Memetic and Opposition-Based Learning Genetic Algorithms for Sorting Unsigned Genomes by Translocations

[†]Lucas A. da Silveira, [†]José L. Soncco-Álvarez, [‡]Thaynara A. de Lima, and [†*]Mauricio Ayala-Rincón

Departaments of [†]Computer Science and [*]Mathematics, Universidade de Brasília
[‡]Department of Mathematics, Universidade de Goiás — Campus II

Abstract. A standard genetic algorithm for sorting unsigned genomes by translocations is improved in two different manners: 1. a memetic algorithm ($\mathcal{GA}_{\text{MA}}$) is provided, which embeds a new stage of local search, based on the concept of mutation applied in only one gene; 2. an opposition-based learning ($\mathcal{GA}_{\text{OBL}}$) mechanism is provided, which explores the concept of internal opposition applied to a chromosome. The proposed approaches include a convergence control mechanism of the population using the Shannon entropy. Additionally, non-parametric statistical tests were performed to compare the proposed algorithms with the standard genetic algorithm. For the experiments, both biological and synthetic genomes were used. The results from these experiments showed that the $\mathcal{GA}_{\text{MA}}$ outperforms the $\mathcal{GA}_{\text{OBL}}$ and the genetic algorithm. The statistical tests confirmed these results showing that the $\mathcal{GA}_{\text{MA}}$ has a better performance regarding the other algorithms.

Keywords: Sorting permutations, Sorting Unsigned Genomes, Genetic Algorithms, Memetic Algorithms, Opposition-Based Learning Algorithms.

1 Introduction

In order to estimate the evolutionary relationships between species, molecular biologists compare gene and genome sequences of different species to reveal the degree of similarity between them. While algorithms for gene comparison are based on local mutations such as insertions and deletions of biological data, algorithms for genome comparison are based on global mutations such as reversals, transpositions and translocations. In this paper we adopt the model of genetic evolution in genomes by translocations. The translocation operation involves the interchange of blocks of genes between the chromosomes of a genome. So the genome comparison can be modelled as an optimisation problem which consists in finding the minimum number of translocations necessary to transform one genome into another one. This problem is known as the translocation distance problem, which has two versions: one using *signed* genomes and the other using *unsigned* genomes, where the difference remains respectively in considering or not the orientation of the genes between the chromosomes.

The first polynomial algorithm ($O(n^3)$) for the signed translocation distance problem (STD) was proposed by Hannenhalli in [9]. Almost a decade later, Wang et al in [19] proposed an $O(n^2)$ algorithm taking also advantage of the techniques

originally proposed by Hannenhalli. More recently, Bergeron et al in [2] improved the complexity to $O(n)$.

While STD belongs to the complexity class $\mathcal{P}$, the unsigned translocation distance problem (UTD) is a problem of high complexity. Indeed, Zhu and Wang proved that this is an $\mathcal{NP}$-hard problem in [20]. Thus, developments of approximate solutions for this problem are necessary. In [4] Cui et al presented an 1.75-approximation algorithm for computing the translocation distance between unsigned genomes, and further improved the ratio to $1.5+\varepsilon$ in [5]. The best known approximation algorithm in the literature has ratio $1.408+\varepsilon$ and was proposed in [10]. More recently, we proposed a genetic algorithm (GA) approach in [6], in which the fitness function is given as the translocation distance for signed versions of the unsigned genomes, that is linearly computed as in [2]. To verify the quality of the solutions computed by the GA, an implementation of the $1.5+\varepsilon$ approximation algorithm was used as control mechanism. Results of the experiments in [6] showed that on average the GA computes better results than the $1.5+\varepsilon$-approximation algorithm.

This paper proposes two hybrid evolutionary algorithms: a memetic algorithm ($\mathcal{GA}_{\text{MA}}$) and an opposition-based learning (OBL) genetic algorithm ($\mathcal{GA}_{\text{OBL}}$) for the UTD problem. $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ embed the local search and OBL in the following stages of the GA: improvement of the initial population, restart of the population, and after the breeding cycle. For controlling the converge of the population the Shannon entropy was used. The quality of the results obtained by the proposed algorithms were compared with the results computed with the algorithms proposed in [6] and [5] for groups of hundred unsigned genomes of different length, single unsigned genomes, and genomes based on biological data (built from [3]). Additionally, non-parametric statistical tests were performed to compare the proposed algorithms and the standard GA (in [6]). Results from these tests showed that the $\mathcal{GA}_{\text{MA}}$ has better performance compared with the other algorithms, and that there is no significant difference in the performance of the $\mathcal{GA}_{\text{OBL}}$ and the GA. Algorithms $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ were implemented in `C` and for the benefit of the reviewers, the code is available at `www.mat.unb.br/~ayala/publications.html`.

