Sorting Permutations by Reversals through a Hybrid Genetic Algorithm based on Breakpoint Elimination and Exact Solutions for Signed Permutations

José Luis Soncco Álvarez
Department of Computer Science
University of Brasilia
Brasilia, D.F., Brazil
Email: josesoal@hotmail.com

Mauricio Ayala-Rincón
Departments of Computer Science and
Mathematics, University of Brasilia
Brasilia, D.F., Brazil
Email: ayala@unb.br

***Abstract*—Sorting permutations by reversals is one of the most challenging problems related with the analysis of the evolutionary distance between organisms. Genome rearrangement can be done through several operations with biological significance, such as block interchange, transposition and reversal, among others; but sorting by reversals, that consists in finding the shortest sequence of reversals to transform one genome into another, came arise as one of the most challenging problems from the combinatorial and algebraic points of view. In fact, sorting by reversal unsigned permutations is a $\mathcal{NP}$-hard problem, for which the question of $\mathcal{NP}$-completeness remains open for more than two decades and for which several interesting combinatorial questions, such as the average number of reversals needed to sort permutations of the same size, remain without solution. In contrast to the unsigned case, sorting by reversals signed permutations belongs to $\mathcal{P}$. In this paper, a standard genetic algorithm for solving the problem of sorting by reversals unsigned permutations is proposed. This approach is based on Auyeung and Abraham's method which uses exact solutions for the signed case in order to build approximate solutions for the unsorted one. Additionally, an improved genetic algorithm is proposed, that in the initial generations applies reversals that simultaneously eliminate two breakpoints, a heuristic mechanism used by several approximation algorithms. As control mechanism for estimating the precision of the results, a correct implementation of an 1.5-approximation algorithm was developed. Also, the results were compared with permutations for which exact solutions are known, such as Gollan's permutations and their inverses. Several experiments with randomly generated permutations were performed and the results showed that in average the precision of the outputs provided by both the standard and improved genetic algorithms overcome the results given by the 1.5-approximation algorithm as well as those results provided by previous known genetic approaches.**

I. Introduction

Comparison of biological sequences is a relevant problem in bioinformatics for determining the evolutionary relationships between organisms. This problem can be addressed using algorithms that take into consideration local mutations (deletions, insertions and substitutions), such as the classical algorithm of dynamic programming to align two DNA sequences. But when one tries to understand how genetic sequences mutate at the chromosome level, it is necessary to consider global operations such as reversals, block interchanges and transpositions. One of the operations that occurs commonly in genome rearrangements is the *reversal* of a substring.

The order of genes in a genome can be represented in string notation as a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the set $\{1, \ldots, n\}$, that is a bijective function from $\{1, \ldots, n\}$ into itself, where n is the number of genes. Two different types of permutations have received attention from the biological point of view: signed and unsigned permutations. On unsigned permutations, genes are abstracted without any orientation, while on signed permutations, each π_i has a positive or negative sign reflecting its orientation within the genome; either from left to right or from right to left, denoted respectively as $\overrightarrow{\pi_i}$ or $\overleftarrow{\pi_i}$.

Given two permutations we wish to determine the minimum number of reversals to transform one permutation into another, that is the reversal distance between two permutations. By simple algebraic properties of permutations, this problem results equivalent to the problem of determining the minimum number of reversals to transform one permutation into the identity permutation, denoted as $\imath$ (i.e., in string notation, for the unsigned case, the permutation sorted in increasing order and for the signed case the permutation sorted in increasing order and in which each π_i has positive orientation, $\overrightarrow{\pi_i}$). This problem is known as *sorting by reversals*.

The problem of sorting by reversals has been extensively studied both in the field of combinatorics of permutations and in bioinformatics for decades. For the case of sorting signed permutations by reversals, initially, Kececioglu and Sankoff [1] conjectured that the problem was $\mathcal{NP}$-hard and proposed a 2-approximation algorithm. Afterwards, Bafna and Pevzner [2] improved the approximation ratio to 1.5, by using the data structure of breakpoint graphs. Finally, Hannenhalli and Pevzner [3] gave an exact polynomial ($O(n^4)$) algorithm that computes the reversal distance without providing the sequence of sorting reversals, by using the data structure of overlap graph. Further, more efficient algorithms based on this polynomial algorithm ([3]) have been introduced; among them, the $O(n\alpha(n))$ time algorithm proposed by Berman and Hannenhalli in [4], where $\alpha(n)$ is the inverse of Ackermann function and, the linear time algorithm proposed by Bader, Moret and Yang in [5] that uses a new data structure called overlap forest. Increasing these complexities, these algorithms can be applied not only to compute the reversal distance, but also to build a minimal sequence of reversals.

