ALGORITHMS FOR MULTIPLE GENOME REARRANGEMENT BY SIGNED REVERSALS

SHIQUAN WU, XUN GU
Center of Bioinformatics and Biological Statistics
Iowa State University, Ames, IA 50011, USA
sqwu@cs.iastate.edu, xgu@iastate.edu

We discuss a multiple genome rearrangement problem by signed reversals: Given a collection of genomes, we generate them in the minimum number of signed reversals. It is NP-hard and equivalent to finding an optimal Steiner tree to connect the genomes by reversal paths. We design two algorithms to find the optimal Steiner nodes of the problem: Neighbor-perturbing algorithm and branch-and-bound algorithm. The first one is a polynomial running time approximation algorithm. It searches for the optimal Steiner nodes by perturbing initial Steiner nodes nearby their neighborhoods and improving them better and better until convergence. The second one is an exact exponential running time algorithm for a median problem. It finds the optimal Steiner node by checking all candidates that satisfy the necessary conditions for optimal Steiner nodes. We implement the algorithms into two programs respectively and show by experimental examples that they are more efficient than other similar ones, such as GRAPPA, BPAnalysis, and MGR, etc.

1 Introduction

Phylogenetic trees show the evolutionary relationship of species. They are inferred based on DNA or amino acid sequences. Three major methodologies are widely used for the purpose: Parsimony method, maximum likelihood method, and distance-based method [10,11]. Genome projects have generated enormous types of informative genome data for phylogeny reconstructions [9]. One interesting and challenging problem is about comparative genome-wide gene orders for multiple genomes and phylogeny inference. Most earlier discussions focus on edit distances defined by local mutations such as insertion, deletion, substitution [10,11], and furthermore recombination [17]. Recently, distance was widely discussed based on the orders of genes. Breakpoint analysis first investigated such kind of distances [14,15]. However, the biological meaning of breakpoint distance is not very clear. To overcome this, reversal distance was then introduced. It is defined as the minimum number of signed/unsigned reversals needed to account the difference of the gene orders of two genomes [14,15]. Sorting by reversal is an important problem in comparative genomics. It transforms one genome into another by signed or unsigned reversals [2,3,4]. Sorting by unsigned reversals is NP-hard [7,8], while sorting by signed reversals is polynomial-time solvable [12,13]. Indeed, the signed reversal distance between any two signed

permutations can be computed by linear running time algorithms [1].

A multiple genome rearrangement problem discusses how to reconstruct optimal distance-based phylogenetic trees for a collection of genomes [15,16,18]. The problem is NP-hard [8]. Therefore, the practical purpose turns out to find good approximation solutions. Various approximation algorithms/programs (such as GRAPPA, BPAnalysis, MGR, etc.) are developed for solving different versions of multiple genome rearrangement problems [5,6,15,16,18].

In this paper, we discuss a multiple genome rearrangement problem by signed reversals [18]: Given a collection of genomes, we generate them in the minimum number of signed reversals. It is NP-hard and equivalent to finding an optimal Steiner tree to connect the genomes by reversal paths. We focus on designing efficient algorithms to find the optimal Steiner nodes for the genomes.

The rest of the paper consists of five parts. In Section 2, we introduce the mathematical model of our multiple genome rearrangement problem. In Section 3, we review some related previous algorithms on the problem. In Section 4, we design an approximation algorithm and an exact algorithm for the problem. In Section 5, we compare our algorithms/programs with other similar ones (such as GRAPPA, BPAnalysis, and MGR, etc.) and show that ours are more efficient than those. And finally, we discuss some possible further work and appliocations in Section 6. We obtain the following results.

(1) Design a neighbor-perturbing algorithm. It is a polynomial running time approximation algorithm and searches for the optimal Steiner nodes for all given genomes by perturbing initial Steiner nodes nearby their neighborhoods and improving them better and better until convergence.

(2) Find the necessary conditions for optimal Steiner nodes.

(3) Based on the necessary conditions, we design a branch-and-bound algorithm. It is an exact exponential running time algorithm for a median problem and finds the optimal Steiner node by checking all possible candidates that satisfy the necessary conditions for optimal Steiner nodes.

(4) Two programs are implemented from the algorithms, respectively. Experimental examples show that our algorithms/programs are more efficient than other similar ones, such as GRAPPA, BPAnalysis, and MGR, etc.

