Information Processing Letters 110 (2010) 345–350

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An improved algorithm for sorting by block-interchanges based on permutation groups

Yen-Lin Huang [a], Cheng-Chen Huang [b,c], Chuan Yi Tang [a], Chin Lung Lu [b,c,*]

[a] Department of Computer Science, National Tsing Hua University, Hsinchu 300, Taiwan
[b] Institute of Bioinformatics and System Biology, National Chiao Tung University, Hsinchu 300, Taiwan
[c] Department of Biological Science and Technology, National Chiao Tung University, Hsinchu 300, Taiwan

ARTICLE INFO

Article history:
Received 13 July 2009
Received in revised form 28 February 2010
Accepted 3 March 2010
Available online 4 March 2010
Communicated by F.Y.L. Chin

Keywords:
Algorithm
Data structure
Genome rearrangement
Permutation group
Block-interchange
Generalized transposition
Permutation tree

ABSTRACT

Given a chromosome represented by a permutation of genes, a block-interchange is proposed as a generalized transposition that affects the chromosome by swapping two non-intersecting segments of genes. The problem of sorting by block-interchanges is to find a minimum series of block-interchanges for sorting one chromosome into another. In this paper, we present an $\mathcal{O}(n+\delta\log\delta)$ time algorithm for solving the problem of sorting by block-interchanges, which improves a previous algorithm of $\mathcal{O}(\delta n)$ time proposed by Lin et al. (2005) [14], where n is the number of genes and δ is the minimum number of block-interchanges required to sort a chromosome.

1. Introduction

Genome rearrangement studies based on genome-wide analysis of gene orders play an important role in the phylogenetic tree reconstruction [20,11,12,19,3]. In these studies, the homologous genes of two genomes are usually identified with the integers $1, 2, \ldots, n$, with a plus or minus sign to indicate their direction. Then a chromosome can be represented by a signed permutation of $\{1, 2, \ldots, n\}$. To evaluate the evolutionary distance between two related genomes in gene order, various rearrangement events acting on genes within or among chromosomes have been proposed, such as reversals [12,13,1,21], transpositions [2,7,8], block-interchanges [6,14,8], translocations [11,10,4,18], fusions [11,17,15] and fissions [11,17,15]. *Reversals*, also called *inversions*, affect a block of consecutive integers on a chromosome by reversing the order of the integers and flipping their signs; *transpositions* affect two adjacent blocks of consecutive integers on a chromosome by exchanging their positions; *block-interchanges* are *generalized transpositions* by allowing the exchanged blocks not being adjacent on a chromosome. In genomes with multiple chromosomes, *translocations* exchange the end segments between two chromosomes; *fusions* join two chromosomes into a bigger one; *fissions* break a chromosome into two smaller ones.

Recently, the study on the genome rearrangement using block-interchanges has increasingly drawn great attention, since the block-interchange event is a generalization of transposition and, currently, its computational models measuring the genetic distance are more tractable than those modeled by transposition. Christie [6] first introduced the concept of block-interchange and also proposed an $\mathcal{O}(n^2)$ time algorithm using the breakpoint graph approach, where n is the number of genes, to solve the so-

* Corresponding author at: Institute of Bioinformatics and System Biology, National Chiao Tung University, Hsinchu 300, Taiwan. Tel.: +886 3 5712121x56949; fax: +886 3 5729288.
E-mail addresses: slippers.bi92g@nctu.edu.tw (Y.-L. Huang), sosorovo@yahoo.com.tw (C.-C. Huang), cytang@cs.nthu.edu.tw (C.Y. Tang), cllu@mail.nctu.edu.tw (C.L. Lu).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2010.03.003

called *problem of sorting by block-interchanges* that is to find a minimum series of block-interchanges for sorting one chromosome into another. However, the chromosomes he considered were restricted to linear chromosomes. Later, by making use of permutation groups in algebra, Lin et al. [14] designed a simpler algorithm for solving the same problem on both linear and circular chromosomes with time complexity of $\mathcal{O}(\delta n)$, where δ is the minimum number of block-interchanges required to sort a chromosome and can be calculated in $\mathcal{O}(n)$ time in advance. Recently, Feng and Zhu [8] proposed a new data structure called permutation tree to improve the Christie's algorithm by reducing the time complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. In this paper, we derive a new property that helps us to improve Lin et al.'s algorithm by using the data structure of permutation tree such that the overall time complexity can be further reduced down to $\mathcal{O}(n+\delta \log \delta)$. Notice that Bóna and Flynn [5] have shown recently that δ is close to $\frac{n}{2}$ on average.

