An \(O(n^3)\) Algorithm for Sorting Signed Genomes by Reversals, Transpositions, Transreversals and Block-Interchanges

SHUZHI YU
Department of Computer Science, National University of Singapore, 13 Computing Drive, Singapore 117417, Republic of Singapore
yushuzhi@u.nus.edu

FANCHANG HAO
School of Information and Key Laboratory of Evidence-Identifying in Universities of Shandong, Shandong University of Political Science and Law, Jinan, Shandong, P.R. CHINA 250014
haofine@gmail.com

HON WAI LEONG
Department of Computer Science, National University of Singapore, 13 Computing Drive, Singapore 117417, Republic of Singapore
leonghw@comp.nus.edu.sg

Received (15 July 2015)
Revised (27 Sep 2015)
Accepted (12 Oct 2015)

We consider the problem of sorting signed permutations by reversals, transpositions, transreversals, and block-interchanges. The problem arises in the study of species evolution via large-scale genome rearrangement operations. Recently, Hao, Zhang and Leong gave a 2-approximation scheme called genome sorting by bridges (GSB) for solving this problem. Their result extended and unified the results of (i) He and Chen – a 2-approximation algorithm allowing reversals, transpositions and block interchanges (by also allowing transversals) and (ii) Hartman and Sharan – a 1.5-approximation algorithm allowing reversals, transpositions and transversals (by also allowing block-interchanges). The GSB result is based on introduction of three bridge structures in the breakpoint graph, the L-bridge, T-bridge, and X-bridge that models good reversal, transposition/transreversal, and block-interchange, respectively. However, the paper by Hao et al focused on proving the 2-approximation GSB scheme and only mention a straight-forward \(O(n^6)\) algorithm. In this paper, we give an \(O(n^3)\) algorithm for implementing the GSB scheme. The key idea behind our faster GSB algorithm is to represent cycles in the breakpoint graph by their canonical sequences, which greatly simplifies the search for these bridge structures. We also give some comparison results (running time and computed distances) against the original GSB implementation.

Keywords: algorithm, genome rearrangement, genome sorting, approximation algorithm, reversal, transreversal, transposition, block-interchange.
1. Introduction

In the genome rearrangement problem, we are given the gene sequence (genomes) $A$ and $B$ of two species, and a set of rearrangement operations (such as reversals and transpositions), we want to compute the evolutionary distance between $A$ and $B$, which is defined as the minimum number of rearrangement operations that will transform $A$ to $B$.\(^{11}\)

This paper considers the special case of the genome rearrangement problem for two unichromosomal genomes $A$ and $B$ with no duplicated genes and allowing the following rearrangement operations: reversals, transpositions, transreversals, and block-interchanges. A reversal operation (also called inversion) involves reversing a segment (a contiguous block of genes) of the genome; a transposition operation transposes two adjacent segments of the genome; a block-interchange is a generalized transposition that allows the two segments to be non-adjacent in the genome. Finally, a transversal operation involves two adjacent segments of the genome; it reverses one of the segments and then transposes them.

The genome rearrangement problem has a rich history. Here, we give a very brief survey. (We refer the reader to Huang and Lu\(^{11}\) for a more detailed survey.) Early works consider only the reversal operation.\(^{13,2}\) A major break-through was the result by Hannenhalli and Pevzner showing that sorting signed permutations with reversals can be done in polynomial time.\(^{5}\) The algorithm uses a breakpoint graph to study the cycle decomposition and to find good reversal operations to decompose these cycles. The theory was later simplified by Bergeron and faster algorithm has been developed by Kaplan, Shamir and Tarjan.\(^{3,12}\)

There are many variants of this problem that allows different sets rearrangement operations. Sorting by block-interchanges was solved by Christie and faster algorithms were later proposed.\(^{4}\) However, sorting by transposition was shown to be NP-hard by Bafna and Pevzner\(^{2}\) and they gave a 1.5-approximation algorithms. Another 1.5 approximation was given by Hartman.\(^{7}\)

The results that are most relevant to our work are as follows: For sorting by reversals and block-interchanges (but no transreversals), He and Chen gave a 2-approximation algorithm in 2003.\(^{10}\) For sorting by transpositions and transreversals (but no block-interchanges), the best result is 1.5-approximation algorithm by Hartman and Sharan.\(^{8}\)

Recently, Hao, Zhang and Leong gave a 2-approximation algorithm called genome sorting by bridges (GSB) for sorting signed permutations by reversals, transpositions, transreversals, and block-interchanges.\(^{6}\) Their result extends and unifies the results of He and Chen (by allowing transversals) and that of Hartman and Sharan (by allowing block-interchanges).\(^{10,8}\) The GSB is based on three bridge structures in the breakpoint graph, called the $L$-bridge, $T$-bridge, and $X$-bridge, and they model “good” reversals, transpositions/transreversals, and block-interchanges, respectively. The paper focused mostly on proving the 2-approximation GSB scheme and only gave a straight-forward $O(n^6)$ algorithm.\(^{6}\)
In this paper, we present an $O(n^3)$ algorithm for implementing the GSB scheme. The key idea behind our faster GSB algorithm is to represent the cycles in the breakpoint graph by their canonical sequences (defined later). This greatly simplifies the search for the various bridge structures. We give some comparison (of the running times and computed distances) with the original GSB implementation.

