

Journal of Bioinformatics and Computational Biology
© Imperial College Press

**COMPUTING THE REVERSAL DISTANCE BETWEEN GENOMES
IN THE PRESENCE OF MULTI-GENE FAMILIES
VIA BINARY INTEGER PROGRAMMING**

JAKKARIN SUKSAWATCHON

*Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Chulalongkorn University
Bangkok 10330, Thailand
and
Department of Computer Science, Faculty of Science,
Burapha University, Chonburi, Thailand
E-mail: jakkarin@buu.ac.th*

CHIDCHANOK LURSINSAP

*Advanced Virtual and Intelligent Computing (AVIC) Center
Department of Mathematics, Chulalongkorn University
Bangkok 10330, Thailand
E-mail: lchidcha@chula.ac.th*

MIKAEL BODÉN

*School of Information Technology and Electrical Engineering
The University of Queensland
Queensland, Australia
E-mail: mikael@itee.uq.edu.au*

Received (17 January 2006)
Revised (Day Month Year)
Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

Hannenhalli and Pevzner developed the first polynomial-time algorithm for the combinatorial problem of sorting signed genomic data. Their algorithm determines the minimum number of reversals required for rearranging a genome to another – but only in the absence of gene duplicates. However, duplicates often account for 40% of a genome. In this paper, we show how to extend Hannenhalli and Pevzner’s approach to deal with genomes with multi-gene families. We propose a new heuristic algorithm to compute the nearest reversal distance between two genomes with multi-gene families via binary integer programming. The experimental results on both synthetic and real biological data demonstrate that the proposed algorithm is able to find the reversal distance with high accuracy.

Keywords: Genome Rearrangement; Multi-gene Families; Binary Integer Programming; Reversal Distance; Gene Duplication.

1. Introduction

The proliferation of gene content and gene order data has taken phylogenetic studies to a new level. Gene order and orientation change very slowly from evolutionary events such as reversal, transposition and translocation. The difference between the order and orientation of genes on a chromosome in two genomes can be used as a measure of evolutionary distance.⁵ Hannenhalli and Pevzner¹ developed a theory for *signed permutations* (of genes) and an algorithm for computing the reversal distance between two genomes in polynomial time – indicating the extent of genome rearrangement required. Henceforth, a sign (+ or -) is associated with each gene representing its transcriptional orientation, *i.e.* on which of the two complementary DNA strands the gene is located. El-Mabrouk^{2,3} extended the Hannenhalli-Pevzner theory to compute the edit distance for inversions and deletions, and for inversions and insertions not resulting from gene duplication. The best running time for *sorting by reversals* is sub-quadratic¹⁴, while the *signed reversal distance* can be computed in linear time.⁴

Duplicated genes account for about 38% of the human genome, 44% in bacterial genomes, and about 40% in archaeobacterial genomes.⁶ The Hannenhalli-Pevzner theory does not account for duplicated genes and the algorithm is consequently producing misleading results for real genomic data. Several models have been developed to account for gene duplication for the analysis of genome rearrangement. Sankoff⁷ designed a branch-and-bound algorithm to find an *exemplar* distance between genomes. The idea is to delete all but one gene in a gene family to minimize the reversal distance between any two genomes. The *exemplar* identification problem is unfortunately NP-hard.⁸ Nguyen *et al.*⁹ proposed a divide-and-conquer approach for calculating the *exemplar* distance by partitioning the gene families into disjoint subsets. Again, all copies but one of each gene had to be deleted in each two genomes causing side-effects like loss of data and accuracy.¹⁰ Recently, Chen *et al.*¹¹ proposed an efficient and effective heuristic algorithm for solving the *signed reversal distance with duplicates* problem using the two NP-Hard techniques of *minimum common partition* and *maximum cycle decomposition*. The signed reversal distance with duplicates problem is NP-Hard, even when the maximum size of a gene family is limited to two.¹¹

In this paper, we ask how the minimum reversal distance can be determined efficiently and accurately in the presence of multi-gene families. We propose a new heuristic algorithm to find the canonical permutation of gene duplicates that obtains the nearest *minimal signed reversal distance for multi-gene families* (SRDMF). We use the concept of a breakpoint graph¹ to cope with gene duplicates. The key idea of our method is to create an *incomplete breakpoint graph* which allows for exploring gene-gene relationships across the two genomes, efficiently and accurately, by maximizing the number of graph cycles. The optimization technique *Binary Integer Programming* (BIP) is applied to find the set of *gray* edges (edges that play a central role in the breakpoint graph) that minimize the reversal distance.

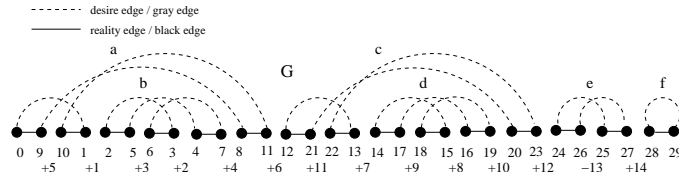


Fig. 1. A breakpoint graph G for the permutation π with respect to the identity permutation of size $n = 14$. Cycles a, b, c, d, e, and f have size 3, 3, 3, 3, 2 and 1.

The SRDMF algorithm is tested on both synthetic and real biological datasets (from human, mouse and rat X chromosomes), and compared with the ALG-SRDD algorithm.¹¹ The experimental results demonstrate that our algorithm outperforms the ALG-SRDD algorithm in terms of accuracy of determining the reversal distance (based on the known minimal reversal distance).

The rest of the paper is organized as follows. Section 2 outlines the necessary preliminaries. Section 3 explains SRDMF – the proposed heuristic algorithm. Section 4 presents the experimental results and section 5 concludes the study.

2. Preliminaries

The proposed approach is based on Hannenhalli and Pevzner’s *breakpoint graph*¹ for sorting signed permutations by reversals. To understand our extension, we briefly outline the basic concepts of the Hannenhalli-Pevzner theory.

