2008

Exact and Approximation Algorithms for Computing Reversal Distances in Genome Rearrangement

Euna Park
San Jose State University

Exact and Approximation Algorithms for Computing Reversal Distances in Genome Rearrangement

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Euna Park

Spring 2008

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Sami Khuri

Dr. Robert Chun

Mr. Frank Butt

APPROVED FOR THE UNIVERSITY

Acknowledgements

First of all I would like to express my gratitude to Dr. Sami Khuri of San Jose State University for providing the motivation, resources, and invaluable advice for me to accomplish this writing project.

I would also like to express my appreciation to Dr. Robert Chun of San Jose State University for his support, comments on my work, and valuable academic advice.

And special thanks go to Mr. Frank Butt not only for providing useful comments but also for encouraging and pushing me to accomplish this long academic goal.

I am also grateful for my co-workers, Alvin Cho, Amy Cho, and Frank Chan, who proofread this paper patiently taking the time out of their busy schedules.

Finally, I would like to dearly thank my husband, Sean, for his enduring patient and my two kids, Erin and Andrew, for understanding my busy schedule.

Abstract

Genome rearrangement is a research area capturing wide attention in molecular biology. The reversal distance problem is one of the most widely studied models of genome rearrangements in inferring the evolutionary relationship between two genomes at chromosome level. The problem of estimating reversal distance between two genomes is modeled as sorting by reversals. While the problem of sorting signed permutations can have polynomial time solutions, the problem of sorting unsigned permutations has been proven to be NP-hard [4]. This work introduces an exact greedy algorithm for sorting by reversals focusing on unsigned permutations. An improved method of producing cycle decompositions for a 3/2-approximation algorithm and the consideration of 3-cycles for reversal sequences are also presented in this paper.

TABLE OF CONTENTS:

LIST OF ALGORITHMS:

LIST OF FIGURES:

LIST OF TABLES:

1. Introduction

Gene ordering is changed by a genome rearrangement event and the genomic architecture of a species is altered by a series of genome rearrangements [1]. While the gene level evolution is measured by local mutations such as insertions, substitutions, and deletions, genetic evolution at the chromosome level is based on global rearrangement events. Reversals and translocations are the most common rearrangement events observed in mammalian genetic evolution. A reversal event changes the order of genes acting on the same chromosome while a translocation rearranges the genes swapping segments between two chromosomes.

Since Dobzhansky and Sturtevant introduced an evolutionary tree showing a scenario with 17 reversals for *Drosophila pseudoobscura* and *Miranda* in 1938, analysis through genome rearrangement by reversals is actively being used to infer the evolutionary relationship between two different species. In 1984 Nadeau and Taylor estimated that about 250 genomic rearrangement events have occurred between human and mice genomes in the process of divergence for approximately 80 million years [1]. In 1987, O'Brien showed that the only difference between the two most well-known bacteria, *Escherichia coli* and *Salmonella typhimurium*, is a reversal of a long substring of the genomic sequence in terms of gene orders [2].

Genome rearrangement by reversals alone has been considered a worthwhile study to understand evolutionary distance of different species at the chromosome level. Algorithmic study of genome rearrangement by reversals has been widely discussed since Watterson, Ewens, Hall, and Morgan introduced the first definition of the reversal distance problem in 1982 [3]. Genome rearrangement by reversals provides a good method of studying evolutionary history between two species. However, finding a series of rearrangements that transforms one genome into another is a challenging combinatorial problem. Estimating the reversal distance between two genomes has been modeled as sorting by reversals problem and the problem has been proven to be NP-hard by Caprara in 1997 [4].

In 1995 Kececioglu and Sankoff introduced the first approximation algorithm, which guarantees an approximation ratio of 2, for the problem by using a greedy technique [5]. Algorithmic approach for sorting by reversals problem has been further studied and a 7/4-approximation algorithm has been proposed by Bafna and Pevzner in 1996 [6]. In their paper, Bafna and Pevzner introduced the breakpoint graph and the cycle decomposition technique for the problem. They also showed sorting signed permutations can be solved easily compared to sorting unsigned permutations.

A signed permutation can be used to represent gene orders for a genome when the directions of genes are known. Directions of genes are determined by sequencing entire genomes, which requires significant amount of effort by biologists. Gene orders are also derived by comparative physical mapping, which most available experimental genome data are based on. However, physical maps do not provide correct information about gene directions, thus, unsigned permutations are used to present genomes in most cases. In this work sorting by reversals refers to sorting unsigned permutations unless it specifically mentions signed permutation cases.

In 1998 Christie proposed an improved approximation, which achieves a performance ratio of 3/2. So far the best known algorithm for sorting by reversals has an

approximation ratio of 1.375 [7]. However, Christie introduced a milestone approach of maximizing edge-disjoint cycles for cycle decompositions using the maximum matching technique.

This work describes an exact algorithm using the greedy technique and an improved method of finding the maximum number of edge-disjoint cycles for cycle decompositions. An improved method of finding a sequence of reversals for achieving lower reversal distances by using 3-cycles is also presented in this work. The problem and important terminologies are formally defined in the following section. Section 3 and Section 4 describe the greedy approach and the cycle decomposition approach, respectively. The exact greedy algorithm for the problem is presented in Section 5. In Section 6 a new graph, called the two-cycle graph, is introduced and an improved technique of generating edge-disjoint cycles using the two-cycle graph is presented. The consideration of 3-cycles for finding a series of reversals is described in Section 7. The running time of the approximation algorithm that uses the proposed methods is described in Section 8. In Section 9, the experimental results of the proposed algorithms are described. Some concluding remarks are followed in the last section.

2. Terminologies

A couple of milestone papers introduced important concepts used to srdrmtug dy sorting by reversals problem. Most terms were defined either by Kececioglu and Sankoff [5], or Bafna and Pevzner [6]. Some of those terms are described in what follows.

Given two permutations $\sigma = (\sigma_1\ \sigma_2\ \ldots\ \sigma_n)$ and $\tau = (\tau_1\ \tau_2\ \ldots\ \tau_n)$, which represent the order of n genes in two chromosomes, a reversal ρ on interval $[i, j]$ of the permutation ($1 \leq i < j \leq n+1$) transforms σ into the permutation,

$$\sigma \cdot \rho[i, j] = (\sigma_1\ \sigma_2\ \ldots\ \sigma_{i-1}\ \underline{\sigma_j\ \sigma_{j-1}\ \ldots\ \sigma_{i+1}\ \sigma_i}\ \sigma_{j+1}\ \ldots\ \sigma_n).$$

In essence, $\sigma \cdot \rho$ reverses genes $\sigma_i, \sigma_{i+1}, \ldots, \sigma_{j-1}, \sigma_j$. The reversal distance problem is to find a series of reversals $\rho_1, \rho_2, \ldots, \rho_t$ such that $\sigma \cdot \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_t = \tau$ and t is minimum. The minimum t denotes the reversal distance between σ and τ.

An ***inverse permutation*** is a permutation in which the value of elements and the position of elements are exchanged. In essence, an inverse permutation undoes the action of performing a permutation. Applying a permutation and then its inverse permutation is equivalent to applying the identity permutation, where the identity permutation is given by $\iota = (1\ 2\ \ldots\ n)$. The reversal distance between σ and τ is equal to the reversal distance between $\tau^{-1} \cdot \sigma$ and the identity permutation ι, where τ^{-1} is the inverse of τ that satisfies $\tau^{-1} \cdot \tau = \iota$. For example, consider

$\tau = (2\ 3\ 1\ 5\ 4)$,
where $\tau_1 = 2$, $\tau_2 = 3$, $\tau_3 = 1$, $\tau_4 = 5$, and $\tau_5 = 4$. Then,

$\tau^{-1} = (3\ 1\ 2\ 5\ 4)$,
where $\tau_1^{-1} = 3$, $\tau_2^{-1} = 1$, $\tau_3^{-1} = 2$, $\tau_4^{-1} = 5$, and $\tau_5^{-1} = 4$.

We can verify
$\tau^{-1} \cdot \tau_1 = \tau_2^{-1} = 1$
$\tau^{-1} \cdot \tau_2 = \tau_3^{-1} = 2$
$\tau^{-1} \cdot \tau_3 = \tau_1^{-1} = 3$
$\tau^{-1} \cdot \tau_4 = \tau_5^{-1} = 4$
$\tau^{-1} \cdot \tau_5 = \tau_4^{-1} = 5$,
which is $\tau^{-1} \cdot \tau = \iota$.

Now we have $\sigma = (5\ 2\ 4\ 1\ 3)$,
where $\sigma_1 = 5$, $\sigma_2 = 2$, $\sigma_3 = 4$, $\sigma_4 = 1$, and $\sigma_5 = 3$. Then,

$\tau^{-1} \cdot \sigma_1 = \tau_5^{-1} = 4$
$\tau^{-1} \cdot \sigma_2 = \tau_2^{-1} = 1$
$\tau^{-1} \cdot \sigma_3 = \tau_4^{-1} = 5$
$\tau^{-1} \cdot \sigma_4 = \tau_1^{-1} = 3$
$\tau^{-1} \cdot \sigma_5 = \tau_3^{-1} = 2$

We can see that transforming σ into τ is equivalent to transforming $\tau^{-1} \cdot \sigma$ into ι.

$\sigma \rightarrow \tau$	$\tau^{-1} \cdot \sigma \rightarrow \iota$
5 2 4 1 3	4 1 5 3 2
5 2 3 1 4	4 1 2 3 5
1 3 2 5 4	3 2 1 4 5
2 3 1 5 4	1 2 3 4 5

When we take $\pi = \tau^{-1} \cdot \sigma$ as an input permutation, the reversal distance problem can be modeled as the problem of finding the reversal distance between π and ι, and is called sorting by reversals problem. The reversal distance between π and ι is denoted by ***d(π)***.

A ***breakpoint*** of a permutation π is defined as a pair of adjacent positions in its identity permutation but not adjacent in π. In other words, $(i, i+1)$ forms a breakpoint if $|\pi_{i+1} - \pi_i| \neq 1$, where $0 \leq i \leq n$. A pair of consecutive elements π_i and π_{i+1}, $0 \leq i \leq n$, is called an ***adjacency*** if $|\pi_{i+1} - \pi_i| = 1$. The value 0 for π_0 and the value $n + 1$ for π_{n+1} are added to handle the boundaries of permutation π. (0, 1) forms a breakpoint if $\pi_1 \neq 1$ and $(n, n+1)$ forms a breakpoint if $\pi_n \neq n$. Thus, the maximum number of breakpoints, which is denoted by ***b(π)***, is $n + 1$ for the permutation of size n and $b(\iota) = 0$, for the identity permutation ι. Figure 1 shows breakpoints of a permutation $\pi = 2\ 3\ 1\ 6\ 5\ 4$.

0 • 2 3 • 1 • 6 5 4 • 7f

Figure 1. Breakpoints of π = 2 3 1 6 5

A ***strip*** is defined as a sequence of consecutive positions surrounded by breakpoints but has no internal breakpoints. In other words, an interval [*i, j*] is a strip if (*i*-1, *i*) and (*j*, *j*+1) are breakpoints and there is no breakpoint between them. A strip is defined as an ***increasing strip*** if the elements in the strip are in increasing order or as a ***decreasing strip*** if those are in decreasing order. From Figure 1, [2 3] is an increasing strip and [6 5 4] is a decreasing strip. A single element strip, except for π_0 and π_{n+1}, can be considered as either an increasing strip or a decreasing strip. Strips that π_0 and π_{n+1} belong to are always considered as increasing strips.

A ***breakpoint graph*** G(π), which is an edge colored graph derived from permutation π, is defined with *n*+2 vertices, 0, 1,…, *n*, *n*+1, by connecting two vertices with a black edge if the two vertices represent two elements that form a breakpoint and with a gray edge if the two vertices represent two elements that are adjacent in the identity permutation but not adjacent in π. In other words, two vertices, *i* and *j*, are connected by a black edge if $|i - j| = 1$ but $|\pi_j - \pi_i| \neq 1$ and by a gray edge if $|\pi_j - \pi_i| = 1$ but $|i - j| \neq 1$, where $0 \leq i, j \leq n+1$.

Figure 2 shows a breakpoint graph of G(π), for π = 3 1 5 2 6 4. In Figure 2 thick black lines represent black edges and dotted lines represent grey edges.