2. Problem Formulation and Previous Result

Our problem formulation and definitions follow closely those in Hao, Zhang and Leong. We are given two unichromosomal genomes $A$ and $B$ with no duplicated genes, we want to find a shortest sequence of operations that transforms $A$ to $B$ using reversals, transpositions, transreversals, and block-interchanges.

Each genome is represented by a signed permutation $X = (x_1, x_2, ..., x_n)$, where each gene $x_i$ is represented by an integer from $1, 2, ..., n$ and has a sign “+” or “-” to indicate its orientation in the genome. If we assume, without loss of generality, that $B$ is the identity genome, then transforming genome $A$ to $B$ is akin to sorting the signed permutation $A$. We extend genome $X$ by adding dummy genes $x_0=0$ and $x_{n+1} = n+1$ at both ends as sentinels, to get $X = (0, x_1, x_2, ..., x_n, n + 1)$. The dummy genes are unsigned and are not involved in any rearrangement operations. For genome $A = (1 -3 -2 4 6 -5)$ with 6 genes, the extended genome is $A = (0 1 -3 -2 4 6 -5 7)$. (For readability, we omit the “+” sign for genes 1, 4, and 6 and the commas in between genes.) The identity genome is given by $Id = (0 1 2 3 4 ... n n+1)$. Henceforth, we shall use the term genome to mean the extended genome.

2.1. Breakpoint Graph

We define the breakpoint graph $G_A(V,E)$ of genome $A=(0, x_1, ..., x_n, n + 1)$ with respect to identity genome $B$. We first define the vertices in $G_A$: each positively oriented gene $+y$ is replaced by two vertices $(yt, yh)$ where the suffixes $t$ and $h$ stand for tail and head, respectively. Each negatively oriented gene $-y$ is replaced by two vertices $(yh, yt)$. Let $L(x_i)$ and $R(x_i)$ denote the left and right vertex for gene $x_i$, respectively. Then if $x_i=-y$, then $L(x_i)=yt$ and $R(x_i)=yh$; if $x_i=+y$, then $L(x_i)=yh$ and $R(x_i)=yt$. We also replace dummy gene 0 by vertex $0h$ and $(n + 1)t$ by $(n + 1)$. This is shown in Fig. 1(a), where the gene $x_1=1$ is replaced by the vertices $1t$ and $1h$, and the gene $x_2=-4$ is replaced by $4h$ and $4t$.

Next, the edges of $G_A$ are colored either gray or black. We first define the gray (dotted) edge in $G_A$ to be $(x_{i-1}h, x_it)$, for $i=1, 2, ..., n + 1$. The gray edges define the adjacencies of the genes in the target genome $B = Id$ (the identity genome). We define the black (solid) edges in $G_A$ for the adjacencies of the genes in $A$, namely, between gene $x_{i-1}$ and $x_i$ for $i=1, ..., n + 1$. Thus, we connect the vertex $R(x_{i-1})$ to the vertex $L(x_i)$. Notice that depending on the orientation of $(x_{i-1}, x_i)$, there are four types of black edges: $(h, t)$, $(h, h)$, $(t, t)$, and $(t, h)$ black edge, for a $(+, +)$, $(+, -)$, $(-, +)$ and $(-, -)$ combination, respectively.
Fig. 1(a) shows the breakpoint graph $G_A$ for the genome $A = (0\ 1\ -4\ 3\ 2\ 5\ 6)$. The black edges are shown as solid horizontal lines, while the gray edges are shown as curved dotted lines. There are 12 vertices, 6 black edges and 6 gray edges. Fig. 1(b) shows the breakpoint graph for the identity genome $Id = (0\ 1\ 2\ 3\ 4\ 5\ 6)$.