The rest of this paper is organized as follows. Some basic concept and properties of the permutation groups in algebra are introduced in Section 2. In Section 3, we introduce the idea of algorithm proposed by Lin et al. [14] and present an improved algorithm. Finally, we make a conclusion in Section 4.

2. Preliminaries

In group theory, a *permutation* is defined to be a one-to-one mapping from a set E into itself. For example, we may define a permutation α of the set $\{1, 2, 3, 4, 5\}$ by specifying $\alpha(1) = 4$, $\alpha(2) = 1$, $\alpha(3) = 2$, $\alpha(4) = 5$ and $\alpha(5) = 3$. In the study of genome rearrangement, it is convenient to express the permutation α in *cycle* form as $\alpha = (1, 4, 5, 3, 2)$, in which for each $x \in E$, $\alpha(x)$ is placed directly right to x. A cycle of length k, say $(a_1, a_2, \ldots, a_k)$, is simply called *k-cycle* and it can also be rewritten as $(a_i, a_{i+1}, \ldots, a_k, a_1, \ldots, a_{i-1})$ (i.e., indices are cyclic), where $2 \leqslant i \leqslant k$. Any two cycles are said to be *disjoint* if they have no element in common. In fact, any permutation, say α, can be written in a unique way as the product of disjoint cycles, which is called the *cycle decomposition* of α, if we ignore the order of the cycles in the product. Usually, a cycle of length one in α is not explicitly written and its element, say x, is said to be *fixed* by α since $\alpha(x) = x$. Especially, the permutation whose elements are all fixed is called an *identity permutation* and is denoted by $\mathbf{1}$, i.e., $\mathbf{1} = (1)(2)\cdots(n)$ if $E = \{1, 2, \ldots, n\}$.

Given two permutations α and β of E, the *composition* (or *product*) of α and β, denoted by $\alpha\beta$, is defined to be a permutation of E with $\alpha\beta(x) = \alpha(\beta(x))$ for all $x \in E$. For instance, if we let $E = \{1, 2, 3, 4, 5, 6\}$, $\alpha = (2, 3)$ and $\beta = (2, 1, 5, 3, 6, 4)$, then $\alpha\beta = (2, 1, 5)(3, 6, 4)$. If α and β are disjoint cycles, then $\alpha\beta = \beta\alpha$. The *inverse* of α is defined to be a permutation, denoted by α^{-1}, such that $\alpha\alpha^{-1} = \alpha^{-1}\alpha = \mathbf{1}$. If a permutation is expressed by the product of disjoint cycles, then its inverse can be obtained by just reversing the order of the elements in each cycle. For example, if $\alpha = (2, 1, 5)(3, 6, 4)$, then $\alpha^{-1} = (5, 1, 2)(4, 6, 3)$. Clearly, $\alpha^{-1} = \alpha$ if α is a 2-cycle. Furthermore, it is well known that every permutation can be written as a product of 2-cycles. For example, $(1, 2, 3, 4) = (1, 4)(1, 3)(1, 2)$. However, there are many ways of expressing a permutation as a product of 2-cycles [9]. Given a permutation α, let $f(\alpha)$ denote the number of the disjoint cycles in the cycle decomposition of α. Notice that $f(\alpha)$ counts also the non-expressed cycles of length one. For example, if $\alpha = (1, 5)(2, 4)$ is a permutation of $E = \{1, 2, \ldots, 5\}$, then $f(\alpha) = 3$, instead of $f(\alpha) = 2$, since $\alpha = (1, 5)(2, 4)(3)$.

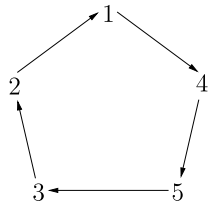

Fig. 1. The illustration of a permutation $\alpha = (1, 4, 5, 3, 2)$, where $\alpha(1) = 4$, $\alpha(2) = 1$, $\alpha(3) = 2$, $\alpha(4) = 5$ and $\alpha(5) = 3$.