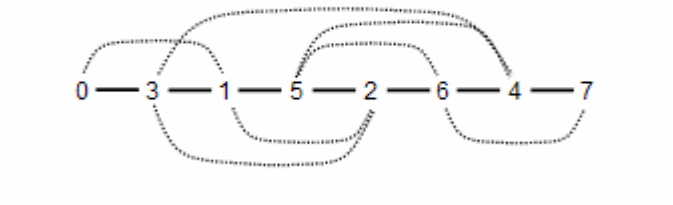


Figure 2. A breakpoint graph G(π) of π = 3 1 5 2 6 4

A ***cycle*** in a graph G(*V, E*) is defined for a sequence of vertices, $v_1 v_2 \ldots v_m = v_1$, if $(v_i, v_{i+1}) \in E$ *for* $1 \leq i \leq m - 1$. In an edge colored graph, a cycle is called ***alternating*** if the colors of every two consecutive edges of the cycle are distinct. A ***balanced vertex*** *v* in an edge colored graph is a vertex in which the colors are equally distributed among the edges incident to it. A graph in which every vertex is balanced is called a ***balanced graph***.

The breakpoint graph G(π) is a balanced graph since every vertex has the same number of black edges and gray edges. A balanced graph has at least one alternating Eulerian cycle in every connected component, where an ***Eulerian cycle*** is a cycle that visits every edge exactly once and returns to the starting point. Therefore, a breakpoint graph G(π) contains one or more alternating Eulerian cycles. Figure 3 shows two Eulerian cycles in the breakpoint graph G(π) from Figure 2.

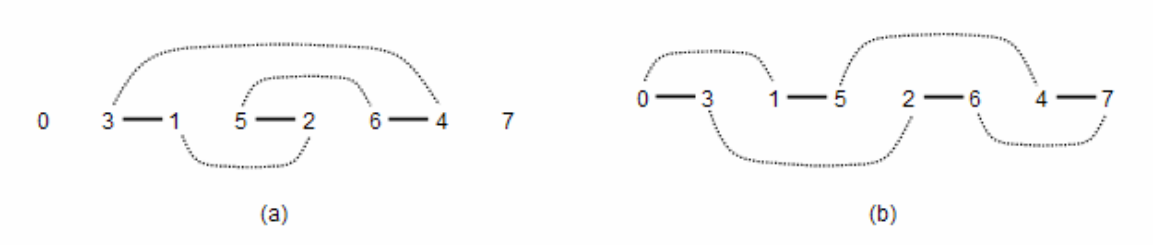


Figure 3. A cycle decomposition of G(π)

A balanced edge colored graph can be decomposed into cycles which are edge-disjoint from one another. Thus, an edge disjoint cycle decomposition can be found for a breakpoint G(π). However, there are many different cycle decompositions for the same G(π). Figure 4 shows another cycle decomposition for the G(π) from Figure 2.

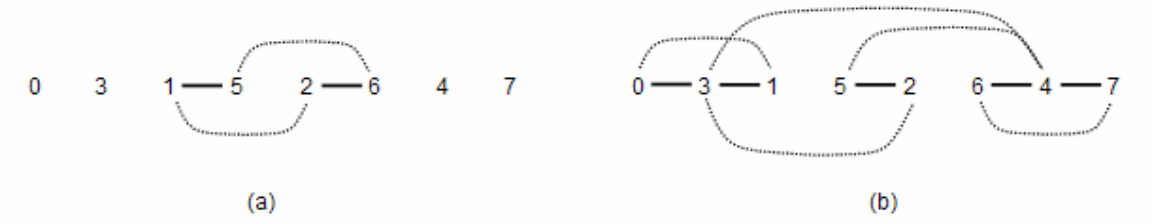


Figure 4. Another cycle decomposition of G(π)

c(π) denotes the maximum number of cycles derived from a cycle decomposition of G(π). The size of a cycle is determined by the number of black edges that belongs to it. An *l*-cycle denotes a cycle with *l* black edges. A cycle is called ***oriented*** if the cycle contains a reversal that removes one or two breakpoints. Otherwise, the cycle is called ***unoriented***. A reversal that removes *k* breakpoints is denoted by ***k-reversal***, where $k \in \{ 0, 1, 2 \}$.

3. Greedy Approach for 2-Approximation Algorithm

The first serious approach for sorting by reversals problem was introduced by Kececioglu and Sankoff in 1995 [5]. They found a greedy algorithm, which provides a performance ratio of 2 for computing reversal distances. The greedy algorithm is based on two concepts of gene permutation: breakpoint and strip.

As seen in Section 2, a breakpoint of a permutation π is an adjacent pair of elements in its identity permutation but not adjacent in π. The identity permutation ι is the only permutation whose number of breakpoints is zero. Solving sorting by reversals aims to decrease $b(\pi)$ to zero. A reversal changes only two breakpoints at the end of the interval. In other words, a reversal ρ on interval $[i, j]$ affects only two positions, $(i\text{-}1, i)$ and $(j, j\text{+}1)$, and removes at most 2 breakpoints.

$b(\pi) - b(\pi \cdot \rho) \in \{0, 1, 2\}$

Thus, the greedy strategy is to choose a reversal that decreases $b(\pi)$ by the largest number in the current permutation π.

Kececioglu and Sankoff focused on decreasing strips for choosing a reversal that reduces the number of breakpoints based on Theorem 1.

Theorem 1 [5]
If a permutation π contains a decreasing strip, then there is a reversal ρ that decreases the number of breakpoints in π, that is, $b(\pi \cdot \rho) < b(\pi)$.

A naive greedy algorithm for sorting by reversals can be implemented in the following way, which achieves a performance guarantee of 4.

Let π be a permutation and ρ be a reversal operation.

SORTINGBYREVERSALS4(π)

```
1  while b(π) > 0
2    if π has a decreasing strip
3      Among all reversals, choose reversal ρ minimizing b(π • ρ)
4    else
5      Choose a reversal ρ that flips an increasing strip in π
6    π = π • ρ
7  return π
```

Algorithm 1. 4-approximation algorithm [1]

The greedy algorithm tries to reduce the number of breakpoints. The worst case occurs when there are no decreasing strips in the current permutation. In that case, one reversal of any increasing strip is taken to produce a decreasing strip even though it does not remove any breakpoint. Thus, two reversals are required to remove at least one breakpoint when there is no decreasing strip remaining in the current permutation. In other words, *SORTINGBYREVERSALS4(π)* removes at least one breakpoint in every step if the current permutation has a decreasing strip and the algorithm eliminates at least one breakpoint in every two steps if the permutation has no decreasing strips. Thus, *SORTINGBYREVERSALS4(π)* performs the permutation sorting in at most $2b(\pi)$. The approximation ratio is at most $2b(\pi)/d(\pi)$, where $d(\pi)$ refers to the minimal number of reversals. Considering that at most two breakpoints are eliminated by a reversal, $d(\pi) \geq b(\pi)/2$ can be the lower bound of reversal distance. Therefore, *SORTINGBYREVERSALS4(π)* guarantees a performance ratio of 4 since $2b(\pi) / [b(\pi)/2] = 4$ [1].

Kececioglu and Sankoff introduced an important theorem which improves the performance ratio of the greedy algorithm.

Theorem 2 [5]
Suppose π is a permutation with a decreasing strip. If all reversals that remove breakpoints from π leave no decreasing strips, then there is a reversal that removes two breakpoints from π.

This theorem was nicely proven in [5]. Based on Theorem 2 the greedy approach can achieve a guaranteed approximation ratio of 2.

SORTINGBYREVERSALS2(π)

```
1  while b(π) > 0
2   if π has a decreasing strip
3      k = the smallest label in a decreasing strip
4      ρ = the reversal that cuts after k and after k-1
5      if π • ρ has no decreasing strip
6           l  = the largest label in a decreasing strip
7           ρ = the reversal that cuts before l and before l+1
8   else
9     ρ = the reversal that cuts the first two breakpoints
10  π = π • ρ
11 return π
```

Algorithm 2. 2-approximation algorithm [8]

SORTINGBYREVERSALS2(π), shown in Algorithm 2, chooses a reversal which reduces two breakpoints if there is no decreasing strip that remains after the reversal. The elimination of two breakpoints compensates the case of the permutation with no decreasing strip, which requires two reversal steps for removing at least one breakpoint, after the reversal. Thus, *SORTINGBYREVERSALS2(π)* requires at most $b(\pi)$ reversals to sort the permutation π into its identity permutation guaranteeing a performance ratio of 2.

4. Cycle Decomposition Approach for 3/2-Approximation

Bafna and Pevzner demonstrated sorting by reversals of signed permutations can be solved with guaranteed error bound 3/2 [6]. They also proposed a 7/4-approximation algorithm for unsigned permutations [6]. In 1998, Christie showed the performance ratio of 3/2 can be achieved for computing the reversal distance of unsigned permutations [11]. The 3/2-approximation algorithm for signed permutations and Christie's 3/2-approximation algorithm are described in this section.

Bafna and Pevzner first introduced a new lower bound of the reversal distance by showing that there is a strong relationship between the reversal distance of a permutation π and the number of cycles in a maximum cycle decomposition of the breakpoint graph $G(\pi)$. Theorem 3 was proven in [6] and Theorem 4 is derived from Theorem 3.

Theorem 3 [6]
Let ρ be an arbitrary reversal of π for a given breakpoint graph G(π).
Then, $\wedge b(\pi, \rho) - \wedge c(\pi, \rho) \leq 1$ for every permutation π and reversal ρ, where $\wedge$b(π, ρ) is the decrease in the number of breakpoints and $\wedge$c(π, ρ) is the decrease of the number of cycles in a maximum decomposition by the reversal ρ.

Theorem 4 [6]
$d(\pi) \geq b(\pi) - c(\pi)$ for every permutation π.

Suppose that $\rho_t \rho_{t-1} \ldots \rho_1$ is a shortest series of reversals that transforms π into the identity permutation π_0. Let $\pi_{i-1} = \pi_i \rho_i$, where $i = 1, 2, \ldots, t$. By applying Theorem3 we get

$$\begin{aligned} d(\pi_i) &= d(\pi_{i-1}) + 1 \\ &\geq d(\pi_{i-1}) + \wedge b(\pi_i, \rho_i) - \wedge c(\pi_i, \rho_i) \\ &= d(\pi_{i-1}) + (b(\pi_i) - b(\pi_{i-1})) - (c(\pi_i) - c(\pi_{i-1})) \quad [6] \end{aligned}$$

Since $d(\pi_0) = b(\pi_0) = c(\pi_0) = 0$, $d(\pi_i) - (b(\pi_i) - c(\pi_i)) \geq d(\pi_{i-1}) - (b(\pi_{i-1}) - c(\pi_{i-1})) \geq \ldots \geq d(\pi_0) - (b(\pi_0) - c(\pi_0)) = 0$. Therefore, $d(\pi) \geq b(\pi) - c(\pi)$.

Theorem 4 offers a tighter lower bound than $d(\pi) \geq b(\pi)/2$ in terms of the number of cycles in a maximum cycle decomposition. 2-approximation algorithm by Kecceciogl u and Sankoff was based on the lower bound $d(\pi) \geq b(\pi)/2$. Bafna and Pevzner proposed an improved approximation algorithm based on the new lower bound, $d(\pi) \geq b(\pi) - c(\pi)$. Rather than finding a maximum cycle decomposition of π, they focused on finding a cycle decomposition with the maximum number of 2-cycles.

The lower bound, $d(\pi) \geq b(\pi) - c(\pi)$, can be described in terms of the number of 2-cycles in a cycle decomposition of G(π) as follows.

$$\begin{aligned} d(\pi) &\geq b(\pi) - c_2(\pi) - (c(\pi) - c_2(\pi)) \\ &\geq b(\pi) - c_2(\pi) - 1/3\ (b(\pi) - 2c_2(\pi)) \\ &= 2/3\ b(\pi) - 1/3\ c_2(\pi) \quad [11] \end{aligned}$$

where $c_2(\pi)$ is the number of 2-cycles and $1/3\ (b(\pi) - 2c_2(\pi))$ is the maximum number of cycles of greater than 2 black edges in a permutation π.

Bafna and Pevzner showed a signed permutation can be sorted in at most $b(\pi) - 1/2\ c_2(\pi)$ steps, achieving the performance ratios of 3/2. For the problem of sorting signed permutations, they defined a transformation of a signed permutation π into an unsigned permutation by substituting $-i$ with $2i$, $2i$-1 and $+i$ with $2i$-1, $2i$. The breakpoint graph for the unsigned permutation π' which transformed from a signed permutation π is defined in the same way. Figure 5 shows the transformation of a singed permutation π = +4 -1 +3 -2 into π' = 7 8 2 1 5 6 4 3.