2 Background

Definitions and Terminology

In a simplified model, a *signed* chromosome can be represented by a non empty sequence of integer numbers of the form $X = (x_1, \ldots, x_l)$, where $x_i \in \{\pm 1, \ldots \pm n\}$ for all $i \in \{1, \ldots, n\}$, each integer denotes a gene and $|x_i| \neq |x_j|$, whenever $i \neq j, i, j \in \{1, \ldots, n\}$. With the restriction that for all $i \in \{1, \ldots, l\}$, $x_i \in \{1, \ldots, n\}$, the chromosome is called *unsigned*. Chromosomes do not have orientation, thus if $X = (x_1, \ldots, x_l)$ is a signed chromosome then X and $X' = (-x_l, \ldots, -x_1)$ are equal, whereas if X is an unsigned chromosome, X and $X'' = (x_l, \ldots, x_1)$ are equal. A genome G with N chromosomes and n genes is a list of chromosomes of the form $(x_{11}, \ldots, x_{1r_1}) \ldots (x_{N1}, \ldots, x_{Nr_N})$, where for all its genes $|x_{ij}| \neq |x_{km}|$ whenever $i \neq k$ or $j \neq m$, and $\sum_{i=1}^{N} r_i = n$. Unsigned and signed genomes consist respectively of unsigned and signed chromosomes.

Let $X = (x_1, \ldots, x_i, \ldots x_l)$ and $Y = (y_1, \ldots, y_j, \ldots, y_m)$ be two chromosomes of a signed genome. On the one hand, a *translocation by prefix-prefix* , denoted as $\rho(X, Y, x_i, y_j)$, transforms X and Y into the chromosomes $X^\rho = (x_1, \ldots, x_i, y_{j+1}, \ldots, y_m)$ and $Y^\rho = (y_1, \ldots, y_j, x_{i+1}, \ldots, x_l)$. On the other hand, a *translocation by prefix-suffix*, $\theta(X, Y, x_i, y_j)$, produces the new chromosomes $X^\theta = (x_1, \ldots, x_i, -y_j, \ldots, -y_1)$ and $Y^\theta = (-y_m, \ldots, -y_{j+1}, x_{i+1}, \ldots, x_l)$. Notice that a translocation by prefix-suffix $\theta(X, Y, x_i, y_j)$ can be mimicked by a translocation $\rho(X, Y', x_i, -y_{j+1})$ by prefix-prefix and vice-versa, where $Y' = (-y_m, \ldots, -y_1)$. For chromosomes $X = (x_1, \ldots, x_i, \ldots x_l)$ and $Y = (y_1, \ldots y_j, \ldots, y_m)$ of an unsigned genome, a *translocation by prefix-prefix* $\rho(X, Y, x_i, y_j)$ transforms X and Y into $X^\rho = (x_1, \ldots, x_i, y_{j+1}, \ldots, y_m)$ and $Y^\rho = (y_1, \ldots, y_j, x_{i+1}, \ldots, x_l)$ and, a *translocation by prefix-suffix* $\theta(X, Y, x_i, y_j)$ creates the chromosomes $X^\theta = (x_1, \ldots, x_i, y_j, \ldots, y_1)$ and $Y^\theta = (y_m, \ldots, y_{j+1}, x_{i+1}, \ldots, x_l)$.

The translocation distance problems are defined as follows.

Signed translocation distance problem (STD): consider two signed genomes A and B, with the same number of genes and chromosomes, where the genes of B are positive integers in increasing order. The STD consists in finding the minimum number of translocations needed to transform A into B.

Unsigned translocation distance problem (UTD): the UTD is defined as the STD but restricted to unsigned genomes.

Genomes as B above, are called *identity* genomes. Given a signed chromosome $X = (x_1, \ldots, x_l)$, the elements x_1 and $-x_l$ are called *tails* of X. Two signed genomes A and B are said to be *co-tails* if the sets T_A and T_B composed by the tails of the chromosomes of A and B, respectively, are equal. For example, $A = (+1, +6, +7)(-4, -3, -2, -5)$ and $B = (+1, +2, +3, +4)(+5, +6, +7)$ are co-tails. When $X = (x_1, \ldots, x_l)$ is an unsigned chromosome, x_1 and x_l are its *tails* and *co-tails* are correspondingly defined.

Notice that translocations by prefix-prefix or prefix-suffix do not modify the tails of a genome. Thus, in order to be able to transform a genome A into B by translocations, A and B must satisfy the property below.

Property 1 The genomes A and B have the same number of genes (and chromosomes) and A and B are co-tails.

Notice that when A and B are co-tails by renaming the genes, B can be rewritten as an identity. Thus, without loss of generality, whenever A and B are co-tails one can assume that B is an identity.

STD belongs to the complexity class $\mathcal{P}$ ([9]), while UTD is $\mathcal{NP}$-hard ([20]). The construction of polynomial algorithms for solving STD as well as the proof that UTD is $\mathcal{NP}$-hard explore the relation between these problems and the problem of maximum cycle decomposition of the *breakpoint graph* of a genome. This data structure is defined below and illustrated in Fig. 1.

