Parallel Memetic Genetic Algorithms for Sorting Unsigned Genomes by Translocations

Lucas A. da Silveira
Department of
Computer Science
Universidade de Brasília
Email: lucas.angel9@gmail.com

José L. Soncco-Álvarez
Department of
Computer Science
Universidade de Brasília
Email: jose.soncco.alvarez@gmail.com

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

***Abstract*—The rearrangement of genomes is an important task to study the evolution of genomes. Rearrangement by translocations is a suitable operation when treating genomes with multiple chromosomes, indeed translocations occur between two chromosomes. The problem of sorting by translocations unsigned genomes is an optimization problem that was shown to be NP-Hard. Many approximation algorithms were proposed, among them, in previous work the authors have introduced competitive genetic algorithms improved with opposition based learning and memetic mechanisms. In this paper, two approaches for parallelizing a memetic algorithm are presented. The first approach, which uses a parallel calculation of the fitness, was proposed in order to obtain a speed-up over the memetic algorithm. The second approach, which deals in parallel with multiple populations, was proposed for improving precision (that is, for reducing the number of translocations). Many experiments were performed with randomly generated synthetic and biologically based genomes. Results show that the parallel approaches outperform the previously proposed memetic algorithm in terms of speed-up and accuracy providing solutions with less translocations.**

I. Introduction

The study of the evolutionary distance between two species using biological genomes requires the reconstruction of the sequence of evolutionary events that transform a genome into another. Currently, the mechanisms of rearrangements most used include global operations such as reversion, transposition, duplication and translocation. This work deals with genomes given as sequences of genes arranged into chromosomes; thus, it considers the operation of translocation, that combines genes of pairs of chromosomes, and which was first studied in [1]. More specifically, this operation consist in the interchange of blocks of genes between pairs of chromosomes of a genome. The problem of *sorting genomes by translocation* consists in given to genomes, build a minimal sequence of translocations necessary to transform a genome into another. The *translocation distance problem* consists in computing only the length of a minimal sequence of translocations. Two different representations of genomes have received attention: *signed* and *unsigned* genomes. In the latter one, the genes are abstracted without any orientation inside the genome, and at the former, each gene has either a positive or negative sign reflecting its orientation within in the genome.

For the *signed translocation distance* problem (STD), Hannenhalli in [2] provided the first polynomial time algorithm which provides the minimum distance in $O(n^3)$. A decade later, Wang et al in [3] taking advantage of the ideas used in [2] proposed a new algorithm which provides both the sequence as the distance of translocation in $O(n^2)$. Shortly after that, Bergeron et al in [4] provided two new algorithms using methods applied to the reversal distance problem [5]. The first one provides the minimum distance in $O(n)$ and the second one provides the minimal sequence of translocations used to transforming a genome into another in $O(n^3)$.

When considering unsigned genomes, the *unsigned translocation distance problem* (UTD), that is the one addressed in this paper, was shown to be $\mathcal{NP}$-hard by D. Zhu et al. in [6]. Before proving the $\mathcal{NP}$-hardness of this problem, Kececioglu and Ravi in [1] designed an approximation algorithm of ratio 2. More recent work intensifies the search for better approximate solutions. Namely, in 2007 Yun Cui et at in [7] proposed an approximate algorithm with ratio 1.75. A year later, the same authors improved the ratio to $1.5 + \varepsilon$. To the best of our knowledge, the best approximate algorithm reported in the literature was proposed very recently, in [8], and has ratio $1.408 + \varepsilon$; however, such algorithm demonstrates to be of theoretical interest, but not of practical interest, since the key routines of the $1.408 + \varepsilon$-approximation algorithm used to compute approximate solutions for the following $\mathcal{NP}$-complete problems: maximum set packing with size at most 3 ([9]) and maximum independent set with size at most 4 ([9]), and these routines can not be implemented in a straightforward manner. Recently, in [10], the authors proposed a genetic algorithm (referred in this paper as $\mathcal{GA}_\text{S}$) to solve UTD. Essentially, $\mathcal{GA}_\text{S}$ maps a given unsigned genome of size n (that is, with n genes) on a subset of its 2^n possible signed versions. This is done assigning randomly either a positive or negative signal to each gene of the input genome. The subset of signed genomes constitutes the population of $\mathcal{GA}_\text{S}$. Each solution for STD of each individual of this population corresponds to a feasible solution for the input unsigned genome. Thus, the fitness function of each one of these signed genomes in the population is given as the exact translocation distance provided in linear time by the algorithm for STD proposed in [4]. As a quality control mechanism for the solutions provided by

$\mathcal{GA}_{\text{S}}$ the $1.5+\varepsilon$-approximation algorithm was implemented in [10]. $\mathcal{GA}_{\text{S}}$ provides solutions that improve precision of those computed with the $1.5+\varepsilon$-approximation algorithm in 12%. Subsequently, the authors proposed improvements to $\mathcal{GA}_{\text{S}}$ and two new evolutionary algorithms were proposed and reported in [11]. The first improved algorithm incorporates memetic and local search techniques and the second one the technique of opposition based learning (OBL), and are referred in this paper respectively as $\mathcal{MA}_{\text{S}}$ and $\mathcal{GA}_{\text{OBL}}$. Through experiments and statistical tests it was concluded that $\mathcal{MA}_{\text{S}}$ has better performance regarding precision of the results than $\mathcal{GA}_{\text{S}}$, and that $\mathcal{GA}_{\text{OBL}}$ did not provides any improvement in the quality of the solutions computed by $\mathcal{GA}_{\text{S}}$.