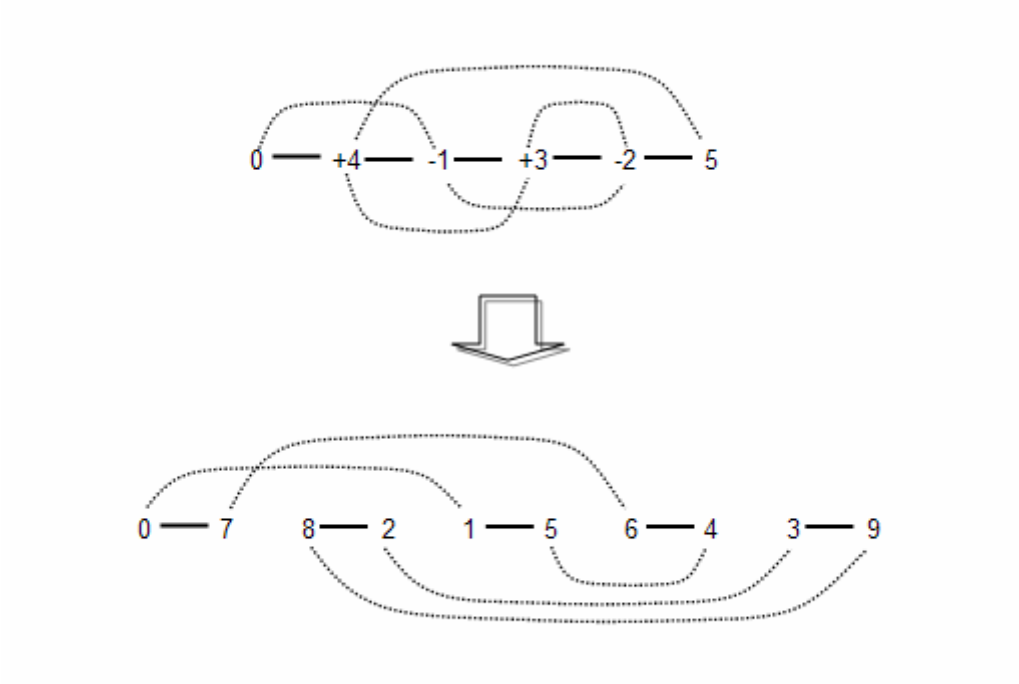


Figure 5. Breakpoint graphs G(π) and G(π')

Given a cycle decomposition of G(π), a reversal on a cycle is the reversal ρ = [*i* , *j*], where (*i-1*, *i*) and (*j*, *j+1*) are two breakpoints that belong to the same cycle. A cycle is considered oriented when there is a 1 or 2-reversal on it. A cycle is considered unoriented when there are no reversals that eliminate a breakpoint. Two cycles are called ***crossing*** when one or more black edges of one cycle interleaves a black edge of the other in the cycle decomposition. In the breakpoint graph G(π') from Figure 5 the cycle [0 7 6 4 5 1] and the cycle [8 2 3 9] are crossing.

Since each element in the transformed unsigned permutation π' always contains an adjacency on one side, every vertex of G(π') has the maximum edge degree of 2. When the maximum edge degree of each vertex is 2, there is no edge sharing between any two cycles, therefore, there exists a unique cycle decomposition for G(π'). To show that a permutation π' can be sorted in less than $b(\pi)$ steps, 2-reversals that do no have to be compensated against 0-reversals need to be observed.

In the breakpoint graph a 2-reversal removes a 2-cycle while a 1-reversal shrinks a longer cycle, thus, 2-reversals correspond to 2-cycle eliminations. However, 2-cycles are not always oriented. When a 2-cycle is unoriented, there is no reversal that leads to the elimination of the cycle. However, Bafna and Pevzner noted that any unoriented 2-cycle has a crossing cycle and there exists a reversal on its crossing cycle that can orient the 2-cycle.

Suppose a permutation π has an unoriented 2-cycle $[x\ y\ y'\ x']$ in its breakpoint graph G(π) as shown in Figure 6.

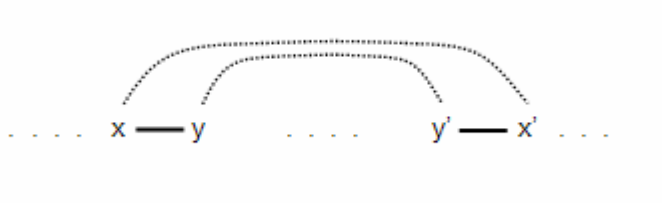


Figure 6. An unoriented 2-cycle in G(π)

Since $(x\ x')$ and $(y\ y')$ form gray edges, $(y\ y')$ cannot be adjacent and at least one black edge exists between y and y'. $[x\ y\ y'\ x']$ forms a 2-cycle, thus, the black edge between y and y' belongs to another cycle. If the black edge between y and y' does not form a cycle with a black edge which interleaves with $(x\ y)$ or $(y'\ x')$, the cycle $[x\ x'\ y'\ y]$ remains isolated, which means there is no way of bringing x and x', y and y' together. Having an isolated cycle cannot be a situation in sorting by reversals problem, therefore, an unoriented 2-cycle has at least one crossing cycle. A reversal on the black edge between y and y' and a black edge which interleaves $(x\ y)$ or $(y'\ x')$ orients the cycle $[x\ y\ y'\ x']$ since it switches the position of x and y or y' and x'.

From Figure 5, [8 2 3 9] is an unoriented 2-cycle. A reversal taking from 7 to 1 on its crossing cycle orients the unoriented cycle as shown in Figure 7.

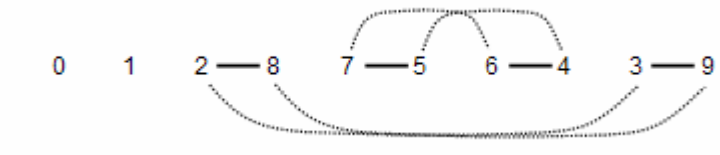


Figure 7. Breakpoint graph after a reversal [7, 1] from Figure 5

Bafna and Pevzner introduced the 3/2-approximation algorithm, shown in Algorithm 3, for sorting the unsigned permutation transformed from a signed permutation.

Bafna and Pevzner considered a round in the sorting by reversals as a series of reversals until only 0-reversals remain, which means there are no decreasing strips in the current permutation π'. If there is a 2-cycle in G(π') at the beginning of any round, there are at least two 2-reversals in that round since any round that starts with a 0-reversal is followed by a 2-reversal based on Theorem 2. Since the number of 2-cycles decreases only by 2-reversals and at most half of the 2-reversals need to be compensated against 0-reversals, a permutation π' can be sorted in at most $b(\pi') - 1/2c_2(\pi')$ achieving an approximation ratio of 3/2.

SORTINGSIGNEDBYREVERSALS3OVER2(π')

1 ***while*** *b(π') > 0*
2 ***if*** *π' has a decreasing strip*
3 *ρ = SORTINGBYREVERSALS2(π')* *(see Algorithm 2)*
4 ***else***

5 ***if*** *any 2-cycle C remains in G(π')*
6 *find a cycle C' that crosses C and take a 0-reversal on C' to orient C*
7 *ρ = The 2-reversal on the 2-cycle C*
8 ***else***
9 *ρ = any 0-reversal*
10 *π' = π' • ρ*
11 ***return*** *π*

Algorithm 3. 3/2-approximation algorithm for signed permutations [6]

Christie showed that even unsigned permutations can be sorted in at most $b(\pi) - 1/2\, c_2(\pi)$ reversal steps in [11]. He also focused on 2-cycles of a cycle decomposition of a breakpoint graph G(π). However, every vertex in G(π) of a unsigned permutation π has the maximum edge degree of 4. A unique cycle decomposition cannot be guaranteed when the maximum edge degree of each vertex is greater than 2. There are multiple cycle decompositions since one cycle can have common black edges or gray edges with other cycles.

Christie introduced a matching graph F(π) of a permutation π for a cycle decomposition of a breakpoint graph G(π). Each vertex in F(π) represents each black edge in G(π). Two vertices are connected if the two vertices represent the black edges that form a 2-cycle in G(π). Each edge in F(π), thus, represents a 2-cycle in G(π).

Given a graph $G = (V, E)$, a set of edges which share no common vertex is called a matching M in G. A maximum cardinality matching M of F(π) offers a set of 2-cycles formed by disjoint black edges. However, 2-cycles represented by M are not always edge-disjoint in terms of both black and gray edges. While M has no common black edges since M cannot be a matching if it has a common vertex in F(π), M may have common gray edges [11]. Figure 8 shows a gray edge shared by two 2-cycles in the breakpoint graph G(π), π = 9 3 4 6 5 8 7 1 10 2. In G(π) from Figure 8 two 2-cycles, [0, 1, 10, 9] and [9, 10, 2, 3] have a common edge (9, 10) [11].

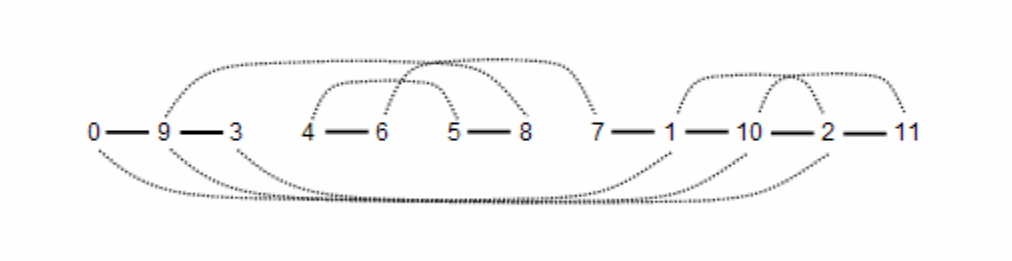
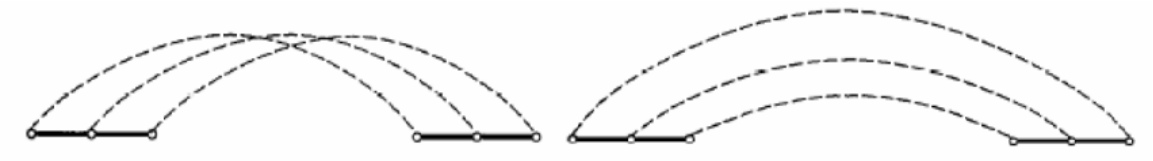


Figure 8. Breakpoint graph G(π) [11]

Thus, the M of F(π), which offers the cycles [0, 1, 10, 9] and [9, 10, 2, 3], does not represent complete edge-disjoint 2-cycles.

Christie introduced another graph, called the ladder graph L(M), which is derived from M of F(π). Each vertex in a ladder graph L(M) represents a 2-cycle from a matching M. Isolated vertices in L(M) represent independent 2-cycles and vertices are connected by an edge if 2-cycles have a common gray edge. The maximum edge degree of each

vertex of L(M) is 2 since a 2-cycle contains two grey edges and each gray edge can be shared by two different 2-cycles. Christie pointed out two 2-cycles that share a gray edge are either all oriented or all unoriented as shown in Figure 9. Figure 9(a) shows all oriented 2-cycles and Figure 9(b) shows all unoriented 2-cycles in a breakpoint graph $G(\pi)$.



(a) (b)

Figure 9. Two 2-cycles which share a gray edge in a breakpoint graph $G(\pi)$ [11]

The edge connected vertices are called ladder vertices. A ladder is defined when a collection of vertices line up forming a path in L(M). The collections of 2-cycles shown in Figure 9 are two types of the simplest forms of a ladder. Gray edge disjoint 2-cycles can be simply determined by selecting every other vertex from a ladder. Thus, $\lceil m/2 \rceil$ 2-cycles can be chosen for reversals among m ladder vertices from L(M).

Christie introduced the following propositions in [11]. Proofs of the propositions are not described in this work. In what follows a component of C represents a collection of cycles which interleave one another.

Proposition 1 [11]

If A is an oriented component of $\mathcal{C}$ which includes gray edges originating from k different cycles of $G(\pi)$, there is a series of elimination for A in k 2-reversals and all the other 1-reversals.

Proposition 2[11]

If A is an unoriented component of $\mathcal{C}$ which includes gray edges originating from k different cycles of $G(\pi)$, there is a series of elimination for A in one 0-reversals and k 2-reversals with all the other 1-reversals.

