Computing Translocation Distance by a Genetic Algorithm

Lucas A. da Silveira[†], José L. Soncco-Álvarez[†], Thaynara A. de Lima[‡] and Mauricio Ayala-Rincón[†*]
Departments of [†]Computer Science and [*]Mathematics
Universidade de Brasília
70900–010 Brasília D.F., Brazil
[‡]Department of Mathematics
Universidade Federal de Goiás — Campus II
74690–900 Goiânia, Brazil

***Abstract*—Translocation is a useful operation on strings with challenging questions in combinatorics of permutations and interesting applications in analysis of sequences. A translocation operation essentially is the interchange of prefixes and suffixes among two substrings of a string. For the case of genomes represented as strings, symbols that represent genes and chromosomes are modeled as substrings of the genomes; thus, translocation is an operation that models the interaction between chromosomes inside a genome. The translocation distance between two genomes is defined as the minimum number of translocations to convert one genome into another and has been proved to be a meaningful manner of modeling the evolutive distance between organisms. The particular case of unsigned genomes, those in which the orientation of the genes are not considered, is particularly difficult, while the signed case, in which the orientation of genes is considered, has been proved to be polynomially decidable. This paper presents an innovative Genetic Algorithm (GA) approach to solve the unsigned translocation distance problem. A distinguishing feature of the proposed GA is that it uses as fitness function the translocation distance for randomly generated signed versions of the input (that is an unsigned genome). Experiments over randomly generated strings (synthetic genomes) showed that the proposed GA approach computes answers that are better than those computed by an 1.5+ε-approximation algorithm, the latter also implemented as part of this work.**

Keywords*—Unsigned genomes, genetic algorithm, translocation distance, approximation algorithms, bioinformatics.

I. Introduction

The comparison of biological sequences is of great relevance in the field of bioinformatics in order to determine the evolutionary relationships between organism through the reconstruction of the sequence of evolutionary events that transform one genome into another. These rearrangement mechanisms include operations such as: reversals, transpositions, and translocations. The reversal and transposition operations are generally applied to genomes of only one chromosome ([1], [2], [3]), however translocations are operations that are applied over multiple chromosomes ([4], [5]).

A. Related work

The unsigned translocation distance problem (UTD, for short) was proved $\mathcal{NP}$-hard by Zhu and Wang in [6] using previous results that relate the complexity of other problems such as the decomposition of a graph into k-cliques [7], maximum cycle decomposition of an Eulerian graph [8], and maximum decomposition of a bi-colored graph into alternating cycles [8]. As part of its background, this work revisits all steps of this proof including a compreensive explanation of the polynomial reduction of the problem of maximum cycle decomposition of Eulerian graphs into the problem of maximum decomposition into alternating cycles.

The signed translocation distance problem results much simpler than the unsigned case. Indeed, the orientation of genes provides a strong constraint in the genomes that reduces drastically the combinatorics of the problem. The first polynomial $O(n^3)$ solution was prosed by Hannenhalli in [4] giving rise to several other polynomial algorithms such as a quadratic one proposed in [9] and a linear algorithm proposed by Bergeron et al in [5]. The latter has been implemented as part of the system `UniMoG` [10] and was reimplemented in the current work in the `C` language in order to compute the fitness function of the proposed GA.

For unsigned genomes, that are the ones treated in this work, the following approximate solutions were proposed. In [11], Kececioglu and Ravi gave a 2-approximation algorithm for computing the translocation distance between unsigned genomes; Cui et al. presented a 1.75-approximation algorithm in [12], and further improved the ratio to 1.5+ε in [13]. Currently, to the best of our knowledge, the best approximation algorithm is one of ratio 1.408+ε proposed in [14].

B. Contribution

We revisit the proof of $\mathcal{NP}$-hardness of the unsigned translocation distance problem and present a GA approach for solving this problem. The fitness function is based on the translocation distance for signed versions of the input genomes, linearly computed as in [5]. To verify the quality of the solutions computed by the GA, the 1.5+ε approximation algorithm [13] was implemented as part of this work.