2 Problem and model

First of all, we introduce the notations and set up the mathematical model of the multiple genome rearrangement problem by signed reversals [18].

Definition (1) Let X denote an alphabet (e.g., the set of genes) and $|X| = n$. Assume $p = (p_1 p_2 \cdots p_{i-1} \underline{p_i p_{i+1} \cdots p_j} p_{j+1} \cdots p_n)$ is a signed permutation (i.e., genome) on X. Each p_i stands for a gene and its sign represents its strand

of DNA. A positive (or negative) sign means that the gene is in the forward (or backward) strand. A signed reversal on a segment $[i, j]$ of p is defined as the mutation $r(p; i, j) = (p_1 p_2 \cdots p_{i-1} \underline{-p_j \cdots - p_{i+1} - p_i} p_{j+1} \cdots p_n)$, where the segment $[i, j]$ changes both its orientation and signs (i.e., strand).

(2) Define $N(p) = \{q | q = r(p; i, j)$ for all $1 \leq i \leq j \leq n\}$, which is the collection of all permutations that can be obtained from p by one signed reversal. $N(p)$ is called a reversal neighborhood (or sphere) of p. Let P be a collection of permutations. A reversal neighborhood (or sphere) of P is defined as $N(P) = \cup_{p \in P} N(p)$. Define $N_1(P) = N(P)$, $N_2(P) = N(N_1(P))$, and $N_k(P) = N(N_{k-1}(P))$, the $k-$neighborhood (or $k-$sphere) of P.

MGRBSR Problem (Multiple Genome Rearrangement By Signed Reversal) Given a collection of permutations $G = \{g_1, g_2, \cdots, g_m\}$, we generate G from some $p \in G$ in the minimum number of signed reversals, i.e., to find a collection of signed permutations $t_k (1 \leq k \leq s)$ on X such that (1) any g_j is obtained from p by a series $t_{k_1}, t_{k_2}, \cdots, t_{k_j}$, where each $t_{k_{i+1}}$ is obtained from t_{k_i} by one signed reversal, i.e., $t_{k_{i+1}} \in N(t_{k_i})$, and (2) s is minimized. Denote $d(p, G) = s$.

Various kinds of multiple genome rearrangement problems are widely discussed. The problems are NP-hard [8] and equivalent to finding optimal Steiner trees for G in the space of permutations [6,10,14,15,16,18]. We here discuss the **MGRBSR Problem** involving only pure signed reversals.

3 Previous related work

Many algorithms are designed to solve various kinds of multiple genome rearrangement problems. Most of them are approximation algorithms.

Breakpoint analysis The breakpoint distance between two genomes is defined as the number of consecutive pairs of genes that are adjacent in one genome but not in the other. Breakpoint analysis is one of the earliest methods for genome rearrangement problem based on gene orders. There are several efficient approximation algorithms and programs for the problems, such as GRAPPA and BPAnalysis [5,15]. However, breakpoint analysis gives many approximation solutions that do not have a clear biological meaning. It could not find a more biologically accurate rearrangement (reversal) distance [6].

Grid search Sankoff *et al* [16] discussed a multiple genome rearrangement problem for reversals and transpositions. For three genomes, a local optimal solution was searched upon a grid consisting of a series of reversal paths (see Fig.1a). A general problem with m genomes is recursively approximated by a

series of groups of three genomes.

Nearest path search Wu and Gu [18] discussed a multiple genome rearrangement problem for pure signed reversals. An approximation solution was obtained by a nearest path search algorithm upon a simple grid (see Fig.1b). For three genomes, suppose g_1 and g_2 are the pair with the greatest reversal distance. The algorithm at first finds a shortest reversal path P_1 from g_1 to g_2, then improves P_1 to another better reversal path P_2 (where P_2 is in some neighborhood of P_1 and closer to g_3 than P_1), and repeatedly improves a series of reversal paths to get a closest reversal path P_k (in some neighborhood of P_{k-1}) to g_3. Next, it constructs a grid by P_k and g_3. Finally, a local optimal solution is obtained by searching on the grid. This simplifies Sankoff's *Grid search* algorithm [16] and is shown to be efficient by experimental examples.