3. Sorting a permutation by block-interchanges

Recall that the problem of sorting by block-interchanges is to find a minimum series of block-interchanges for sorting a source chromosome into a target chromosome. By convention, the target chromosome is usually denoted by $I = (1, 2, \ldots, n)$. Meidanis and Dias [16,17] first noted that each cycle of a permutation may represent a circular chromosome of a genome with each element of the cycle corresponding to a gene and the order of the cycle corresponding to the gene order of the chromosome. Fig. 1, for example, shows a circular chromosome, which is represented by $(1, 4, 5, 3, 2)$. Moreover, they observed that rearrangements, such as fusions and fissions (respectively, transpositions), correspond to the composition of a 2-cycle (respectively, 3-cycle) and the permutation representing a chromosome. For instance, let α be any permutation whose cycle decomposition is $\alpha_1\alpha_2\cdots\alpha_r$. If $\rho = (x, y)$ is a 2-cycle and x and y are in the different cycles of α, say $\alpha_p = (a_1 \equiv x, a_2, \ldots, a_i)$ and $\alpha_q = (b_1 \equiv y, b_2, \ldots, b_j)$ where $1 \leqslant p, q \leqslant r$, then in the composition $\rho\alpha$, α_p and α_q are joined into a cycle $(a_1, a_2, \ldots, a_i, b_1, b_2, \ldots, b_j)$, i.e., ρ is a fusion event affecting on α (and also called a *join operation* of α here). If $\rho = (x, y)$ is a 2-cycle and x and y are in the same cycle of α, say $\alpha_p = (a_1 \equiv x, a_2, \ldots, a_i \equiv y, a_{i+1}, \ldots, a_j)$ where $1 \leqslant p \leqslant r$, then in the composition $\rho\alpha$, this cycle α_p is broken into two disjoint cycles $(a_1, a_2, \ldots, a_{i-1})$ and $(a_i, a_{i+1}, \ldots, a_j)$, i.e., ρ is a fission event affecting on α (and also called a *split operation* of α here). If $\rho = (x, y, z)$ is a 3-cycle and x, y and z are in the same cycle of α, say $\alpha_p = (a_1 \equiv x, a_2, \ldots, a_i, b_1 \equiv y, b_2, \ldots, b_j, c_1 \equiv z, c_2, \ldots, c_k)$ where $1 \leqslant p \leqslant r$, then in the composition $\rho\alpha$, the cycle α_p becomes $(a_1, a_2, \ldots, a_i, c_1, c_2, \ldots, c_k, b_1, b_2, \ldots, b_j)$, i.e., ρ is a transposition event affecting on α. In [14], Lin et al. further observed that a block-interchange affecting on α corresponds to two 2-cycles, say ρ_1 and ρ_2, such that ρ_1 is a split operation of α and ρ_2 is a join operation of $\rho_1\alpha$. More clearly, let $\alpha_p = (a_1, a_2, \ldots, a_k)$ be a cycle of α, $\rho_1 = (a_1, a_i)$ and $\rho_2 = (a_h, a_j)$, where $1 < i \leqslant k$, $1 \leqslant h \leqslant i - 1$ and $i \leqslant j \leqslant k$.

Algorithm SortBI
Input: A permutation $\alpha = (a_1, a_2, \ldots, a_n)$ of $E = \{1, 2, \ldots, n\}$;
Output: A minimum series of block-interchange operations $\sigma_1, \sigma_2, \ldots, \sigma_\delta$ for sorting α into I;
1: Let $\delta = \frac{n - f(I\alpha^{-1})}{2}$;
2: **for** $i = 1$ to δ **do**
2.1: Arbitrarily choose two adjacent elements x and y in $I\alpha^{-1}$;
/* Note that (x, y) is a split operation of α. */
2.2: Circularly shift $(a_1, a_2, \ldots, a_n)$ such that $a_1 = x$ and assume $y = a_k$;
2.3: **for** $j = 1$ to n **do**
$\text{index}(a_j) = j$;
end for
2.4: Find two adjacent elements u and v in $I\alpha^{-1}(x, y)$ such that $\text{index}(u) \leqslant k - 1$ and $\text{index}(v) \geqslant k$;
/* Note that (u, v) is a join operation of $(x, y)\alpha$. */
2.5: $\sigma_i = (u, v)(x, y)$;
2.6: Compute $(u, v)(x, y)\alpha$ and denote it by α again;
2.7: Compute $I\alpha^{-1}(x, y)(u, v)$ and denote it by $I\alpha^{-1}$ again;
end for
3: Output $\sigma_1, \sigma_2, \ldots, \sigma_\delta$;

Then $\rho_2\rho_1\alpha$ is the permutation obtained from α by exchanging the blocks $[a_h, a_{i-1}]$ and $[a_j, a_k]$ of α_p. That is, $\rho_2\rho_1$ is a block-interchange event that affects on α by swapping $[a_h, a_{i-1}]$ and $[a_j, a_k]$, two non-intersecting blocks in α.

As discussed above, any series of block-interchanges required to sort one chromosome α into another I can be expressed by a product of 2-cycles, say $\rho_k\rho_{k-1}\ldots\rho_1$, such that $\rho_k\rho_{k-1}\ldots\rho_1\alpha = I$. Hence, $\rho_k\rho_{k-1}\ldots\rho_1 = I\alpha^{-1}$, suggesting that $I\alpha^{-1}$ contains all information, which can be utilized to derive $\rho_1, \rho_2, \ldots, \rho_k$ for sorting α into I.

