Finding an Optimal Inversion Median: Experimental Results

Adam C. Siepel[1] and Bernard M.E. Moret[2]

Albuquerque, NM 87131, USA
and National Center for Genome Resources
Santa Fe, NM 87505, USA
`acs@ncgr.org`

[2] Department of Computer Science, University of New Mexico
Albuquerque, NM 87131, USA
`moret@cs.unm.edu`

Abstract. We derive a branch-and-bound algorithm to find an optimal inversion median of three signed permutations. The algorithm prunes to manageable size an extremely large search tree using simple geometric properties of the problem and a newly available linear-time routine for inversion distance. Our experiments on simulated data sets indicate that the algorithm finds optimal medians in reasonable time for genomes of medium size when distances are not too large, as commonly occurs in phylogeny reconstruction. In addition, we have compared inversion and breakpoint medians, and found that inversion medians generally score significantly better and tend to be far more unique, which should make them valuable in median-based tree-building algorithms.

1 Introduction

Dobzhansky and Sturtevant [7] first proposed using the degree to which gene orders differ between species as an indicator of evolutionary distance that could be useful for phylogenetic inference, and Watterson *et al.* [23] first proposed the minimum number of chromosomal inversions necessary to transform one ordering into another as an appropriate distance metric. The 1992 study by Sankoff *et al.* [21] included a heuristic algorithm for finding rearrangement distance (which considered transpositions, insertions, and deletions, as well as inversions); it was the first large-scale application and experimental validation of rearrangement-based techniques for phylogenetic purposes and initiated what is now nearly a decade of intense interest in computational problems relating to genome rearrangement (see summaries in [16,19,22]).

While much of the attention given to rearrangement problems may be due to their intriguing combinatorial properties, rearrangement-based approaches to phylogenetic inference are of genuine biological interest in cases in which sequence-based approaches perform poorly, such as when species diverged early or are rapidly evolving [16]. In addition, rearrangement-based phylogenetic methods can suggest probable gene orderings of ancestral species [17,18], while other methods cannot. Furthermore, mathematical models of genome rearrangement have applications beyond phylogeny (see [8,20]).

Recent work on rearrangement distance and sorting by inversions (or *reversals*, as they are often called in computer science) has produced a duality theorem and polynomial-time algorithm for inversion distance between two signed permutations [10], a duality theorem and polynomial-time algorithm for distance in terms of equally weighted translocations and inversions for signed permutations [11], polynomial-time algorithms for sorting by reversals [2,12], and a linear-time algorithm for computing inversion distances [1]. Note that "signed permutations" correspond to genomes for which the direction of transcription of each gene is known as well as the ordering of the genes.

Much recent work on rearrangement-based phylogeny [5,6,14,15,18] stems from an algorithm by Sankoff and Blanchette [17] that iterates over a prospective tree, repeatedly finds medians of the three permutations adjacent to each internal vertex, and uses them to improve the tree until convergence occurs. This method finds locally optimal trees and simultaneously allows an estimation of the configurations of ancestral genomes. These studies have generally used *breakpoint distance* as the basis for finding medians, because it is more easily computable than inversion distance, it assumes no particular mechanism of rearrangement, and the problem of finding a breakpoint median has a straightforward reduction to the well known Travelling Salesman Problem (TSP) [17]. The number of breakpoints between two genomes is the number of genes that are adjacent in one but not the other genome; the breakpoint median of a set of genomes is the ordering of genes that minimizes the sum of the number of breakpoints with respect to each genome in the set.

Breakpoint distance is related to inversion distance (an inversion can remove at most two breakpoints) but the relationship is a loose one. Because it is believed that inversions are the primary mechanism of genome rearrangement for many taxa [13,3], we seek a solution to the median problem based directly on inversion distance. Finding an *inversion median* is known to be NP-hard [4], and to date, no one has reported a reasonably efficient algorithm (approximate or exact) for this problem. (Although in one study [9], inversion medians were obtained for a particular data set using a bounded exhaustive search.)

In this paper, we present a simple yet effective branch-and-bound algorithm to solve the median of three problem exactly. Our approach does not depend on properties specific to inversions, but can be used with any rapidly computable metric. We have evaluated its effectiveness for the case of inversion medians, and found that it obtains optimal medians with reasonable computational effort for a range of parameters that include most realistic instances encountered in phylogenetic analysis. In addition, we have performed a comparison of inversion and breakpoint medians, and found that inversion medians score significantly better in terms of total induced edge length, and tend to be far more unique. These findings suggest that inversion medians, when used within the algorithm of Sankoff and Blanchette, will allow better trees to be computed in fewer iterations.

2 Notation and Definitions

We consider the case where all genomes have identical sets of n genes and inversion is the single mechanism of rearrangement. We represent each genome G_i as a permutation π_i of size n, and we let all pairs of genomes $G_i = (g_{i,1} \ldots g_{i,n})$ and $G_j = (g_{j,1} \ldots g_{j,n})$, in a set of genomes G, be represented by $\pi_i = (\pi_{i,1} \ldots \pi_{i,n})$ and $\pi_j = (\pi_{j,1} \ldots \pi_{j,n})$ such that $\pi_{i,k} = \pi_{j,l}$ iff $G_{i,k} = G_{j,l}$, and $\pi_{i,k} = -1 \cdot \pi_{j,l}$ iff $G_{i,k}$ is the reverse complement of $G_{j,l}$.