Greedy split Bourque and Pevzner [6] designed a multiple genome rearrangement algorithm based on recursively greedy splitting. For the given genomes $g_1, g_2, \cdots, g_m$, the algorithm at first connects the nearest pair, say g_1 and g_2, by a shortest reversal path. Suppose $g_1, g_2, \cdots, g_{m-1}$ have been connected by some reversal paths. Then the algorithm finds a nearest point (the split site) on all these paths and connects g_m to the split site (see Fig.1c). The algorithm can be applied to both unichromosomal and multichromosomal genomes.

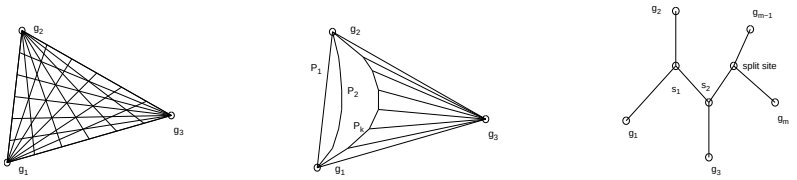

Figure 1: (a) Grid search (b) Nearest path search (c) Greedy split

4 Approximation algorithms

In this section, we design two algorithms: Neighbor-perturbing algorithm and branch-and-bound algorithm. Multiple genome rearrangement problems by signed reversals can be efficiently solved by applying these two algorithms.

4.1 Neighbor-perturbing algorithm

Any minimum spanning tree is a 2-approximation solution of the optimal Steiner tree (to prove this, add a multiple edge for each edge of the optimal

Steiner tree T^* and compare a spanning tree T_0 with these edges. Then we have an inequality on the lengths of the trees: $L(T_{MinSpanningTree}) \leq L(T_0) \leq 2L(T^*)$). Any minimum spanning tree can be regarded as a trivial Steiner tree with each given g_i as a trivial Steiner node. Within $N(g_i)$, we can find some Steiner node better than g_i. For a median problem, we at first choose g_2 as an initial Steiner node s_0. Then find a Steiner node $s_1 \in N(s_0)$ better than s_0, and next find $s_2 \in N(s_1)$ better than s_1, and so on. The Steiner nodes are improved better and better. This is the idea of our neighbor-perturbing algorithm. Neighbor-perturbing algorithm is to perturb an initial Steiner node within its neighborhood and improve it better and better until convergence.

The neighbor-perturbing algorithm consists of two steps: Initialization and iteration. For a median problem, in the initialization step, we rearrange the genomes so that the path $g_1g_2g_3$ forms a minimum spanning tree. g_2 is then chosen as the initial Steiner node, i.e., $s_0 = g_2$. We start perturbing from g_2 so that the iteration converges fast. In the iteration step, we perturb each s_i and find a better $s_{i+1} \in N(s_i)$ (Fig.2b). The Steiner nodes are improved better and better from $s_0(= g_2)$ to $s_1, s_2, \cdots, s_i, \cdots$, until s_i finally converges at s (Fig.2a).

Algorithm	**Neighbor-perturbing (Median problem)**
Input	Permutations: $G = \{g_1, g_2, g_3\}$.
Output	Optimal Steiner node.
Step 1	Initialization:
(1.1)	Rearrange each g_i: $d(g_1, g_3) = max\{d(x, y) \mid x, y \in G\}$.
(1.2)	Initial Steiner node $s_0 = g_2$.
Step 2	Iteration:
(2.1)	Suppose s_i have been defined. Check each $x \in N(s_i)$: Denote $d^*(x, G) = d(x, g_1) + d(x, g_2) + d(x, g_3)$. If x is better than s_i, i.e., $d^*(x, G) < d^*(s_i, G)$, then define $s_{i+1} = x$.
(2.2)	Repeat (2.1) until convergence.

For the general multiple genome rearrangement problem, in the initialization step, we choose a minimum spanning tree T as an initial Steiner tree. We rearrange the genomes so that g_1 and g_m are the two longest leaves in T. There are at most $m - 2$ Steiner nodes. If we perturb g_1 and g_m to get two Steiner nodes, it may take a longer time for the convergences. So we choose $g_2, g_3, \cdots, g_{m-1}$ as the initial Steiner nodes so as to get a better approximation solution and a fast convergence. In the iteration step, we perturb and improve each Steiner node s by checking all $x \in N(s)$. If $S \cup \{x\} - \{s\}$ generates a better Steiner tree than S does, then replace s by x (Fig.2c). The Steiner nodes/tree are improved from the initial ones better and better until convergence.

