WILEY-BLACKWELL

Cladistics (2013) 1–13

Cladistics

10.1111/cla.12047

Extended implied weighting

Pablo A. Goloboff*

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

Accepted 06 June 2013

Abstract

Several extensions to implied weighting, recently implemented in TNT, allow a better treatment of data sets combining morphological and molecular data sets, as well as those comprising large numbers of missing entries (e.g. palaeontological matrices, or combined matrices with some genes sequenced for few taxa). As there have been recent suggestions that molecular matrices may be better analysed using equal weights (rather than implied weighting), a simple way to apply implied weighting to only some characters (e.g. morphology), leaving other characters with a constant weight (e.g. molecules), is proposed. The new methods also allow weighting entire partitions according to their average homoplasy, giving each of the characters in the partition the same weight (this can be used for dynamically weighting, e.g. entire genes, or first, second, and third positions collectively). Such an approach is easily implemented in schemes like successive weighting, but in the case of implied weighting poses some particular problems. The approach has the peculiar implication that the inclusion of uninformative characters influences the results (by influencing the implied weights for the partitions). Last, the concern that characters with many missing entries may receive artificially inflated weights (because they necessarily display less homoplasy) can be solved by allowing the use of different weighting functions for different characters, in such a way that the cost of additional transformations decreases more rapidly for characters with more missing entries (thus effectively assuming that the unobserved entries are likely to also display some unobserved homoplasy). The conceptual and practical aspects of all these problems, as well as details of the implementation in TNT, are discussed.

For cladistic analysis, available evidence suggests that weighting characters inversely to their homoplasy (as in the pioneer work of Farris, 1969; modified by Goloboff, 1993) improves the results. Goloboff (1997) and Goloboff et al. (2008a) showed that, for morphological data sets at least, applying implied weights improves measures normally associated with quality of results, such as bootstrap or jackknife frequencies, and measures of stability. Goloboff et al. (2008a) also observed that results in large molecular data sets are improved (if only slightly so) by weakly weighting against homplasy.

Even if by those meta measures the results seem to improve under implied weights, Goloboff et al. (2008a) recognized the potential, in molecular data sets, of using weighting schemes not formerly available under standard implied weights:

> For prealigned sequence data, establishing categories of characters or transformations may be preferable to evaluating every character separately (as implied weighting normally does). This does not necessarily mean the often used ts/tv ratios, but other less explored —and probably simpler—possibilities, such as collectively weighting all the sites in a region of the sequence according to the average homoplasy of all the sites in that region… As a consequence, combined analyses of morphology and molecules may remain problematic, at least until a reasonable way to apply implied weighting to some parts of the data set but not others is developed (Goloboff et al., 2008a: 769).

The purpose of the present paper is to describe several improvements along those lines, implemented in recent versions of TNT (Goloboff et al., 2008b). First, a method for combining characters subject to implied

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

weights with characters of constant weight is described. In the same vein, TNT now allows the user to define sets of characters (e.g. different genes), and apply to each of the characters in the partition a uniform weight that depends on the average homoplasy of the partition. This allows a sort of "gene weighting", instead of weighting individual positions; while this is easily accomplished in methods such as successive weighting, achieving the proper results under implied weights requires special precautions, as discussed below, and was formerly impossible under implied weights. Finally, recent versions of TNT also allow the additional homoplasy likely to be observed if missing entries were completed to be taken into account, thus countering the objection that implied weights may be biased in favour of characters with many missing entries.

Fitting functions and the cost of adding steps

For implied weighting, TNT uses by default the complement of the fitting function proposed by Goloboff (1993). This is a measure of the "weighted homoplasy" or "distortion" for each character on the tree (Fig. 1A). Thus, if h is homoplasy, and k is the concavity constant, the value of weighted homoplasy w is:

$$w = h/(h+k). \quad (1)$$

The sum of character distortions Σw_i is then used to score trees, and tree searches attempt to find trees minimizing those scores (Goloboff, 1993). Under such a measure, the addition of successive steps beyond the first step of homoplasy costs less and less. The increase in score implied by adding a step to a character with h steps of homoplasy could be termed the instantaneous cost $C_{(h)}$; for the particular weighting function of eqn 1, $C_{(h)} = w_{(h+1)} - w_{(h)}$. Figure 1B shows $C_{(h)}$ rescaled so that the cost of adding the first step of homoplasy costs unity. Weighting against homoplasy results from the fact that $C_{(h)}$ is a decreasing function. Of course, one could design a function such that the weight of the character increases with homoplasy, i.e. $C_{(h)}$ is an increasing function.

TNT also allows the user to define any kind of weighting function by specifying the values of $C_{(h)}$ for every possible number of extra steps. TNT converts these values of $C_{(h)}$ into score values, $w_{(h)} = \sum_{i=0}^{h-1} C_{(i)}$, which are precalculated and stored in a lookup table; note that the distortion values themselves must always be monotonically increasing with h, but the shape of the line in the case of weights increasing with homoplasy will be concave instead of convex. All the new implementations described here can also be applied to these user-defined weighting functions.

Giving some characters constant weight

The recent versions of TNT allow the user to give any specified character a constant weight, while the rest of the characters are subject to implied weights. For this, it is necessary to specify the cost of every step in the constant-weight characters, relative to a certain number of steps of homoplasy, under the reference concavity k. Once the equivalence has been specified, TNT simply operates by calculating tree-scores using, for the corresponding characters, a lineal function instead of the weighting function. Explicitly establishing an equivalence for the cost of every step in the constant-weight characters and the cost $C_{(h)}$ for a specific value of h in a character of variable weight is inescapable. To make a constant-weight character as influential as a character with N steps of homoplasy, TNT will simply multiply the steps in that character by $C_{(N)}$ (the instantaneous step cost in the reference weighting function, i.e. the one to be applied to the rest of the characters). This prior weight will normally

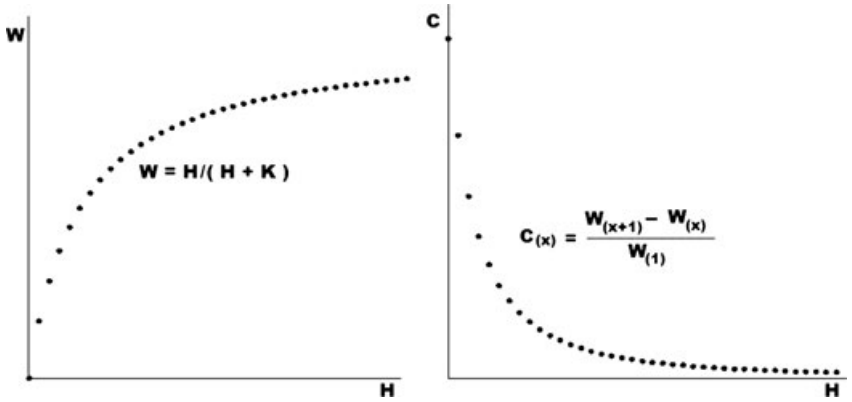

Fig. 1. A, shape of the default function used in implied weights, or weighted homoplasy (a function to be minimized); B, instantaneous step costs for different numbers of extra steps.

be fractional and (as TNT does not allow users to give characters fractional prior weights) is handled internally by TNT.