We give some properties of the breakpoint graph $G_A$. The graph $G_A$ has exactly $2(n+1)$ vertices, and $(n+1)$ black edges and $(n+1)$ gray edges. Each vertex in $G_A$ has degree 2, and connects to exactly one black and one gray edge (see Fig. 1). A path in $G_A$ from $v_1$ to $v_m$ (denoted by $\langle v_1, v_m \rangle$) is a sequence, $v_1, v_2, ..., v_m$, of distinct vertices where the edge $(v_i, v_{i+1})$ is in $E(G_A)$, for $i=1,2,...,m-1$. A cycle is a path where $v_m = v_1$. For example, in Fig. 1(b), there are 6 cycles in the identity genome. A path $P$ is said to be alternating if the colors of every two consecutive edges in $P$ are distinct. All paths and cycles in $G_A$ are alternating. The length of a path or cycle is the number of edges in it. The graph $G_A$ is uniquely decomposed into several disjoint alternating cycles. A cycle is short if it is of length 2, with one black and one gray edge. Cycles with length bigger than 2 are long cycles. In Fig. 1(a), the breakpoint graph $G_A$ contains two short cycles and one long cycle.

![Fig. 1](image-url)  
Fig. 1. (a) Breakpoint graph $G_A$ for genome $A = (0\ 1\ -4\ 3\ 2\ 5\ 6)$. (b) Breakpoint graph for Identity Genome $(0\ 1\ 2\ 3\ 4\ 5\ 6)$.

### 2.2. Quick Review of the GSB Algorithm

We give a quick review of the GSB algorithm. It is well-known that when sorting the genome, the number of cycles in $G_A$ increases until it reaches $(n+1)$. Hao et al defines 3 bridge structures to model good rearrangement operations (that increase the number of cycles). Let $\Delta(\rho)$ denote the increase in the number of cycles in $G_A$ after performing rearrangement operation $\rho$. Then, an $L$-bridge (shown in Fig. 2(a)) models reversal with $\Delta = 1$. A $T$-bridge models transposition, left-transreversal, or right-transreversal with $\Delta = 2$. These are shown in Fig. 2(d), (e), or (f), respectively. Finally, an $X$-bridge (Fig. 2(b), (c)) models block interchange with $\Delta = 2$.

It was shown in Hao et al (Theorem 2) that it is always possible to find one of these 3 bridges in any breakpoint graph. And their 2-approximation result holds it even if the bridges are found in different order. Hence, to reduce the number of operations, Algorithm GSB first searches for a $T$-bridge or $X$-bridge (with $\Delta = 2$) and only use an $L$-bridge (with $\Delta = 1$) when they do not exist. Hao et al focused only on proving the 2-approximation GSB scheme and only gave a straight-forward $O(n^6)$ algorithm. This paper gives a more efficient $O(n^3)$ algorithm.
3. A Faster GSB Algorithm: GSB-I

3.1. An Overview

The main idea behind a more efficient algorithm for GSB is that each bridge structure can be characterized by a canonical pattern defined by the relative parity relationship between a few black edges in the cycle(s). And once we have computed the parity of all the black edges in a cycle \( C \) relative to a fixed black edge, we know the parity between any pair of black edges. We can then design algorithms to efficiently search for the pattern that corresponds to each of the bridge structures.

Given two black edges \( b_i \) and \( b_j \), we say that \( b_i < b_j \) iff \( b_i \) is to the left of \( b_j \) in the genome. In Fig. 1(a), \([4t, 3t] < [3h, 2t]\).

**Parity between Two Black Edges:** Given two black edges \( b_i \) and \( b_j \) in a long cycle \( C \), we define \( \text{parity}(b_i, b_j) = + \) if the left end point of \( b_i \) is connected (by a gray path) to the right end point of \( b_j \). Otherwise, \( \text{parity}(b_i, b_j) = - \). In Fig. 1(a), \( \text{parity}(b_1, b_2) = + \) because the left end point of \( b_1 \), vertex \( 1h \) is connected by a gray edge to the right end point of \( b_2 \). Similarly, \( \text{parity}(b_1, b_3) = - \).

**Canonical Sequence of a Cycle:** Let \( C \) be a long cycle with \( k > 1 \) black edges. Define \( b_1 \) to be the leftmost black edge in \( C \). Then starting from the left end point of \( b_1 \), we traverse through the edges in \( C \) to reach the black edges \( b_2, b_3, \ldots, b_k \). Then, we define the canonical sequence for cycle \( C \) to be the cyclic signed sequence \( cs(C) = (c_1, c_2, \ldots, c_k) \) where \( c_j = + b_j \) if \( \text{parity}(b_1, b_j) = + \) and \( c_j = - b_j \) if \( \text{parity}(b_1, b_j) = - \). Note that, like the cycle \( C \), \( cs(C) \) is cyclic and hence when traversing \( cs(C) \), the black edge after \( c_k \) is \( c_1 \) (with wraparound). (By definition, the sign of \( b_1 \) is always positive.) Note that, for any two black edges \( b_i \) and \( b_j \), we can easily compute \( \text{parity}(b_i, b_j) \) since \( \text{parity}(b_i, b_j) = + \) iff \( \text{parity}(b_1, b_i) = + \) and \( \text{parity}(b_1, b_j) = + \).
parity$(b_1, b_j)$.

