Parallel Genetic Algorithms with Sharing of Individuals for Sorting Unsigned Genomes by Reversals

Lucas A. da Silveira José L. Soncco-Álvarez
Department of Computer Science
Universidade de Brasília
Email: {lucas.angel9, jose.soncco.alvarez}@gmail.com

Mauricio Ayala-Rincón
Departments of Computer Science and Mathematics
Universidade de Brasília
Email: ayala@unb.br

Abstract—**Rearrangement by reversals is a suitable global operation when treating genomes with a single chromosome. The problem of sorting by reversals unsigned genomes is an optimization problem that was shown to be $\mathcal{NP}$-hard. Several approximation algorithms were proposed, among them, in previous work, was introduced a competitive genetic algorithm and its standard parallel version which provided a substantial speedup. In this paper, two approaches using island models to parallelize such algorithm are presented. The first approach uses the unidirectional ring communication topology to exchange individuals between neighboring islands and, the second uses a complete graph scheme for the distribution of individuals among islands. Both approaches were proposed with the objective of improving precision (that is, for reducing the number of reversals) and decreasing the runtime regarding the sequential version. Experiments were performed with randomly generated synthetic genomes and the results show that the parallel approach using the ring communication topology outperforms the previously proposed genetic algorithm and its parallel version in terms of accuracy, providing solutions with less reversals and, that the parallel approach using the complete graph topology does not provide significant improvements regarding the others algorithms. Regarding execution time, both new parallel approaches get competitive speedups regarding the speedup achieved by the standard parallel version of the genetic algorithm.**

I. INTRODUCTION

The study of the evolutionary distance between different species using biological genomes requires the reconstruction of the sequence of evolutionary events which takes one genome to another. There is a diversity of rearrangement mechanisms using global operations such as reversals, transpositions, duplications and translocations. In this work, we are working with genomes given as sequences of genes arranged into just one chromosome using the *reversal* operation as rearrangement mechanism.

A unichromosomal genome can be represented as a permutation $\pi = (\pi_1\, \pi_2\, \dots\, \pi_n)$ defined over the set $\{1, \cdots, n\}$ where n is the number of genes in the genome. In addition, two variants of permutations are taken into account: signed and unsigned permutations. In the first case, each π_i has a positive or negative sign reflecting its orientation within the genome; in the unsigned case, the gene orientation is not taken into consideration. Permutations can be seen as sequences and the *reversal operation* converts a contiguous sub sequence into its reverse changing also the sign of each gene when orientation is considered. The *reversal distance problem* consists in computing the minimum number of reversals that transform a genome into another.

For the *signed reversal distance problem* (*SRD*), Hannenhalli and Pevzner showed that the problem belongs to the class $\mathcal{P}$ by giving an exact polynomial algorithm ($O(n^4)$) [1]. A year later, Berman and Hannenhalli improved this providing an $O(n^2\alpha(n))$ running time algorithm, where α is an inverse Ackerman's function [2]. After that, Bader et al. proposed a linear time algorithm [3].

The *unsigned reversal distance problem* (*URD*) is addressed in this paper and considers unsigned permutations. It was shown to be $\mathcal{NP}$-Hard by Caprara [4]. Before proving the $\mathcal{NP}$-Hardness of the problem, Kececioglu and Sankoff designed an approximation algorithm of ratio 2 [5], and Bafna and Pevzner provided an algorithm that improved the ratio to 1.75 [6]. Moreover, Cristie improved the ratio to 1.5 ([7]) and then, Berman et al. to 1.375 ([8]). Recently, Soncco-Álvarez and Ayala-Rincón proposed a genetic algorithm (for short GA) [9], referred in this paper as $\mathcal{GA}_\mathrm{S}$, and a parallel version [10], referred as $\mathcal{GA}_\mathrm{P}$, to solve the *URD* problem. The algorithm $\mathcal{GA}_\mathrm{S}$ maps a given unsigned permutation of length n into a subset of its 2^n possible signed versions (each unsigned gene might assume two directions). This is done by assigning randomly a positive or negative sign to each gene of the input permutation. The subset of signed permutations forms the population of $\mathcal{GA}_\mathrm{S}$, where each solution for *SRD* of each individual in the population is indeed a *feasible solution* for the input unsigned permutation. The fitness of the individuals of the population is computed by using the linear time algorithm proposed by Bader et al. [3] for computing the reversal distance of signed permutations. The algorithm $\mathcal{GA}_\mathrm{P}$ is based on the model of multiple-population coarse-grained, also known as the island model, where each island maintains its own $\mathcal{GA}_\mathrm{S}$ instance with a population of size $n \log n$ evolving independently of the other islands. The output of $\mathcal{GA}_\mathrm{P}$ is the best result among all islands. It is important to highlight that the authors implemented the 1.5-approximation algorithm [7] as a quality control mechanism for the solutions provided by the genetic algorithms. Through experiments, it was concluded that $\mathcal{GA}_{\text{S}}$ has better performance regarding precision of the results than the 1.5-approximation algorithm, and that $\mathcal{GA}_{\text{P}}$ improves the quality of the solutions computed by $\mathcal{GA}_{\text{S}}$ using 23 islands (and one master) with populations of size $n \log n$, which is not surprising since the whole population is much larger than the used by the $\mathcal{GA}_{\text{S}}$.

The main contribution of this work is the development and implementation of two parallel genetic algorithms. These algorithms follow the island model and propose different migration mechanisms between the islands with the objective of improving the time and accuracy of the results. Two different topologies are considered with different migration policies.