As reported in [11], $\mathcal{MA}_{\text{S}}$ provides the best known solutions for UTD compared with the other evolutionary algorithms found in the literature. So, in this work two parallel approaches for improving $\mathcal{MA}_{\text{S}}$ were provided.

- The first one (referred as $\mathcal{MA}_{\text{PF}}$) searches a better execution performance maintaining the accuracy of the results, making use of the parallel global model (master-slave), where the master process keeps the stages of $\mathcal{MA}_{\text{S}}$ with a single population, while the slave processes are responsible of computing the fitness value of each individual.
- The second parallel approach seeks to increase the accuracy of the results using multi-populations in such a manner that each process maintains an instance of $\mathcal{MA}_{\text{S}}$ dealing with its respective population of size $n \log n$. Two variants were proposed:
  - The first variant shares the best individuals between the different populations during the breeding cycle, and is referred as $\mathcal{MA}_{\text{MPE}}$. In this strategy, each process deals with its own population and through of the exchange of the best individuals it is expected to accelerate the convergence of each population to the best adapted general solutions of the problem.
  - The second variant, referred as $\mathcal{MA}_{\text{MPnE}}$, evolves each population in a homogeneous way without sharing individuals of the different populations. The algorithm $\mathcal{MA}_{\text{MPnE}}$ outputs the best solution found among all the processes involved.

  In both variants, one searches essentially to improve the accuracy of the results exploring a bigger population, of size $n \log n$ times the number of processes, among the UTD search space.

Results from the experiments showed that $\mathcal{MA}_{\text{PF}}$ significantly reduces the runtime of the $\mathcal{MA}_{\text{S}}$. For the variants with multi-populations, $\mathcal{MA}_{\text{MPnE}}$ increases the accuracy of the obtained results (number of translocations) when compared to $\mathcal{MA}_{\text{S}}$; however, the variant with exchange of individuals $\mathcal{MA}_{\text{MPE}}$ does not provide any increase in the accuracy of the results.

The source code of the whole development is available at www.mat.unb.br/~ayala/publications.html.

II. Background

A. Definitions and Terminology

The definitions and terminology in this paper are compatible with those used in [2], [4], [12].

A signed genome is composed by a set of chromosomes $\{X_1, \cdots, X_l\}$, where each X_u for $0 < u <= l$, consists of a gene sequence, such that each gene x_i is represented by a different natural number with positive or negative signal. Each gene appears only once, either positively or negatively, in a genome. Unsigned genomes are defined similarly but genes are naturals without signals. Naturals representing genes in a signed or unsigned genome of length n range from 1 to n.

The orientation of a chromosome does not matter. So, a signed chromosome $X = (x_1, x_2, \ldots, x_k)$ is equal to $X = (-x_k, \ldots, -x_2, -x_1)$, and an unsigned chromosome $X = (x_1, x_2, \ldots, x_k)$ is equal to $X = (x_k, \ldots, x_2, x_1)$.

Consider two chromosomes $X = (x_1, x_2, \ldots, x_k)$ and $Y = (y_1, y_2, \ldots, y_m)$ of a **signed** genome. There are two kind of translocations:

- A *prefix-prefix* translocation $\rho_{pp}(X, Y, i, j)$, $1 < i \leq k$, $1 < j \leq m$ transforms the chromosomes X and Y into two new chromosomes $(x_1, \ldots, x_{i-1}, y_j, \ldots, y_m)$ and $(y_1, \ldots, y_{j-1}, x_i, \ldots, x_k)$.
- A *prefix-suffix* translocation $\rho_{ps}(X, Y, i, j)$, $1 < i \leq n$, $1 < j \leq m$ transforms the chromosomes X and Y into two new chromosomes $(x_1, \ldots, x_{i-1}, -y_{j-1}, \ldots, -y_1)$ and $(-y_m, \ldots, -y_j, x_i, \ldots, x_k)$.

For **unsigned** genomes, the definitions of *prefix-prefix* translocation and *prefix-suffix* translocation are the same eliminating the negative signals in the new chromosomes. So one obtains respectively the new chromosomes $(x_1, \ldots, x_{i-1}, y_j, \ldots, y_m)$ and $(y_1, \ldots, y_{j-1}, x_i, \ldots, x_k)$ as before, for the *prefix-prefix* translocation, and $(x_1, \ldots, x_{i-1}, y_{j-1}, \ldots, y_1)$ and $(y_m, \ldots, y_j, x_i, \ldots, x_k)$, for the *prefix-suffix* translocation.

The *Signed Translocation Distance* problem (briefly written as STD) consists in finding the minimum number of translocations for transforming a signed genome A into another signed genome B. The *Unsigned Translocation Distance* problem (for short written as UTD) consists in finding the minimum number of translocations for transforming an unsigned genome A into another genome B. In both cases, A and B must have the same size, that is the same number of genes n, and chromosomes N. In this paper the genome B (for both the signed and the unsigned versions of the problem) is always considered as an identity genome, that is a genome with all its genes (with positive signals) and sorted in increasing order. For instance, $\{(1,2,3)(4,5,6,7)\}$ and $\{(1,2,3)(4,5),(6,7)\}$ are identity genomes of size 7 with two and three chromosomes, respectively.

