# Improving the Performance of a Genome Sorting Algorithm with Inverted Block-Interchange

Deen Md Abdullah[1,2,3,4], Wali Md Abdullah[1,2,3,4], M. Sohel Rahman[3,5]
[1]IICT, Bangladesh University of Engineering and Technology, Dhaka-1000, Bangladesh
[2]Military Institute of Science and Technology, Dhaka, Bangladesh
[3]AℓEDA group
Department of CSE, Bangladesh University of Engineering and Technology, Dhaka-1000, Bangladesh
Email: [4]{rabi.mist, shashi.mist}@yahoo.com, [5]msrahman@cse.buet.ac.bd

***Abstract*— A classic problem in comparative genomics is to find a shortest sequence of evolutionary operations that transform one genome into another. There are different types of genome rearrangement operators such as reversals, transpositions, translocations, block interchange, double cut and join (DCJ) etc. In this paper we consider reversals and block-interchanges simultaneously and incorporate *inverted block-interchange* in a heuristic algorithm, which inverts one or both of the two swapped segments of a block-interchange. Experimental results confirm that incorporation of *inverted block-interchange* always results in a better (or at least equal) sorting sequence.**

***Index Terms*— Genome rearrangement, genome sorting, inverted block-interchange, sorting permutations.**

## I. Introduction

THE study of genome rearrangements has been one of the most promising methods for tracing the evolutionary history using gene order comparisons between organisms. In a species, there are a large number of genes and the ordering of the genes is hugely important. Since we are interested in the order of genes, we label each gene with a unique number. This numbers are represented by $+$ or $-$ signs. We measure the similarity of two genomes by measuring how easy it is to transform one genome to another by some operations. Among these operations *reversal*, *block interchange*, *tandem duplication*, *deletion* etc. are commonly used for genome rearrangement.

*Reversals* is the most widely studied type of global mutations [2]–[4], which inverts a segment in the permutation and changes the sign of each integer in that segment. In a genome rearrangement problem if we only consider *reversals*, then the problem of *sorting by reversals* is to find the shortest series composed of reversals that transform the given permutation into another [4].

*Block-interchanges* are also global mutations that act on a permutation [5]–[8]. A universal double-cut-and-join (*DCJ*) operation was introduced by Yancopoulos et al. that accounts for reversals, translocations, fissions, fusions and block-interchanges by assigning a weight of 1 to all operations except *block-interchanges*, which get a weight of 2 [8].

A preliminary version of the paper was appeared at [1].

Ying et al. introduced several approaches to examine genome rearrangement problems by considering reversals and block-interchanges together under various weight assignments [9]. Their algorithm reports an acceptable solution with theoretical guarantees and experimental evidences. Being motivated by the work of [9], we here consider *reversals* and *block interchanges* simultaneously and apply *inverted block-interchange* to improve a heuristic algorithm presented by Bader [10]. Bader considered signed permutation. So we have also considered signed permutation in our work.

Our contribution in this paper lies in improving a heuristic algorithm of [10] from a practical point of view by introducing the operation of *inverted block-interchange*. In particular, we have experimentally shown that the inclusion of *inverted block-interchange* operation can provide better results for genome rearrangement. We believe, this finding can will have some impact in tracing the evolutionary history using gene order comparisons between organisms.

We organize the rest of the paper as follows. In Section III, we give the definitions and discuss preliminary concepts. Result of our contribution is described in Section IV. Finally, Section V concludes the paper.

## II. Related Work

More precisely, given two genomes, one wants to find an optimal (shortest) sequence of rearrangements that transforms this genome into the other. In the classical approach, each gene has exactly one copy in each genome, and only operations that do not change the genome content are considered. These "classical operations" are nowadays a well-studied subject. The most important operations are reversals (also called inversions) and *transpositions*. A section of the genome is excised, reversed in orientation, and reinserted in *reversals* operation. For *transpositions* operation a section of the genome is excised and reinserted at a new position in the genome. While the problem of *Sorting by reversals* can be solved in polynomial time only if the underlying permutation is signed [2]–[4], and the reversal distance can be determined in linear time [11], the problem gets more complicated if one also considers transpositions, and there are only approximation algorithms known [5]–[7]. Also, very recently, it has been proved that *Sorting by Transpositions* is NP-Hard [12]. To simplify the existing algorithms, Yancopoulos et al. invented the *double cut and join* operator, which can simulate reversals and block interchanges (a more generalized form of a transposition), resulting in a simple and efficient algorithm [8].

However, restricting the genes to be unique in each genome does not reflect the biological reality very well, as in most genomes that have been studied, there are some genes that are presented in two or more copies. This holds especially for the genomes of plants, and one of the most prominent genomes is the one of the flowering plant *Arabidopsis thaliana*, where large segments of the genome have been duplicated (see e.g. [13]). There are various evolutionary events that can change the content of the genome, like duplications of single genes, horizontal gene transfer, or tandem duplications. For a nice overview in the context of comparative genomics, see [14]. From an algorithmic point of view, the existence of duplicated genes complicates many existing algorithms, for example the problem of sorting arbitrary strings by reversals [15] and the problem of sorting by reversals and duplications [16] have been proven to be NP-hard. So far, most of the existing algorithms restrict duplications to have a fixed length [17], or simulate duplications by arbitrary insertions [18]–[20]. Even with these restrictions, it is hard to solve most of the problems exactly, and heuristics have to be used.

