syntax and conventions. Examples are Psoda (Carroll *et al.*, 2009), RevBayes (Höhna *et al.*, 2014, 2016) and Mesquite (Maddison and Maddison 2019). These scripting languages generally have the advantage of simplicity, and of having been designed specifically for a phylogenetic setting. A drawback, however, is that their syntax tends to be significantly different from standard programming languages. The tutorial at https://revbayes.github.io/tutorials/intro/rev.html, for RevBayes, is an example. In the case of TNT, assignments must be made by means of the command ***set*** (because TNT parses input for commands first), accessing user variables requires the use of quotation marks as sigils (so that the parser can distinguish between standard numbers or user variables to be read as numbers), and other nonstandard aspects (http://www.lillo.org.ar/phylogeny/tnt/scripts/General_Documentation.pdf).

Among programming languages, C continues being one of the most widely used languages (second only after Java; see https://stackify.com/popular-programming-languages-2018/). C is widely considered to be a programming lingua franca, and the basics of it are learnt at some point or another by most programmers. To make it easier to take advantage of the facilities for tree search and manipulation in TNT, we have included a C-language interpreter into TNT. We have chosen for that purpose a C interpreter noted for its lightweight and versatility, PicoC (Saleeba, 2019), which is widely used for scripting in robotic and embedded applications. Despite having only about 3.5 K lines of code and using little memory at run-time, PicoC implements all of the essentials of ISO (International Organization for Standardization) for the C language, including data types, structures, pointers, functions and pre-processor directives.

2 Running C scripts from TNT; communication with TNT

The command ***runc*** runs a C script from TNT. The stack size for the interpreter is by default 512 kb, but this can be changed by setting an environment variable STACKSIZE. C scripts can be run from TNT in two modes:

a. By invoking a separate file with the script, ***runc filename arg1 arg2 … argn***. In this case, the entry point for script execution is ***main()***. The file containing the script can have any extension, although we recommend using ****.pic*** instead of the standard ****.c*** to avoid confusion with regular C files.
b. On the fly, e.g. interspersed within other TNT instructions. This uses the ***runc!*** option (ending C-style instructions with a line containing only "!"):

```
... (regular TNT commands here)...
runc!
fprintf (stderr, "Hello, world!\n");
!
... (other TNT commands follow)...
```

Only the first mode admits arguments being passed to the C script (via ***argc*** and ***argv*** in ***main***). The advantage of the second mode is that it allows mixing C and TNT syntax in a single file. For additional flow control, it is also possible to use TNT-style scripting to call different C-style scripts, e.g.:

```
if (ntax > 100)
        runc big_data_script.pic;
else
        runc small_data_script.pic;
end
```

Execution of a C script ends when reaching a ***return*** statement in ***main()***, or when the ***exit()*** function is called. The exit value of the script will be subsequently accessible to TNT if set with ***exit()*** (the ***return*** value from ***main()*** is not).

All the headers and libraries implemented in PicoC (***ctype.h***, ***math.h***, ***stdio.h***, ***stdlib.h***, ***string.h***, ***time.h*** and ***unistd.h***) are automatically included in any C script, with no need to ***#include*** any library. In addition to those, another library, which we call the ***tnt*** library, is included. These libraries are included from TNT itself; note that external libraries (e.g. ***ncurses*** or ***libpll***; Flouri *et al.*, 2015) cannot be included, as there is no compilation and linking –just interpretation. Likewise, there is at the time no independent ***tnt*** library which could be used to compile and link the scripts. The C-style scripts are intended for simplicity; they can be executed as soon as they are typed (at the file or console), but minimalism always comes at a cost.

The ***tnt*** library defines functions that enable communication between the C script and TNT, allowing C scripts access to values of internal TNT variables (e.g. numbers of taxa, characters or trees; lengths of trees; etc.), displaying formatted text in the output system of TNT, and executing standard TNT instructions. All of this is done internally, in memory, without the need to create temporary files or pipes; this is one of the biggest advantages of a C interpreter with a ***tnt*** library. Execution of TNT built-in commands from a C script is done with the ***tnt()*** and ***tntlog()*** functions, the prototypes of which are

```
int tnt (char* format, ...)
int tntlog (FILE* fileptr, char* format, ...)
```

The ***tnt()*** function runs the instructions in string ***format*** as if they were commands typed at the TNT prompt. The ***tntlog()*** function does the same, except that it automatically writes any TNT output into the file pointed to by ***fileptr*** (bypassing output files opened in TNT, if any). In both cases, the instructions in ***format*** can be formatted (with escape sequences as in the standard ***printf()*** function), and can be any TNT commands, up to 512 characters long (exceeding this triggers an error message). The instructions passed to TNT can be any valid commands, including commands for TNT-style scripting. The only command that is not valid within ***format*** is the ***runc*** command itself (i.e. precluding recursive calls to the C interpreter).