We define an *inversion* acting on permutation π from i to j, for $i \leq j$, as that operation which transforms π into $\phi = (\pi_1, \pi_2, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n)$. The minimal number of inversions required to change one permutation π_i into another permutation π_j is the *inversion distance*, which we denote by $d(\pi_i, \pi_j)$ (sometimes abbreviated as $d_{i,j}$).

Let the *inversion median* M of a set of N permutations $\Pi = \{\pi_1, \pi_2, \ldots, \pi_N\}$ be the signed permutation that minimizes the sum $S(M, \Pi) = \sum_{i=1}^{N} d(M, \pi_i))$. Let this sum $S(M, \Pi) = S(\Pi)$ be called the *median score* of M with respect to Π.

For a given number of genes n, we can construct an undirected graph $G_n = (V, E)$ such that each vertex in V corresponds to a signed permutation of size n and two vertices are connected by an edge if and only if one of the corresponding permutations can be obtained from the other through a single inversion; formally, $E = \{\{v_i, v_j\} \mid v_i, v_j \in V$ and $d(\pi_i, \pi_j) = 1\}$. We will call G_n the *inversion graph* of size n. In this graph, the distance between any two vertices, v_i and v_j, is the same as the inversion distance between the corresponding permutations, π_i and π_j. Furthermore, finding the median of a set of permutations Π is equivalent to finding the minimum unweighted Steiner tree of the corresponding vertices in G_n. Note that G_n is very large ($|V| = n! \cdot 2^n$), so this representation does not immediately suggest a feasible graph-search algorithm, even for small n.

Definition 1. *A* shortest path *between two permutations of size n, π_1 and π_2, is a connected subgraph of the inversion graph G_n containing only the vertices v_1 and v_2 corresponding to π_1 and π_2, and the vertices and edges on a single shortest path between v_1 and v_2.*

Definition 2. *A* median path *of a set of permutations Π each of size n is a connected subgraph in the inversion graph of G_n containing only the vertices corresponding to permutations in Π, the vertex corresponding to a median M of Π, and a shortest path between M and each $\pi \in \Pi$.*

Definition 3. *A* trivial median *of a set of permutations Π is a median M that is a member of that set, $M \in \Pi$.*

Definition 4. *A* trivial median path *of a set of permutations Π is a median path that includes only the elements of Π and shortest paths between elements of Π.*

3 General Median Bounds

Because phylogenetic reconstruction algorithms generally work with binary trees in which each internal node has three neighbors, the special case of the median of three genomes is of particular interest. In this section we develop a general bound for the median-of-three problem, one that relies only on the metric property of the distance measure used.

Lemma 1. *The median score $S(\Pi)$ of a set of equally sized permutations $\Pi = \{\pi_1, \pi_2, \pi_3\}$, separated by pairwise distances $d_{1,2}, d_{1,3}$, and $d_{2,3}$, obeys these bounds:*

$$\left\lceil \frac{d_{1,2} + d_{1,3} + d_{2,3}}{2} \right\rceil \leq S(\Pi) \leq \min\left\{ (d_{1,2} + d_{2,3}), (d_{1,2} + d_{1,3}), (d_{2,3} + d_{1,3}) \right\}$$

Proof. The upper bound follows directly from the possibility of a trivial median, and the lower bound from properties of metric spaces (a median of lower score would necessarily violate the triangle inequality with respect to two of π_1, π_2, and π_3; see Figure 1).

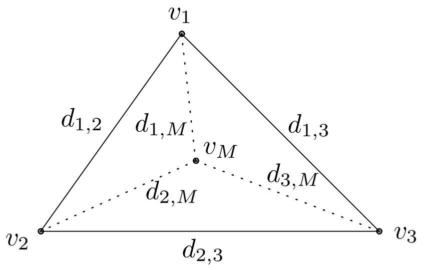

Fig. 1. Let vertices v_1, v_2, and v_3 correspond to permutations π_1, π_2, and π_3, and let vertex v_M correspond to a median M.

Lemma 2. *If three permutations π_1, π_2, and π_3 have a median M that is part of a trivial median path, then M must be a trivial median.*

Proof. Assume to the contrary that π_1, π_2, and π_3 have a trivial median path and have a median M that is not trivial. By Definition 4, M must be on a shortest path between two of π_1, π_2, and π_3. Without loss of generality, assume that the median path runs from π_1 to M to π_2 to π_3. Let $d_{1,2}$, $d_{1,3}$, and $d_{2,3}$ be the pairwise distances between $\{\pi_1, \pi_2\}$, $\{\pi_1, \pi_3\}$, and $\{\pi_2, \pi_3\}$, respectively, and let $d_{M,2} > 0$ be the distance of M from π_2. Then the median score of M is $(d_{1,2} - d_{M,2}) + d_{M,2} + (d_{M,2} + d_{2,3}) = d_{1,2} + d_{M,2} + d_{2,3}$. But this score is greater by $d_{M,2}$ than the score of a trivial median at π_2, so M cannot be a median.