Collectively weighting partitions

Rationale and concepts

Goloboff et al. (2008a) suggested that, in the case of molecular sequences, collectively (and uniformly) weighting all the sites in a region according to the average homoplasy in the region may be a more appropriate weighting scheme. This may be desirable, for example, to differentially weight regions coding for protein active sites, or to collectively weight regions coding for stems and loops in rRNA or tRNA. Taken to the limit, the different genes in a combined analysis may be given each its own weight, thus providing a "gene weighting" scheme.

This partition weighting cannot be done by simply using totals of homoplasy for each partition, as this would introduce unwanted side effects. Such an approach to implied weights was implemented in older versions of POY (Wheeler et al., 2006), which calculated the total homoplasy ($\boldsymbol{T_i}$) of each fragment or partition $\boldsymbol{i}$, and tried to find the tree(s) minimizing the sum $\boldsymbol{\Sigma T_i/(T_i + k)}$ over all partitions. That specific implementation was probably chosen because there are no predefined "characters" in POY, and so there is no direct equivalent of implied weights. However, that implementation will give different weights to different partitions, but not necessarily in the way intended by Goloboff (1993); long fragments will usually have a higher $\boldsymbol{T_i}$, and thus postulating an additional step occurring on those fragments may well be less costly (in terms of final score) than postulating an additional step in a shorter fragment—even if the average homoplasy per site in the longer fragment is less than in the shorter. In other words, this implementation will (other things being equal) weight against longer fragments, favouring shorter ones.

In the case of successive weighting, the desired weighting can be achieved easily by calculating the average homoplasy of all the sites in the partition and using this average to reassign weights to each of the characters in the partition; this is straightforward. Under implied weights, therefore, it is necessary to produce an equivalent of that approach, which can be achieved by considering that each of the $\boldsymbol{N}$ characters of the partition should receive the same weight, and that the average homoplasy is $\bar{H} = T/N$. If the homoplasy were uniformly distributed among all the characters in the partition, then the score for each of the characters would be $\bar{H}/(\bar{H}+k)$, and the score $\boldsymbol{P}$ for the partition would be $P = N \times \bar{H}/(\bar{H}+k)$, equivalent to

$$P = T/(T/N + k). \quad (2)$$

Thus, the tree that minimizes eqn 2 will evenly distribute weights among all characters of a partition, independently of the number of characters included in the partition.

Equation 2 may be applied to collectively weighted partitions, while at the same time some individual characters are weighted with eqn 1 (eqn 2 reduces to eqn 1 when $\boldsymbol{N = 1}$).

Implications

One of the interesting implications of this approach is that uninformative characters influence the outcome of analyses—they do so indirectly, influencing the (implied) weights of the partitions. Consider the matrix in Fig. 2, with two partitions. There are only two informative characters in the matrix, one supporting the group C+D, the other supporting the group B+C. Under standard implied weights (or under equal weights), the matrix would produce two trees. However, when collectively weighting partitions, the tree with group C+D is preferred: the character supporting that group is the only one with up to one step of homoplasy, out of five characters (the other four characters are uninformative, but they *do* lack homoplasy). In contrast, the character supporting the alternative grouping is the only one in the partition; the partition consists solely of characters with up to one step of homoplasy. The average homoplasy is therefore lower in the larger partition, and thus the tree supported by that partition is preferred in this case.

Note that the influence of uninformative characters just discussed does not arise from this particular implementation of the idea of collectively weighting partitions; it is instead a consequence of the very idea of using the average homoplasy of the partitions. Under successive weighting, reweighting the partitions with average homoplasy would have exactly the same dependence.

Of course, uninformative characters can always be excluded from consideration, by simply deactivating

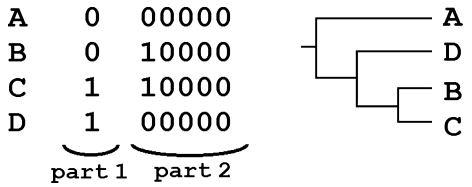

Fig. 2. Example showing the influence of uninformative characters in partition-weighting. See text for discussion.

them (e.g. with the ***xinact*** command of TNT). However, all the invariant positions of a gene do provide information regarding the evolution of that gene, and it then follows that (if the rationale for this method is accepted) uninformative positions should probably be considered and allowed to influence the implied weights.

Implementation details

Weighting partitions by their average homoplasy poses some difficulties for fast computations during tree searches.

Ideally, the evaluation of each tree generated during TBR branch swapping should be done as discussed by Goloboff (1996: 202): calculating the score for the two subtrees that result from dividing the tree in two, and then sequentially examining all the characters in the matrix for each reinsertion point, increasing the score for those characters that will require additional steps. In this way, because the initial score is the sum of scores for the two subtrees, the bound (the score of the best tree found so far) is often exceeded after checking the first few characters in the matrix, allowing for a faster rejection of most rearrangements. These calculations are exact and straightforward in the case of equal weights.