Consider cycle $C_1$ in Fig. 1(a). The leftmost black edge in $C$ is $b_1 = [1b, 4h]$. Then, the black edges $b_1, b_2, b_3,$ and $b_4$ are defined by start at the black edge $b_1$ and following the edge sequence in the cycle $C_1$. The canonical sequence is $cs(C_1) = (+b_1, +b_2, -b_3, -b_4)$. Note that the sign of each black edge is always defined with respect to its parity with the leftmost black edge $b_1$. Since $cs(C_1)$ is cyclic, when traversing $cs(C_1)$, after $-b_4$ is $+b_1$ (with wrap-around).

### 3.2. Canonical Patterns of the L, T, and X Bridges

We are now ready to define the canonical patterns $cp(B)$ that characterizes the bridge structure $B$, for the $L$, $T$, and $X$-bridges.

**Canonical Pattern of the L-Bridge:** The simplest is the L-bridge which is defined by two black edges $\alpha = [a, b]$ and $\beta = [c, d]$ in the cycle $C$ with $(\alpha < \beta)$ and where parity$(\alpha, \beta) = "$-". This is shown in Fig. 2(a). We call $cp(L) = (\alpha, -\beta)$ the canonical pattern for the L-bridge. Note that $(\alpha, -\beta)$ is a subsequence of $cs(C)$. Hence, a search for an L-bridge in a cycle $C$ become a search for a subsequence (of length-2) of $cs(C)$ that satisfies the canonical pattern of the L-bridge.

In Fig. 1(a), the cycle $C_1$ has canonical sequence $cs(C_1) = (+b_1, +b_2, -b_3, -b_4)$. Then, the subsequence $(+b_1, -b_2)$ is a canonical pattern for an L-bridge in $C_1$ (with $\alpha = b_1$ and $\beta = b_2$) since $b_1 < b_3$ and parity$(b_1, b_2) = "$-". Here, we note that $(+b_1, -b_4)$ is another L-bridge in $C_1$.

**Canonical Pattern of the 3 types of T-Bridges:** There are three types of T-bridges (Fig. 1(d)-(f)) and they are defined by three black edges $\alpha, \beta, \gamma$ with $(\alpha < \beta < \gamma)$. Their canonical patterns are given by:

- $cp(tr-T) = (\alpha, +\beta, +\gamma)$ for transposition T-bridge,
- $cp(ltr-T) = (\alpha, +\beta, -\gamma)$ for left-transreversal T-bridge, and
- $cp(rtr-T) = (\alpha, -\gamma, -\beta)$ for right-transreversal T-bridge.

Hence, a search for a T-bridge in a cycle $C$ become a search for a subsequence (of length 3) of $cs(C)$ that satisfies the corresponding canonical pattern.

In the cycle $C_1$ in Fig. 1(a) with $cs(C_1) = (+b_1, +b_2, -b_3, -b_4)$, we can easily verify that the subsequence $cp(ltr-T) = (+b_1, +b_2, -b_4)$ is a canonical pattern for a left-transreversal T-bridge in $C_1$ (with $\alpha = b_1$, $\beta = b_2$ and $\gamma = b_4$).

**Canonical Pattern of the two X-Bridges:** The two types of X-bridges (shown in Fig. 2(b)-(c)) are quite different and we discuss them separately.

The Single-Cycle-X-bridge shown in Fig. 2(b) is defined by four black edges $\alpha, \beta, \gamma, \delta$ in cycle $C$ with $(\alpha < \beta < \gamma < \delta)$. The canonical pattern is given by the subsequence $cp(sc-X) = (\alpha, -\beta, -\gamma, +\delta)$ in $cs(C)$. Hence, a search for a Single-Cycle-X-bridge in a cycle $C$ is also a search for a subsequence (length 4) of $cs(C)$ that satisfies the corresponding canonical pattern.

