JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 15, Number 8, 2008

Pp. 1079–1092
DOI: 10.1089/cmb.2008.0116

Improving Reversal Median Computation Using Commuting Reversals and Cycle Information

WILLIAM ARNDT and JIJUN TANG

ABSTRACT

In the past decade, genome rearrangements have attracted increasing attention from both biologists and computer scientists as a new type of data for phylogenetic analysis. Methods for reconstructing phylogeny from genome rearrangements include distance-based methods, MCMC methods, and direct optimization methods. The latter, pioneered by Sankoff and extended with the software suites GRAPPA and MGR, is the most accurate approach, but is very limited due to the difficulty of its scoring procedure—it must solve multiple instances of the reversal median problem to compute the score of a given tree. The reversal median problem is known to be NP-hard and all existing solvers are extremely slow when the genomes are distant. In this paper, we present a new reversal median heuristic for unichromosomal genomes. The new method works by applying sets of reversals in a batch where all such reversals both commute and do not break the cycle of any other. Our testing using simulated datasets shows that this method is much faster than the leading solver for difficult datasets with only a slight accuracy penalty, yet retains better accuracy than other heuristics with comparable speed, and provides the additional option of searching for multiple medians. This method dramatically increases the speed of current direct optimization methods and enables us to extend the range of their applicability to organellar and small nuclear genomes with more than 50 reversals along each edge.

Key words: algorithms, combinatorial optimization, computational molecular biology, genomic rearrangements, phylogenetic analyses.

1. INTRODUCTION

DUE TO THE ADVENT OF HIGH-THROUGHPUT SEQUENCING and the consequent reduction in costs, we are seeing an explosion in the amount of genomic data of all types. In particular, the availability of fully sequenced and well annotated genomes allows us to move beyond the mere sequence level in the study of genomic evolution. Once a genome has been annotated to the point where gene homologs can be identified, each gene family can be assigned a unique integer where the sign indicates strand and a chromosome is able to be represented by a permutation of such integers. Rearrangements of genes under

Department of Computer Science and Engineering, University of South Carolina, Columbia, South Carolina.

reversals, transpositions and other operations then amount to rearrangements of these orderings. Such rearrangements are known to be an important evolutionary mechanism (Downie and Palmer, 1992) and their use in reconstructing phylogenies has been studied intensely since the pioneering papers of Sankoff and colleagues (Blanchette et al., 1997; Sankoff and Blanchette, 1998). Biologists have embraced this new source of data in their phylogenetic work (Belda et al., 2005; Bhutkar et al., 2007) and also in comparative genomics (Pevzner and Tesler, 2003), while computer scientists are slowly solving the difficult problems posed by the manipulations of these gene orders (Moret et al., 2005). During the past several years, computer scientists have been able to make substantial progress in genome rearrangement research: with the solution for reversal distance (Hannenhalli and Pevzner, 1995) and reversal median (Caprara, 2001), researchers were able to estimate phylogenies and ancestral genomes based on reversals (the dominant events in organellar genomes).

There are several widely used methods for phylogenetic analysis using genome rearrangement data, including distance-based methods such as neighbor-joining (Saitou and Nei, 1987), `GRAPPA` (Moret et al., 2001), and `MGR` (Bourque and Pevzner, 2002). The latter two will generally achieve better accuracy than distance-based methods such as neighbor-joining. Their basic optimization tool is an algorithm for computing the reversal (or breakpoint) median of three genomes. However, using `GRAPPA` and `MGR` to compute phylogeny for organismal genomes with many events is extremely expensive, because the median computation takes time exponential in both the size of the genomes and the distances among genomes.

In this paper, we present a fast yet accurate heuristic using commuting reversals to improve the reversal median computation for both distant and large genomes. Integrated with `GRAPPA`, this method extends the capability of the existing direct optimization method so that accurate reconstruction of phylogenies can be achieved for larger genomes. We will also provide some discussions regarding reversal medians when the number of events approaches saturation.

2. BACKGROUND

2.1. Genome Rearrangements

We assume a reference set of n genes $\{1, 2, \ldots, n\}$; thus, a unichromosomal genome can be represented as a signed permutation of these genes, and each gene is given an orientation that is either positive, written i, or negative, written $-i$. Genomes can evolve through events including reversals, transpositions and transversions.

Let π be a genome with the signed ordering of $1, 2, \ldots, n$. A *reversal* between indices i and j $(i \leq j)$, transforms π to a new genome with linear ordering

$$1, 2, \ldots, i-1, -j, -(j-1), \ldots, -i, j+1, \ldots, n.$$

A *transposition* on genome π acts on three indices i, j, k, with $i \leq j$ and $k \notin [i, j]$, picking up the interval $i, i+1, \ldots, j$ and inserting it immediately after k. Thus, genome π is replaced by (assume $k > j$):

$$1, \ldots, i-1, j+1, \ldots, k, i, i+1, \ldots, j, k+1, \ldots, n.$$

A *transversion* is a transposition followed by a reversal of the transposed subsequence; it is also called an *inverted transposition*.