Based on the above idea, Lin et al. [14] used permutation groups to design an $\mathcal{O}(\delta n)$ time algorithm to sort a permutation by block-interchanges, where δ is the minimum number of block-interchanges required to sort the permutation. In [14], Lin et al. first showed that $\delta = \frac{n - f(I\alpha^{-1})}{2}$, which can be calculated in $\mathcal{O}(n)$ time in advance. They then used a greedy method to derive an optimal series of δ block-interchanges from $I\alpha^{-1}$. Recall that a block-interchange can be modeled by a split operation followed by a join operation. Indeed, in each iteration of the greedy method, they showed that the block-interchange can be obtained by first arbitrarily choosing two adjacent elements from $I\alpha^{-1}$ that serve as a split operation of α and then finding the corresponding join operation in $\mathcal{O}(n)$ time, where two different elements x and y are said to be *adjacent* in a permutation β if $\beta(x) = y$ or $\beta(y) = x$. The details of this algorithm are described in Algorithm SortBI in this paper. It is not hard to see, the bottleneck of computation in the above algorithm is to find the join operations. Here, we observe the following property that can help us find the join operations in a more efficient way, so that the overall time complexity of Algorithm SortBI can be further improved.

Lemma 1. *Let α' be a permutation of $E = \{1, 2, \ldots, n\}$ and $z \in E$. Suppose that z and $I(z)$ are in different cycles of α'. Then $\alpha'(z)$ and $I(z)$ are adjacent in $I\alpha'^{-1}$, and $(\alpha'(z), I(z))$ is a join operation of α'.*

Proof. It is not hard to see that $I\alpha'^{-1}(\alpha'(z)) = I(z)$, which means that $\alpha'(z)$ and $I(z)$ are adjacent in $I\alpha'^{-1}$. Moreover, since z and $\alpha'(z)$ are in the same cycle of α', $\alpha'(z)$ and $I(z)$ are in different cycles of α'. Therefore, $(\alpha'(z), I(z))$ is a join operation of α'. □

Algorithm FastSortBI
Input: A permutation $\alpha = (a_1, a_2, \ldots, a_n)$ of $E = \{1, 2, \ldots, n\}$;
Output: A minimum series of block-interchange operations $\sigma_1, \sigma_2, \ldots, \sigma_\delta$ for sorting α into I;
1: Let $\delta = \frac{n - f(I\alpha^{-1})}{2}$;
2: **for** $i = 1$ to δ **do**
2.1: Arbitrarily choose two adjacent elements x and y in $I\alpha^{-1}$;
/* Note that (x, y) is a split operation of α. */
2.2: $\alpha' = (x, y)\alpha$;
/* Note that α' has two cycles, say C_1 and C_2. */
2.3: Find the maximum elements m_1 in C_1 and m_2 in C_2;
Let $z = \min(m_1, m_2)$;
Let $u = \alpha'(z)$ and $v = I(z)$;
/* Note that (u, v) are adjacent in $I\alpha'^{-1}$ and (u, v) is a join operation of α'. */
2.4: $\sigma_i = (u, v)(x, y)$;
2.5: Compute $(u, v)\alpha'$ and denote it by α again;
2.6: Compute $I\alpha^{-1}(x, y)(u, v)$ and denote it by $I\alpha^{-1}$ again;
end for
3: Output $\sigma_1, \sigma_2, \ldots, \sigma_\delta$;

Lemma 2. *Let α' be a permutation of $E = \{1, 2, \ldots, n\}$, C be a cycle of α', and z be the maximum element in C. Suppose that $z \neq n$. Then $\alpha'(z)$ and $I(z)$ are adjacent in $I\alpha'^{-1}$, and $(\alpha'(z), I(z))$ is a join operation of α'.*

Proof. Since $z \neq n$, $I(z) = z + 1$. Moreover, since z is the maximum element in C, $I(z)$ is not in C. That is, z and $I(z)$ are in different cycles of α'. Clearly, this lemma holds according to Lemma 1. □

By Lemma 2, we have the following observation immediately.

Observation 1. Let α' be a permutation of $E = \{1, 2, \ldots, n\}$ with two cycles C_1 and C_2. Let m_1 and m_2 be the maximum elements in C_1 and C_2, respectively. Suppose that $m_1 < m_2$. Then $m_2 = n$, $\alpha'(m_1)$ and $I(m_1)$ are adjacent in $I\alpha'^{-1}$, and $(\alpha'(m_1), I(m_1))$ is a join operation of α'.