For example, the long cycle $C$ in Fig. 3(b) has canonical sequence $cs(C) = (+b_1, -b_2, -b_3, -b_4, +b_5)$, and we can easily verify that the subsequence $cp(sc-X) = (+b_1, -b_3, -b_4, +b_5)$ is a canonical pattern for a Single-Cycle-X-bridge in $C_1$ (with...
Algorithm 1: Detection of \(L\)-bridge

1: procedure Detect-L-Bridge(C);
   /*C has \(k\) (\(\geq 1\)) black edges, \(cs(C)\) has been precomputed*/
   /*Looking for a pattern \((b_1, \ldots, b_i)\)*/
2:   for \((1 < i \leq k)\) do
3:     if \(parity(b_1, b_i) = \text{"-"}\) then /*recall \(b_1\) is leftmost black edge in \(C\)*/
4:        found an \(L\)-bridge \((b_1, \ldots, b_i)\); /*\(\alpha = b_1, \beta = b_i\)*/

In Fig. 3(a), the cycle \(C_a\) has \(cs(C_a) = (b_1, +b_2, -b_3)\), where \(b_1 < b_2 < b_3\). Hence the algorithm finds \(-b_3\) as the first negative black edge in \(cs(C_a)\) and detects the \(L\)-bridge with \(cp(L) = (b_1, -b_3)\) (shown on the right).
3.3.2. Detecting a T-bridge

There are three types of T-bridges with very similar canonical patterns. We give the algorithm for detecting a left-transreversal T-bridge. The algorithms for detecting a right-transreversal and transposition T-bridge are very similar and omitted here.

Algorithm 2 shows the algorithm to detect a left-transreversal T-bridge in a cycle $C$. It searches for the canonical pattern $cp(ltr\text{-}T) = (\alpha, +\beta, -\gamma)$ in $cs(C)$. We note that (unlike the L-bridge) the canonical pattern for the ltr-T-bridge may not start with $\alpha = b_1$; it may start with any black edge $b_i$ in $cs(C)$. We also note that $cs(C)$ is cyclic, and so the canonical pattern $cp(ltr\text{-}T) = (\alpha, +\beta, -\gamma)$ is also cyclic. Hence the algorithm that searches for the canonical pattern for the ltr-T-bridge tries each black edge $b_i$ in $cs(C)$ as candidate for $\alpha$. For each candidate $b_i$, the algorithm traverses (with wrap-around) the black edge $b_j$ in $cs(C)$ starting after $b_i$. Notice that the traversal from $b_i$ is toward the direction following the gray edge from the left end point of $b_i$. For each $b_j$ in the traversal, it looks for the pattern $(b_i, +b_{lp}, -b_j)$, where $+b_{lp}$ is the leftmost positive black edge scanned that also satisfies the condition that $(b_i < b_{lp} < b_j)$.

The correctness of the algorithm follows from the fact that if a pattern $(b_i, +b_m, -b_j)$ exists with $(b_i < b_m < b_j)$ for some $b_m$, then our algorithm will find the pattern where $m = lp$. Furthermore, if $b_{lp}$ cannot be found by the algorithm, then no such pattern exists. The complexity of the algorithm is $O(k^2)$.

In Fig. 4 the cycle $C$ has $cs(C) = (b_1, -b_2, +b_3, -b_4)$. In this example, there is no pattern that starts with $b_1$. The algorithm continues to search with $b_2$ ($i=2$) and traverses in anticlockwise direction $(b_2, b_1, b_4, b_3)$. Then it finds the pattern $(b_2, +b_4, -b_3)$ with $b_2 < b_4 < b_3$ which corresponds to the ltr-T-bridge shown on the right. (Note the reversal of signs of the black edges since the starting black edge is $b_2$.)
We discuss separately the two types of X-bridge.

3.3.3. Detecting an X-bridge

We discuss separately the two types of X-bridge, namely Single-Cycle-X-bridge and Double-Cycle-X-bridge as shown in Fig. 2(b) and (c).

For Single-Cycle-X-bridge: Algorithm 3 shows the pseudo code of detecting a Single-Cycle-X-bridge in a long cycle \( C \). It searches for the canonical pattern \( cp(sc-X) = (\alpha, -\beta, -\delta, +\gamma) \) in \( cs(C) \). Without loss of generality, we can assume that \( \alpha = b_1 \). The proof is given in Section 4.

The black edge \( \beta \) in \( cp(sc-X) \) can be any negative black edge in \( cs(C) \). Hence the algorithm tries all the negative black edges \( -b_i \) as candidate for \( \beta \). For each candidate \( -b_i \), the algorithm traverses through all black edges \( b_j \) in \( cs(C) \) starting after \( b_j \) looking for the pattern \( (b_1, -b_i, -b_{rn}, +b_j) \) where \( -b_{rn} \) is the rightmost negative black edge scanned that also satisfies the condition \( b_i < b_j < b_{rn} \).

The correctness of the algorithm follows from the fact that if a pattern \( (\alpha, -\beta, -\delta, +\gamma) \) exists with \( (\alpha < \beta < \gamma < \delta) \) for some \( \delta \), then our algorithm will find the pattern with \( \delta = b_{rn} \) where \( -b_{rn} \) is the rightmost negative black edge between \( \beta \) and \( \gamma \) in \( cs(C) \). Furthermore, if \( -b_{rn} \) cannot be found by the algorithm for any negative black edge \( \beta \) in \( cs(C) \), then no such pattern exists. The complexity of the algorithm is \( O(k^3) \).