In the literature, the best proposed solution for UTD is the 1.408+ε-approximation algorithm, however we chose to implement the 1.5+ε-approximation algorithm, because the former one requires calculating approximate solutions for the maximum set packing problem with set size at most 3 (which is $\mathcal{NP}$-complete [15]) and for the maximum independent set problem with maximum degree 4 (which is $\mathcal{NP}$-Complete [15]), whose computation can not be done in a straightforward manner. Moreover, for our requirements, the quality of the solutions provided by both algorithms are similar since the ratio values are very close. Thus, we implemented the latter algorithm for using it as a mechanism to control the quality of the solutions of the proposed GA approach.

The proposed GA takes as input two unsigned genomes A and B (the latter identified with the identity genome). Initially, a population is created by generating signed versions of the unsigned genome A. Then, this population is improved by applying genetic operators such as crossover and mutation. The quality of the individuals is measured by the fitness function, which is calculated using the linear time algorithm in [5] for the signed version of the translocation problem. At the end of all generations the individual (signed genome) with the best fitness value is chosen.

Several experiments were performed for calculating the translocations distance, indeed, sets of hundred genomes were randomly generated as input, with each set of genomes with 10, 20 until 150 genes. The results of these experiments showed that the proposed GA outperforms the quality of solutions computed by the 1.5+ε approximation algorithm. Regarding running time, the GA takes approximately 10 seconds for genomes of length 150, which is an admissible time for analyzing such sequences. The code was implemented in C and is available at www.mat.unb.br/∼ayala/publications.html.

C. Organization

Initially, Section II presents the necessary background to understand the UTD and Section III revisits polynomial reductions involved in the proof of $\mathcal{NP}$-hardness of the UTD. Afterwards, Section IV explains the 1.5+ε-approximation algorithm and Section V introduces the GA for solving the UTD. Finally, before concluding and presenting future work in Section VIII, Sections VI and VII respectively present the experiments and discusses quality of the results.

II. Background

Standard notation is used (e.g. [5], [4], [13]).

A. Genes, Chromosomes and Genomes

In order to represent the genes inside genomes, each gene is associated with an integer number. Signed and unsigned integers represent oriented and non oriented genes respectively. A chromosome is a sequence of genes and a genome is an ordered set of chromosomes. To simplify the model, we consider that each gene appears only once in the genome. So, a genome G with n oriented genes and N chromosomes can be seen as a set $\{(x_{11},\ldots,x_{1r_1}),\ldots,(x_{k1},\ldots,x_{kr_k}),\ldots(x_{N1},\ldots,x_{Nr_N})\}$, where $\sum_{i=1}^{N} r_i = n$, $x_{ij} \in \{\pm 1,\ldots,\pm n\}$ and $|x_{ij}| \neq |x_{lk}|$ whenever $i \neq l$ or $j \neq k$. For the unsigned version, $x_{ij} \in \{1,\ldots,n\}$.

Chromosomes do not have orientation. Thus, the chromosomes $X = (x_1, x_2, \ldots, x_k)$ and $X' = (-x_k, -x_{k-1}, \ldots, -x_1)$ are the same in the signed case; whereas X and $X'' = (x_k, x_{k-1}, \ldots, x_1)$ are equal in the unsigned case. So, for example $G = \{(+1,-3),(-4,+2,-5)\}$ is a genome with 5 genes and 2 chromosomes; furthermore, G and $G' = \{(+1,-3),(+5,-2,+4)\}$ are the same genome.

To distinguish signed from unsigned genomes the former are denoted with an arrow: $\vec{G}$.

Genomes with a sole chromosome can be seen as permutations π in the symmetric group S_n. Indeed, a permutation is a bijective function from and into the set of naturals $\{1,\ldots,n\}$. A permutation π can be represented as $(\pi_1,\ldots,\pi_n)$, where π_i abbreviates $\pi(i)$, for $1 \le i \le n$.