Consider a chromosome $X = (x_1, x_2, \cdots, x_k)$. The genes $x1$ and $-x_k$ are called tails of X for signed genomes. For unsigned genomes the tails are x_1 and x_k. Two genomes are called co-tails if their sets of tails are the same. Example: Consider two unsigned genomes $A = \{(1,5,7)(4,6,2,3)\}$

and $B = \{(1, 2, 3)(4, 5, 6, 7)\}$, both genomes are co-tails, with the tail set being $\{1, 3, 4, 7\}$.

A translocation does not modify the tails of a genome. In order to transform by translocations a genome A into another B, they must satisfy the following property.

Property 1 ([11]) The genomes A and B have the same number of genes n and chromosomes N, and they are co-tails.

Observe that, without loss of generality, when the genomes A and B are co-tails and have the same length and number of chromosomes, the genes of A and B can be renamed so that B can be rewritten as an identity.

B. Genetic Algorithms

A Genetic Algorithm (GA) is a meta-heuristic inspired in the principles of natural evolution, which was developed by J. H. Holland in the 1970s.

As mentioned in the introduction, da Silveira et al. proposed a standard genetic algorithm ($\mathcal{GA}_S$) for UTD in [10] which is based on two crucial observations:

(a) the solution for a signed version of an unsigned genome is a feasible solution;

(b) the solution for one of the 2^n signed versions of a unsigned genomes is an optimal solution.

Thus, $\mathcal{GA}_S$ considers an initial population that consists of individuals that are signed genomes randomly built from the input unsigned genome, and the fitness function is the translocation distance from these individuals to the corresponding identity unsigned genome, that is computed in linear time using the algorithm for solving STD introduced in [4]. The pseudo-code of the $\mathcal{GA}_S$ is shown in Algorithm 1.

Algorithm 1: Genetic Algorithm for UTD ($\mathcal{GA}_S$)

```
Input: Unsigned genomes $A$ and $B$ (identity genome)
Output: Number of translocations for transforming $A$
        into $B$
1 Generate the initial population of signed genomes;
2 Compute fitness of the initial population;
3 for $i = 1$ to numberGenerations 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 offspring;
8     Perform replacement of the worst individuals;
```

C. Memetic Algorithms

Memetic Algorithms (for short, MA) are the combination of evolutionary algorithms and local search heuristics [13], [14].

As mentioned in the introduction, da Silveira et al. proposed memetic techniques for improving $\mathcal{GA}_S$ and introduce algorithm $\mathcal{MA}_S$ in [11]. The distinguishing feature of $\mathcal{MA}_S$ is the inclusion of several stages of local search. The local search procedure consist in modifying the sign of a gene for an individual of the population, and then verifying if this modification indeed improves the fitness value of the individual. In the case in which the fitness is not improved the individual is maintained otherwise it is substituted. The local search procedure is shown in Algorithm 2, and $\mathcal{MA}_S$ in Algorithm 3.

Algorithm 2: Local Search

```
Input: A signed genome $A$
Output: An improved signed genome
1  bestFitness = Compute the fitness of $A$;
2  for $i = 1$ to numberIterations do
3      Generate a random position $k$ for a gene of $A$;
4      Modify the sign of gene at position $k$;
5      $fitness$ = Compute the fitness of the modified $A$;
6      if fitness ¡ bestFitness then
7          Update new fitness of $A$;
8          break;
9      else
10         Recover last state of $A$;
```

Algorithm 3: Memetic Algorithm for UTD ($\mathcal{MA}_S$)

```
Input: Unsigned genomes $A$ and $B$ (identity genome)
Output: Number of translocations for transforming $A$
        into $B$
1  Generate the initial population of signed genomes;
2  Compute fitness of the initial population;
3  Improve initial population by applying Local Search;
4  for $i = 1$ to numberGenerations do
5      Perform the selection and save the best solution
       found;
6      Apply the crossover operator;
7      Apply the mutation operator;
8      Compute the fitness of the offspring;
9      Perform replacement of the worst individuals;
10     Apply Local Search to the current population;
11     if entropyThreshold is reached then
12         Restart population improved by Local Search;
```

III. Parallel MA Approaches for UTD

According to Cantu-Paz [15] there exist three main models for parallelizing a genetic algorithm:

1) global single-population master-slave;
2) single-population fine-grained;
3) multiple-population coarse-grained.

The first and third models are suitable for the purposes of this research since the used computational platforms are single computers with multiple cores. The second model is more suitable for experiments with massively parallel and clusters of computers.

Following, the models (and variations) used for parallelizing the memetic algorithm for UTD are presented. The standard commands `MPI_Send` and `MPI_Recv` were used for the exchange of messages among processes, both commands are taken from the Message Passing Interface `MPI` library.