2.2. Distance computation

Given two genomes π_1 and π_2, we define the *edit distance* $d(\pi_1, \pi_2)$ as the minimum number of events required to transform one of these genomes into the other. When only reversals are allowed, the edit distance is the *reversal distance*. Hannenhalli and Pevzner (1995) developed a mathematical and computational framework for signed gene-orders and provided a polynomial-time algorithm to compute the edit distance between two signed gene-orders under reversals; Bader et al. (2001) later showed that this edit distance can be computed in linear time. However, computing the reversal distance is NP-hard

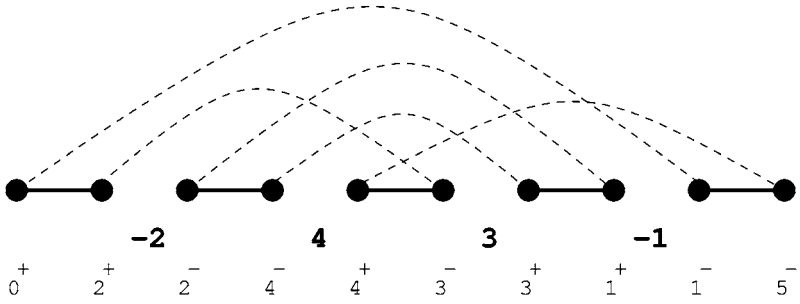


FIG. 1. Breakpoint graph between genome (−2 4 3 −1) and the identity genome (1 2 3 4).

in the unsigned case (Caprara, 1999b). Two genes i and j are said to be *adjacent* in genome G if i is immediately followed by j, or, equivalently, $-j$ is immediately followed by $-i$. A *breakpoint* in π_1 is defined as an ordered pair of genes (i, j) such that i and j are adjacent and have the same relative orientation in π_1 but not in π_2. The *breakpoint distance* (Sankoff and Blanchette, 1998), which is simply the number of breakpoints in π_1 relative to π_2, is an approximation of evolutionary distance measurement because breakpoints do not directly correspond to evolutionary events.

The Hannenhalli and Pevzner (HP) algorithm is based on the breakpoint graph (Fig. 1). Given two permutations π_1 and π_2 with n genes, we can assume without loss of generality that π_2 is the identity. Begin by padding π_1 with gene 0 on the left end and gene $n + 1$ on the right end. For each gene i in π_1, two vertices are created, i^- and i^+. These vertices are connected with two sets of undirected edges, one for each genome. One set of edges, called *desire edges*, connecting i^+ and $(i + 1)^-$ for all $0 \leq i \leq n$, represents the identity genome and is shown with dashed arcs in Figure 1. For each adjacency (i, j) in π_1 add a *reality edge*. If gene i is positive this edge begins at vertex i^+; if gene i is negative this edge begins at vertex i^-. Similarly, if gene j is positive this edge ends at vertex j^-; if gene j is negative this edge ends at vertex j^+.

The arrangment of these edges form cycles which alternate between reality and desire edges; the crucial concept is the relationship between the number of cycles, denoted by $c(\pi_1, \pi_2)$, and the number of reversals needed to transform π_1 into π_2. Overlapping cycles in certain configurations create structures known as *hurdles*; we use $h(\pi_1, \pi_2)$ to represent the number of hurdles. A very unlikely configuration of hurdles can form a *fortress* (Hannenhalli and Pevzner, 1995). Hannenhalli and Pevzner (1995) proved that the reversal distance between two signed permutations of n genes is given by:

$$n - c(\pi_1, \pi_2) + h(\pi_1, \pi_2) + (1 \text{ if fortress present}, 0 \text{ otherwise}).$$

2.3. Sorting and commuting reversals

The HP algorithm also returns *one* (and only one) minimum sorting sequence that transform one permutation into another. Siepel (2003) extended the HP theorem to find *all sorting reversals*, i.e., all possible reversals that appear as the first step in the sorting. Figure 2 gives one example of sorting

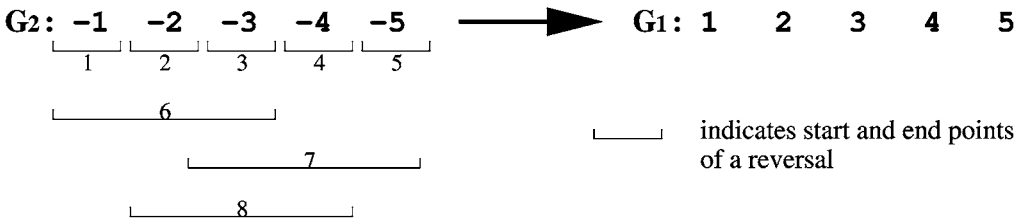


FIG. 2. Eight sorting reversals that bring (−1 −2 −3 −4 −5) one step closer to (1 2 3 4 5).

reversals: there are eight possible reversals that bring π_1 one step closer to π_2 (the identity genome). This algorithm can be easily extended to enumerate all minimum sorting sequences by identifying every sorting reversal at each step of the sorting. This enumeration can be very time consuming and Braga et al. (2007) later provided an algorithm so that a representative set of these sorting sequences can be found.

The concept of commuting reversals was introduced in Bergeron et al. (2002). To define commuting reversals create three sets of permutation elements: those which are only members of the first reversal, those which are only members of the second, and those which are members of both. The two sorting reversals *commute* if and only if one of these three sets is empty. Commuting reversals have the desirable property that applying them to a permutation will always give the same result no matter the order in which they are applied.

2.4. Reversal median problem

Given three genomes (permutations) π_1, π_2, π_3 and another genome π_0, we define the *median score* of π_0 as $d(\pi_0, \pi_1) + d(\pi_0, \pi_2) + d(\pi_0, \pi_3)$. The median problem for these three genomes is to find a genome π_0 that minimizes the median score. We also define the *perfect median score* as $\left\lceil \frac{d(\pi_1,\pi_2)+d(\pi_1,\pi_3)+d(\pi_2,\pi_3)}{2} \right\rceil$, which is a lower bound of the score for a median problem.