- The first parallel approach maintains several instances of $\mathcal{GA}_{\text{S}}$ each one dealing with a population of size $n \log n$ and uses a unidirectional ring communication topology to exchange individuals between neighboring islands. Based on this approach two variants were proposed,
  - In the first variant, denoted as $\mathcal{GA}_{\text{RNP}}$, different populations for each island are used.
  - In the second variant, denoted as $\mathcal{GA}_{\text{RP}}$, the population of the $\mathcal{GA}_{\text{S}}$ is partitioned into populations of equal size and distributed among the islands.
- The second parallel approach uses a complete graph (*clique*) topology for exchanging individuals among island populations. As for the first approach, this one has two variants that are denoted as $\mathcal{GA}_{\text{CNP}}$ and $\mathcal{GA}_{\text{CP}}$. The former is a variant with a different population for each island and in the latter the population of $\mathcal{GA}_{\text{S}}$ is partitioned among the islands.

Section II presents the necessary definitions and terminology; Section III presents the parallel GA approaches for the URD problem; Section IV presents experiments and results; and, before concluding and discussing future work in Section VI, Section V discusses the results. The source code and data used in the experiments is available at `genoma.cic.unb.br`.

II. Definitions and Terminology

This section presents the definitions and terminology related to unsigned permutations which were taken from [11] and [12].

Consider the group of permutations of length $n > 0$, where $n \in \mathbb{N}$. In this group we distinguish the identity permutation that is represented by ι and defined as $\iota_k = k$ for all $k = 1, ..., n$. Permutations of length n are also abstracted as bijective functions π from and onto the set $\{1, \ldots, n\}$; in the functional setting, instead π_i we can write $\pi(i)$ to denote the i^{th} element of the permutation. The *size* of a permutation is its length. The inverse of a permutation π is denoted as π^{-1} and defined as $\pi^{-1}(j) = i$ if and only if $\pi(i) = j$, for all $1 \leq j \leq n$. A reversal $\rho_{(l,m)}$, where $1 \leq l \leq m \leq n$, is an operation that (is an special kind of permutation by itself and) acts over a permutation π, and reverses the elements included in the interval from position l to position m.

Given two permutations π and δ, the reversal distance between them is the minimum number of reversals that are required to transform π into δ. By algebraic properties of permutation groups, reversals $\rho_1, \ldots, \rho_k$ are such that $\pi\rho_1 \cdots \rho_k = \delta$ if and only if $\delta^{-1}\pi\rho_1 \cdot \rho_k = \iota$. Thus, a solution to the problem of determining the minimum number of reversals to transform π into δ also transforms the unsigned permutation $\delta^{-1}\pi$ into the identity permutation. Therefore we abstract the URD problem as the problem of transforming by reversal a permutation into the identity.

Figure 1 shows how the sequence of reversals $\rho_{(2,5)}$, $\rho_{(3,5)}$, $\rho_{(4,5)}$ sorts the permutation $(1\,3\,5\,4\,2\,6)$.

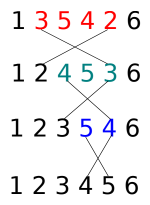

Fig. 1. Application of a sequence of reversals to transform a permutation into the identity permutation (pivots 0 and 7 are not depicted)

By convenience we will extend any permutation π with the initial and final fixed *pivots* 0 and $n+1$: $\pi_0 = 0$ and $\pi_{n+1} = n+1$. Let $l \sim m$ denote the property $|i-j| = 1$. Two elements π_l and π_m of π are said to be *consecutive* if $l \sim m$. Consecutive elements π_l and π_m of π are said

- to be *adjacent* if $\pi_l \sim \pi_m$ and
- to form a *breakpoint* if $\pi_l \nsim \pi_m$.

Note that in the first permutation in Figure 1 the consecutive elements 5 and 4 are adjacent while all other consecutive elements (1 and 3, 3 and 5, 4 and 2 and 2 and 6) form breakpoints. Notice also that the first two reversals eliminate one breakpoint while the third one eliminates two breakpoints (3 and 5, and 4 and 6).

Let $b(\pi)$ denote the number of breakpoints in π. The only permutation that holds $b(\pi) = 0$ is the identity permutation, for any other permutation, $b(\pi) > 0$. Notice that this happens because we are considering extended permutations in which the pivots $\pi_0 = 0$ and $\pi_{n+1} = n+1$ are fixed; thus the permutation defined as $\pi_i = n+1-i$, for $1 \leq i \leq n$, has indeed two breakpoints given by the consecutive elements 0 and n, and 1 and $n+1$.

Let ρ be a reversal that transforms π into π', then $b(\pi) - b(\pi') \in \{-2, -1, 0, 1, 2\}$, which means that ρ might mantain the number of breakpoints, or eliminate or add one or two breakpoints. In the literature an i-reversal is defined as a reversal that reduces the number of breakpoints by i. The fact that at most two breakpoints can be eliminated after application of a reversal is used as a heuristic in approximate algorithms: the application of 2-reversals is prioritized in order to sort permutations ([5], [6], [7]) expecting that through this local optimization the convergence to the identity permutation might be quicker than through applications of 1- and/or 0-reversals. Notice that this is the case for the initial permutation in Figure 1: applying the 2-reversal $\rho_{(3,5)}$ we obtain the permutation $(1\,3\,2\,4\,5\,6)$ and then applying the 2-reversal $\rho_{(2,3)}$ we obtain the identity ι.