A. MA with Parallel Calculation of the Fitness Function

This parallelization (called $\mathcal{MA}_{\text{PF}}$) is based on the model of global single-population master-slave. The algorithm $\mathcal{MA}_{\text{PF}}$ maintains a single population of size $n \log n$ (as adopted by the $\mathcal{MA}_{\text{S}}$) in the master process, the slaves are used for the fitness calculation which is done in parallel. The aim of this parallelization is to speed up the runtime of $\mathcal{MA}_{\text{S}}$, while keeping precision of results (number of translocations).

The Algorithm 4 shows the pseudo-code of the master and slave processes, while Algorithm 5 shows the evaluation of the fitness in parallel.

Algorithm 4: MA with Parallel Calculation of the Fitness Function for UTD ($\mathcal{MA}_{\text{PF}}$)

```
   Input: Unsigned genomes A and B (identity genome)
   Output: Number of translocations for transforming A
           into B
   /* Pseudocode for the master process:
      */
1  Generate initial population of signed genomes;
2  Compute fitness of the initial pop. in parallel (Alg. 5);
3  Improve initial pop. by applying Local Search (fitness
   eval. using Alg. 5);
4  for $i = 1$ to numberGenerations do
5      Perform the selection and save the best solution
       found;
6      Apply the crossover operator;
7      Apply the mutation operator;
8      Compute the fitness of the offspring in parallel (Alg.
       5);
9      Perform replacement of the worst individuals;
10     Apply Local Search to the current population;
11     if entropyThreshold is reached then
12         Restart population improved by Local Search;
   /* Pseudocode for a slave process:    */
13 while 1 do
14     Receive an individual (signed genome) from the
       master process (MPI_Send);
15     Evaluate the fitness;
16     Send the result to the master process (MPI_Recv);
```

B. MA with Multiple Populations and Exchange of Individuals

This parallelization (called $\mathcal{MA}_{\text{MPE}}$) is based on the model of multiple-population coarse-grained. The algorithm $\mathcal{MA}_{\text{MPE}}$ maintains multiple different populations where each population, of size $n \log n$, is located in a separate process. In each generation a stage is implemented in which all processes

Algorithm 5: Evaluation of Fitness in Parallel

```
1 for all the slaves processes do
2     Send an individual (signed genome) to a slave
      process (MPI_Send);
3 while individual without fitness value do
4     Receive result from a slave process (MPI_Recv);
5     Send an individual (signed genome) to a slave
      process (MPI_Send);
6 for all the slaves processes do
7     Receive result from a slave process (MPI_Recv);
```

exchange their best individuals, all processes restart their breeding cycle before they reach a barrier (`MPI_Barrier`), which is executed for synchronizing the processes. As previously mentioned, the aim of this parallelization is to improve precision by exploring a wider search space.

Algorithm 6 shows the pseudo-code of $\mathcal{MA}_{\text{MPE}}$. Algorithm 7 describes the procedure for exchanging individuals in parallel.

A variant of the Algorithm 6 is proposed, called $\mathcal{MA}_{\text{MPnE}}$. The main feature of this variant regarding 6 is that there is no exchange of individuals. The pseudo-code of $\mathcal{MA}_{\text{MPnE}}$ is obtained by eliminating line 6 and 7 from Algorithm 6. The aim of doing the last modification was to test the behaviour of the algorithm without exchanging individuals. In principle this modification brings out a much more different mechanism than $\mathcal{MA}_{\text{S}}$ since it will apply the whole genetic process over isolated populations.

The parameter p that appears in the Algorithm 7 is used for controlling whether or not to replace an individual.

IV. Experiments and Results

The implementation of the algorithms were developed using the `MPI` library of the `C` language. The experiments were executed on a computer with 64GB of RAM, and two processors Xeon E5-2620 with hyper-threading. Each processor has 6 cores with a CPU clock rate of 2.4Ghz.

The parameters used by the parallel implementations are the same established for $\mathcal{MA}_{\text{S}}$ in [11], these parameters are shown in Table I. Additionally, a new parameter p was included for controlling the replacement of a worse individual by the best individual received from other process, as shown in the Algorithm 7. Fine-tuning of this parameter p was performed through preliminary experiments. Results from the sensitivity analysis performed over this parameter showed that the best results are obtained when using $p = 0.2$.

Also, a sensitivity analysis was performed over the number of processors to be used by $\mathcal{MA}_{\text{PF}}$. The number of processors was varied from 3 to 24, and results showed that using just 12 processor gives the best speed-up. For the case of $\mathcal{MA}_{\text{MPE}}$ and $\mathcal{MA}_{\text{MPnE}}$ the number of processor was fixed to 24 in order to obtain a better precision of the results.