Theorem 1. *Let π_1, π_2, and π_3 be permutations such that π_2 and π_3 are separated by distance $d_{2,3}$, and let ϕ be another permutation separated from π_1, π_2, and π_3 by distances $d_{1,\phi}, d_{2,\phi}$, and $d_{3,\phi}$, respectively. Suppose that ϕ is on a median path P_M of π_1, π_2, and π_3 such that ϕ is on a shortest path between π_1 and a median M. Then the* *score $S(M)$ of M obeys these bounds:*

$$d_{1,\phi} + \left\lceil \frac{d_{2,\phi} + d_{3,\phi} + d_{2,3}}{2} \right\rceil \leq S(M)$$
$$\leq d_{1,\phi} + \min\big\{(d_{2,\phi} + d_{2,3}), (d_{2,\phi} + d_{3,\phi}), (d_{2,3} + d_{3,\phi})\big\}$$

Proof. Let v_1, v_2, and v_3 be vertices corresponding to π_1, π_2, and π_3, in the inversion graph of the appropriate size. In addition, let there be a vertex v_ϕ corresponding to ϕ, as illustrated in Figure 2. We claim that a median path P_M including v_ϕ and M, such that

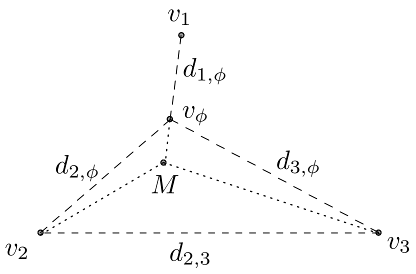

Fig. 2. A median path including v_ϕ can be constructed using a shortest path from v_1 to v_ϕ and any median path of v_ϕ, v_2, and v_3.

v_ϕ is on a shortest path from v_1 to M, can be constructed by combining a shortest path between v_1 and v_ϕ and a median path of v_ϕ, v_2, and v_3. Assume to the contrary that there exists a shorter median path P_{short}, which also includes v_ϕ and M, but does not include the shortest path between v_1 and v_ϕ or does not include a median path of v_ϕ, v_2, and v_3. P_{short} has to include v_1 via a vertex other than v_ϕ and consequently other than M (because v_ϕ is on a shortest path between v_1 and M). By Definition 2, P_{short} must consist only of v_1, v_2, v_3, M, and vertices between them (including v_ϕ), so v_1 must be connected to P_{short} via v_2 or v_3. Consequently, M must be on a shortest path between v_2 and v_3; otherwise including M in P_{short} would result in a score greater than that of a trivial median. Therefore, M is part of a trivial median path, which means that by Lemma 2, M is a trivial median. In particular, M must be the vertex $v_i \in \{v_2, v_3\}$ to which v_1 is connected. Furthermore, our assumptions about ϕ require that v_ϕ be on the shortest path between v_1 and v_i. Then P_{short} includes both the shortest path between v_1 and v_ϕ and the median path of v_ϕ, v_2, and v_3, and we obtain the desired contradiction.

Because P_M can be constructed by combining a shortest path between v_1 and v_ϕ, and a median path of v_ϕ, v_2, and v_3, its score is equivalent to the sum of the distance between v_1 and v_ϕ ($d_{1,\phi}$), and the score of the median of v_ϕ, v_2, and v_3. By applying Lemma 1 to the latter, we obtain the desired bound.

4 The Algorithm

Algorithm `find_inversion_median` is presented below. It is essentially a branch-and-bound search for an optimal inversion median that uses Theorem 1 to prune a search tree based on the inversion graph and to prioritize among search branches.

Prioritization is managed using a *priority stack*—which always returns an item of highest priority, but returns items of equal priority in *last-in-first-out* order. Because the range of possible priorities is small, we use a fixed array of priority values, each pointing to a stack and so can execute `push` and `pop` operations in fast constant time. Using stacks rather than the more conventional queues in this application is not required for correctness, but, by inducing depth-first searching among alternatives of equal cost, rapidly produces a good upper bound for the search.

The algorithm begins by establishing upper and lower bounds for the solution using Lemma 1 (steps 1 and 2) and priming the priority stack with a best-scoring vertex (steps 3 and 4). Then it enters a main loop (step 6) in which it repeatedly `pop`s the "most promising" vertex from the priority stack, finds all of its as-yet-unvisited neighbors (step 8), and evaluates each one for feasibility. Neighbors are obtained by generating all $\binom{n}{2}$ possible permutations that can be produced from a vertex by a single inversion. Neighbors of a vertex v can be ignored if they are not farther from the origin than is v (step 9); such vertices will be examined as neighbors of another vertex if they can feasibly belong to a median path. The best possible score (i.e., lower bound) for a vertex w is is used as the basis for prioritization. Best and worst possible scores are calculated using the bounds of Theorem 1 (steps 10 and 11) and maintained for all vertices present in the priority stack. Vertices can be pruned when their best possible scores exceed the current global upper bound. The global upper bound can be lowered when a vertex is found that has a lesser upper bound (step 13). The search ends when no vertex in the queue has a best-possible score lower than the upper bound (step 7) or a score equal to the global lower bound is found (step 12).