The elements of a signed permutation π are either oriented positively $(+\pi_i)$ or negatively $(-\pi_i)$ for $0 < i < n$ for n the size of π. The reversal operation over signed permutations is defined as for unsigned permutations except that the sign of all elements between the range of action of the reversal operation changes; for instance, if we apply the permutation $\rho_{(2,5)}$ to the permutation $(-1\,+3\,+5\,-4\,+2\,+6)$ we obtain the permutation $(-1\,-2\,+4\,-5\,-3\,+6)$. An unsigned permutation π' of length $2n$ can be obtained from a signed permutation π by replacing each positive element $+\pi_i$ by the pair $(2\pi_i-1, 2\pi_i)$, and each negative element $(-\pi_i)$ by the pair $(2\pi_i, 2\pi_i-1)$. For instance, the permutation $(1\,-3\,-5\,+4\,+2\,+6)$ is transformed into the unsigned permutation $(1\,2\,6\,5\,10\,9\,7\,8\,3\,4\,11\,12)$ and the *signed identity* $(+1\,+2\,+3\,+4\,+5\,+6)$ corresponds to the unsigned identity of length 12: $(1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\,12)$. Also, pivots 0 and $2n+1$ need to be added. Since this transformation provides permutations that consist of consecutive pairs of adjacent elements (of the form $2i-1$ and $2i$) the reversals to be considered for sorting them should not break these adjacencies. Thus, the class of reversals acting over these class of permutations are of the form $\rho_{2l-1,2m}$ where $1 \leq l \leq m \leq n$, which is a subgroup of the group of permutations of length $2n$ [13]. This transformation is applied to solve the *SRD* problem in exact algorithmic approaches such as those proposed in [14] and [3].

A. Genetic Algorithm

The last two authors presented in [9] a modified version of the standard genetic algorithm $\mathcal{GA}_S$ (introduced in [15]) briefly described below. Initially, a random population of signed permutations is generated based on the input unsigned permutation. Posteriorly, in each generation of $\mathcal{GA}_S$ the reproduction phase is performed as follows: select two individuals of the population, which are among the best current individuals for which crossover and mutation operations are applied generating two new individuals. The fitness function used is the reversal distance for signed permutations. If the new individuals are better than those in the current population, they become members of the population. The algorithm $\mathcal{GA}_S$ finishes after all the generations, that is fixed as the length of the input permutation, have been completed.

In $\mathcal{GA}_S$ as in all our other approaches, the fitness function is computed through application of the algorithm for solving the *SRD* problem proposed by Bader in [3]. We use an implementation provided by the author and available as part of a framework at `www.cs.unm.edu/∼moret/GRAPPA`. The pseudo-code of $\mathcal{GA}_S$ is shown in Algorithm 1.

B. Parallel GA with Independent Runs

Soncco-Álvarez et al. use the simplest version of the parallel island model ([16]) to propose $\mathcal{GA}_P$, a parallel version of $\mathcal{GA}_S$. In this approach, based on the master-slave model, each slave process (representing an island) executes an instance of $\mathcal{GA}_S$ (selection, crossover, mutation, replacement) within its respective population of size $n \log n$; the master process

Algorithm 1: GA for Calculating URD

```
  Input: Unsigned permutation $\pi$
  Output: Number of reversals to sort $\pi$
1 Generate the initial population of signed permutations;
2 Compute fitness of the initial population;
3 for $i = 1$ to Length($\pi$) do
4     Perform the selection and save the best solution
        found;
5     Apply the crossover operator;
6     Apply the mutation operator;
7     Compute the fitness of the current population;
8     Perform replacement of the worse individuals;
```

receives the individual and the best fitness values found by each slave process in each cycle of $\mathcal{GA}_S$. At the end of the stage of generations the master process returns the solution of the best individual as output of $\mathcal{GA}_P$. It is worth mentioning that when a process is sending or receiving data it gets blocked. The pseudo-code of this approach is show in Algorithm 2.

Algorithm 2: Parallel $\mathcal{GA}_S$ with Multiple Populations for URD ($\mathcal{GA}_P$)

```
   Input: Unsigned permutation $\pi$
   Output: Number of reversals to sort $\pi$
   /* For the master process:                */
1  for each generation do
2      for all slave processes do
3          receive the best fitness from a slave;
4          if solution < best fitness then
5              solution = best fitness;
   /* For a slave process:                    */
6  Generate the initial pop. of signed permutations for $\pi$;
7  Compute fitness of the initial population;
8  for $i = 1$ to Length($\pi$) do
9      Perform the selection and save the best solution
         found;
10     Apply the crossover operator;
11     Apply the mutation operator;
12     Compute the fitness of the current population;
13     Perform replacement of the worse individuals;
14     Send to the master the best fitness found;
```

III. Parallel GA Approaches for URD

The discussion on our parallel *GAs* follows the stratification of parallel evolutionary algorithms proposed by Cantú-Paz in [16] and Sudholt in [17].

Independent runs is the strategy used in the algorithm $\mathcal{GA}_P$. This strategy consists in the execution of independent runs of the same algorithm (in our case $\mathcal{GA}_S$) in parallel and after that, the results are collected and the best solution is given as output. This strategy is used to streamline the demand of experiments trying to avoid interference due to communication between processes. In addition, there is a variant of this strategy referred as island model, also called of coarse-grained model, where the population of each parallel execution is maintained in an island. In this model each island evolves independently most of the time. However, after some time the solutions are exchanged between islands through migration.

When migration is allowed, it is necessary to organize communication between islands in a specific topology. At determined points of time individuals selected from each island are sent and received to and from the neighboring islands; such individuals are called (emi and im)migrants and are included on the target island according to some criterion. Different communication topologies and migration strategies are proposed with the intention to favor islands that are trapped in regions of low aptitude in the search space so that they can be populated by individuals from more successful islands. This helps to coordinate the search to focus on the most promising regions of the search space and use the available resources effectively. Here we are using two island communication strategies: unidirectional ring and complete graph topologies, which can be seen in Figure 2.

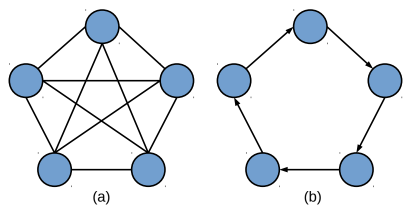

Fig. 2. (a) complete graph topology and (b) unidirectional ring topology

After a *migration topology* is fixed, other design options or migration policies that affect the behavior of an island model should be established. Below we highlight design options of great importance:

- Emigration policy: *remove* or *clone* individuals taking into account the choice among worse, random, or better individuals.
- Immigration policy: *replace* individuals considering the choice among worse, random, or better individuals in the target population.
- Number of immigrants and emigrants: number of individuals that will be sent and received from one population to another.
- Migration interval: time or number of generations between one migration and another.