Given a (signed or unsigned) chromosome $X = (x_1, \ldots, x_l)$ of a genome, the elements x_i and x_{i+1}, $1 \leq i \leq l-1$ are said to be *adjacent*; otherwise, the elements are *not adjacent*. Elements in different chromosomes are not adjacent.

Consider two signed genomes A and B satisfying the Property 1. The breakpoint graph $G_s(A, B)$ is built as follows: for each chromosome $X = (x_1, \ldots, x_l)$ of

A, we associate to each x_i an ordered pair of vertices $(l(x_i), r(x_i)) = (-x_i, +x_i)$; there is a black edge between $r(x_i)$ and $l(x_{i+1})$, i.e., between $+x_i$ and $-x_{i+1}$, for every $1 \leq i \leq l-1$; if $+i$ and $+(i+1)$ are adjacent in B, there is a gray edge between $+i$ and $-(i+1)$.

Now, consider two unsigned genomes A and B satisfying the Property 1. The breakpoint graph $G_u(A, B)$ is built as follows: the vertices are given by the genes of A; furthermore, for each chromosome $X = (x_1, \ldots, x_l)$ of A, there is a black edge between x_i and x_{i+1}, $1 \leq i \leq l-1$, and there is a gray edge between i and $i+1$ (since i and $i+1$ are adjacent in B).

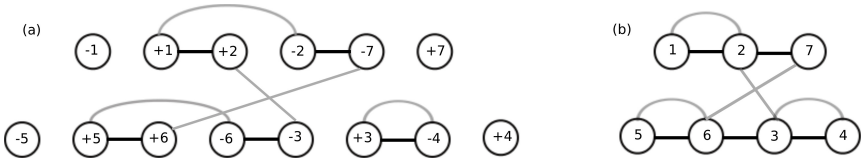

Fig. 1. Breakpoint graphs for (a) $A = (+1, -2, +7)(+5, -6, +3, +4)$ and $B = (+1, +2, +3, +4)(+5, +6, +7)$; and (b) $A = (1, 2, 7)(5, 6, 3, 4)$ and $B = (1, 2, 3, 4)(5, 6, 7)$

The decomposition of the breakpoint graph of signed genomes into cycles with edges of alternating colors (alternating cycles) is unique, because each vertex has at most one black and one gray incident edge. However, the same is not true for unsigned genomes: roughly, the latter observation is the key to understand why STD is a polynomial problem, whereas UTD is $\mathcal{NP}$-hard; indeed, the translocation distance between two genomes is closely related with a maximum decomposition into alternating cycles of their breakpoint graph (see [20] and [9]).

A Standard Genetic Algorithm for UTD

A genetic algorithm for UTD was proposed in [6] which works as follows for an input genome A with n genes. Initially, a random population of $n \log n$ individuals, that are signed genomes with n genes, is generated based on the unsigned input genome, where each individual is obtained by randomly assigning either a positive or negative sign to each gene of A. Solutions for one signed genome are also consistent solutions for the input, so the GA searches for the minimum solution of this population of signed genomes. In each generation: the fitness function calculates the translocation distance for each individual in the population. This is done using the linear algorithm proposed in [2] for solving the STD. Then, the population is classified according to the fitness value of each individual, maintaining the best individuals at the top. The individuals selected for reproduction are part of the current better solutions for which crossover and mutation are applied producing new individuals. Finally, these new individuals are incorporated into the current population. The *GA* finishes after n generations have been completed. The pseudo-code of the *GA* proposed in [6] is shown in Algorithm 1.

The GA was implemented in [6] proved to compute solutions that have better quality than those computed with the approximate algorithm in [5].

3 Memetic and OBL Algorithms for UTD

$\mathcal{GA}_{\mathrm{MA}}$: Memetic algorithms (MA) are a class of algorithms that combine population-based search as in the genetic algorithms with one or more phases of local search, and even heuristics or approximate methods [14], [11]. MAs maintain a population of individuals which perform independent explorations (local optimisation),

cooperating by means of reproduction operators, and continuously competing by means of selection and substitution operators [13].

The proposed $\mathcal{GA}_{\text{MA}}$ for UTD is based on the standard GA introduced in [6], maintaining, for inputs of length n, a population of $n \log n$ signed genomes, that are signed versions of the input and the fitness is the same as the one used in the standard GA.

Algorithm 1: Genetic algorithm for computing UTD — standard GA

Input: Unsigned genomes A and B, where B is an identity satisfying Prop. 1
Output: Number of translocations to sort genome A

```
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 current population;
8      Perform replacement of the worst individuals;
```

The $\mathcal{GA}_{\text{MA}}$ includes phases of local search into the stage of generation of the initial population and the stage of restarting population, and also, it applies local search at the end of the breeding cycle. The stage of restarting the population is executed whenever the population converges to a degenerate state, which is measured using the Shannon entropy [16] and is defined as follows: $H(S) = -\sum_i p_i \log_2 p_i$, where S is the set of different elements of the current population, and p_i is the probability of occurrence of the element i in the current population. When the entropy has a high value it is said that the population has a good diversity, but when the entropy tends to lower values it is said that the population is converging to a degenerate state.