In the case of standard implied weights, the same approach can be used. To do so, TNT uses the (precalculated) instantaneous step costs for each possible number of extra steps. Thus, when the sequential examination of characters to evaluate a given reinsertion point reveals a character of X steps for which the steps will be increased, TNT simply increases the total tree score in $\boldsymbol{C_{(x)}}$ (clearly, when the tree is divided in two, the scores are decreased in the same manner). This allows the evaluation of rearrangements, during branch swapping, at almost the same speed as under equal weights; some extra time is needed because of the extra cost involved in floating point operations and the additional bookkeeping (e.g. it is necessary to store and repeatedly access the number of extra steps for each of the individual characters). For example, in the case of Källersjö et al.'s (1999) 2594-taxon data set (on a 4-GB machine running under Windows Vista), TNT evaluates rearrangements under equal weights at 320×10^6 trees per second, while the speed under implied weights is 144×10^6 trees per second (2.2 times slower). But although the method described allows for a fast evaluation of rearrangements, the score values it yields may be slightly different (due to the imprecision implied in subtractions and additions in floating point numbers) from the scores that would result from simply calculating *de novo* the scores for each of the characters in the matrix. This imprecision could be avoided by calculating the score of a rearrangement by first counting steps for each of the characters in the matrix (in the manner described by Goloboff, 1996), and only then calculating the sum of $\boldsymbol{w}$ scores for all characters; but in such a case, it is necessary to sequentially examine all (or most) of the characters, as the bound will almost never be exceeded as the first characters in the matrix are checked (because the initial score is not the sum of scores for the divided tree, but instead zero). Although not as slow as completely re-optimizing each character (because the step count for each character is done with a single comparison between the states of the nodes to be joined; with a full down-pass optimization per rearrangement, a search for this data set would evaluate about 30 trees/s on this machine), this is much slower than the approximate procedure used in TNT. Apparently this is what PAUP* (Swofford, 2002) uses for implied weighting, because while PAUP* (on the same machine and data set) evaluates rearrangements under equal weights at 2.4×10^6 trees/s, the speed is enormously decreased under implied weights, examining only 68×10^3 trees/s. That is, for this data set, PAUP* under implied weights is over 30 times slower than under equal weights; while PAUP* under equal weights runs 135 times slower than TNT, PAUP* under implied weights runs over 2100 times slower than TNT.

The problem of the imprecision in calculating scores by adding and subtracting score differences in individual characters is avoided in TNT by using a small margin of tolerance, accepting all rearrangements which are within 10^{-3} units of score beyond the best trees and subsequently rechecking the score, for those rearrangements only, using the slower but more precise method (which must yield exactly the same value as an optimization from scratch, as long as the characters are ordered with the same sequence). This margin of tolerance is well above the errors that may be expected in floating point calculations, yet it is small enough that the score differences among trees will usually exceed this value (requiring rechecking of only a small proportion of trees).

In the case of collectively weighted partitions, additional problems arise. The score that results from a given subtree reinsertion can still be calculated easily as the sequential examination of characters progresses, by tallying totals per partition. Then, for a character belonging to partition $\boldsymbol{i}$ that requires an additional step, the score must be modified by increasing $\boldsymbol{T_i}$, subtracting from the total tree score $\boldsymbol{S}$ the previous score for the partition and adding the score that results from increasing $\boldsymbol{T_i}$ by 1:[1]

[1]The example assumes non-additive characters, for simplicity. Clearly, Sankoff, continuous, or additive characters may require a different step increase.

$$S = S + [(T_i + 1)/((T_i + 1)/N_i + k) - T_i/(T_i/N_i + k)].$$

The simplest approach for computing the score of a rearrangement would reinitialize the values of T_i for each of the partitions to the values corresponding to the divided tree, for each rearrangement to be evaluated. But if doing so, when the data set comprises numerous partitions, branch-swapping would be significantly slowed down by this repeated reinitialization. The data set of Källersjö et al. (1999) was used for testing the effect of numerous partitions: when it is divided into 476 partitions of three positions each, a near-optimal tree is swapped at a speed of about 26.25×10^6 trees/s (12.25 times slower than under equal weights, 5.5 times slower than under standard implied weights). The reinitialization of the values of T_i for each of the partitions for each rearrangement to be evaluated consists of simply copying from a buffer, so it does not in principle seem very costly. But most of those reinitializations will not be needed for the evaluation of a given rearrangement: only those characters which increase their steps with the reinsertion will require that T_i for their corresponding partition is considered. These are a minimum fraction of the characters (because few characters increase their steps with the reinsertion, and only a fraction of characters are examined, as the rearrangement can be rejected as soon as the current bound is exceeded without the need to examine further characters). In the case of Källersjö et al.'s data set, the proportion of characters which increase their steps with a reinsertion is (on average) less than 0.15% of the total number of characters. Thus, most of the copying from the buffer done in the complete reinitialization of T_i values is completely unnecessary. TNT then avoids the full reinitialization by using counters, exemplified in the pseudocode of Appendix 1. With code similar to that in Appendix 1, swapping for Källersjö et al.'s matrix (divided into partitions of three positions each) proceeds at about 52.44×10^6 trees/s (two times faster than with a full reinitialization of partitions per rearrangement, or only 6.1 times slower than under equal weights).[2]

In the case of user-defined weighting functions, there is a further complication: the user defines the values of instantaneous cost C only for an integer number of steps, but average numbers of steps for partitions are usually fractions (even if TNT allowed defining values of C for fractional numbers as well, there is no way for the user to know all the possible values of average numbers of steps that will be needed during a search). Thus, TNT interpolates the values for (integer) extra steps defined by the user (this interpolation is done directly on the cumulative scores of weighted homoplasy calculated from the user-defined costs C, instead of done on the costs C themselves). The interpolation used by TNT smooths the points given by the user in such a way that the resulting curve deviates little from a straight line connecting the points (see Fig. 3 for an example); for users interested in examining in more detail the interpolation function used in TNT, the scripting expression of TNT ***intpol*** allows the individual values of interpolation for any array to be checked (in Windows, the values can be plotted as a scattergram by TNT itself, with the ***var+*** command). To avoid repeated interpolations during searches, which would be time consuming, TNT fills a lookup table when packing the data, in such a way that the possible final score values for each partition are contained in an array indexed with the absolute T_i values (clearly, the array must contain pre-calculations for each possible value of T_i, from 0 to the maximum possible). Using this approach, branch swapping of collectively weighted partitions under user-defined weighting functions proceeds at about the same speed as swapping for collectively weighted partitions under the standard weighting function (i.e. about six times slower than branch swapping under equal weights).

Taking into account missing entries

In conversations with colleagues, a criticism sometimes levelled at implied weights is that characters with many missing entries may receive artificially high implied weights, because those characters will (on average) display less homoplasy than characters for which all the entries have been scored. This may be a concern for palaeontological studies, where rarely preserved characters tend to have much larger proportions of missing entries. But it also becomes very relevant when analysing data sets which combine several genes sequenced for different subsets of taxa, to be weighted collectively with the method described in the previous section. In such a case, the characters in the gene sequenced for the fewest taxa will normally receive higher implied weights—the average homoplasy for those partitions will always be much lower than for genes sequenced in all the taxa.