Let π and ϕ be signed permutations of size n , such that $\pi = (\pm\pi_1, \dots, \pm\pi_n)$ and $\phi = (\pm\phi_1, \dots, \pm\phi_n)$. Let the unsigned permutation $\pi' = (\pi'_0, \pi'_1, \dots, \pi'_{2n}, \pi'_{2n+1})$ be defined such that $\pi'_0 = 0$, $\pi'_{2n+1} = 2n + 1$ and for all i , each π_i is replaced by $(2\pi_i - 1, 2\pi_i)$ if $\pi_i > 0$, and replaced by $(2|\pi_i|, 2|\pi_i| - 1)$, otherwise. The unsigned permutation ϕ' is defined in the same way as π' . We will only discuss the distance from a genome π to the *identity* genome $\phi = (1, 2, \dots, n)$. Without loss of generality, we always compute the distance between any two genomes π and ϕ .

To find the *minimum number of reversals* required to transform π to ϕ , Hannenhalli and Pevzner¹ introduced a cyclic graph (G) called *breakpoint graph* with two types of edges, i.e. black and gray edges, constructed from π' and ϕ' . Our proposed solution to the studied problem is based on the concept of breakpoint graph.¹ Throughout this paper, the configuration of the breakpoint graph is adopted. An example of the breakpoint graph for $\pi = (+5, +1, +3, +2, +4, +6, +11, +7, +9, +8, +10, +12, -13, +14)$ is shown in Fig. 1. This graph decomposes into a set of disjoint color-alternating cycles. By the size of a cycle, we mean the number of black edges the cycle contains. The number of cycles of breakpoint graph is maximized when $\pi = \phi$.

The k -th *reversal* $\rho_k(i, j)$, $1 \leq i, j \leq n$, of $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ transforms π into $(\pi_1, \dots, -\pi_j, -\pi_{j-1}, \dots, -\pi_i, \dots, \pi_n)$. The *reversal distance*, denoted by $d_{rev}(\pi)$, is the minimum number of series of reversals $\rho_1(i_1, j_1) \rho_2(i_2, j_2), \dots, \rho_t(i_t, j_t)$ required

to transform π to ϕ . The value of $d_{rev}(\pi)$ of permutation length n is given by the formula:¹

$$d_{rev}(\pi) = b(\pi) - c(\pi) + h(\pi) + f(\pi) \quad (1)$$

where $b(\pi)$ is the number of black edges, $c(\pi)$ is the number of cycles in G , $h(\pi)$ its number of hurdles, and $f(\pi)$ is equal to 1 if G is a fortress and zero otherwise.^a

From Fig. 1, the permutation π has 15 black edges, 6 cycles, 2 hurdles, and is a non-fortress. By using the Eq.(1), the reversal distance of the permutation is $15 - 6 + 2 + 0 = 11$.

It is obvious that to achieve the optimum $d_{rev}(\pi)$, the number of cycles, $c(\pi)$ must be increased. This is well-recognized in relation to the *single-gene family* reversal problem.^{16,17} However, the maximization of $c(\pi)$ has not yet been exploited for the multi-gene family reversal problem. Additionally, the values of $b(\pi)$, $h(\pi)$, and $f(\pi)$ also have the influence on the value of $d_{rev}(\pi)$. The value of $b(\pi)$ can be resolved by using the definition of breakpoint graph. However, the values of $h(\pi)$ and $f(\pi)$ can not be known in advance and estimated values must be assigned to them.

3. Computing the Nearest Minimal Signed Reversal Distance for Multi-gene Families (SRDMF)

In case of duplicated genes, let $A = \{g_1, g_2, \dots, g_k\}$ be a set of genes (a uni-chromosome permutation) relating to π of length n in the following way. Each g_i is represented by either a symbol, a character or a number. For each gene g_i ($1 \leq i \leq m$), let $K(g_i)$ be the number of occurrences of g_i having either + or - sign in π and ϕ such that $K(g_i) \geq 1$. Gene g_i is called a *single* if it is the only member of a gene family in that genome, $K(g_i) = 1$. Otherwise g_i is referred to as a member of a *multi-gene family*. Note that, $K(g_i)$ in π must be equal to $K(g_i)$ in ϕ .^b Since the breakpoint graph cannot be applied to capture multi-gene families, each multi-gene member element, π_i , must be systematically renamed so that each name is unique. The renaming allows the breakpoint graph to be used to determine a reversal distance.

Consider the example shown in Table 1. Elements π_1 and π_4 contain the same gene named 2 and π_3 , π_8 , and π_{10} contain the same gene named 3. These two gene families are rearranged and relocated at new positions in ϕ . To uniquely rename each element, first, all elements in ϕ are named according to their positions and assigned to $\phi^{(new)}$. Note that $\phi^{(new)}$ is the identity genome. Each π_i in π is named in correspondence with the element in $\phi^{(new)}$. For example, ϕ_{10} , whose name is +9, is renamed as +10 in $\phi_{10}^{(new)}$. Therefore, π_{11} , whose name is +9, must also be renamed as +10 in $\pi_{11}^{(new)}$. Here, genes 2 and 3 are from multi-gene families.

^aSee Hannenhalli and Pevzner¹ for the definitions of black and gray edges, hurdles, and fortresses.

^bWe use the method of X. Chen¹¹ to construct the groups of gene families.

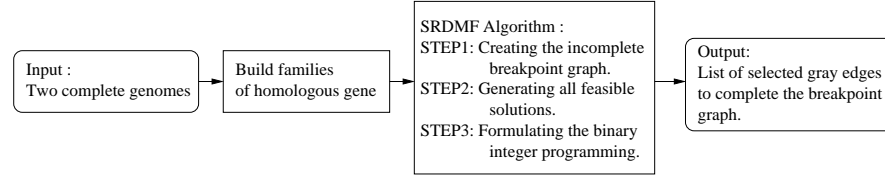


Fig. 2. The outline of SRDMF algorithm.