The pseudo-code for the $\mathcal{GA}_{\text{MA}}$ for UTD is shown in Algorithm 2 excluding lines 3, 10 and 12.

Algorithm 2: Memetic and OBL Algorithms for UTD — $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$

Input: Unsigned genomes A and B, where B is an identity satisfying Prop. 1
Output: Number of translocations to sort genome A

```
1   Generate the initial population of signed genomes;
2   Compute fitness of the initial population;
3   Improve initial population by applying Local Search; /*  For GA_MA  */
3   Improve initial population by applying OBL Heuristic; /*  For GA_OBL  */
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 current population;
9       Perform replacement of the worst individuals;
10      Apply Local Search to the current population; /*  For GA_MA  */
10      Apply OBL Heuristic to the current population; /*  For GA_OBL  */
11      if entropyThreshold is reached then
12          Restart population improved by Local Search; /*  For GA_MA  */
12          Restart population improved by OBL Heuristic; /*  For GA_OBL  */
```

The local search consists in a very simple step of modifying the sign of an element of one signed genome at a random position, and verify whether this change improves the fitness. In the case that the fitness is improved it is updated, otherwise the last state of the signed genome is recovered. For each genome, the

number of possible modifications was restricted to two. The pseudo-code of the local search for one signed genome is shown in Algorithm 3.

Algorithm 3: Local Search

```
Input: A signed genome A
Output: An improved signed genome A
1  bestFitness = Compute fitness of A;
2  for i to numberIterations do
3      generate a random position k for A;
4      swap the sign of the element at position k;
5      fitness = calculate fitness of A;
6      if fitness < bestFitness then
7          update new fitness for A;
8          break;
9      else
10         recover last state of A;
```

Algorithm 4: OBL Heuristic

```
Input: A signed genome A
Output: An improved signed genome A
1  bestFitness = Compute fitness of A;
2  Ã = generate a type-I opposite for genome A;
3  fitness = calculate fitness of Ã;
4  if fitness < bestFitness then
5      keep the genome Ã;
6      discard genome A;
7  else
8      discard genome Ã;
```

$\mathcal{GA}_{\text{OBL}}$: OBL is a searching technique proposed by Tizhoosh [17]. The main idea about OBL is that when one is searching for a solution in one direction, it would be a good idea to search in an opposite way. This increases the chance of improving the solution specially when one is in a worst case scenario. In [18], [1] the type-I and type-II opposite points were defined as below.

Definition 1 (Opposite Number and Points [18, 12]).
Opposite Number. *Let x be a real number in the interval $[a, b]$, the opposite number $\tilde{x}$ is defined in the following way: $\tilde{x} = a + b - x$.*
Type-I Opposite Point. *Let $P = (x_1, \ldots, x_n)$ be an n-dimensional point with x_i being a real number in the interval $[a_i, b_i]$. The type-I opposite point is defined by $\tilde{P} = (\tilde{x_1}, \ldots, \tilde{x_n})$, where each coordinate is defined in the following way: $\tilde{x_i} = a_i + b_i - x_i \quad i = 1, 2, \ldots, n$.*
Type-II Opposite Point. *Let f be an arbitrary $\mathbb{R}^n \to \mathbb{R}$ function with image in the interval $[y_{min}, y_{max}]$. For every n-dimensional point $P = (x_1, \ldots, x_n)$, the type-II opposite point is defined by $\tilde{f}(x_1, \ldots, x_n) = y_{min} + y_{max} - f(x_1, \ldots, x_n)$.*

In simple words, the type-I opposition refers to calculating the opposite of a P point, that is $\tilde{P}$. Given a function f, the type-II opposition refers to calculating its opposite, that is a function $\tilde{f}$ with opposite images to the ones of f. In this work, for sake of simplicity just the type-I opposition is used.

The proposed opposition-based genetic algorithm ($\mathcal{GA}_{\text{OBL}}$) for UTD is also based on the GA in [6], and applies OBL exactly in the stages where $\mathcal{GA}_{\text{MA}}$ applies local search: generation of the initial population, restarting of the population and after the breeding cycle. Also, convergence of the population is controlled using the Shannon Entropy [16]. The fitness function used in the $\mathcal{GA}_{\text{OBL}}$ is the same as the used by *GA* and $\mathcal{GA}_{\text{MA}}$. The pseudo-code of the $\mathcal{GA}_{\text{OBL}}$ is shown also in the Algorithm 2 excluding lines 3, 10 and 12.

The OBL heuristic consists in applying the type-I opposition for one signed genome. This new opposite genome is kept in the population whenever its fitness

value is better than the original. Otherwise, it is discarded. The pseudo-code of the OBLheuristic is shown in Algorithm 4.

