*Regular Paper*

# An Approximation Algorithm for Genome Rearrangements with Reversals and Transpositions

Hideo Matsuda,[†] Hiroyuki Yamanaka[†,☆] and Akihiro Hashimoto[†]

Recently a new approach has been proposed for inferring the evolutionary process of genomes based on comparison of gene orders. It involves searching for the minimum number of reversals and/or transpositions to sort a permutation of genes between genomes. Since the complexity of the problem is conjectured as NP-hard, we developed a 2-approximation algorithm for genome rearrangements with both reversals and transpositions. In comparison with the optimum solution in randomly generated permutations, the performance of our algorithm is 1.19 for the average approximation ratio and 1.67 for the worst case. In the application of our algorithm to comparison between complete bacterial genomes, we obtained good results in just several seconds.

## 1. Introduction

Tools for constructing phylogenetic trees have been used extensively in studies of molecular evolution[1),2)]. Their use has been aimed at comparing strings representing single genes or single proteins. For example, those studies have usually focused on a single gene, examining how the DNA sequence for the gene differs in different species. That is, such analysis uses only a specific portion of genetic information in those species.

Recently, a different approach to the evolutionary analysis among species was proposed[3)]. It focuses on comparison of not *gene sequences* but *gene orders* on whole DNA sequences (so-called *genomes*) of different species, taking into account several genome-level mutations: *inversions*, where a segment of DNA is reversed; *transpositions*, where two adjacent segments of DNA exchange places; and *translocations*, where the ends of chromosomes are exchanged. These kinds of large-scale mutations are collectively called *genome rearrangements*[4)].

In the comparison of two genomes based on genome rearrangements, the goal is to infer a sequence of mutations that change gene orders on the two genomes from their common ancestor genome. Since all the genome-level mutations mentioned above are commutative, the problem can be described as the inference of a sequence of operations that transforms the gene order of one genome into that of the other genome (via the gene order of their common ancestor). Moreover, it is known that gene-level mutations occur rarely (about $10^{-9}$ substitutions per site on DNA sequence per year) and genome-level mutations occur more slowly than gene-level mutations in many species. Thus, among the possible combinations of operations for that transformation, a sequence having the smallest number of operations is the most likely one. The problem here is to find the minimum number of operations that transform gene orders between different genomes.

The combinatorial problem of *sorting by reversals* (corresponding to genome rearrangements by inversions) has been studied intensively. Kececioglu and Sankoff suggested the first performance-guaranteed (2-approximation) algorithm for this problem[5)]. Later Bafna and Pevzner improved the approximation ratio to 7/4[6)]. It has been shown that the problem is NP-hard[7)].

Bafna and Pevzner studied a similar *sorting by transpositions* problem. They gave a (3/2)-approximation algorithm[8)]. Gu, Peng and Sudborough studied an extended problem, *sorting by reversals and transpositions*[9)]. In the problem, they introduced an extra operation, *reversal+transposition*, that in one operation both inverts a segment of DNA and also inserts it into another place on the same DNA sequence. They gave two approximation algorithms for this problem[9)]: a 2-approximation algorithm of which time complexity may not be bound

† Graduate School of Engineering Science, Osaka University
☆ Presently with NTT Data Corporation

by a polynomial of the length of the permutation, and a $2(1+1/k)$-approximation algorithm, where $k$ is any fixed integer ($k \geq 3$), which runs in polynomial time. Later they devised a polynomial-time 2-approximation algorithm[10].

In this paper we deal with a restricted version of the problem studied by Gu et al., which includes only reversal and transposition operations. A reversal+transposition operation is represented by a combination of reversal and transposition operations. We restrict the problem to only two operations under consideration of the following. Biological analyses of genome mutations have usually dealt with either inversions only or combinations of inversions and transpositions. Furthermore, the proportion of mutations which occur tends to differ by genome: for example, inversions alone are dominant in plant mitochondrial DNA[11], whereas combinations of inversions and transpositions occur in herpesvirus genomes[12]. Thus we deal with the problem without the reversal+transposition operation to make it easier to correspond with the results of those biological analyses.

We developed a 2-approximation algorithm for this problem, independently of the algorithm by Gu et al. In the next section, we give the definitions and notations of the paper and introduce the lower bound on the number of operations for sorting permutations by reversals and transpositions. Section 3 describes the approximation algorithm and Section 4 shows performance results for sorting randomly-generated permutations and sorting gene orders on bacterial genomes that have recently been completely determined.

## 2. Genome rearrangements and breakpoint graph

### 2.1 Preliminaries

Let $\Pi = (\pi_1 \pi_2 \cdots \pi_n)$ represent a signed permutation of integers 1 through $n$, where $\pi_i$ is the number at the $i$-th position. For example, if $\Pi = (2\,3\,1)$ then $\pi_1 = 2$, $\pi_2 = 3$ and $\pi_3 = 1$. Note that $\pi_{i+1}$ is the number to the right of $\pi_i$ in $\Pi$, whereas $\pi_i + 1$ is the number that is one greater than $\pi_i$.