According to the specific *GA* approach (and related problem) being parallelized, setting adequately these design options is of great importance in order to adjust properly the whole population convergence. Negligentiating this will compromise the quality of results.

According to these design options, the parallel *GAs* proposed in this work ($\mathcal{GA}_{\text{RNP}}$, $\mathcal{GA}_{\text{RP}}$, $\mathcal{GA}_{\text{CNP}}$ and $\mathcal{GA}_{\text{CP}}$) use the parameters showed in table I. Below, we explain these parameters.

- Crossover and mutation probability and, selection and replacement percentages, that are the basic parameters of a *GA* [18].
- *NumMigIndividuals* represents the number of individuals for migration.
- *TypeEmIndividual* represents the type of individuals selected for emigration among
  1) Better individuals.
  2) Worse individuals.
  3) Random individuals.
- *EmPolicy* represents the emigration policy:
  1) Clone individuals.
  2) Remove individuals.
- *TypeImIndividual* represents the immigration policy, that is the kind of individuals selected in the target island to be replaced by the immigrants:
  1) Worse individuals.
  2) Random individuals.
  3) Similar individuals.

  By similar individuals one understand individuals with the same fitness than the immigrants.
- *MigrationInterval* represents the migration interval that is given from the percentage of reproduction cycles that needs to be performed to initiate the exchange of individuals between islands.

We use the *(Message Passing Interface) MPI* library. The standard (parallel) commands *MPI_Send* and *MPI_Recv* were used for the exchange of messages among processes (processes that represent the islands), and *MPI_Barrier* to perform the synchronization of running processes.

A. Parallel GA with Unidirectional Ring Topology

First we will describe the approach $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$ that are presented in the Algorithms 3, 4 and 5. These parallelizations are based on the unidirectional ring communication topology. In $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$ each process represents an island and maintains a different population of size $n \log n$. At each migration interval (parameter *MigrationInterval* in the Algorithm 3) each process selects an amount of individuals (parameter *NumMigIndividuals* in the Algorithm 4) of a given type, according to the parameter *TypeEmIndividual* (Algorithm 4), which can be removed or not, according to the parameter *EmPolicy* (Algorithm 4), to send to their respective neighbor (parameter P_1 in Algorithm 4). The destination process inserts immigrant individuals into its population by replacing individuals of a determined profile, defined by parameter *TypeImIndividual* (Algorithm 4) by them. The use of a barrier is to prevent a process from having more reproductive cycles than another process and thus to lose stages which need to be performed to exchange individuals. The output of $\mathcal{GA}_{\text{RNP}}$ is the best result among all processes.

Algorithm 3 shows the pseudo-code of both $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$. The difference between the algorithms is that the former does not execute line 1 and the latter does not execute line 2. Algorithm 5 describes how the initial population is generated by process 0 (used only by $\mathcal{GA}_{\text{RP}}$) which constructs a population of size $p*n\log n$, where p represents the number of processes, and sends fragments with $n\log n$ individuals to each other process. Finally, Algorithm 4 describes the procedure for exchanging individuals in parallel.

Algorithm 3: Parallel GA with Unidirectional Ring Topology for URD

```
   Input: Unsigned permutation $\pi$
   Output: Number of reversals for sort $\pi$
   /* Only for $\mathcal{GA}_{\text{RP}}$ :                          */
1  Load initial pop. of signed permutations for $\pi$ (Alg. 5);
   /* Only for $\mathcal{GA}_{\text{RNP}}$ :                         */
2  Generate the initial pop. of signed permutations for $\pi$;
3  Compute fitness of the initial population;
4  count = MigrationInterval * Length(population);
5  for $i = 1$ to Length($\pi$) do
6      Perform the selection and save the best solution
        found;
7      count = count − 1;
8      if count==0 then
9          Send and receive individuals (Alg. 4);
10         count = MigrationInterval * Length(population);
11     MPI_Barrier;
12     Apply the crossover operator;
13     Apply the mutation operator;
14     Compute the fitness of the current population;
15     Perform replacement of the worse individuals;
16     Save the best individual;
```

B. Parallel GA with Complete Graph Topology

The algorithms $\mathcal{GA}_{\text{CNP}}$ and $\mathcal{GA}_{\text{CP}}$ are very similar to $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$, respectively. The difference between such algorithms is the topology for the exchange of individuals that is a complete graph or *clique* communication topology. The approaches make use of Algorithms 5 and 6, and a simple adaptation of Algorithm 3 that is given as follows: where we read $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$ (comments before and lines 1 and 2) we will read $\mathcal{GA}_{\text{CNP}}$ and $\mathcal{GA}_{\text{CP}}$, respectively and, instead of calling Algorithm 4 (line 9) the procedure described in Algorithm 6 is called to perform the necessary exchange of individuals according to the migration policy for the complete graph topology. The Algorithm 5 has no need of adaptation and runs as explained in Section III-A.

IV. Experiments and Results

The algorithms were implemented using the MPI library of the C language. The experiments were executed on a computer with 128GB of RAM, and two processors Xeon E5-2620 with hyper-threading. Each processor has 6 cores with a CPU clock rate of 2.4Ghz; thus, in this platform 24 threads were executed simultaneously without degrading execution performance.