The number of generations for $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ as for the standard GA is fixed as n. The stages completed by $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ in the lines 1, 2, 6, 7, 8 and 9 have time complexity $O(n^2 \log n)$ and in line 5 $O(n \log n)$ as for the standard GA (see [6]). Also, restarting the population (lines 12 and $\underline{12}$) has complexity $O(n^2 \log n)$. Applications of Algorithms 3 and 4 (lines $3, \underline{3}, 10$ and $\underline{10}$) over (resp. 60% and 70%) of the current population have complexity $O(n^2 \log n)$. For calculating the entropy value (line 11), a hash table was implemented indexing each different fitness value found in the current population obtaining time complexity $O(n^2)$. Thus, $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ have both time complexity in $O(n^3 \log n)$.

4 Experiments

Setting the Parameters

In order to obtain a configuration of parameters for $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ which provides the best results (number of translocations), combinations of parameters were tested. Parameters were split in 3 groups. G.1: probability of crossover and mutation; G.2: percentage of selection and replacement over population; G.3: percentage of population for which local search or OBL heuristic is applied, percentage of current population that will be preserved after restarting the population, and the threshold entropy.

Tab. 1. Avg. translocations for synthetic genomes with 2, 3, 4 and 5 chromosomes

	2 chrom.				3 chrom.			
n	1.5App	GA	$\mathcal{GA}_{\text{OBL}}$	$\mathcal{GA}_{\text{MA}}$	1.5App	GA	$\mathcal{GA}_{\text{OBL}}$	$\mathcal{GA}_{\text{MA}}$
20	11.240	10.107	10.106	10.090	10.210	9.186	9.185	9.180
30	19.130	16.886	16.861	16.792	17.710	15.602	15.586	15.552
40	26.910	23.538	23.536	23.367	25.530	22.327	22.289	22.200
50	35.540	30.620	30.602	30.358	34.080	29.650	29.627	29.424
60	44.150	37.931	37.908	37.577	42.100	36.422	36.371	36.093
70	52.220	45.250	45.180	44.745	50.040	43.410	43.312	42.944
80	59.580	51.602	51.548	51.065	58.400	50.659	50.592	50.117
90	68.250	59.098	59.072	58.493	66.220	57.331	57.341	56.769
100	76.650	66.601	66.520	65.852	74.740	64.935	64.905	64.303
110	85.860	74.905	74.846	74.165	83.230	72.372	72.337	71.648
120	93.880	81.859	81.749	81.039	91.720	79.921	79.842	79.105
130	102.010	89.192	89.065	88.292	99.240	86.971	86.880	86.102
140	109.930	96.153	96.175	95.324	108.100	94.625	94.571	93.740
150	118.560	103.940	103.801	103.092	116.420	101.792	101.769	100.943
	4 chrom.				**5 chrom.**			
n	1.5App	GA	$\mathcal{GA}_{\text{OBL}}$	$\mathcal{GA}_{\text{MA}}$	1.5App	GA	$\mathcal{GA}_{\text{OBL}}$	$\mathcal{GA}_{\text{MA}}$
20	9.240	8.461	8.460	8.460	7.830	7.320	7.320	7.320
30	16.330	14.619	14.619	14.612	15.730	14.070	14.064	14.061
40	23.870	20.925	20.896	20.862	22.240	19.792	19.784	19.753
50	31.680	27.554	27.527	27.396	29.710	26.298	26.275	26.197
60	39.370	34.156	34.131	33.913	36.930	32.275	32.249	32.112
70	47.940	41.361	41.320	41.036	44.870	39.321	39.300	39.069
80	55.350	48.024	48.011	47.614	51.600	44.912	44.900	44.619
90	62.240	54.404	54.391	53.930	59.670	52.036	52.036	51.659
100	71.110	62.033	61.956	61.407	66.790	58.399	58.343	57.911
110	78.660	68.544	68.477	67.846	74.940	65.646	65.639	65.084
120	87.330	76.177	76.101	75.420	81.820	71.828	71.747	71.143
130	95.010	83.198	83.195	82.432	90.190	79.062	79.086	78.380
140	101.970	89.313	89.328	88.515	97.280	85.325	85.273	84.515
150	111.210	97.480	97.511	96.651	104.150	91.729	91.646	90.858

Let x be a parameter of one group, a discrete interval was generated for this parameter (f. ex., for mutation probability $0.01, 0.02, \ldots, 0.09$) and the other parameters were fixed with an estimated value. Then, several tests were performed for the possible values of x. At the end of the tests the best discrete value found was taken. This process is performed for each parameter in each group. The parameters for $\mathcal{GA}_{\text{MA}}$ are the following: crossover with single point and probability 90%, mutation probability 2%, selection percentage 80%, replacement percentage 70%, percentage of local search 60%, percentage of preservation 60% and threshold entropy 0.3. For $\mathcal{GA}_{\text{OBL}}$, parameters in G.1 and G.2 resulted in the same values than $\mathcal{GA}_{\text{MA}}$, and for G.3 the parameters are the following: percentage of OBL 70%, percentage of preservation 50%, and threshold entropy 0.1.