A *reversal* $r(i,j)$ of the interval $[i, j-1]$ is an inversion of the subsequence $\pi_i \pi_{i+1} \cdots \pi_{j-1}$ of $\Pi$ ($1 \leq i < j \leq n+1$), represented by the permutation $(1 \cdots i-1 \;\; \mathbf{j-1} \cdots \mathbf{i} \;\; j \cdots n)$ and it changes the signs of the numbers in the interval. Composition of $\Pi$ with $r(i,j)$ yields $\Pi \cdot r(i,j) = (\pi_1 \cdots \pi_{i-1} \; -\pi_{j-1} \cdots -\pi_i \;\; \pi_j \cdots \pi_n)$ where the order of elements $\pi_i, ..., \pi_{j-1}$ is reversed and those signs are changed.

A *transposition* $t(i,j,k)$ inserts an interval $[i, j-1]$ of $\Pi$ between $\pi_{k-1}$ and $\pi_k$ ($1 \leq i < j \leq n+1, 1 \leq k \leq n+1$), represented by the permutation $(1 \cdots i-1 \;\; \mathbf{j} \cdots \mathbf{k-1} \;\; \mathbf{i} \cdots \mathbf{j-1} \;\; k \cdots n)$. Composition of $\Pi$ with $t(i,j,k)$ yields $\Pi \cdot t(i,j,k) = (\pi_1 \cdots \pi_{i-1} \pi_j \cdots \pi_{k-1} \pi_i \cdots \pi_{j-1} \pi_k \cdots \pi_n)$ where elements $\pi_i, ..., \pi_{j-1}$ and $\pi_j, ..., \pi_{k-1}$ exchange their order in $\Pi$ (those signs are not changed).

**Example 1** Let $\Pi = (+2 +4 +5 -3 -1)$. Then $\Pi \cdot r(2,5) = (+2 +3 -5 -4 -1)$ and $\Pi \cdot t(2,4,5) = (+2 -3 +4 +5 -1)$.

Using these operators, the genome rearrangement problem is formalized as follows.

**Definition 1 Genome Rearrangement Problem**: Given two genomes (hereafter we assume that a genome has only one chromosome), extract identical gene pairs from these genomes and let $n$ be the number of these pairs. Let the identity permutation $I = (+1 +2 \cdots +n)$ represent the matched genes on one genome and a signed permutation $\Pi = (\pi_1 \pi_2 \cdots \pi_n)$ of $\{1, 2, ..., n\}$ with $+$ or $-$ signs represent those on the other genome. Here two genes corresponding to one another on the two genomes are represented by the same number. The sign of $\pi_i$ denotes the matching of directions between the corresponding genes on the two genomes ($+$ sign, the same direction; $-$ sign, opposite direction). The problem is to find a shortest sequence of operations $\rho_1, \rho_2, ..., \rho_t$ such that $\Pi \cdot \rho_1 \cdot \rho_2 \cdots \rho_t = I$ where $\rho_i (1 \leq i \leq t)$ is either reversal or transposition.

**Figure 1** shows the relationship between gene order and signed permutation. Each gene is oriented according to the direction of two DNA sequences that are complementary to each other. In Fig. 1, the gene order on genome A is set to be a reference (i.e, the identity permutation $I$) and the gene order on genome B is encoded as a signed permutation $\Pi$. The sign of each element of $\Pi$ is set to be $+$ if its direction is the same as the corresponding gene on genome A, or $-$ otherwise.

### 2.2 Breakpoint graph

Bafna and Pevzner introduced the notion of a *breakpoint graph* for representing the structure of the problem[6]. First, they introduced breakpoint graphs of *unsigned* permutations. Then they extended them for signed permutations. We describe these notions in a similar way.