In order to obtain a parameter setting for $\mathcal{GA}_{\text{CNP}}$, $\mathcal{GA}_{\text{CP}}$, $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$, which provide the best results (the minimum

Algorithm 4: Procedure for sending and receiving individuals in parallel for $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$

```
1  Neighbor $P_1$ is the process that will receive emigrant
    individuals;
2  Select NumMigIndividuals of TypeEmIndividual in
    current process;
3  Send selected individuals (line 2) to process $P_1$
    (MPI_Send);
4  if EmPolicy == Remove then
5      Remove selected individuals from the current process;
6  Neighbor $P_2$ is the process that will send immigrant
    individuals;
7  Receive NumMigIndividuals from process $P_2$;
8  (MPI_Recv);
9  if TypeImIndividual == Worse then
10     Replace NumMigIndividuals worse individuals by the
        received individuals;
11 else if TypeImIndividual == Random then
12     Replace NumMigIndividuals random individuals by
        the received individuals;
13 else
14     Replace NumMigIndividuals similar individuals by
        the received individuals of $P_2$;
```

Algorithm 5: Procedure to generate the population of each process

```
   Input: Unsigned permutation $\pi$
   Output: Each process with its population
1  p = numberProcesses;
2  Process 0 generate $n\log n$ signed permutations of $\pi$ to
    itself;
3  for $i = 1$ to $p$ do
4      Process 0 generate $n\log n$ signed permutations of $\pi$
        and send to process $i$ (MPI_Send);
5  for $i = 1$ to $p$ do
6      Process $i$ Receive $n\log n$ signed permutations of
        process 0 (MPI_Recv);
```

number of reversals possible for sorting permutations), exhaustive experiments were performed. These experiments were conducted as follows: groups of twenty permutations containing permutations with n genes for $n \in \{20, 50, 100, 150\}$ were given as input. Each algorithm was executed 10 times for each permutation in each group. To adjust each parameter (Table I), different values were empirically estimated on a scenario of possible good values, and random values were set for parameters that were not yet defined through the experiments. For example, to define the crossover parameter according to the values estimated, the other parameters (mutation, selection, etc) were fixed with arbitrary values. In the end of the experiment the parameters values that gave the best

Algorithm 6: Procedure for sending and receiving individuals for $\mathcal{GA}_{\text{CNP}}$ and $\mathcal{GA}_{\text{CP}}$

```
1  for $i = 0$ to numberProcesses do
2      if $i$ is not current process then
3          Select NumMigIndividuals of TypeEmIndividual of its population;
4          Send selected individuals to process $i$ (MPI_Send);
5      if EmPolicy == Remove then
6          Remove selected individuals from the current process;
7  for $i = 0$ to numberProcesses do
8      if $i$ is not current process then
9          Receive NumMigIndividuals from process $i$;
10         (MPI_Recv);
11         if TypeImIndividual == Worse then
12             Replace NumMigIndividuals worse individuals by the received individuals;
13         else if TypeImIndividual == Random then
14             Replace NumMigIndividuals random individuals by the received individuals;
15         else
16             Replace NumMigIndividuals similar individuals by the received individuals;
```

TABLE I
PARAMETERS FOR THE PARALLEL ALGORITHMS

Parameter	estimated values
Crossover probability	$80\%, 81\%, \cdots, 99\%, 100\%$
Mutation probability	$1\%, 2\%, \cdots, 5\%$
Percentage for selection	$80\%, 81\%, \cdots, 99\%, 100\%$
Percentage for replacement	$10\%, 20\%, \cdots, 80\%, 90\%$
NumMigIndividuals	1,2,3,4,5
TypeEmIndividual	Better, Worse, Random
EmPolicy	Clone, Remove
TypeImIndividual	Worse, Random, Similar
MigrationInterval	$10\%, 20\%, \cdots, 90\%, 100\%$

TABLE II
PARAMETERS FOR THE GENETIC ALGORITHMS PARALLEL PROPOSED

Parameter	$\mathcal{GA}_{\text{CNP}}$	$\mathcal{GA}_{\text{CP}}$	$\mathcal{GA}_{\text{RNP}}$	$\mathcal{GA}_{\text{RP}}$
Crossover probability	90%	90%	70%	96%
Mutation probability	1%	1%	1%	1%
Perc. for selection	80%	90%	90%	90%
Perc. for replacement	30%	30%	40%	40%
NumMigIndividuals	4	1	3	3
TypeEmIndividual	3	3	1	2
EmPolicy	1	1	1	1
TypeImIndividual	1	2	2	1
MigrationInterval	30%	10%	40%	50%

results for $\mathcal{GA}_{\text{CNP}}$, $\mathcal{GA}_{\text{CP}}$, $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{RP}}$ were chosen. These parameters values are shown in Table II.

A. Experiments with One Hundred Synthetic Permutations

In order to compare properly the algorithms $\mathcal{GA}_{\text{S}}$ and $\mathcal{GA}_{\text{P}}$ with the parallelizations proposed in this work, several experiments using groups of one hundred unsigned permutations were performed in the following way:

- One hundred unsigned permutations were randomly generated for each length in the set $\{50, 60, \cdots, 140, 150\}$.
- For each permutation in each set of permutations of the same length $\mathcal{GA}_{\text{S}}$, $\mathcal{GA}_{\text{P}}$, $\mathcal{GA}_{\text{RP}}$, $\mathcal{GA}_{\text{RNP}}$, $\mathcal{GA}_{\text{CP}}$ and $\mathcal{GA}_{\text{CNP}}$ were executed ten times with the same set of random seeds (populations are generated with these seeds).
- The average of the results of these ten executions for each algorithm was calculated. These averages represent the result (number of reversals) for each unsigned permutation of a given length.

The average results (and standard deviation) for each set of one hundred permutations and each algorithm are shown in Table III. Regarding processing time, Table IV shows the speed-up (computed as the sequential processing time divided by the time of the parallel algorithm) for each parallel algorithm taking in consideration permutations of length 150, that are the most difficult instances of this experiment.