Use the long cycle \( C_b \) in Fig. 3(b) as an example. \( C_b \) has \( cs(C_b) = (b_1, -b_2, -b_3, -b_4, +b_5) \). In this example, there is no pattern that starts with \( -b_2 \) as the negative black edge \( \beta \). Then the algorithm continue to search with \( -b_3 \) (i = 3)

---

**Algorithm 2**: Detection of LTR-T-bridge

1. procedure DETECT-LTR-T-BRIDGE\( (C) \)
   1. /*C has \( k(\geq 1) \) black edges, \( cs(C) \) has been precomputed*/
   2. /*Looking for a pattern \( (b_i, +b_{p}, -b_j)* \)

3. for \( i = 1 \) to \( k \) do /*search with \( b_i * \)
4. for each black edge \( b_j \) in the traversal starting from \( b_i \) (with wrap-around) do
5. if \( b_i < b_j \) then
   6. if parity\( (b_i, b_j) = + + \) and \( b_j < b_{lp} \) then
     7. \( lp = j \)
   8. else if parity\( (b_i, b_j) = - - \) and \( b_j < b_{lp} \) then
     9. found a left-transreversal T-bridge \( (b_i, +b_{lp}, -b_j) \)

---

**Fig. 4.** Illustration of left-transreversal T-bridge detection algorithm. In this example, there is no canonical pattern for ltr-T-bridge with \( \alpha = b_1 \). The algorithm finds the canonical pattern \( cp(ltr-T) = (b_2, +b_3, -b_4) \) that starts with \( \alpha = b_2 \). The figure on the right show the corresponding left-transreversal T-bridge, where the the thick dashed line represents the gray path \( (2t, 1h, 2h, 3t) \).
and finds the pattern \((b_1, -b_3, -b_4, +b_5)\) with \(b_1 < b_3 < b_5 < b_4\), which corresponds to the Single-Cycle-X-bridge shown on the right.

**For Double-Cycle-X-bridge:** Recall that a Double-Cycle-X-bridge is defined by two interleaving cycles \(C_1\) and \(C_2\), with two black edges \(e_1, e_3\) from \(C_1\) and \(e_2, e_4\) from \(C_2\), where \(e_1 < e_3 < e_2 < e_4\) (See Fig. 2(c)).

Hence, in any cycle \(C\), only pairs of black edges whose parity is “+” can possibly be part of a Double-Cycle-X-bridge. Hence, for each long cycle \(C\), we partition the edges into \(C^+\) containing all the black edges in \(cs(C)\) with sign “+”, and \(C^-\) containing all the black edges in \(cs(C)\) with sign “−”.

Given two long cycles \(C_1\) and \(C_2\), we consider four combinations: between \(C_1^+\) and \(C_2^+\), between \(C_1^+\) and \(C_2^-\), between \(C_1^-\) and \(C_2^+\), and between \(C_1^-\) and \(C_2^-\).

For each combination, we search for the appropriate canonical pattern of interleaving cycles of the Double-Cycle-X-bridge. We present Algorithm 4 that searches for a Double-Cycle-X-bridge between \(C_1^+\) and \(C_2^+\). (The other cases are similar and are omitted here.)

Suppose \(C_1^+ = \{f_1, f_2, ..., f_m\}\) and \(C_2^+ = \{g_1, g_2, ..., g_n\}\) where the order of black edges in \(C_1^+\) and \(C_2^+\) is based on their relative positions. This means the first black edge in \(C_1^+\) \(f_1\) is the leftmost black edge in \(C_1^+\), and the last black edge in \(C_1^+\) \(f_m\) is the rightmost black edge in \(C_1^+\). Suppose \(f_1 < g_1\). The algorithm traverses each black edge in \(C_1^+\) after \(f_1\) looking for a pattern \((f_1, g_1, f_1, g_n)\) where \(f_1\) is a scanned black edge in \(C_1^+\) that satisfies the condition \((g_1 < f_1 < g_n)\).

For any two long cycles (with no T-bridge or Single-Cycle-X-bridge), we run Algorithm 4 on each of the four combinations to detect possible Double-Cycle-X-bridge. Suppose the number of black edges in cycle \(C_i\) is \(m_i\), the total number of black edges in the whole genome is \(k\) and there are \(l\) long cycles. The complexity of using Algorithm 4 to detect possible Double-Cycle-X-bridge on any two long cycles \(C_1\) and \(C_2\) would be \(O(m_1 + m_2)\). We check each long cycle with all previous long cycles to detect possible Double-Cycle-X-bridge. Thus for one round, it takes \((0 + O(m_1 + m_2)) + O(m_1 + m_2 + 2 \cdot m_4) + ... + O(m_1 + ... + (m_{l-1}) + (l-1) \cdot m_l)) = O(l \cdot k) = O(k^3)\) to detect Double-Cycle-X-bridge.