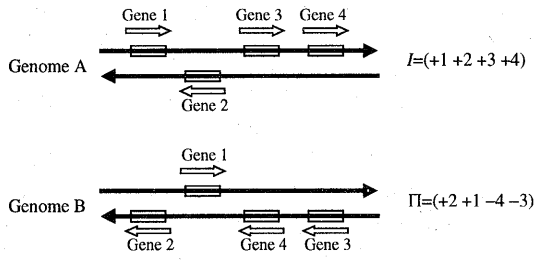


**Fig. 1** Relationship between gene order and signed permutation

(1) Breakpoint graph of unsigned permutation.
Add two extra numbers $\pi_0 = 0$ and $\pi_{n+1} = n + 1$ into an unsigned permutation $\Pi = (\pi_1 \pi_2 \cdots \pi_n)$ of $\{1, 2, ..., n\}$. Let $i \sim j$ if $|i-j| = 1$ and $i \not\sim j$ otherwise. A pair of consecutive elements $\pi_i$ and $\pi_{i+1}$ $(0 \leq i \leq n)$ of $\Pi$ is called a *breakpoint* if $\pi_i \not\sim \pi_{i+1}$. A pair of non-consecutive elements $\pi_i$ and $\pi_j$ $(i \not\sim j)$ is called an *adjacency* if $\pi_i \sim \pi_j$. A breakpoint graph $G(\Pi) = (V, E)$ of $\Pi$ is a graph such that each vertex $v \in V$ is an element of $\Pi$ and each edge $e \in E$ is a link between either a breakpoint pair or an adjacency pair. Hereafter, we refer to a link between a breakpoint pair as a *b-edge* and to a link between an adjacency pair as an *a-edge*. These edges form *alternating cycles*, namely every two consecutive edges are different types, either a-edge or b-edge. Hereafter we refer to alternating cycles simply as "cycles".
(2) Breakpoint graph of signed permutation.
To extend the notion of a breakpoint graph of an unsigned permutation to that of a signed permutation, Bafna and Pevzner introduced a transformation from a signed permutation of length $n$ to a corresponding unsigned permutation of length $2n+2$ [6]: a positive integer $+i$ is replaced by two unsigned integers, $2i-1$ and $2i$; a negative integer $-i$ is replaced by $2i$ and $2i-1$; and two more integers, 0 and $2n+1$, are added at each end respectively. After this transformation, a breakpoint graph of a signed permutation is constructed similarly to that of an unsigned permutation.

**Figure 2** shows a breakpoint graph of a signed permutation $\Pi_{signed} = (+2 +1 -4 -3)$. In Fig. 2, its transformed unsigned permutation $\Pi_{trans} = (3\,4\,1\,2\,8\,7\,6\,5)$ and two extra integers 0 and 9 are added at the ends. Hereafter we refer to an unsigned permutation transformed from a signed permutation simply as a "permutation".

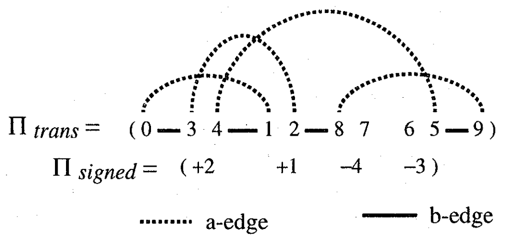


**Fig. 2** A breakpoint graph of a signed permutation $\Pi = (+2 +1 -4 -3)$.

## 2.3 Lower Bound

In a breakpoint graph $G(\Pi)$ of an arbitrary permutation $\Pi$ of $\{0, 1, ..., 2n, 2n+1\}$ (transformed from a signed permutation of $\{1, 2, ..., n\}$), let $b(\Pi)$ and $d(\Pi)$ be the number of b-edges (i.e., breakpoints) and the distance between $\Pi$ and $I$ (i.e., the minimum number of operations that transform $\Pi$ into $I$), respectively. Denote $\Delta b = \Delta b(\Pi, \rho) = b(\Pi) - b(\Pi\rho)$ (the decrease in the number of breakpoints by an operation $\rho$, either a reversal or a transposition).

Gu et al.[10] proves that the lower bound of sorting by reversals, transpositions and reversal+transpositions is:

$$d(\Pi) \geq (b(\Pi) - c_{odd}(\Pi))/2, \tag{1}$$

where $c_{odd}(\Pi)$ denotes the number of odd cycles, which are the cycles whose number of b-edges is odd.

Since the set of operations in our problem described in Definition 1 is a subset of the problem studied by Gu et al., the lower bound of our problem is at least the value given in Eq. (1).

## 3. 2-Approximation Algorithm

As mentioned in Section 1, Gu et al. devised a 2-approximation algorithm for sorting by reversals, transpositions and reversal+transpositions[10]. Independently of their work, we have devised a 2-approximation algorithm for a restricted version of their formulation (i.e., sorting by only reversals and transpositions).

Eq. (1) implies that decreasing $b(\Pi)$ without decreasing $c_{odd}(\Pi)$ reduces the distance between $\Pi$ and $I$. Thus we designed this algorithm as a greedy algorithm to explore operations that removes as many breakpoints as possible without decreasing the number of cycles.

We classified operations by the number of breakpoints to be removed: *i-transposition* denotes a transposition that reduces $i$ breakpoints $(-3 \leq i \leq 3)$ and *j-reversal* denotes a reversal that reduces $j$ breakpoints $(-2 \leq j \leq 2)$, respectively. Note that $\Delta b = i$ for an $i$-transposition whereas $\Delta b = j$ for a $j$-reversal.

To remove breakpoints without decreasing the number of cycles, we introduce the notion of *one-cycle* operations. Suppose a cycle $C = (V_C, E_C)$ in a breakpoint graph $G(\Pi)$. A transposition $t(i,j,k)$ is called a *one-cycle transposition* if $(\pi_{i-1}, \pi_i) \in E_C$, $(\pi_{j-1}, \pi_j) \in E_C$ and $(\pi_{k-1}, \pi_k) \in E_C$. Similarly, a reversal $r(i,j)$ is called a *one-cycle reversal* if $(\pi_{i-1}, \pi_i) \in E_C$ and $(\pi_{j-1}, \pi_j) \in E_C$.