The algorithm will return a permutation M only if M is a true median of the inputs π_1, π_2, and π_3. Assume to the contrary that a permutation M' returned by the algorithm is not a true median. Because the algorithm returns the permutation having the lowest median score of all of the permutations (vertices) it visits (steps 5 and 13), it must not have visited some median. If the algorithm did not visit some median, then either it pruned all paths to medians or it exited before reaching any median.

Suppose the algorithm pruned all paths to medians. It only prunes vertices when their best possible scores are lower than the current global upper bound, M_{max}. Note that the global upper bound always corresponds to the actual median score of a vertex that has been visited (steps 2 and 13), so it cannot be wrong. Consider a median M with at least one median path P_M. By Definition 2, P_M must include at least one path between M and each of the vertices v_1, v_2, and v_3 corresponding to π_1, π_2, and π_3. The algorithm proceeds by examining neighbors of an origin $\psi_{orig} \in \{\pi_1, \pi_2, \pi_3\}$. Therefore, if the algorithm pruned all paths to M, then it must have pruned a vertex on the path between ψ_{orig} and M. But the best scores of such vertices are calculated using the lower bound of Theorem 1 (step 10), which we have shown to be correct. Therefore, the algorithm cannot have pruned the shortest paths to medians.

Algorithm 1: find_inversion_median

```
Input: Three signed permutations of size $n$: $\pi_1$, $\pi_2$, and $\pi_3$. Assume a function distance($\pi_i, \pi_j$) that returns the inversion distance between $\pi_i$ and $\pi_j$ in linear time.
Output: An optimal inversion median $M$.
begin
       $d_{1,2} \leftarrow$ distance($\pi_1, \pi_2$);
       $d_{1,3} \leftarrow$ distance($\pi_1, \pi_3$);
       $d_{2,3} \leftarrow$ distance($\pi_2, \pi_3$);
1      $M_{min} \leftarrow \lceil \frac{d_{1,2}+d_{1,3}+d_{2,3}}{2} \rceil$;
2      $M_{max} \leftarrow \min\{(d_{1,2}+d_{2,3}), (d_{1,2}+d_{1,3}), (d_{2,3}+d_{1,3})\}$;
       Initialize priority stack $s$ for range $M_{min}$ to $M_{max}$;
       $(\psi_{orig}, \psi_1, \psi_2) \leftarrow (\pi_i, \pi_j, \pi_k)$ such that $\{\pi_i, \pi_j, \pi_k\} = \{\pi_1, \pi_2, \pi_3\}$ and
       $d_{i,j} + d_{i,k} = M_{min}$;
3      create vertex $v$ with $v_{label} = \psi_{orig}$, $v_{dist} = 0$, $v_{best} = M_{min}$, $v_{worst} = M_{max}$;
4      push($s, v$);
5      $M \leftarrow \psi_{orig}$;
       $d_{sep} \leftarrow d_{\psi_1,\psi_2}$;
       $stop \leftarrow$ false ;
6      while s is not empty and stop = false do
           pop($s, v$);
7          if $v_{best} \geq M_{max}$ then $stop \leftarrow$ true ;
           else
8              foreach $\{w \mid w$ is an unmarked neighbor of $v\}$ do
                   $w_{dist} \leftarrow$ distance($w_{label}, \psi_{orig}$);
9                  if $w_{dist} \leq v_{dist}$ then continue;
                   mark $w$;
                   $d_{\psi_1} \leftarrow$ distance($w_{label}, \psi_1$);
                   $d_{\psi_2} \leftarrow$ distance($w_{label}, \psi_2$);
10                 $w_{best} \leftarrow w_{dist} + \lceil \frac{d_{\psi_1}+d_{\psi_2}+d_{sep}}{2} \rceil$;
11                 $w_{worst} \leftarrow w_{dist} + \min\{(d_{\psi_1}+d_{sep}), (d_{\psi_1}+d_{\psi_2}), (d_{sep}+d_{\psi_2})\}$;
12                 if $w_{worst} = M_{min}$ then $M \leftarrow w_{label}$; stop $\leftarrow$ true ;
                   else
                       if $w_{best} < M_{max}$ then push($s, w, w_{best}$);
13                     if $w_{worst} < M_{max}$ then
                           $M \leftarrow w_{label}$; $M_{max} \leftarrow w_{worst}$;
end
```

Suppose instead that the algorithm exited before reaching a median. The algorithm can exit for one of three reasons:

1. The priority stack s becomes empty (step 6);
2. The next item returned from s has a best possible score greater than or equal to the current global upper bound (step 7);
3. A vertex w is found with a worst possible score equal to the global lower bound (step 12);

Case 1 can occur only if all vertices have been visited, or if all remaining neighbors have been pruned (because except when the algorithm stops for another reason, each new neighbor is either pruned or pushed onto s). If all vertices have been visited, then a median must have been visited. We have shown above that all neighbors on paths to a median cannot have been pruned. Because s always returns a vertex v such that no other vertex in s has a lower best-possible score than v, and because all neighbors that are not pruned are added to s, case 2 can only occur if a median has been visited or if all paths to medians have been pruned. We have shown that all paths to medians cannot have been pruned. Therefore, if case 2 occurs, a median must have been visited. In case 3, w must be a median, since the global lower bound is set directly according to Lemma 1 (step 1), which we have shown to be correct.