Table 1. An example of two copies of an element **2** and three copies of an element **3**, and the renaming results, $\pi^{(new)}$ and $\phi^{(new)}$ by temporary names x_1, x_2 and y_1, y_2 , and y_3 .

position	1	2	3	4	5	6	7	8	9	10	11	12
π	+2	-1	+3	-2	-5	+6	+7	+3	-8	+3	+9	+4
ϕ	+1	-2	+3	+2	+5	+6	+7	+8	+3	+9	+3	+4
$\pi^{(new)}$	x_1	-1	y_1	x_2	-5	+6	+7	y_2	-8	y_3	+10	+12
$\phi^{(new)}$	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12

Symbols x_1 and x_2 are introduced as *temporary names* for gene 2 at $\pi_1^{(new)}$, and $\pi_4^{(new)}$, and symbols y_1, y_2 and y_3 are introduced as *temporary names* for gene 3 at $\pi_3^{(new)}$, $\pi_8^{(new)}$, and $\pi_{10}^{(new)}$. The renaming results for both $\pi^{(new)}$ and $\phi^{(new)}$ (the identity genome) are shown in Table 1.

Since gene 2 in π is signed, the elements x_1 and x_2 in $\pi^{(new)}$ can be assigned four possible names from sets $\{+2, -2\}$ and $\{+4, -4\}$ and, similarly, the elements y_1, y_2 , and y_3 in $\pi^{(new)}$ can be assigned three possible names from $\{+3, +9, +11\}$. Hence, for this example, there are $(2 \times 2!) \times 3!$ possible name assignments. Each assignment may result in a different reversal distance, $d_{rev}(\pi)$, calculated from Eq. (1) or from any other reliable reversal distance tools.^{4,13} The minimal $d_{rev}(\pi)$ can be found by generating and testing all possible assignments. However, the processing time grows exponentially with the number of duplications. Therefore, we turn to an efficient heuristic method (called SRDMF algorithm) for assigning a final name to each temporary name. The steps are illustrated in Figure 2 and each step of SRDMF is described separately below.

STEP 1: Creating the incomplete breakpoint graph

In order to find the reversal distance, the breakpoint graph of $\pi^{(new)}$ must be created. The breakpoint graph created by connecting only some black and gray edges to those $\pi_j^{(new)}$ with definite names is considered an *incomplete* breakpoint graph. The remaining black and gray edges will be augmented after all $\pi_j^{(new)}$ with temporary names are assigned definite names. This proposed concept is different from the concepts of incomplete graph in multi-chromosomal genome rearrangements.^{15,16,17}

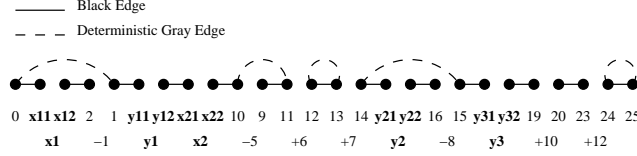


Fig. 3. An example of an *incomplete* breakpoint graph for $\pi^{l(new)}$. The names of x_1 , x_2 can be either $+2$: (3, 4), or -2 : (4, 3), and either $+4$: (7, 8), or -4 : (8, 7) and the names of y_1 , y_2 and y_3 can be either $+3$: (5, 6), $+9$: (17, 18) or $+10$: (19, 20). Hence, the names of x_{11} , and x_{21} must be 3, 4, 7, or 8, and the names of x_{12} , and x_{22} must be 3, 4, 7, or 8. In the same way, the name of y_{11} , y_{21} , and y_{31} must be 5, 17, or 19 and the name of y_{12} , y_{22} , and y_{32} must be 6, 18, or 20.

Their breakpoint graphs have no temporary names and all elements of π_i are in single-gene families. We are only concerned with a uni-chromosome. Consider the example shown in Table 1. A signed permutation $\pi^{(new)}$ becomes an unsigned permutation $\pi^{l(new)}$, where x_1 , x_2 are replaced by the variables, (x_{11}, x_{12}) , (x_{21}, x_{22}) , and y_1 , y_2 and y_3 are replaced by (y_{11}, y_{12}) , (y_{21}, y_{22}) and (y_{31}, y_{32}) , respectively. The final names of x_i , for $1 \leq i \leq 2$ and y_j , for $1 \leq j \leq 3$ are so far unknown. The example incomplete breakpoint graph of $\pi^{l(new)}$ is shown in Fig. 3.

With all variables in place, all black edges can be identified. These black edges are $(0, x_{11})$, $(x_{12}, 2)$, $(1, y_{11})$, (y_{12}, x_{21}) , $(x_{22}, 10)$, $(9, 11)$, $(12, 13)$, $(14, y_{21})$, $(y_{22}, 16)$, $(15, y_{31})$, $(y_{32}, 19)$, $(20, 23)$, and $(24, 25)$. However, only the gray edges not coinciding with variables can be identified in the first instance. The remaining potential gray edges connecting variables are identified during the resolution of the actual names of the variables. All gray edges are classified into two groups, *deterministic* and *non-deterministic* edges, according to the following characteristics.

Deterministic gray edge: A deterministic gray edge is a gray edge connecting $(\pi_i^{l(new)}, \pi_j^{l(new)})$, $1 \leq i, j \leq 2n + 2$, if **(1)** $\pi_i^{l(new)}$ is even and $\pi_j^{l(new)}$ is odd, and **(2)** $\pi_i^{l(new)} = \pi_j^{l(new)} + 1$ or $\pi_i^{l(new)} = \pi_j^{l(new)} - 1$. For example, the deterministic gray edges in Fig. 3 are $(0, 1)$, $(10, 11)$, $(12, 13)$, $(14, 15)$, and $(24, 25)$.