B. Sub-permutations

Let A and B be genomes with the same genes and $S = (x_1, x_2, \ldots, x_n)$ be a chromosome in A. A sub-permutation in the chromosome S in A to B (for short, *SP* in A to B) is a segment $[x_i, x_{i+1}, \ldots, x_{i+l}]$ occurring in S with at least 3 genes, such that exactly the naturals between $|x_i|$ and $|x_{i+l}|$ occur in the set $\{|x_{i+1}|,\ldots,|x_{i+l-1}|\}$ and there is another segment $[y_j, y_{j+1}, \ldots, y_{j+l}]$ in some chromosome T of the genome B, satisfying:

- $|x_i| = |y_j|$ and $|x_{i+l}| = |y_{j+l}|$;
- $\{|x_{i+1}|,\ldots,|x_{i+l-1}|\} = \{|y_{j+1}|,\ldots,|y_{j+l-1}|\}$;
- $[x_i, x_{i+1}, \ldots, x_{i+l}] \neq [y_j, y_{j+1}, \ldots, y_{j+l}]$.

A *MinSP* is a *SP* in A to B that does not contain any other *SP*. For instance, consider the genomes:
$A = \{(1,3,2,4,5,8,6),(7,9)\}$ and
$B = \{(1,2,3,4,5,6),(7,8,9)\}$;
$[1,3,2,4,5]$ is a *SP* and $[1,3,2,4]$ is a *MinSP*. For the signed genomes, $[+1,-3,+2,+4,+5]$ is a *SP* and $[+1,-3,+2,+4]$ is a *MinSP*:
$\vec{A} = \{(+1,-3,+2,+4,+5,+8,+6),(+7,+9)\}$ and
$\vec{B} = \{(+1,+2,+3,+4,+5,+6),(+7,+8,+9)\}$.

C. Breakpoint Graphs

Breakpoint graphs are an important data structure used in combinatorics of permutations and also useful for sorting genomes by translocations and other biological mutations.

Given a chromosome $X = (x_1, x_2, \ldots, x_n)$ of a (signed or unsigned) genome, we say that the genes x_i and x_{i+1}, for $1 \le i \le n-1$, are *adjacent*; otherwise, they are *not adjacent*. Also, genes in different chromosomes are not adjacent.

Consider two signed genomes $\vec{A}$ and $\vec{B}$ with the same genes and number of chromosomes. We can build the breakpoint graph $G_s(\vec{A},\vec{B})$ as follows (Fig. 1 illustrates this notion). For all chromosomes $X = (x_1, x_2, \ldots, x_n)$ of $\vec{A}$ and $Y = (y_1, y_2, \ldots, y_m)$ of $\vec{B}$ the following elements are included:

- Vertices: a left-right ordered pair of vertices $(l(x_i), r(x_i)) = (-x_i, +x_i)$, for each gene x_i, $1 \le i \le n$;
- Edges: there is a black edge between $r(x_i)$ and $l(x_{i+1})$, for $1 \le i < n$ and, there is a gray edge between $+y_j$ and $-y_{j+1}$, if y_j and y_{j+1} are adjacent in B, $1 \le j < m$.

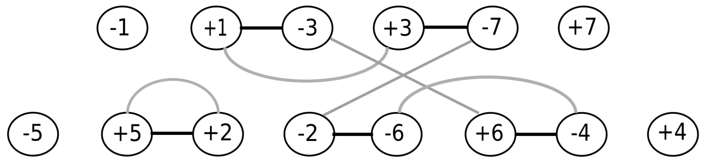

Fig. 1. $G_s(\vec{A},\vec{B})$ for $\vec{A} = \{(+1,+3,+7),(+5,-2,+6,+4)\}$ and $\vec{B} = \{(+1,-3,-6,+4),(+5,-2,+7)\}$.

Notice one gray and one black edge are incident to each vertex in $G_s(\vec{A},\vec{B})$, except for those vertices at the ends of chromosomes. Therefore, breakpoint graphs can only be decomposed into color alternating cycles and univocally (cf. [8]). A cycle is called *long*, if it contains at least two black (or gray) edges, otherwise it is *short*.

Breakpoint graphs are also defined for the unsigned case. Consider two unsigned genomes A and B with the same genes and number of chromosomes. The breakpoint graph $G_u(A,B)$ for A and B is constructed as follows: vertices are given by the genes in A and for all chromosomes $X = (x_1, x_2, \ldots, x_n)$ in A, there is a black edge between x_i and x_{i+1}, $1 \le i < n$; and, for all chromosomes $Y = (y_1, y_2, \ldots, y_m)$ of B there is a gray edge between y_j and y_{j+1}, whenever y_j and y_{j+1} are adjacent in B. Fig. 2 illustrates this notion.