Thus, none of these three cases can arise before a median has been found, and the algorithm must return a median. The worst-case running time of the algorithm is $O(n^{3d})$, with $d = \min\{d_{1,2}, d_{2,3}, d_{1,3}\}$, but as would be expected with a branch-and-bound algorithm, the average running time appears to be much better.

5 Experimental Method

We implemented `find_inversion_median` in C, reusing the linear-time distance routine (as well as some auxiliary code) from GRAPPA [1], and we evaluated its performance on simulated data. All test data was generated by a simple program that creates multiple sets of three permutations by applying random inversions to the identity permutation, such that each set of three permutations represents three taxa derived from a common ancestor under an inversions-only model of evolution. In addition to the number of genes n to model and the number of sets s to create, this program accepts a parameter i that determines how many random inversions to apply in obtaining the permutation for each taxon. Thus, if $n = 100$, $i = 10$, and $s = 10$, the program generates 10 sets of 3 signed permutations, each of size 100, and obtains each permutation by applying 10 random inversions to the permutation $+1, +2, \ldots, +100$. A random inversion is defined as an inversion between two random positions i and j such that $1 \leq i, j \leq n$ (if $i = j$, a single gene simply changes its sign). When i is small compared to n, each permutation in a set tends to be a distance of $2i$ from each other.

We used several algorithmic engineering techniques to improve the efficiency of `find_inversion_median`. For example, we avoided dynamic memory allocation and reused records representing graph vertices. We were able to gain a significant speedup by optimizing the hash table used for marking vertices: a custom hash table offered a fourfold increase in the overall speed of the program, as compared with UNIX's *db* implementation. With circular genomes, we achieved a further improvement in performance by hashing on the *circular identity* of each permutation rather than on the permutation itself. We define the circular identity of a permutation as that equivalent permutation that begins with the gene labeled $+1$. By hashing on circular identities, we reduced the number of vertices to visit and the number of permutations to mark by approximately a factor of $2n$.

To improve performance further, we adapted our sequential implementation to run in parallel on shared-memory architectures. Two steps in the algorithm are readily parallelizable: the major loop (step 6), during each iteration of which a new vertex is popped from the priority stack, and the minor loop (step 8), in which the neighbors of a vertex v are generated, examined for marks, and evaluated for feasibility as medians. We enabled parallel processing at both levels, using `pthreads` for maximum portability across shared-memory architectures. With careful use of semaphores and `pthreads` mutex functions, we were able to reduce the cost of synchronization among threads to an acceptable level.

6 Experimental Results

6.1 Performance of Bounds

Being especially concerned with the effectiveness of the pruning strategy, we have chosen as a measure of performance the number of vertices V of the inversion graph that the algorithm visited. In particular, we have taken V to be the number of times the program executed the loop at step 8 of the algorithm. Note that the number of calls to `distance` is approximately $3V$. We recorded the distribution of V over many experiments, in which we used various values for the number of genes n and the number of inversions per tree edge i. Figure 3 is typical of our results. It summarizes 500 experi-

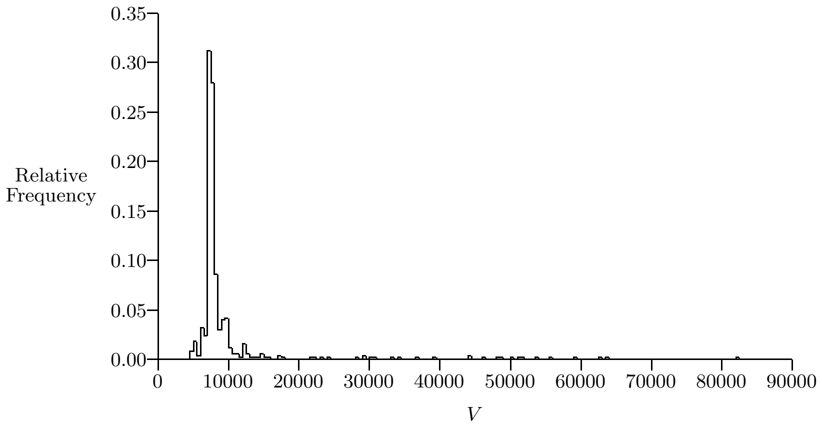

Fig. 3. Distribution of the number of vertices visited in the course of 500 experiments with $n = 50$ and $i = 7$.

ments with $n = 50$ and $i = 7$ and shows a roughly exponential distribution, with high relative frequencies in a few intervals having small V: in 87% of the experiments, fewer than 10,000 vertices were visited, and in 95%, fewer than 20,000 were visited. This figure demonstrates that the algorithm generally finds a median rapidly, but occasionally becomes mired in an unprofitable region of the search space. We have observed that the tail of the exponential distribution becomes more substantial as i grows larger with respect to n.