Many TNT commands require or accept ranges of taxa, characters or trees, on which to execute operations. These lists can be long, and the number of elements to include in the selections may be unknown ahead of time. Our implementation then allows defining lists of elements to be used in any TNT command invoked from the C script. Function prototypes are

```
int charlist (char* list)
int taxlist (char* list)
int treelist (char* list)
```

list must have as many cells as characters, taxa, or trees; use ***list [i]*** = 0 or 1 to indicate exclusion or inclusion of element ***i*** in any subsequent TNT command. Modifications to ***list*** (e.g. adding or removing an element from the list) take effect without the need to use ***charlist()***, ***taxlist()*** or ***treelist()*** again; what remains connected to the TNT parser of ranges is the pointer ***list***, not the values themselves. If ***list=NULL***, then the use of character, taxon, or tree lists is disconnected (using the default selections for all TNT commands). The functions return the resulting status of the corresponding type.

C scripts can also access names in the matrix. The function prototype is

```
char* name (char* option, ... [int number])
```

Some of the values for ***option*** can be 'dataset' (which requires no further arguments), 'taxon', 'character' or 'ttag' (which require a taxon, character or branch number), or 'state' (which requires a character and state number as second and third arguments). The opposite (retrieve an element number, given its name) is done by

```
int namtonum (char* type, char* name, ... [int exact_match])
```

Type can be "taxon", "character" or "block" (among others); ***name*** is the name of the element. Using these functions, it is easy to, e.g. save a list of all the taxa in the matrix to the file pointed to by ***fileptr***:

```
for (i = 0; i < Ntax; ++ i)
        fprintf (fileptr, "Taxon %i: %s\n", i, name ("taxon", i) );
```

TNT-style user variables can be declared and accessed from a C script. The function ***vardec()*** declares TNT variables to be used during execution of the C script (they cease to exist when the script finishes execution); the ***varget()*** and ***varset()*** functions access the values of TNT user variables, for reading and writing, respectively. The prototypes of these functions are:

```
void vardec (char* varname, ... [int dims])
void varget (void* saveto, char* varname, ... [int dims])
void varset (void* readfrom, char* varname, ... [int dims])
```

TNT user variables defined before execution of the C script are thus accessible to the script, and the script can modify those values for subsequent use in the TNT script, e.g.:

```
macro=;
var: n;
set n 666;
runc! /*** begin C-script *****/
        double k;
        varget (&k, "n");
        printf ("C-script: variable in TNT is %d\n", k);
        k = 12345;
        varset (&k, "n");
!/**** return to TNT script ****/
quote TNT-script: C-script changed value to 'n';
proc/;
```

This produces the following two lines of output:

```
C-script: variable in TNT is 666
TNT-script: C-script changed value to 12345
```

Specifying appropriate dimensions, the ***varget()*** and ***varset()*** functions can be used to transfer whole arrays or matrices, instead of single values.

3 Memory management

The standard functions ***malloc()***, ***calloc()***, ***free()*** and ***realloc()*** are included in ***stdlib.h***. In normal execution, PicoC will run a script and exit after termination, thus automatically releasing any memory allocated during the run. In the current implementation, however, exiting the interpreter goes back to TNT; any memory allocated within the script and not freed explicitly before returning to TNT would remain allocated. To prevent this, by default, the C interpreter in TNT keeps track of all memory allocations, and automatically frees all unfreed memory when returning control to TNT (explicitly freeing pointers does no harm, though). This incurs a small penalty in time and memory, but makes it possible to write simpler scripts. The maximum number of simultaneous allocations to track is set from TNT itself, prior to running the script, with ***runc[N*** (where ***N*** is the number of allocations; use ***N = 0*** to track no allocations)]. The default number of allocations to track is 5000; if more than 5000 simultaneous allocations are done, some of those will be untracked and may cause memory leaks if the script is run repeatedly.