The most logical way to take into account missing entries is by assuming that the unobserved entries would add homoplasy if one could effect those obser-

[2]Part of the extra time here is because with extended implied weights, the full implementation in TNT needs to consider not only partitions, but also whether each character is of fixed weight or not, and select the corresponding concavity. Those choices imply computational work in addition to the partitions themselves. Despite all the extra work, TNT in this data set can still swap under extended implied weights with numerous partitions 22 times faster than PAUP* under equal weights. Most of that speed difference comes from the union shortcuts of Goloboff (1996, 1999).

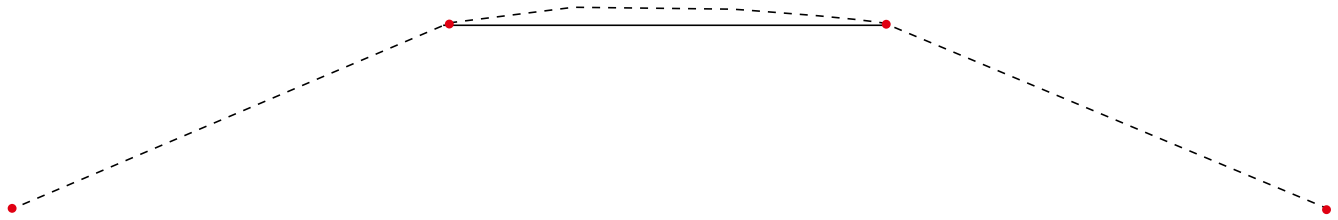

Fig. 3. Example of the smoothing using by TNT to interpolate user-defined weighting functions. See text for discussion.

vations, the more so the more entries are missing. If a number o of entries has been observed for a total of t taxa, then the observed homoplasy h would be expected to increase by an extrapolation factor $f = t/o$. Alternatively, one may expect the missing entries not to display as much homoplasy as the observed entries, but instead a fraction r of the observed homoplasy (where $0 < r \leq 1$), so that the extrapolation factor for expected homoplasy is $f = r.t/o$. For a character with no missing entries, $f = 1$; f increases as missing entries are more numerous.

The score values to assign to a character with some missing entries can be derived by considering the instantaneous cost values C for the character, in the same way as done for user-defined weighting functions. While a character without missing entries would have a cost $C_{(h)}$, the cost of adding a step to a character with extrapolation factor f and observed homoplasy h should instead be $C_{(f.h)}$; under the default weighting function of TNT this is:

$$C_{(f.h)} = \frac{fh+1}{fh+1+k} - \frac{fh}{fh+k}$$

that is, the cost of adding a step to a character with f times more extra steps than observed for the character.

To convert these instantaneous step costs into score values, $w_{(h)}$, a lookup table for the scores themselves, indexed with possible numbers of extra steps, h, can be constructed with $w_{(h)} = \sum_{i=0}^{h-1} C_{(i)}$. The difference between no homoplasy and the first step of homoplasy is defined in such a way that the first step of homoplasy always has the same cost, regardless of missing entries (i.e. regardless of the f factor). Evidently, characters with different numbers of missing entries will require different lookup tables.

A serious problem with this approach is that it is not easily applicable to continuous characters: lookup tables must be indexed with integers and extra steps in continuous characters are fractional. The similar problem for the previous case (partitions weighted according to average homoplasy, with a user-defined weighting function; see above) is not as acute, because there are fewer possible numbers of extra steps (Ti) for a partition (when the partition consists of discrete characters); the number of possible steps of homoplasy for a continuous character is potentially much larger (in fact, user-defined weighting functions are not allowed for continuous characters, precisely for that reason).

The simplest solution which allows considering continuous characters naturally, and the one adopted by TNT, is using different concavity values. For two standard weighting functions, with concavity values k_1 and k_2 (where $k_1 > k_2$), the score difference decreases with numbers of extra steps more rapidly for the lower concavity value, k_2. As the difference between no homoplasy and the first step of homoplasy should be made identical (so that homoplasy-free characters are considered equivalent), all the values for the function of k_2 should be rescaled by a factor Φ:

$$\Phi = w_{(1,k_1)}/w_{(1,k_2)}$$

In this way, the first step of homoplasy under the Φ-rescaled k_2 costs the same as the first step of homoplasy in the (reference) concavity k_1.

When a function of concavity k_2 is rescaled in this way with Φ by reference to a concavity k_1, the instantaneous costs for N steps under k_2 are approximately equal to the instantaneous costs for $N.k_1/k_2$ under k_1. This is illustrated in Fig. 4, using reference concavity $k_1 = 15$, and $k_2 = 5$ (that is, k values in ratio 3:1). That figure shows that the instantaneous cost for one step under a Φ-rescaled $k_2 = 5$ is approximately the same as the instantaneous cost of three steps under k_1. More generally, the instantaneous costs for N steps under the Φ-rescaled $k_2 = 5$ are approximately the same as the instantaneous cost for $3N$ steps under $k_1 = 15$.

In this way, when reference concavity is k_1 and the extrapolation factor of a character with missing entries is f, evaluating the characters using a Φ-rescaled function of concavity k_1/f, the desired effect (having the implied weights for the character with missing entries decrease more rapidly, in proportion to its number of missing entries) is automatically achieved.

While the correspondence is not exact (i.e. the instantaneous cost for one step under k_1 is not *exactly* the same as the instantaneous cost of k_1/k_2 steps under a Φ-rescaled k_2), it is very close. For the example

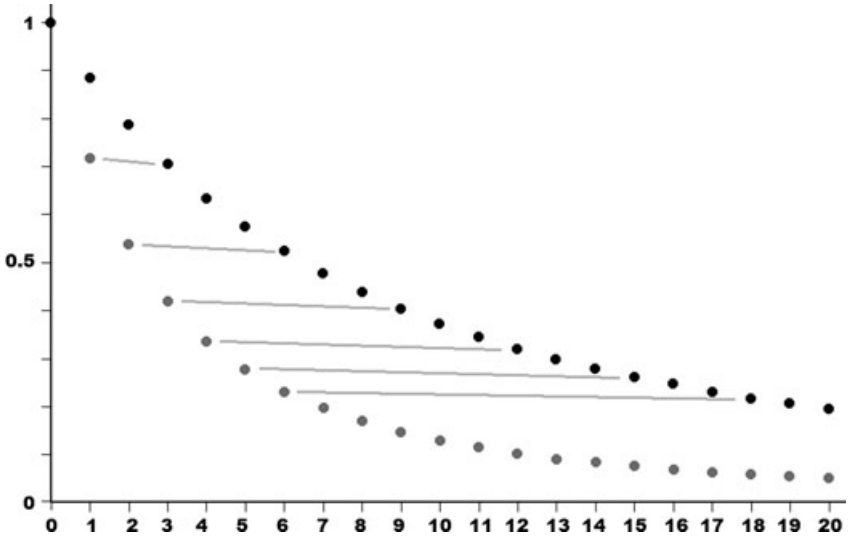

Fig. 4. Correspondence between instantaneous step costs for concavities $k = 15$ (black) and $k = 5$ (grey), with values for $k = 5$ rescaled so that $C_{(0)}$ equals $C_{(0)}$ under $k = 15$. See text for discussion.