V. DISCUSSION

Initially, it is worth mentioning that for the experiments the size of the search space used for both the sequential version ($\mathcal{GA}_{\text{S}}$) and their parallel versions was the same. The parallel algorithms used 24 threads with their respective populations of size $n \log n$ and $\mathcal{GA}_{\text{S}}$ used a unique population of size $24 * n \log n$. This allows a fair comparison regarding improvements in accuracy obtained by parallel approaches.

At first glance, Table III shows that $\mathcal{GA}_{\text{RP}}$ has better results regarding the other algorithms (bold font). Nevertheless, some of the results are very close one to another, so it was necessary to perform a statistical comparison in order to validate the statistical significance of these results. The methodology proposed by Demšar [19] was applied for the statistical comparison of algorithms, and it is as follows: first, the Friedman test is used to test the null hypothesis that our algorithms have the same performance; second if the latter test rejects the null hypothesis then the Holm test (post-hoc test) is performed, in which a control algorithm is compared against the remaining algorithms to test the null hypothesis that the control algorithm and one of the other algorithms have the same performance. Both the Friedman and Holm tests are available as a `JAVA` package called `CONTROLTEST` (at `sci2s.ugr.es/sicidm`) and use as default $\alpha = 0.05$ as significance level.

In the case of the Friedman test the null hypothesis is rejected when p-value $\leq \alpha$. In the case of the Holm test a null hypothesis is rejected when p-value $\leq \alpha/i$, where i decreases from (number of algorithms minus 1) until 1. For our data the Friedman test rejected the null hypothesis, so we performed the Holm test. Table V shows the results of the Holm test, where bold p-values are those that reject the null hypothesis. It can

TABLE III
AVERAGE RESULTS (A) AND STANDARD DEVIATION (SD) FOR THE EXPERIMENT WITH HUNDRED SYNTHETIC PERMUTATIONS

Length	$\mathcal{GA}_{\text{RP}}$ A	$\mathcal{GA}_{\text{RP}}$ SD	$\mathcal{GA}_{\text{RNP}}$ A	$\mathcal{GA}_{\text{RNP}}$ SD	$\mathcal{GA}_{\text{CP}}$ A	$\mathcal{GA}_{\text{CP}}$ SD	$\mathcal{GA}_{\text{CNP}}$ A	$\mathcal{GA}_{\text{CNP}}$ SD	$\mathcal{GA}_{\text{S}}$ A	$\mathcal{GA}_{\text{S}}$ SD	$\mathcal{GA}_{\text{P}}$ A	$\mathcal{GA}_{\text{P}}$ SD
50	**36.38**	1.138	36.46	1.172	36.39	1.15	36.39	1.156	36.46	1.193	36.4	1.152
60	**44.55**	1.423	44.68	1.462	44.58	1.432	44.59	1.44	44.77	1.562	44.61	1.461
70	**52.72**	1.524	52.9	1.572	52.74	1.543	52.76	1.543	53.01	1.668	52.81	1.553
80	**61.12**	1.413	61.36	1.433	61.15	1.413	61.2	1.439	61.76	1.753	61.26	1.47
90	**69.53**	1.574	69.88	1.607	69.59	1.575	69.58	1.562	70.53	1.924	69.79	1.61
100	**77.62**	1.721	78.06	1.811	77.73	1.74	77.74	1.73	79.13	2.348	78.07	1.766
110	**86.29**	1.641	86.73	1.72	86.34	1.658	86.38	1.684	88.07	2.234	86.81	1.696
120	**94.84**	1.671	95.41	1.696	94.98	1.647	95.0	1.634	97.35	2.313	95.71	1.732
130	**103.58**	1.806	104.25	1.868	103.73	1.821	103.73	1.805	106.55	2.549	104.74	1.91
140	**112.11**	2.133	112.8	2.198	112.28	2.131	112.3	2.169	115.84	2.951	113.72	2.272
150	**120.97**	1.59	121.78	1.572	121.08	1.577	121.16	1.532	125.43	2.125	122.96	1.658

TABLE IV
SPEED-UP FOR THE EXPERIMENT WITH PERMUTATIONS OF SIZE 150

Length	$\mathcal{GA}_{\text{RNP}}$	$\mathcal{GA}_{\text{RP}}$	$\mathcal{GA}_{\text{CNP}}$	$\mathcal{GA}_{\text{CP}}$	$\mathcal{GA}_{\text{P}}$
150	11.53	11.07	10.46	7.19	**13.20**

be observed that $\mathcal{GA}_{\text{RP}}$ algorithm is the control algorithm for all cases (permutation lengths), that is the one with the minimum rank (best performance). Note that $\mathcal{GA}_{\text{RP}}$ has statistically significant difference regarding $\mathcal{GA}_{\text{S}}$ and $\mathcal{GA}_{\text{RNP}}$ for all cases; also, $\mathcal{GA}_{\text{RP}}$ has statistically significant difference regarding all algorithms in 4 cases (permutation lengths of 60, 120, 130, and 140); and finally, the $\mathcal{GA}_{\text{CP}}$ algorithm is the one for which in many cases $\mathcal{GA}_{\text{RP}}$ has not a statistically significant difference.