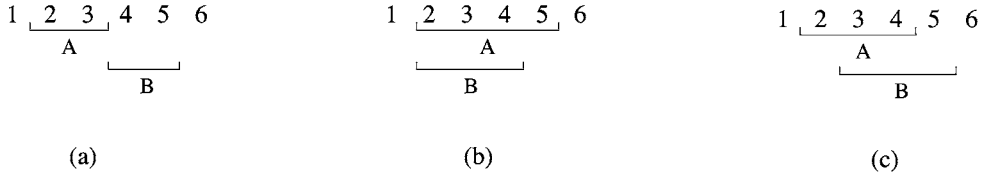
The median problem is NP-hard (Caprara, 1999a; Pe'er and Shamir, 1998) even for simple distance definitions such as breakpoint distance. Seeking a median that minimizes the breakpoint distance can be transformed into a special instance of the well-studied Traveling Salesperson Problem (Blanchette and Sankoff, 1997) and can be solved relatively fast. But in practice, the breakpoint median is not effective—it is easy to obtain trivial solutions (where the median gene-order coincides with one of the input genomes), and thus using breakpoint median is not as accurate as using the reversal median for genome rearrangement analysis (Moret et al., 2002).

The *reversal median* problem is to find a median genome that minimizes the sum of reversal distances from it to the three input genomes. Several reversal median solvers have been proposed. Caprara's solver (Caprara, 2001) is based on an extension of the breakpoint graph, while that developed by Siepel and Moret (2001) runs a direct search, which is later improved to use sorting reversals (Siepel, 2001). Both Caprara's and Siepel's median solvers are exact and are included in `GRAPPA`. In practice, Caprara's median solver is faster than Siepel's solver when the genomes are not close (Moret et al., 2002). The solver provided by `MGR` (Bourque and Pevzner, 2002) uses a heuristic approach: it seeks good reversals that bring a genome closer to the ancestral genome. For three genomes, the `MGR` algorithm evaluates all possible reversals for each of the three genomes π_1, π_2 and π_3, identifying good reversals that bring a genome closer to the ancestral genome. Since the ancestral genome is unknown, the algorithm chooses reversals which make π_1 closer to both π_2 and π_3 as *good reversals*. Thus, the algorithm will iteratively carry on good reversals in the three genomes until all three are transformed into an identical genome, which is viewed as the most likely ancestral median.

All these median solvers become extremely slow for large and distant genomes. A common speedup process used by all methods makes use of the concept of conserved adjacency. A gene pair (i, j) is *conserved adjacent* if (i, j) or its inverse $(-j, -i)$ is present in all genomes as consecutive elements (Hannenhalli and Pevzner, 1996). A block of k conserved adjacent genes can be replaced by a new gene and the total number of genes reduced by $k-1$ (Bourque and Pevzner, 2002). This condensation procedure improves memory performance and is most effective when the genomes are close: a median of genomes with 1000 genes and 50 reversals per edge can be condensed to $\sim$200 genes only.

3. REVERSAL MEDIAN COMPUTATION USING COMMUTING REVERSALS

We set out to find an improved reversal median heuristic which has a better tradeoff between speed and accuracy than existing methods. The new algorithm is different from `MGR` in that it will conduct a direct search from one of the known genomes, using sorting reversals to limit the search space. Our algorithm also improves over Siepel's (2001) median solver by using commuting reversals in the set of sorting reversals from the start genome to the other two genomes. Our new median solver will also report multiple solutions, a property lacking in almost all existing methods.



set only affected by A: [2, 3]	set only affected by A: [5]	set only affected by A: [2]
set only affected by B: [4, 5]	set only affected by B: ϕ	set only affected by B: [5]
set only affected by A and B: ϕ	set only affected by A and B: [2, 3, 4]	set only affected by A and B: [3, 4]

FIG. 3. Examples of commuting reversals **(a,b)** and non-commuting reversal **(c)**.

3.1. A naive approach

Let us first present a naive approach. Suppose the three input permutations are π_1, π_2, and π_3. Define a recursive function which has input π_1, π_2, π_3, and π_4, where π_4 is initially set equal to π_1. This function first computes two sets of sorting reversals: set α which contains sorting reversals from π_4 to π_2, and set β which contains sorting reversals from π_4 to π_3. Let set γ be the intersection of α and β. If γ is empty, then determine the reversal median score of π_4 relative to π_1, π_2, and π_3. If this median score is less than or equal to the lowest score yet seen then report, as a pair, π_4 and the median score. If γ is not empty, then repeat the following process until it is: remove one reversal from γ and apply it to π_4 to obtain π_4', and call the function recursively with arguments π_1, π_2, π_3, and π_4'.

Several concerns make this method undesirable. First, the amount of computation required increases exponentially with the number of reversals separating the three permutations, as the method is searching all orderings of all reversals which sort towards both π_2 and π_3. Second, it can be shown, by exhaustively searching permutations against a small reversal median problem, that a median permutation does not necessarily lie on a sorting path between two of the three initial permutations. The presented naive approach thus cannot guarantee an optimal solution because some and possibly all paths to medians would require that one or more reversals which are never members of γ be chosen. We will not attempt to improve this aspect of the naive method, since doing so would require a large number of additional reversals to be considered in set γ with relatively little return compared to the massive amount of additional computation being performed.

Another problem of the naive approach, one that can be addressed, is that it performs a large amount of redundant computation by visiting the same permutation multiple times. This can be reduced by using information about commuting reversals. Imagine a set of sorting reversals which sort π_1 towards both π_2 and π_3. Select any pair of these reversals A and B which occur along the path to a median. If reversals A and B do not commute, then changing the order in which A and B are applied affects the resulting permutation (Fig. 3c); If A and B commute (Fig. 3a,b) then the naive method will search the same permutation at least twice because both choices of ordering the application of A and B result in the same permutation.

3.2. An improved algorithm

The above analysis leads to a method to speed up the search by removing a large portion of the redundancy. Obtain from γ a set of reversals with the additional property that all pairs of reversals commute. This allows the order of applying these reversals to be ignored; every permutation that can be reached by applying any number of these commuting reversals can be enumerated and have its median score checked one time instead of enumerating permutations by the paths which lead to them. If n is the number of commuting reversals, then 2^n permutations can be reached, but the total number of paths to these permutations is $O(n^n)$.