Non-deterministic gray edge: Any gray edge, $(\pi_i^{l(new)}, \pi_j^{l(new)})$ $1 \leq i, j \leq 2n + 2$, not in the deterministic gray edge is called *non-deterministic* gray edge. If **(1)** $\pi_i^{l(new)}$ is even and $\pi_j^{l(new)}$ is odd, **(2)** $\pi_i^{l(new)} = \pi_j^{l(new)} + 1$ or $\pi_i^{l(new)} = \pi_j^{l(new)} - 1$, and **(3)** either $\pi_i^{l(new)}$ or $\pi_j^{l(new)} \notin \pi^{l(new)}$ or both $\pi_i^{l(new)}$ and $\pi_j^{l(new)} \notin \pi^{l(new)}$. For example, the non-deterministic gray edges in Fig. 3 are $(2, 3)$, $(4, 5)$, $(6, 7)$, $(8, 9)$, $(16, 17)$, $(18, 19)$, $(20, 21)$, and $(22, 23)$.

STEP 2: Generating all feasible solutions for all non-deterministic gray edges

Let $N = \{N_1, N_2, \dots, N_k\}$ be the set of ordered pair of non-deterministic gray edges, where k is the number of non-deterministic gray edges. $N_i = (l_{i1}, l_{i2})$ be

the ordered pair of non-deterministic gray edges in set N . For each N_i , the set of feasible gray edges (denoted by E_i) are generated by the algorithm in Fig. 4. The following notations are used in the algorithm.

- T_a : Set of *temporary* names assigned to the same duplicated genes a .
- V_a : Set of ordered pairs of *variables* transformed from all temporary names in T_a .
- B_a : Set of possible *final name* pairs assigned to all variable pairs in V_a .

Algorithm *Generate_Feasible_Solutions* ($N, \pi^{(new)}$)

Begin

1. /* Split V_a and B_a into two sets, respectively */
 2. **For** each same duplicated gene a in $\pi^{(new)}$ **Do**
 3. V_{a1} = Set of the *first* elements for all ordered pairs of V_a ;
 4. V_{a2} = Set of the *second* elements for all ordered pairs of V_a ;
 5. B_{a1} = Set of the *first* elements for all ordered pairs of B_a ;
 6. B_{a2} = Set of the *second* elements for all ordered pairs of B_a ;
 7. **If** $B_{a1} = B_{a2}$ **Then** $V_{a1} = V_{a1} \cup V_{a2}$ **End**
 8. **End**
 9. /* Generate the feasible gray edges */
 10. **For** each $N_i = (l_{i1}, l_{i2})$ **Do**
 11. **If** $l_{i1} \in \pi^{(new)}$ and $l_{i2} \notin \pi^{(new)}$ **Then**
 12. **If** $l_{i2} \in B_{a1}$ **Then** $E_i = \{l_{i1}\} \times V_{a1}$
 13. **Else** $E_i = \{l_{i1}\} \times V_{a2}$ **End**
 14. **Else If** $l_{i1} \notin \pi^{(new)}$ and $l_{i2} \in \pi^{(new)}$ **Then**
 15. **If** $l_{i1} \in B_{a1}$ **Then** $E_i = V_{a1} \times \{l_{i2}\}$
 16. **Else** $E_i = V_{a2} \times \{l_{i2}\}$ **End**
 17. **Else** /*Both l_{i1} and $l_{i2} \notin \pi^{(new)}$ */
 18. **If** $l_{i1} \in B_{a1}$ and $l_{i2} \in B_{a1}$ **Then** $E_i = V_{a1} \times V_{a1}$
 19. **Else If** $l_{i1} \in B_{a2}$ and $l_{i2} \in B_{a1}$ **Then** $E_i = V_{a2} \times V_{a1}$
 20. **Else If** $l_{i1} \in B_{a1}$ and $l_{i2} \in B_{a2}$ **Then** $E_i = V_{a1} \times V_{a2}$
 21. **Else** $E_i = V_{a2} \times V_{a2}$
 22. **End**
 23. **End**
- End**

Fig. 4. The outline for generating all feasible solutions for all non-deterministic gray edges.

From the *Generate_Feasible_Solutions* algorithm in Fig. 4, the steps for creating the set of feasible gray edges are shown in the following examples and all of the feasible sets E_i are shown in Table 2.

Table 2. The eight feasible edge groups ($E_1 - E_8$) and the 44 feasible gray edges ($e_1 - e_{44}$) of the *incomplete* breakpoint graph.

$E_1(2, 3)$	$E_2(4, 5)$	$E_3(6, 7)$	$E_4(8, 9)$
$e_1: 2-x_{11}$	$e_5: x_{11}-y_{11}$	$e_{11}: x_{21}-y_{11}$	$e_{17}: y_{12}-x_{11}$
$e_2: 2-x_{12}$	$e_6: x_{11}-y_{21}$	$e_{12}: x_{21}-y_{21}$	$e_{18}: y_{12}-x_{12}$
$e_3: 2-x_{21}$	$e_7: x_{11}-y_{31}$	$e_{13}: x_{21}-y_{31}$	$e_{19}: y_{12}-x_{21}$
$e_4: 2-x_{22}$	$e_8: x_{12}-y_{11}$	$e_{14}: x_{22}-y_{11}$	$e_{20}: y_{12}-x_{22}$
	$e_9: x_{12}-y_{21}$	$e_{15}: x_{22}-y_{21}$	$e_{21}: x_{21}-y_{11}$
	$e_{10}: x_{12}-y_{31}$	$e_{16}: x_{22}-y_{31}$	$e_{22}: y_{22}-x_{12}$
			$e_{23}: y_{22}-x_{21}$
			$e_{24}: y_{22}-x_{22}$
			$e_{25}: y_{32}-x_{11}$
			$e_{26}: y_{32}-x_{12}$
			$e_{27}: y_{32}-x_{21}$
			$e_{28}: y_{32}-x_{22}$
$E_5(16, 17)$	$E_6(18, 19)$	$E_7(20, 21)$	$E_8(22, 23)$
$e_{33}: 16-y_{11}$	$e_{36}: y_{12}-19$	$e_{39}: 20-y_{11}$	$e_{42}: y_{12}-23$
$e_{34}: 16-y_{21}$	$e_{37}: y_{22}-19$	$e_{40}: 20-y_{21}$	$e_{43}: y_{22}-23$
$e_{35}: 16-x_{31}$	$e_{38}: y_{32}-19$	$e_{41}: 20-y_{31}$	$e_{44}: y_{32}-23$