While genome rearrangement problems without duplications are a well studied subject, considering genomes with duplicated genes is a rather new field of research. One of the first works on this topic was done by Sankoff [21], where the following problem was examined: Given two genomes with duplicated genes, identify in both genomes the “true exemplars” of each gene and remove all other genes, such that the rearrangement distance between these modified genomes is minimized. This approach minimizes the number of classical rearrangement operations, but not the one of duplications and deletions. In the work of El Mabrouk [22], for a given genome with duplicated gene content, one searches for a hypothetical ancestor with unique gene content such that the reversal and duplication distance towards this ancestor is minimized. Bertrand et al. [17] developed an algorithm for the following problem: Given two genomes with duplicated gene content, find a hypothetical ancestor such that the sum of the reversal and duplication distance of both genomes to this ancestor is minimized. However, in this work, duplications are restricted to have the length of one marker, i.e., a duplication can only duplicate segments that are identical in the initial genomes. Therefore, this approach is disadvantageous if large segmental duplications happened during evolution. Fu et al. extended this approach to the greedy algorithm MSOAR for assigning orthologous genes, which works well in practice [16], [23]. Other approaches [18]–[20] simulate duplications by arbitrary insertions. Recently, Yancopoulos and Friedberg provided a mathematical model of a genome rearrangement distance for genomes with unequal gene content [24], combining the DCJ operator [8] with arbitrary but lengthweighted insertions and deletions. Another field of research is the “Genome halving problem”, where a rearrangement scenario consists of a whole genome duplication followed by a series of classical rearrangement operations. It has been studied first for reversals and translocations [25], [26] and recently has been extended to the double cut and join operator [27], [28].

To the best of our knowledge, the only approach that creates a rearrangement scenario between two genomes, consisting of duplications of arbitrary length and classical genome rearrangements, is the one of Ozery-Flato and Shamir [29]. They use a greedy algorithm that starts with one genome and in each step applies the simplest and most evident operation that brings this genome closer to the target genome. If there is no evident operation, the algorithm aborts. Although this approach fails on complicated rearrangement scenarios, they were able to find rearrangement scenarios for more than 98% of the karyotypes in the “Mitelman database of chromosome aberrations in cancer”.

## A. Previous work

Bader focused on the genome arrangement problem assuming *reversals*, *block interchanges*, *tandem duplications*, and *deletions* operations [10]. In contrast to most of the previous works, in [10], tandem duplications and deletions can be of arbitrary length.

There is another approach which extends to multichromosomal genomes [30]. An exact algorithm for the weight proportion of 1 : 2 is developed, and then, its idea is extended to design approximation algorithms for other weight assignments [9]. A framework is given to establish an efficient correction for two models, one that includes insertions, deletions and double cut and join (DCJ) operations, and one that includes substitutions and DCJ operations [31].

In Bader’s work [10], instead of searching for a sequence of operations that sorts identity genome into a given genome, Bader searched for the inverse sequence that sorts a given genome into identity genome and he mentioned this sequence as the distance. He proved that, the breakpoint graph of identity genome has $n+1$ number of components and no loops. If the given genome is not an identity genome then it must contain a breakpoint. Then there will either be a loop or it will contain less than $n+1$ number of components. He showed that, at most $n+1$ sequence of operations are needed to make $n+1$ number of components in the given genome that transforms it into identity genome. If the given genome already contains $C(\pi)$ number of components, then one can avoid that number of operations. Again if there are $S(\pi)$ number of loops then one has to perform additional $\lceil \frac{S_i}{2} \rceil$ number of operations. He developed a lower bound for this distance and also proved that this lower bound can be decreased by at most 1 [10]. Based on this lower bound he developed a heuristic algorithm (Algorithm 1).

The lower bound $lb(\pi)$ of the distance $d(\pi)$ is as follows:

$d(\pi) \geq lb(\pi) = n + 1 - C(\pi) + \sum_{components} \lceil \frac{S_i}{2} \rceil$

where $S_i$ is the number of vertices with a loop in component $C_i$.

Bader's algorithm (Algorithm 1) uses a greedy strategy to sort the genome. In each step, it searches for operations that decrease the lower bound, i.e. it searches for operations that increase $C$ or decrease $S$, and check their effect on the lower bound. If there is no such operation, it will use additional heuristics to search for small sequences of operations that reduce the number of missing elements or duplications and creates adjacencies. The main idea behind these heuristics is to reduce the number of missing elements and duplicates and to create adjacencies.

We follow Bader's algorithm and include an operation *inverted block-interchange* in the set of operations of Bader's algorithm. Experimental results show that after including *inverted block-interchange* in Bader's algorithm, we get sorting sequences those are better or equal for all cases than sorting sequences which are found by the former set of operations.

**Algorithm 1** Bader's Algorithm [10]

```
while $\pi \neq id$ do
  Find all operations that decrease $lb(\pi)$
  if operation found then
    apply an operation that maximizes $\tau(\pi)$
  else
    find inverse tandem duplications
    find sequences for segments with multiplicity $\geq 3$
    find operations that create adjacencies
    find sequences for cases $A, B, C$
    apply a sequence that maximizes $\tau(\pi)$
  end if
end while
```

## III. Inverted Block Interchange and Other Operations

Genome is represented as a string over the alphabet $1, ..., n$. Each element may have a positive or negative orientation. A genome is called *augmented genome* if the element 0 is added at the beginning and the element $n + 1$ is added at the end. For simplicity we shall use the term *genome* instead of augmented genome. The number of occurrences of an element in genome is its *multiplicity*. In a *genome rearrangement problem*, we are given two genomes and a set of possible operations, where each operation is assigned a weight, and we need to find a sequence with minimum cumulative weight that transforms one into another.