Experiments with Hundred Synthetic Genomes

Experiments with $\mathcal{GA}_{\text{OBL}}$, $\mathcal{GA}_{\text{MA}}$, GA ([6]) and the 1.5+ε-approximation algorithm ([5] as implemeted in [6]) were conducted as follows: Initially, hundred unsigned genomes with n genes, for $n \in \{20, 30, \ldots, 150\}$, and with N chromosomes, for $N \in \{2, 3, 4, 5\}$ were ramdomly generated. The three GAs were executed ten times and the 1.5+ε-approximation algorith only once for each

genome. For the three GAs, the average of the result (number of translocations) for each genome was calculated. Finally, for each family of hundred genomes of size (n, N) the average of the results was computed. See Tab. 1.

Experiments with Single Genomes
Nine benchmark unsigned genomes proposed in [6] were used. The genomes have the following nomenclature *GenomeLxCy*, where L stands for length, C stands for number of chromosomes. For each genome, GA, $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ were executed fifty times, and then the following measures calculated: mean, median, minimum and maximum. See Tab. 2, where the best values are highligted.

Tab. 2. Results for GA, $\mathcal{GA}_{\text{MA}}$, and $\mathcal{GA}_{\text{OBL}}$ using benchmak genomes from [6]

	Mean			Median			Minimum			Maximum		
	GA	$\mathcal{GA}_{\text{MA}}$	$\mathcal{GA}_{\text{OBL}}$	GA	$\mathcal{GA}_{\text{MA}}$	$\mathcal{GA}_{\text{OBL}}$	GA	$\mathcal{GA}_{\text{MA}}$	$\mathcal{GA}_{\text{OBL}}$	GA	$\mathcal{GA}_{\text{MA}}$	$\mathcal{GA}_{\text{OBL}}$
GL150C2	102.52	**101.58**	102.48	102.50	**102.00**	**102.00**	101.00	**100.00**	**100.00**	106.00	**104.00**	106.00
GL150C3	109.54	**108.82**	109.40	**109.00**	**109.00**	109.50	108.00	**107.00**	**107.00**	**111.00**	**111.00**	112.00
GL150C4	100.94	**100.04**	101.08	101.00	**100.00**	101.00	99.00	**98.00**	99.00	104.00	**102.00**	104.00
GL150C5	96.92	**96.32**	96.82	97.00	**96.00**	97.00	**95.00**	**95.00**	**95.00**	99.00	**98.00**	99.00
GL150C6	84.8	**84.06**	84.58	85.00	**84.00**	**84.00**	**84.00**	**84.00**	**84.00**	87.00	**85.00**	86.00
GL150C7	91.08	**90.38**	91.14	91.00	**90.00**	91.00	**90.00**	**90.00**	**90.00**	93.00	**92.00**	94.00
GL150C8	77.52	**76.88**	77.46	77.50	**77.00**	78.00	**76.00**	**76.00**	**76.00**	79.00	**78.00**	**78.00**
GL150C9	78.80	**78.22**	78.74	79.00	**78.00**	79.00	**78.00**	**78.00**	**78.00**	81.00	**79.00**	81.00
GL150C10	77.52	**77.10**	77.48	**77.00**	**77.00**	**77.00**	**77.00**	**77.00**	**77.00**	81.00	**78.00**	79.00

Experiments with Genomes based on Biological Data
Three different mammals species were chosen: cat, mouse, and human. Their genomes were taken from [3], considering only the genetic material in common. Modifications in these genomes were done to fulfill the Property 1: genes 82, 83 and 88 were removed in the original mapping proposed in [3], then each genome remains with 18 chromosomes; also, auxiliary genes were added at the extremes of each chromosome in each genome, in order to obtain genomes co-tails.

Tab. 3. Results using biological data

Genome	1.5-App	GA	$\mathcal{GA}_{\text{MA}}$	$\mathcal{GA}_{\text{OBL}}$
Human-Cat	43	43	43	43
Human-Mouse	53	51	51	51
Cat-Mouse	53	50	50	50

Initially, the human genome was fixed as an identity, and the corresponding mapping of genes over the cat and mouse genomes were performed to generate the human-cat and human-mouse data. Then, the cat genome was fixed as the identity and the corresponding mapping of genes was applied over the mouse genome to generate the cat-mouse data. The GA, $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ were executed ten times for these biological data (the 1.5ε-algorithm only once), and then the average was calculated. See Tab. 3.

5 Discussion

From the experiments using families of hundred randomly generated genomes (Tab. 1), one can observe that $\mathcal{GA}_{\text{MA}}$ outperforms the results of the other algorithms. $\mathcal{GA}_{\text{OBL}}$ presents just a small improvement regarding GA.

From experiments using biological data (Tab. 3), it can be observed that $\mathcal{GA}_{\text{MA}}$, $\mathcal{GA}_{\text{OBL}}$ and GA give the same results. This is explained because these genomes are instances that are very easy to solve since cat, human and mouse, have very similar sequences of genes, when compared to randomly generated inputs. Additionally, the $1.5 + \varepsilon$-approximation algorithm computes the worst results from the four algorithms.