Based on the above propositions, Christie showed unsigned permutations can be sorted at most $b(\pi) - 1/2c_2(\pi)$ reversal steps as described below. [11]

When L(M) has x ladder vertices and y isolated vertices, in other words, $|M| = x + y$, there exists a cycle decomposition $\mathcal{C}$ of $G(\pi)$ which includes at least $\lceil x/2 \rceil$ 2-cycles from ladders and y isolated 2-cycles.

Let k be the number of oriented 2-cycles selected in $\mathcal{C}$. Then, there are at least k 2-reversals and no 0-reversals since there is a series of reversals that removes the oriented 2-cycles based on Proposition 1.

Let m be the number of unoriented components of $\mathcal{C}$ that contains gray edges representing the 2-cycles which are not selected and also vertices representing l selected

2-cycles. Then there is a series of reversals that contains at least $l + m$ 2-reversals and m 0-reversals to eliminate m unoriented components based on Proposition 2.

Let n be the number of unoriented components of $\mathcal{C}$ that contains gray edges representing p independent selected 2-cycles. Then there is a series of reversals that includes at least p 2-reversals and n 0-reversals according to Proposition 2. We have $n \leq \lfloor y/2 \rfloor$ since every component represents at least two cycles, where y is the number of independent 2-cycles in L(M).

Besides 1-reversals we can observe $(k + l + p + m)$ 2-reversals and $(m + n)$ 0-reversals to sort a permutation π. This derives that a permutation π can be sorted in no more than $b(\pi) - (k + l + p + m) + (m + n)$ reversals. Since $k + l + p \geq \lceil x/2 \rceil + y$, $k + l + p - n \geq \lceil x/2 \rceil + y - n$, and $n \leq \lfloor y/2 \rfloor$, we have $\lceil x/2 \rceil + y - n \geq \lceil y/2 \rceil + \lceil x/2 \rceil$. Furthermore $x + y = |M|$ and $|M| \geq c_2(\pi)$, therefore, $\lceil y/2 \rceil + \lceil x/2 \rceil \geq c_2(\pi)/2$, which means $(k + l + p - n) \geq c_2(\pi)/2$. This shows that no more than $b(\pi) - 1/2c_2(\pi)$ reversals are required to sort a permutation π [11].

Algorithm 4 shows the main steps of sorting a permutation π, which guarantees an approximation ratio of 3/2.

SORTINGBYREVERSALS3OVER2MATCHING(π)

1 construct the breakpoint graph G(π) of π
2 construct the matching graph F(π) of G(π)
3 find a matching M for the F(π)
4 construct the ladder graph L(M) from M
5 find a cycle decomposition $\mathcal{C}$ that includes edge-disjoint 2-cycles using L(M)
6 find an elimination sequence of $\mathcal{C}$

Algorithm 4. 3/2-approximation algorithm for sorting by reversals [11]

5. Exact Greedy Algorithm

The greedy algorithm that achieves 2-approximation is described in Section 2. The greedy strategy can be applied for an optimal solution for sorting by reversals. An exact algorithm using the greedy approach is proposed in this section.

Since sorting by reversals is a combinatorial problem, there are $\binom{n}{2}$ reversals to consider. The exact greedy algorithm finds an optimal solution by choosing a reversal which eliminates breakpoints by the largest number in each reversal step. For the algorithm all possible reversals on two breakpoints in a permutation π are observed. In other words, it considers $\binom{b}{2}$ reversals for an initial reversal, where b is $b(\pi)$ for a permutation π. A reversal ρ on two breakpoints can decrease $b(\pi)$ by at most 2. The reversal does not increase $b(\pi)$ since it affects only the two breakpoint positions in the current permutation π. In other words, any reversal that results in the following can be a candidate.

$$b(\pi) - b(\pi \cdot \rho) \in \{0, 1, 2\}$$

The algorithm chooses all reversals which result in the same $b(\pi)$, the lowest value of $b(\pi)$, after every reversal step. The result permutations are kept as candidates for the next reversal step. The same greedy strategy is continued until a result permutation with no breakpoints is achieved. A sequence of reversals that results in the identity permutation offers an optimal solution for sorting by reversals.

The exact greedy algorithm is motivated by the fact that the largest decrease of $b(\pi)$ in each reversal step drives $b(\pi)$ to zero in the minimal steps.

Let $\wedge b(\pi, \rho_{ai})$, $\wedge b(\pi, \rho_{bi})$, and $\wedge b(\pi, \rho_{ci})$ be the decreases of b(π) after i-th reversal ρ_a, ρ_b, and ρ_c respectively, where $1 \leq i \leq$ t for the reversal distance t. Let the reversal ρ_a be a sequence of reversals that decreases $b(\pi)$ by the largest number in each reversal step. Let the reversal ρ_b and ρ_c be two arbitrary reversals that decrease $b(\pi)$ but not necessarily by the largest number.

In other words,

$$\wedge b(\pi, \rho_{a1}) \geq \wedge b(\pi, \rho_{b1}) \geq \wedge b(\pi, \rho_{c1}),$$
$$\wedge b(\pi, \rho_{a2}) \geq \wedge b(\pi, \rho_{b2}) \geq \wedge b(\pi, \rho_{c2}),$$
$$\wedge b(\pi, \rho_{a3}) \geq \wedge b(\pi, \rho_{b3}) \geq \wedge b(\pi, \rho_{c3}),$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots..$$
$$\wedge b(\pi, \rho_{at}) \geq \wedge b(\pi, \rho_{bt}) \geq \wedge b(\pi, \rho_{ct}),$$

then,

$$\wedge b(\pi, \rho_{a1}) + \wedge b(\pi, \rho_{a2}) + \ldots. + \wedge b(\pi, \rho_{at}) \geq$$
$$\wedge b(\pi, \rho_{b1}) + \wedge b(\pi, \rho_{b2}) + \ldots. + \wedge b(\pi, \rho_{bt}) \geq$$
$$\wedge b(\pi, \rho_{c1}) + \wedge b(\pi, \rho_{c2}) + \ldots. + \wedge b(\pi, \rho_{ct}).$$

When $b(\pi) = \wedge b(\pi, \rho_{a1}) + \wedge b(\pi, \rho_{a2}) + \ldots. + \wedge b(\pi, \rho_{at})$, the number of reversals t through the reversals of ρ_a guarantees an optimal reversal distance while either the reversals ρ_b and ρ_b do not guarantee the minimal t. When $b(\pi) = \wedge b(\pi, \rho_{b1}) + \wedge b(\pi, \rho_{b2}) + \ldots. + \wedge b(\pi, \rho_{bt})$ or $b(\pi) = \wedge b(\pi, \rho_{c1}) + \wedge b(\pi, \rho_{c2}) + \ldots. + \wedge b(\pi, \rho_{ct})$, we might get a reversal distance which is smaller than t through the reversals of ρ_a. Thus, the optimal reversal distance $d(\pi)$ of a permutation π can be achieved by choosing a reversal which decreases $b(\pi)$ by the largest number in every reversal step.

Algorithm 5 shows the exact greedy algorithm.

SORTINGBYREVERSALSEXACT(π)

1 takes all possible reversals ρ_t on two breakpoints
2 ρ_t = reversals that result the smallest $b(\pi)$
3 $\pi_t = \pi \bullet \rho_t$
4 ***while*** *$b(\pi_t) > 0$*
5 ρ_t = reversals that result the smallest $b(\pi_t)$
6 $\pi_t = \pi_t \bullet \rho_t$

Algorithm 5**.** Exact greedy algorithm

The visual explanation of Algorithm 5 is shown in Figure 10. In Figure 10 the square block on the top represents an initial permutation π and the lines inside the block show breakpoints in π. Square blocks after the top block represent permutations resulted by possible reversals from a previous step. Colored square blocks are shown as the result permutations chosen by the greedy strategy. Colored square blocks with no lines at the bottom show the identity permutations, which can be more than one solutions for the permutation. Any reversal path generates the identity permutation at the bottom offers an optimal reversal distance $d(\pi)$.

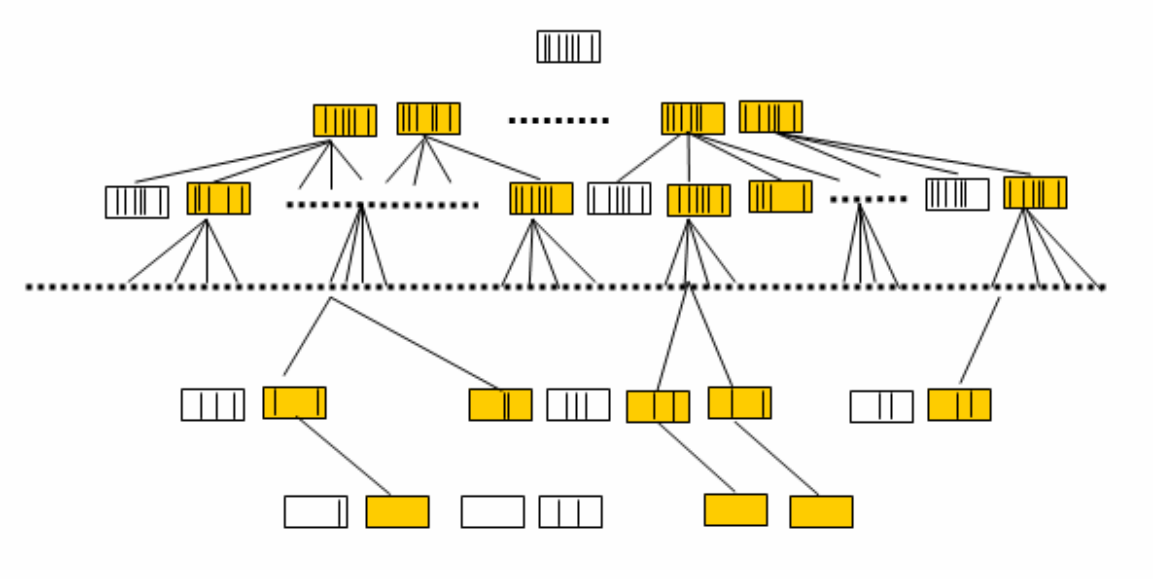


Figure 10. Exact greedy algorithm path

Even though the greedy strategy achieves an optimal solution, the exact algorithm requires keeping intermediate result permutations to determine the next best reversal. The computational resource required by the algorithm increases as it approaches to middle steps and decreases towards the end of the reversals.

6. Cycle Decomposition using Vertex Cover

The 3/2-approximation algorithms for sorting by reversals for signed permutations and unsigned permutations are described in Section 3. Both approximation algorithms focus on 2-cycles for a cycle decomposition of $G(\pi)$ to maximize 2-reversals. Maximizing the number of 2-cycles for a cycle decomposition of $G(\pi)$ for unsigned permutations requires finding a maximum edge-disjoint 2-cycles. Christie proposed a method of using a maximum matching M of $F(\pi)$ and ladder graph L(M) as described in Section 3. This section introduces a new graph, called two-cycle graph, and an improved method of generating complete edge-disjoint 2-cycles using the two-cycle graph to maximize the number of 2-reversals.

A maximum matching M of matching graph F(π) offers 2-cycles which do not have common black edges with one another. The 2-cycles from a maximum matching M of F(π), however, does not guarantee edge-disjoint 2-cycles since the 2-cycles may have common gray edges. Ladder graph L(M) from the maximum matching M is used to find complete edge-disjoint 2-cycles.

Figure 11 shows the matching graph F(π) for π = 9 3 4 6 5 8 7 1 10 2. F(π) can be easily built from G(π), which is shown in Figure 8.

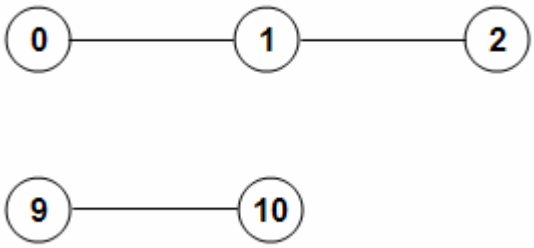


Figure 11. Matching Graph F(π) for π = 9 3 4 6 5 8 7 1 10 2

Vertices of F(π) are labeled in the left elements of black edges from the breakpoint graph G(π). The black edge belonging to element 0 forms a 2-cycle with the black edge belonging to 1 as shown in Figure 8. Thus, vertex 0 is connected to vertex 1 by an edge in F(π). Vertex 1 and vertex 9 are connected to vertex 2 and vertex 10 respectively since those black edges form 2-cycles in G(π). As Figure 11 shows, the black edge represented by vertex 1 in F(π) is shared by two 2-cycles. Therefore, a maximum matching M chooses only one 2-cycle formed by the black edge represented by vertex 1 since a matching cannot have two edges which have a common vertex. Figure 12 shows a maximum matching M of F(π).