For duplicated gene 2,

$$T_2 = \{x_1, x_2\}, V_2 = \{(x_{11}, x_{12}), (x_{21}, x_{22})\}, B_2 = \{(3, 4), (4, 3), (7, 8), (8, 7)\}$$

$$V_{21} = \{x_{11}, x_{21}\}, V_{22} = \{x_{12}, x_{22}\}, B_{21} = \{3, 4, 7, 8\}, B_{22} = \{3, 4, 7, 8\}.$$

Because $B_{21} = B_{22}$, so $V_{21} = \{x_{11}, x_{12}, x_{21}, x_{22}\}$ (line 7).

For duplicated gene 3,

$$T_3 = \{y_1, y_2, y_3\}, V_3 = \{(y_{11}, y_{12}), (y_{21}, y_{22}), (y_{31}, y_{32})\},$$

$$B_3 = \{(5, 6), (17, 18), (19, 20)\}.$$

$$V_{31} = \{y_{11}, y_{21}, y_{31}\}, V_{32} = \{y_{12}, y_{22}, y_{32}\}, B_{31} = \{5, 17, 19\}, B_{32} = \{6, 18, 20\}.$$

Generating the feasible gray edges (Lines 9-23 in *Generate_Feasible_Solutions*).

Example 1: the pair (2, 3) in E_1 shown in Table 2 has four feasible gray edges ($e_1 - e_4$). This is because gene name $2 \in \pi^{(new)}$ but the name $3 \notin \pi^{(new)}$ (**case in line 11**). Because name 3 is in B_{21} , so $E_1 = \{2\} \times \{x_{11}, x_{12}, x_{21}, x_{22}\}$. Note that neither of variables x_{11} , x_{12} , x_{21} , and x_{22} can be assigned the final name 3.

Example 2: Consider the pair (4, 5). Both gene names 4 and 5 $\notin \pi^{(new)}$ (**line 17**). Gene name 4 must be assigned to only one variable in $\{x_{11}, x_{12}, x_{21}, x_{22}\}$ and gene name 5 must be assigned to only one variable in $\{y_{11}, y_{21}, y_{31}\}$. Therefore, there are 12 feasible gray edges in E_2 (shown in Table 2).

All feasible and correct gray edges, labelled $e_1 - e_{44}$, are shown in Table 2. Each gray edge is denoted by a variable e_k . This variable is used in the integer programming procedure (described in the STEP 3) as a *decision* variable. Its value is set to 1 if its corresponding gray edge is selected, otherwise it is set to 0.

STEP 3: Formulating the binary integer programming problem

For any incomplete breakpoint graph, since a black edge must be directly connected with a gray edge and vice versa, a path can be formed by traversing these alternating

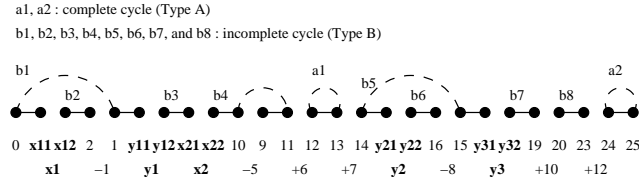


Fig. 5. Cycles a_1 and a_2 are *type A* and $b_1, b_2, b_3, b_4, b_5, b_6, b_7,$ and b_8 are *type B*.

edges. The path can be classified into the following two types.

- (a) *Type A*: The path forms a cycle¹ and is named a *complete cycle*.
- (b) *Type B*: Any path not in *type A* is named an *incomplete cycle*. A path can form a cycle by connecting the starting and ending vertices in the path with a gray edge or with other paths of *type B* only after the variable names of some $\pi_i^{(new)}$ in $\pi^{(new)}$ are assigned the final names and some gray edges are connected. The vertex that is not connected by any gray edge is named a *non-linking vertex*.

An example with both path types is shown in Fig. 5. There are two *complete cycles* (*type A* paths), and eight *incomplete cycles* (*type B* paths).

While completing the breakpoint graph, the set of gray edges needs to satisfy the following properties: (1) every non-linking vertex is connected to exactly one gray edge, and (2) only one gray edge from each edge group (E_p) is selected. Eq. (1) provides the essential clue to achieve the minimum number of reversals by minimizing $b(\pi) - c(\pi)$. The selected gray edges should create the maximum number of cycles. Note that the number of cycles in a breakpoint graph is maximized when $\pi = \phi$ (all of them are of cycle size 1). Since all paths of *type A* are complete cycles, there is no need to consider them in detail. Only the number of these complete cycles is used to calculate the reversal distance. Instead the focus is on finding the maximum number of cycles^c formed from the paths of *type B*. To find the nearest maximum number of complete cycles created from *type B* paths, the weight w_i for each edge e_i is defined according to the following heuristic rules (see Fig. 6).

case 1: We consider two sub-cases for setting the weight w_i for a feasible gray edge e_i :

case 1.1 The e_i forms the complete cycle size 1. For example, edges e_2 in Table 2.

case 1.2 The e_i forms the complete cycle size greater than 1. For example, edges e_5, e_{13}, e_{32} in Table 2.

case 2: The e_i joins two vertices in different incomplete cycles. For example, edges e_1, e_3, e_4 in Table 2.