Which subset of the 2^n permutations should be chosen? We have experimented with three methods:

- Method 1 is a brute force method which scores each of the 2^n permutations and chooses the one with the best median score as π_4', with good results but an obvious time complexity drawback.

- Method 2 draws samples from the 2^n permutations and chooses the best median score found among them as π'_4. This approach reduces both the time required and the accuracy; in general the quality of the results is proportional to the fraction of the space being searched.
- Method 3, the simplest method of all and surprisingly effective, is to apply all reversals in the set, that is, obtaining π'_4 by applying all commuting reversals to π_4. This approach works well until the number of events among the genomes is 30%–40% of the number of genes. Beyond that point each search step normally increases the median score and tends to converge with worse results than a trivial solution.

The fact that applying all commuting reversals can result in a worse median score suggests there is a more complex interaction between the application of a single sorting reversal to a permutation and its influence on other sorting reversals. This interference between sorting reversals comes from the breaking of cycles in the breakpoint graphs of the problem instance. Imagine a breakpoint graph with one cycle containing two sorting reversals that commute. Applying either of those sorting reversals will alter the breakpoint graph to create two cycles. Afterwards, two possibilities exist: either both of the reality edges of the second reversal will remain in the same cycle, in which case this reversal will be a sorting reversal, or the reality edges of the second reversal will be separated into different cycles, in which case it will no longer be a sorting reversal. This line of thought leads to an explanation of breakpoint graph cycle interactions which shares similarities with finding and using commuting reversals.

3.3. Parallel and perpendicular sorting reversals

We call a pair of sorting reversals *parallel* on a breakpoint graph if they both commute and only break reality edges in the same cycle of the breakpoint graph and if applying both reversals to the permutation creates two additional cycles. On the other hand, a pair of sorting reversals is *perpendicular* on a single breakpoint graph if they commute and break reality edges in the same cycle of the breakpoint graph and if applying both reversals to the permutation creates one additional cycle.

When multiple breakpoint graphs are considered, we can also call a pair of reversals parallel if in all graphs the reality permutation is the same (the need for the desired permutation to be the identity is relaxed), both reversals are sorting reversals on all graphs, and the reversals are not perpendicular on at least one of the considered graphs. A pair of reversals is perpendicular over multiple breakpoint graphs if in all considered graphs the reality permutation is the same, the reversals sort each graph, and the reversals are perpendicular on one or more graphs. If any of the reversals acts on two different cycles then the pair is also perpendicular.

Parallel reversals are very useful because the median score almost always decreases by k when applying k parallel reversals, due to the role played by the cycles of the breakpoint graph in the reversal distance formula and the low probability of hurdles.

Figure 4 describes a simple graph method to visualize the parallel and perpendicular reversal properties. We first obtain the set of sorting reversals between two permutations, but additionally save the cycle membership and order in which each of the reality edges appears when traversing a cycle. For each cycle, do the following: create a vertex called a *break location node* for each reality edge in the cycle and label the node with the vertex labels that appear on each side of the edge in the cycle. Place the break location nodes in the order that they appear when traversing the cycle. After all break location nodes have been arranged draw an edge called a *cut chord* connecting both of the corresponding break location nodes in the ring for each sorting reversal which acts on two reality edges in the same cycle (not reversals which merge hurdles or cut a hurdle or fortress). These chords correspond to the cut that divides the cycle into two smaller cycles when a reversal is applied, and shows which break location nodes will remain in the same cycle and which will be separated. For every pair of reversals in the same cycle, if the cut chord for each intersects, then this pair of reversals is perpendicular, otherwise, the reversals are parallel.

A special case exists where two reversals share a break location node, since sometimes such reversals will be parallel and sometimes perpendicular depending on the permutation layout. For example, in Figure 4, reversals a and f share a reality edge and are parallel, yet reversals f and b share a reality edge but are perpendicular. We have been unable to determine a simple, general case rule to differentiate these two situations. As a result, our definitions and implementation sacrifice a small bit of performance by marking all reversals which share a reality edge as perpendicular.

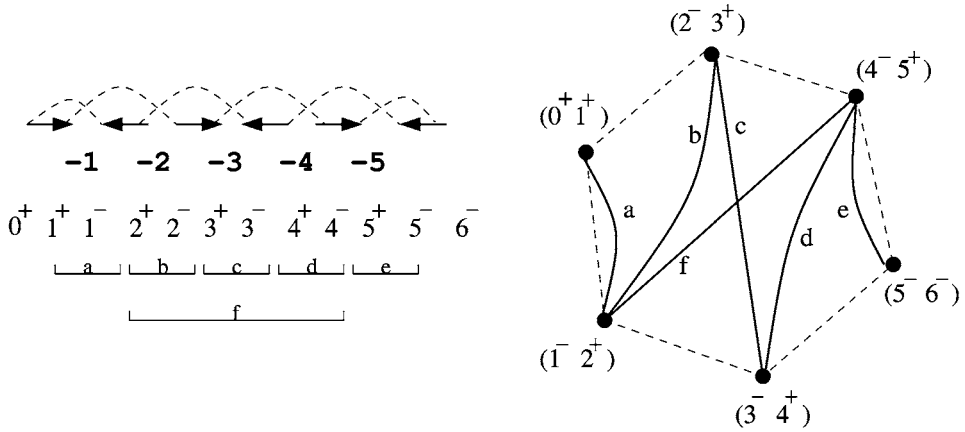


FIG. 4. Graph representation of cycle interferences on a set of commuting reversals, genes 0 and 6 represent linear chromosome endpoints.