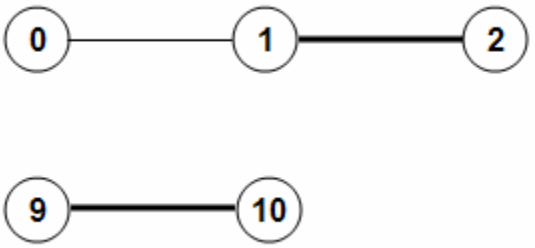


Figure 12. A maximum matching M of F(π) for π = 9 3 4 6 5 8 7 1 10 2

The maximum matching M, shown in Figure 12, chooses 2-cycle [9 10 2 3] and [1 2 11 10] from Figure 8. A ladder graph L(M) built from the maximum matching M is shown in Figure 13. Each vertex of L(M) represents a 2-cycle chosen by the maximum matching M and labeled by the left most element belonging to each 2-cycle.

Figure 13. Ladder graph L(M) from M in Figure 12

Since two 2-cycles in Figure 13 do not share a gray edge, two vertices in L(M) are not edge connected. Therefore, both 2-cycles are chosen for reversals. Two edge-disjoint 2-cycles, which is the maximum number of edge-disjoint 2-cycles for a permutation π = 9 3 4 6 5 8 7 1 10 2, are selected through a maximum matching M of F(π) followed by the ladder vertex determination from L(M). However, the method of using a maximum matching M of F(π) and ladder graph L(M) does not guarantee the maximum number of edge-disjoint 2-cycles.

Figure 14 shows another maximum matching M of F(π) from Figure 11.

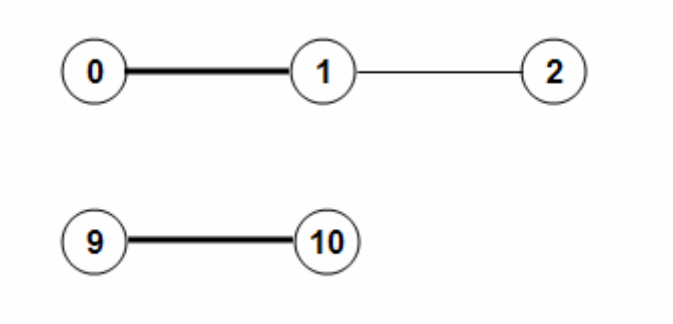


Figure 14. Another maximum matching M of F(π) for π = 9 3 4 6 5 8 7 1 10 2

The maximum matching M of F(π) shown in Figure 14 also offers two black edge disjoint 2-cycles. However, the two 2-cycles form a ladder in the ladder graph L(M) from the M as shown in Figure 15.

(a) Choosing 2-cycle [0 1 10 9] (b) Choosing 2-cycle [9 10 2 3]

Figure 15. Ladder graph L(M) from M in Figure 14

Since 2-cycle [0 1 10 9] and [9 10 2 3] have a common gray edge as shown in G(π) from Figure 8, both of the 2-cycles cannot be selected for reversals. Thus, only one 2-cycle, either [0 1 10 9] or [9 10 2 3], can be used for 2-reversals. This does not maximize 2-reversals for sorting by reversals.

A maximum matching M of F(π) does not guarantee the best matching M of F(π) for L(M) since information of gray edge sharing is unknown in F(π). Finding the

maximum number of edge-disjoint 2-cycles can be improved by considering both black edge sharing and gray edge sharing in the same selection step.

A two-cycle graph T(π) is introduced for the improved method of maximizing edge-disjoint 2-cycles for cycle decompositions of G(π). Each vertex in T(π) represents a 2-cycle in G(π). Two vertices are connected by an edge if the 2-cycles represented by the vertices have either a common black edge or a common gray edge. Figure 16 shows the two-cycle graph T(π) from G(π) in Figure 8.

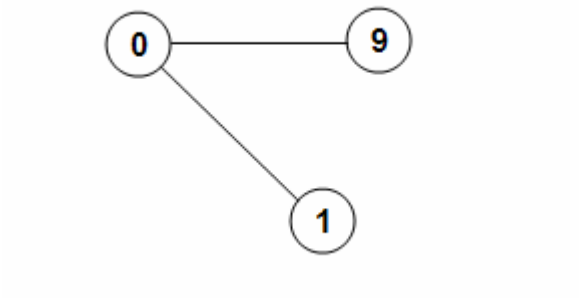
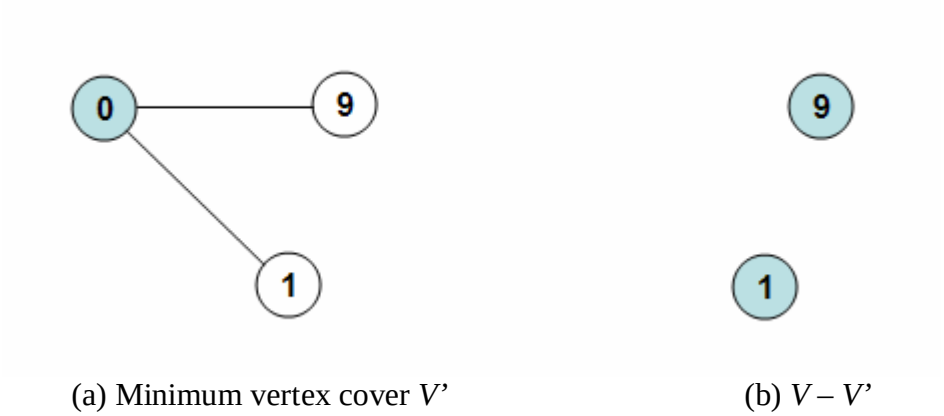


Figure 16. Two-cycle graph T(π) from Figure 8

Each vertex is labeled by the left most element belonging to each 2-cycle. As shown in G(π) from Figure 8, Vertex 0 and 9 are connected since 2-cycle [0 1 9 10] and [9 10 2 3] have a common gray edge. Vertex 0 and 1 are connected since 2-cycle [0 1 9 10] and [1 2 11 10] have a common black edge.

Given a graph $G = (V, E)$, a set of vertices which cover all edges in E is called ***vertex cover*** of G. A minimum subset of vertex cover V' in T(π) offers the minimum number of 2-cycles that cover all common edges. The maximum number of edge-disjoint 2-cycles can be achieved by excluding V' from V.

From Figure 16, {0} is the only minimum vertex cover V' in T(π). $V - V'$ of T(π) gives {9, 1}, which is the maximum set of edge-disjoint 2-cycles for reversals as shown in Figure 17.



(a) Minimum vertex cover V' (b) $V - V'$

Figure 17. Minimum vertex cover V' and V-V' in T(π) from Figure 16

Complementing a minimum vertex cover of T(π) guarantees a maximum set of edge-disjoint 2-cycles while a maximum matching M of F(π) followed by L(M) does not always offer an edge disjoint 2-cycle set of the maximum size.

The method of complementing a minimum vertex cover for finding a maximum set of edge-disjoint 2-cycles is demonstrated using a more complicated instance of edge sharing 2-cycles in the following.

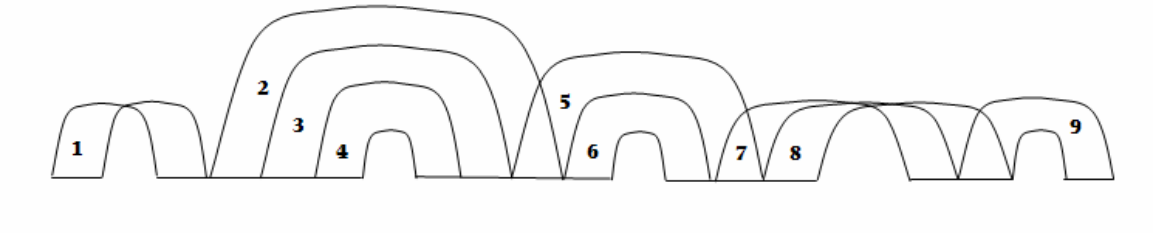


Figure 18. A 2-cycle set of a breakpoint graph G(π)

Figure 18 shows a set of 2-cycles from an arbitrary breakpoint G(π). A two-cycle graph T(π) is constructed from the G(π) as shown in Figure 19.

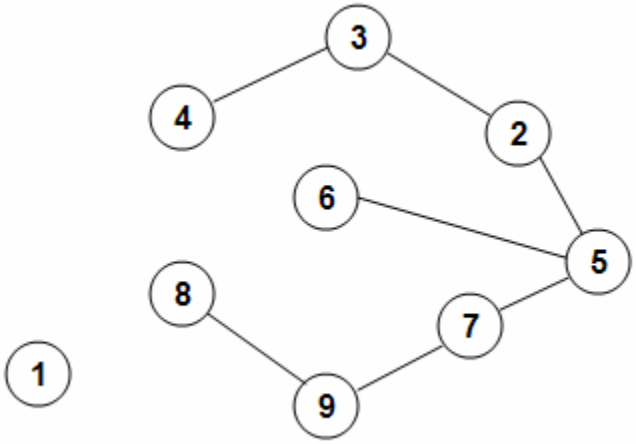
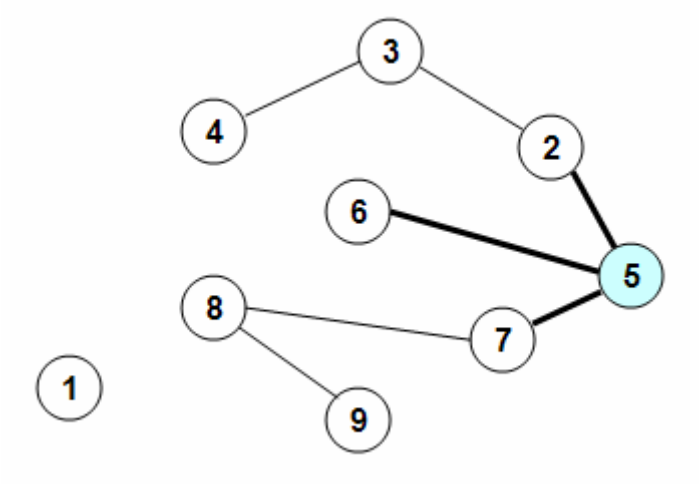
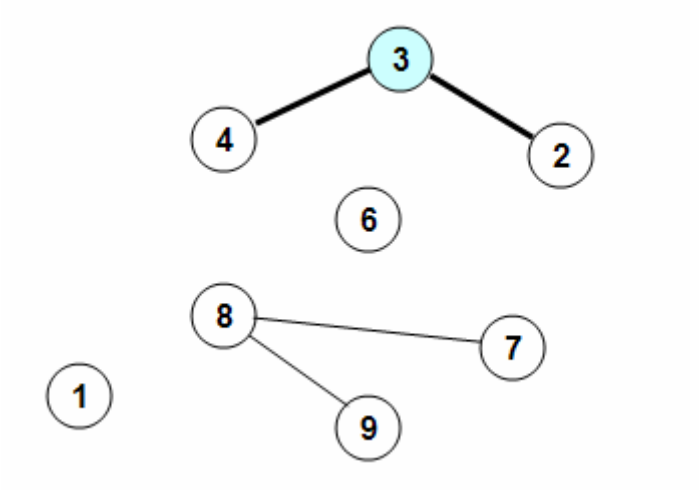


Figure 19. Two-cycle graph T(π) from Figure 18

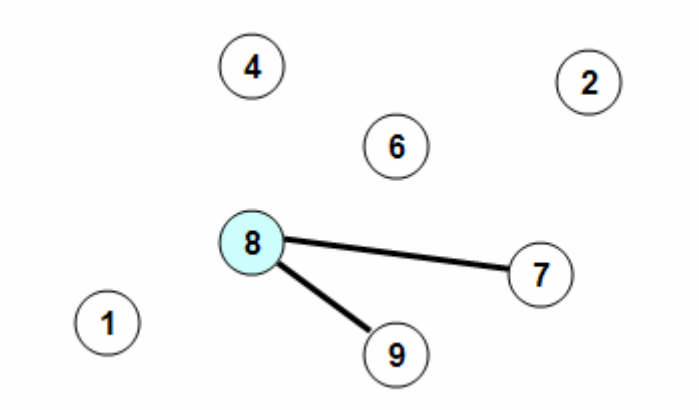
Figure 19 shows a vertex set V = {1, 2, 3, 4, 5, 6, 7, 8, 9} in T(π). Vertex 1 is isolated since it does not share either a black edge or a gray edge with any other vertex. The other vertices are edge connected based on its black edge or gray edge shared. To find a minimum vertex cover V' of T(π), each edge degree of every vertex is observed. The highest edge degree vertex can be the first candidate for V' since it covers the largest number of edges. V' of T(π) can be determined by selecting the highest edge degree vertex and removing it along with edges incident to it from T(π). Figure 20 shows the steps of determining V' of T(π) from Figure 19.