given, concavities 5 and 15 and costs rescaled so that first extra step costs unity, the cost under $\boldsymbol{k_1 = 5}$ is 0.714 for adding a step to one extra step and under $\boldsymbol{k_2 = 15}$ it is 0.702 for adding a step to three extra steps (that is, a difference of 1.6%). As the idea of extrapolating values of homoplasy is in itself an approximation (will the currently unobserved entries have exactly as much homoplasy as that in the observed entries, or what proportion of it?), using different concavities seems acceptable for practical purposes. As noted by Goloboff et al. (2008a: 769), the same is true of implied weights; the estimation of weights from homoplasy can hardly be considered an exact procedure.

When a user-defined weighting function is in effect, TNT interpolates the costs given by the user to fill a lookup table as indicated above, based on the ***f*** factor for each character. Imprecision of the interpolation aside, this procedure does produce the exact cost ratios: for a character with extrapolation factor ***f***, the cost of adding a step to ***N*** steps of homoplasy will be exactly the same as the cost of adding a step to a character with no missing entries and ***f.N*** steps of homoplasy. Continuous characters cannot be analysed with user-defined weighting functions, so they pose no additional problems in this context.

Using the new options in TNT

Generalities

The commands that handle extended implied weighting in TNT are ***piwe*** and ***xpiwe***. The command ***piwe*** enables implied weighting (extended or not). For a data set to be analysed under extended implied weighting, it is necessary to include a ***piwe=*** command before the data set itself (i.e. before the ***xread*** command). This is because using extended implied weighting requires that some special memory buffers are allocated when reading the data set, so TNT needs to know ahead of time whether these new weighting methods will be used.

Once the data set has been read under ***piwe=***, the ***xpiwe*** command allows the definition of options for sets of characters with constant weight, sets weighted by average homoplasy, and different concavities for different characters. In addition to defining these sets, the user must activate the use of extended implied weighting, with ***xpiwe=*** (to subsequently deactivate it, use ***xpiwe–***, which goes back to standard implied weighting). Typing instead ***piwe–*** the use of implied weights is deactivated altogether (weighting characters only according to prior weights).

For all the options of extended implied weighting, a reference concavity is needed (to normalize the step costs, see previous sections); the default reference concavity is the one defined with the ***piwe*** command (***piwe=k***, where ***k*** is any number $\boldsymbol{0 < k \leq 1000}$; default is $\boldsymbol{k = 3}$).

Users should keep in mind that implied weighting, because of its finely grained precision, rarely produces many equally parsimonious trees. It is then crucial (as noted by Goloboff et al., 2008a) to conservatively take into account the effect of small changes in the analytical parameters, as in a "sensitivity" analysis. Although there is no fixed protocol for this, and different problems may require examining different aspects of the analysis, the scripting language of TNT greatly facilitates such testing.

Characters with constant weight

Sets of characters with constant weight can be defined with

xpiwe/NxF L

where ***L*** is the list of characters for which every step costs the same as ***F*** times the cost of adding one step to a character with ***N*** extra steps (under the "reference" concavity or the user-defined weighting function). ***N*** and ***F*** must be any numbers greater than zero; ***xF*** is optional (if not given, ***F* = 1**). The prior weights given to the characters still apply; if the user wishes some of the constant-weight characters to be given more weight than others, the ***ccode*** command must be used to weight those.

A possible way to determine the number of steps to establish the equivalence in constant-weight characters is by using the average numbers of steps of the rest of the characters (i.e. the characters to be subject to implied weights) on some tree (e.g. the tree resulting from a previous search). A similar (although not identical) approach was followed by Mirande (2009). With the new options in TNT, this can be done easily by first defining a partition for collective weighting (as indicated in the next section) for all other characters, and then reporting (as indicated below, under "Checking sets") the average homoplasy with the ***xpiwe&homoplasy*** command.

To remove some character(s) from the list of fixed-weight characters, use ***xpiwe ! R*** (where ***R*** is any valid list of characters).

Partitions weighted by average homoplasy

These sets are defined with ***xpiwe]*** and ***xpiwe[***. The basic syntax is ***xpiwe [L***, where ***L*** is the list of characters to be collectively weighted by average homoplasy. It is possible to define single-character partitions (although superfluous, as eqn 2 converges to eqn 1 in that case).

The list ***L*** can be optionally followed by ***=N***, case in which the set ***L*** is divided into subsets or "chunks" of ***N*** consecutive characters each. For example, in DNA sequences, if the first character in list ***L*** is a first position, then ***xpiwe[L* = 3** will weight each codon by its average homoplasy. If ***L*** is followed instead by ***/subpartition*** (where "subpartition" is in the form of, for example, "***/12:3***" or "***/1:2:3***") then the first, second, and third positions within list ***L*** are grouped as indicated (this is reminiscent of the ***charset first = 1-.\3*** option of PAUP*, used to define sets of characters to facilitate data manipulations). Not specifying 1, 2, or 3 leaves those positions outside of the defined set(s). As many sets as desired can be defined, with successive ***xpiwe*** commands; new set definitions are simply added to previous ones. A set may comprise some continuous and some discrete characters. As discussed above, user weighting functions are allowed (requiring interpolation of the score function assembled from the user-provided costs).

With a closing square bracket instead, i.e. ***xpiwe]***, there is no need to specify a list of characters—TNT automatically makes a partition out of each data block. The ***=N*** and ***/subpartition*** options are identical to the previous case in which an explicit character list is given. If a data set comprises several genes and morphology (each in its own block), morphological characters (best weighted individually) can be subsequently removed from the collectively weighted partitions with ***xpiwe ! M*** (where ***M*** is the list of morphological characters). If the morphological data block has been named (e.g. "morphology"), the removal can be done with ***xpiwe ! @morphology .*** (the period indicates "*all characters in the block*").