In order to characterize typical performance, we recorded the statistical medians of V as n and i varied independently. The results are shown in Figures 4 and 5. For

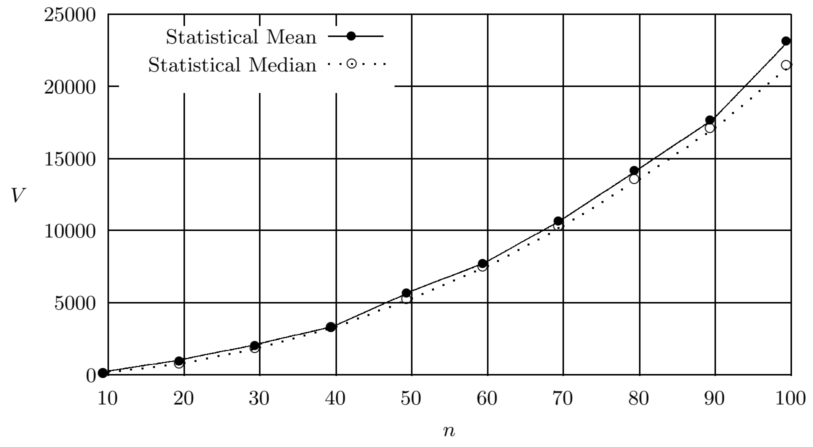

Fig. 4. Statistical median of the number of vertices visited V for $i = 5$ and $10 \leq n \leq 100$, over 50 experiments for each value of n.

comparison, we have also plotted the mean values of V. Note that, at least for $i = 5$, the median and mean of V appear to grow quadratically over a significant range of values for n; a simple fit yields $f(n) = 2.1n^2$ for the median values. Note also that, for $n = 50$, the median of V grows approximately linearly with i, at least as long as i remains small (mean V grows somewhat faster than median V). To put the observed rate of growth into perspective, note that in the theoretical worst-case of $O(n^{3d})$, because $d \approx 2i$ and $V = O(\frac{n^{3d}}{n}) = O(n^{(6i-1)})$, one would see (given $i = 5$ and $n = 50$) growth of V with n^{29} and 50^{6i-1}.

6.2 Running Time and Parallel Speedup

We have tested program `find_inversion_median` sequentially on a 700 MHz Intel Pentium III with 128MB of memory, and using various levels of parallelism on a Sun E10000 with 64 333 MHz UltraSPARC processors and 64GB of memory. Figure 6 shows average running times for $i = 5$ and n between 50 and 125. Sequential running times are shown for the Sun and Intel processors and parallel running times for the Sun with the number of processors $p \in \{1, 2, 4, 6\}$. In all cases, the average time to find

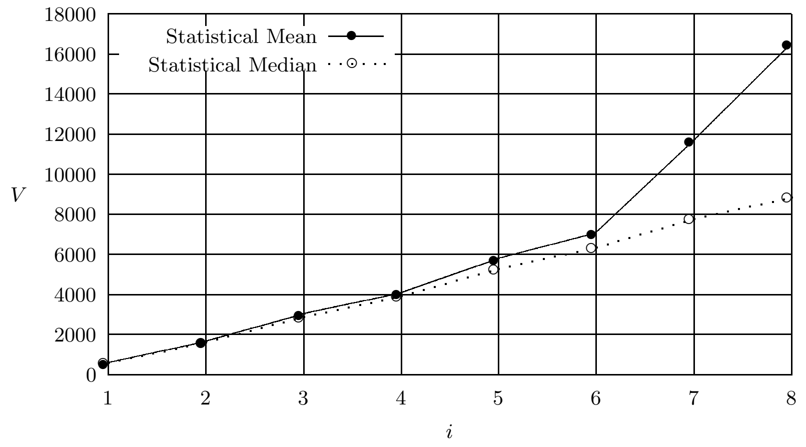

Fig. 5. Statistical median of the number of vertices visited V for $n = 50$ and $1 \leq i \leq 8$, plotted with mean of V. The number of experiments for each value of i is 50.

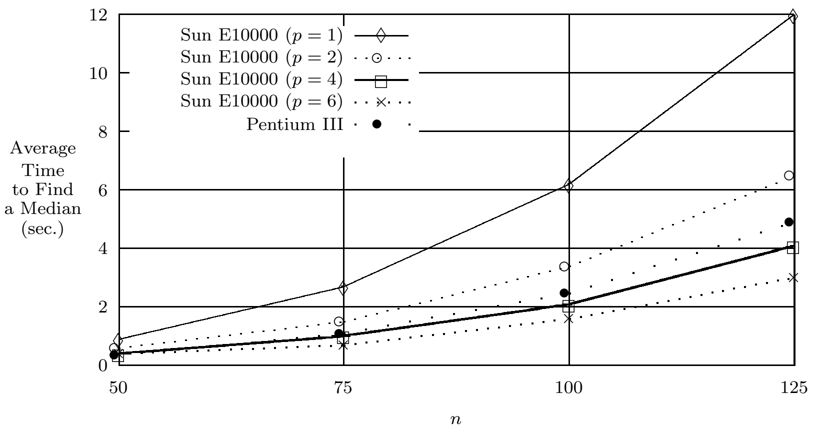

Fig. 6. Sequential and parallel running times for $i = 5$ and $n \in \{50, 75, 100, 125\}$. Each data point represents an average taken over 10 experiments. Parallel configurations used parallelism only in the minor loop of the algorithm.

a median is about 12 seconds or less. Observe that for $n = 100$ (a realistic size for chloroplast or mitochondrial genomes) medians can generally be found in an average of about 2 seconds using a reasonably fast computer. We should note that the memory requirements for the program are considerable, and that the level of performance shown here is partly a consequence of the large amount of RAM available on the Sun.

