An efficient and accurate distance based algorithm to reconstruct tandem duplication trees

Olivier Elemento\textsuperscript{1, 2} and Olivier Gascuel\textsuperscript{1}

\textsuperscript{1} Département d’Informatique Fondamentale et Applications, LIRMM, 161 rue Ada, 34392, Montpellier, and \textsuperscript{2} IMGT, the international ImMunoGeneTics database http://imgt.cines.fr, Laboratoire d’Immunogénétique Moléculaire (LIGM), Université Montpellier II, UPR CNRS 1142 , 141, rue de la Cardonille, 34396, Montpellier cedex 5, France

Received on April 8, 2002; accepted on June 15, 2002

ABSTRACT

The problem of reconstructing the duplication tree of a set of tandemly repeated sequences which are supposed to have arisen through unequal recombination, was first introduced by Fitch (1977, \textit{Genetics}, 86, 93–104), and has recently received a lot of attention. In this paper, we describe DTSCORE, a fast distance based algorithm to reconstruct tandem duplication trees, which is statistically consistent. As a cousin of the ADDTREE algorithm (Sattath and Tversky, 1977, \textit{Psychometrika}, 42, 319–345), the raw DTSCORE has a time complexity in $O(n^5)$, where $n$ is the number of observed repeated sequences. Through a series of algorithmic refinements, we improve its complexity to $O(n^4)$ in the worst case, but stress that the refined DTSCORE algorithm should perform faster with real data. We assess the topological accuracy of DTSCORE using simulated data sets, and compare it to existing reconstruction methods. The results clearly show that DTSCORE is more accurate than all the other methods we studied. Finally, we report the results of DTSCORE on a real dataset.

Supplementary information: http://www.lirmm.fr/w3ifa/MAAS/
Contact: gascuel@lirmm.fr

INTRODUCTION

Tandemly repeated DNA sequences consist of two or more adjacent copies of a stretch of DNA, together forming an array of consecutive repeated sequences. They arise from tandem duplication, in which a sequence of DNA (which may itself contain several repeats) is transformed into two adjacent copies. Since copies are then free to evolve independently and are likely to undergo additional mutation events, they become approximate copies over time. Unequal recombination is widely viewed as the predominant biological mechanism responsible for the production of tandemly repeated sequences (Ohno, 1970; Smith, 1976; Fitch, 1977; Jeffreys and Harris, 1981; Elemento \textit{et al.}, 2001, 2002).

Gene duplication (in tandem or not) is one of the most important evolutionary mechanisms for producing genes with novel functionalities (Ohno, 1970). Unravelling the pattern of duplications that has given rise to these duplicated genes, i.e. reconstructing their duplication history, would be very beneficial to the scientists studying their function and evolution.

The problem of reconstructing the duplication history of tandemly repeated sequences was first considered by Fitch (1977). It has not received much attention until recently, probably due to the lack of available repeated sequence data, and also because there has been no dedicated computer program available to reconstruct duplication histories from sequence data.

Indeed, the reconstruction strategy presented in (Fitch, 1977), and also in (Elemento \textit{et al.}, 2001, 2002), consists in using traditional phylogenetic reconstruction algorithms to reconstruct (near) optimal trees, without restriction to duplication trees, and to check whether these best trees are valid duplication trees. While this strategy is fast and simple, it is not guaranteed at all to find a duplication tree, as illustrated in (Fitch, 1977).

However, several specific algorithms which take into account the ordered nature of tandemly repeated sequences have been recently described in the literature. In (Benson and Dong, 1999; Tang \textit{et al.}, 2001; Jaitly \textit{et al.}, 2001), the authors provide reconstruction algorithms based on the parsimony principle, but these are limited either to the special case where only single copy duplications (i.e. when the duplicated fragment always contains a single basic copy) occurred, or to the analysis of short tandem repeats (minisatellites). In Elemento \textit{et al.} (2001, 2002), we presented the DTEXPLORE algorithm, which performs an exhaustive exploration of the space of duplication trees, and selects the best ones according to the parsimony criterion. Although this procedure is
guaranteed to find the optimal trees, it is relatively slow
and limited to small data sets, i.e. \( n < 15 \), where \( n \) is
the number of taxa.