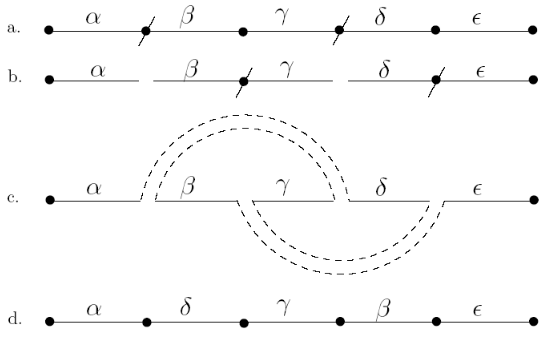


Figure 1. Example of *block interchanges* which is simulated by *DCJ*.

### A. Operations

In this paper we shall consider the operations described below. A *reversal* is an operation that inverts the order of the elements of a segment. A *block interchange* changes the position of the segments. A *tandem duplication* is an operation that inserts an identical copy of a segment immediately after this segment in a genome. A *deletion* cuts a segment out of a genome.

An important point is why such a weight scheme is used here. The idea here is to find a sequence of minimum weight that transforms one genome into another. Yancopoulos et al. introduced a universal *double-cut-and-join* operation that accounts for *reversals*, *block-interchanges* by assigning weight 1 and 2 respectively [8]. So during assigning weight for *inverted block-interchange* we follow this scheme to relate it with the work of Bader [10].

### B. Inverted block-interchange

To simplify the existing algorithms, Yancoupoulos et al. invented the *Double Cut and Join* operator [8], which can simulate *block interchanges*. Two *DCJ's* are equivalent to a single *Block interchange* (i.e. of weight 2) as shown in Figure 1.

*Inverted block-interchange* is a more specified form of a *block interchange*. During rejoining the blocks, this operation gives the option to the blocks for reversing their orientation. These blocks reverse their orientation when they are in non-increasing order. If no blocks are in non-increasing order then *Inverted block-interchange* performs like *Block interchange*. According to Ying et al. [9], two *DCJ's* are also equivalent to an *Inverted block-interchange*. So, it also has weight 2.

There may be three cases:

- CASE 1: Two blocks are in non-increasing order. So, both blocks reverse their orientation before interchange as shown in Figure 2;
- CASE 2: Only first block is in non-increasing order. So, first block reverses its orientation before interchange as shown in Figure 3;
- CASE 3: Only second block is in non-increasing order. So, second block reverses its orientation before interchange as shown in Figure 4.

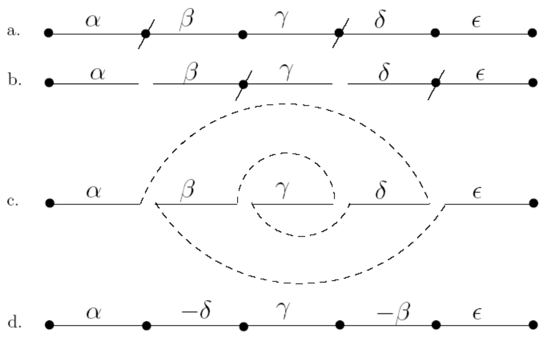


Figure 2. Example of *Inverted block-interchange* where two blocks reverse their orientation before interchange.

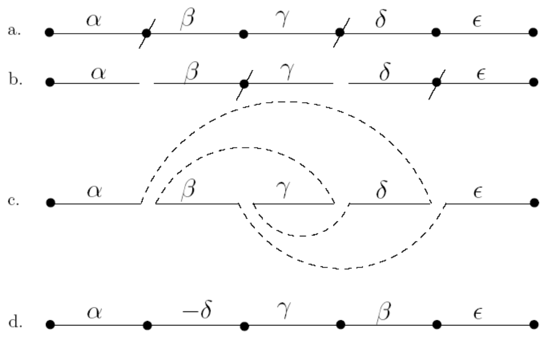


Figure 3. Example of *Inverted block-interchange* where first block reverses its orientation before interchange.

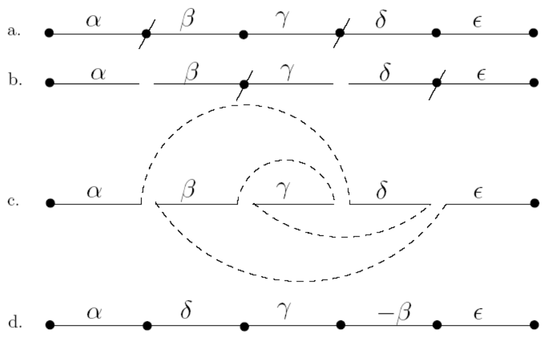


Figure 4. Example of *Inverted block-interchange* where second block reverses its orientation before interchange.

## IV. Results

In this section we present our experimental results. The experiments were performed using Intel Celeron M CPU 430 at 173GHz processor having 256MB RAM running Linux operating system. Using the same environment we have implemented Bader's work as well. During our experiment we followed the same procedures that Bader followed in his work [10] and also used the same simulated data as follows.

We generated test cases by creating the identity genome $id$ of size $n$ and applying random sequences of $\alpha n$ operations for different values of $\alpha$ $(0.1 \leq \alpha \leq 1)$. For each value of $\alpha$, we created 11 test cases for different values of $n$ such as $n = 20, n = 50, n = 80, n = 100$ etc. We tested the algorithm with Bader's set of operations as well as with the set of operations we proposed. For all values of $n$ we got equal or better performance than Bader.