Checking sets

The definition of sets may be complex, and thus it is advisable to check whether the definition actually read by TNT is as expected. Typing ***xpiwe*** without arguments (i.e. followed by a semicolon) will display a table which shows the set-membership for all characters, as well as fixed-weight and individually weighted characters. Typing instead ***xpiwe****, the current sets will be saved in a format that can be subsequently read by TNT (this can be used to save currently defined options to a log file). Last, with ***xpiwe&L***, set statistics for the trees in list ***L*** can be outputted. The list ***L*** itself can be preceded by the strings ***steps***, ***score***, ***homoplasy***, or ***size*** (which is the default); when using different concavities (see below), the concavity values for each set (***kvalue***) or the cost of adding a step to an average character in the set (***cost***) can also be reported.

Setting character concavities

The use of different concavities for different characters is changed using opening and closing parentheses as argument for ***xpiwe***; ***xpiwe (*** activates the use of different concavities, while ***xpiwe)*** deactivates it.

When different concavities are activated, the opening parenthesis may be followed by an asterisk (*) or by a list ***L*** of characters.

The command ***xpiwe(**** automatically counts the missing entries in all characters, calculates for each character ***i*** the extrapolation factor f_i, and uses this to reset the concavity for each character to $k_i = k_{ref}/f_i$ (where k_{ref} is the reference concavity). The full syntax is ***xpiwe (*R <M/K***, where ***R*** is the proportion of homoplasy in the observed entries that missing entries

are assumed to have (default is 0.5), and **<*M*** is the maximum possible extrapolation factor. For example, a character with five entries out of 100 taxa with $R = 1$ would have an extrapolation factor $f = 25$, but it may be more conservative to limit f to a maximum which produces concavities that are not so wildly different. ***M*** must be equal or greater than 1; default maximum is 5. Last, ***/K*** indicates the reference concavity to use; if not specified, the one defined with the ***piwe=*** command is used (or the default, $k = 3$). The reference concavity simply serves the purpose of rescaling the first step of homoplasy, in such a way that the first step in a character with modified concavity costs the same as the first step under the reference concavity. Thus, using different reference concavities changes the score values themselves, but it does not affect tree ranking or tree selections; searches should produce the same trees.

When the default weighting function is in effect, the ratios of the concavities determined in this way will automatically take into account missing entries in each of the characters. When user weighting functions are in effect, the cost of adding a step to a character with N steps of homoplasy and concavity k_1 will be made (by interpolation) identical to the cost of adding a step to a character with $N.k_{ref}/k_1$ steps of homoplasy. In other words, the user-defined weighting function is adapted (via interpolation) for each character, depending on its k value (which in turn depends on the missing entries).

When the previous option (counting missing entries and calculating k values) is used, all TNT does is determine k values from the (current[3]) proportion of missing entries. With the manual alternative, ***xpiwe*** **(** followed by a list ***L*** of characters, the concavities are set by the user. The concavity to apply to the characters in list ***L*** must be specified after the list, with ***/K*** (if no concavity is specified, then the one defined with ***piwe=*** is used). In this case, TNT will simply honour the concavities defined by the user, regardless of missing entries. This can be used to adjust the concavities resulting from counting of missing entries. Or, additionally, TNT scripts can be used to extrapolate from missing entries in more complex ways than already provided by TNT and calculate the specific values of k to be applied to each character.

If setting concavities manually, in combination with collectively weighted partitions, keep in mind that each of the characters in a partition must have the same concavity. When using the ***xpiwe(**** option, TNT automatically considers weighting partitions, using average numbers of missing entries for the set to calculate the extrapolation factor for each of the characters in the set.

After having used these options, TNT can be reset to use, for all characters, the same concavity (or the same user-defined weighting function) with ***xpiwe)***.

Checking character concavities

To examine the concavities for each of the characters, they can be listed with ***piwe&***, which displays all concavities in table form. If ***piwe&*** is followed by a list ***L*** of characters, then the cost of adding a step to different numbers of extra steps is displayed, for each of the characters in list ***L*** (this is the equivalent to what ***piwe/*** does for the standard weighting function).

Weighting character state transformations

A previous extension of Goloboff (1993) had been presented by Goloboff (1997), as "self-weighted optimization", a method to dynamically weight individual character state transformations instead of entire characters. The methods described in this paper consider only standard implied weights, i.e. weighting of whole characters.

One might envisage circumstances where it would be desirable to combine self-weighted optimizations with the methods described here. However, the computational cost of combining self-weighted optimization with multi-character weighting seems enormous: this would mean that the numbers of possible character state transformations should be averaged over all characters in the partition, thus requiring that the character-state reconstructions (see Goloboff, 1997, for the problems associated with finding optimal reconstructions under self-weighted optimization) be done simultaneously for all the characters in the partition. The missing entries could perhaps be handled more easily, but not necessarily in a more meaningful way; this would require that (in a given reconstruction) the number of $i \rightarrow j$ transformations in the observed taxa is extrapolated to the $i \rightarrow j$ transformations expected when entries are completed, but without extrapolating to $j \rightarrow i$ transformations. Whether the evidence presented by each of the possible transformations between states is enough to establish those conclusions is arguable; in most cases, it seems that such type of extrapolation would be squeezing out of the data more than the data can give—a serious hair splitting, in other words. Thus, restricting the extensions proposed here to standard implied weighting seems acceptable.

[3]Numbers of missing entries can change because the data set can be edited in TNT; the concavities are not automatically changed when editing the data set; doing so requires executing a new ***xpiwe(**** command.

Discussion and conclusions

The present paper shows that a number of improvements to the original method of Goloboff (1993) are still possible. Some of the extensions presented in this paper follow from the discussion in Goloboff et al. (2008a), which suggested that a universal application of standard implied weights to data sets combining morphology and sequences may not be ideal.

The decision to use extended implied weights, just like the decision to use implied instead of equal weighing, could be based on different arguments. Two possible ways of arriving at a decision are (1) arguments about which method is more likely to provide the correct answer (either empirical tests, using some measure of improvement for the results, or based on simulations), and (2) methodological, based on consideration of the rationale and justification for each type of analysis.

Existing empirical tests (e.g. Goloboff, 1997; Goloboff et al., 2008a, and presented below) seem to point towards the superiority of weighting over equal weights, but none of those tests is unambiguous—they are always subject to interpretation. In addition, while empirical tests so far point to some advantage of differential over equal weighting, they do not seem to indicate significant differences among alternative forms of homoplasy-based weighting.