In (Tang et al., 2001), the authors proposed a distance
based method, called the WINDOW method, that uses
an agglomeration scheme similar to UPGMA (Sneath and
Sokal, 1973) and NJ (Saitou and Nei, 1987). The
WINDOW method has the advantage of being very simple
and fast, but the cost function used to judge potential
duplications supposes that the sequences followed a
molecular clock mode of evolution. It is well known that
some multigene families, such as the immunoglobulin
genomes (Ota and Nei, 1994) for example, do not follow
this mode of evolution: some duplicated genes become
pseudogenes in the course of evolution and then evolve
at a much faster rate than the other copies.

In this paper, we describe an efficient and accurate
distance based algorithm, called DTSCORE, to deal with
the problem of reconstructing tandem duplication trees.
This algorithm belongs to the scoring method family
(Barthélemy and Guénoche, 1991), whose most famous
representative is the popular ADDTREE (Sattath and
Tversky, 1977) algorithm. It is statistically consistent
(Felsenstein, 1978; Atteson, 1999), i.e. the quality of
the reconstruction improves as the sequences get longer,
and can be used with sequences that do not follow the
molecular clock mode of evolution. As a distance based
method, our algorithm can use sophisticated sequence
evolution models, such as those taking into account
heterogeneous rates of evolution among sites (Golding,
1983), and obviates some fallbacks of the parsimony
criterion: inconsistency, long branch attraction (Felsenstein,
1978). In its basic form, DTSCORE is an \( O(n^5) \) algo-
rithm, but we provide here two algorithmic refinements to
lower its time complexity to \( O(n^3) \) in the worst case, and
show how the refined DTSCORE should perform even
better with real data. The computational efficiency of this
method makes it suitable for heavy statistical reliability
analysis of the reconstructed histories, such as the boost-
rap procedure (Felsenstein, 1985). Using simulations, we
show that DTSCORE has better topological accuracy than
every other tested reconstruction method, even when the
molecular clock hypothesis is respected.

**DUPLICATION MODEL**

The duplication model we assume in this paper, introduced
by Fitch (1977) in the first place, is closely based on the
unequal recombination process, which we suppose to be
the only evolutionary mechanism (except point mutations)
acting on the duplicated sequences.

Let \( O = (1,2,\ldots,n) \) be the ordered set of sequences
representing a locus, as it can be observed now. Initially
containing a single copy, the locus has grown through a
series of consecutive duplications. As shown in Figure 1a,
a duplicated fragment may only contain a single repeat,
in which case we say that the duplication event is a 1-
duplication. When the duplicated fragment contains 2, 3
or \( k \) repeats, we call the duplication event a \( 2-,3- \) or \( k-
duplication.

Under this duplication model, a duplication history is a
tandem tree with \( n \) labelled and ordered leaves, in which
internal nodes correspond to duplication events. In a real
duplication history (Figure 1a), the time intervals between
consecutive duplications are completely known, and the
internal nodes are ordered from top to bottom according
to the moment they occurred in the course of evolution.
However, in the absence of a molecular clock mode of
evolution (which is most often the case), both the order
between the duplication events of two different lineages
and the root location are impossible to recover from the
sequences. In this case, we are only able to infer a
duplication tree (Figure 1b), i.e. an unrooted phylogeny
with ordered leaves, whose topology is compatible with at
least one duplication history.

Recovering the position of the root can sometimes be
achieved, through the use of rooting procedures (out-
groups, midpoint, etc.), and creates a partially ordered
duplication history (Figure 1c), i.e. a duplication history
in which the duplication events are partially ordered. In
Elemento et al. (2001, 2002), we show that rooting a
duplication tree is different from rooting a phylogeny:
the root of a duplication tree necessarily lies on the tree path
between the most distant repeats on the locus; moreover,
it can be shown (Gascuel et al., 2002) that, assuming
that the duplication trees have uniform distribution, the
expected number of possible positions for the root is
only equal to 2. For example, if a 2-duplication happens
immediately after the initial 1-duplication in a duplication
history, the root must be located ‘above’ the 2-duplication,
and there is only one possible root location.