For testing and performance comparison, first we applied 110 test cases to Bader's algorithm with his given set of operations. From each test case we got its calculated operation. Then we calculated the average number of calculated operations which are reported in Table I, III, V and VII.

Then we applied the same test cases including *inverted block-interchange* into the previous set of operations and do the same calculations as before. The results are reported in Tables II, IV, VI and VIII. The results for which we got better results are highlighted with boldfacing. From performance comparison we observed that, for all values of $n$ including *inverted block-interchange* gives better or equal performance.

The results of these experiments have been plotted in Figure 5(a)-5(d). The X-axis corresponds to the sequence weight used to obtain the test case, while the Y-axis corresponds to the weight of the reconstructed sequence. Each value is the average of 11 created test cases. The number of operations required to obtain the test cases is called performed operation and the number of operations required to reconstruct the sequence is called calculated operation. From the experiment we can see that, if we increase the number of performed operation, our work gives less number of calculated operation than the Bader's work. So, our work will give better performance for increasing the probability of *Inverted block-interchange*. i.e. for higher value of $\alpha$ and $n$.

### A. Discussion

*Inverted block-interchange* is a more specified form of a *block interchange*. Experimental results of our work showed that after including *inverted block-interchange* in Bader's algorithm, we got sorting sequences those were better or equal for all cases than sorting sequences which were found by the former set of operations. Thus, *inverted block-interchange* could be easily included in any heuristic algorithm where *block interchanges* or *transpositions* are mainly used. On the other hand, when frequency of *inverted block-interchange* is high in any test cases of an algorithm then this could give better performance.

## V. Conclusion

We mainly studied genome rearrangement problem by considering *inverted block-interchange*, which we used with a heuristic algorithm for better performance. Experimental results showed that our work finds sorting sequences which are better or equal for all cases than the previous heuristic algorithm. In Section III-B we showed three cases where *inverted block-interchange* is better than *block interchange*. Further research can apply *inverted block-interchange* in different heuristic algorithm for finding better performance.

## Acknowledgment

The authors would like to thank Martin Bader for his valuable support and the reviewers for their useful comments.

TABLE I.
PERFORMANCE WITHOUT INVERTED BLOCK INTERCHANGE ($n = 20$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | calculated operation |
| 0.1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 0.2 | 4 | 4 | 4 | 5 | 2 | 4 | 4 | 4 | 4 | 2 | 4 | 4 | 3.73 |
| 0.3 | 6 | 5 | 6 | 6 | 6 | 6 | 7 | 6 | 5 | 6 | 6 | 6 | 5.91 |
| 0.4 | 8 | 7 | 8 | 5 | 6 | 7 | 6 | 6 | 7 | 8 | 8 | 7 | 6.82 |
| 0.5 | 10 | 9 | 10 | 6 | 12 | 10 | 8 | 10 | 10 | 7 | 7 | 10 | 9 |
| 0.6 | 12 | 9 | 12 | 16 | 10 | 12 | 12 | 9 | 11 | 12 | 9 | 12 | 11.27 |
| 0.7 | 14 | 13 | 7 | 18 | 15 | 14 | 8 | 13 | 11 | 13 | 13 | 14 | 12.64 |
| 0.8 | 16 | 10 | 17 | 14 | 16 | 13 | 16 | 13 | 11 | 13 | 17 | 14 | 14 |
| 0.9 | 18 | 14 | 16 | 15 | 12 | 15 | 11 | 15 | 11 | 15 | 12 | 13 | 13.55 |
| 1 | 20 | 18 | 11 | 13 | 12 | 13 | 15 | 17 | 17 | 13 | 18 | 13 | 14.55 |

TABLE II.
PERFORMANCE WITH INVERTED BLOCK INTERCHANGE ($n = 20$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | calculated operation |
| 0.1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 0.2 | 4 | 4 | 4 | 5 | 2 | 4 | 4 | 4 | 4 | 2 | 4 | 4 | 3.73 |
| 0.3 | 6 | 5 | 6 | 6 | 6 | 6 | 7 | 6 | 5 | 6 | 6 | 6 | 5.91 |
| 0.4 | 8 | 7 | 8 | 5 | 6 | 7 | 6 | 6 | 7 | 8 | 8 | 7 | 6.82 |
| 0.5 | 10 | 9 | 10 | 6 | 12 | 10 | 8 | 10 | 10 | 7 | 7 | 10 | 9 |
| 0.6 | 12 | 9 | 12 | 16 | 10 | 12 | 12 | 9 | 11 | 12 | 9 | 12 | 11.27 |
| 0.7 | 14 | 13 | 7 | 18 | 15 | 14 | 8 | 13 | 11 | 13 | 13 | 14 | 12.64 |
| 0.8 | 16 | 10 | 17 | 14 | 16 | 13 | 16 | 13 | 11 | 13 | 17 | **13** | **13.91** |
| 0.9 | 18 | 14 | 16 | 15 | 12 | 15 | 11 | 15 | 11 | 15 | 12 | **12** | **13.45** |
| 1 | 20 | 18 | 11 | 13 | **11** | 13 | 15 | 17 | 17 | 13 | 18 | **12** | **14.36** |