For unsigned permutations, that are the ones treated in this paper, the problem was shown to be $\mathcal{NP}$-hard by Caprara [6]. Before the complexity was known, Kececioglu and Sankoff [1] gave a 2-approximation algorithm, and Bafna and Pevzner[2] presented an 1.75-approximation algorithm. Later on, the approximation ratio was improved to 1.5 by Christie [7] and then to 1.375 by Berman, Hannenhalli and Karpinski [8]. The latter approximation algorithm is of theoretical interest being its practical implementation of great difficulty.

Evolutionary techniques like genetic algorithms were proposed to deal with the unsigned case due to the complexity of the problem. Auyeung and Abraham [9] suggested a genetic algorithm (GA) approach to solve the problem of sorting unsigned permutations by reversals based on mapping unsigned permutations of size n into a subset of the 2^n possible signed versions of each permutation of size n. For a given unsigned permutation, a set of signed permutations is generated by randomly assigning either a positive or a negative parity to each component of the permutation. The exact solution of one of this signed permutations corresponds to a feasible solution of the original unsigned permutation. The fitness function of each signed permutation is given by its exact reversal distance that is computed by Hannenhalli's et al polynomial time algorithm.

Subsequently, modifications to Auyeung and Abraham's method were reported in [10], but without changing the central premisses of this approach. More recently, Ghaffarizadeh, Ahmadi and Flann [11] proposed a modified version of the standard GA using individuals of different sizes to reduce the runtime of the algorithm. All these approaches have been reported to improve the results obtained applying Christie's 1.5-approximation algorithm in order to control the precision of the solutions. But two problem arise in these papers: firstly, Christie's approximation algorithm presented some conceptual problems whose solutions were not reported in [9], [11] or [10], and secondly, because of the given results, the way in which permutations were randomly generated in some of these papers is unclear as we will discuss. In [12], the authors, presented a simple GA approach that is based on heuristics of well-known approximation mechanisms (e.g., [7], [8]) focused in applying in each generation reversals that maximally reduce the number of breakpoints appearing in each individual. A fixed version of Christie's approach was initially reported by the authors in [12] and used as quality control mechanism, as is done in this paper too. Additionally, experiments were performed over random permutations generated in two different manners: by direct application of the `C` language `rand` function and by randomly applying reversals to the identity permutation.

In this paper we propose a GA based on the Auyeung and Abraham's method. Firstly, for an approach *à la* Auyeung and Abraham in which we established in a precise manner the parameters of the GA, the experiments gave results that improved the ones previously reported and that are better than the approximate solutions given by the fixed 1.5-approximation algorithm. Secondly, a hybrid GA-approximate modification is proposed in which, initially, the input permutation is sorted by randomly applying all possible reversals that simultaneously eliminate two breakpoints and then, the GA approach is applied for the obtained permutation. This modification improves our results when the permutations are large. The differences between our solutions and the approximated ones is smaller than the differences previously reported in other papers, but our comparison is more precise since here a fixed 1.5-approximation algorithm is applied. Finally, experiments were performed with permutations for which exact solutions are known such as Gollan's permutations and their inverses for which both GA approaches compute exact outputs.

The paper is organized in the following sections: in Section II, the necessary notations and notions are given; in Section III, the proposed GA and its modification are presented; in Section IV, experiments and results are given; in Section V, the method and the results are discussed and; finally, concluding remarks and future work are presented in Section VI.

II. Background

A. Terminology

Most definitions and terminology presented in this section, were introduced by Bafna and Pevzner in their seminal paper [2].

A permutation in the symmetry group S_n is a bijection π from $\{1, \ldots, n\}$ into itself. A permutation π, denoted in string notation as $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ is extend by adding an initial and a final pivot, $\pi_0 = 0$ and $\pi_{n+1} = n+1$. A *reversal* $\rho^{i..j}$ of an interval $[i, j]$, for $1 \leq i \leq j \leq n$, transforms the extended permutation π into

$$\pi' = (\pi_0, \ldots, \pi_{i-1}, \pi_{\mathbf{j}}, \ldots, \pi_{\mathbf{i}}, \ldots, \pi_{n+1})$$

For example, consider the permutation

$$\pi = (0, 3, \mathbf{1}, \mathbf{5}, \mathbf{2}, 4, 6)$$

The reversal $\rho^{2..4}$ transforms π into

$$\pi' = (0, 3, \mathbf{2}, \mathbf{5}, \mathbf{1}, 4, 6)$$

Note that the reversal reverts the interval $[2, 4]$ of π.

Observe that a reversal is also a permutation in S_n:

$$(\rho^{i..j})_k = \begin{cases} k, \text{ if } k < i \text{ or } k > j; \\ i + (j - k), \text{ if } i \leq k \leq j. \end{cases}$$

Thus, in postfix notation $\pi\rho^{i..j}$ denotes the application of the reversal $\rho^{i..j}$ to the permutation π.