Algorithm **Neighbor-perturbing (General case,** see Fig.2c)
Input Permutations: $G = \{g_1, g_2, \cdots, g_m\}$.
Output Optimal Steiner nodes S.
Step 1 Initialization:
(1.1) Define the reversal distance graph for G (vertex set) by all pairwise signed reversal distances $d(x, y)(x, y \in G)$.
(1.2) Find a minimum spanning tree and its two longest leaves x_0 and y_0. Choose the initial $S = G - \{x_0, y_0\}$.
Step 2 Iteration:
(2.1) For each $s \in S$, check all $x \in N(s)$:
If the optimal Steiner tree generated by $S \cup \{x\} - \{s\}$ is better than that generated by S, then replace s by x.
(2.2) Repeat (2.1) until convergence.

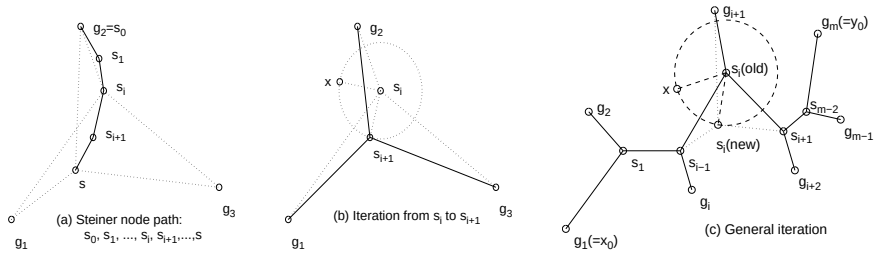

Figure 2: (a) Initial Steiner node $s_0 = g_2$. Recursively iterate/update the Steiner nodes: $s_0, s_1, \ldots, s_i, \cdots$ until convergence (at s). (b) Process of iteration from s_i to s_{i+1}: Suppose s_i have been generated. Check each $x \in N(s_i)$, if x is better than s_i, then update $s_{i+1} = x$. If no better x is found, then return s_i (optimal). (c) General iteration: Assume g_1 and g_m are the two longest leaves in the minimum spanning tree. Then $g_2, g_3, \cdots, g_{m-1}$ are chosen as the initial Steiner nodes. Each s_i is generated from $g_{i+1}(i = 2, 3, \cdots, m-1)$ by a series of iterations. In each iteration, for Steiner node s_i, check all $x \in N(s_i)$, if $S \cup \{x\} - \{s_i\}$ generates a better Steiner tree, then replace s_i by x, i.e., s_i is updated by a new s_i. The three edges $s_{i-1}s_i$, $g_{i+1}s_i$, $s_{i+1}s_i$ are replaced by three new $s_{i-1}s_i$, $g_{i+1}s_i$, $s_{i+1}s_i$, respectively. The old Steiner tree is updated by a new one. The iterations are repeated until convergence.

Theorem 1 *The Neighbor-perturbing algorithm is a 2-approximation algorithm and has a running time* $O(m^3n^4logm)$.

Proof The algorithm is obvious 2-approximation. $O(m^3n^4logm)$ is from all steps: $O(mn)$ for all $d(g_i, g_j)$ and $O(m^2n^3logm)$ for minimum spanning trees.

4.2 Branch-and-bound algorithm

In this part, we design an exact algorithm, branch-and-bound algorithm, to find the optimal Steiner node for a median problem. Assume $G = \{g_1, g_2, g_3\}$. We

rearrange the genomes so that the path $g_1g_2g_3$ forms a minimum spanning tree. Denote $d^*(x,G) = d(x,g_1) + d(x,g_2) + d(x,g_3)$. For any optimal Steiner s, the correspoding optimal Steiner tree must at first connect s to some $s_1 \in N(g_2)$, and then connect s_1 to g_2 (see Fig.3a). $g_2s_1g_1$ and $g_2s_1g_3$ are two optimal reversal paths from g_2 to g_1 and g_3, respectively. They share a common part g_2s_1 ($s_1 \in N(g_2)$). We can see that $d(g_2,s_1) + d(s_1,g_i) = d(g_2,g_i)(i = 1,3)$ and $d^*(s_1,G) < d^*(g_2,G)$, i.e., s_1 is a Steiner node better than g_2.