TABLE III.
PERFORMANCE WITHOUT INVERTED BLOCK INTERCHANGE ($n = 50$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | calculated operation |
| 0.1 | 5 | 5 | 5 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 4.82 |
| 0.2 | 10 | 11 | 8 | 10 | 9 | 10 | 9 | 10 | 11 | 9 | 10 | 10 | 9.73 |
| 0.3 | 15 | 14 | 12 | 16 | 16 | 15 | 16 | 10 | 16 | 13 | 9 | 13 | 13.64 |
| 0.4 | 20 | 15 | 18 | 19 | 17 | 26 | 18 | 21 | 18 | 15 | 16 | 15 | 18 |
| 0.5 | 25 | 30 | 25 | 29 | 32 | 23 | 25 | 27 | 28 | 26 | 21 | 28 | 26.73 |
| 0.6 | 30 | 30 | 37 | 23 | 30 | 29 | 23 | 27 | 33 | 29 | 29 | 29 | 29 |
| 0.7 | 35 | 27 | 36 | 42 | 47 | 35 | 36 | 23 | 36 | 30 | 35 | 35 | 34.73 |
| 0.8 | 40 | 33 | 30 | 35 | 32 | 31 | 27 | 42 | 34 | 29 | 46 | 34 | 33.91 |
| 0.9 | 45 | 36 | 40 | 39 | 69 | 29 | 57 | 54 | 42 | 42 | 34 | 42 | 44 |
| 1 | 50 | 42 | 50 | 45 | 60 | 35 | 37 | 41 | 22 | 51 | 26 | 41 | 40.91 |

TABLE IV.
PERFORMANCE WITH INVERTED BLOCK INTERCHANGE ($n = 50$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | calculated operation |
| 0.1 | 5 | 5 | 5 | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 4.82 |
| 0.2 | 10 | 11 | 8 | 10 | 9 | 10 | 9 | 10 | 11 | 9 | 10 | **9** | **9.64** |
| 0.3 | 15 | 14 | **11** | 16 | 16 | **14** | 16 | **9** | 16 | **12** | 9 | **12** | **13.18** |
| 0.4 | 20 | **14** | 18 | 19 | 17 | 26 | 18 | 21 | 18 | 15 | 16 | **14** | **17.82** |
| 0.5 | 25 | 30 | 25 | 29 | 32 | 23 | 25 | 27 | 28 | 26 | 21 | 28 | 26.73 |
| 0.6 | 30 | 30 | 37 | 23 | 30 | 29 | 23 | 27 | 33 | 29 | 29 | 29 | 29 |
| 0.7 | 35 | 27 | 36 | 42 | 47 | 35 | 36 | 23 | 36 | 30 | 35 | 35 | 34.73 |
| 0.8 | 40 | 33 | 30 | 35 | 32 | 31 | 27 | 42 | 34 | 29 | 46 | 34 | 33.91 |
| 0.9 | 45 | 36 | 40 | 39 | 69 | 29 | 57 | 54 | 42 | 42 | 34 | 42 | 44 |
| 1 | 50 | 42 | 50 | 45 | 60 | 35 | 37 | 41 | 22 | 51 | 26 | 41 | 40.91 |

## REFERENCES

[1] D. M. Abdullah, W. M. Abdullah, and M. S. Rahman, "An improved heuristic algorithm for sorting genomes with inverted block-interchanges," in *Computer and Information Technology (ICCIT), 2011 14th International Conference on*, 2011, pp. 128–133.

[2] S. Hannenhalli and P. A. Pevzner, "Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals," *Journal of the ACM*, vol. 46, no. 1, pp. 1–27, Jan. 1999.

[3] E. Tannier, A. Bergeron, and M.-F. Sagot, "Advances on sorting by reversals," *Discrete Applied Mathematics*, vol. 155, pp. 881–888, Apr. 2007.

[4] Y. Han, "Improving the efficiency of sorting by reversals," in *BIOCOMP'06*, 2006, pp. 406–409.

[5] T. Hartman and R. Shamir, "A simpler and faster 1.5-approximation algorithm for sorting by transpositions,"

TABLE V.
PERFORMANCE WITHOUT INVERTED BLOCK INTERCHANGE ($n = 80$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of calculated operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
| 0.1 | 8 | 9 | 8 | 7 | 8 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 7.91 |
| 0.2 | 16 | 15 | 17 | 17 | 15 | 15 | 18 | 11 | 14 | 14 | 18 | 15 | 15.36 |
| 0.3 | 24 | 29 | 23 | 25 | 20 | 20 | 30 | 31 | 25 | 26 | 26 | 26 | 25.55 |
| 0.4 | 32 | 31 | 24 | 27 | 27 | 34 | 37 | 34 | 26 | 29 | 35 | 31 | 30.45 |
| 0.5 | 40 | 40 | 52 | 41 | 31 | 48 | 53 | 34 | 38 | 45 | 33 | 41 | 41.45 |
| 0.6 | 48 | 60 | 42 | 51 | 54 | 30 | 43 | 66 | 68 | 54 | 49 | 51 | 51.64 |
| 0.7 | 56 | 85 | 84 | 83 | 47 | 56 | 84 | 69 | 32 | 35 | 48 | 56 | 61.73 |
| 0.8 | 64 | 79 | 75 | 61 | 87 | 61 | 80 | 43 | 64 | 67 | 50 | 67 | 66.73 |
| 0.9 | 72 | 97 | 96 | 81 | 60 | 50 | 62 | 79 | 83 | 101 | 91 | 81 | 80.09 |
| 1 | 80 | 106 | 88 | 73 | 52 | 85 | 79 | 100 | 59 | 32 | 73 | 73 | 74.55 |