Given two permutations π and σ, the *reversal distance problem* is the problem of finding a shortest sequence of reversals needed to transform π into σ. The *reversal distance* between π and σ is the minimum number of reversals required to transform π into σ.

By simple algebraic properties of symmetry groups, the reversal distance between π and σ is equal to the reversal distance between $\sigma^{-1}\pi$ and the identity permutation $\imath$. In fact, notice that if $\rho_1 \ldots \rho_k$, is a sequence of (reversal) permutations that transforms π into σ, then it holds that $\pi\rho_1 \ldots \rho_k = \sigma$, if and only if $(\sigma^{-1}\pi)\rho_1 \ldots \rho_k = \sigma^{-1}\sigma = \imath$. Thus, the problem of sorting by reversals corresponds to find the reversal distance between a permutation π and the identity permutation $\imath$, that is denoted as $d(\pi)$. Since $\imath$ is the sorted permutation, the reversal distance problem is also known as the problem of *sorting by reversals*.

Let $i \sim j$ denote the property $|i - j| = 1$. Given two consecutive elements π_i and π_j of π, for $0 < i < n+1$ and either $j = i - 1$ or $j = i + 1$,

- they are said to be *adjacent* if $\pi_i \sim \pi_j$ and
- they are said to form a *breakpoint* if $\pi_i \nsim \pi_j$.

Observe that the identity permutation is the unique permutation without breakpoints. The number of breakpoints in π is denoted by $b(\pi)$.

Let ρ be a reversal that transforms π into π', then it is easy to observe that $b(\pi) - b(\pi') \in \{-2, -1, 0, 1, 2\}$. Reversals that reduce the number of breakpoints by i, are called i-reversals.

Given a permutation π, one defines the *cycle graph* of π (also called as *breakpoint graph*) ,$G(\pi)$ as a undirected edge-colored graph derived from the adjacency and breakpoint relations in π with $n+2$ vertices labeled by $0, 1, \ldots, n, n+1$. Two vertices i and j are joined by a *black edge* if (i, j) is a breakpoint of π. Two vertices i and j are joined by a *gray edge* if $i \sim j$ and i, j are not consecutive in π. An example of a cycle graph is shown in Fig.1.

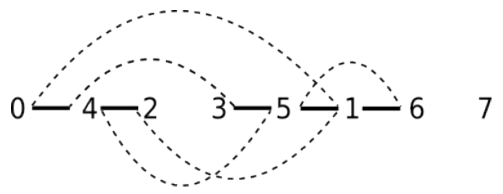

Fig. 1. Cycle graph $G(\pi)$ for the permutation $\pi = (4, 2, 3, 5, 1, 6)$

Note that for all permutations π, $G(\pi)$ can be completely decomposed into disjoint cycles of alternated colored edges, since each node has an equal number of black and gray incident edges. However, there are probably many different cycle decompositions of $G(\pi)$ of alternated colored edges. For simplicity, cycles of alternating colored edges will be called either alternating cycles or simply cycles. The graph in Fig. 1 decomposes either in one or two cycles. The maximum number of cycles in a cycle decomposition of $G(\pi)$, denoted as $c(\pi)$, provides an useful bound for the reversal distance [2]: $d(\pi) \geq b(\pi) - c(\pi)$. The observation that this bound is in fact close to the reversal distance (mostly for signed permutation) motivated the development of approximation algorithms that are based on the elimination of breakpoints.

For $\pi = (4, 2, 3, 5, 1, 6)$, whose cycle graph is presented in Fig. 1, $d(\pi) \geq 5 - 2 = 3$. Eliminating two breakpoints, by applying the 2-reversal $\rho^{2..4}$, one obtains the permutation $\pi' = (4, 5, 3, 2, 1, 6)$, whose cycle graph is presented in Fig. 2, and for which this bound gives $d(\pi') \geq 3 - 1 = 2$.

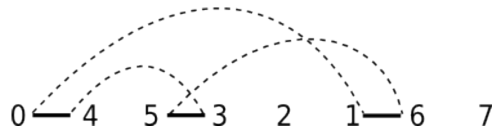

Fig. 2. Cycle graph $G(\pi')$ for the permutation $\pi' = (4, 5, 3, 2, 1, 6)$

Observe that no 2-reversal is applicable to π'. Applying the 1-reversal $\rho^{1..2}$, one obtains $\pi'' = (5, 4, 3, 2, 1, 6)$, for which the bound gives $d(\pi'') = 2 - 1 = 1$. See Fig. 3. Finally, π'' can be sorted by applying the 2-reversal $\rho^{1..5}$.