Gu et al. proved the property that one-cycle operations can be performed in a cycle if the cycle has at least one pair of crossing a-edges[10]. Here a pair of crossing a-edges consists of two a-edges $(\pi_i, \pi_j)$ $(i < j)$ and $(\pi_k, \pi_l)$ $(k < l)$ such that the endpoints of the two edges are interleaved in $\Pi$ (i.e., $i < k < j < l$ or $k < i < l < j$).

Gu et al. used the reversal+transposition operation in their proof. Since the problem we introduce here lacks this operation, we prove the property mentioned above without using it as described below.

Hereafter, we denote $\Delta c = \Delta c(\Pi, \rho) = c(\Pi) - c(\Pi\rho)$, which represents the decrease in the number of cycles by an operation $\rho$ that is either a reversal or a transposition.

**Lemma 1** One-cycle operations can be performed in a cycle with at least one pair of crossing a-edges.

*Proof.* Let $(\pi_{i-1}, \pi_i)$ and $(\pi_{j-1}, \pi_j)$ be two consecutive b-edges that are connected by an a-edge. There are four types depending on the position of the a-edge (see **Fig. 3**).

(a) $(\pi_{i-1}, \pi_{j-1})$ is an a-edge.
A reversal $r(i,j)$ is a one-cycle reversal that removes a b-edge incident with $\pi_{i-1}$. Moreover, if another a-edge $(\pi_i, \pi_j)$ exists, $r(i,j)$ is a one-cycle 2-reversal that deletes the cycle. Otherwise, $r(i,j)$ is a one-cycle 1-reversal.

(b) $(\pi_{i-1}, \pi_j)$ is an a-edge.
If the a-edge is crossed with another a-edge in the same cycle, there exists another b-edge $(\pi_{k-1}, \pi_k)$ outside interval $[i-1, j]$ (i.e., $k < i$ or $j < k-1$) in the cycle. Either $t(k,i,j)$ (if $k < i$) or $t(j,k,i)$ (if $j < k-1$) is a one-cycle transposition that removes a b-edge by making $\pi_{i-1}$ and $\pi_j$ be two consecutive numbers in $\Pi\rho$. Figure 3 (b) shows the case $j < k-1$. Moreover, if both $(\pi_i, \pi_{k-1})$ and $(\pi_{j-1}, \pi_k)$ are a-edges, $t(k,i,j)$ (or $t(j,k,i)$) is a 3-transposition that deletes the cycle. If either of the two edges is an a-edge, $t(k,i,j)$ (or $t(j,k,i)$) is a one-cycle 2-transposition. Otherwise, $t(k,i,j)$ (or $t(j,k,i)$) is a one-cycle 1-transposition.

(c) $(\pi_i, \pi_{j-1})$ is an a-edge.
If the a-edge is crossed with another a-edge in the same cycle, there exists another b-edge $(\pi_{k-1}, \pi_k)$ $(i+1 < k < j-1)$ in the cycle, and a transposition $t(k,j,i)$ is a one-cycle transposition that removes a b-edge incident with $\pi_i$. Moreover, if both $(\pi_{i-1}, \pi_k)$ and $(\pi_{k-1}, \pi_j)$ are a-edges, $t(k,j,i)$ is a 3-transposition that deletes the cycle. If either of the two edges is an a-edge, $t(k,j,i)$ is a one-cycle 2-transposition. Otherwise, $t(k,j,i)$ is a one-cycle 1-transposition.