Algorithm 6: Parallel MA with Multiple Populations for UTD ($\mathcal{MA}_{\text{MPE}}$), where each process maintains a different initial population

```
Input: Unsigned genomes A and B (identity genome)
Output: Number of translocations for transforming A into B
1  Generate initial population of signed genomes;
2  Compute fitness of the initial population;
3  Improve initial pop. by applying Local Search;
4  for $i = 1$ to numberGenerations do
5      Perform the selection and save the best solution found;
6      Send the best individual, and receive individuals (Alg. 7);
7      MPI_Barrier;
8      Apply the crossover operator;
9      Apply the mutation operator;
10     Compute the fitness of the offspring;
11     Perform replacement of the worst individuals;
12     Apply Local Search to the current population;
13     if entropyThreshold is reached then
14         Restart population improved by Local Search;
15     Save the best individual into the process 0;
```

Algorithm 7: Procedure for sending and receiving individuals in parallel

```
1  for $i = 1$ to numberProcesses do
2      if $i$ is not current process then
3          Send the best individual to process $i$ (MPI_Send);
4  for $i = 1$ to numberProcesses do
5      if $i$ is not current process then
6          Receive best individual from process $i$ (MPI_Recv);
7          Generate a float number $r$ between 0 and 1;
8          if $r < p$ then
9              Replace a worse individual by the received individual;
```

TABLE I
PARAMETERS SETTING FOR THE PARALLEL MEMETIC ALGORITHMS

Parameter	Best Value
Crossover probability	0.90
Mutation probability	0.02
Percentage for selection	80%
Percentage for replacement	70%
Percentage for local search	60%
Percentage for preservation	60%
Threshold entropy	0.3

A. Experiments with Hundred Synthetic Genomes

The input data for this experiment consists of six packages of hundred genomes, that were randomly generated, each package with synthetic genomes of lengths varying by 10 genes from 100 until 150. For all genomes the number of chromosomes was fixed from 3 to 5. For each package of hundred genomes the algorithms $\mathcal{MA}_{\text{S}}$, $\mathcal{MA}_{\text{MPE}}$ and $\mathcal{MA}_{\text{MPnE}}$ were executed in order to solve UTD. The results of this experiment are shown in Tables II, III, and IV where the value in each cell represents the average translocation distance computed for each package of hundred genomes. From these tables, it can be seen (highlighted in bold font) that $\mathcal{MA}_{\text{MPnE}}$ computes the best results, that is the minimum average number of translocations.

TABLE II
AVERAGE RESULTS FOR THE EXPERIMENT WITH HUNDRED SYNTHETIC GENOMES HAVING 3 CHROMOSOMES

	3 Chromosomes		
Length	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$
100	64.31	64.82	**63.80**
110	71.43	72.10	**70.74**
120	78.62	79.50	**77.73**
130	86.45	87.45	**85.34**
140	94.16	95.46	**92.89**
150	100.55	102.08	**99.17**

TABLE III
AVERAGE RESULTS FOR THE EXPERIMENT WITH HUNDRED SYNTHETIC GENOMES HAVING 4 CHROMOSOMES

	4 Chromosomes		
Length	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$
100	61.24	61.53	**60.72**
110	68.07	68.65	**67.49**
120	75.01	75.73	**74.30**
130	81.53	82.41	**80.77**
140	88.85	90.13	**87.91**
150	95.43	96.83	**94.40**

TABLE IV
AVERAGE RESULTS FOR THE EXPERIMENT WITH HUNDRED SYNTHETIC GENOMES HAVING 5 CHROMOSOMES

	5 Chromosomes		
Length	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$
100	57.95	58.16	**57.69**
110	64.78	65.12	**64.35**
120	72.15	72.62	**71.62**
130	78.48	79.25	**77.82**
140	85.25	86.23	**84.58**
150	91.89	93.07	**91.20**

B. Experiments with Benchmark (Single) Genomes

For this experiment the benchmarks proposed in [11] were used as inputs. The algorithms $\mathcal{MA}_{\text{S}}$, $\mathcal{MA}_{\text{MPE}}$, and $\mathcal{MA}_{\text{MPnE}}$ were executed fifty times for each benchmark genome and the following measures were calculated: mean, median, minimum and maximum of the resulting number of translocations.

The results of this experiment are shown in Tables V and VI. From these results, it can be observed (highlighted in bold font) that $\mathcal{MA}_{\text{MPnE}}$ computes the best results for all measures.

Also, a statistical comparison was performed, in order to verify that $\mathcal{MA}_{\text{MPnE}}$ is in fact the best algorithm. The following

TABLE V
MEAN AND MEDIAN RESULTS FOR THE EXPERIMENT WITH BENCHMARK GENOMES

	Mean			Median		
Benchmark	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$
L150C2	101.56	104.00	**100.00**	102.00	104.00	**100.00**
L150C3	108.34	109.00	**106.00**	108.00	109.00	**106.00**
L150C4	99.82	103.00	**99.00**	100.00	103.00	**99.00**
L150C5	96.08	97.00	**95.00**	96.00	97.00	**95.00**
L150C6	84.06	86.00	**84.00**	84.00	86.00	**84.00**
L150C7	90.22	92.00	**90.00**	90.00	92.00	**90.00**
L150C8	76.90	77.00	**76.00**	77.00	77.00	**76.00**
L150C9	78.12	78.00	**78.00**	78.00	78.00	**78.00**
L150C10	77.00	78.00	**77.00**	77.00	78.00	**77.00**