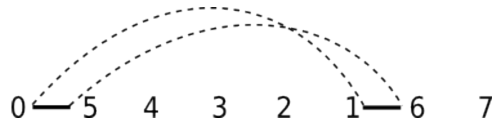

Fig. 3. Cycle graph $G(\pi'')$ for the permutation $\pi'' = (5, 4, 3, 2, 1, 6)$

III. The genetic algorithm for sorting unsigned permutations by reversals

Our method is based on the approach introduced in [9]. The search space consists of 2^n signed permutations, that are the signed versions of the initial unsigned permutation to be sorted. For example let the initial unsigned permutation be $\pi = (2, 1)$, the search space would be $\{(\overrightarrow{2}, \overrightarrow{1}), (\overrightarrow{2}, \overleftarrow{1}), (\overleftarrow{2}, \overrightarrow{1}), (\overleftarrow{2}, \overleftarrow{1})\}$. Signed permutations can be sorted polynomially and one knows that an optimal solution that solves any signed permutation of the search space also solves the initial unsigned permutation. So, one of the optimal solutions of the elements of the search space is an optimal solution for the initial unsigned permutation. This fact guides the proposed genetic algorithm.

The standard GA approach is used. Initially, a random population of signed permutations for the input permutation is generated; after that, for each generation, the reproduction is performed in the following manner: two individuals of the population are taken for which crossover and mutation operations are applied producing two offspring. Then, the two new obtained offspring are returned to the population. The GA finishes after all the generations have been completed.

The developed improvement made over the standard GA approach consists of applying just to the initial unsigned permutation reversals that eliminate simultaneously two breakpoints, that is 2-reversals. This is done randomly until no additional 2-reversals are applicable. Although this is a greedy mechanism that not necessarily will produce an optimal sorting, giving priority to the elimination of (2-)breakpoints has been applied as local optimization method in several approximation algorithms because a high number of 2-reversals tend to be present in optimal solutions (e.g., [1], [2], [7], [8]).

The four stages of the breeding cycle of the GA are described below.

The selection is the stage where all the population is sorted by their fitness value in order to make it easier to choose individuals for the crossover. It is also here where we save the best solution found.

The crossover chooses the best individuals of the population and makes them breed, producing an offspring.

The mutation is applied over the offspring produced by the crossover operator. The mutation should not occur very often because it could alter too much some of the individuals of the offspring that represent good solutions.

The replacement is the last stage where the offspring must return to the population, replacing the worst individuals.

The value of the fitness is the optimal number of reversals for sorting the signed permutation that represents each individual. We use the implementation of the exact algorithm proposed, and provided by the authors, in [5] to calculate the fitness of each individual in the population. With his algorithm the reversal distance of signed permutations is computed in linear running time.

The fact that the bound $d(\pi) \geq b(\pi) - c(\pi)$ was observed to be very close to the reversal distance for signed permutations gave rise to other bounds. In [3] the concept of *hurdles*, denoted as $h(\pi)$ was developed and proved that $b(\pi) - c(\pi) + h(\pi) \leq d(\pi) \leq b(\pi) - c(\pi) + h(\pi) + 1$. The proof of this bound requires additional elaborated notions such as the ones of *fortress* and this development, started in [3], is the basis of the development of polynomial algorithms for solving the problem of sorting by reversals signed permutation.

The pseudo-code of our proposed genetic algorithms is shown in Algorithm 1.

Algorithm 1: Sorting by Reversals' GA with parameter for applying the GA improved version

```
Input: An unsigned permutation $\pi$, a boolean "improve"
       indicating whether or not to apply initially
       two-reversals
Output: A number of reversals to sort the permutation $\pi$
1  if "improve" then
2      apply 2-reversals until no reversals that eliminate two
       breakpoints are applicable updating $\pi$;
3  generate an initial population of signed permutations for
   $\pi$;
4  evaluate fitness for each individual of the initial
   population;
5  for $i = 2$ to number of generations do
6      selection;
7      crossover;
8      mutation;
9      evaluate fitness of offspring;
10     replacement;
```

Let n be the length of the input permutation. The initial population size is fixed as $n \log n$. Each individual of the population was generated from the input permutation in linear time by randomly assigning a sign to each of its elements. Thus, generating the initial population takes running time $O(n^2 \log n)$.

The algorithm used to sort the population by their fitness, in the selection stage, is the *counting sort* that is well-known to take running time $O(n \log n)$ since we have to order $n \log n$ elements.

In the crossover stage, the best individuals of the population are chosen to be the parents. For each pair of parents the crossover is done in linear time generating two offspring. So the total running time for executing the crossover stage is $O(n^2 \log n)$.

In the mutation stage, mutation is applied to each offspring produced by the crossover. For each element of the individuals in the offspring, it should be checked whether the mutation occurs or not. The total time taken by the mutation is $O(n^2 \log n)$.

Evaluating the fitness values and making the necessary replacements of individuals take running time $O(n^2 \log n)$ each.

The genetic algorithm finishes after n generations, then the overall time complexity is $O(n^3 \log n)$.