A cherry is a pair of leaves \( \{g,d\} \) separated by a
single internal node in a duplication tree \( T \). The
outcome of a terminal \( k \)-duplication in \( T \) defines what
Tang et al. (2001) called a window. A window is a set
\( \{g_1,g_2,\ldots,g_k,d_1,d_2,\ldots,d_k\} \) of \( 2k \) adjacent copies in
\( O \), such that every \( \{g_i,d_i\} \) with \( 1 \leq i \leq k \) is a cherry
in \( T \). For example, the windows in Figure 1 are \( (1,2,\)
\(4,5,6,7) \) and \( (8,9) \). In (Tang et al., 2001; Elemento et
al., 2001, 2002), the authors described a very simple
window-based algorithm for determining whether a
given phylogeny \( T \) with ordered leaves is a duplication

unrooted phylogenies), is a recursive procedure which
progressively reduces \( T \) by deleting (agglomerating) the
cherries that belong to recognized windows. The cherries
(windows) are deleted until: (a) \( T \) has been reduced to a
tree with 2 or 3 leaves, meaning that it constitutes a valid
duplication tree, (b) it cannot go further, in which case T cannot be a duplication tree.

AN ALGORITHM FOR RECONSTRUCTING TANDEM DUPLICATION HISTORIES

We describe here the DTSCORE algorithm to reconstruct a duplication tree from the matrix of pairwise evolutionary distances between sequences. DTSCORE follows the same agglomerative process as PDH, i.e. it reconstructs a duplication tree by re-creating and agglomerating the cherries that belong to recognized windows. In this section, we present an initial version of DTSCORE, and describe its basic principles and properties. In the following section, we introduce two algorithmic improvements to decrease its computational time complexity.

The DTSCORE algorithm

Let $O = (1, 2, 3, \ldots, n)$ be the initial ordered set of copies, and $D$ the distance matrix between the $n$ copies in $O$. The first step of the DTSCORE algorithm consists in computing the score for every pair of copies, and storing each score in an $n \times n$ matrix $S$. As in the ADDTREE algorithm, computing the score of each pair relies on the four-point condition (Zaretskii, 1965; Buneman, 1974) and can be summarized as follows: if we consider a quartet of different copies $[i, j, k, l]$ and $D$ as a tree distance among copies, the smallest sum among $(D(i,j) + D(k,l))$, $(D(i,k) + D(j,l))$, and $(D(i,l) + D(j,k))$ defines the two external pairs. For example, if $(D(i,j) + D(k,l))$ is the smallest sum, then the external pairs are $[i, j]$ and $[k, l]$. However, in practice, $D$ is only close to a tree distance. If $(D(i,j) + D(k,l))$ is the smallest sum, then $[i, j]$ and $[k, l]$ are good candidates as external pairs in the quartet $[i, j, k, l]$. The score of a pair $[i, j]$, denoted $S(i,j)$, is then simply the number of times the pair $[i, j]$ is seen as a good candidate.

As in PDH, DTSCORE does not operate on single pairs, but on windows. At each step of DTSCORE, the window $W = (g_1, \ldots, g_k, d_1, \ldots, d_k)$ whose fitness is optimal is selected to be agglomerated. The fitness of a window of $2k$ adjacent copies may be defined as the average score over the $k$ pairs forming the window. In this case, we have the following formula:

$$AVG(W) = \frac{1}{k} \sum_{i=1}^{k} S(g_i, d_i)$$

The problem with this window fitness function is that the low score of poorly supported pairs can be offset by several other well supported pairs within the same window. To overcome this problem, a simple solution is to take the minimum score over every pair covering the window. In this case, the window fitness function becomes:

$$MIN(W) = \min_{i=1}^{k} S(g_i, d_i)$$

We will see how this window fitness function is used as the basis for an efficient refinement to the DTSCORE algorithm.

Finally, it is also possible to use both the $MIN$ and $AVG$ fitness functions together to form what we call the $MINAVG$ fitness function. In this fitness function, the $MIN$ function is used first, and the $AVG$ function is only used to decide between $W$ and $W'$, when $MIN(W') = MIN(W)$.

Fig. 1. (a) duplication history, (b) duplication tree, and (c) partially ordered duplication history.
Reconstructing tandem duplication trees

part of a complete window. For example, the score $S(1, r)$, where $r$ is the number of copies in the running matrix, is always useless. More generally, only the scores $S(i, i + k)$ with $i \geq 1, i + k \leq r$, and $k \leq r/2$ need to be calculated.