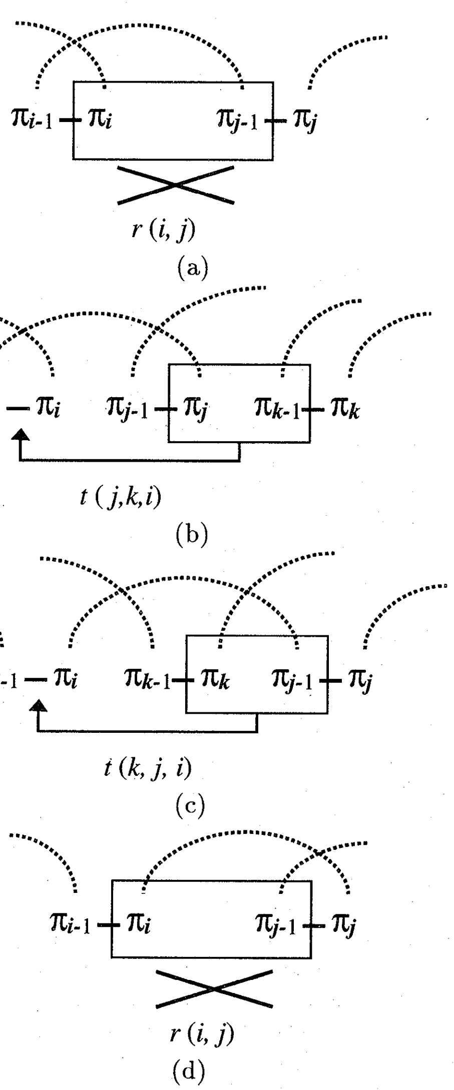


**Fig. 3** Relationship between breakpoint types and one-cycle operations

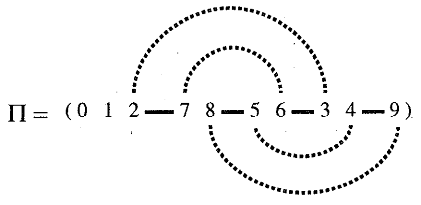


**Fig. 4** Examples of non-oriented cycles

(d) $(\pi_i, \pi_j)$ is an a-edge.
A reversal $r(i,j)$ is a one-cycle reversal that removes a b-edge incident with $\pi_j$. Moreover, if another a-edge $(\pi_{i-1}, \pi_{j-1})$ exists, $r(i,j)$ is a one-cycle 2-reversal as mentioned in (a). Otherwise, $r(i,j)$ is a one-cycle 1-reversal. □

The following lemma describes a property of one-cycle operations.

**Lemma 2** $\Delta c \leq 0$ for one-cycle 2-transposition, one-cycle 1-transposition and one-cycle 1-reversal operations.

*Proof.* Let $\rho$ be any of these operations. Let $C_1$ and $C_i (2 \leq i \leq c(\Pi))$ be the cycle where $\rho$ is performed and the other cycles in $G(\Pi)$, respectively. Since $C_1 \not\subseteq G(\Pi\rho)$ and $C_i \subseteq G(\Pi\rho)$, $c(\Pi\rho) \geq c(\Pi) - 1$. Since any of the operations yields at least one new b-edge, $\rho$ yields at least one new cycle $C' \not\subseteq G(\Pi)$. This implies $c(\Pi\rho) \geq c(\Pi)$. Thus $\Delta c = c(\Pi) - c(\Pi\rho) \leq 0$. □

If there exists a cycle with no crossing a-edges, one-cycle operations cannot be performed in the cycle. Bafna and Pevzner call such a cycle *non-oriented cycle*[8]. A non-oriented cycle has one a-edge of type (b) (between the leftmost end and the rightmost end in the permutation) and at least one a-edge of type (c). **Figure 4** shows a breakpoint graph of $\Pi = (0\,1\,2\,7\,8\,5\,6\,3\,4\,9)$ has two non-oriented cycles.

Gu et al. proved that a non-oriented cycle is always interleaved with another cycle[10]. Using this property, they provided a set of two transpositions that removes two b-edges from two interleaved non-oriented cycles (one b-edge per cycle) and does not change the length of any other cycles.

**Figure 5** shows an example of such a set of two transpositions. The first transposition removes one b-edge and merges the two non-oriented cycles into a cycle that has crossing a-edges. Then the second transposition also removes one b-edge and split the cycle into two new non-oriented cycles.

In this way, two b-edges are removed, while the number of cycles remains the same before

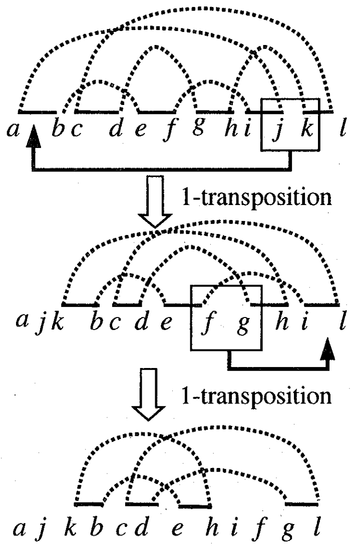


**Fig. 5** A set of two transpositions in two interleaved non-oriented cycles.

and after performing the two transpositions unless either of the two cycles is deleted (i.e., $\Delta b - \Delta c = 2$ by two operations). When either of the two cycles is removed, the number of the cycles decrease by one but the two operations remove an extra b-edge since they remove a cycle (i.e., $\Delta b - \Delta c = 3 - 1 = 2$ by two operations). Similarly when both of the two cycles are removed by the two operations, the number of the cycles decrease by two but the two operations remove two extra b-edges since they remove two cycles (i.e., $\Delta b - \Delta c = 4 - 2 = 2$ by two operations). By repeating these transpositions, all b-edges in non-oriented cycles can eventually be removed and $\Delta b - \Delta c = 1$ holds per operation on average.

The outline of our algorithm is as shown in **Fig. 6**. In this algorithm, we give higher priority to the operations that remove a greater number of b-edges, while Gu et al. dealt with every one-cycle operation on an equal basis[10]. This ordering of operations may contribute to a decrease in the expected approximation ratio of our algorithm.