IV. Experiments and Results

In order to validate the proposed GA approaches several tests were performed. Tests were done for permutations for which, it is known the reversal distance and for randomly generated permutations. For the latter ones, test differs in the way the input permutations were generated, but in general the tests share the following characteristics.

Hundred permutations were generated for each length $i \in \{10, 20, 30, ..., 150\}$. For each length i, the average of the results over the hundred generated permutations for the 1.5-approximation algorithm was calculated. Also, averages for the standard GA and the improved GA were calculated. It is worth mentioning that for a given permutation the standard and improved GAs were executed ten times and then, the average of the ten obtained results was calculated. This average represents the result for each permutation. Moreover, for each set of hundred permutations of length i, the standard deviation was calculated, that represents how far the results are from the average.

The crossover operation was performed with a single point crossover. The crossover rate used is 0.9 and the mutation rate used is 0.02. The selection of parents for the crossover was made over the 60% of the population that represent the best individuals. The replacement of the offspring was made over 60% of the population that represent the worst individuals.

The 1.5-approximation algorithm and the standard and improved GAs were implemented in C language and executed in OS X platforms with Intel core I5, I7 processors and other similar platforms.

A. Experiments with Gollan's permutations

This experiment was done as a control mechanism considering the most difficult unsigned permutations to be sorted by

TABLE I
RESULTS OF THE 1.5-APPROXIMATION ALGORITHM AND THE STANDARD AND IMPROVED GAS FOR GOLLAN'S PERMUTATIONS

n	Size of pop.	Avg. 1.5-approx.	Avg. Standard GA	Avg. Improved GA
		γ_n/γ_n^{-1}	γ_n/γ_n^{-1}	γ_n/γ_n^{-1}
10	33	**9/11**	9/9	9/9
20	86	19/19	19/19	19/19
30	147	**29/31**	29/29	29/29
40	212	39/39	39/39	39/39
50	282	**49/51**	49/49	49/49
60	354	59/59	59/59	59/59
70	429	**69/71**	69/69	69/69
80	505	79/79	79/79	79/79
90	584	**89/91**	89/89	89/89
100	664	99/99	99/99	99/99
110	745	**109/111**	109/109	109/109
120	828	119/119	119/119	119/119
130	912	**129/131**	129/129	129/129
140	998	139/139	139/139	139/139
150	1084	**149/151**	149/149	149/149

reversals, that are Gollan's permutations and their inverses. In [2] it was proved that in the symmetry group S_n, only both Gollan's permutation, denoted as γ_n, and its inverse, γ_n^{-1}, need exactly $n-1$ reversals to be sorted. All other permutations in S_n can be sorted with less than $n-1$ reversals.

Gollan's permutation, in cycle notation, is defined as follows:

$$\gamma_n = \begin{cases} (1\,3\,5\,7...n-1\,n...8\,6\,4\,2), & \text{for } n \text{ even,} \\ (1\,3\,5\,7...n\,n-1...8\,6\,4\,2), & \text{for } n \text{ odd.} \end{cases}$$

For instance, $\gamma_{10} = (1\,3\,5\,7\,9\,10\,8\,6\,4\,2)$ and $\gamma_{10}^{-1} = (2\,4\,6\,8\,10\,9\,7\,5\,3\,1)$.

Let i and j be two consecutive elements of Gollan's permutation γ_n or its inverse γ_n^{-1} in cycle notation, the corresponding permutation in string notation is generated putting the element i in the position j and, in addition, the last element is placed in the position given by the first element. For instance, γ_{10} in string notation is given by $(2, 4, 1, 6, 3, 8, 5, 10, 7, 9)$ and its inverse γ_n^{-1} by $(3, 1, 5, 2, 7, 4, 9, 6, 10, 8)$.

Since exact solutions are known, we used these permutations in order to control validity of the outputs provided by our implementations of the 1.5-approximation algorithm and the GAs.

The results of this experiment are presented in Table I. The values before the slash are the results for γ_n and after, for its inverse. Observe that all answers given by both the standard and the improved GA are exact, while the 1.5-approximate algorithm fails to compute exact solutions for γ_n^{-1}, for $n = 10, 30, 50, 70, 90, 110, 130$ and 150.

B. *Experiments with randomly generated permutations*

For this experiment we generated permutations of size n applying repeatedly the function `rand` available in the standard library of the language `C` in the following way: π_1 is generated as a random number between 1 and n and then, for each $1 < i \leq n$, π_i is generated as a random number between 1 and n excluding $\{\pi_1, \ldots, \pi_{i-1}\}$. Assuming that in fact `rand` generates each random selection independently, this mechanism generates each possible permutation correctly with probability $1/n!$

The results of this experiment are shown in Table II

Additional experiments were performed with permutations of size n generated starting from the identity permutation and then, applying n random reversals. This was done in order to obtain additional information to compare our results with related ones. The results of this experiment are shown in Table III. The motivation to perform this experiment is that the average number of reversals to sort by reversals permutations of length n is unknown and related works have used permutations as inputs that provide results that suggest different averages.