The correctness of the algorithm follows from the fact that if a pattern \((+e_1, +e_2, +e_3, +e_4)\) exists with \((e_1 < e_2 < e_3 < e_4)\) for some \(e_3\), then our al-
Algorithm 4 Detection of Double-Cycle-X-bridge

1: procedure DETECT-Doubel-Cycle-X-Bridge(C₁⁺, C₂⁺)
   /*Looking for a (f₁, g₁, fᵢ, gᵣ) pattern*/
   /*suppose f₁ < g₁*/
   /*suppose C₁⁺ has m elements and let fᵢ denotes iᵗʰ element*/
   2: for each k = 2 to m do
   3: if g₁ < fᵢ < gᵣ then
   4: found Double-Cycle-X-bridge (f₁, g₁, fᵢ, gᵣ)

Use Fig. 3(c) as an example. C₁⁺ = e₁, e₃, C⁻₁ = e₄, C⁺₂ = e₂, and C⁻₂ = e₃, e₆ where e₃ < e₅ < e₆. Thus we find Double-Cycle-X-bridge (e₁, e₃, e₅, e₆), which is shown on the right side of (c).

3.3.4. Algorithm GSB-I

GSB-I, as shown in Algorithm 5, tries to solve T- and X-bridges first and then the leftmost L-bridge. L-bridge must exist given that no T- or X-bridge exists.

Algorithm 5 : GSB-I

1: procedure GSB-I(Genome)
2: while bridge exist do
   /*suppose genome has k long cycles and Cᵢ represents 𝓵ᵗʰ long cycle*/
   onlyLBridge = true
   3: for each i = 1 to k do
   4: if found T-/X-bridge in Cᵢ then
   5: do transposition/left-/right-transreversal or block-interchange
   6: onlyLBridge = false, then exit
   7: else
   8: separate and store positive and negative edges
   9: if detect Double-Cycle-X-bridge with previous long cycle then
   10: do block-interchange, onlyLBridge = false, then exit
   11: if onlyLBridge then
   12: find leftmost L-bridge and do reversal /*since no T-/X-bridge exists*/

4. Correctness of GSB-I Algorithm

We need to prove that GSB-I is able to detect and operate correspondingly on an existing bridge inside an unsorted genome (bridges exist). We have shown in
Section 3.3 that GSB-I is able to detect an existing $L/T$-bridge within a long cycle. In addition, we have shown GSB-I is able to detect Single-Cycle-X-bridge whose leftmost black edge is $b_1$. However, we still need to show the correctness of the assumption that let $\alpha$ be $b_1$ in Lemma 1. Since the algorithm traverses each long cycle, it is able to detect one of the existing bridges that reside in one long cycle.

If none is detected, we also have shown in Section 3.3.3 that one of the Double-Cycle-X-bridges (guaranteed to exist by work of Hao et al) will be detected.

**Lemma 1.** In a long cycle $C$ without any $T$-bridges, there always exists Single-Cycle-X-bridge whose leftmost black edge is $b_1$, if there exists any Single-Cycle-X-bridge.

**Proof.** We first claim that in a long cycle $C$, if there exists a Single-Cycle-X-bridge whose leftmost edge is not $b_1$, then there exist $T$-bridge or Single-Cycle-X-bridges whose leftmost edge is $b_1$.

To prove this claim is to discuss four cases. The four cases (eight sub-cases), which are shown in Fig. 5, correspond to which of the four paths (marked in Fig. 2(b)) contains $b_1$ and how $b_1$ is connected in the path. The cut black edges and their connecting paths make up the bridge noted below the cycle. By checking all the four cases, we proved the claim. However, as there is no $T$-bridge in $C$, there must exist Single-Cycle-X-bridge whose leftmost edge is $b_1$.

![Fig. 5. Four cases (eight sub-cases) where the leftmost black edge $b_1$ of $C$ is contained in Path 1, 2, 3, or 4 as marked in Fig. 2(b).](image)

5. Time Complexity Analysis

Suppose the number of black edges in long cycle $C_i$ is $m_i$, the total number of black edges in the whole genome is $k$ and there are $l$ long cycles.