TABLE VI.
PERFORMANCE WITH INVERTED BLOCK INTERCHANGE ($n = 80$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of calculated operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
| 0.1 | 8 | 9 | 8 | 7 | 8 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 7.91 |
| 0.2 | 16 | 15 | 17 | 17 | 15 | 15 | 18 | 11 | 14 | 14 | 18 | 15 | 15.36 |
| 0.3 | 24 | 29 | 23 | 25 | 20 | 20 | 30 | 31 | 25 | 26 | 26 | 26 | 25.55 |
| 0.4 | 32 | 31 | 24 | 27 | 27 | 34 | 37 | 34 | 26 | 29 | 35 | 31 | 30.45 |
| 0.5 | 40 | 40 | 52 | 41 | 31 | 48 | **52** | **33** | 38 | 45 | 33 | 41 | **41.27** |
| 0.6 | 48 | 60 | 42 | **50** | 54 | 30 | 43 | 66 | 68 | 54 | 49 | 51 | **51.55** |
| 0.7 | 56 | **84** | **83** | 83 | 47 | 56 | **83** | 69 | 32 | 35 | 48 | 56 | **61.45** |
| 0.8 | 64 | 79 | 75 | **60** | 87 | **60** | **79** | 43 | 64 | 67 | 50 | 67 | **66.45** |
| 0.9 | 72 | 97 | 96 | 81 | 60 | 50 | 62 | 79 | 83 | 101 | 91 | 81 | 80.09 |
| 1 | 80 | 106 | 88 | 73 | **51** | 85 | 79 | 100 | 59 | 32 | 73 | 73 | **74.45** |

TABLE VII.
PERFORMANCE WITHOUT INVERTED BLOCK INTERCHANGE ($n = 100$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of calculated operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
| 0.1 | 10 | 8 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 11 | 10 | 10 |
| 0.2 | 20 | 25 | 21 | 23 | 23 | 21 | 18 | 21 | 26 | 25 | 22 | 23 | 22.55 |
| 0.3 | 30 | 22 | 41 | 34 | 28 | 30 | 23 | 26 | 23 | 28 | 26 | 28 | 28.09 |
| 0.4 | 40 | 40 | 29 | 41 | 47 | 45 | 33 | 39 | 33 | 41 | 39 | 40 | 38.82 |
| 0.5 | 50 | 58 | 45 | 47 | 70 | 57 | 55 | 65 | 36 | 40 | 47 | 47 | 51.55 |
| 0.6 | 60 | 47 | 77 | 47 | 61 | 67 | 51 | 61 | 114 | 93 | 68 | 67 | 68.45 |
| 0.7 | 70 | 91 | 89 | 70 | 51 | 86 | 60 | 90 | 51 | 106 | 54 | 70 | 74.36 |
| 0.8 | 80 | 121 | 139 | 62 | 78 | 124 | 87 | 75 | 52 | 125 | 119 | 87 | 97.18 |
| 0.9 | 90 | 126 | 66 | 87 | 84 | 134 | 73 | 106 | 117 | 148 | 63 | 106 | 100.91 |
| 1 | 100 | 120 | 126 | 102 | 107 | 79 | 89 | 157 | 56 | 70 | 81 | 102 | 99 |

TABLE VIII.
PERFORMANCE WITH INVERTED BLOCK INTERCHANGE ($n = 100$)

| $\alpha$ | $\alpha n$ | Calculated Operations in different test cases | | | | | | | | | | | Average # of calculated operation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
| 0.1 | 10 | 8 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | 11 | 10 | 10 |
| 0.2 | 20 | 25 | 21 | 23 | 23 | 21 | 18 | 21 | 26 | 25 | 22 | 23 | 22.55 |
| 0.3 | 30 | 22 | 41 | 34 | 28 | 30 | 23 | 26 | 23 | 28 | 26 | 28 | 28.09 |
| 0.4 | 40 | 40 | 29 | 41 | 47 | 45 | 33 | 39 | 33 | 41 | 39 | 40 | 38.82 |
| 0.5 | 50 | 58 | 45 | 47 | 70 | 57 | 55 | 65 | 36 | 40 | 47 | 47 | 51.55 |
| 0.6 | 60 | 47 | 77 | 47 | 61 | 67 | 51 | 61 | 114 | 93 | 68 | **65** | **68.27** |
| 0.7 | 70 | 91 | 89 | 70 | 51 | 86 | 60 | 90 | 51 | 106 | 54 | 70 | 74.36 |
| 0.8 | 80 | 121 | 139 | 62 | 78 | 124 | 87 | **73** | **51** | 125 | 119 | **86** | **96.82** |
| 0.9 | 90 | 126 | 66 | **86** | 84 | **132** | **71** | 106 | 117 | **146** | **62** | 106 | **100.18** |
| 1 | 100 | 120 | 126 | 102 | 107 | 79 | 89 | 157 | 56 | 70 | 81 | 102 | 99 |

*Information and Computation*, vol. 204, no. 2, pp. 275–290, 2006.