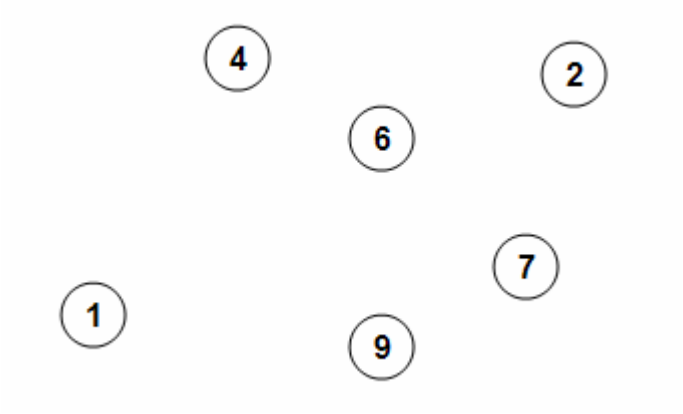
(a)



(b)



(c)



(d)

Figure 20. Steps of determining edge disjoint 2-cycles using vertex cover *V*'

First, vertex 5 in T(π) has the highest edge degree of 3, thus, it is removed along the edges incident to it as shown in Figure 20(b). After vertex 5 is removed, the next highest edge degree is 2, covered by vertex 3 or vertex 9. Vertex 3 is chosen in Figure 20(b) and it is removed along its edges. In Figure 20(c), vertex 9 shows the highest edge degree, therefore, it is removed along with its edges. After excluding vertex 9 and the edges incident to it, no edges remain. All edges are covered by vertex 5, 3, and 9, thus, a minimum vertex cover *V*' = {5, 3, 8}. Since *V*' represents a set of minimum 2-cycles that cover all shared edges, *V* - *V*' offers a maximum set edge-disjoint 2-cycles. The set of 2-cycles {1, 2, 4, 6, 7, 9} shown in Figure 20(d) is the maximum size of edge-disjoint 2-cycle set for G(π) from Figure 18. Figure 21 shows the edge-disjoint 2-cycles determined by *V* - *V*' from Figure 20.

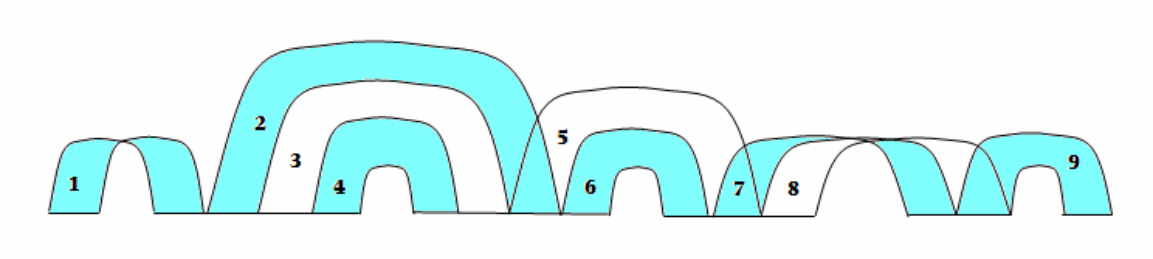


Figure 21. Edge-disjoint 2-cycles of Figure 18

As shown in Figure 21, all 2-cycles determined by *V* - *V*' do not have either a common black edge or a common gray edge with one another. *V* - *V*' achieves the maximum number of edge-disjoint 2-cycles since *V*' provides the minimum number of 2-cycles which covers all common edges between the 2-cycles.

Algorithm 6 shows the algorithm using vertex cover.

SORTINGBYREVERSALS3OVER2VERTEXCOVER(π)

```
1  construct the breakpoint graph G(π) of π
2  construct the two-cycle graph T(π) of G(π)
3  find a minimum vertex cover V' of the T(π)
4  exclude V' from  T(π)
5  find a cycle decomposition C that contains edge-disjoint 2-cycles V-V'
6  DoReversals(π)
```

DoReversals(π)

```
1  while b(π) > 0
2   if there is a 2-cycle
3      if there is an oriented 2-cycle
4           ρ = the reversal on the oriented 2-cycle
5      else
6           if there is an oriented crossing cycle on any unoriented 2-cycle
7                ρ = the reversal on the crossing cycle
8           else
9                ρ = the reversal on any unoriented crossing cycle
10  else
11     if there is an oriented cycles
12          if there is an oriented 2-cycle
13               ρ = the reversal on the oriented 2-cycle
14          else
15               ρ = the reversal on any oriented cycle
16     else
17          ρ = the reversal on any unoriented cycle
18   π = π • ρ
```

Algorithm 6. Algorithm using vertex cover [11]

Algorithm 6 also achieves a performance ratio of 3/2. However, it improves the reversal distance by maximizing the number of edge-disjoint 2-cycles, which can be used for 2-reversals, based on the lower bound of $b(\pi) - 1/2\ c_2(\pi)$.

7. Generating 2-reversals using 3-cycles

The 3/2-approximation algorithms described in this work focused on the number of 2-cycles in cycle decompositions since 2-cycles produce 2-reversals, which remove breakpoints by the largest number. 2-reversals are easily observed in 2-cycles since each 2-cycle is eliminated only by a 2-reversal. However, it can be observed that 3-cycles can have potential 2-reversals. The strategy of finding 2-reversals is expanded to 3-cycles in this section.

A 3-cycle must be converted into a 2-cycle by a 1-reversal to be eliminated. Since any cycle in a cycle decomposition of G(π) contains at least 2 black edges, a 3-cycle can be removed only by a 1-reversal followed by a 2-reversal. All types of 3-cycles are observed in what follows.

Let i, i+1, j, j+1, k, and k+1 be six elements in a permutation π, where $0 \leq i < j < k \leq n$, $j \neq i + 1$, and $k \neq j + 1$. Suppose (i, i+1), (j, j+1), and (k, k+1) are pairs of breakpoints that form a 3-cycle in a cycle decomposition of G(π), which means $|\pi_{i+1} - \pi_i| \neq |\pi_{j+1} - \pi_j| \neq |\pi_{k+1} - \pi_k| \neq 1$. The three pairs of elements that form three black edges are included in the same 3-cycle. The shape of the 3-cycle is determined by gray edges.

First, any elements of the six elements cannot form a gray edge with its neighbor element since $|\pi_{i+1} - \pi_i| \neq |\pi_{j+1} - \pi_j| \neq |\pi_{k+1} - \pi_k| \neq 1$. Thus, one element can form a gray edge with one of four other elements except for its neighbor element. Its neighbor elements can form a gray edge with one of the remaining elements except for the neighbor element of its gray edge element since it forms a 2-cycle if two elements of a black edge form gray edges with two elements of another black edge. The cases that one pair of black edge elements forms gray edges with another pair of black edge elements are shown in Figure 22. Thus, two elements that belong to a black edge must form gray edges with two elements that belong to two different black edges.

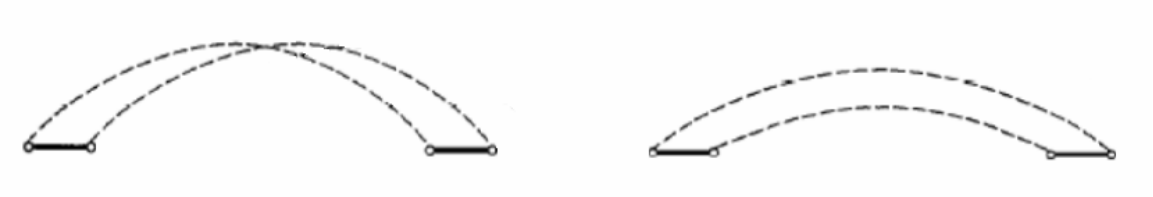


Figure 22. Two shapes of 2-cycles in G(π)

After its neighbor element determines an element for a gray edge, there are only two elements left. The remaining two elements form a gray edge.

In other words, the element i can form a gray edge with either j, j+1, k, or k+1. If i chooses j+1 for a gray edge, i+1 can choose either k or k+1. If i+1 chooses k+1, then j and k form another gray edge. Thus, there are only 8 possible shapes of 3-cycles. The 8 shapes of 3-cycles are shown in Figure 23.

It is observed that the 3-cycles shown in Figure 22(b) and Figure 22(e) are unoriented 3-cycle types since any reversal cannot bring two elements connected by a gray edge together. 3-cycles shown in Figure 22(a), Figure 22(d), and Figure 22(h) form oriented cycles and generate unoriented 2-cycles after a 1-reversal. 3-cycles shown in Figure 22(c), Figure 22(f), and Figure 22(g) are oriented cycle types and can generate oriented 2-cycles after a 1-reversal. The cycles after the reversals on each two breakpoints on the 3-cycles are shown in Figure 24.

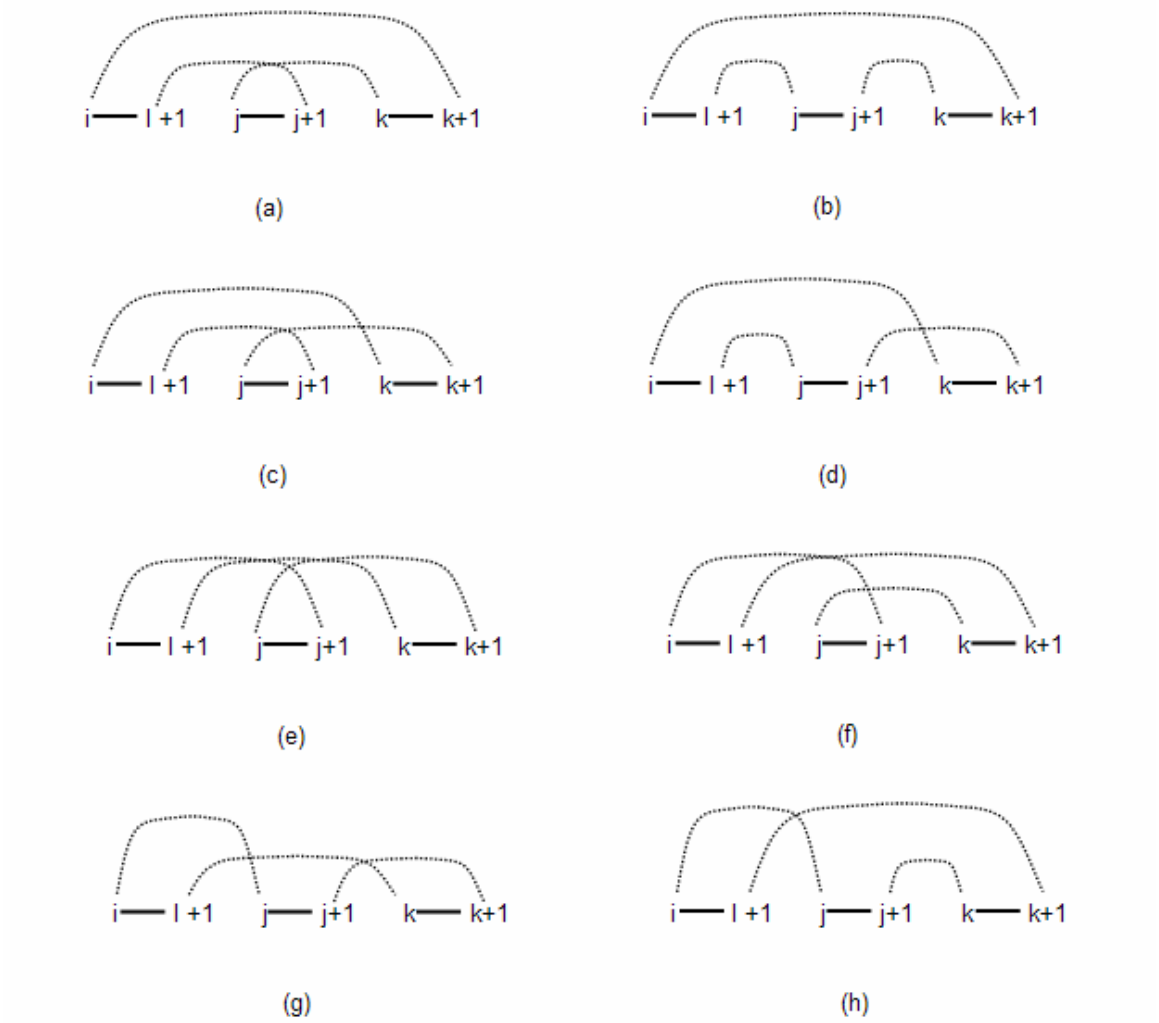
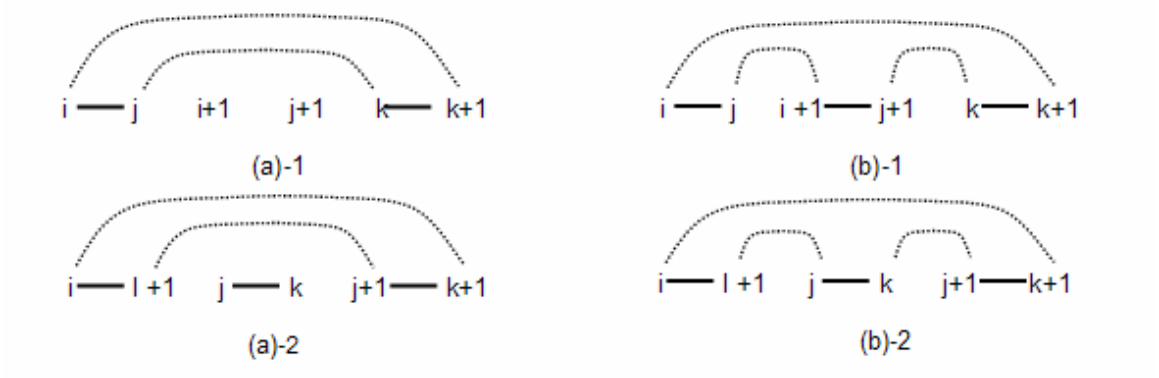