A necessary consideration is the method used to find sets of parallel reversals. Taken separately, sets of commuting reversals and sets of parallel reversals fit conveniently into circle graph descriptions, so an independent set algorithm can find maximal sets in low polynomial time. However, our problem requires the simultaneous satisfaction of two sets of constraints for each of two breakpoint graphs. Casual investigation has shown that the combined constraints often do not produce a valid circle graph; consequently we instead implemented two simple heuristics which produce fair results in all situations. These heuristics are described in Section 3.4.

3.4. The implemented algorithm

The overall algorithm of our new unichromosomal median solver is presented in Algorithm 1.

Function *ReversalMedianSolver* begins the search of the median space by calling *RecursiveSearch*. *RecursiveSearch* generates the two relevant lists of sorting reversals and calls *UseGamma* to generate lists of reversals with the desirable commuting and parallel properties. *UseGamma* is a combination of the following two heuristics which attempt to maximize the size of a set of reversals that are both parallel and commute.

The first heuristic tries to find a maximal set of commuting reversals. The input is a set of reversals which sort π_1 towards both π_2 and π_3, and each reversal is assigned an initial weight of 0. Each pair of the input reversals are then checked and if two reversals do not commute, the weight of each is increased by 1. Until all reversals have a weight of 0, the reversal with the highest weight is repeatedly removed from the set and all reversals that does not commute with this reversal will have their weight decreased by 1. The set of reversals which have not been removed are then used as input to the second heuristic, which follows the exact same pattern, except it is modified to produce a set with no pairs of perpendicular reversals. It additionally uses the breakpoint graph cycle traversal orderings of reality edges; this allows it to add weights based upon if pairs of reversals are perpendicular.

Several details are worth mentioning as well. First, the choice of the start permutation has some impact on the results and our experiments show that, of the three initial permutations, using the one with the lowest sum of its pairwise reversal distances with the other two will produce the best median scores. Second, despite our efforts to prevent it, a very large amount of redundant computation, in the form of scoring the same permutation multiple times, still occurs. We used a permutation hash table to check for redundant search paths. This is not a critical aspect and can be removed without significant impact—in fact, due to memory constraints it must be removed for genomes larger than approximately 400 genes. Third, we use the set Δ to contain some reversals at the current level of depth first search which have not yet been applied. Our implementation preserves in Δ the reversals which have been removed due to commuting constraints, but does not preserve reversals which have been removed due to perpendicular interference. This decision was made primarily as a tradeoff of some loss in search thoroughness for increased speed; preserving either or both groups of reversals in Δ would also be possible.

Algorithm 1. Algorithm overview for the new reversal median solver.

function ReversalMedianSolver (input three permutations π_1, π_2, π_3)
Compute the pairwise reversal distances between π_1, π_2, π_3
Choose the one with the smallest sum of its two distances as π_4
if π_1 was not assigned to π_4 **then**
  swap it with the permutation assigned as π_4
end if
$BestSoFar \Leftarrow \infty$
Call **function** *RecursiveSearch* with π_1, π_2, π_3, π_4
return list of saved π_4' with score equal to $BestSoFar$
end function ReversalMedianSolver

function RecursiveSearch (input four permutations π_1, π_2, π_3, π_4)
$\alpha \Leftarrow$ sorting reversals from π_4, π_2
$\beta \Leftarrow$ sorting reversals from π_4, π_3
$\gamma \Leftarrow$ intersection of α and β
$\Delta \Leftarrow$ *UseGamma*(γ, π_1, π_2, π_3, π_4)
while set Δ contains elements **do**
  Set γ to Δ and clear Δ
  $\Delta \Leftarrow$ *UseGamma*(γ, π_1, π_2, π_3, π_4)
end while
end function RecursiveSearch

function UseGamma (input set γ and permutations π_1, π_2, π_3, π_4, output set Δ)
Initialize the weight of each reversal to 0
for each pair of reversals in γ **do**
  if reversals do not commute **then** add weight 1 to each
end for
repeat
  Find the reversal A with largest weight
  Remove A from γ and place it in set Δ
  Reduce weight of each reversal not commuting with A by 1
until the weight of all reversals in γ is 0
for each pair of reversals in γ **do**
  if pair is perpendicular **then** add 1 weight to each
end for
repeat
  Find the reversal A with largest weight and remove A from γ
  Reduce weight of each reversal perpendicular to A by 1
until the weight of all reversals in γ is 0
Apply the reversals in set γ to π_4 to create π_4'
Calculate the median score of π_4' with respect to π_1, π_2, π_3
if median score of $\pi_4' \leq BestSoFar$ **then**
  Assign the score to *BestSoFar* and save π_4'
end if
if median score of $\pi_4' <$ median score of π_4 **then**
  Call **function** *RecursiveSearch* with π_1, π_2, π_3, π_4'
end if
return Δ
end function UseGamma

4. EXPERIMENTAL RESULTS ON THREE GENOMES

4.1. Setup of simulations

We examine the performance (in terms of speed and accuracy) of the new method using simulated datasets. Because all existing median solvers have very good performance when genomes are close, we only test distant genomes and compare our method against Caprara's solver (slower but exact), and MGR (faster but less accurate).

We focused our experiments on organelle genomes and generated datasets of three genomes with 100 genes for each genome (larger genome sizes were also tested). We first generated trees with three leaves and one internal node, assigned the identity permutation on the internal node, and generated the three leaves by applying rearrangement events along each edge respectively. The number of events on each edge is governed by two parameters: the number of overall evolutionary events and the tree shape. We used various number of evolutionary rates: letting r denote the total number of events along all three edges, we used values of r in the range of 80 to 140. We found from our experience that the tree shape plays an important role in median computation, so we used three tree shapes for each r: a tree with almost equal length edges, i.e., the ratio of three edges are (1:1:1); a tree with one edge a bit longer than the other two, i.e., of ratio (2:1:1); a tree with one edge much longer than the other two, i.e., of ratio (3:1:1). While all computations were based on reversal distances and reversal medians, we generated the data with a deliberate model mismatch to test the robustness of the method, using a mix of 80% reversals and 20% transpositions. For each combination of parameter settings, we ran 10 datasets and averaged the results. Experiments were conducted on a Linux cluster with 152 Intel Xeon CPUs, but each CPU works independently on a test task. MGR command line options *-c -H1* were used.