^cThe problem of maximum cycle decomposition is NP-Hard.¹¹

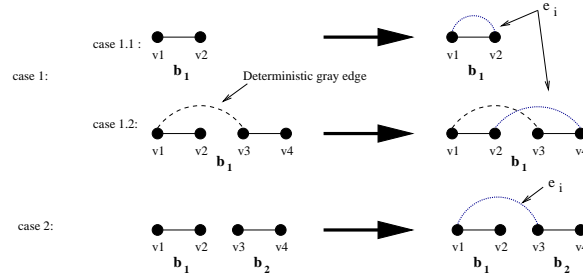


Fig. 6. An example with the three cases for connecting edge e_i . The *deterministic* gray edge is denoted by a dashed line while the *non-deterministic* gray edges (e_i) are denoted by dotted lines. v_i is a vertex in the *incomplete* breakpoint graph.

A greedy weight setting is recommended. The weight setting we use is based on the following rational observation. While selecting the gray edges, any incomplete cycles in cases 1.1 and 1.2 should be resolved first since only one gray edge is required to form a complete cycle in any case. Consequently the number of complete cycles in the incomplete breakpoint graph is increased if all cycles in cases 1.1 and 1.2 are formed. Therefore, the weights on the corresponding gray edges (cases 1.1 and 1.2) are set to a large constant to ensure that they get high priority in the optimization. For case 2, forming a cycle is more complicated. Therefore, remaining gray edges receive a lower priority by a small constant weight value.

The following notations are used for describing the application of binary integer programming.

- l_{ik} : equal to 1 if edge e_k belongs to an *incomplete* cycle b_i , otherwise 0.
- N_B : the number of *incomplete* cycles type B .
- $N_B^{(i)}$: the number of *variables* for cycle i of type B .
- N_g : the number of feasible gray edges.
- G_p : the number of feasible gray edges for group p .

$$\text{Objective function: } \text{Maximize } \sum_{i=1}^{N_g} w_i \cdot e_i \quad (2)$$

$$\text{Constraints: } \sum_{k=1}^{N_g} l_{ik} = N_{B_i} \quad ; \text{ for } i = 1 \text{ to } N_B \quad (3)$$

$$\sum_{i=1}^{G_p} e_i = 1 \quad ; \text{ for } p = 1 \text{ to } N_g \quad (4)$$

$$e_i - e_j = 0 \quad ; \text{ every } E_k \text{ and } E_{k+1} \quad (5)$$

$$; \text{ for } i=1 \text{ to } G_k \quad (5)$$

$$; \text{ for } j=1 \text{ to } G_{k+1} \quad (5)$$

$$e_i \in \{0, 1\} \quad (6)$$

The objective function (2) maximizes the weight of candidate edges (the maximal weight forms the maximum cycles). The constraint (3) ensures that the number of

selecting edges e_k does not exceed total number of variables for each cycle b_i and constraint (4) ensures that only one possible gray edge is selected for each group, and the constraint (5) ensures that two selected edges are correctly transformed from the same temporary name. The conclusion of SRDMF is shown in Fig. 7.

Algorithm SRDMF(π, ϕ)

Begin

1. Rename each family of size 1 of π according to its position in ϕ , and rename each family of size greater than 1 by temporary variables and keep the mapping position.
2. Transform signed to unsigned permutation.
3. Create the incomplete breakpoint graph.
4. Identify all feasible gray edges and set weight for each edge.
5. Formulate the binary integer programming from Eqs.(2)-(6).
6. Solve the binary integer programming and output all selected gray edges to complete the breakpoint graph.

End

Fig. 7. The conclusion of the SRDMF algorithm.

Once the algorithm is completed, all variables are assigned *final* values. Finally, Eq. (1) can be applied to compute the reversal distance (d_{rev}^{SRDMF}) between the two re-labelled genomes. The bound on the reversal distance can be derived by the observation on the number of different types of cycles in the incomplete breakpoint graph. The following Lemmas and Theorem summarize the bounds.

Lemma 1. *Given an incomplete breakpoint graph for multi-gene family case, the number of possible cycles of type B in case 2 denoted by $N_B^{(2)}$ is*

$$N_B^{(2)} \leq \lfloor \frac{b - b_A - b_1}{2} \rfloor$$

where b is the total number of black edges in the incomplete breakpoint graph, b_A is the number of black edges for cycles of type A. b_1 is the number of black edges for cycles of type B in case 1.

Proof. The number of black edges for cycles of types A and B in case 1 is obviously fixed. Then, the cycles of type B in case 2 must be formed from the remaining black edges not in case 1. Note that each cycle of type B in case 2 must use at least two black edges to complete the cycle. Therefore, the number of cycles for type B in case 2 is $\lfloor \frac{b - b_A - b_1}{2} \rfloor$. \square

Lemma 2. *Given an incomplete breakpoint graph for multi-gene family case, the total number of cycles denoted by C , has the following upper bound*

$$C \leq N_A + N_B^{(1)} + \frac{b - b_A - b_1}{2}$$

where N_A is the total number of cycles of type A. $N_B^{(1)}$ is the total number of cycles of type B case 1.

Proof. Every cycle of type A is completed by itself without involving any cycle of type B. Similarly, every cycle of type B needs no cycle of type A as a part of itself. In addition, each case of type B is independent from the other cases of type B. Therefore, counting the total number of cycles of type A and all cases of type B can be considered separately. Without loss of generality, a given *incomplete* graph has cycles of both types. Therefore, the upper bound for the number of cycles is

$$C \leq N_A + N_B^{(1)} + N_B^{(2)}$$

Substitute $N_B^{(2)}$ from Lemma 1, we have

$$C \leq N_A + N_B^{(1)} + \frac{b - b_A - b_1}{2} \quad \square$$

Theorem 1. *Given two genomes π and ϕ , the SRDMF has the reversal distance*

$$d_{rev}^{SRDMF} \geq \frac{b - 2N_A - 2N_B^{(1)} + b_A + b_1}{2}$$

when the number of hurdles is 0 and the fortress factor is also 0.