The number of while-loop iterations is proportional to the number of b-edges and the condition-checking process in each if-statement requires time proportional to the number of b-edges. Since the number of b-edges is proportional to the permutation length in the worst case, the time complexity of this algorithm is bound by $O(n^2)$, where $n$ is the length of permutation.

```
Algorithm 1
begin
    Construct $G(\Pi)$;
    while there exist b-edges in $G(\Pi)$
    begin
        if one-cycle 3-transposition is executable
            do it;
        elseif one-cycle 2-transposition is executable
            do it;
        elseif one-cycle 2-reversal is executable
            do it;
        elseif one-cycle 1-transposition is executable
            do it;
        elseif one-cycle 1-reversal is executable
            do it;
        elseif two interleaved non-oriented cycles exist
            perform two transpositions in the cycles;
    end
end
```

**Fig. 6** The outline of approximation algorithm for genome rearrangement

The following theorem holds regarding algorithm performance.

**Theorem 1** For each while-loop iteration of the algorithm shown in Fig. 6, $\Delta b - \Delta c \geq 1$ per operation.

*Proof.* The value of $\Delta b - \Delta c$ for each operation in this algorithm is: $\Delta b - \Delta c = 2$ for a one-cycle 3-transposition, $\Delta b - \Delta c \geq 2$ for a one-cycle 2-transposition by Lemma 2, $\Delta b - \Delta c = 1$ for a one-cycle 2-reversal, $\Delta b - \Delta c \geq 1$ for a one-cycle 1-transposition and one-cycle 1-reversal by Lemma 2.

As mentioned above, a set of two operations in two interleaved non-oriented cycles removes two breakpoints and does not change the number of cycles. Thus $\Delta b - \Delta c \geq 1$ holds per operation on average. □

Thus we can derive the following theorem.

**Theorem 2** In the algorithm shown in Fig. 6, $d(\Pi) \leq b(\Pi) - c(\Pi)$.

*Proof.* Let $t = d(\Pi)$. Denote $\rho_t, \rho_{t-1}, \ldots, \rho_1$ as operations that give the minimum number of operations. Then the transformation from $\Pi$ into $I$ is represented by a recurrence relation $\Pi_{(i-1)} = \Pi_{(i)}\rho_i$ $(1 \leq i \leq t)$ where $\Pi_{(t)} = \Pi$ and $\Pi_{(0)} = I$.

By Theorem 1,

$$\begin{aligned} d(\Pi_{(i)}) &= d(\Pi_{(i-1)}) + 1 \\ &\leq d(\Pi_{(i-1)}) + \Delta b(\Pi_{(i)}, \rho_i) \\ &\quad -\Delta c(\Pi_{(i)}, \rho_i)) \\ &= d(\Pi_{(i-1)}) + b(\Pi_{(i)}) - b(\Pi_{(i-1)}) \\ &\quad -(c(\Pi_{(i)}) - c(\Pi_{(i-1)})) \end{aligned} \tag{2}$$

Eq. (2) can be transformed into the following equation.

$$\begin{aligned} &d(\Pi_{(i)}) - (b(\Pi_{(i)}) - c(\Pi_{(i)})) \\ &\leq d(\Pi_{(i-1)}) - (b(\Pi_{(i-1)}) - c(\Pi_{(i-1)})) \\ &\leq \cdots \leq d(\Pi_{(0)}) - (b(\Pi_{(0)}) - c(\Pi_{(0)})) \end{aligned} \tag{3}$$

Since the breakpoint graph of the identity permutation $I$ has no b-edges and no cycles, $d(\Pi_{(0)}) = b(\Pi_{(0)}) = c(\Pi_{(0)}) = 0$. Thus, by replacing $i$ with $t$ in Eq. (3), the upper bound is derived from Eq. (3) as follows.

$$d(\Pi) \leq b(\Pi) - c(\Pi). \tag{4}$$

□

By Eq. (1) and Theorem 2, the approximation ratio is $2(b(\Pi) - c(\Pi))/(b(\Pi) - c_{odd}(\Pi))$. Since $b(\Pi) - c_{odd}(\Pi) \geq b(\Pi) - c(\Pi)$, the approximation ratio is at most 2. Thus we conclude that the approximation ratio of our algorithm is 2.

## 4. Performance Results

To evaluate the performance of our algorithm, we compared results with the optimum solution for random permutations. We generated 100 permutations of length 8. The optimum solution (i.e., the minimum number of op-

**Table 1** Bacterial complete genomes used for the analysis of genome rearrangements

| ID | Species | GenBank Accession No. | Number of genes |
|---|---|---|---|
| G1 | *Mycoplasma genitalium* | L43967 | 468 |
| G2 | *Mycoplasma pneumoniae* | U00089 | 677 |
| G3 | *Methanococcus jannaschii* | L77117 | 1735 |
| G4 | *Methanobacterium thermoautotrophicum* | AE000666 | 1871 |
| G5 | *Haemophilus influenzae* | L42023 | 1680 |
| G6 | *Escherichia coli* | U00096 | 4290 |