Generally, any optimal Steiner tree connects s and g_2 through a series of Steiner nodes $s_0(= g_2), s_1, s_2, \cdots, s_j, \cdots$ such that $s_j \in N(s_{j-1})(j \geq 0)$ (see Fig.3b). $s_0s_1s_2\cdots s_j\cdots g_1$ and $s_0s_1s_2\cdots s_j\cdots g_3$ are two optimal reversal paths from g_2 to g_1 and g_3, respectively. They share a common part $s_0s_1s_2\cdots s_j$. Each s_q is better than s_{q-1} ($q \geq 1$). We state this as a theorem.

Theorem 2 (Necessary conditions for an optimal Steiner node) *Let* $G = \{g_1, g_2, g_3\}$ *(genomes). If* s *is an optimal Steiner node, then*

$$g(g_2,s) + d(s,g_i) = g(g_2,g_i)\ (i = 1,3). \tag{1}$$

Moreover, there exist a series of Steiner nodes $s_0 = g_2, s_1, s_2, \cdots, s_j, \cdots, s_p = s$ *such that* $g(g_2,s_j) + d(s_j,g_i) = g(g_2,g_i)$ $(i = 1,3; j \geq 0)$ *and* $d^*(s,G) < d^*(s_j,G) < d^*(s_{j-1},G) < d^*(s_0,G)$, $(p > j \geq 2)$.

Proof By the previous statements and Fig.3.

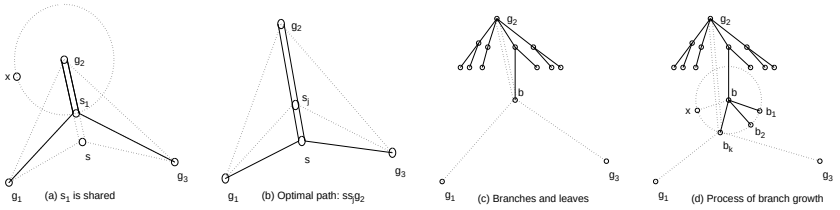

Figure 3: (a) Two optimal reversal paths $g_2s_1g_1$ and $g_2s_1g_3$ share a common part g_2s_1. $d(g_2,s_1) + d(s_1,g_i) = d(g_2,g_i)(i = 1,3)$ and $d^*(s_1,G) < d^*(g_2,G)$. (b) s is an optimal Steiner node. $g_2\cdots s_j\cdots g_1$ and $g_2\cdots s_j\cdots g_3$ are two optimal reversal paths from g_2 to g_1 and to g_3, respectively. $g(g_2,s_j) + d(s_j,g_i) = g(g_2,g_i)(i = 1,3; j \geq 0)$ and $d^*(s,G) < d^*(s_j,G) < d^*(s_{j-1},G) < d^*(s_0,G)$ for each j. (c) Generation relations are shown by the branches/leaves, each leaf is obtained from one branch (candidate) of last generation. (d) Process of growing branches: For each leaf b, check all $x \in N(b)$, if x satisfies Eq. (1), then take x as a new branch/leaf. $b_1, b_2, \cdots, b_k$ are from b. Update s by x if $d^*(x,G) < d^*(s,G)$.

Theorem 2 gives us the idea of our branch-and-bound algorithm. Any s satisfying Eq. (1) is a candidate of the optimal Steiner node. By $s_j \in N(s_{j-1})(j \geq 0)$ and Eq. (1), we can recursively search for the optimal Steiner node from all possible candidates starting from g_2. At first, we check all $x \in N(g_2)$. If

$s_1 \in N(g_2)$ satisfies Eq. (1), then it is one of the first generation candidates of g_2. Next, we find all second generation candidates s_2 from each first generation candidate s_1 by $s_2 \in N(s_1)$ and Eq. (1). We repeatedly find all candidates for all generations until convergence. The optimal Steiner node can be found by $min_j d^*(s_j, G)$ over all candidates s_j. The relations between all candidates of different generations can be expressed as a tree (see Fig.3c-d). Suppose at some iteration step we have found some candidates of some generations (see Fig.3c). Let B be the set of all candidates (denoted by branches) and L the set of all the current generations (denoted by leaves). For each leaf $b \in L$, check all $x \in N(b) - B$, if x satisfies Eq. (1), then x is a new candidate and it is then put into both the branch set B and the leaf set L. If x is better than s, i.e., $d^*(x, G) < d^*(s, G)$, then update the optimal Steiner node s by x. New leaves (new generation candidates) $b_1, b_2, \cdots, b_k$ are obtained from b (see Fig.3d). b is removed from L and becomes an internal node in next iteration. We repeat the process until convergence, i.e., no new branch can be found.