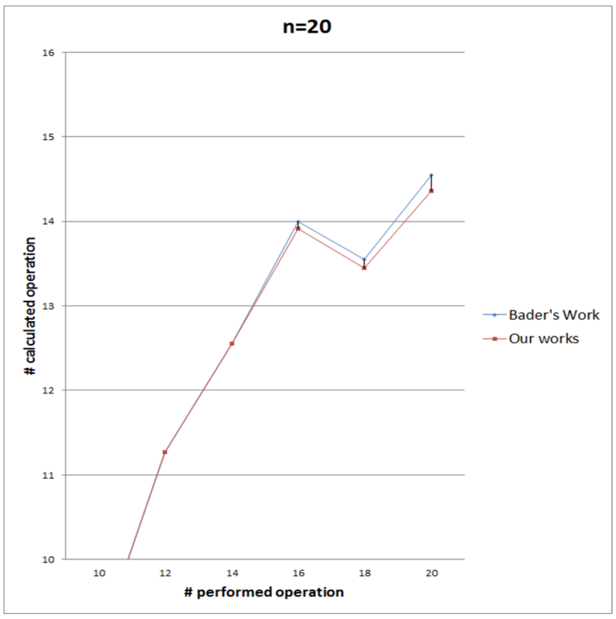
[6] I. Elias and T. Hartman, "A 1.375-approximation algorithm for sorting by transpositions," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 3, no. 4, pp. 369–379, 2006.

[7] T. Hartman and R. Sharan, "A 1.5-approximation algorithm for sorting by transpositions and transreversals," *Journal of Computer and System Sciences*, vol. 70, no. 3, pp. 300–320, 2005.

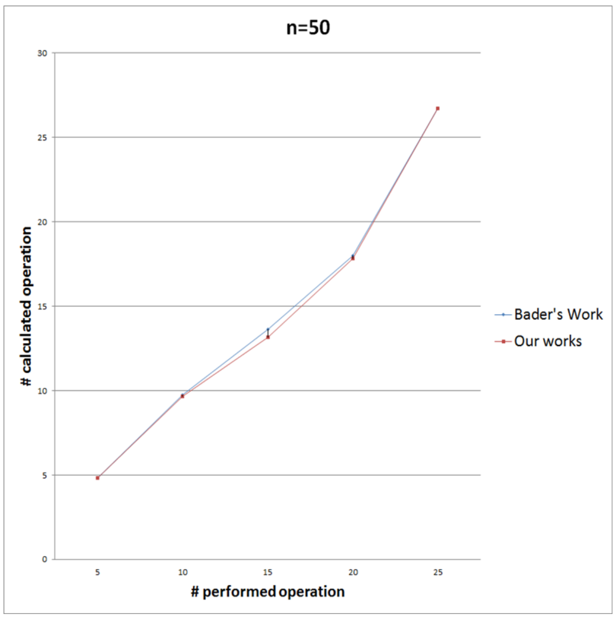
[8] S. Yancopoulos, O. Attie, and R. Friedberg, "Efficient sorting of genomic permutations by translocation, inversion and block interchange," *Bioinformatics*, vol. 21, no. 16, pp. 3340–3346, 2005.

[9] Y. C. Lin, C.-Y. Lin, and C. R. Lin, "Sorting by reversals and block-interchanges with various weight assignments," *BMC Bioinformatics*, vol. 10, p. 398, Dec. 2009.

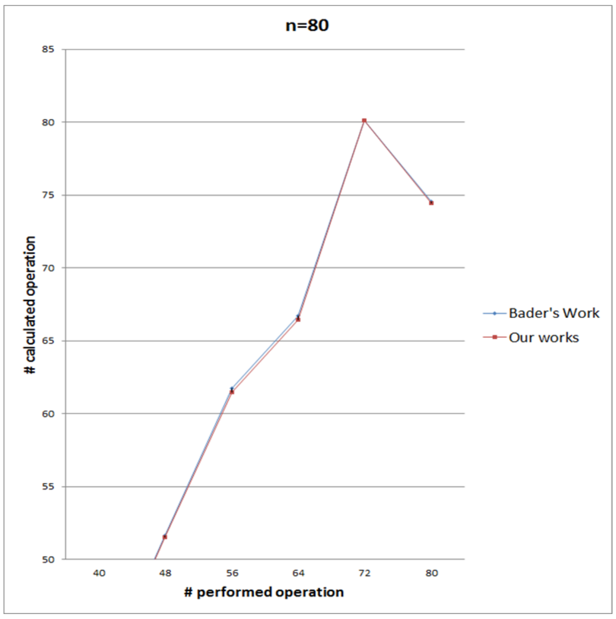
[10] M. Bader, "Sorting by reversals, block interchanges, tan-



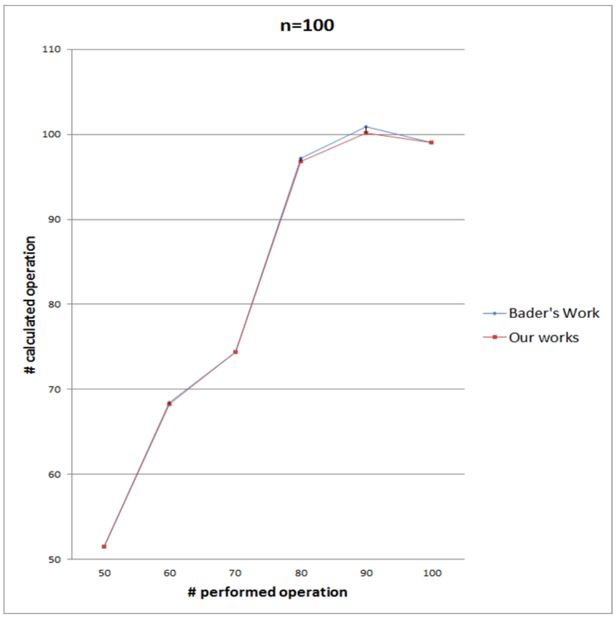
(a)



(b)



(c)



(d)

Figure 5. Performance Comparison

dem duplications and deletions," *BMC Bioinformatics*, vol. 10, no. S1, Jan. 2009.