**Table 2** Sorting results between bacterial genomes

| Genomes | Number of orthologous genes | $b(\Pi)$ | $c(\Pi)$ | $c_{odd}(\Pi)$ | Lower bound | Number of operations | Execution time (s) |
|---|---|---|---|---|---|---|---|
| G1 vs G2 | 425 | 6 | 2 | 0 | 3 | 3 | 0.02 |
| G3 vs G4 | 289 | 117 | 2 | 1 | 58 | 63 | 0.23 |
| G5 vs G6 | 857 | 349 | 2 | 1 | 174 | 181 | 4.17 |

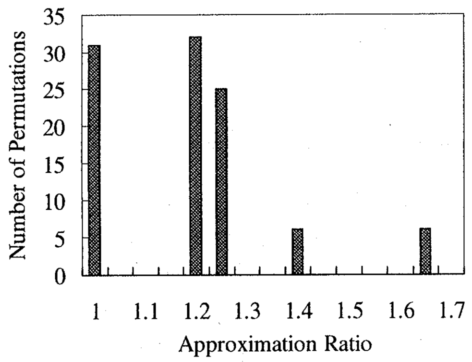


**Fig. 7** Distribution of the ratio of results to the optimum number

erations that transform these permutations into the identity permutation) was computed by using an exhaustive search method. Although we used a machine with a relatively large amount of memory (Sun SPARCstation 20 with 512 MB memory) for this computation, we could not obtain the optimum solution for permutations of length more than 8 due to exponential growth of the number of possible operations to be explored.

**Figure 7** shows the distribution of approximation ratios (the ratio of the number of operations computed by our algorithm to the optimum number). Although the theoretical approximation ratio is 2 as described in Section 3, the worst-case ratio in this experiment is 1.67 and the average of the ratios is 1.19 for our algorithm.

Gu et al. proposed a heuristic method to improve approximation ratio results for their algorithm[13]. In an experiment in randomly-generated permutations, their method achieved nearly optimum performance (results differed from the lower bound in Eq. (1) only by a small constant). To integrate such heuristics with our algorithm remains as a topic of our future research.

For the comparison between real genomes, we used the genomes shown in **Table 1**. The gene sequences and their orders on these genomes are obtained from the GenBank database[14]. Table 1 shows their accession numbers (a database identifier) in GenBank. The genomes G1 and G2 belong to the same genus (the layer that is one level higher than species), both G3 and G4 are methanogenic archaebacteria, and G5 and G6 are referred to closely related bacteria although G5 is a parasite bacterium but G6 is a free-living one[15]. Thus it may be considered that G1 and G2 are closest to each other, while G5 and G6 are furthest from each other.

To extract identical pairs (more exactly, orthologous pairs) of genes, we used a method based on bi-directional best hits on sequence similarity[15]. In this method, first compute the sequence similarity of every possible one-to-one match between a set of genes in one genome and another set in the other genome, and then extract those pairs of genes that exhibit the highest similarity to each other (a pair of genes, $i$ from genome $A$ and $j$ from genome $B$, such that $i$ exhibits the highest similarity to $j$ than any other genes in $A$ and $j$ exhibits the highest similarity to $i$ than any other genes in $B$). For computation of sequence similarity, we performed pairwise global alignment[16].

In the comparison of G1 with G2 shown in **Table 2**, we confirmed that the orthologous pairs of genes are consistent with the result of an intensive comparison between these two genomes[17]. Since it is time-consuming process to extract such orthologous pairs from possible combinations of gene pairs of two genomes based on pairwise global alignment, the speedup of this process remains as a topic of our future research.

Table 2 shows the results. Execution times were measured on a Sun SPARCstation 20 (SuperSPARC-II, clock 75 MHz). The number of operations measured by our algorithm is close to the lower bound, $(b(\Pi) - c_{odd}(\Pi))/2$. Moreover the increasing order of the number of operations corresponds biological knowledge mentioned above. The computational speed is sufficiently high even when there are hundreds of breakpoints to be removed.

## 5. Conclusions

We have developed a 2-approximation algorithm for genome rearrangements with reversals and transpositions. From comparison of our algorithm results with the optimum solution for random permutations, we note that our algorithm achieves good performance with an average approximation ratio of 1.19. We also applied our algorithm to some bacterial genomes whose complete DNA sequences have recently been determined. The performance of the algorithm was close to the lower bound. Furthermore, the computational speed is high enough to perform the comparison even if the complete set of genes in organisms is used.

Our future research topics include the development of algorithms that improve practical performance (such as expected approximation ratio) in solving this problem.

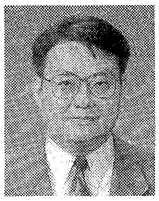
**Acknowledgments** We are grateful to the reviewers for their valuable comments and assistance in improving this manuscript. This work was supported in part by a Grant-in-Aid (08283103) for Scientific Research on Priority Areas from The Ministry of Education, Science, Sports and Culture in Japan.