It is evident from Figure 6 that we achieve a good parallel speedup for small p, but that the benefits of parallelization begin to erode between $p = 4$ and $p = 6$ (this tendency becomes more pronounced at $p = 8$, which we have not plotted here for clarity of presentation). Anecdotal evidence suggests that the cause of this trend is a combination of the overhead of synchronization and uneven load balancing among the computing threads. We also observed that parallelism in the minor loop of the algorithm was far more effective than parallelism in the major loop, presumably because the heuristic for prioritization is sufficiently effective that the latter strategy results in a large amount of unnecessary work.

6.3 Inversion Medians vs. Breakpoint Medians

Using program `find_inversion_median`, we evaluated the significance of inversion medians, by comparing them with breakpoint medians, trivial medians, and "actual" medians (i.e., the ancestral permutations from which observed taxa actually arose - in this case, always equal to the identity permutation). Figure 7, which shows results over $1 \leq i \leq 5$ for $n = 25$, is typical of what we observed. It demonstrates that

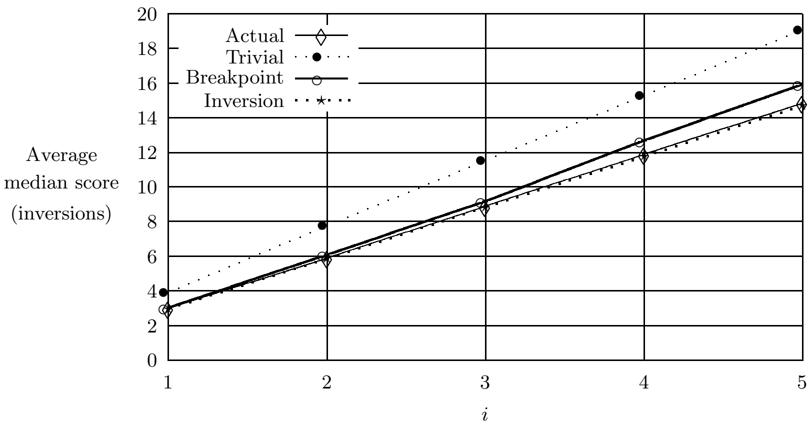

Fig. 7. Comparison of inversion medians with breakpoint medians, trivial medians, and actual medians, for $n = 25$. Averages were taken over 50 experiments.

inversion medians achieve comparable scores to actual medians[1] and that breakpoint medians, when scored in terms of inversion distance, perform significantly worse. A comparison in terms of inversion median scores is clearly biased in favor of inversion medians; however, if it is true that inversion distances are (in at least some cases) more meaningful than breakpoint distances, then these results suggest that inversion medians are worth obtaining.

We used a slight modeification of program `find_inversion_median` to find *all* optimal medians and thus to characterize the extent to which inversion medians are unique. An example of our results is shown in Figure 8, which describes the number

[1] Inversion medians are slightly better than actual medians when i becomes large with respect to n, because saturation begins to cause convergence between taxa.

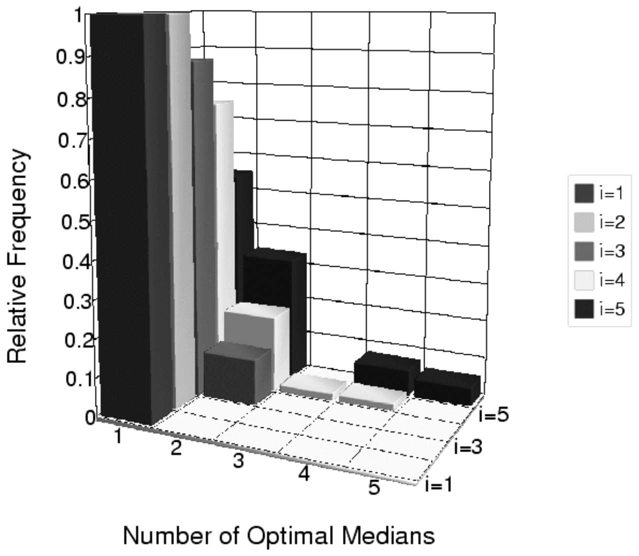

Fig. 8. Distribution of number of optimal medians in the course of 50 experiments for $n = 15$ and $1 \leq i \leq 5$.

of optimal inversion medians for $n = 15$ and $1 \leq i \leq 5$, over 50 experiments for each value of i. Observe that, when i is small compared to n (roughly $i \leq 0.15n$), the inversion median is virtually always unique; and even when i is moderately large with respect to n (roughly $0.15n < i \leq 0.3n$)[2], the inversion median is unique or nearly unique most of the time. This finding stands in stark contrast with breakpoint medians, which are only very rarely unique.