After selection of the optimal window, an ordered set of $k$ nodes $C = (c_1, \ldots, c_k)$ is created, and every $g_i$ and $d_i$ are linked to $c_i$ in $T$. $(g_1, \ldots, g_k, d_1, \ldots, d_k)$ is then replaced by $(c_1, \ldots, c_k)$ in $O$. The distance from $c_i$ to every other copy $p$ in $O$ is calculated in the following way: $D(c_i, p) = |D(g_i, p) + D(d_i, p)|/2$. The distance from $c_i$ to $c_j$ is given by $D(c_i, c_j) = |D(g_i, d_j) + D(g_j, d_i) + D(d_i, d_j) + D(g_i, g_j)|/4$.

Depending on the size of the last window that has been agglomerated, DTSCORE terminates when $O$ contains 2 or 3 copies. The DTSCORE algorithm is described in Figure 2.

Note that the branch lengths of the resulting tree are not estimated with DTSCORE, but can be obtained by least-squares methods, such as (Gascuel, 1997).

Properties

Some of the interesting properties of DTSCORE are inherited from ADDTREE. The most important one is certainly that DTSCORE is consistent, i.e. it always recovers the correct topology when $D$ is a tree distance.

In practice, $D$ is estimated from sequences using a stochastic model of evolution, e.g. (Jukes and Cantor, 1969; Kimura, 1980). Assuming this model as correct, we have consistent evolutionary distance estimators, which have consistent evolutionary distance estimators, which can be uniquely represented by the true phylogeney. Therefore, assuming that the sequence evolution model is correct, DTSCORE is statistically consistent, just as ADDTREE is (Atteson, 1999). When the sequence length increases, the probability that DTSCORE recovers the correct duplication tree converges to 1.0. This is due to the fact that when the noise level in the distance matrix $D$ is smaller than half the length of the shortest branch of this tree, then DTSCORE will reconstruct the correct tree. This property, established for ADDTREE when only considering pairs of copies, remains valid when using $AVG$, $MIN$, or $MINAVG$ for windows. Indeed, when the noise level is smaller than half the minimum branch length in the true tree, the four-point rule systematically designates correct external pairs. Then, the score of any correct pair is equal to its maximum value, i.e. $(r - 2)(r - 3)/2$, while the incorrect pairs have a strictly lower score. The score of any correct window is then equal to $(r - 2)(r - 3)/2$, whichever of $MIN$, $AVG$, or $MINAVG$ is used, while the score of an incorrect window is strictly lower. It must be noted that this consistency property for duplication trees does not hold with the NJ selection criterion, nor with the UPGMA-like criterion used in the WINDOW method (Tang et al., 2001), when dealing with sequences that do not follow a molecular clock mode of evolution.

At each step, there are $\sum_{k=1}^{r/2} (r - k)$ different scores to compute. Although this number is smaller than the number of scores ADDTREE calculates at each step, it is still in $O(r^k)$. Computing $S(i, j)$ is done by considering every other pair of copies $(y, z)$. Since we have $1 \leq y < z \leq r$, there are $r(r - 1)/2$ pairs contributing to $S(i, j)$. Computing the contribution of $(y, z)$ to $S(i, j)$ is done in $O(1)$ time, thus calculating $S(i, j)$ takes $O(r^3)$ time, and computing the score for every needed $S(i, j)$ requires $O(r^4)$ time. Supposing that we agglomerate one cherry at a time, the basic DTSCORE algorithm requires $O(n^3)$ time and $O(n^2)$ space (the size of $S$ and $D$).

REFINING THE DTSCORE ALGORITHM

First algorithmic refinement

If we take $MIN$ as the window fitness function, it is often not necessary to compute every score $S(i, i + k)$ for a given $k > 1$. Let $F^*$ be the fitness of the optimal window found so far. Now consider the window starting at $i$ and containing $2k$ adjacent copies. We first compute the score $S(i, i + k)$. If $S(i, i + k) \leq F^*$, then it is clearly neither useful nor possible to compute the scores of the other pairs in the same window, nor to consider the fitness of the other windows that include the pair $(i, i + k)$. Therefore we can ‘jump’ to the pair $(i + k, i + 2k)$. If $S(i, i + k) > F^*$, we compute the score of the neighbor pairs of $(i, i + k)$ and seek a complete $k$-window whose fitness is greater than $F^*$. If successful, we update $F^*$, otherwise we jump to the next pair as described above. Note that this refinement can also be used with the $MINAVG$ fitness function, but we will jump to the pair $(i + k, i + 2k)$ only when $S(i, i + k) \lt MIN^*$, where $MIN^*$ is the optimal $MIN$ fitness found so far.