Figure 23. Eight shapes of 3-cycles in G(π)



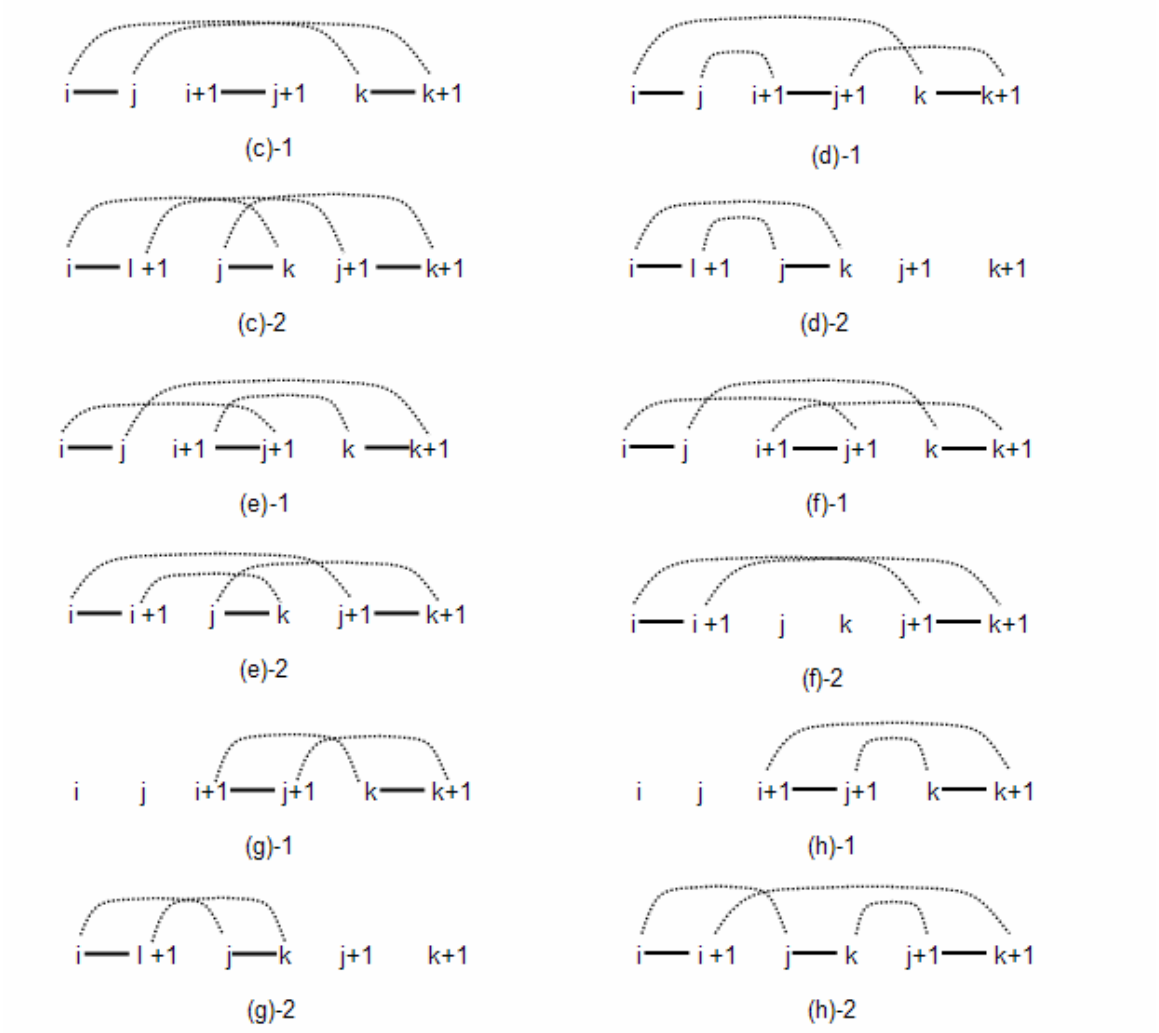


Figure 24. The cycles after each reversal from Figure 23

As described in Section 4, an unoriented 2-cycle has a crossing cycle that orients it. An unoriented 3-cycle also has a crossing cycle that transforms it into an oriented cycle since any cycle cannot be isolated in sorting by reversals problem. However, orienting an unoriented 3-cycle can be expensive compared to taking a 1-reversal belonging to a k-cycle, where $k > 3$, since a crossing cycle that orients the 3-cycle may be a 0-reversal. Thus, unoriented 3-cycles are not considered in the proposed algorithm. Oriented 3-cycles that generate unoriented 2-cycles are not considered in the algorithm either since it is assumed that taking at least three reversal steps does not bring benefits compared to taking 1-reversal from any oriented cycle at each reversal step. Therefore, only three types of 3-cycles, which generate oriented 2-cycles, are considered in the algorithm. The algorithm is shown in what follows.

SORTINGBYREVERSALS3OVER2VERTEXCOVER(π)

1 construct the breakpoint graph G(π) of π
2 construct the two-cycle graph T(π) of G(π)
3 find a minimum vertex cover V' of the T(π)
4 exclude V' from T(π)
5 find a cycle decomposition C that contains edge-disjoint 2-cycles V-V'
6 ***DoReversals2N3(π)***

DoReversals2N3(π)

```
1  while b(π) > 0
2   if there is a 2-cycle
3      if there is an oriented 2-cycle
4           ρ = the reversal on the oriented 2-cycle
5      else
6          if there is an oriented crossing cycle on any unoriented 2-cycle
7              ρ = the reversal on the crossing cycle
8          else
9              ρ = the reversal on any unoriented crossing cycle
10  else
11     if there is an oriented cycles
12         if there is an oriented 2-cycle generated by 1-reversal from a 3-cycle
13             ρ = the reversal on the oriented 2-cycle
14         else if there is a 3-cycle that generates an oriented 2-cycle
15             ρ = the reversal on the oriented 3-cycle
16         else
17             ρ = the reversal on any oriented cycle
18     else
19         ρ = the reversal on any unoriented cycle
20   π = π • ρ
```

Algorithm 7. Approximation algorithm using 2 and 3-cycles

Algorithm 7 takes a reversal on 2-cycles first. If there remain no 2-cycles, it takes a 1-reversal on any oriented cycle. Among oriented cycles, if there is a 3-cycle which generates an oriented 2-cycle, a 1-reversal from the 3-cycle is taken first. To take advantage of the oriented 2-cycle brought from the 3-cycle, the algorithm observes a 2-reversal first before finding a 1-reversal. If there are no oriented cycles left, then a 0-reversal is taken.

8. Running Time

The approximation algorithm described in Section 7 uses vertex cover to find a cycle decomposition with the maximum number of 2-cycles. The algorithm considers 3-cycles which generate oriented 2-cycles in the first priority for a reversal when there remain no 2-cycles chosen for the cycle decomposition.

The breakpoint graph $G(\pi)$ can be constructed in $O(n)$ time by using the inverse permutation π^{-1}. Black edges can be determined by checking π_{i-1} and π_{i+1} and gray edges can be determined by checking π_{i-1}^{-1} and π_{i+1}^{-1} for π_i of the permutation π, where $0 < i < n$.

The two-cycle graph $T(\pi)$ is constructed in $O(n^2)$ time. Every vertex, which represents a 2-cycle, in $T(\pi)$ is determined by checking the two gray edges associated with one black edge. If the other black edge that two gray edges are associated with is the same black edge, then the two black edges and two gray edges form a 2-cycle. Since black edges and gray edges are already determined in $G(\pi)$, the vertices of $T(\pi)$ are can be found in $O(n)$ time. Edges in $T(\pi)$ are constructed in $O(n^2)$ time since it can be determined by checking if the other 2-cycles contains the same black edge or gray edge that belongs to a 2-cycle.

Given a graph $G = (V, E)$, a minimum vertex cover can be found in $O(|V||E|)$, where $|V|$ denotes the number of vertices and $|E|$ denotes the number of edges in the graph. When considering the two-cycle graph has at most n/2 vertices and n/2 edges for a permutation with n elements, the minimum vertex cover V' of $T(\pi)$ can be found in $O(n^2)$ time. The cycle decomposition C from V-V' is found in $O(n)$ time.

A 3-cycle that generates an oriented 2-cycle can be found in $O(n)$ time by checking black edges and gray edges in the current permutation. The series of reversals can be found in $O(n)$ time using $T(\pi)$ and $G(\pi)$. Therefore, the algorithm achieves the time complexity of $O(n^2)$.

9. Experimental Runs

To test the efficiency of the algorithms that described in Section 6 and Section 7, unsigned permutations with various sizes, 10, 20, …., 100, were randomly generated. Ten permutations of each size are applied for estimating reversal distances. The experiments have three main goals. The first is to observe how close reversal distance values to optimal values are estimated by two 3/2-approximation algorithms. For the performance comparison the reversal distance obtained by the exact greedy algorithm is used as an optimal reversal distance. The performance measure used would be to see how close the result is to the optimal values. Since the exact greedy algorithm requires a significant computational time and storage for an optimal solution when the number of breakpoints is large, as described in Section 5, only permutations with a small number of breakpoints are applied for the first part of the experiments. In this section and the Appendix A the 3/2-approximation algorithm using a vertex cover for a cycle decomposition is referred to as the ‘vertex cover algorithm’ or ‘Vertex Cover’ in short. The 3/2-approximation algorithm using a maximum matching followed by the ladder graph for a cycle

decomposition is referred to as the ‘maximum matching algorithm’ or ‘Maximum Matching’ in short. The results of the first part of the experimental runs is shown in Table 1 in Appendix A.

As can be seen in Table 1, the vertex cover algorithm produces the same results as the exact greedy algorithm in 7 out of 10 cases. The result shows the two algorithms achieve the reversal distances which are close to the optimal values.

The second goal of the experimental runs is to compare the performances of the vertex cover and the maximum matching algorithms. Ten permutations of sizes, 20, 30, …., 90, 100, are applied for estimating reversal distances. We counted the number of 2-cycles and compared the reversal distances obtained by the two approximation algorithms for each permutation. The results are shown in Table 2 to Table 10 in Appendix A.

In most cases the two algorithms result in the same number of 2-cycles and reversal distances as shown in Table 2 to Table 10. In several cases the vertex cover algorithm shows slightly higher numbers of 2-cycles. However, the higher number of 2-cycles does not always produce lower reversal distances since unoriented 2-cycles, which are selected in a cycle decomposition, require 0-reversals followed by 2-reversals. Taking 1-reversals without considering unroiented 2-cycles occasionally take advantage of oriented 2-cycles generated in the intermediate reversal steps, which are not identified in the cycle decomposition step. However, it is observed in several cases that the higher numbers of 2-cycles obtained by the vertex cover algorithm produce the lower reversal distances in Table 6 and Table 8. Note that the size of the permutations does not affect the result of the experiments for the second part.