V. DISCUSSION AND COMPARISON WITH RELATED WORK

As previously mentioned from Table I we can see that the results given by both the standard and improved GAs are exact for Gollan permutations and their inverses, in contrast to a few inexact results obtained by the 1.5-approximation algorithm. The unique previous work that reports outputs for specific permutations was [10] in which the permutation in S_{36}

$$(12, 31, 34, 28, 26, 17, 29, 4, 9, 36, 18, 35, 19, 1, 16, 14, 32, 33, \\ 22, 15, 11, 27, 5, 20, 13, 30, 23, 10, 6, 3, 24, 21, 8, 25, 2, 7)$$

was sorted with 26 reversals as it was also done by both our GA approaches.

A few considerations are necessary before discussing our results for randomly generated permutations. Here it is relevant to stress that not much information is know about the form of solutions of this problem. In fact, several properties of the outputs that are known for other sorting operations different from reversals are unknown; for instance, the average number of reversals to sort unsigned permutations of S_n is an interesting open question, while this number is known, for example, for the problem of sorting by block interchange [13].

Surprisingly, previous related works report contrasting results on randomly generated permutations. From the obtained results in [9], [10] and [11], respectively the following percents of number of reversals over the length of permutations needed

TABLE II
RESULTS OF THE 1.5-APPROXIMATION AND THE STANDARD AND IMPROVED GAS WITH RANDOM PERMUTATIONS

n	Size of pop.	Avg. 1.5-approx.	Avg. Std GA	Avg. Improved GA	Std Deviation
10	33	5.84	5.83	5.98	0.0685
20	86	13.69	13.28	13.36	0.1775
30	147	21.88	21.08	21.07	0.3795
40	212	30.27	29.05	29.08	0.5682
50	282	39.64	37.53	37.46	1.0116
60	354	48.45	45.75	45.67	1.2921
70	429	57.56	54.28	54.17	1.5728
80	505	66.66	62.72	62.59	1.8887
90	584	75.86	71.65	71.54	2.0110
100	664	85.93	80.82	80.78	2.4184
110	745	94.03	88.91	88.87	2.4231
120	828	104.37	98.59	98.52	2.7414
130	912	113.38	107.46	107.37	2.8122
140	998	123.15	117	116.92	2.9182
150	1084	132.76	126.53	126.51	2.9416

TABLE III
RESULTS OF THE 1.5-APPROXIMATION AND THE GAS WITH PERMUTATIONS OBTAINED BY APPLYING n RANDOM REVERSALS TO $\imath$

n	Size of pop.	Avg. 1.5-approx.	Avg. Std GA	Avg. Improved GA	Std Deviation
10	33	5.3	5.27	5.34	0.0287
20	86	12.01	11.72	11.77	0.1266
30	147	19.37	18.51	18.57	0.3920
40	212	27.2	25.82	25.85	0.6436
50	282	33.96	32.39	32.39	0.7401
60	354	42.12	39.72	39.69	1.1385
70	429	49.75	46.65	46.62	1.4685
80	505	57.3	53.95	53.88	1.5960
90	584	65.85	61.79	61.71	1.9330
100	664	73.64	69.26	69.11	2.1010
110	745	81.8	76.58	76.54	2.4702
120	828	89.34	84.23	84.15	2.4280
130	912	98.26	92.5	92.37	2.7464
140	998	105.49	99.8	99.72	2.7013
150	1084	114.04	108.04	107.98	2.8427

to sort permutations of medium sizes (lengths between 50 and 150) are suggested: $\sim 83\%, \sim 46\%$ and $\sim 80\%$. Namely, the manner in which input random permutations are generated in [10] was not reported in that paper, but because of the results from the other works, including the current one, the average number of reversals needed to sort permutations of length n is much greater than $n/2$, which suggest us the authors of that work have applied a very peculiar mechanism for the generation of input permutations.

Additionally, as previously mentioned, previous works compare GA results with implementations of approximation algorithms without reporting problems in Christie's original 1.5 approximation ratio proposal [7], initially fixed in [12]. For example, the approximate outputs in [9] for permutation of medium sizes (lengths between 50 and 150) suggest a number of reversals of $\sim 94\%$ for sorting permutations of size n, while the fixed 1.5-algorithm used in this paper (as well as in [12]) of $\sim 87\%$, allowing in this way a fair analysis of the real improvement over the approximation algorithm obtained with the corresponding GA approaches.

The results for randomly generated permutations in the table II can be compared with the ones obtained by the simple GA in [12] and are illustrated in the bars Fig.4. Results reported in [12] are close to the ones provided by the fixed 1.5-approximation algorithm and both the standard and improved GA approaches proposed in this paper provide better results than the 1.5-approximation algorithm.