4.2. Accuracy

Caprara's median solver had no problem finishing all the datasets with evolutionary rate $r = 80$ and $r = 100$; however, it finished only a very small number of datasets for $r = 120$ and $r = 140$: only four out of 60 datasets finished within 48 hours of computation. Here we report the results separately using slightly different criteria for $r \leq 100$ and $r \geq 120$.

For $r \leq 100$, we report the average median score from our method, Caprara's solver, and MGR. We also report the average perfect (lower bound) median score. We choose to show the actual median score instead of the error rates compared to the optimal score returned by Caprara's solver because this optimal solution could not be produced for all tests. Table 1 shows the results, which indicate our method is very accurate, with <1.5% difference from Caprara's results. Our method is most accurate when all three edges have nearly equal length. Approximately 70% of datasets report an equal median score to that found by Caprara's, while the other 30% differ by at most 1.

For $r \geq 120$, since Caprara's solver does not finish all datasets, we only report the average median score from our method and compare it to MGR and the average lower bound of the median score. Table 2 shows the result, which indicates that our method can find better medians than MGR.

An additional measure of the usefulness of results for phylogenetic reconstruction is the reversal distance to the simulated ancestor genome. Here we report the average distance to the ancestor for the three methods. Test cases Caprara's method could complete appear in Table 3, while the remaining test cases appear in Table 4.

4.3. Speed

We recorded the running time for each test case as well. Since our method will report all results it can find, there are two measures: (1) the time within which it finds the first result, and (2) the average number of results it finds within the limit of one hour.

Tables 5 and 6 show the first time comparison. When the datasets are relatively easy ($r = 80$), Caprara's solver is much faster than our method. However, it slows down very quickly as the difficulty increases, and almost no dataset can be finished for $r \geq 120$. Meanwhile, the running time of our method is quite consistent: fewer than 30 minutes were used even for the most difficult datasets, which is comparable to the speed of MGR.

TABLE 1. COMPARISON OF MEDIAN SCORES FOR $r \leq 100$

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 80	*r = 100*	*r = 80*	*r = 100*	*r = 80*	*r = 100*
Score lower bound	86.2	104.2	89.4	105.8	85.7	101.3
New method's median score	88.2	109.5	91.8	111.4	89.1	106.7
Caprara's median score	87.9	107.6	91.4	109.8	88.0	105.2
MGR median score	90.3	113.7	94.3	116.8	89.8	110

TABLE 2. COMPARISON OF MEDIAN SCORES FOR $r \geq 120$

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 120	*r = 140*	*r = 120*	*r = 140*	*r = 120*	*r = 140*
Score lower bound	116.1	123.5	116.1	122.7	110.3	117.6
New method's median score	125.8	135.3	124.5	134.7	117.9	127.0
MGR median score	132.9	143.6	131.4	142.8	123.6	135.1

TABLE 3. COMPARISON OF REVERSAL DISTANCE TO SIMULATED ANCESTOR FOR $r \leq 100$

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 80	*r = 100*	*r = 80*	*r = 100*	*r = 80*	*r = 100*
New method	9.3	21.7	9.6	20.4	7.2	18.2
Caprara's method	4.5	18.2	7.2	16.8	5.2	15.7
MGR	9.3	23.5	11	25.4	9.1	18.6

TABLE 4. COMPARISON OF REVERSAL DISTANCE TO SIMULATED ANCESTOR FOR $r \geq 120$

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 120	*r = 140*	*r = 120*	*r = 140*	*r = 120*	*r = 140*
New method	39.8	49.3	35.2	45.4	23.1	32.1
MGR	40.7	51.6	37.5	49.5	29.7	37.7

TABLE 5. COMPARISON OF RUNNING TIME FOR $r \leq 100$ (IN SECONDS)

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 80	*r = 100*	*r = 80*	*r = 100*	*r = 80*	*r = 100*
New method's time	324	551	123	409	1.6	9.3
Caprara's time	3.6	12,876	57.2	31,387	4.3	6908
MGR time	11.2	51.9	11.6	78.2	10.3	35

TABLE 6. COMPARISON OF RUNNING TIME FOR $r \geq 120$ (IN SECONDS)

	(1:1:1)		*(2:1:1)*		*(3:1:1)*	
	r = 120	*r = 140*	*r = 120*	*r = 140*	*r = 120*	*r = 140*
New method's time	1485	1187	673	453	30	226
Caprara's time	>172,880	>172,880	>172,880	>172,880	>172,880	>172,880
MGR time	271.6	560.1	237.8	626.9	135.3	385.4

TABLE 7. COMPARISON OF MEDIAN SCORES FOR GENOMES WITH 1000 GENES

	r = 100	*r = 110*	*r = 120*	*r = 130*	*r = 140*	*r = 150*
New method	480.1	528.4	576.6	626.0	676.4	729.4
Caprara's	479.7	527.1	575.4	623.0	670.1	—

— means a method cannot finish most of the datasets in that test case.

TABLE 8. COMPARISON OF DISTANCE TO SIMULATED ANCESTOR FOR GENOMES WITH 1000 GENES

	r = 100	*r = 110*	*r = 120*	*r = 130*	*r = 140*	*r = 150*
New method	1.3	4.7	2.8	8.4	13.9	32.3
Caprara's	2.1	2.3	1.9	3.8	4.1	—

— means a method cannot finish most of the datasets in that test case.