Regarding the performance of running time execution (see Table IV), $\mathcal{GA}_{\text{P}}$ obtained a better speed-up than those of all other parallel algorithms, which was already expected since $\mathcal{GA}_{\text{P}}$ does not perform exchange of individuals during the breeding cycle as all other proposed parallelizations. In particular, $\mathcal{GA}_{\text{RP}}$ which provided the best overall results in the experiments, obtained a speed-up approximately 19% slower than $\mathcal{GA}_{\text{P}}$. Observe that $\mathcal{GA}_{\text{CP}}$ is the algorithm that provides solutions with degree of accuracy closest to $\mathcal{GA}_{\text{RP}}$, however, it requires 54% to more processing time. Thus,looking at the cost benefit with respect to performance and accuracy it is preferable to use $\mathcal{GA}_{\text{RP}}$.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed two approaches based on island models for parallelizing the genetic algorithm $\mathcal{GA}_{\text{S}}$ introduced in [15]. Both approaches use exchange of individuals between breeding cycles in order to improve the quality of the solutions. The first approach uses a ring communication topology through which individuals migrate between neighbor islands. This approach has two variants: $\mathcal{GA}_{\text{RNP}}$ with different populations for each island and $\mathcal{GA}_{\text{RP}}$ that uses a partition of the whole population of the standard $\mathcal{GA}_{\text{S}}$ among the islands. The second approach uses a complete graph communication topology so that individuals migrate between all the islands and, as the first approach, this has two variants: $\mathcal{GA}_{\text{CNP}}$ with different population for each island and $\mathcal{GA}_{\text{CP}}$ which also distributes the whole population of $\mathcal{GA}_{\text{S}}$ among the respective islands.

Several experiments were performed using packages of one hundred permutations randomly generated of sizes $\{50, 60, \cdots, 140, 150\}$. From the experiments, it was observed that $\mathcal{GA}_{\text{RP}}$ computes the best results in terms of accuracy providing sorting solutions with the smallest number of reversals and with a relatively low standard deviation; i.e., the solutions provided at each execution of the algorithm are always closer. The significance of these observations were confirmed by the Holm test in most of the analyzed samples as can be seen in Table V. Using permutations of size 150, an additional experiment was performed for measuring the speed-up of $\mathcal{GA}_{\text{RP}}$, $\mathcal{GA}_{\text{RNP}}$, $\mathcal{GA}_{\text{CP}}$, $\mathcal{GA}_{\text{CNP}}$, $\mathcal{GA}_{\text{P}}$ over $\mathcal{GA}_{\text{S}}$. This experiment confirmed that $\mathcal{GA}_{\text{P}}$ has a speed-up when compared to all the other algorithms as expected, since it does not perform exchange of individuals. Despite this, the variant $\mathcal{GA}_{\text{RP}}$, which is the best parallel algorithm regarding accuracy, has a competitive speed-up with respect to the sequential *GA* (6.25) as well as to all other parallel algorithms that exchange individuals.

The solutions obtained with the both communication topologies with different populations for each island, $\mathcal{GA}_{\text{RNP}}$ and $\mathcal{GA}_{\text{CNP}}$, were not satisfactory: $\mathcal{GA}_{\text{RNP}}$ provides better solutions than $\mathcal{GA}_{\text{P}}$ for inputs of length larger than 90, and $\mathcal{GA}_{\text{CNP}}$ presents just a small improvement for all the inputs. Indeed, we were expecting that having multiple populations and also sharing the best individual around all populations it was possible to achieve more expressive results than those obtained without sharing information, as done by $\mathcal{GA}_{\text{P}}$. An additional interesting fact is that creating a population for each island does not produce good results. So, as a future work, we are planning to study parallel evolutionary algorithms for the general problem of rearrangement of genomes with reversals as well as with other evolutionary operations such as translocations. In this context, we are planing to develop variants of the island model with more sophisticated communication topologies and policies, such as: grid, torus and trees, and also maintaining diversity of exchange behavior in the islands through specialized parameter setting for each island

TABLE V
RESULTS OF THE HOLM TEST FOR SETS OF HUNDREDS SYNTHETIC PERMUTATIONS WITH LENGTHS FROM 50 TO 150.