In addition to tracking memory automatically, the ***tnt*** library implements three other functions that facilitate memory management:

```
int tagmem(void)
void freetag (int mem_ID)
void freeall (void)
```

The ***tagmem()*** function returns an int (32 bit) that can be used to reset memory status. Subsequent invocation of ***freetag()*** resets the system, releasing any memory allocated subsequently to the ***tagmem()*** call which had returned the value used as argument for ***freetag()***. Thus, in the following example:

```
void internal_function (void) {
        int mytag = tagmem ();
        int* ptr1=malloc (...), * ptr2 =malloc(...);
        char* ptr3=malloc (...);
        ...
        do stuff with ptr1 – ptr3 here
        ...
        freetag (mytag);
}
```

the use of ***freetag()*** at the end of ***internal_function()*** allows returning to the calling function without individually having to release every one of the pointers allocated within ***internal_function()***. The ***freeall()*** function releases all memory allocated within the script—this is equivalent to ***freetag (0)***.

4 TNT library

The functions in the ***tnt*** library are the most important addition to the interpreter. These allow not only internally executing any TNT command from a C script, but also accessing internal variables, lengths and dimensions of trees, exchanging trees and data between TNT memory and arrays, handling and enumerating most parsimonious reconstructions for characters, exploring TBR and SPR neighborhoods of trees, and comparing trees. We have taken care to include functions that facilitate most of the operations that may be needed in phylogenetic analyses and comparisons. The ***tnt*** library defines 40 read-only values (i.e. constants) and over 150 functions, the code for which is about 4.5 K lines (i.e. more than the code for PicoC itself).

The read-only values are set by TNT on reading data and calculating trees. Table 1 exemplifies some of the most commonly used expressions and their meanings. Several simple functions (Table 2) allow retrieving information about trees and groups. These functions find ancestors, sisters, or descendants, as well as monophyly of groups; they also allow exchanging trees (as ancestor lists) between a C-script and internal TNT memory.

Other functions (Table 3) perform more involved types of operations, such as calculation of group frequencies, Bremer (1994) supports, tree comparisons, exchange of values between TNT tables and scripts, calculation of tree scores and retrieving character states at terminal or internal nodes (i.e. mapped states). We discuss here only a simple example, to illustrate the potential of the ***tnt*** library. The function ***swaptree()*** takes a tree and generates all the trees in the SPR or TBR neighborhood. For every tree, a user-defined function (the name of which must be passed as argument to ***swaptree()***) is

Table 1. Expressions recognized in the context of C scripts, defined in the ***tnt*** library

Expression	Meaning
exstatus	Last exit status of TNT (e.g. closing TNT instructions with '***return N***' sets ***exstatus*** to N)
listsize	In functions that write lists (e.g. ***deslist()***, ***downlist()***, ***grptogrp()***), number of elements written to list
maxtrees	Maximum number of trees to hold in memory (set in TNT)
missing	Value of a missing entry (this depends on maximum number of states, set in TNT)
nblocks	Number of blocks in the current dataset
Nchar	Number of characters in current dataset (lower case n: minus 1)
Ntax	Number of taxa in current dataset (lower case n: minus 1)
Ntrees	Number of trees now in memory (lower case n: minus 1)
numhits	Number of hits to best length found in last TNT search
outgroup	Current outgroup taxon