Differences in bootstrap values with and without weighting have sometimes been used to argue against homoplasy weighting (Källersjö et al., 1999; Kluge, 2005), but when looked at in more detail seem to point towards the superiority of (properly) weighted analyses (Goloboff et al., 2008a). However, as it is possible to construct data sets for which the addition of random characters increases the average bootstrap frequencies (see Goloboff et al., 2008a), increased bootstrap frequencies are obviously no unequivocal indication of improvement. Thus, while it is important to know that these quality measures are in fact improved when morphological data sets are analysed under implied weights (Goloboff et al., 2008a), it would be desirable to obtain additional evidence that analyses are improved with weighting.

Testing whether the recovery of already established taxonomic groups is improved under weighting is not completely conclusive, either. First, it relies on the idea that previous taxonomic groups are correct. And second, the improvements obtained in the data sets tested so far are hard to interpret. As example, consider the rate of approximate recovery of taxonomic groups (see Goloboff et al., 2009: 215; Goloboff and Catalano, 2012: 509) for all the *rbcL* data contained in GenBank (~13 000 sequences, downloaded and handled with Gb→TNT, Goloboff and Catalano, 2012). For weighted analyses, the gene (a coding sequence with a very clean alignment) was divided into fragments of six codons (18 contiguous positions). If some regions of the sequence (e.g. those involved in structure or active sites) provide better evidence of relationships, then this might be expected to improve results. When the *rbcL* gene is weighted in such a manner, the number of taxonomic groups is improved relative to equal weights (of 2891 total groups defined for this taxon set, equal weights misses 1033 and extended implied weighting 944, or an improvement of almost 10%). As observed by Goloboff et al. (2008a) for other large molecular data sets, application of standard implied weighting (with $\boldsymbol{k = 100}$) produced inferior results, recovering fewer taxonomic groups than equal weights. But the improvement obtained by weighting six-codon partitions is called into question by the observation (in a smaller *rbcL* data set, corresponding to all magnoliids) that the recovery of taxonomic groups is also consistently increased when the gene is divided into *random* fragments of 18 non-contiguous positions—this is probably because under implied weights, the chances of exact ties between trees are much lower (due to the finely grained scoring function), thus leading to selecting fewer trees and a better resolution. The improvement obtained by weighting contiguous segments of the gene is therefore somewhat inconclusive: any method which selects a reduced subset of the trees supported under equal weights (regardless of how that selection is done) may improve the result, but the rationale for multi-codon weighting is that the degree of phylogenetic correlation (and thus the reliability) may have a positional dependence along the sequence. As that positional dependence is absent in the random fragments (which also consistently recover many groups), the improvement observed for the six-codon set weighting may therefore not be due to the positional dependence, but instead to other factors. This points to a difficulty with empirically testing the methods proposed here: the methods give the user an enormous range of possibilities on how to weight the data, but not many data sets exist today which combine morphology and sequences at a level of detail sufficient to test all available alternatives.

In the end, while some evidence (improved group supports, better recovery of taxonomic groups) suggests that implied weighting (as well as the extended version presented here) produces better results, this evidence is by no means totally conclusive. Perhaps what is in itself misguided is the expectation to solve the problem by deciding which method provides better results. With the boom, in recent years, of maximum-likelihood and statistically based methods of phylogenetic reconstruction, the problem of method choice is often treated in an overly reductionistic way—much as if phylogenetics consisted of no more than black and white balls in boxes, and as if the only possible justifi-

cation for a method were its probability of obtaining "truth", instead of more methodological or conceptual considerations. A good example of such an attitude is Anisimova et al.'s (2011) approach to group support. For many authors (e.g. Farris et al., 1996; Farris, 2002; Goloboff et al., 2003) support is a concept involving the strength of a pattern; of course, probabilistic reasoning can be used to try to detect that pattern, as in resampling, but that does not *define* the support: the "support" is an abstraction which is not definable in purely statistical terms. In contrast, Anisimova et al. think an ideal method for measuring group support should be that which produces support values, $\boldsymbol{S}$, such that groups of support $\boldsymbol{S}$ are incorrect (not present in the true tree) with a frequency no greater than $\mathbf{1} - \boldsymbol{S}$. No concept other than the probability of recovering the correct tree enters their definition. Of course, such "support" can be tested only by reference to simulations, as if simulations could provide conditions that even remotely approach all the hidden and unknown factors operating in the real world. Moreover, the conditions they simulate are quite favourable for their method, so it performs well, and hence their test could hardly be called "empirical". But a worse problem is that the approach considers that how "good" a method is depends exclusively on its probability to provide the "right" answer, regardless of its rationale and regardless of the factors it takes into account. Another prime example of this attitude is Nguyen et al.'s (2012) "MRL" supertree method, which consists of creating supertrees by analysing the matrix representing the input trees with two-state maximum-likelihood under RaxML (Stamatakis, 2006). The authors find that (in simulations) they obtain supertrees closer to the model tree with MRL; this they find "surprising", because there is no theoretical reason whatsoever why using a molecular model to analyse variables representing groups should produce better results. Despite their surprise, they think their MRL method is "promising"—even when they have no idea of why, in their simulations, it fares better than parsimony supertrees (which are already very hard to justify logically; see Goloboff and Pol, 2002; Goloboff, 2005).

It is with this same attitude that weighting methods are often considered with skepticism: it is feared that they might provide the wrong answer. Existing empirical tests seem to indicate otherwise, but as they are never conclusive and are always subject to interpretation, the fear persists.

Methodological considerations, instead of the chances of providing the true tree, must ultimately provide much of the justification for method choice. This continues being as true today as it was when Farris (1969) proposed his successive weighting method. In this regard, Goloboff et al. (2008a) showed that most of the objections to character weighting (e.g. that it is based on strong assumptions; that it assumes that a character is equally reliable in different parts of the tree; that because every extra step counts as an ad hoc hypothesis, what has to be minimized in phylogenetics is merely extra steps) are simply equivocations or also apply to equal weights. Forcing all characters to be equally reliable is a stronger assumption than allowing some characters to be more reliable than others; every parsimony method, not just implied weights, assumes that a character is equally reliable in all parts of the tree; some ad hoc hypotheses are more disturbing than others, so there is no reason to consider all ad hoc hypotheses as identical. Thus, once unjustified criticisms are weeded out, character weighting appears as an entirely acceptable practice, and the uncritical acceptance of equal weights has no basis other than tradition. Extending implied weighting so that it takes into account additional aspects of the data (such as the homoplasy expected to also occur in missing entries, the possibility of differences in reliability along parts of a sequence, or the fact that the reliability of the entire gene may be assessed more confidently than that of its individual positions) can be justified along the same lines.