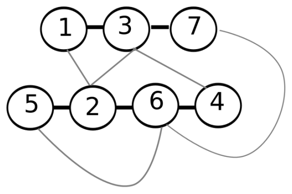

Fig. 2. $G_u(A,B)$ for $A = \{(1,3,7),(5,2,6,4)\}$ and $B = \{(1,2,3,4),(5,6,7)\}$.

The graph $G_u(A,B)$ can be partitioned into a set of alternating cycles. Note that each vertex in $G_u(A,B)$ has the same number of black and gray incident edges: vertices associated with genes at the end of chromosomes in A have only one black and one gray edge and internal genes have exactly two black and two gray edges. Thus, there is more than one way to partition $G_u(A,B)$ into alternating cycles.

Breakpoint graphs will also be defined for permutations and we will see that these are almost those graphs obtained for genomes A and B, where A and B have only one chromosome.

D. Translocation

A translocation is said to be *active* in two chromosomes X and Y when both are cut and represented as $X = (X_1, X_2)$ and $Y = (Y_1, Y_2)$ and the segments produced on both chromosomes are interchanged, transforming X and Y in two new chromosomes X' and Y'. A translocation operation works with the assumption that the four segments are not empty.

In the translocation scenario, two types of operations over segments of chromosomes are presented: *Prefix-Prefix* and *Prefix-Suffix*. Given two signed chromosomes $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ in a genome, applying the translocation by *Prefix-Prefix* $\rho(X, Y, x_i, y_k)$, one obtains two new chromosomes $X' = (x_1, \ldots, x_i, y_{k+1}, \ldots, y_m)$ and $Y' = (y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$. On the other hand, the translocation by *Prefix-Suffix* $\theta(X, Y, x_i, y_k)$ produces the new chromosomes $X' = (x_1, \ldots, x_i, -y_k, \ldots, -y_1)$ and $Y' = (-y_m, \ldots, -y_{k+1}, x_{i+1}, \ldots, x_n)$ (See Fig. 3).

Example: Consider the genome $\vec{A} = \{X, Y, Z\}$ with $X = (+1, +2, -7, +5)$, $Y = (+4, +3)$ and $Z = (+6, -8, +9)$. The translocation $\rho(X, Y, +2, +4)$ transforms $\vec{A}$ into the genome

$$\vec{A'} = \{(+1, +2, +3), (+4, -7, +5), Z\}.$$

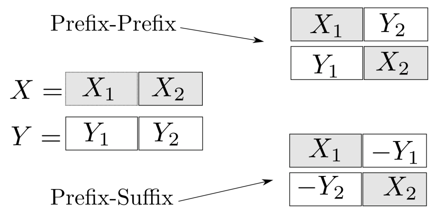

Fig. 3. *Prefix-Prefix* and *Prefix-Suffix* type translocations.

Applying the translocation $\theta(X, Z, +2, +6)$ to $\vec{A}$, one obtains the genome

$$\vec{A''} = \{(+1, +2, -6), Y, (-9, +8, -7, +5)\}.$$

For the unsigned case, translocation is defined as follows. Consider two unsigned chromosomes $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$. A translocation by *Prefix-Prefix* $\rho(X, Y, x_i, y_k)$ transforms X and Y into two new chromosomes $X' = (x_1, \ldots, x_i, y_{k+1}, \ldots, y_m)$ and $Y' = (y_1, \ldots, y_k, x_{i+1}, \ldots, x_n)$; whereas a translocation by *Prefix-Suffix* $\theta(X, Y, x_i, y_k)$ transforms X and Y into $X' = (x_1, \ldots, x_i, y_k, \ldots, y_1)$ and $Y' = (y_m, \ldots, y_{k+1}, x_{i+1}, \ldots, x_n)$.