Table 2. Functions for basic handling of trees, defined in the *tnt* library

Type	Function	Arguments	Action
int	***anc***	int ***T***, int ***N***	Ancestor of node ***N*** in tree ***T***
int	***comnod***	int ***T***, char* ***list***	Return common node, in tree ***T***, of all nodes listed in ***list*** (***list*** is an array, with at least 2****Ntax*** values; ***list[i]=0*** or ***1*** excludes or includes node ***i***). Alternatively, passing three or more int arguments to ***comnod***, list of nodes is taken from args2, arg3, . . ., argn
int	***deslist***	int* ***save***, int ***T***, int ***N***	Save list of immediate descendants of node ***N***, in tree ***T***, into array ***save***. Return (and write to ***listsize***) number of values saved in array
int	***distnode***	int ***T***, int N_1, int N_2	Distance (=branches) between nodes N_1-N_2, for tree ***T***
int	***downlist***	int* ***save***, int ***T***, . . . [int ***N***]	Save list for down-pass (htu's only) of tree ***T***, into array ***save***. Return (and write to ***listsize***) number of values saved in array. Optionally, indicate node ***N*** for building list.
int	***eqgroup***	int T_1, int ***N***, int T_2	Node of tree T_2 that corresponds to node ***N*** of tree T_1
int	***eqtrees***	int T_1, int T_2	Are trees T_1 and T_2 the same? 0: different, 1: same
int	***gettree***	int* ***save***, . . . [int ***T***]	Copy TNT tree number ***T*** onto array ***save***. If no arg2 given, then it copies from last memory tree. Special cases of arg2: -1 constraint, -2 tagtree.
int	***ismono***	int ***T***, char* ***list***	Return common node for taxa listed in ***list*** if they form a monophyletic group in tree ***T***, 0 otherwise. Array ***list*** must have at least ***Ntax*** values; ***list[i]=0*** or ***1*** excludes or includes taxon ***i***). If more than two arguments passed, they are read as int's, and ***ismono*** returns node common to taxa N_1-N_n if they form a monophyletic group in tree ***T***.
int	***nnodes***	int ***T***	Number of nodes of tree ***T***, minus 1
int	***mono***	int ***T***	Does tree ***T*** satisfy defined constraints? (defined with TNT command "***force***")
int	***nodfork***	int ***T***, int ***N***	Number of immediate descendants of node ***N*** in tree ***T***
int	***numdes***	int ***T***, int ***N***	Number of terminals belonging to node ***N*** of tree ***T***
int	***sister***	int ***T***, int ***N***	Sister of node ***N***, in tree ***T***; if node polytomous, last sister = –1
int	***settree***	int* ***source***, . . . [int ***T***]	Copy ***source*** tree onto TNT memory tree ***T***. If no arg2 given, copy onto last memory tree (and increase number of trees in memory). ***Source*** is a list of ancestors, and it must fulfill certain rules (see online help of TNT for details). TNT performs a sanity check before storing the tree, possibly renumbering nodes when storing in memory (so that they follow TNT's internal rules for node numbering, enabling tree handling and comparison)
int	***tagset***	int ***N***, char* ***format***, . . .	Copy string in ***format*** into tree-tag of node number ***N*** (formatted as in ***printf()***). Return length of resulting tree-tag.
int	***tagtree***	int ***T***	Make tree ***T*** the tag tree (and clear all tags)
int	***tnodes***	int ***T***	Number of nodes (=groups) in tree ***T***
int	***tsize***	int ***T***	Number of terminal taxa included in tree ***T***

called, which can be used to process the tree. For example, the following code in a file ***swapdemo.pic***,

```
int numrearrangs = 0;
int best_length;
int tree_num;
int tree_processing (int cut, int rooted_at, int location) {
        int i = length (tree_num);
        tntprintf ("Rearrangement nr. %i. Length %i\n",
                numrearrangs ++, i);
        tnt ("tplot %i;", tree_num);
        if (i < best_length) best_length = i;
        return 1;
}
void main (int argc, char ** argv) {
        tree_num = atoi (argv [1] );
        best_length = length (tree_num);
        swaptree ("tree_processing", tree_num);
        tntprintf ("Tried %i rearrangs. Best tree: %i steps.\n",
                numrearrangs, best_length);
}
```

will perform SPR on tree number ***N*** (passed as argument, with ***runc swapdemo.pic N***). This simply explores the neighborhood of the initial tree, without moving to better trees, if found. The function ***swaptree()*** switches to a new neighborhood when the ***tree_processing()*** function returns 2 (instead of 1). The function ***progress()*** produces a progress report, and the expression ***percswap*** gives the (estimated) percentage of swapping completed (on a 0–100 scale). Thus, modifying the processing function to be:

```
int tree_processing (int cut, int rooted_at, int location) {
        int i = length (tree_num);
        ++ numrearrangs;
        progress (percswap, 100, "Swapping, best length: %i",
                best_length);
        if (i < best_length) {
                best_length = i;
```

Table 3. Functions for various types of phylogenetic calculations, defined in the *tnt* library