Based on the above observation, step 2.4 of Algorithm SortBI can be done through finding the maximum elements of the two cycles in the permutation. The details are described in Algorithm FastSortBI. The maximum element in a cycle of n different elements can be found in $\mathcal{O}(n)$ time. Actually, we observe that this job can be done more efficiently by using a data structure called permutation tree, which was first proposed by Feng and Zhu [8].

Given a permutation $\alpha = (a_1, a_2, \ldots, a_n)$ of $E = \{1, 2, \ldots, n\}$, a *permutation tree* corresponding to α is defined to be a balanced binary tree with n leaves, whose nodes are labeled according to the following two rules. (1) The leaves are labeled by $a_1, a_2, \ldots, a_n$, respectively, and in this order. (2) Each of other nodes is labeled by the maximum of its children. By definition, the root r of a permutation tree has a label that is the maximum element in α. As an example, Fig. 2 is a permutation tree corresponding to $\alpha = (2, 5, 3, 1, 4, 6, 7, 8)$. Basically, there are following three operations that can be applied to a permutation tree.

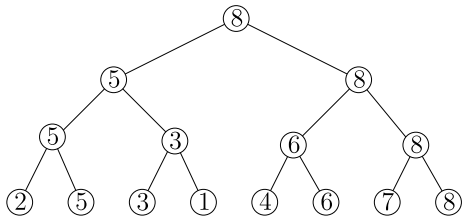

Fig. 2. A permutation tree corresponding to $\alpha = (2, 5, 3, 1, 4, 6, 7, 8)$.

They are: (1) BUILD, which is to build a permutation tree corresponding to a permutation, (2) JOIN, which is to join two permutation trees into a bigger one, and (3) SPLIT, which is to split a permutation tree into two smaller ones. The explicit definitions of the three operations are given in Definitions 1, 2 and 3, respectively. Feng and Zhu [8] have shown that the costs of the BUILD, JOIN and SPLIT operations are $\mathcal{O}(n)$, $\mathcal{O}(\log n)$ and $\mathcal{O}(\log n)$, respectively.

Definition 1. (See Feng and Zhu [8].) BUILD(α) creates a permutation tree, which corresponds to a given permutation α.

Definition 2. (See Feng and Zhu [8].) SPLIT(T, m) separates a permutation tree T corresponding to $\alpha = (a_1, \ldots, a_{m-1}, a_m, \ldots, a_n)$ into two trees such that one corresponds to $\alpha_l = (a_1, a_2, \ldots, a_{m-1})$, and the other to $\alpha_r = (a_m, a_{m+1}, \ldots, a_n)$.

Definition 3. (See Feng and Zhu [8].) JOIN(T_l, T_r) merges two permutation trees T_l corresponding to $\alpha_l = (a_1, a_2, \ldots, a_{m-1})$ and T_r corresponding to $\alpha_r = (a_m, a_{m+1}, \ldots, a_n)$ into a bigger permutation tree, which corresponds to $(a_1, \ldots, a_{m-1}, a_m, \ldots, a_n)$.

Below, we describe how to apply the data structure of permutation tree to Algorithm FastSortBI. Initially, we build a permutation tree corresponding to the source genome α. In step 2.2 of FastSortBI algorithm, (x, y) splits α into α' and hence the permutation tree corresponding to α is separated into two smaller trees corresponding to two cycles of α', say C_1 and C_2, respectively. Then, it takes constant time to obtain the maximum elements of C_1 and C_2 from the roots of their corresponding permutation trees. In step 2.5, (u, v) joins α' into α and the permutation trees corresponding to C_1 and C_2, respectively, are then merged into a bigger tree corresponding to α.

It should be noted that because a permutation is represented in cycle form, there is no difference for this permutation to take which of its elements as the first one in a cycle of the permutation. For example, $(2, 5, 3, 1, 4, 6, 7, 8) = (3, 1, 4, 6, 7, 8, 2, 5)$. However, for a permutation tree, its leaves are labeled according to a linear order of the elements in the corresponding permutation. Hence in step 2.2 of FastSortBI algorithm, although a 2-cycle splits α into $\alpha' = C_1C_2$ with two cycles, we may not obtain the two smaller permutation trees corresponding to C_1 and C_2, respectively, by using one SPLIT operation. For example, $(3, 6)$ is a split operation to $\alpha = (2, 5, 3, 1, 4, 6, 7, 8)$ and $(3, 6)\alpha = (3, 1, 4)(6, 7, 8, 2, 5)$. However, we cannot use a SPLIT operation to transform the permutation tree corresponding to $(2, 5, 3, 1, 4, 6, 7, 8)$ into two permutation trees corresponding to $(3, 1, 4)$ and $(6, 7, 8, 2, 5)$, respectively. In this case, two SPLIT and one JOIN operations are needed for its correct transformation (see Lemma 3 for details). A similar argument is applicable to the join operation of step 2.5.