Example: Consider the unsigned genome $A = \{X, Y, Z\}$, with chromosomes $X = (1, 2, 7, 5)$, $Y = (4, 3)$ and $Z = (6, 8, 9)$. $\rho = (X, Y, 2, 4)$ transforms A into

$$A' = \{(1, 2, 3), (4, 7, 5), Z\}.$$

$\theta(X, Z, 2, 6)$ transform A into

$$A'' = \{(1, 2, 6), Y, (9, 8, 7, 5)\}.$$

Consider a chromosome $X = (x_1, x_2, ..., x_k)$, the genes x_1 and $-x_k$ are called *tails* of X in the signed case; for the unsigned case, the *tails* of X are x_1 and x_k. Two genomes are called *co-tails* if their sets of *tails* are the same. The genomes $\vec{B}$ and $\vec{C}$ below are co-tails since they share the same set of tails, that is $\{+1, -6, +7, -10\}$.

$$\vec{B} = \{(+1, +2, -4, +3, +5, +6), (+7, -9, +8, +10)\},$$

$$\vec{C} = \{(+1, +2, +3, +4, +5, +6), (+7, +8, +9, +10)\}.$$

Notice also that genomas $\vec{A}, \vec{A'}$ and $\vec{A''}$ as well as A, A' and A'' of previous example are co-tails.

This property is essential, when we consider genome rearrangement through translocations, because translocations by *Prefix-Prefix* and *Prefix-Suffix* do not alter the set of *tails* of a genome. So, in order to transform the genome A into B by translocations the following conditions must be satisfied: the number of chromosomes and genes of A and B must be the same, and A and B must be co-tails.

In the rest of the paper, unless otherwise stated, we will consider only unsigned genomes.

E. Translocation distance

We are interested in studying the problem of sorting genomes by translocations. The problem can be described as follows: consider two unsigned genomes A and B with n genes, where the genes of the genome B are in increasing order and A and B are co-tails. Our goal is to find a sequence $\delta_1, \delta_2, \dots, \delta_t$ of translocations that transform A into B, and t is minimum; the number t is called the translocation distance between A and B. For the signed case, the problem is defined analogously with the genes of the genome B being positive and in increasing order.

The complexity of the translocation distance problem is related with the maximum decomposition into alternating cycles of breakpoint graphs. Since there is only one possible decomposition into alternating cycles of the breakpoint graph of signed genomes, the translocation distance problem results of polynomial complexity; however, for the unsigned case, the problem is $\mathcal{NP}$-hard as will be explained in the next section.

III. Unsigned Translocation Distance is NP-hard

The original proof of this fact is in [6]; here, we will compile all necessary steps providing a self-contained proof.

A. Screenplay of the Proof

For a good understanding of the stages of the proof, a screenplay will be presented containing all necessary steps that were implicitly or explicitly used in [6]. Some problems should be defined first.

k-cliques: check if the set of edges of a graph H can be partitioned into cliques of size k. For $k \geq 3$, *k-cliques* is known to be an $\mathcal{NP}$-complete problem.

MAX-ECD: consider an Eulerian graph H, the problem consists in finding a maximum decomposition in cycles of H, i.e., a partition of the set of edges of H in a maximum number of cycles.

MAX-ACD: given a breakpoint graph $G(\pi)$ of a permutation π, the problem consists in finding a maximum decomposition in alternating cycles of $G(\pi)$.

The proof is organized in the following steps (See Fig. 4):

- Initially, Section III-B demonstrates that the problem of graph partitioning in cycles for $k = 3$ is an instance of the *MAX-ECD* problem [7]; thus, *MAX-ECD* is an $\mathcal{NP}$-complete problem.
- Afterwards, Section III-C presents a polynomial reduction from *MAX-ECD* into *MAX-ACD* [8]; consequently, *MAX-ACD* is an $\mathcal{NP}$-hard problem;
- Finally, Section III-D polynomially reduces *MAX-ACD* into the translocation distance problem [6]; thus, the translocation distance problem is $\mathcal{NP}$-hard.