Branch-and-bound algorithm consists of two steps: Initialization and iteration. It starts at g_2, grows branches and update the optimal s (see Fig.3d).

Algorithm **Branch-and-bound** (Median problem, see Fig.3)
Input Permutations: $G = \{g_1, g_2, g_3\}$.
Output Optimal Steiner node s.
Step 1 Initialization:
(1.1) Rearrange each g_i: $d(g_1, g_3) = max\{d(x, y) | x, y \in G\}$.
(1.2) Initial $B = \{g_2\}$, $L = \{g_2\}$, $s = g_2$.
Step 2 Iteration: (see Fig.3d)
(2.1) Grow branches and update the optimal Steiner node:
For each $b \in L$, check every $x \in N(b) - B$:
If $g(g_2, x) + d(x, g_i) = g(g_2, g_i)(i = 1, 3)$, then
x is a new candidate: $B = B \cup \{x\}$, $L = L \cup \{x\}$.
If $d^*(x, G) < d^*(s, G)$, then update $s = x$.
Remove b from L after all $x \in N(b) - B$ is checked.
(2.2) Repeat (2.1) until convergence.

Theorem 3 *The branch-and-bound algorithm finds the optimal Steiner node in a running time* $O(n^3 e_n^{\lfloor r_0/2 \rfloor})$. *Where* $r_0 = min\{d(x, y) | x, y \in G\}$ *and* e_n *deponds on* n *(e.g.,* $e_n \leq 3$ *in the examples in next section).*

Proof By Theorem 2 for the optimum and Theorem 3[18] for the running time.

5 Comparisons and examples

We show that our algorithms are more efficient than other similar ones. Theoretically, our neighbor-perturbing algorithm is more efficient than the greedy-

split algorithm [6] and other similar algorithms. The greedy-split algorithm greedily connects all given genomes one after another. Usually, a greedy algorithm may generate an approximation solution far away from the optimal one in the worse case. However, our neighbor-perturbing algorithm starts from a minimum spanning tree, which is a 2-approximation solution. The neighbor-perturbing algorithm improves each Steiner node again and again and has more chances to get the optimal one. This is its advantage over other algorithms.

The two algorithms are implemented into two programs, GenomeNP and GenomeBnB, respectively, which take a reversal distance program "signed.c" as a subroutine written by Hannenhalli (http:// www-hto.usc.edu/software). We show by experimental examples that our algorithms/programs are more efficient than other similar ones, such as GRAPPA, BPAnalysis, MGR, etc.

Example 1 $G = \{p, q_1, q_2\}$ (the genomes of human, sea urchin,fruit fly) [6,16].

$p =$ 26 13 17 12 -24 15 18 32 -2 -16 -3 -33 4 -28 7 5 1 10 19 25 22 11 29 14 20 -21 -8 6 30 -23 9 27 31

$q_1 =$ 26 4 25 22 5 1 -28 19 11 29 20 -21 6 9 27 8 30 23 -24 16 14 -2 32 3 -31 15 -7 33 10 13 17 12 18

$q_2 =$ 26 14 17 -28 -6 -21 23 -30 22 11 20 9 -8 -29 -16 -25 -2 -19 -10 -1 -7 -5 -13 -4 33 3 -32 -18 -15 24 -12 27 31

By GenomeNP, we obtain an optimal Steiner node:

$s =$ 26 13 17 12 -24 15 18 32 -2 -3 -33 4 14 20 -21 -8 6 -7 28 -29 -11 -22 -25 -19 -10 -1 -5 -16 30 -23 9 27 31

Therefore, $d(s,p) = 4, d(s,q_1) = 20, d(s,q_2) = 15$, and $d(p,\{q_1,q_2\}) \leq 39$. GRAPPA and BPAnalysis obtained $d(p,\{q_1,q_2\}) = 43$, **MGR** also obtained $d(p,\{q_1,q_2\}) \leq 39$ [5,6,15]. However, can 39 be improved better? **MGR** did not answer. Since $d(s,p)+d(s,q_1) \geq d(p,q_1) = 24$, $d(s,p)+d(s,q_2) \geq d(p,q_2) = 19$, $d(s,q_1) + d(s,q_2) \geq d(q_1,q_2) = 32$, so $d(p,\{q_1,q_2\}) \geq [(24+19+32)/2] = 38$. GenomeBnB checks all possible candidates and obtains $d(p,\{q_1,q_2\}) = 39$.