Lemma 3. *Let a permutation $\alpha = (a_1, a_2, \ldots, a_n)$ and ρ_1 be a split operation that separates α into two cycles, say C_1 and C_2, that is $\rho_1\alpha = C_1C_2$. Then, by using at most two SPLIT and one JOIN operations, we transform a permutation tree T corresponding to α into two smaller permutation trees corresponding to C_1 and C_2, respectively.*

Proof. Let $\rho_1 = (a_i, a_j)$, where $1 \leqslant i < j \leqslant n$. If $a_i = a_1$, we can use SPLIT(T, j) to split T into two permutation trees that correspond to $C_1 = (a_1, a_2, \ldots, a_{j-1})$ and $C_2 = (a_j, a_{j+1}, \ldots, a_n)$, respectively. On the other hand, if $a_i \neq a_1$, SPLIT(T, i) leads to two permutation trees, T_l and T_r. Then, JOIN(T_r, T_l) merges the two permutation trees into a new one corresponding to the permutation $(a_i, \ldots, a_j, \ldots, a_n, a_1, \ldots, a_{i-1})$. We reassign the new tree as T, which still corresponds to α, but the first element in the cycle of α is changed from a_1 to a_i. Finally, SPLIT($T, j - i + 1$) operates on T to produce two permutation trees that correspond to $C_1 = (a_i, \ldots, a_{j-1})$ and $C_2 = (a_j, \ldots, a_n, a_1, \ldots, a_{i-1})$, respectively. □

Lemma 4. *Let $\alpha' = C_1C_2$ be a permutation with two cycles C_1 and C_2, and ρ_2 be a join operation of α', that is $\alpha = \rho_2\alpha'$. Then, by using at most two SPLIT and three JOIN operations, we transform the two permutation trees corresponding to C_1 and C_2, respectively, into a permutation tree corresponding to α.*

Proof. Let $C_1 = (a_1, a_2, \ldots, a_m)$ and $C_2 = (b_1, b_2, \ldots, b_p)$. Let T_1 and T_2 be permutation trees corresponding to C_1 and C_2, respectively, and $\rho_2 = (a_i, b_j)$, where $i \leqslant m$ and $j \leqslant p$. If $a_i \neq a_1$, then, as was discussed in Lemma 3, we can use one SPLIT followed by one JOIN to obtain a new permutation tree T_1 that corresponds to $(a_i, \ldots, a_m, a_1, \ldots, a_{i-1})$. Similarly, if $b_j \neq b_1$, one SPLIT and one JOIN can be used to obtain T_2 corresponding to $(b_j, \ldots, b_p, b_1, \ldots, b_{j-1})$. Finally, JOIN($T_1, T_2$) merges the two trees T_1 and T_2 into a bigger permutation tree corresponding to α. □

Theorem 1. *The problem of sorting by block-interchanges can be solved by Algorithm FastSortBI in $\mathcal{O}(n + \delta \log n)$ time.*

Proof. As discussed previously, Algorithm FastSortBI sorts α into I using a minimum number of block-interchange operations. Now, we analyze its time complexity as follows. In step 1, we build a permutation tree corresponding to α in $\mathcal{O}(n)$ time. As to step 2, there are δ iterations. In step 2.2, by Lemma 3, we obtain the permutation trees corresponding to C_1 and C_2, respectively, by at most two SPLIT and one JOIN operations, which totally take only $\mathcal{O}(\log n)$ time. In step 2.3, the maximum elements of the two cycles are obtained in constant time

from the roots of their corresponding permutation trees. In step 2.5, by Lemma 4, it takes $\mathcal{O}(\log n)$ time to obtain a new permutation tree corresponding to α. As a result, step 2 costs $\mathcal{O}(\log n)$ time in each of δ iterations. Therefore, the total time complexity of Algorithm FastSortBI is $\mathcal{O}(n + \delta \log n)$. □

In the following, we furthermore improve the cost of step 2 in Algorithm FastSortBI from $\mathcal{O}(\log n)$ to $\mathcal{O}(\log \delta)$. Hence, the overall time complexity of Algorithm FastSortBI can be further reduced down to $\mathcal{O}(n + \delta \log \delta)$.