The results for permutations obtained by applying random reversal to the identity presented in the table III can only be compared with precision with those presented in [9], because it is the only work that presents a table with numeric results. Comparisons with results in [11] are presented, but they are imprecise because instead numeric values, the authors presented only a graphic image (restricted to permutations of length less than or equal to 110) from which approximate numerical values were extracted and included in the figure 5. In this figure, graphs are built for the approximate mechanism applied in [9] and [11] and Auyeung's GA approach using numerical results taken directly from [9]. It can be observed that all approaches perform better than the approximate method presented in [9] and [11], but only the results of the (standard) and improved GA approaches presented in this paper are better than the results provided by the fixed 1.5-approximation algorithm. In this point we observe that the crossover and mutation rates used here are 0.9 and 0.02, respectively, while in [9] these rates are respectively 0.3 and 0.8. A very small crossover rate and very high mutation rate affects directly the results in the experiment.

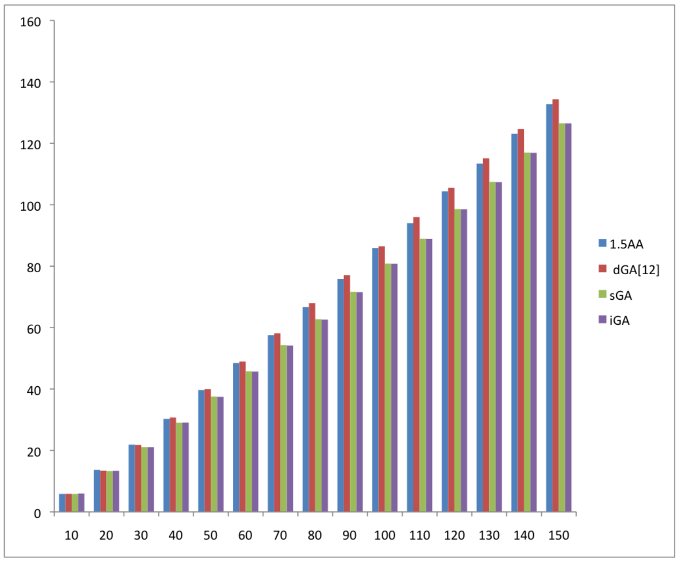

Fig. 4. Comparison of results for the 1.5-approximate algorithm (1.5AA), the simple GA in [12] (dGA) and the standard and improved GAs (sGA and iGA)

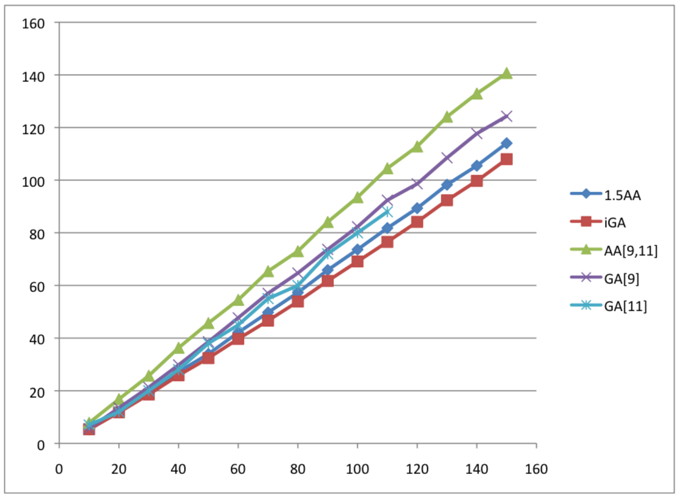

Fig. 5. Comparison of results for the 1.5-approximate algorithm (1.5AA), the improved GA (iGA), the approximate algorithm (AA[9], [11])and GA in [9] and the GA in [11]

The standard deviation of the results was also included in tables II and III. It can be seen that when the length of the permutation grows the standard deviation also grows, this indicates that the results are far from the average. This also indicates that the results provided by the GAs are more distant from the results obtained by the approximation algorithm, since we always have better results we can say that our results overcome the results by the fixed 1.5-approximation algorithm.

In summary, from the experiments, one can conclude that both the standard and improved GAs compute better results on average than the ones obtained by the fixed 1.5-approximation algorithm. These results are better than the ones reported previously by the authors in [12] for the simple GA based purely on elimination of breakpoints in each generation. Also, the standard and improved proposed GAs outperform the GA approaches presented in [9] and in [11]. Finally, it can be observed that for permutations o length greater than or equal to 50, the improved GA outperforms the standard GA.