B. k-cliques $\subseteq$ MAX-ECD

In the early 1980's Holyer proved that partitioning a graph in k-cliques, for $k \geq 3$ is an $\mathcal{NP}$-complete problem [7]. For $k = 3$ one wants to check if the edge set of the graph can

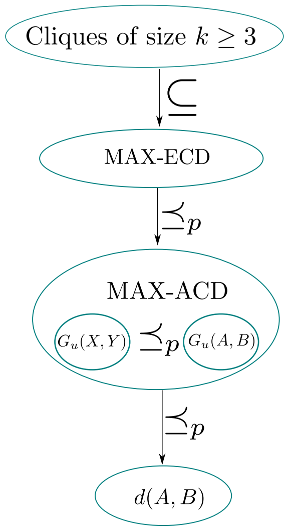

Fig. 4. Reductions for $\mathcal{NP}$-hardness of unsigned translocation distance.

be partitioned into triangles. In this case, the graph can be assumed Eulerian. Thus, the problem of finding a partition of the set of edges of an Eulerian graph into triangles is $\mathcal{NP}$-complete. Furthermore, since the decomposition of a graph into triangles gives the maximum Eulerian decomposition, one can conclude that 3-cliques $\subseteq$ *MAX-ECD*.

C. MAX-ECD $\preceq_p$ MAX-ACD

Before presenting details of the polynomial reduction from *MAX-ECD* to *MAX-ACD* proposed by Caprara in [8], some definitions and properties must be given.

1) Breakpoint Graph $G(\pi)$: Consider a permutation $\pi = (\pi_1, \dots, \pi_n)$ in S_n representing a genome constituted by only one chromosome.

The breakpoint graph $G(\pi) = \langle V, E = R \cup B \rangle$ of π is constructed as follows: initially, add to π the elements $\pi_0 = 0$ and $\pi_{n+1} = n + 1$, and consider $\pi' = (0, 1, \dots, n + 1)$. Each node $v \in V$ represents an element of π'.

The breakpoint graph $G(\pi)$ is a bi-colored graph, where the set of edges E is partitioned into two subsets: red (R) and blue (B) edges. There is a red edge (π_i, π_{i+1}) whenever $|\pi_i - \pi_{i+1}| \neq 1$, for $0 \leq i \leq n$, i.e., there is a red edge between consecutive vertices π_i and π_{i+1} that have non consecutive labels. In this case, the pair π_i, π_{i+1} is called a *breakpoint* of $G(\pi)$. There is a blue edge between vertices labelled with i and $i + 1$, $1 \leq i \leq n$, whenever i and $i + 1$ are not in consecutive positions in π. Fig. 5 illustrates this notion.

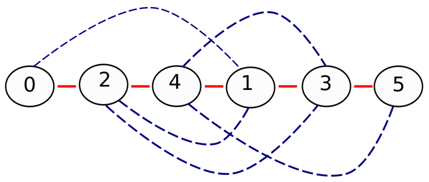

Fig. 5. Breakpoint graph $G(\pi)$ associated to genome $\pi = (2, 4, 1, 3)$, where red and blue edges are represented respectively as solid and dashed lines.

Theorem 1 (Properties of $G(\pi)$ - Th. 4 in [8]). *A bi-colored graph $G = \langle V, R \cup B \rangle$ is the breakpoint graph of some genome π iff*

- *Each connected component of the subgraphs G(R) and G(B), induced by the red and blue edges resp., is a simple path;*
- *Each node $i \in V$ has the same degree (0, 1 or 2) in G(R) and G(B);*
- *There are no edges in G(R) and G(B) connecting the same vertices.*

Sufficiency follows from definition of breakpoint graphs. Necessity requires construction of a permutation π through Hamiltonian matchings [8].

An alternating cycle of $G(\pi)$, is a sequence of edges $r_1, b_1, r_2, b_2, \ldots, r_m, b_m$, where $r_i \in R$, $b_i \in B$ for $i = 1, \ldots, m$; r_i and b_j have a common node for $i = j = 1, \ldots, m$ and for $i = j + 1$, $j = 1, \ldots, m$ (where $r_{m+1} = r_1$); and $r_i \neq r_j$, $b_i \neq b_j$ for $1 \leq i < j \leq m$.

A decomposition in alternating cycles of $G(\pi)$ is a collection of alternating disjoint edge cycles, such that each edge of $G(\pi)$ is contained in exactly one cycle of the collection. Thus, *MAX-ACD* is an optimization problem that consists in searching a maximum decomposition of $G(\pi)$ in alternating cycles. For instance see the *MAX-ACD* in Fig. 6.