TABLE VI
MINIMUM AND MAXIMUM RESULTS FOR THE EXPERIMENT WITH BENCHMARK GENOMES

	Minimum			Maximum		
Benchmark	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$	$\mathcal{MA}_{\text{S}}$	$\mathcal{MA}_{\text{MPE}}$	$\mathcal{MA}_{\text{MPnE}}$
L150C2	100.00	104.00	**100.00**	103.00	104.00	**100.00**
L150C3	107.00	109.00	**106.00**	110.00	109.00	**106.00**
L150C4	99.00	103.00	**99.00**	101.00	103.00	**99.00**
L150C5	95.00	97.00	**95.00**	98.00	97.00	**95.00**
L150C6	84.00	86.00	**84.00**	86.00	86.00	**84.00**
L150C7	90.00	92.00	**90.00**	91.00	92.00	**90.00**
L150C8	76.00	77.00	**76.00**	78.00	77.00	**76.00**
L150C9	78.00	78.00	**78.00**	79.00	78.00	**78.00**
L150C10	77.00	78.00	**77.00**	77.00	78.00	**77.00**

TABLE VII
RESULTS OF THE WILCOXON RANK SUM TEST, WHERE NUMBERS ARE THE MEDIANS, AND "$s+$" REPRESENTS A STATISTICALLY SIGNIFICANCE DIFFERENCE, OTHERWISE THE "$s-$" IS USED

Benchmark	$\mathcal{MA}_{\text{MPnE}}$	$\mathcal{MA}_{\text{S}}$		$\mathcal{MA}_{\text{MPE}}$	
L150C2	100.00	102.00	$s+$	104.00	$s+$
L150C3	106.00	108.00	$s+$	109.00	$s+$
L150C4	99.00	100.00	$s+$	103.00	$s+$
L150C5	95.00	96.00	$s+$	97.00	$s+$
L150C6	84.00	84.00	$s-$	86.00	$s+$
L150C7	90.00	90.00	$s+$	92.00	$s+$
L150C8	76.00	77.00	$s+$	77.00	$s+$
L150C9	78.00	78.00	$s+$	78.00	$s-$
L150C10	77.00	77.00	$s-$	78.00	$s+$

methodology as discussed in [16], [17], [18] was applied: firstly, the *Kolmogorov-Smirnov* test was applied in order to determine the non-normality of the samples (results from 50 runs); after this, the *Wilcoxon Rank Sum* test was performed in order to compare the medians of the samples of each algorithm. The results of this test are shown in the Table VII, using a significance level of 5% (p-value $\leq$ 0.05).

In Table VII, $s+$ can be interpreted as a different behavior between $\mathcal{MA}_{\text{MPnE}}$ and the other algorithm (either $\mathcal{MA}_{\text{S}}$ or $\mathcal{MA}_{\text{MPE}}$), in case the median is smaller we can say that $\mathcal{MA}_{\text{MPnE}}$ has the best performance. Otherwise $s-$ can be interpreted as a similar behavior between $\mathcal{MA}_{\text{MPnE}}$ and the other algorithm.

Using the same benchmarks, an additional experiment was performed for measuring the speed-up of $\mathcal{MA}_{\text{PF}}$. For this experiment, the algorithms $\mathcal{MA}_{\text{S}}$ and $\mathcal{MA}_{\text{PF}}$ were executed ten times for each benchmark and then the average time

TABLE VIII
RESULTS OF EXPERIMENT FOR MEASURING THE SPEED-UP USING BENCHMARK PERMUTATIONS

Benchmark	$\mathcal{MA}_{\text{S}}$ (sec)	$\mathcal{MA}_{\text{PF}}$ (sec)	Speed-up	$\mathcal{MA}_{\text{S}}$ #individuals	$\mathcal{MA}_{\text{PF}}$ #individuals
L150C2	27.08	5.04	5.37	11773	83216
L150C3	29.05	5.34	5.44	1111	78852
L150C4	29.99	5.52	5.43	10860	78850
L150C5	31.54	5.77	5.47	10542	79680
L150C6	32.61	6.49	5.02	10269	73110
L150C7	33.92	6.45	5.26	10106	74023
L150C8	35.62	6.89	5.17	9913	70654
L150C9	36.86	7.09	5.20	9561	62831
L150C10	39.28	8.13	4.83	9345	54585

TABLE IX
RESULTS OF EXPERIMENTS WITH BIOLOGICALLY-BASED GENOMES

Pairs of organisms	$\mathcal{MA}_{\text{S}}$ (sec)	$\mathcal{MA}_{\text{PF}}$ (sec)	Speed-up	$\mathcal{MA}_{\text{S}}$ #individuals	$\mathcal{MA}_{\text{PF}}$ #individuals
Human-Cat	54.24	10.80	5.02	7198	43238
Human-Mouse	50.64	10.97	4.62	7774	42968
Cat-Mouse	50.18	11.09	4.52	7874	42618

(in seconds) was calculated. Also, the average number of individuals processed per second was calculated. An individual is considered to be processed whenever its fitness value was calculated. The speed-up is calculated by dividing the results of $\mathcal{MA}_{\text{S}}$ by $\mathcal{MA}_{\text{PF}}$. The results of this experiment are shown in Table VIII.

In Table VIII can be observed that for all benchmarks the algorithm $\mathcal{MA}_{\text{PF}}$ has a speed-up factor of around 5 regarding $\mathcal{MA}_{\text{S}}$. Also, the average number of individuals processed per second by the algorithm $\mathcal{MA}_{\text{PF}}$ is always higher than those obtained by $\mathcal{MA}_{\text{S}}$.