Proof. Since the value of C has the total number of the cycles as proved in Lemma 2 and the number of black edges for the given *incomplete breakpoint graph* is a constant. Note that, the number of black edges for the given incomplete breakpoint graph is equal to the number of black edges for the given complete breakpoint graph. By substituting C from Lemma 2 in Eq.(1) such that parameters h and f are 0, the reversal distance, d_{rev}^{SRDMF} , is bounded by

$$\begin{aligned} d_{rev}^{SRDMF} &\geq b - [N_A + N_B^{(1)} + N_B^{(2)}] \\ d_{rev}^{SRDMF} &\geq b - [N_A + N_B^{(1)} + \frac{b - b_A - b_1}{2}] \\ d_{rev}^{SRDMF} &\geq \frac{2b - 2N_A - 2N_B^{(1)} - b + b_A + b_1}{2} \\ d_{rev}^{SRDMF} &\geq \frac{b - 2N_A - 2N_B^{(1)} + b_A + b_1}{2} \quad \square \end{aligned}$$

4. Experimental Results

We implemented the SRDMF algorithm in MATLAB version 7.0 and tested it for performance.^d Test data fell into two categories: synthetic and real biological data.

4.1. Synthetic data

In order to rigorously test for the performance of our method, we compare the calculated reversal distance with the *known minimal reversal distance* obtained by the authoritative program GRAPPA version 2.0,⁴ module of *invdist* only, to compute the reversal distance for all possibilities and find the best one that has the minimal reversal distance. The accuracy of our method was compared to that achieved with the ALG-SRDD algorithm.^e Although ALG-SRDD was not originally proposed to compute the reversal distance, its objective is closely related to that of this problem. The synthetic data set is generated as follows:

- (1) Start from π with m distinct symbols whose signs are also randomly generated.
- (2) Randomly generate f families, where the i -th family, denoted by f_i , has random size $3 \leq K(f_i) \leq 5$, *i.e.* recursively combining single gene until the size equals to $K(f_i)$.

To synthesize the genome ϕ as the reversal target of π , we perform k random reversals on the genome π , which is generated by the above 2-step procedure. The boundaries of these random reversals are uniformly distributed within the size of genome. Therefore, the double (m, k) specifies parameters for generating a pair of input genomes. We ran the SRDMF and ALG-SRDD on 10 random instances. Fig. 8 shows the average performance of both algorithms over 10 instances in term of reversal distance, compared with the known minimal reversal distance as determined by GRAPPA (*invdist*), and the lower bound that computed from the Theorem 1.

On average, each run of both SRDMF and ALG-SRDD algorithm takes less than 5 seconds. Our heuristic algorithm consistently produces a closer estimate of the known minimal reversal distance compared to ALG-SRDD, when $k > 35$ (we believe this is because SRDD uses a simple greedy algorithm to find the maximum number of cycles). These statistics indicate that our algorithm is quite reliable in finding the nearest minimal reversal distance with multi-gene families.

4.2. Real biological data

We downloaded the X chromosome of human, (*Homo Sapien*, NCBI build 34, July 2003 UCSC hg 16) mouse (*Mus musculus*, NCBI build 32, October 2003; UCSC

^dAll tests were performed on Personal Computer with 2.4 MHz Pentium IV processor and 2 GB of RAM, and running the Linux operating system on Redhat 8.0 and Microsoft Windows XP. The optimization toolbox for MATLAB version 7.0 was used for implementing Binary Integer Programming (module *binprog*).

^eThe executable program was kindly provided by the authors.¹¹

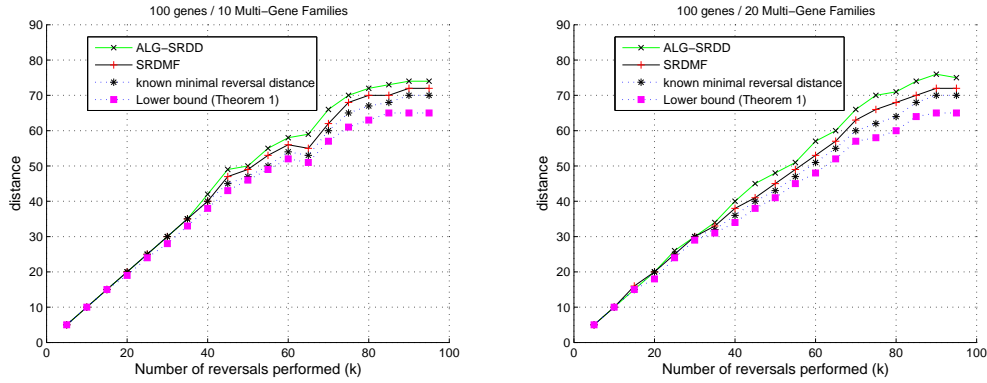


Fig. 8. The reversal distance of both ALG-SRDD and SRDMF comparing with the known minimal reversal distance. The number of reversals performed means the value of k discussed in Section 4.1.

Table 3. Results of reversal distance and breakpoint distance from SRDMF comparing with ALG-SRDD. There are 355, 321, and 348 families of size one for human-mouse, human-rat, and mouse-rat. The results in parenthesis are obtained using the SRDMF algorithm. The lower bounds are calculated from theorem 1.

X chromosome of	# number of families of size			ALG-SRDD vs (SRDMF)		lower bound	Computing time SRDMF (minute)
	2-10	11-20	≥ 20	reversal distance	breakpoint distance		
human-mouse	72	4	3	123(123)	143(143)	119	12
human-rat	82	4	2	117(117)	135(135)	108	8
rat-mouse	98	2	2	155(153)	188(184)	136	13

mm4) and rat (*Rattus norvegicus*, Baylor HGSC v.31, June 2003; UCSC rn3) from the SOAR web page (<http://www.cs.ucr.edu/xinchen/soar.html>). There are 922 genes from human X chromosome, 1,030 genes from mouse and 899 genes from rat, respectively. Then, we build the families of homologous gene by BLASTp. The BLASTp generates several high scoring segment pairs (HSPs) and E-value for each pair of genes. Two genes are considered homologous (same gene family) if (1) the E-value is less than $1e-20$ and (2) the HSP span 50% of each gene length.¹¹ Since, we do not consider gene loss or gene insertion in a genome rearrangement process (we assume that there were no gene loss or insertion events after the evolution), genes without the family in the genome will be removed. We also removed the members of gene families that are the least homologous (i.e. with the largest E-value) to the members of the counterpart family so that corresponding gene families from both genomes always have equal sizes.