From experiments using specific genomes (Tab. 2), $\mathcal{GA}_{\text{MA}}$ computes the best results compared with those computed by GA and $\mathcal{GA}_{\text{OBL}}$. For the same measures $\mathcal{GA}_{\text{OBL}}$ computes slightly better results than GA, and for certain instances the results are the same.

Regarding running time, observing the time necessary to compute the most difficult instances (Tab. 2), the experiments showed that $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ take approximatelly 323% and 160% of the running time of GA, respectively. Despite

this, both $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ are of practical interest since the first algorithm takes just 1 minute to sort genomes with 150 genes and 10 chromosomes in a modest OSX Intel Core I5 processor platform.

Statistical Analysis
The samples used for the statistical analysis are the results of 50 runs of GA, $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ for benchmark unsigned genomes, which were used in the Section 4. The statistical analysis was performed using the following methodology as discussed in [7, 8, 15]. First, the *Kolgomorov-Smirnov* test was applied in order to determine whether the samples have or not a normal distribution. Then, after determining that the samples have a non-normal distribution the *Wilcoxon Rank Sum* test was applied to compare the medians of two algorithms. These statistical tests were performed in `Matlab`.

The Wilcoxon Rank Sum test tests the null hypothesis (H_0) that two samples have distributions with equal medians, this result is represented with the symbol "$s-$". Otherwise, the null hypothesis is rejected and one assumes the alternative hypothesis (H_A) that the samples have distributions with different medians, which is represented with the symbol "$s+$". For this test a significance level of 5% ($p\text{-}value <= 0.05$) or a confidence level of 95% were used.

The results of the Wilcoxon Rank Sum test to compare $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ with the other algorithms are shown respectively in Tabs. 4 and 5. In these tables two columns were included, one with the median result and the other with the result of the statistical test("$s-$"or "$s+$").

Tab. 4. Statistical Comparison between $\mathcal{GA}_{\text{MA}}$ and the other Algorithms using the Wilcoxon Rank Sum Test

	$\mathcal{GA}_{\text{MA}}$	GA		$\mathcal{GA}_{\text{OBL}}$	
GL150C2	**102.00**	102.50	$s+$	**102.00**	$s+$
GL150C3	**109.00**	**109.00**	$s+$	109.50	$s+$
GL150C4	**100.00**	101.00	$s+$	101.00	$s+$
GL150C5	**96.00**	97.00	$s+$	97.00	$s+$
GL150C6	**84.00**	85.00	$s+$	**84.00**	$s+$
GL150C7	**90.00**	91.00	$s+$	91.00	$s+$
GL150C8	**77.00**	77.50	$s+$	78.00	$s+$
GL150C9	**78.00**	79.00	$s+$	79.00	$s+$
GL150C10	**77.00**	**77.00**	$s+$	**77.00**	$s+$

Tab. 5. Statistical Comparison between $\mathcal{GA}_{\text{OBL}}$ and the other Algorithms using the Wilcoxon Rank Sum Test

	$\mathcal{GA}_{\text{OBL}}$	GA		$\mathcal{GA}_{\text{MA}}$	
GL150C2	**102.00**	102.50	$s-$	**102.00**	$s+$
GL150C3	109.50	**109.00**	$s-$	**109.00**	$s+$
GL150C4	101.00	101.00	$s-$	**100.00**	$s+$
GL150C5	97.00	97.00	$s-$	**96.00**	$s+$
GL150C6	**84.00**	85.00	$s-$	**84.00**	$s+$
GL150C7	91.00	91.00	$s-$	**90.00**	$s+$
GL150C8	78.00	77.50	$s-$	**77.00**	$s+$
GL150C9	79.00	79.00	$s-$	**78.00**	$s+$
GL150C10	**77.00**	**77.00**	$s-$	**77.00**	$s+$

It can be observed in Tab. 4 that the memetic approach $\mathcal{GA}_{\text{MA}}$ has different performance regarding GA and $\mathcal{GA}_{\text{OBL}}$. In most cases, $\mathcal{GA}_{\text{MA}}$ has better performance, that is when its medians have the lowest values. From Tab. 5, it can be observed that the OBL approach $\mathcal{GA}_{\text{OBL}}$ has different performance regarding just $\mathcal{GA}_{\text{MA}}$. In most cases, $\mathcal{GA}_{\text{OBL}}$ has worse performance, that is when its medians have not the lowest value.