Example 2 $G = \{A, B, \cdots, K\}$ are the following the genomes of: human, asterina pectinifera, paracentrotus lividus, drosophila yakuba, artemia franciscana, albinaria coerulea, cepaea nemoralis, katharina tunicata, lumbricus terrestris, ascaris suum, onchocerca volvulus, respectively [6].

$A =$ 1 -32 17 2 23 12 3 20 6 30 7 8 21 31 24 9 -10 -18 11 33 -28 19 14 34 13 25 4 22 -29 26 5 35 -15 -27 -16 -36

$B =$ 1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 -13 -5 -22 -4 -25 -20 -36 17 -26 34 -16 -35 15 -24 -27 29 -28

$C =$ 1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 28 -29 27 24 -15 35 16 -34 26 -17 36 20 25 4 22 5 13

$D =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 22 -29 26 5 35 -16 -36

$E =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 26 5 35 -22 -29 -16 -36

$F =$ 1 34 13 24 28 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 -26 -6 -32 31 8 -33 -3 22 5 23

$G =$ 1 34 13 24 15 10 28 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 26 -6 -32 -33 -3 31 8 22 5 23

$H =$ 1 17 2 12 -19 -9 -21 -8 -7 33 -32 -11 -10 28 -4 -25 -24 -13 -34 -14 -26 -16 -36 -35 -29 -20 -18 3 23 15 30 27 22 6 31 5

$I =$ 1 27 2 17 36 20 3 29 10 11 35 12 30 21 9 19 18 28 33 7 8 16 26 14 34 13 24 15 32 25 4 22 23 6 31 5

$J =$ 1 16 26 17 20 2 21 13 6 9 15 28 34 10 7 35 18 14 32 27 36 4 12 23 25 31 5 22 30 29 19 11 24 3 33 8

$K =$ 1 35 10 30 29 11 24 3 23 15 25 27 26 7 14 36 4 19 12 22 20 2 21 13 6 16 32 28 17 34 9 18 31 5 33 8

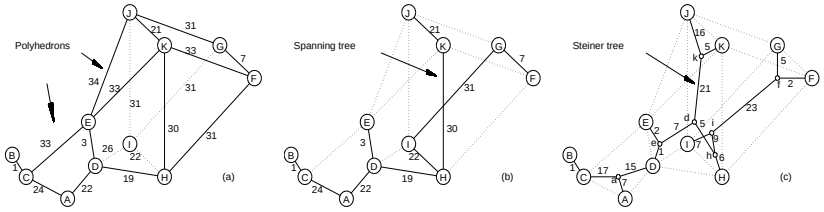

Figure 4: (a) The distance graph for the 11 genomes. (b) The minimum spanning tree of the genomes. (c) Optimal Steiner tree. The total distance $149 < 150$ (obtained by **MGR**).

We at first construct the signed reversal distance graph for the genomes from all pairwise signed reversal distances (Fig.4a), then find a minimum spanning tree T of the graph (Fig.4b), and finally generate the following optimal Steiner nodes a, d, e, f, h, i and k by perturbing T by GenomeNP (Fig.4c).

$a =$ 1 -32 17 -11 10 -9 -27 18 33 -28 19 14 34 13 25 4 22 -29 26 5 35 30 7 8 21 31 24 2 23 12 3 20 6 -15 -16 -36

$d =$ 1 25 31 5 -35 -29 -22 -12 -17 -23 -3 14 34 13 24 15 30 27 2 20 6 4 -32 -11 -10 28 -33 7 8 21 9 19 -18 -26 -16 -36

$e =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 22 29 26 5 35 -16 -36

$f =$ 1 34 13 24 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 -26 -6 -32 31 8 -33 -28 -3 22 5 23

$h =$ 1 17 12 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -25 -23 -3 18 22 29 35 36 16 26 14 34 13 24 15 30 27 2 20 6 31 5

$i =$ 1 27 2 16 26 14 34 13 24 15 10 11 35 12 -18 -19 -9 -21 -8 -7 -33 -28 30 17 36 20 -4 -25 -23 -3 22 29 32 6 31 5

$k =$ 1 35 18 27 32 -9 -34 -17 -28 10 30 29 11 24 3 23 -7 -26 -16 -6 -13 -21 -2 -20 -22 -12 -19 -4 -36 -14 15 25 31 5 33 8

GenomeNP perturbs A, E, F, I, K, D, H into a, e, f, i, k, d, h, respectively. The minimum spanning tree (see Fig.4b, length: 180) is perturbed into the optimal Steiner tree (Fig.4c) with a length 149, which is better than 150 obtained by **MGR**[6]. Our algorithms/programs are better than **MGR** and others.