Type	Function	Arguments	Action	Return value
int	***bremlist***	double* ***save*** char* ***options***	Run ***options*** (as in TNT command '***bsupport***'), write Bremer values to array ***save.***	Largest node of tree
int	***collectable***	double* ***save***	If any table displayed in TNT, store values in array ***save*** (if ***save***=***NULL***, stop retrieving).	Status of table retrieval
double	***combine***	int* ***list***, char* ***funcname***, int ***num***, int ***min***, . . .[int ***max***]	Produce all combinations of ***min*** out of ***num*** elements; if ***max*** indicated [optional], also produce ***min***+1, ***min***+2, . . ., ***max***. Write elements combined into ***list***. For each combination, call function ***funcname***. Function ***funcname*** must take a single argument (the number of elements picked in that combination) and return an int to ***combine*** (which indicates whether to stop process, 0, or continue generating combinations, 1).	Return number of combinations generated. Note: if ***list*** NULL, it returns number of combinations to generate (calling no other functions)
void	***disptable***	double* ***values***, int ***nvals***, char* ***title***,. . . [int ***prec***], . . .[int ***names***]	Display ***values*** in a TNT table (formatted as set with TNT command '***tables***'), with ***nvals*** values, and ***title***. Optionally, precision can be given as ***prec***. For tables with optional format ('***tables***=;' command of TNT), character or taxon ***names*** can be used (if '***charnames***=;' or '***taxnames***=;' set in TNT), passing a fifth argument; 1 character names, or 2 taxon names (0 no names used).	
int	***freqlist***	int* ***save***, char* ***options***	Run ***options*** (as in '***majority***' command of TNT), write values of group frequencies into ***save.***	Largest node of tree
int	***gcomp***	int T_1, int N_1, int T_2, int N_2	Are nodes N_1 of tree T_1, and node N_2 of tree T_2 compatible? (0) incompatible; (1) node N_1 includes N_2; (2) node N_1 included in N_2; (3) node N_1 equals N_2; (4) N_1 and N_2 are disjunct taxon sets.	Result of comparison
double	***gfreq***	int ***T***, int ***N***	Frequency of group ***N*** of tree ***T***. Functions ***charlist()*** and ***taxlist()*** determine which trees and taxa to consider (otherwise, all).	Group frequency
int	***grouplist***	int* ***save***, int ***T***, int ***N***	Write to ***save*** the list of the terminals that belong to node ***N*** of tree ***T***. Return value is also written in ***listsize***.	Number of terminals written in ***save***
int	***grptogrp***	int* ***save***, int T_1, int T_2	Write to ***save*** node correspondences between two trees, saving for each node of tree T_1 the number of equivalent node in tree T_2 (none = −1). Can ignore positions of taxa with ***taxlist()***.	Number of nodes of target tree.
double	***implik***	int ***T***	Calculate likelihood of tree ***T*** by finding optimal branch length for each character (the same for each branch, different for each character, as in Goloboff and Arias, 2019), summing over alternative reconstructions. This can use a common morphospace (possibly changed to ***N*** states with TNT command '***lset gstate N***', or the	Log likelihood value

Table 3. (continued)