Len.	**Control Algorithm**	**i**	**Algorithm**	**Rank**	**P-value**	α/i
50	$\mathcal{GA}_{\text{RP}}$ (Rank: 3.13)	**5**	$\mathcal{GA}_{\text{RNP}}$	4.025	**7.1756E-4**	0.01
		4	$\mathcal{GA}_{\text{S}}$	4.01	**8.8074E-4**	0.0125
		3	$\mathcal{GA}_{\text{P}}$	3.37	0.3643	0.0167
		2	$\mathcal{GA}_{\text{CP}}$	3.235	0.6915	0.025
		1	$\mathcal{GA}_{\text{CNP}}$	3.23	0.7055	0.05
60	$\mathcal{GA}_{\text{RP}}$ (Rank: 2.52)	**5**	$\mathcal{GA}_{\text{S}}$	4.56	**1.4533E-14**	0.01
		4	$\mathcal{GA}_{\text{RNP}}$	4.27	**4.2388E-11**	0.0125
		3	$\mathcal{GA}_{\text{P}}$	3.45	**4.7199E-4**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	3.11	**0.0245**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	3.075	**0.0376**	0.05
70	$\mathcal{GA}_{\text{RP}}$ (Rank: 2.55)	**5**	$\mathcal{GA}_{\text{S}}$	4.78	**3.4980E-17**	0.01
		4	$\mathcal{GA}_{\text{RNP}}$	4.415	**1.8016E-12**	0.0125
		3	$\mathcal{GA}_{\text{P}}$	3.475	**4.7199E-4**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	3.015	0.0788	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.765	0.4164	0.05
80	$\mathcal{GA}_{\text{RP}}$ (Rank: 2.255)	**5**	$\mathcal{GA}_{\text{S}}$	5.28	**2.8665E-30**	0.01
		4	$\mathcal{GA}_{\text{RNP}}$	4.44	**1.4749E-16**	0.0125
		3	$\mathcal{GA}_{\text{P}}$	3.55	**9.8486E-7**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.925	**0.0113**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.55	0.2649	0.05
90	$\mathcal{GA}_{\text{RP}}$ (Rank: 2.005)	**5**	$\mathcal{GA}_{\text{S}}$	5.665	**1.6004E-43**	0.01
		4	$\mathcal{GA}_{\text{RNP}}$	4.525	**1.6551E-21**	0.0125
		3	$\mathcal{GA}_{\text{P}}$	3.88	**1.3721E-12**	0.0167
		2	$\mathcal{GA}_{\text{CP}}$	2.475	0.0757	0.025
		1	$\mathcal{GA}_{\text{CNP}}$	2.45	0.0926	0.05
100	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.81)	**5**	$\mathcal{GA}_{\text{S}}$	5.71	**3.5355E-49**	0.01
		4	$\mathcal{GA}_{\text{P}}$	4.415	**7.1344E-23**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	4.28	**1.0030E-20**	0.0167
		2	$\mathcal{GA}_{\text{CP}}$	2.4	0.0257	0.025
		1	$\mathcal{GA}_{\text{CNP}}$	2.385	0.0298	0.05
110	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.8)	**5**	$\mathcal{GA}_{\text{S}}$	5.875	**1.5856E-53**	0.01
		4	$\mathcal{GA}_{\text{P}}$	4.46	**8.8342E-24**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	4.255	**1.7103E-20**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.405	**0.222**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.205	0.1258	0.05
120	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.695)	**5**	$\mathcal{GA}_{\text{S}}$	5.965	**1.3554E-58**	0.01
		4	$\mathcal{GA}_{\text{P}}$	4.8	**8.3536E-32**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	3.98	**5.7967E-18**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.31	**0.0201**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.25	**0.0359**	0.05
130	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.59)	**5**	$\mathcal{GA}_{\text{S}}$	5.98	**7.8780E-62**	0.01
		4	$\mathcal{GA}_{\text{P}}$	4.955	**4.6671E-37**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	3.93	**9.2047E-19**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.305	**0.0069**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.24	**0.0140**	0.05
140	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.555)	**5**	$\mathcal{GA}_{\text{S}}$	5.985	**6.2817E-63**	0.01
		4	$\mathcal{GA}_{\text{P}}$	5.015	**4.4235E-39**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	3.885	**1.2903E-18**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.295	**0.0052**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.265	**0.0073**	0.05
150	$\mathcal{GA}_{\text{RP}}$ (Rank: 1.7)	**5**	$\mathcal{GA}_{\text{S}}$	5.99	**3.9717E-59**	0.01
		4	$\mathcal{GA}_{\text{P}}$	5.01	**6.5309E-36**	0.0125
		3	$\mathcal{GA}_{\text{RNP}}$	3.94	**2.5308E-17**	0.0167
		2	$\mathcal{GA}_{\text{CNP}}$	2.32	**0.0191**	0.025
		1	$\mathcal{GA}_{\text{CP}}$	2.04	0.1988	0.05

as suggested in [20]. We expect that a more elaborated island model for parallelizing $\mathcal{GA}_{\text{S}}$ will outperform the algorithm $\mathcal{GA}_{\text{RP}}$. It is worth mentioning that we are developing topology models using problems related to evolutionary metrics in order to apply the proposed genetic mechanisms for the construction of phylogenetic trees, not only with reversal distances but also with other measures such as the *DCJ* distance [21].

ACKNOWLEDGMENT

The second and third authors are funded by a CAPES Ph.D. scholarship and a *CNPq* grant, respectively.

REFERENCES

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

[2] P. Berman and S. Hannenhalli, *Fast sorting by reversal*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 168–185.

[3] D. A. Bader, B. M. 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, no. 5, pp. 483–491, 2001.

[4] A. Caprara, "Sorting by reversals is difficult," in *Proceedings of the first annual international conference on Computational molecular biology*. ACM, 1997, pp. 75–83.

[5] J. Kececioglu and D. Sankoff, *Exact and approximation algorithms for the inversion distance between two chromosomes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 87–105.

[6] V. Bafna and P. Pevzner, "Genome rearrangements and sorting by reversals," in *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*. IEEE Computer Society, 1993, pp. 148–157.

[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*. Society for Industrial and Applied Mathematics, 1998, pp. 244–252.

[8] P. Berman, S. Hannenhalli, and M. Karpinski, *1.375-Approximation Algorithm for Sorting by Reversals*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 200–210.

[9] J. L. Soncco-Álvarez and M. Ayala-Rincón, "A genetic approach with a simple fitness function for sorting unsigned permutations by reversals," in *Computing Congress (CCC), 2012 7th Colombian*. IEEE, 2012, pp. 1–6.

[10] J. L. Soncco-Álvarez, G. M. Almeida, J. Becker, and M. Ayala-Rincón, "Parallelization and virtualization of genetic algorithms for sorting permutations by reversals," in *Nature and Biologically Inspired Computing (NaBIC), 2013 World Congress on*. IEEE, 2013, pp. 29–35.

[11] J. Kececioglu and D. Sankoff, "Exact and approximation algorithms for the inversion distance between two chromosomes," in *Annual Symposium on Combinatorial Pattern Matching*. Springer, 1993, pp. 87–105.

[12] V. Bafna and P. A. Pevzner, "Genome rearrangements and sorting by reversals," *SIAM Journal on Computing*, vol. 25, no. 2, pp. 272–289, 1996.

[13] T. A. Lima and M. Ayala-Rincón, "On the average number of reversals needed to sort signed permutations," 2016, available ayala.mat.unb.br/publications.html.

[14] 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*. ACM, 1995, pp. 178–189.

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

[16] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs paralleles, reseaux et systems repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[17] D. Sudholt, *Springer Handbook of Computational Intelligence*. Springer Berlin Heidelberg, 2015, ch. Parallel Evolutionary Algorithms, pp. 929–959.

[18] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer, 2007.

[19] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[20] F. Herrera, M. Lozano, and C. Moraga, "Hybrid distributed real-coded genetic algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1998, pp. 603–612.

[21] 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.