The third goal of the experimental runs is to observe how the use of 3-cycles affects the reversal distance in the approximation algorithm. In this section and Appendix A the approximation algorithm which considers 3-cycles is referred to as the ‘3-cycle algorithm’ or ‘3-cycle’ in short. The 3-cycle algorithm uses the vertex cover method, thus, the performance is compared to the vertex cover algorithm. The results are shown in Table 11 to Table 20 in Appendix A.

As shown in the result tables, the 3-cycle algorithm outperforms the reversal distances in many cases. The result shows 3-cycles can be actively used for producing 2-reversals in intermediate reversal steps to achieve the better reversal distance.

10. Conclusion

Genome rearrangement by reversals has served an important role in understanding evolutionary history between two species in terms of reversal distance. The sorting by reversals problem is modeled to find a series of reversals that converts one genome into another.

This paper introduced a new exact greedy algorithm. The exact algorithm offers an optimal reversal distance, however, it requires significant computational time and storage for the solution compared to approximation algorithms when the number of breakpoints is large.

An improved method of finding a cycle decomposition with the maximum number of 2-cycles for the problem is also proposed in this work. The proposed method of finding a maximum 2-cycles using a minimum vertex cover outperforms the method of using a

maximum matching and a ladder graph in terms of maximizing the number of edge-disjoint 2-cycles. The algorithm also achieves better time complexity compared to the algorithm introduced in [11], which uses a maximum matching and a ladder graph for a cycle decomposition. Christie described the proposed algorithm in [11] requires $O(n^4)$ running time. The proposed algorithm in this paper achieves the time-complexity of $O(n^2)$, which is better than $O(n^4)$.

This work also proposed the method of producing 2-reversals by using oriented 3-cycles. The result of the experiments shows that oriented 3-cycles can be actively used to reduce the reversal distance in general. The 3-cycles have not been focused in cycle decompositions for sorting by reversals. Formulating a more efficient way of using 3-cycles can be further studied to achieve a better approximation ratio based on 2-cycles and 3-cycles for sorting by reversals.

Appendix A

π	b(π)	d(π)		
		Exact Greedy	Maximum Matching	Vertex Cover
1	11	7	8	**7**
2	8	5	**5**	**5**
3	10	6	7	7
4	9	5	6	**5**
5	9	6	**7**	8
6	11	7	**7**	8
7	2	1	1	**1**
8	8	5	6	**5**
9	9	6	**6**	**6**
10	9	5	**5**	**5**

Table 1. Performance comparison of the three algorithms

Permutation size = 20

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	15	2	11	2	11
2	19	2	14	2	14
3	11	4	6	4	6
4	11	3	8	3	8
5	13	1	11	1	11
6	15	2	13	2	13
7	8	4	4	4	4
8	15	0	13	0	13
9	15	1	11	1	11
10	14	2	11	2	12

Table 2. Performance comparison of the two approximation algorithms, permutation size 20

Permutation size = 30

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	20	2	15	2	15
2	28	2	22	3	23
3	28	2	22	3	22
4	30	4	26	4	26
5	30	2	24	2	24
6	29	2	22	2	23
7	12	0	12	0	12
8	28	2	22	2	23
9	30	1	25	1	25
10	27	2	22	2	22

Table 3. Performance comparison of the two approximation algorithms, permutation size 30

Permutation size = 40

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	40	2	34	2	34
2	39	2	32	2	32
3	25	1	20	1	20
4	41	4	32	4	32
5	38	4	31	4	31
6	39	2	33	**3**	33
7	39	1	31	1	31
8	29	4	23	4	23
9	25	4	17	4	17
10	21	2	17	2	17

Table 4. Performance comparison of the two approximation algorithms, permutation size 40

Permutation size = 50

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	33	2	27	2	27
2	31	1	27	1	27
3	21	0	18	0	18
4	16	0	15	0	15
5	28	1	24	1	25
6	45	2	38	2	38
7	24	1	21	1	21
8	49	1	42	1	42
9	42	5	36	5	37
10	19	2	18	2	18

Table 5. Performance comparison of the two approximation algorithms, permutation size 50

Permutation size = 60

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	28	4	23	4	23
2	37	4	30	4	30
3	50	**2**	**45**	**3**	**43**
4	47	2	42	2	42
5	45	4	36	5	36
6	27	1	26	1	26
7	26	1	22	1	22
8	29	2	25	2	25
9	33	1	27	1	27
10	35	1	31	1	31

Table 6. Performance comparison of the two approximation algorithms, permutation size 60

Permutation size = 70

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	37	1	34	1	34
2	44	2	36	3	36
3	39	0	35	0	35
4	43	1	37	1	37
5	49	0	43	0	43
6	50	0	46	0	46
7	47	1	41	1	41
8	37	2	34	2	34
9	14	0	14	0	14
10	33	0	28	0	28

Table 7. Performance comparison of the two approximation algorithms, permutation size 70

Permutation size = 80

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	44	0	41	0	41
2	48	4	42	4	42
3	50	4	44	4	44
4	39	1	36	1	36
5	58	1	52	1	52
6	55	1	49	1	49
7	58	4	58	4	58
8	49	**2**	**46**	**3**	**45**
9	50	2	46	2	**42**
10	47	0	45	0	45

Table 8. Performance comparison of the two approximation algorithms, permutation size 80

Permutation size = 90

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	65	2	60	**3**	60
2	65	2	56	**3**	56
3	50	2	47	**3**	47
4	55	1	50	1	50
5	68	4	62	4	62
6	40	0	37	0	37
7	59	3	50	3	50
8	58	2	49	**3**	49
9	57	2	51	**3**	51
10	53	3	51	3	51

Table 9. Performance comparison of the two approximation algorithms, permutation size 90

Permutation size = 100

π	b(π)	Maximum Matching		Vertex Cover	
		$c_2(\pi)$	d(π)	$c_2(\pi)$	d(π)
1	51	2	47	**3**	47
2	81	6	72	6	72
3	70	2	63	**3**	63
4	57	2	51	**3**	51
5	60	1	56	1	56
6	84	2	74	**3**	74
7	76	4	69	4	69
8	70	0	64	0	64
9	68	2	60	2	61
10	68	1	63	1	63

Table 10. Performance comparison of the two approximation algorithms, permutation size 100

Permutation size = 20

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	$c_3(\pi)$ used	d(π)
1	15	11	11	2	11
2	19	14	14	2	14
3	11	6	6	0	6
4	11	8	8	0	8
5	13	11	**11**	**3**	**10**
6	15	13	13	2	13
7	8	4	4	0	4
8	15	13	**13**	**4**	**12**
9	15	11	11	3	12
10	14	11	12	1	12

Table 11. Performance comparison for the consideration of 3-cycles, permutation size 20

Permutation size = 30

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	$c_3(\pi)$ used	d(π)
1	20	15	15	3	15
2	**28**	**22**	23	**5**	**22**
3	**28**	**22**	**22**	**5**	**20**
4	**30**	**26**	**26**	**5**	**24**
5	**30**	**24**	**24**	**6**	**23**
6	29	22	23	**6**	**20**
7	12	12	12	1	12
8	**28**	**22**	**23**	**5**	**22**
9	**30**	**25**	**25**	**7**	**23**
10	27	22	22	3	22

Table 12. Performance comparison for the consideration of 3-cycles, permutation size 30

Permutation size = 40

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	40	34	**34**	**10**	**32**
2	39	32	32	7	32
3	25	20	20	5	21
4	41	32	32	5	32
5	38	31	**31**	**5**	**30**
6	39	33	**33**	**6**	**31**
7	39	31	31	7	33
8	29	23	23	3	23
9	25	17	17	2	17
10	21	17	17	3	17

Table 13. Performance comparison for the consideration of 3-cycles, permutation size 40

Permutation size = 50

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	33	27	**27**	**4**	**26**
2	31	27	27	3	28
3	21	18	**18**	**3**	**17**
4	16	15	15	1	15
5	28	24	**25**	**5**	**23**
6	45	38	**38**	**8**	**35**
7	24	21	21	3	21
8	49	42	42	8	42
9	42	36	**37**	**6**	**33**
10	19	18	18	3	18

Table 14. Performance comparison for the consideration of 3-cycles, permutation size 50

Permutation size = 60

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	28	23	23	4	23
2	37	30	30	4	30
3	50	45	43	5	43
4	47	42	**42**	**8**	**38**
5	45	36	36	7	36
6	27	26	26	2	26
7	26	22	**22**	**4**	**21**
8	29	25	25	3	25
9	33	27	27	5	29
10	35	31	**31**	**5**	**29**

Table 15. Performance comparison for the consideration of 3-cycles, permutation size 60

Permutation size = 70

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	37	34	**34**	**6**	**31**
2	44	36	36	5	36
3	39	35	35	5	35
4	43	37	**37**	**6**	**36**
5	49	43	**43**	**9**	**42**
6	50	46	**46**	**8**	**42**
7	47	41	**41**	**6**	**40**
8	37	34	34	3	34
9	14	14	14	1	14
10	33	28	28	5	29

Table 16. Performance comparison for the consideration of 3-cycles, permutation size 70

Permutation size = 80

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	44	41	**41**	**6**	**39**
2	48	42	**42**	**6**	**41**
3	50	44	44	8	44
4	39	36	**36**	**4**	**34**
5	58	52	**52**	**8**	**50**
6	55	49	**49**	**7**	**48**
7	58	58	**58**	**7**	**50**
8	49	46	**45**	**3**	**44**
9	50	46	42	5	43
10	47	45	**45**	**6**	**40**

Table 17. Performance comparison for the consideration of 3-cycles, permutation size 80

Permutation size = 90

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	65	60	60	5	60
2	65	56	**56**	**8**	**55**
3	50	47	47	2	47
4	55	50	**50**	**10**	**45**
5	68	62	**62**	**7**	**60**
6	40	37	**37**	**5**	**36**
7	59	50	**50**	**7**	**48**
8	58	49	49	6	49
9	57	51	51	7	51
10	53	51	**51**	**5**	**48**

Table 18. Performance comparison for the consideration of 3-cycles, permutation size 90

Permutation size = 100

π	b(π)	Matching	VertexCover	3-Cycle	
		d(π)	d(π)	c_3(π) used	d(π)
1	51	47	**47**	**5**	**45**
2	81	72	72	9	73
3	70	63	**63**	**11**	**57**
4	57	51	51	6	50
5	60	56	56	4	58
6	84	74	**74**	**10**	**73**
7	76	69	**69**	**9**	**66**
8	70	64	**64**	**9**	**61**
9	68	60	**61**	**11**	**59**
10	68	63	**63**	**10**	**59**

Table 19. Performance comparison for the consideration of 3-cycles, permutation size 100

References

[1] Jones N, Pevzner P. An Introduction to Bioinformatics Algorithms. MIT Press. 2004.

[2] O'Brien, S.J., editor. Genetic Maps. Cold Spring Harbor Laboratory. 1987.

[3] Watterson, G.A., W.J. Ewens, T.E. Hall, and A. Morgan. The chromosome inversion problem. Journal of Theoretical Biology 99, 1-7, 1982.

[4] A. Caprara. Sorting by reversals is difficult. In Proceedings of the first International Conference on Computational Molecular Biology, pages 75-83, New York, January 19-22 1997. ACM Press.

[5] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica, 13:180-210, 1995.

[6] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. SIAM Journal on Computing, 25(2):272-289, 1996.

[7] P. Berman, S. Hannenhalli, M. Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. 2001.

[8] Setubal C, Meidanis J. Introduction to Computational Molecular Biology. PWS Publishing. 1st edition. 1997.

[9] Skomorokhov A. Genetic Algorithms:APL2 Implementation and a Real Life Application. Institute of Atomic Engergetics. 1997.

[10] Misevicius A. A Fast hybrid Genetic Algorithm for the Quadratic Assignement Problem. Kaunas University of Technology. 2006.

[11] Christie, D.A. A 3/2-Approximation Algorithm for Sorting by Reversals. Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 244-252. 1998.

[12] L. lovasz and M.D. Plummer. Annals of Discrete Mathematics (29): Matching Theory. Number 121 in North-Holland Mathematics Studies. North-Holland, Amsterdam, 1986.