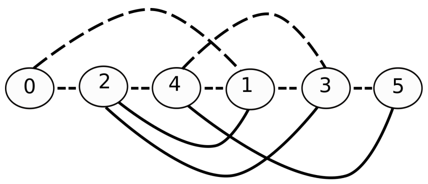

Fig. 6. *MAX-ACD* of size 2 for $G(\pi)$ for $\pi = (2, 4, 1, 3)$, where the solid lines represent one cycle, and the dashed lines represent another cycle.

2) MAX-ACD is $\mathcal{NP}$-Hard: The proof is based on a polynomial reduction from *MAX-ECD* to *MAX-ACD*. Given an Eulerian graph $H = \langle W, E \rangle$ one can built a bi-colored graph G in polynomial time, such that there exists a one-to-one correspondence between cycles of H and alternating cycles of G. The graph G is built replacing each node v of H by a bi-colored subgraph $G(v)$, containing d *bottom* nodes among other vertices, where d is the degree of v (see Fig. 7). All connections between the original vertices of H remain represented by red connections between different base nodes of the bi-colored subgraphs $G(v) \in G$. To complete the proof, the graph G must satisfy the characterization of Theorem 1.

The internal structure of each subgraph $G(v) \in G$ has d base nodes and m levels, abbreviated as $G(d, m)$. Let $s := \frac{d}{2}$ and $r := \lceil \frac{d}{4} \rceil$. Each level l, $l = 1, \ldots, m$ contains $2s+1$ nodes; $s+1$ of them are top-level nodes, denoted by $q_1^l, \ldots, q_{s+1}^l$, and the other s are lower-level nodes, denoted by $p_1^l, \ldots, p_s^l$. Top-level nodes $q_1^l, \ldots, q_{s+1}^l$ are connected to lower-level nodes $p_1^l, \ldots, p_s^l$ by d red edges $(q_i^l, p_i^l), (q_{i+1}^l, p_i^l)$, for $i = 1, \ldots, s$.

Also, for $l = 1, \ldots, m-1$ the top-level nodes $q_1^l, \ldots, q_{s+1}^l$ are connected to the lower-level nodes of the level $l+1$ by d blue edges $(p_i^{l+1}, q_i^l), (p_i^{l+1}, q_{i+1}^l)$, for $i = 1, \ldots, s$.

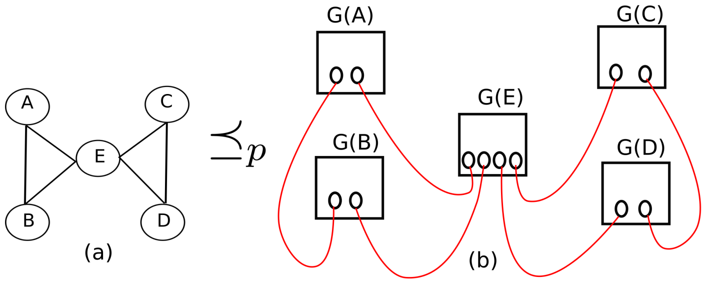

Fig. 7. (a) The Eulerian graph H. (b) The bi-colored graph $G(\pi)$ constructed from H.

Finally, the upper nodes of the last level m are connected to each other by s blue edges (q_i^m, q_{i+1}^m), for $i = 1, \ldots, s$, and the d *bottom* nodes, denoted by $b_1, \ldots, b_d$, are connected to the lower-level nodes of the first level by d blue edges $(b_{2i-1}, p_i^1), (b_{2i}, p_i^1)$, for $i = 1, \ldots, s$ (see Fig. 8).

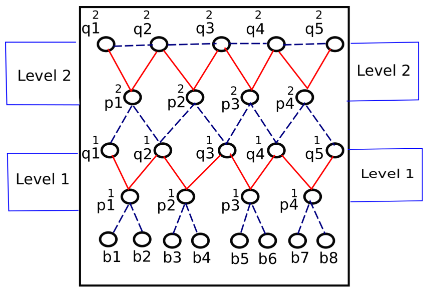

Fig. 8. Subgraph $G(d, m)$ for $d = 8$ and $m = 2$, where red edges are represented as solid lines, and blue edges are represented as dashed lines.