Type	Function	Arguments	Action	Return value
			largest state in matrix) or individual morphospace (with '***lset nogstatespace***').	
void	***iterrecs***	double* ***values***, char* ***funcname***, int ***T***, int ***C***, . . .[int ***options***, int ***sample***, int ***increment***]	Iterate reconstructions for tree ***T***, char ***C***; for every reconstruction, store state for each tree node on ***values***, and call function named ***funcname***. Function ***funcname*** must take as argument a double (the score of the reconstruction), and return an int to ***iterrecs()*** (0 = stop process, 1 = continue). Individual bits in ***options*** determine special ways to reconstruct, including forcing certain states to some nodes (hence the need to pass score as argument to ***funcname***). See online help of TNT for details.	
double	***mklik***	int ***T***	Calculate likelihood for tree ***T***, under Mk model (single rate). Morphospace determined as for ***implik()***.	Log likelihood value
void	***progress***	int ***done***, int ***todo***, char* ***message***, . . .	Draw a progress bar, showing the percentage of ***todo*** that ***done*** represents. Display ***message*** (formatted as in ***printf()***). Every update of the progress bar also checks for user interrupts	
void	***randomlist***	int* ***save***, int ***N***, . . . [int ***first***]	Write to ***save*** a random list, from ***0*** to ***N-1***. Arg3 [optional] is the value to place as ***first*** value of the list.	
double	***rfdist***	char* ***options***	Robinson-Foulds distances between trees (***options*** as in '***tcomp <***' command of TNT). Use ***r*** as prefix (i.e. ***rrfdist***) for a rooted distance; use ***p*** as a suffix (i.e. ***rfdistp***, ***rrfdistp***) to normalize by number of groups present in trees (instead of all possible groups).	Distance value
double	***score***	int ***T***, . . . [int ***C***]	Score of tree ***T*** (optional, character ***C***). If implied weights is off, number of steps; otherwise, return tree fit.	Score value
int	***simgroup***	int T_1, int ***N***, int T_2	Return the group of tree T_2 that is most similar to group ***N*** of tree T_1 (determining similarity as done by the '***rfreq***' command of TNT, and returning 0 when maximum similarity is 0). If T_1= -1, then it uses reference taxonomy; if T_1= -2, then it uses constraint tree. Stores number of taxa shared by both groups (***reg_beta***), added to group in ***N*** (***reg_alfa***), or removed from group in ***N*** (***regr***)	Node number (or 0 for no similar group)
void	***sortlist***	int* ***save***, double* ***values***, int ***N***	Write to ***save*** indices of ordered ***values*** (increasing order), for ***N*** values. Differs from ***qsort()*** in that it preserves the indices of values.	
double	***sprdiff***	int T_1, int T_2, int ***R***, int ***P***	Return number of SPR moves to convert tree T_1 into T_2, using ***R***	Distance value

Table 3. (continued)

Type	Function	Arguments	Action	Return value
			replications, and ***P*** passes (see Goloboff, 2008 for details of the algorithm). If ***P*** is negative, then it alternates passes with and without stratification. Moves can be weighted by distance (set with the TNT command '***sprdiff***')	
int	***states***	void* ***save***, ...[int ***T***, int ***C***, int ***N***]	Save data matrix onto array (int **) pointed to by arg1 (must be large enough to hold ***Nchar*Ntax*** values), as state (=bit) sets. If ***T*** specified, then array will save ***2*Ntax-1*** values, with states for internal nodes resulting from optimizing tree ***T***. If ***T***, ***C*** and ***N*** specified, then arg1 can point to a single number. If ***C*** is -1, then it copies all characters for the node(s) specified (with arg1 pointing to int** or int*, depending on whether ***N*** is negative or not). Likewise for ***N***. Thus, ***states(ptr)*** is equivalent to ***states(ptr,-1,-1,-1)***. If ***save=NULL***, then it only returns number of dimensions required to store values for the given rest of arguments; ***regalfa*** and ***regbeta*** contain the sizes needed along first and second dimension (0=none). Function ***downstates()*** is similar, but copying down-pass states to internal nodes.	Number of dimensions needed for the given arguments (0=a pointer to a single number)
int	***swaptree***	char* ***funcname***, int ***T***, ... [int ***swaptype***]	Perform branch swapping on memory tree ***T***, calling function ***funcname*** for every rearrangement. Function ***funcname*** must return an int to ***swaptree*** (indicating whether to stop search, 0, continue, 1, or resetting swap and start swapping from new tree, 2), and take three arguments (what is cut, where it is rooted, and where it is inserted). ***Swaptype*** indicates whether to perform SPR (0), SPR moving outgroup (1), or TBR (2); default ***swaptype*** is 0. When swapping, variable ***percswap*** indicates percentage of swapping completed.	Number of moves made
double	***symcoeff***	char* ***options***	Calculate symmetric distortion coefficient (modified from Farris 1973), with options as in '***tcomp==***' command of TNT	Distortion value
int	***travtree***	char* ***travel***, char* ***funcname***, int ***T***, ...[int ***N***]	Sequentially visit nodes of tree ***T***, as indicated in ***travel***, calling every time function named ***funcname*** (function must return int, and take as argument the node being visited, as int). Travel can be 'up', 'down', 'path', 'below' or 'des'. For up and down, the node ***N*** used as pivot for the travel is by default the root node (it can be indicated	Number of nodes visited.