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

[12] L. Bulteau, G. Fertin, and I. Rusu, "Sorting by transpositions is difficult," in *ICALP (1)*, 2011, pp. 654–665.

[13] G. Blanc, A. Barakat, R. Guyot, R. Cooke, and M. Delseny, "Extensive duplication and reshuffling in the arabidopsis genome," *The Plant Cell*, vol. 12, pp. 1093–1101, 2000.

[14] D. Sankoff, "Gene and genome duplication," *Current Opinion in Genetics and Development*, vol. 11, pp. 681–684, 2001.

[15] D. A. Christie and R. W. Irving, "Sorting strings by reversals and by transpositions," *SIAM Journal on Discrete Mathematics*, vol. 14, no. 2, pp. 193–206, 2001.

[16] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang, "The assignment of orthologous genes via genome rearrangement," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 2, no. 4, pp. 302–315, 2005.

[17] D. Bertrand, M. Lajoie, N. El-Mabrouk, and O. Gascuel, "Evolution of tandemly repeated sequences through duplication and inversion," in *Proceedings of the RECOMB 2006 international conference on Comparative Genomics*, ser. RCG'06, vol. 4205. Springer-Verlag, 2006, pp. 129–140.

[18] N. El-Mabrouk, "Sorting signed permutations by reversals and insertions/deletions of contiguous segments," *Journal of Discrete Algorithms*, vol. 1, pp. 105–122, 2001.

[19] M. Marron, K. M. Swenson, and B. M. E. Moret, "Genomic distances under deletions and insertions," *Theoretical Computer Science*, vol. 325, no. 3, pp. 347–360, 2004.

[20] K. Swenson, M. Marron, J. Earnest-Deyoung, and B. Moret, "Approximating the true evolutionary distance between two genomes," *ACM J. Experimental Algorithmics*, vol. 12, pp. 1–17, 2008.

[21] D. Sankoff, "Genome rearrangement with gene families," *Bioinformatics*, vol. 15, pp. 909–917, 1999.

[22] N. El-Mabrouk, "Reconstructing an ancestral genome using minimum segments duplications and reversals," *Journal of Computer and System Sciences*, vol. 65, pp. 442–464, 2002.

[23] Z. Fu, X. Chen, V. Vacic, P. Nan, Y. Zhong, and T. Jiang, "A parsimony approach to genome-wide ortholog assignment," in *Proceedings of the 10th annual international conference on Research in Computational Molecular Biology*, ser. RECOMB'06. Springer-Verlag, 2006, pp. 578–594.

[24] S. Yancopoulos and R. Friedberg, "Sorting genomes with insertions, deletions and duplications by dcj," in *Proceedings of the international workshop on Comparative Genomics*, ser. RECOMB-CG '08. Springer-Verlag, 2008, pp. 170–183.

[25] N. El-Mabrouk, J. H. Nadeau, and D. Sankoff, "Genome halving," in *Combinatorial Pattern Matching*. Springer, 1998, pp. 235–250.

[26] N. El-Mabrouk and D. Sankoff, "The reconstruction of doubled genomes," *SIAM Journal on Computing*, vol. 32, no. 3, pp. 754–792, 2003.

[27] R. Warren and D. Sankoff, "Genome halving with double cut and join," in *APBC*, 2008, pp. 231–240.

[28] J. Mixtacki, "Genome halving under dcj revisited," in *Proceedings of the 14th annual international conference on Computing and Combinatorics*, ser. COCOON '08. Springer-Verlag, 2008, pp. 276–286.

[29] M. Ozery-Flato and R. Shamir, "On the frequency of genome rearrangement events in cancer karyotypes," in *Proc 1st RECOMB Satellite Workshop in Computational Cancer Biology*, 2007, p. 17.

[30] M. Bader, "Genome rearrangements with duplications," *BMC Bioinformatics*, vol. 11, no. S-1, p. 27, 2010.

[31] M. D. V. Braga, R. Machado, L. C. Ribeiro, and J. Stoye, "On the weight of indels in genomic distances," *BMC Bioinformatics*, vol. 12 Suppl 9, 2011.

**Deen Md Abdullah** received his B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Military Institute of Science and Technology (MIST), Dhaka, Bangladesh, in 2010. He is currently an M.Sc. student at IICT, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. He is a lecturer in the Department of CSE, Primeasia University, Dhaka, Bangladesh. His research interests include Bioinformatics, Artificial Intelligence, Meta-heuristics and Networks.

**Wali Md Abdullah** received his B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Military Institute of Science and Technology (MIST), Dhaka, Bangladesh, in 2010. He is currently an M.Sc. student at IICT, Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. He is a lecturer in the Department of CSE, MIST. His research interests include Meta-heuristics, Networks, Bioinformatics and Artificial Intelligence.

**M. Sohel Rahman** received his B.Sc. Engg. degree from the Department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, in 2002 and the M.Sc. Engg. degree from the same department, in 2004. He received his Ph.D. degree from the Department of Computer Science , King's College London, University of London, London, UK in 2008. He is currently an Associate Professor in the Department of CSE, BUET. His research interests include String and sequence algorithms, Bioinformatics, Musicolgy, Design and analysis of Algorithms, Meta-heuristics etc.