This refinement modifies the total number of scores $S(i, j)$ to be computed, but it does not improve the worst case complexity of the DTSCORE algorithm. However, in practice, it considerably reduces the number of scores $S(i, j)$ to be computed (Table 1). Let us consider the special case where the duplication history to be reconstructed contains only single duplications, which often seems to be the case with real data (Fitch, 1977; Tang et al., 2001; Elemento et al., 2001, 2002). Suppose that DTSCORE always finds one of the $S(i, i + 1)$ scores to be optimal (which is likely to be the case if the correct duplication history only contains 1-duplications). In this case, the number of scores to be calculated at each step is the sum of the following terms: $(r - 1)$ scores $S(i, i + 1)$, approximately one half of $(r - 2)$ scores $S(i, i + 2)$, one third of the $(r - 3)$ scores $S(i, i + 3)$, and so on. This sum can also be expressed as $\sum_{k=1}^{r/2} \frac{r - k}{k}$, and the total number
BASIC DTSCORE algorithm

input an \( n \times n \) distance matrix \( D \);
let \( S \) be an \( n \times n \) matrix;
let \( O = \{1, 2, 3, \ldots, n\} \), \( r = |O| \);
while \( r \) is greater than 3 do
    compute \( S(i, i + k) \) for every pair \( \{i, i + k\} \) with \( 0 < k \leq r/2 \), \( i \geq 1 \), \( i + k \leq r \);
    find the window \( W = \{g_1, \ldots, g_k, d_1, \ldots, d_k\} \) from \( O \) with maximal fitness;
    define \( k \) new nodes \( C = \{c_1, \ldots, c_k\} \) and link \( g_i \) and \( d_i \) to \( c_i \) (\( 1 \leq i \leq k \));
    compute distances from \( c_i \) to every other copy \( p (\notin C) \) with \( D(c_i, p) = \frac{[D(g_i, p) + D(d_i, p)]/2}{1} \);
    and between \( c_i \) and \( c_j \) with \( D(c_i, c_j) = \frac{[D(g_i, d_j) + D(d_i, c_j) + D(g_i, g_j) + D(d_i, d_j)]/4}{1} \);
    replace \( W \) by \( C \) in \( O \) and \( D \);
    \( r \leftarrow r - k \);
end while

link the 2 or 3 remaining copies together and output \( T \);

Fig. 2. Basic DTSCORE algorithm.

Second algorithmic refinement

In the basic DTSCORE algorithm, computing the score \( S(i, j) \) is done from scratch and therefore always requires considering every other pair \( \{y, z\} \). The second refinement is based on the fact that some of the scores in \( S \) do not need to be calculated from scratch. At step \( t \), let \( \{g_1, \ldots, g_k, d_1, \ldots, d_k\} \) be the optimal window, and let \( \{c_1, \ldots, c_k\} \) be the newly created copies. We agglomerate one cherry at a time; so let \( \{g, d\} \) be the cherry with root \( c \) currently being agglomerated. Due to this agglomeration, some of the previously computed \( S(i, j) \) scores can be updated, while some others must be computed from scratch:

(a) When \( i \neq g, d \) and \( j \neq g, d \), \( S(i, j) \) can be updated by considering the contribution of \( \{g, d\} \), \( \{g, z\} \), \( \{d, z\} \), and \( \{e, z\} \) where \( z \) is any copy \( \neq i, j, g, d, e \). The removal of \( g \) and \( d \) induces that the contribution of \( \{g, z\} \), \( \{d, z\} \), and \( \{g, d\} \) may decrease \( S(i, j) \) by 1, while the addition of \( \{e, z\} \) may or may not increase \( S(i, j) \) by 1.

(b) Every \( S(c, p) \) score requires to be computed from scratch.