Table 3. (continued)

Type	Function	Arguments	Action	Return value
			as argument, after T). For ‘path’, two node numbers must be indicated after tree number. Return values of ***funcname*** indicate whether to stop travel (0), continue with next node (1), or whether to skip the descendants of the current node (2, only for mode ‘up’).	

```
            return 2;
        }
        return 1;
        }
```

and erasing the progress bar in ***main()*** after returning from ***swaptree()***:

```
    swaptree ("tree_processing", tree_num);
    progress (0, 0, NULL);
```

the resulting code will find the best tree it can, by doing SPR on the starting tree, reporting progress as it works. The ***tree_processing()*** function itself could be designed to use a different criterion to score trees [e.g. certain groups monophyletic, a certain maximum Robinson-Foulds distance (Robinson and Foulds 1981) to a reference tree etc.] instead of a standard parsimony search (which, after all, is already implemented internally in TNT).

Financial Support: none declared.

Acknowledgements

The authors thank Steve Farris for his insistence that a more standard scripting interpreter should be included in TNT. The support from CONICET (PUE 0070) was deeply appreciated. We also thank Santiago Catalano for testing many of the functions of the ***tnt*** library and suggesting some features. Comments from three reviewers (Mark Simmons, and two anonymous) helped improve the article, for which we are grateful.

Conflict of Interest: none declared.

References

Bremer,K. (1994) Branch support and tree stability. *Cladistics*, **10**, 295–304.

Carroll,H. *et al.*, (2009) An open source phylogenetic search and alignment package. *Int. J. Bioinf. Res. App.*, **5**, 349–364.

Chai,J. and Housworth,E. (2011) On the number of binary characters needed to recover a phylogeny using maximum parsimony. *Bull. Math. Biol.*, **73**, 1398–1411.

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

Flouri,T. *et al.*, (2015) The phylogenetic likelihood library. *Syst. Biol.*, **64**, 356–362.

Giannini,N. and Goloboff,P. (2010) Delayed-response phylogenetic correlation, an optimization-based method to test covariation of continuous characters. *Evolution*, **64**, 1885–1898.

Goloboff,P. (2008) Calculating SPR-distances between trees. *Cladistics*, **24**, 591–597.

Goloboff,P. (2014) Hide and vanish: data sets where the most parsimonious tree is known but hard to find, and their implications for tree search methods. *Mol. Phyl. Evol.*, **79**, 118–131.

Goloboff,P. and Arias,J. (2019) Likelihood approximations of implied weights parsimony can be selected over the Mk model by the Akaike information criterion. *Cladistics*, **35**, 695–716.

Goloboff,P. and Catalano,S. (2016) TNT version 1.5, including a full implementation of geometric morphometrics. *Cladistics*, **32**, 221–238.

Goloboff,P. and Pol,D. (2002) Semi-Strict Supertrees. *Cladistics*, **18**, 514–525.

Goloboff,P. and Szumik,C. (2015) Identifying unstable taxa: efficient implementation of triplet-based measures of stability, and comparison with Phyutility and RogueNaRok. *Mol. Phyl. Evol.*, **88**, 93–104.

Goloboff,P. and Wilkinson,M. (2018) On Defining a Unique Phylogenetic Tree with Homoplastic Characters. *Mol. Phyl. Evol*, **122**, 95–101.

Goloboff,P. *et al.* (2003) Improvements to resampling measures of group support. *Cladistics*, **19**, 324–332.

Goloboff,P. *et al.* (2008) TNT, a free program for phylogenetic analysis. *Cladistics*, **24**, 774–786.

Goloboff,P. *et al.*, (2017) Comparing tree-shapes: beyond symmetry. *Zool. Scripta*, **46**, 637–648.

Höhna,S. *et al.* (2014) Probabilistic graphical model representation in phylogenetics. *Syst. Biol.*, **63**, 753–771.

Höhna,S. *et al.* (2016) RevBayes: bayesian phylogenetic inference using graphical models and an interactive model-specification language. *Syst. Biol.*, **65**, 726–736.

Maddison,W. and Maddison,D. (2019). Mesquite: a modular system for evolutionary analysis. Version 3.61. http://www.mesquiteproject.org. (14 March 2020, date last accessed).

Robinson,D. and Foulds,L. (1981) Comparison of phylogenetic trees. *Math. Biosc*, **53**, 131–147.

Saleeba,Z. (2019). https://gitlab.com/zsaleeba/picoc.

Steel,M. and Penny,D. (2005). Maximum parsimony and the phylogenetic information in multistate characters. In: Albert,V. (ed.) *Parsimony, Phylogeny, and Genomics*. Oxford University Press, London, pp. 163–178.