6 Conclusion

Two hybrid evolutionary algorithms were proposed, $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$, for UTD. These new algorithms are based on the standard GA approach proposed in [6], in which the search space consists of signed versions of the initial unsigned genome to be sorted, exploring the property that a sorting translocation sequence for

any of these signed genomes is also a feasible solution for the initial unsigned genome. Thus, as fitness function the translocation distance of signed genomes is applied. The main feature of $\mathcal{GA}_{\text{MA}}$ and $\mathcal{GA}_{\text{OBL}}$ is the application of local search and the OBL heuristic, respectively. These heuristics are embedded for the improvement of the initial population, for restarting the population whenever the entropy limit is reached, and as a new stage after the breeding cycle. In both algorithms the converge of the population is controlled using the Shannon entropy. Experiments were performed to verify the quality of results computed by both algorithms regarding the standard GA and an $1.5+\varepsilon$-approximation algorithm ([5]). First, families of hundred unsigned genomes of different lengths were randomly generated, and also nine benchmark genomes of length 150 were considered. Experiments using this data showed that $\mathcal{GA}_{\text{MA}}$ outperforms the other algorithms, while $\mathcal{GA}_{\text{OBL}}$ showed just a small improvement regarding the standard GA. Additionally, three genomes based on biological data (cat, mouse, and human) were processed and experiments using this data showed that $\mathcal{GA}_{\text{MA}}$, $\mathcal{GA}_{\text{OBL}}$ and GA compute the same results. A statistical analysis was performed using the Wilcoxon Rank Sum test for the nine single unsigned genomes, concluding that $\mathcal{GA}_{\text{MA}}$ has better performance regarding both GA and $\mathcal{GA}_{\text{OBL}}$ and that $\mathcal{GA}_{\text{OBL}}$ did not show different performance regarding the standard GA.

As future work, is it of great interest adding new biological data to the experiments, generated from sequences taken from the GeneBank database. Also, it is of interest implementing a sharing population parallel version of $\mathcal{GA}_{\text{MA}}$ in order to improve the quality of the solutions reducing the running time. Despite its practical limitations, since the $1.408+\varepsilon$ approximation algorithm, proposed in [10], has a better approximation ratio than the $1.5+\varepsilon$ one, it is of great interest implementing this algorithm and comparing its results with the ones obtained by the $\mathcal{GA}_{\text{MA}}$.

References

1. F S Al-Qunaieer, H R Tizhoosh, and S Rahnamayan. Opposition based computing - a survey. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pages 1–7. IEEE, 2010.
2. A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *J. of Comp. Biology*, 13(2):567–578, 2006.
3. G. Bourque and P A Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome research*, 12(1):26–36, 2002.
4. Y. Cui, L. Wang, and D. Zhu. A 1.75-approximation algorithm for unsigned translocation distance. *J. of Comp. and Sys. Sciences*, 73(7):1045–1059, 2007.
5. Y. Cui, L. Wang, D. Zhu, and X. Liu. A $(1.5+\varepsilon)$-approximation algorithm for unsigned translocation distance. *IEEE/ACM T. on Comp. Biology and Bioinformatics*, 5(1):56–66, 2008.
6. L A da Silveira, J L Soncco-Álvarez, T A de Lima, and M Ayala-Rincón. Computing translocation distance by a genetic algorithm. TR Universidade de Brasília, submitted 2015. Available: `www.mat.unb.br/ayala/publications.html`.
7. J Demšar. Statistical comparisons of classifiers over multiple data sets. *The J. of Machine Learning Research*, 7:1–30, 2006.

8. J J Durillo, J García-Nieto, A J Nebro, C A Coello Coello, F Luna, and E Alba. Multi-objective particle swarm optimizers: An experimental comparison. In *Evolutionary Multi-Criterion Optimization*, pages 495–509. Springer, 2009.
9. S Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete App. Math.*, 71(1):137–151, 1996.
10. 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*, pages 128–140. Springer, 2014.
11. N Krasnogor and J Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *Evolutionary Computation, IEEE Transactions on*, 9(5):474–488, 2005.
12. M Mahootchi, H R. Tizhoosh, and K Ponnambalam. Oppositional extension of reinforcement learning techniques. *Inf. Sci.*, 275:101–114, 2014.
13. P Moscato and C Cotta. An introduction to memetic algorithms. *Inteligencia artificial, Revista iberoamericana de inteligencia artificial*, 19:131–148, 2003.
14. P Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.
15. D M Muñoz, C H Llanos, L S Coelho, and M Ayala-Rincón. Opposition-based shuffled pso with passive congregation applied to fm matching synthesis. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 2775–2781. IEEE, 2011.
16. C E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.
17. H R Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *Int. Conf. on Computational Intelligence for Modelling, Control and Automation, 2005 and Intelligent Agents, Web Technologies and Internet Commerce*, volume 1, pages 695–701. IEEE, 2005.
18. H R Tizhoosh, M Ventresca, and S Rahnamayan. Opposition-based computing. In *Oppositional Concepts in Computational Intelligence*, pages 11–28. Springer, 2008.
19. L Wang, D Zhu, X Liu, and S Ma. An $O(n^2)$ algorithm for signed translocation. *J. of Comp. and Sys. Sciences*, 70(3):284–299, 2005.
20. D Zhu and L Wang. On the complexity of unsigned translocation distance. *Theor. Comput. Sci.*, 352(1):322–328, 2006.