Definition 4. Let α be the input permutation of Algorithm FastSortBI. Then, let $\Phi = \{I^{-1}(x) \mid I\alpha^{-1}(x) \neq x$, that is, x is a non-fixed element of $I\alpha^{-1}\}$.

Observation 2. Once an element is fixed in $I\alpha^{-1}$ or $I\alpha'^{-1}$ in the process of Algorithm FastSortBI, it is always fixed in $I\alpha^{-1}$ and $I\alpha'^{-1}$ during the subsequence process of the algorithm.

Theorem 2. *The problem of sorting by block-interchanges can be solved by Algorithm FastSortBI in $\mathcal{O}(n + \delta \log \delta)$ time.*

Proof. In step 2.3, the u and v are adjacent in $I\alpha'^{-1}$ by Lemma 2. This implies that at that moment, v is a non-fixed element of $I\alpha'^{-1}$. According to Observation 2, v is a non-fixed element before running step 2 in Algorithm FastSortBI and, therefore, $z \in \Phi$ since $v = I(z)$ (i.e., $z = I^{-1}(v)$). This suggests that before running step 2, we can use only those elements that are in $\Phi \cup \{n\}$ to build a permutation tree T for more simply corresponding to α and those elements that are not in $\Phi \cup \{n\}$ can be skipped. The reason for including n in the construction of T above is that $\max\{m_1, m_2\} = n$ in step 2.3 and hence n must be in T even though it may not be in Φ. It is not hard to see that there are at most $\mathcal{O}(\delta)$ elements in $\Phi \cup \{n\}$. In this case, however, the x and y in step 2.2 may not be in $\Phi \cup \{n\}$ and hence they may not be in T, resulting in that T cannot be separated into two permutation trees using $\mathsf{SPLIT}(T, x)$ and/or $\mathsf{SPLIT}(T, y)$, as was discussed in Lemma 3. To fix this problem, we first define $\Theta(w) = \alpha^k(w)$, where k is the smallest positive integer such that $\alpha^k(w) \in \Phi \cup \{n\}$. Then we can use $\mathsf{SPLIT}(T, \Theta(x))$ and/or $\mathsf{SPLIT}(T, \Theta(y))$ to correctly divide T into two permutation trees. Again, it is not hard to see that the computation of $\Theta(w)$ for all $w \notin \Phi \cup \{n\}$ can be done in advance in $\mathcal{O}(n)$ time. Therefore, the total time complexity of Algorithm FastSortBI is $\mathcal{O}(n + \delta \log \delta)$. □

Let us take $\alpha = (7, 8, 6, 5, 2, 3, 1, 4)$ for an example. We note that $I\alpha^{-1} = (1, 4, 2, 6)(5, 7)(3)(8)$ and $n = 8$. According to Algorithm FastSortBI, we understand that the block-interchange distance between α and I is $\delta = \frac{(8-4)}{2} = 2$, which means that α can be sorted into I using two block-interchange operations. Below, we show how to find these two block-interchange operations, say $\sigma_1 = \rho_{12}\rho_{11}$ and $\sigma_2 = \rho_{22}\rho_{21}$. Initially, we build a permutation tree T for corresponding to α by just using those elements in $\Phi \cup \{n\}$. In this case, $\Phi \cup \{n\} = \{1, 3, 4, 5, 6, 8\}$ and hence we build a permutation tree that corresponds to $(8, 6, 5, 3, 1, 4)$. By definition, we have $\Theta(2) = 3$ and $\Theta(7) = 8$. Then, we let $\rho_{11} = (4, 2)$ since 4 and 2 are adjacent in $I\alpha^{-1}$. After applying ρ_{11} to α, we obtain $\alpha' = \rho_{11}\alpha = (4, 2)(7, 8, 6, 5, 2, 3, 1, 4) = (4, 7, 8, 6, 5)(2, 3, 1)$. In this case, two SPLIT and one JOIN operations are needed to obtain the two permutation trees. First, $\mathsf{SPLIT}(T, 4)$ divides T into T_l and T_r corresponding to $(8, 6, 5, 3, 1)$ and (4), respectively. Next, $\mathsf{JOIN}(T_r, T_l)$ merges T_r and T_l into a new T corresponding to $(4, 8, 6, 5, 3, 1)$. Finally, $\mathsf{SPLIT}(T, \Theta(2)) = \mathsf{SPLIT}(T, 3)$ further divides T into two permutation trees that correspond to $(4, 8, 6, 5)$ and $(3, 1)$, respectively. Next, in step 2.3, we find $z = 3$ and hence $\rho_{12} = (\alpha'(3), I(3)) = (1, 4)$ is a join operation of α'. In this case, one SPLIT and two JOIN operations are required to merge the two permutation trees into a bigger one corresponding to $(1, 3, 4, 8, 6, 5)$. After the above iteration, we have $\alpha = (1, 2, 3, 4, 7, 8, 6, 5)$ and $I\alpha^{-1} = (1, 6)(5, 7)$. Similarly, in the second iteration, we can find that $\rho_{21} = (1, 6)$ and $\rho_{22} = (\alpha'(6), I(6)) = (5, 7)$.