In addition, we observed a strong relationship between unique inversion medians and actual medians. For example, with $n = 15$ and $i = 1$, for which all inversion medians were unique, 49 out of 50 inversion medians were identical to actual medians; similarly, for $n = 15$ and $i = 2$, 48 out of 50 were identical to actual medians (in both cases the exceptional inversion medians differed from actual medians by a single inversion). As i becomes greater compared to n, this relationship weakens but remains significant. For example, with $n = 15$ and $i = 4$, 38 out of 50 inversion medians were unique, and 22 of those 38 were identical to actual medians (an additional 10 non-unique inversion medians equaled actual medians).

7 Future Work

The strength and weakness of the current algorithm both lie in its generality. On the one hand, our approach depends only on elementary properties of metric spaces and thus

[2] Recall that the distance between permutations is approximately $2i$ and that random permutations tend to be separated by a distance of approximately n. The effects of saturation are evident at $i = 0.2n$ and are pronounced at $i = 0.3n$.

extends easily to the case of equally weighted inversions, translocations, fissions, and fusions; furthermore, it could also be used with weighted rearrangement distances. (One should note, however, that the running time is a direct function of the cost of evaluating distances; we can compute exact breakpoint and inversion distances, but no efficient algorithm is yet known for more complex distance computations.) On the other hand, our approach does not exploit the unique structure of the inversion problem; as shown elsewhere in this volume by A. Caprara, restricting the algorithm to inversion distances only and using aspects of the Hannenhalli-Pevzner theory enables the derivation of tighter bounds and thus also the solution of larger instances of the inversion median problem.

Many simple changes to our current implementation will considerably reduce the running time. For example, the current implementation does not "condense" genomes before processing them—i.e., it does not convert subsequences of genes shared among all three genomes to single "supergenes". Preliminary experiments indicate that condensing genomes yields very significant improvements in performance when i is small relative to n. Distance computations themselves, while already fast, can be further improved by reusing previous computations, since a move by the algorithm makes only minimal changes to the candidate permutation. Finally, we can use the Kaplan-Shamir-Tarjan algorithm, in combination with metric properties, to prepare better initial solutions (by walking halfway through shortest paths between chosen permutations), thus considerably decreasing the search space to be explored.

References

1. D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. In *Proceedings 7th Workshop on Algorithms and Data Structures WADS91*. Springer Verlag, 2001. to appear in LNCS.
2. P. Berman and S. Hannenhalli. Fast sorting by reversal. In D. Hirschberg and E. Myers, editors, *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, pages 168–185, 1996.
3. M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:GC11–GC17, 1996.
4. A. Caprara. Formulations and complexity of multiple sorting by reversals. In S. Istrail, P.A. Pevzner, and M.S. Waterman, editors, *Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB-99)*, pages 84–93, Lyon, France, April 1999.
5. M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L.-S. Wang, T. Warnow, and S. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in *Campanulaceae*. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, pages 99–122. Kluwer Academic Press, 2000.
6. M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L.-S. Wang, T. Warnow, and S. Wyman. A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology ISMB-2000*, pages 104–115, 2000.
7. T. Dobzhansky and A.H. Sturtevant. Inversions in the chromosomes of *Drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.

8. D.S. Goldberg, S. McCouch, and J. Kleinberg. Algorithms for constructing comparative maps. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*. Kluwer Academic Press, 2000.
9. S. Hannenhalli, C. Chappey, E.V. Koonin, and P.A. Pevzner. Genome sequence comparison and scenarios for gene rearrangements: A test case. *Genomics*, 30:299–311, 1995.
10. S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing*, pages 178–189, 1995.
11. S. Hannenhalli and P.A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 581–592, 1995.
12. H. Kaplan, R. Shamir, and R.E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal of Computing*, 29(3):880–892, 1999.
13. A. McLysaght, C. Seoighe, and K.H. Wolfe. High frequency of inversions during eukaryote gene order evolution. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, pages 47–58. Kluwer Academic Press, 2000.
14. B.M.E. Moret, D.A. Bader, S. Wyman, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In R.B. Altman, A.K. Dunker, L. Hunter, K. Lauderdale, and T.E. Klein, editors, *Pacific Symposium on Biocomputing 2001*, pages 583–594. World Scientific Publ. Co., 2001.
15. B.M.E. Moret, L.-S. Wang, T. Warnow, and S.K. Wyman. New approaches for reconstructing phylogenies from gene-order data. In *Proceedings 9th International Conference on Intelligent Systems for Molecular Biology ISMB-2001*, 2001. to appear in *Bioinformatics*.
16. Pavel A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, 2000.
17. D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570, 1998.
18. D. Sankoff, D. Bryant, M. Deneault, B.F. Lang, and G. Burger. Early eukaryote evolution based on mitochondrial gene order breakpoints. *Journal of Computational Biology*, 7(3/4):521–535, 2000.
19. D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, Y. Xu, and M. Zhang, editors, *Topics in Computational Biology*. MIT Press, 2001.
20. D. Sankoff, V. Ferretti, and J.H. Nadeau. Conserved segment identification. *Journal of Computational Biology*, 4(4):559–565, 1997.
21. D. Sankoff, G. Leduc, N. Antoine, B. Paquin, and B.F. Lang. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences*, 89:6575–6579, 1992.
22. J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, 1997.
23. G.A. Watterson, W.J. Ewens, and T.E. Hall. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.