C. Experiments with Biologically-Based Genomes

For this experiment the algorithms $\mathcal{MA}_{\text{S}}$ and $\mathcal{MA}_{\text{PF}}$ were executed ten times for each input (biologically-based genomes). The input data was taken from [19] and consist in the gene order of 114 markers on the genome (complete DNA sequence) of three species: human, cat, and mouse. Before using these gene order sequences a pre-processing was applied over the sequences in [11], which is explained in the following paragraph.

Each input is built considering only those genes in common among the genomes of pairs of different organisms (human, cat and mouse) and the genes of the second genome of each pair are represented by naturals in such a way that its genome is represented as an identity. In the end, the three inputs remains with 18 chromosomes and 147 genes, where auxiliary genes were used at the extreme points of each chromosome in order to fulfil Property 1.

The aim of this experiment was to measure the speed-up of the algorithms for real data. The results are shown in Table IX.

From Table IX, it can be observed that the algorithm $\mathcal{MA}_{\text{PF}}$ brings a speed-up factor greater than 4.5 for all cases. Additionally, as for the experiments with benchmark genomes,

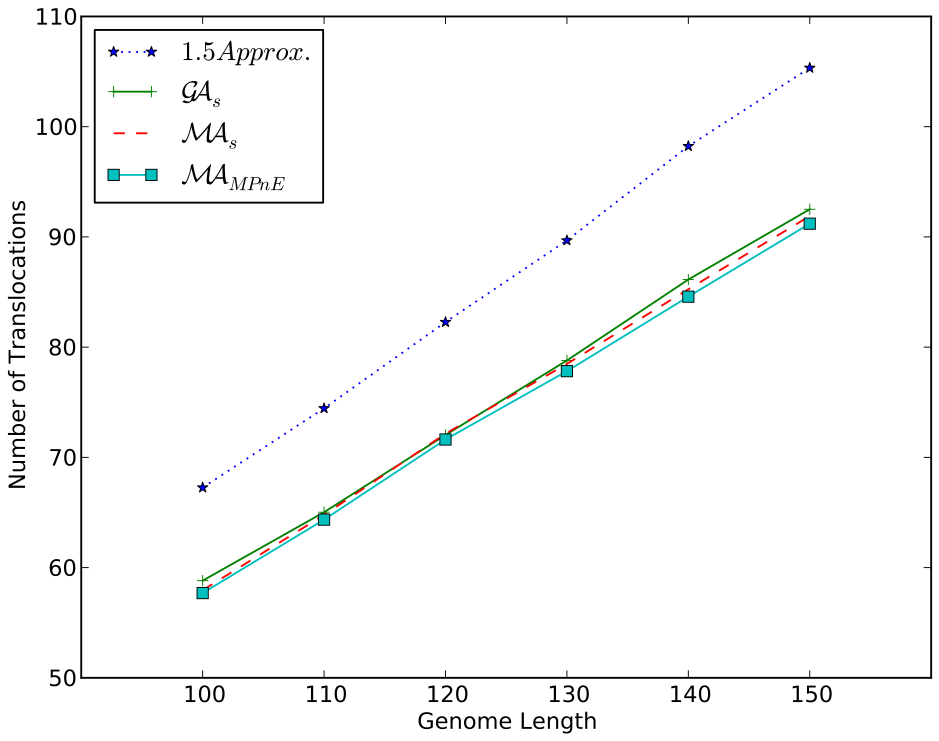

Fig. 1. Comparison of algorithms using hundred genomes

the number of individuals processed per second by the algorithm $\mathcal{MA}_{\text{PF}}$ is higher than those obtained by the algorithm $\mathcal{MA}_{\text{S}}$.

V. Discussion

A comparison of the precision of the $1.5+\varepsilon$-approximation algorithm (For short, 1.5*Approx*), the genetic algorithm ($\mathcal{GA}_{\text{S}}$), the memetic algorithm ($\mathcal{MA}_{\text{S}}$), and the parallel memetic algorithm($\mathcal{MA}_{\text{MPnE}}$ version) for UTD is shown in the Figure 1. The data for the $1.5+\varepsilon$-approximation and the genetic algorithms were taken from [10]. These results are from experiments with hundred genomes of lengths from 100 to 150, with 5 chromosomes.

In the Figure 1, it can be observed that the $\mathcal{MA}_{\text{MPnE}}$ algorithm computes the best results among all algorithms, providing distances that in average have the minimum number of translocations.

VI. Conclusions and future work

This paper proposed two approaches for parallelizing the memetic algorithm $\mathcal{MA}_{\text{S}}$ introduced by the authors in [11]. The first approach, $\mathcal{MA}_{\text{PF}}$, is based on a model that uses a single population where the calculation of the fitness is performed in parallel. The second approach, $\mathcal{MA}_{\text{MPE}}$, is based on a model with multiple populations with exchange of individuals in each generation. Also, a variation of the latter approach was proposed, called $\mathcal{MA}_{\text{MPnE}}$, where there is not exchange of individuals of different populations.

Several experiments were performed using sets of hundred genomes randomly generated, benchmark genomes, and biologically-based genomes.