6 Discussion

We have designed two algorithms (Neighbor-perturbing algorithm and branch-and-bound algorithm) and two programs (GenomeNP and GenomeBnB) to find the optimal Steiner nodes. The experimental examples show that the algorithms/programs are more efficient than other similar ones. Our discussions are based on signed reversals. However, we can extend the discussions to other mutations, such as insertion, deletion, substitution, reversal, transposition, translocation, fusion, fission, etc. The algorithms have great potential applications in a wide range of genomes including multichromosome genomes.

In the iteration steps of the algorithms, we always check all nodes in $N(x)$. We can get better approximation solutions by checking $N_k(x)$ for some greater k. By randomly checking only some nodes in $N(x)$, or $N_k(x)$, we can also expect to get some satisfactory approximation solutions. Similarly, if we randomly check only some of the candidates, the branch-and-bound algorithm then becomes a nice approximation algorithm.

Acknowledgment This work is supported by the NIH grant RO1 GM62118 to X.G. and Wu is also is supported in part by NSF of China (19771025).

References

1. Bader, D.A., Moret, B.M.E. and Yan, M., (2001) *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Lecture Notes in Computer Science, 2125: 365-376.
2. Bafna,V. and Pevzner, P. (1994) *Genome rearrangements and sorting by reversals.* In Proc. 34th IEEE Symp. of the Foundations of Computer Science, 148-157. IEEE Computer Society Press.

3. Bafna, V. and Pevzner, P. (1995) *Sorting permutations by transpositions.* Proceedings of the 6th Annual Symposium on Discrete Algorithms, pages 614-623. ACM Press, January 1995.
4. Bafna, V. and Pevzner, P. (1996) *Genome rearrangements and sorting by reversals.* SIAM Journal on Computing, 25(2):272-289.
5. Blanchette, M., Bourque, G., and Sankoff, D. (1997) *Breakpoint phylogenies.* In *Genome Information Workshop (GIW 1997),* (eds. Mivano S. and Takagi, T.), pp.25-34. University Academy Press, Tokyo.
6. Bourque, G. and Pevzner, P. (2002) *Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species.* Genome Res.12:26-36.
7. Caprara, A. (1997) *Sorting by Reversals is Difficult.* Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97), ACM Press.
8. Caprara, A. (1999) *Formulations and hardness of multiple sorting by reversals.* Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB'99), ACM Press.
9. Gu, X., (2000) *A simple evolutionary model for genome phylogeny inference based on gene content.* Comparative genomics (ed. Sankoff and Nadeau), pp.515-524. Kluwer Academic Publishers.
10. Gusfield,D., Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997.
11. Hillis, D., Motitz, C., Mable, B., Molecular systematics. Sinauer Association, Inc. Massachusetts USA, 1996.
12. Hannenhalli, S. and Pevzner, P. (1995a) *Transforming men into mice (polynomial algorithm for genomic distance problems.* Proc. IEEE Symp. of the Foundations of Computer Science.
13. Hannenhalli, S. and Pevzner, P. (1995b) *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals).* Proc. of 27th Ann. ACM Symp. on Theory of Comput., 178-189.
14. Sankoff, D. (1999) *Genome rearrangement with gene families.* Bioinfomatics, 15:909-917.
15. Sankoff, D. and Blanchette, M. (1998). *Multiple genome rearrangement and breakpoint phologeny.* J. Comp. Biol. 5: 555-570.
16. Sankoff, D., Sudaram, G. and Kececioglu, J. (1996) *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.
17. Wu, S. and Gu, X., (2001) *A greedy algorithm for optimal recombination.* Lecture Notes on Computer Science, 2108:86-90.
18. Wu, S. and Gu, X., (2002) *Multiple Genome Rearrangement By Reversals.* Pacific Symposium on Biocomputing 7:259-270.