To analyze the improvement induced by this second refinement, we assume that the first refinement is not used. At the first step, we have to compute \( O(n^2) \) scores, and each score is computed in \( O(n^3) \) time. So, the first step has \( O(n^4) \) time complexity. At the subsequent steps, it is easily seen that each time a pair is agglomerated, we must (a) update \( O(r^2) \) scores in \( O(r) \) time, and (b) compute from scratch \( O(r) \) scores in \( O(r^2) \) time. So each non-initial step requires \( O(r^3) \) time, and the total time complexity is \( O(n^4) \), instead of \( O(n^5) \) in the basic algorithm.

Note that this algorithmic refinement is not due to any of the duplication tree constraints. Therefore it can also be applied to the original ADDTREE algorithm.

Using both refinements together

In the first refinement, we simply reduce the number of scores to be calculated. In the second one, we modify the way some of the scores are computed: if \( S(i, j) \) has been calculated at the previous step, it is updated in linear time, instead of being re-computed in quadratic time. Therefore, both refinements are complementary.

In Table 1, we show the average number of scores that are calculated during a run of both versions (basic and refined) of DTSCORE, on 50 simulated data sets (using the method described in the next section) and for several values of \( n \). For the refined version of DTSCORE, we show both the total number of scores that are computed and the number of scores calculated in quadratic time. These numbers clearly show that the combined refinements to the DTSCORE algorithm provide large improvements over the basic version. Moreover, assuming that 1-duplications are generally favoured over \( k \)-duplications, it is very likely that the performance increase of the refined DTSCORE algorithm will be even considerably larger with real data. On the speed performance side, the refined version of DTSCORE reconstructs a duplication tree from a 50-copies distance matrix in approximately 0.3 seconds on a 500 MHz Intel PIII machine. On the same data, the basic and the \( O(n^4) \) (only the second refinement) versions of DTSCORE reconstruct trees in 2.5 and 0.7 s, respectively. When combining both refinements, DTSCORE compares quite
favouredly with the FITCH $O(n^4)$ program (Felsenstein, 1989), from the PHYLIP package, which reconstructs a tree from the same distance matrix in 52 seconds, but as expected, it is far behind NJ (also from PHYLIP), which reconstruct a tree from the same distance matrix in only about 0.01 seconds.

**RESULTS**

**Simulation protocol**

We compare the topological accuracy of the 5 following distance methods : WINDOW (Tang et al., 2001), NJ (Saitou and Nei, 1987), and DTSCORE using the different window fitness functions, for $8 \leq n \leq 26$. We do not compare DTSCORE to the other reconstruction methods presented in (Benson and Dong, 1999; Tang et al., 2001; Jaitly et al., 2001), since they are restricted either to 1-duplications, or to short repeated sequences. Since there was no available WINDOW method implementation at the time we were writing this paper, we use our own implementation. For each $n$, we generate 1000 duplication trees and their associated data set, and count the number of times each method reconstructs the full topology of the trees. We study both the cases in which the generated data sets follow, and do not follow the molecular clock mode of evolution.

Simulated data sets are generated using the following procedure. First a duplication tree $T$ with no branch lengths is randomly uniformly generated using the method described in (Gascuel et al., 2002). We randomly value the branches using the same method as in (Kuhner and Felsenstein, 1994) and obtain a totally ordered duplication history that satisfies the molecular clock assumption (as in Figure 1(a)). The branch length expectation in this history is about 0.035 mutations per site. To obtain non-molecular clock trees, we independently multiply every branch by $1+0.8\times X$, where $X$ is drawn from an exponential distribution with parameter value 1.8. To obtain molecular clock trees, we simply multiply each branch by 1.8. The maximum pairwise divergence within the trees produced in this way is in the range $[0.1079; 0.2809]$, and seems in accordance with the real data we studied in (Elemento et al., 2001, 2002).

The SEQ-GEN (Rambaut and Grassly, 1997) program is then used to produce a 1000bp-long nucleotide multiple alignment from the generated tree, using a F84 model of substitution (Felsenstein and Churchill, 1996). A distance matrix is computed from this multiple alignment using the same model of substitution and, finally, each algorithm is ran against this distance matrix.

**Performance comparison**

The results are displayed in Table 2 for the simulations with molecular clock (MC), and in Table 3 for the simulations without molecular clock (NO-MC). They clearly indicate that the DTSCORE algorithm performs much better than both NJ and the WINDOW method, in both simulations. In both the MC and NO-MC simulations, the 3 window fitness functions perform similarly in terms of recovering the correct tree, for every $n$ considered, but $MINAVG$ provides slightly better results overall. Note that the other benefit of using $MINAVG$ is that it makes it possible to use the first refinement described above, whereas using $AVG$ does not allow it.