Acknowledgments

I thank several colleagues for many discussions on weighting, along the years: Salvador Arias, James Carpenter, Santiago Catalano, Jan De Laet, James Farris, Marcos Mirande, Diego Pol, and Claudia Szumik. Support from the CONICET (PIP 112 201101 00687) and FONCyT (PICT 01314) is also acknowledged.

References

Anisimova, M., Gil, M., Dufayrad, J.-F., Dessimoz, C., Gascuel, O., 2011. Survey of branch support methods demonstrates accuracy, power, and robustness of fast likelihood-based approximation schemes. Syst. Biol. 60, 685–699.

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

Farris, J., 2002. RASA attributes highly significant structure to randomized data. Cladistics 18, 334–353.

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

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

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

Goloboff, P., 1997. Self-weighted optimization: tree searches and character state reconstructions 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., 2005. Minority rule supertrees? MRP, compatibility, and minimum flip may display the least frequent groups Cladistics 21, 282–294.

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., Pol, D., 2002. Semi-strict supertrees. Cladistics 18, 514–525.
Goloboff, P., Farris, J., Källersjö, M., Oxelman, B., Ramírez, M., Szumik, C., 2003. Improvements to resampling measures of group support. Cladistics 19, 324–332.
Goloboff, P., Carpenter, J., Arias, S., Miranda, D., 2008a. Weighting against homoplasy improves phylogenetic analysis of morphological data sets. Cladistics 24, 758–773.
Goloboff, P., Farris, J., Nixon, K., 2008b. TNT, a free program for phylogenetic analysis. Cladistics 24, 774–786.
Goloboff, P., Catalano, S., Mirande, M., Szumik, C., Arias, S., Källersjö, M., Farris, J., 2009. Phylogenetic analysis of 73,060 taxa corroborates major eukaryotic groups. Cladistics 25, 211–230.
Källersjö, M., Albert, V.A., Farris, J., 1999. Homoplasy increases phylogenetic structure. Cladistics 15, 91–93.
Kluge, A. 2005. What is the rationale for "Ockham's Razor" (a.k.a. parsimony) in phylogenetic inference? In: Albert, V. (Ed.), Parsimony, Phylogeny, and Genomics. Oxford University Press, Oxford, pp. 15–42.
Mirande, M., 2009. Weighted parsimony phylogeny of the family Characidae (Teleostei: Characiformes). Cladistics 25, 574–613.
Nguyen, N., Mirarab, S., Warnow, T., 2012. MRL and SuperFine+MRL: new supertree methods. Algorithms Mol. Biol. 7, 1–13.
Stamatakis, A., 2006. RAxML-VI HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics 22, 2688–2690.
Swofford, D., 2002. PAUP*: Phylogenetic Analysis Using Parsimony (* and other methods), ver. 4. Sinauer Associates, Sunderland, MA.
Wheeler, W., Aagesen, L., Arango, C., Faivovich, J., Grant, T., D'Haese, C., Janies, D., Smith, W., Varón, A., Giribet, G. 2006. Dynamic Homology and Phylogenetic Systematics: A Unified Approach Using POY. The American Museum of Natural History, New York, NY.

Appendix 1

C-style pseudo-code illustrating the use of counters to avoid repeated reinitialization of partition totals for each rearrangement. The main variables used are: div_Total, an array (ints) with total steps of homoplasy for the divided tree (filled in when optimizing divided tree); quick_Total, an array with total steps of homoplasy for quick evaluation of a rearrangement; div_Score and quick_Score, arrays similar to div_Total and quick_Total (floats); numinserts, a counter (int), used to reset last_check (int) to avoid overflows (for each clipping, it is reinitialized to 0, and increased for each reinsertion point); last_check [n], an array (ints) used to indicate the last reinitialization of quick_Total and quick_Score performed for partition n. This pseudo-code is very simplified, not considering in detail multiple trees or trees of better score.

The pseudo-code uses counters to keep track of when the last reinitialization for a given partition was done; the reference counter (numinserts) is increased for each rearrangement (line 7 of do_tbr_swapping), and reinitialized again every 10^9 rearrangements (line 8), to avoid bit overflow in the counters (32-bit ints). As the characters in the matrix are sequentially examined (in the loop of lines 10–24) and a character which increases steps is found (line 11), the counter for the partition to which the character belongs is compared with the reference counter. If these are different (line 13), it means that the partition has not been reinitialized during the current evaluation; then the partition is reinitialized on the spot (lines 14 and 15) and the counter for the partition is updated (line 16). The counter for the partition needs to be updated because checking subsequent characters for the same rearrangement will require that the T_i values are *not* reinitialized—they must instead be the ones modified by the step increase in the preceding character(s).

```
functioncalibrate_all_partitions()
  {
1        numinserts = 0 ;
2        for (n = 0 ; n < all partitions ; ++ n) {
3           quick_Total[ n ] = div_Total [ n ] ;
4           quick_Score[ n ] = div_Score [ n ] ;
5           last_check[ n ] = 1 ;
6           }
  }   // end calibrate_all_partitions

functiondo_tbr_swapping()
  {
1     total_rearrangs = 0 ;
2     for (i = 0 ;i< all clips ; ++ i) {
3        div_score = optimize_divided_tree() ;
4        calibrate_all_partitions() ;
5        for (j = 0 ; j < all rerootings ; ++ j) {
6           for (k = 0 ; k < all reinsertions ; ++ k) {
7              ++ numinserts ;
8              if (numinserts == 10^9)calibrate_all_partitions() ;
9              new_score = div_score ;
10             for (c = 0 ; c < total num chars ; ++ c)
11                if (inserting i to k increases steps for char. c) {
12                   pnum = partition_of_char[ c ] ;
13                   if (last_check[ pnum ] != numinserts) {
14                      quick_Total[ pnum ] = div_Total [ pnum ] ;
15                      quick_Score[ pnum ] = div_Score [ pnum ] ;
16                      last_check[ pnum ] = numinserts ; }
17                new_score -= quick_Score[ pnum ] ;
18                ++ quick_Total[ pnum ] ;
19                quick_Score[ pnum ] =
20                    quick_Total[ pnum ] /
21                    ((quick_Total[ pnum ]/Part_Size [ pnum ]) + k) ;
22                new_score += quick_Score[ pnum ] ;
23                if (new_score>= lowest_score)break ;
24               }
25             ++ total_rearrangs ;
26             if (new_score<lowest_score)update_all_tree_buffers() ;
27             }
28          }
29       }
30    }
  }   // end do_tbr_swapping
```