From the experiments with set of hundred genomes it can be observed that $\mathcal{MA}_{\text{MPnE}}$ has the best results in terms of accuracy providing sorting solutions with the smallest number of translocations. From the experiments with benchmark genomes it can be observed that $\mathcal{MA}_{\text{MPnE}}$ has the best results for different measures (mean, median, minimum, and maximum), these results were confirmed by statistical tests. Using the same benchmarks an additional experiment was performed for measuring the speed-up of $\mathcal{MA}_{\text{PF}}$ over $\mathcal{MA}_{\text{S}}$. This experiment showed that $\mathcal{MA}_{\text{PF}}$ has a speed-up factor higher than 5 for all cases. Finally, from the experiment with biologically-based genomes, $\mathcal{MA}_{\text{PF}}$ showed a speed-up factor higher than 4.5 for all cases.

As a future work, we are planning to improve $\mathcal{MA}_{\text{MPE}}$ by testing other models for exchanging individuals, and then try to outperform the results of $\mathcal{MA}_{\text{MPnE}}$, in which there is no exchange of individuals. Of course, the more interesting challenge is applying the proposed genetic mechanisms for the construction of phylogenetic threes not only with translocation distances but also with other measures such as the reversal distance.

Acknowledgment

The first two authors are funded by CAPES graduate scholarships and the third by a CNPq grant.

References

[1] J. D. Kececioglu and R. Ravi, "Of mice and men: algorithms for evolutionary distances between genomes with translocation," in *Proc. of the sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1995, pp. 604–613.

[2] S. Hannenhalli, "Polynomial-time algorithm for computing translocation distance between genomes," *Discrete App. Math.*, vol. 71, no. 1, pp. 137–151, 1996.

[3] L. Wang, D. Zhu, X. Liu, and S. Ma, "An $o(n^2)$ algorithm for signed translocation," *J. of Comp. and Sys. Sciences*, vol. 70, no. 3, pp. 284–299, 2005.

[4] A. Bergeron, J. Mixtacki, and J. Stoye, "On sorting by translocations," *J. of Comp. Biology*, vol. 13, no. 2, pp. 567–578, 2006.

[5] A. Bergeron and J. Stoye, "On the similarity of sets of permutations and its applications to genome comparison," in *Computing and Combinatorics*. Springer, 2003, pp. 68–79.

[6] D. Zhu and L. Wang, "On the complexity of unsigned translocation distance," *Theor. Comput. Sci.*, vol. 352, no. 1, pp. 322–328, 2006.

[7] Y. Cui, L. Wang, and D. Zhu, "A 1.75-approximation algorithm for unsigned translocation distance," *J. of Comp. and Sys. Sciences*, vol. 73, no. 7, pp. 1045–1059, 2007.

[8] H. Jiang, L. Wang, B. Zhu, and D. Zhu, "A (1.408+ ε)-approximation algorithm for sorting unsigned genomes by reciprocal translocations," in *Frontiers in Algorithmics*. Springer, 2014, pp. 128–140.

[9] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[10] L. A. da Silveira, J. L. Soncco-Álvarez, T. A. de Lima, and M. Ayala-Rincón, "Computing Translocation Distance by a Genetic Algorithm," in *2015 Latin American Computing Conference, CLEI 2015, Arequipa, October, 2015*. IEEE, 2015, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1109/CLEI.2015.7359994

[11] L. A. da Silveira, J. L. Soncco-Álvarez, T. A. de Lima, and M. Ayala-Rincón, "Memetic and Opposition-Based Learning Genetic Algorithms for Sorting Unsigned Genomes by Translocations," in *Proc. 7th World Congress on Nature and Biologically Inspired Computing, NaBIC 2015, Pietermaritzburg, December, 2015*, ser. Advances in Intelligent Systems and Computing. Springer, 2016, vol. 419, pp. 73–85. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-27400-3_7

[12] Y. Cui, L. Wang, D. Zhu, and X. Liu, "A (1.5+ε)-approximation algorithm for unsigned translocation distance," *IEEE/ACM T. on Comp. Biology and Bioinformatics*, vol. 5, no. 1, pp. 56–66, 2008.

[13] P. Moscato *et al.*, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," *Caltech concurrent computation program, C3P Report*, vol. 826, p. 1989, 1989.

[14] P. Moscato and C. Cotta, "An introduction to memetic algorithms," *Inteligencia artificial, Revista iberoamericana de inteligencia artificial*, vol. 19, pp. 131–148, 2003.

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

[16] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[17] J. J. Durillo, J. García-Nieto, A. J. Nebro, C. A. C. Coello, F. Luna, and E. Alba, "Multi-objective particle swarm optimizers: An experimental comparison," in *Evolutionary Multi-Criterion Optimization*. Springer, 2009, pp. 495–509.

[18] D. M. Muñoz, C. H. Llanos, L. Coelho, and M. Ayala-Rincón, "Opposition-based shuffled pso with passive congregation applied to fm matching synthesis," in *Evolutionary Computation (CEC), 2011 IEEE Congress on*. IEEE, 2011, pp. 2775–2781.

[19] G. Bourque and P. A. Pevzner, "Genome-scale evolution: reconstructing gene orders in the ancestral species," *Genome research*, vol. 12, no. 1, pp. 26–36, 2002.