VI. Conclusion

A standard genetic algorithm based on the Auyeung's et al method ([9]) for solving the problem of sorting by reversals was proposed and subsequently improved including the heuristic of eliminating breakpoints usually applied in approximation algorithms ([2], [3]). In addition to the main distinguishing feature that is the hybrid application of a fitness mechanism based on the computation of exact solutions for signed permutations, through the linear time algorithm in [5], and the elimination of breakpoints in the early generations, in the proposed approaches the parameters of the GA were adequately established and comparison were precisely done with a fixed 1.5-approximation algorithm. The experiments shows that results obtained by both the standard and the improved GAs overcome the results obtained by the 1.5 approximation algorithm as well as the ones presented by previous related works.

It is necessary to stress here, that the fitness calculation is restricted to the reversal distance without computing the sequence of necessary reversals to sort a permutation. From the biological point of view this is not a drawback, since it unnecessary for phylogenetic reconstruction and, moreover, it is usually meaningless because there are many different optimal sorting sequences. From the combinatorial point of view, construction of optimal sequences may be of great interest and it is an interesting future work. In order to obtain the sequences, one might apply sub-quadratic algorithm presented by Swenson et al [14] that works in $O(n\, log\, n)$ time for almost the majority of signed permutations.

Acknowledgment

The authors would like to thank the Brazilian Coordination for the Improvement of Higher Education Personnel CAPES and the Brazilian National Counsel of Technological and Scientific Development CNPq.

References

[1] J. Kececioglu and D. Sankoff, "Exact and approximation algorithms for the inversion distance between two chromosomes," in *Combinatorial Pattern Matching*, ser. Lecture Notes in Computer Science, A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds. Springer Berlin / Heidelberg, 1993, vol. 684, pp. 87–105, 10.1007/BFb0029799. [Online]. Available: http://dx.doi.org/10.1007/BFb0029799

[2] V. Bafna and P. Pevzner, "Genome rearrangements and sorting by reversals," in *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, nov 1993, pp. 148 –157.

[3] S. Hannenhalli and P. Pevzner, "Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals," in *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, ser. STOC '95. New York, NY, USA: ACM, 1995, pp. 178–189. [Online]. Available: http://doi.acm.org/10.1145/225058.225112

[4] P. Berman and S. Hannenhalli, "Fast sorting by reversal," in *CPM*, ser. Lecture Notes in Computer Science, D. S. Hirschberg and E. W. Myers, Eds., vol. 1075. Springer, 1996, pp. 168–185.

[5] 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," in *WADS*, ser. Lecture Notes in Computer Science, F. K. H. A. Dehne, J.-R. Sack, and R. Tamassia, Eds., vol. 2125. Springer, 2001, pp. 365–376.

[6] A. Caprara, "Sorting by reversals is difficult," in *Proceedings of the first annual international conference on Computational molecular biology*, ser. RECOMB '97. New York, NY, USA: ACM, 1997, pp. 75–83. [Online]. Available: http://doi.acm.org/10.1145/267521.267531

[7] D. A. Christie, "A 3/2-approximation algorithm for sorting by reversals," in *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '98. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1998, pp. 244–252. [Online]. Available: http://dl.acm.org/citation.cfm?id=314613.314711

[8] P. Berman, S. Hannenhalli, and M. Karpinski, "1.375-approximation algorithm for sorting by reversals," in *Proceedings of the 10th Annual European Symposium on Algorithms*, ser. ESA '02. London, UK, UK: Springer-Verlag, 2002, pp. 200–210. [Online]. Available: http://dl.acm.org/citation.cfm?id=647912.740832

[9] A. Auyeung and A. Abraham, "Estimating genome reversal distance by genetic algorithm," in *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, vol. 2, dec. 2003, pp. 1157 – 1161 Vol.2.

[10] M. Zhongxi and Z. Tao, "An improved genetic algorithm for problem of genome rearrangement," *Wuhan University Journal of Natural Sciences*, vol. 11, pp. 498–502, 2006, 10.1007/BF02836651. [Online]. Available: http://dx.doi.org/10.1007/BF02836651

[11] A. Ghaffarizadeh, K. Ahmadi, and N. Flann, "Sorting unsigned permutations by reversals using multi-objective evolutionary algorithms with variable size individuals," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*, june 2011, pp. 292 –295.

[12] J. L. Soncco-Álvarez and M. Ayala-Rincón, "A genetic approach with a simple fitness function for sorting unsigned permutations by reversals," Universidade de Brasília, Tech. Rep., 2012, available: www.mat.unb.br/∼ayala/publications.html.

[13] M. Bóna and R. Flynn, "The average number of block interchanges needed to sort a permutation and a recent result of stanley," *Inf. Process. Lett.*, vol. 109, no. 16, pp. 927–931, 2009.

[14] K. Swenson, V. Rajan, Y. Lin, and B. Moret, "Sorting signed permutations by inversions in o (nlogn) time," in *Research in Computational Molecular Biology*, ser. Lecture Notes in Computer Science, S. Batzoglou, Ed. Springer Berlin / Heidelberg, 2009, vol. 5541, pp. 386–399. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02008-7_28