## References

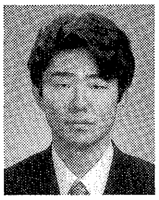
1) Nei, M.: *Molecular Evolutionary Genetics*, Columbia University Press, New York, chapter 11 (1987).
2) Swofford, D. L. and Olsen, G. J.: Phylogeny Reconstruction, In *Molecular Systematics*, ed. Hillis, D.M. and Moritz, C., Sinauer Associates, Sunderland, Mass., pp. 411–501 (1990).
3) Sankoff, D. et al.: Gene Order Comparisons for Phylogenetic Inference: Evolution of the mitochondrial genome, *Proc. Natl. Acad. Sci. USA*, Vol. 89, pp. 6575–6579 (1992).
4) Gusfield, D.: *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, chapter 19, pp. 492–500 (1997).
5) Kececioglu, J. and Sankoff, D.: Exact and Approximation Algorithms for the Inversion Distance between Two Permutations, *Proc. 4th Ann. Symp. Combinatorial Pattern Matching*, LNCS No. 684, Springer Verlag, pp. 87–105 (1993).
6) Bafna, V. and Pevzner P.: Genome Rearrangements and Sorting by Reversals, *SIAM J. Computing*, Vol. 25, No. 2, pp. 272–289 (1996).
7) Caprara, A.: Sorting by Reversals is Difficult, *Proc. 1st Ann. Conf. Research in Computational Molecular Biology (RECOMB97)*, ACM Press, pp. 75–83 (1997).
8) Bafna, V. and Pevzner P.: Sorting Permutations by Transpositions, *Proc. 6th ACM-SIAM Ann. Symp. on Discrete Algorithms*, pp. 614–623 (1995).
9) Gu, Q.-P., Peng, S. and Sudborough, H.: Approximation Algorithms for Genome Rearrangements, *Proc. Genome Informatics 1996*, Universal Academy Press, pp. 13–22 (1996).
10) Gu, Q.-P., Peng, S. and Sudborough, H.: A 2-Approximation Algorithm for Genome Rearrangements by Reversals and Transposition, *Theoretical Computer Science*, to appear.
11) Palmer, J. and Herbon L.: Plant Mitochondrial DNA Evolves Rapidly in Structure, But Slowly in Sequence, *J. Mol. Evol.*, Vol. 28, Nos. 1/2, pp. 87–97 (1988).
12) Hannenhalli, S., Chappey, C., Koonin, E. V. and Pevzner, P. A.: Genome Sequence Comparison and Scenarios for Gene Rearrangements: A Test Case, *Genomics*, Vol.30, No.2, pp.299–311 (1995).
13) Gu, Q.-P., Iwata, K., Peng, S., Chen, Q.-M.: A Heuristic Algorithm for Genome Rearrangements, *Genome Informatics 1997*, Universal Academy Press, pp. 268–269 (1997).
14) Benson, D.A., et al.: GenBank, *Nucleic Acids Research*, Vol. 26, No. 1, pp. 1–7 (1998).
15) Tatusov, R. L. et al.: Metabolism and Evolution of *Haemophilus influenzae* Deduced from a Whole-Genome Comparison with *Escherichia coli*, *Current Biology*, Vol. 6, No. 3, pp. 279–291 (1996).
16) Needleman, S. B. and Wunsch, C. D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins, *J. Mol. Biol.*, Vol. 48, pp. 444–453 (1970).
17) Himmelreich, R., et al.: Comparative Analysis of Genomes of the Bacteria *Mycoplasma pneumoniae* and *Mycoplasma genitalium*, *Nu-*

*cleic Acids Research*, Vol. 25, No. 4, pp. 701–712 (1997).

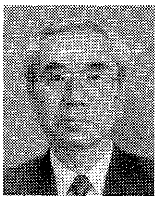
(Received May 1, 1998)
(Revised June 11, 1998)
(Accepted June 29, 1998)



**Hideo Matsuda** was born in 1959. He received his B.Sc., M.E. and Ph.D. degrees from Kobe University in 1982, 1984 and 1987 respectively. From 1984 to 1990, he was a research associate and, from 1990 to 1994 a lecturer of the Department of Systems Engineering, Kobe University. He was a visiting scholar of the Mathematics and Computer Science Division, Argonne National Laboratory from 1991 to 1992. He is now an associate professor of the Department of Informatics and Mathematical Science, Osaka University. His research interests include bioinformatics and molecular biology database. He is a member of IPSJ, IEICE, IEEE CS and ACM.



**Hiroyuki Yamanaka** was born in 1973. He received his B.E and M.E. degrees from Osaka University in 1996 and 1998, respectively. Since 1998, he is with NTT Data Corporation.



**Akihiro Hashimoto** was born in 1938. He received his B.E, M.E. and Dr.Eng. degrees from Osaka University in 1961, 1963 and 1966, respectively. He worked in NTT Laboratories from 1966 to 1989 and was engaged in research on fault diagnosis and design automation in computer systems and development of the DIPS system. He was a visiting assistant professor of the University of Illinois from 1969 to 1971. He is now a professor of the Department of Informatics and Mathematical Science, Osaka University. His research interests include information processing technology in molecular biology. He is a member of IPSJ, IEICE, IEEE and ACM.