As shown in Section 3.3, for a long cycle $C_i$, $L$-bridge detection takes $O(m_i)$ time; $T$-bridge and Single-Cycle-X-bridge detection can be done in $O(m_i^2)$. In addition, the total time used to detect Double-Cycle-X-bridge for one round is $O(k^2)$. 
An $O(n^3)$ Algorithm for Sorting Signed Genomes

There are at most $k$ cycles to increase, and GSB-I increases at least one cycle each round. This means at most we inspect the whole genome $k$ times. Each time we continuously check whether the next long cycle $C_i$ has $T-/Single-Cycle-X-/L$-bridge, which takes $O(m_i^2)$. The worst case is to check all the rest long cycles. We have $m_1 + \ldots + m_l \leq k$. Then $O(m_1^2) + O(m_2^2) + \ldots + O(m_l^2) < O((m_1 + m_2 + \ldots + m_l)^2) = O(k^2)$ is needed. Therefore, the total detection time each round including the detection of Double-Cycle-X-bridge is $O(k \cdot (k^2 + k^2))$, which is $O(k^3)$. In addition, it takes $O(k^2)$ to solve all bridges with at most $k$ operations. Therefore, it takes $GSB-I O(k^3 + k^2) = O(k^3)$ time to detect and solve all existing bridges.

6. Comparison Study

We now present some experimental study comparing GSB-I with the original GSB implementation. We compare both running time (and we expect GSB-I to be faster), and the computed distance. Since GSB-I is essentially the same as GSB except for using a faster algorithm, we expect the computed distances to be very close.

6.1. Experiment Design

We borrowed the methodology by Baders to generate random simulated data for the experiments. We first generate genome $A$ with $n = 200$ genes, then evolve $A$ to $B$ by applying a random sequence of $(l \cdot n/20)$ operations, for different values of $l = 1, 2, \ldots, 10$. For each value of $l$, we generate 50 different random instances. We used this generator to evolve the identity genome, $Id$, in a given number of steps by executing the five different operations (counting both left- and right-transreversal) in some given proportions, which use a mixture of the five operations. The proportions used for the six datasets are listed in Table 1. For example, we use the same proportion of the four operations to generate the original genome to sort in dataset1. Although these probabilities may not match the biological reality, this is still convenient to assess the performance of the two algorithms.

<table>
<thead>
<tr>
<th>Dataset index</th>
<th>rev</th>
<th>tp</th>
<th>ltr</th>
<th>rtr</th>
<th>bi</th>
<th>Dataset index</th>
<th>rev</th>
<th>tp</th>
<th>ltr</th>
<th>rtr</th>
<th>bi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Dataset 4</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Dataset 5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>Dataset 6</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

6.2. Experimental Results

Running Time Comparison: From the experimental results as shown in Fig 6, we conclude that the GSB-I algorithm runs much faster than the previous GSB algorithm in practice.
**Fig. 6.** Comparison of running time over 6 datasets between GSB and GSB-I

**Fig. 7.** Calculated distance comparison among GSB and GSB-I on Dataset1

**Calculated Distance Comparison:** We found that the result for datasets 1-6 are very similar. Thus we only show in Figure 7 the comparison of calculated distance for Dataset 1. The upper bound refers to the difference between the number of genes and the current number of cycles. The lower bound is the optimal solution, which is the half of the upper bound. The reason is that either one or two cycles are increased for each operation. The $dSIM$ refers to the number of operations conducted to create the original genomes to sort. From Figure 7, we can see that the lines for previous and improved GSB algorithms are always overlap. In fact, their average calculated distance for each dataset differs within 1. Therefore, we conclude that GSB-I has the same performance as previous GSB in calculated distance.

**7. Conclusion and Future Work**

We give an improved version of the 2-approximation algorithm, Genome Sorting by Bridges (GSB), for the problem of sorting signed permutation by reversal, transposition, transreversal, and block-interchange. We define an innovative new data structure called *canonical sequence* that helps transform breakpoint graph back into signed permutation and define patterns for all bridge structures, which uniquely identify each bridge from *canonical sequence*. With the two new tools, we identify bridge structures in $O(n^2)$. It allows the computation to be sped up by magnitudes (from $O(n^6)$ to $O(n^3)$), which allows calculating real genome distances possible.
An $O(n^3)$ Algorithm for Sorting Signed Genomes

One extension of this work is to decrease the approximation ratio for GSB-like algorithm. We conjecture that it can be decreased to 1.5, since in our experimental evaluation, we observed that the calculated distances were always below 1.5 times of lower bound. We believe that through more careful analysis of complex structures in breakpoint graph, we are able to decrease the approximation ratio to 1.5, as the work of Hartman and Sharan. 9.

8. Acknowledgments

This work was supported in part by the National University of Singapore under AcRF grant R252-000-461-112.

References

6. Hao F, Zhang M, Leong HW, A 2-approximation algorithm for sorting signed permutations by reversals, transpositions, transreversals, and block-interchanges, "Submitted for publication".