TABLE 9. COMPARISON OF RUNNING TIME FOR GENOMES WITH 1000 GENES (IN SECONDS)

	r = 100	*r = 110*	*r = 120*	*r = 130*	*r = 140*	*r = 150*
New method	42	35	29	36	51	53
Caprara's	54	170	533	14,132	31,716	>4 days

In general, our method found 12 medians with the same score within one hour. However, the number is not consistent: some datasets have only one result, while others have as many as 120 results. Additionally, by checking reversals on the found medians that do not change the median score, on average for each found median two more can be quickly located, though they are not significantly different from those already found.

4.4. Medians of larger genomes

We tested the performance of our method on larger genomes as well. The simulations were created with the same parameters, except the number of genes was increased to 1000. Each tested tree has three edges with the same length, ranging from 100 to 150 events. Since the number of genes after condensation for $r \geq 100$ is more than 600, `MGR` cannot finish any dataset in this test due to the extremely large search space, thus we only present the comparison between our method and Caprara's median.

Tables 7–9 show the results. In this study, Caprara's median solver could not finish 20% of the datasets when $r = 130$ and 140, and the results shown are averaged on the 80% of the datasets it could finish. There is clearly a threshold for Caprara's median solver: it could not finish any dataset when $r \geq 150$. This experiment suggests that our method has accuracy that is comparable to Caprara's solver, and unlike the existing methods, the speed of our method is much more stable and will not increase dramatically as the genomes become distant.

5. EXPERIMENTAL RESULTS ON PHYLOGENETIC RECONSTRUCTION

Our median solver has been integrated with `GRAPPA` so that its accuracy on phylogenetic reconstruction can be assessed. Because our median solver will continue to search all medians until the reversal set is empty, it may take too much time in the context of tree reconstruction since many instances of the median problem have to be solved. To address this problem, we limit the time that each median instance can use and report the best permutation before it is forced to stop. `GRAPPA` is designed to only use one solution to a median problem, so in the event that multiple permutations with the same median score are found by our method, only the first is used by `GRAPPA`. Again we performed our experiments using simulated datasets since this allows us to know the true trees in our simulation. We measure the accuracy of a phylogeny

TABLE 10. RF RATES (%) FOR DATASETS WITH 100 GENES

	$r = 4$	$r = 8$	$r = 12$	$r = 16$	$r = 20$	$r = 24$	$r = 28$	$r = 32$
New method	5	0	0.1	0.1	5	3.8	11.3	10
Caprara's	2.5	0	0.1	0.1	—	—	—	—
MGR	0	0	0.1	0.1	6.3	6.3	16.3	17.5

— means a method cannot finish most of the datasets in that test case.

TABLE 11. RF RATES (%) FOR DATASETS WITH 500 GENES

	$r = 20$	$r = 40$	$r = 60$	$r = 80$	$r = 100$	$r = 120$
New method	0	0	0	0	5	6.3
Caprara's	0	—	—	—	—	—
MGR	0	—	—	—	—	—

— means a method cannot finish most of the datasets in that test case.

method by using the Robinson Foulds (RF) error rate (the percent of edges in error with respect to the true tree) (Robinson and Foulds, 1981).

In this simulation study, we generated model tree topologies from the uniform distribution on binary trees, each with 10 leaves. On each tree, we evolved signed permutations of 100 and 500 genes, using an evolutionary rate, r (the expected numbers of events along a tree edge) of 4–32 for datasets with 100 genes and 20–120 for datasets with 500 genes. For each combination of parameters, we generated 10 trees; the final results are averaged on the 10 datasets.

We compared the results from MGR, GRAPPA with Caprara's median solver, and GRAPPA with our median solver. Tables 10 and 11 show the RF rates of datasets with 100 and 500 genes respectively. For 100 genes, Caprara's solver could not finish any dataset with $r \geq 20$ within two days of computation; hence, its result is not shown for those cases. MGR performs the best when $r = 4$, but our method outperforms it when the genomes become distant ($r \geq 20$). For 500 genes, neither MGR nor Caprara's solver could finish datasets with $r \geq 40$, while our method still performs very well. Compared to the experiments conducted on three genomes (Section 4.2), we notice that Caprara's solver fails with relatively smaller r because the scoring procedure of GRAPPA generates some medians that are very difficult to compute. It is also worth mentioning that, for very low values of r, both our method and Caprara's perform poorly, because in these cases the chance to obtain several trees with the same best score is higher and we had to choose a consensus tree as the best.

6. DISCUSSION

We believe there is a big problem in the general approach of using the reversal median problem to solve phylogenetic trees composed of distant genomes, a topic discussed in detail in Eriksen (2007). The direct optimization methods (GRAPPA and MGR) are based upon minimizing the number of reversal events, which requires either the false assumption that there is only one optimal median solution for all problem instances, or the slightly weaker assumption that although multiple optimal solutions can exist, they are all equally valuable for the construction of trees. Several of our test simulations demonstrate the existence of multiple medians that form trees with equal scores but have edge lengths differing by 30% or more. This shows that instances of a median problem do not contain the amount of exact information which current tree methods presume they do. We do not believe that the cause is hopeless, however. Instead, the notion that any median with optimal score is an equal representative of an internal node should be replaced; new methods or tree building algorithms should be devised which use multiple medians as an intermediate step moving closer to the true ancestor. To obtain more accurate results for large genomes, we may need to find as many medians as possible and choose the one with the minimal total distance to all the others as

the representative, or we may need to consider permutations with slightly less than optimal score if they appear in sufficiently large clusters.

The structure of median problems has further vulnerabilities. We ran a small experiment where a random median problem of 10 genes was created, and the median score of every permutation ($2^{10}10!$) in an exhaustive search was found. There are several findings from this experiment: (1) as confirmed by other researchers (Bernt et al., 2007), there exist multiple medians—we found 81 medians for this experiment, with median score of 15 reversals; and (2) some of the medians were as far as nine reversals from one another. This experiment clearly suggests that the impact of picking distinguishing medians with the same score in the current phylogenetic reconstruction methods should be investigated further.