As expected, the WINDOW method performs better in the MC case than in the NO-MC one, but it is just slightly better than NJ in the MC case. The performances of NJ and WINDOW become very poor as the number of copies increases, especially in the NO-MC case (for $n = 26$, the WINDOW method is only able to recover 2 duplication trees out of 1000 in the NO-MC experiment). The poor results of the NJ algorithm are easily explained by the fact that it explores the space of phylogenies, without restriction to duplication trees. When $n$ increases, its chances of finding a duplication tree rapidly decrease, since the proportion of duplication trees among phylogenies becomes very small (Elemento et al., 2002). Due to its window fitness function, the WINDOW method is likely to suffer from the same weaknesses as UPGMA, which was shown to perform worse than NJ, even with sequences that follow the molecular clock mode of evolution (Saitou and Nei, 1987).

**Application to the TRGV genes**

We also apply DTSCORE to the 9 tandemly repeated genes of the TRGV locus (Lefranc et al., 1986), whose duplication tree has been reconstructed using DTEXPLORE (Elemento et al., 2001, 2002). Using the same data, DTSCORE finds the same duplication tree as DTEXPLORE. In (Elemento et al., 2001, 2002), we showed

**Table 1.** Average (out of 50 runs) number of scores calculated by the basic and refined versions of the DTSCORE algorithm

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>45</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic DTSCORE, # scores</td>
<td>103</td>
<td>337</td>
<td>793</td>
<td>1550</td>
<td>2637</td>
<td>4241</td>
<td>6296</td>
<td>8757</td>
<td>12467</td>
</tr>
<tr>
<td>refined DTSCORE, # scores</td>
<td>67</td>
<td>181</td>
<td>363</td>
<td>614</td>
<td>934</td>
<td>1363</td>
<td>1860</td>
<td>2379</td>
<td>3158</td>
</tr>
<tr>
<td>refined DTSCORE, # scores in $O(n^2)$</td>
<td>45</td>
<td>99</td>
<td>192</td>
<td>310</td>
<td>456</td>
<td>636</td>
<td>859</td>
<td>1088</td>
<td>1387</td>
</tr>
</tbody>
</table>

---

S97
that this reconstructed duplication tree has strong biological support, for two reasons: first it successfully predicts a polymorphism that occurs in the human population; second, we obtain the same tree when using traditional phylogenetic reconstruction programs on the same data, but without restriction to duplication trees. We show in the same paper that this is not likely due to chance, since only 3.5% of phylogenies are also duplication trees for 9 copies.

As in (Elemento et al., 2001, 2002), we use the bootstrap procedure (Felsenstein, 1985) to assess the reliability of this duplication tree. We generate 1000 pseudosamples, apply DTSCORE to each of these pseudosamples, and then compute the bootstrap proportion of every branch in the initial duplication tree. The bootstrap proportions are similar as those previously calculated. However DTSCORE ‘crunches’ the 1000 pseudosamples in less than 3 seconds on a 500 MHz PIII machine, while it takes DTEXPLORE more than 13 hours on a 5-nodes cluster (of nearly identical machines) to process the same amount of data.

CONCLUSION

The DTSCORE algorithm for reconstructing duplication trees we present in this paper performs much better at retrieving the correct duplication tree than any other method we studied. We introduce two refinements to reduce its time complexity, making it suitable for computationally intensive tasks, such as bootstrap analysis, or analysis of large data sets. A DTSCORE implementation in the C language is available on our web page. The DTDRAW program, available at the same address can be used to draw the (rooted) duplication trees obtained from the DTSCORE algorithm. Although both DTSCORE and DTEXPLORE reconstruct the same duplication tree on the TRGV data set, a direction for further research could be to compare the topological accuracy of both algorithms, as well as the effectiveness of both parsimony and distances approaches for duplication trees.

ACKNOWLEDGEMENTS

We thank Andy McKenzie and Marie-Paule Lefranc for their helpful comments on the preliminary versions of the manuscript.

REFERENCES


Reconstructing tandem duplication trees