By using the ALG-SRDD, the breakpoint and reversal distances between human-mouse are 143 and 123, between human-rat the distances are 135 and 117, between

mouse-rat they are 188 and 155, respectively. We ran SRDMF for all genome combinations. The comparative results for all three pairs of genomes are summarized in Table 3. The results show that ALG-SRDD and SRDMF determine identical distances for human-mouse and human-rat. We believe that the inconclusive result is due to the low estimated reversal distance between human-mouse and human-rat (34 and 35% of the number of single genes, respectively). According to the synthetic data, both ALG-SRDD and SRDMF obtain similar reversal distances when $k \leq 35$. However, the SRDMF improves slightly on ALG-SRDD for the mouse-rat comparison, both in terms of breakpoint and reversal distance. Noteworthy, the estimated reversal distance between mouse and rat is 45% of the number of single genes. According to the synthetic data, the SRDMF obtains the nearest reversal distances (based on known reversal distance) when $k \geq 35$. However, the exact minimal reversal distances for all genome pairs are not calculated, because the number of gene family is large. The processing time grows exponentially. So, the estimated lower bounds (from Theorem 1) for all pairs are shown in Table 3.

5. Conclusion

A new heuristic algorithm to efficiently find the nearest minimal reversal distance between genomes with multi-gene families is proposed. The approach uses the notion of a breakpoint graph, but readily provides means for exploring possible combinations of duplicate genes across genomes. The exploration is done using binary integer programming optimization based on pre-determined penalties for properties of an *incomplete* version of the breakpoint graph.

Experimental results on synthetic and real data sets (from human, mouse and rat X chromosome) demonstrate that our approach (SRDMF) generally outperforms the ALG-SRDD algorithm in terms of accuracy of determining the minimal reversal distance when the number of gene family is large. Our reversal distance is almost the same as the known minimal reversal distance for multi-gene family problem. The further studies will be on how to extend the proposed concept to the problem of multi-chromosomal genome, tranlocation, and transposition.

Acknowledgment

This work was supported in part by the National Center for Genetic Engineering and Biotechnology (BIOTEC) and by Commission on Higher Education, Thailand.

References

1. Hannenhalli S, Pevzner P, Transforming Cabbage Into Turnip, *Proc. 27th Ann. ACM-SIAM Symp. on Theory of Computing*, pp. 178-189, 1995.
2. El-Mabrouk N, Genome rearrangement by reversals and insertions/deletions of contiguous segments, *Proc. 11th Ann. Symp. Combinatorial Pattern Matching (CPM'00), Lecture Notes in Computer Science*, **1848**:222-234, 2000.

3. El-Mabrouk N. Reconstructing an Ancestral Genome Using Minimum Segment Duplications and Reversals, *J Comput System Sci*, **65**:442-464, 2002.
4. Bader D, Moret B, and Yan M, A linear-time algorithm for computing inversion distance between signed permutations with an experimental study, *Algorithms and Data Structures: Seventh International Workshop, WADS 2001*, pp. 365-376, 2001.
5. Bafna V, Pevzner P, Genome rearrangements and sorting by reversals, *Proc. 34th Ann. IEEE Symp. on Foundations of Computer Science*, pp.148-157, 1993.
6. Zhang J, Evolution by gene duplication : an update, *TRENDS in Ecology and Evolution*, **18(6)**:292-298, 2003.
7. Sankoff D, Genome Rearrangement with gene families, *Bioinformatics*, **15**:909-917, 1999.
8. Bryant D, The complexity of calculating exemplar distances, *Comparative Genomics*, pp.207-212, 2000.
9. Thach Nguyen C, Tay YC, Zhang L, Divide-and-Conquer approach for the exemplar breakpoint distance, *Bioinformatics*, **21(10)**:2171-2176, 2005.
10. Joel V, Young E, Lerat E, Moret B, Reversing Gene Erosion-Reconstructing Ancestral Bacteria Genomes from Gene Content and Order Data, *Proc. 4th Int'l Workshop on Algorithms in Bioinformatics WABI'04 in Lecture Notes in Computer Science*, **3240**:1-13, 2004.
11. Chen X, et.al, Assignment of orthologous genes via genome rearrangement, *IEEE/ACM Trans. on Computational Biology and Bioinformatics*, **2(4)**:302-315, 2005.
12. Caprara A, On the practical solution of reversal median problem, *Proc. Algorithm in Bioinformatics, WABI 2001*, **2149**:238-251, 2001.
13. Mantin I, Shamir R, An Algorithm for Sorting Signed Permutations by reversal, <http://www.math.tau.ac.il/~rshamir/GR>, 1999.
14. Tannier E, Sagot M, Sorting by Reversals in Subquadratic Time, *Proc. 15th Ann. Symp. Combinatorial Pattern Matching, Lecture Note in Computer Science (LNCS)*, **3109**:1-13, 2004.
15. Hannenhalli S, Pevzner P, Transforming Men into Mice (Polynomial Algorithm for Genomic Distance Problem), *36th Ann. Symp. on Foundations of Computer Science*, pp.581-592, 1995.
16. Yancopoulos S, Attie O, Friedberg F, Efficient Sorting of Genomic Permutations by Translocation, Inversion and Block Interchange, *Bioinformatics*, **21(16)**:3340-3346, 2005.
17. Tesler G, Efficient Algorithms for Multichromosomal Genome Rearrangements, *Journal of Computer and System Sciences*, **65**:587-609, 2002.