7. CONCLUSIONS

In this paper we present a new reversal median solver using commuting reversals, and introduced the concept of parallel and perpendicular sorting reversals. This is a first step towards finding reversal medians faster by using searches which are able to step multiple events at a time. Further theoretical work can extend the definitions to include situations we have not been able to address such as events which share a reality edge. Also, the definitions could be extended to consider component and hurdle concerns. There is further potential for improvement in the method of finding sets of parallel reversals. This method would likely take advantage of the similarity with the *Maximal Independent Set* problem; a polynomial time algorithm to produce the maximal parallel reversal set may exist.

We extensively tested the method and compared its performance with the leading median solvers using simulated datasets. The experimental results show that our method is very accurate, and is much faster than the leading solver when the datasets are difficult. Our new method is a better choice for datasets with more than 500 genes, and will be useful when analyzing emerging small animal nuclear genomes. In the future, we will further improve the accuracy of our method by investigating better search strategies, and extend this work to deal with multi-chromosomal genomes.

ACKNOWLEDGMENTS

We were supported by the U.S. National Institutes of Health (NIH grant number R01 GM078991-01) and by the University of South Carolina. W.A was also supported by the Rothburg Fellowship at the University of South Carolina.

DISCLOSURE STATEMENT

No conflicting financial interests exist.

REFERENCES

Bader, D., Moret, B., and Yan, M. 2001. A fast linear-time algorithm for reversal distance with an experimental comparison. *J. Comput. Biol.* 8, 483–491.

Belda, E., Moya, A., and Silva, F. 2005. Genome rearrangement distances and gene order phylogeny in γ-proteobacteria. *Mol. Biol. Evol.* 22, 1456–1467.

Bergeron, A., Chauve, C., Hartman, T., et al. 2002. On the properties of sequences of reversals that sort a signed permutation. *J. Biol. Inform. Math.*, 99–108.

Bernt M., Merkle, D., and Middendorf, M. 2007. Using median sets for inferring phylogenetic trees. *Bioinformatics* 23, 129–135.

Bhutkar, A., Gelbart, W., and Smith, T. 2007. Inferring genome-scale rearrangement phylogeny and ancestral gene order: a Drosophila case study. *Genome Biol.* 8, R236.

Blanchette, M., Bourque, G., and Sankoff, D. 1997. Breakpoint phylogenies, 25–34. In Miyano, S., and Takagi, T., eds., *Genome Informatics*. Universal Academy Press, Tokyo.

Blanchette, M., and Sankoff, D. 1997. The median problem for breakpoints in comparative genomics. *Lect. Notes Comput. Sci.* 1276, 251–263.

Bourque, G., and Pevzner, P. 2002. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Res.* 12, 26–36.

Braga, M., Sagot, M., Scornavacca, C., et al. 2007. The solution space of sorting by reversals. *Lect. Notes Bioinform.* 4463, 293–304.

Caprara, A. 1999a. Formulations and hardness of multiple sorting by reversals. *RECOMB 1999* 84–93.

Caprara, A. 1999b. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Math.* 12, 91–110.

Caprara, A. 2001. On the practical solution of the reversal median problem. *Lect. Notes Comput. Sci.* 2149, 238–251.

Downie, S.R., and Palmer, J.-D. 1992. Use of chloroplast DNA rearrangements in reconstructing plant phylogeny, 14–35. In: Soltis, D., Soltis, P., and Doyle, J.J., eds. *Molecular Systematics of Plants*. Chapman and Hall, New York.

Eriksen, N. 2007. Reversal and transposition medians. *Theoret. Comput. Sci.* 374, 111–126.

Hannenhalli, S., and Pevzner, P. 1995. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Proc. 27th Annu. Symp. Theory Comput.* 178–189.

Hannenhalli, S., and Pevzner, P. 1996. To cut ... or not to cut (applications of comparative physical maps in molecular evolution). *Proc. 7th ACM-SIAM Symp. Discrete Algorithms* 304–313.

Moret, B., Siepel, A., Tang, J., et al. 2002. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. *Lect. Notes Comput. Sci.* 2452, 521–536.

Moret, B., Tang, J., and Warnow, T. 2005. Reconstructing phylogenies from gene-content and gene-order data, 321–352. In Gascuel, O., ed. *Mathematics of Evolution and Phylogeny*. Oxford University Press, New York.

Moret, B., Wyman, S., Bader, D., et al. 2001. A new implementation and detailed study of breakpoint analysis. *Proc. 6th Pacif. Symp. Biocomput.* 583–594.

Pe'er, I., and Shamir, R. 1998. The median problems for breakpoints are NP-complete. *Elec. Colloq. Comput. Complexity* 71.

Pevzner, P., and Tesler, G. 2003. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proc. Natl. Acad. Sci. USA* 100, 7672–7677.

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

Saitou, N., and Nei, M. 1987. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* 4, 406–425.

Sankoff, D., and Blanchette, M. 1998. Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.* 5, 555–570.

Siepel, A. 2001. Exact algorithms for the reversal median problem [Master's thesis]. University of New Mexico, Albuquerque, NM.

Siepel, A. 2003. An algorithm to enumerate sorting reversals for signed permutations. *J. Comput. Biol.* 10, 575–597.

Siepel, A., and Moret, B. 2001. Finding an optimal inversion median: experimental results. *Lect. Notes Comput. Sci.* 2149, 189–203.

Address reprint requests to:
Dr. Jijun Tang
Department of CSE
University of South Carolina
315 Main Street
Columbia, SC 29208

E-mail: jtang@cse.sc.edu