4. Conclusions

In this paper, we studied the problem of sorting by block-interchanges that is to find a minimum series of block-interchanges to sort the permutation of the given chromosome. Here, we proposed an improved algorithm of $\mathcal{O}(n + \delta \log \delta)$ time based on permutation groups and permutation trees. It will be interesting to study whether the permutation groups and permutation trees can be further applied to other problems of genome rearrangement involved with other rearrangements, such as reversals and translocations.

References

[1] D.A. Bader, B.M.E. Moret, M. Yan, A linear-time algorithm for computing inversion distance between signed permutations with an experimental study, Journal of Computational Biology 8 (2001) 483–491.

[2] V. Bafna, P.A. Pevzner, Sorting by transpositions, SIAM Journal on Discrete Mathematics 11 (1998) 224–240.

[3] E. Belda, A. Moya, F.J. Silva, Genome rearrangement distances and gene order phylogeny in γ-Proteobacteria, Molecular Biology and Evolution 22 (2005) 1456–1467.

[4] A. Bergeron, J. Mixtacki, J. Stoye, On sorting by translocations, Journal of Computational Biology 13 (2006) 567–578.

[5] M. Bóna, R. Flynn, The average number of block interchanges needed to sort a permutation and a recent result of Stanley, Information Processing Letters 109 (2009) 927–931.

[6] D.A. Christie, Sorting permutations by block-interchanges, Information Processing Letters 60 (1996) 165–169.

[7] I. Elias, T. Hartman, A 1.375-approximation algorithm for sorting by transpositions, IEEE/ACM Transactions on Computational Biology and Bioinformatics 3 (2006) 369–379.

[8] J. Feng, D. Zhu, Faster algorithms for sorting by transpositions and sorting by block interchanges, ACM Transactions on Algorithms 3 (2007) 25.

[9] J.B. Fraleigh, A First Course in Abstract Algebra, 7th ed., Addison-Wesley, Boston, 2003.

[10] S. Hannenhalli, Polynomial-time algorithm for computing translocation distance between genomes, Discrete Applied Mathematics 71 (1996) 137–151.

[11] S. Hannenhalli, P.A. Pevzner, Transforming men into mice (polynomial algorithm for genomic distance problem), in: Proceedings of the

36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 581–592.

[12] S. Hannenhalli, P.A. Pevzner, Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals), Journal of the ACM 46 (1999) 1–27.

[13] H. Kaplan, R. Shamir, R.E. Tarjan, Faster and simpler algorithm for sorting signed permutations by reversals, SIAM Journal on Computing 29 (1999) 880–892.

[14] Y.C. Lin, C.L. Lu, H.-Y. Chang, C.Y. Tang, An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species, Journal of Computational Biology 12 (2005) 102–112.

[15] C.L. Lu, Y.-L. Huang, T.C. Wang, H.-T. Chiu, Analysis of circular genome rearrangement by fusions, fissions and block-interchanges, BMC Bioinformatics 7 (2006) 295.

[16] J. Meidanis, Z. Dias, An alternative algebraic formalism for genome rearrangements, in: D. Sankoff, J.H. Nadeau (Eds.), Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families, Kluwer Academic Publisher, New York, 2000, pp. 213–223.

[17] J. Meidanis, Z. Dias, Genome rearrangements distance by fusion, fission, and transposition is easy, in: G. Navarro (Ed.), Proceedings of String Processing and Information Retrieval, IEEE Computer Society, Los Alamitos, CA, 2001, pp. 250–253.

[18] M. Ozery-Flato, R. Shamir, An $\mathcal{O}(n^{3/2}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations, in: Lecture Notes in Computer Science, vol. 4009, 2006, pp. 258–269.

[19] P.A. Pevzner, G. Tesler, Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes, Genome Research 13 (2003) 37–45.

[20] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B.F. Lang, R. Cedergren, Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome, Proceedings of the National Academy of Sciences 89 (1992) 6575–6579.

[21] E. Tannier, A. Bergeron, M.F. Sagot, Advances on sorting by reversals, Discrete Applied Mathematics 155 (2007) 881–888.