Observe that all nodes of $G(d, m)$ except *bottom* nodes have the same number of red and blue incident edges. Given two nodes of $G(d, m)$ an alternating path between such nodes is a path where the colors of the edges are alternate. Given any two different *bottom* nodes b_i and b_j, it is always possible to build an alternating path between b_i and b_j, whenever we have enough number of levels in $G(d, m)$. Indeed, by Theorem 5 in [8], the edge set of $G(d, m)$ can be decomposed into s alternating paths, connecting any selection of different pairs of *bottom* nodes, iff $m \geq r(s-1)+1$. Thus, one can guarantee that it is possible from a *bottom* node b_i achieve a different bottom node b_j by an alternating path and then connect b_j to another subgraph belonging to G by a red edge (see Fig. 7). w Thus, if there is a cycle in H then it can be represented by an alternating cycle in G and vice-versa; consequently there is a correspondence between a cycle decomposition of H and G. The Fig. 9 illustrates this correspondence.

To conclude, one can observe that G satisfies the conditions of the Theorem 1. Consequently, G is a breakpoint graph. The reduction is done in polynomial time choosing $m = r(s-1)+1$. Thus, we have a polynomial time reduction from *MAX-ECD* to *MAX-ACD*, and *MAX-ACD* is $\mathcal{NP}$-hard.

D. *MAX-ACD* $\preceq_p$ *UTD*

In this section, a polynomial reduction from *MAX-ACD* to *UTD* is presented following the presentation in [6]. This allows one concluding that the latter problem is $\mathcal{NP}$-hard.

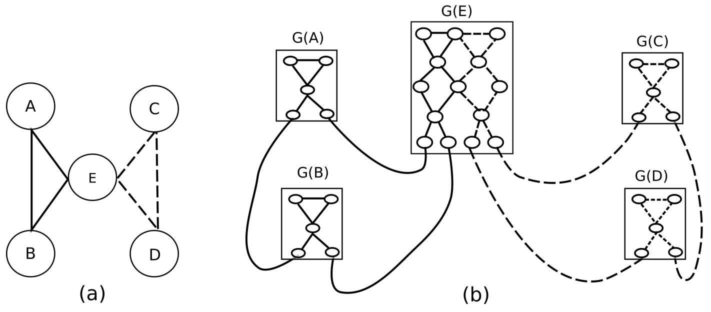

Fig. 9. (a) Eulerian graph H containing two cycles. (b) The bi-colored graph G, representing the same two cycles of H, where one cycle is represented with solid lines, and the other with dashed lines.

Let X and Y be two unsigned chromosomes. Without loss of generality, let $X = (g_1, g_2, \dots g_n)$ and $Y = (1, 2, \dots, n)$, where $\{g_1, g_2, \dots, g_n\} = \{1, 2, \dots, n\}$ and $g_1 = 1$, $g_n = n$. From X and Y, one can build two genomes $A = \{X_1, X_2\}$ and $B = \{Y_1, Y_2\}$ as will be described. Also, consider an integer d that will be used to control the amount of short cycles in the decomposition of $G_u(A, B)$; this number is justified by Lemma 2. The chromosome X_1 of the genome A is constructed by inserting $n - 1$ new genes between two adjacent genes in X as follows:

$$X_1 = (1,\ t_{1,1},\ g_2,\ t_{1,2},\ \dots,\ g_{n-1},\ t_{1,n-1},\ n)$$

where, $t_{1,k} = 3n - 2 + k, 1 \le k \le n - 1$.
X_2 contains two types of new genes, $t_{2,l} = n + l$, $1 \le l \le 2(n-1)$ and $s_i = 4n - 3 + i$, $1 \le i \le (n-2)d$.

$$\begin{aligned} X_2 \;=\; & (t_{2,1},\ t_{2,2},\ s_1,\ s_2, \dots, s_d, \\ & t_{2,3},\ t_{2,4},\ s_{d+1},\ s_{d+2}, \dots, s_{2d}, \\ & \vdots \\ & t_{2,2(n-2)-1},\ t_{2,2(n-2)},\ s_{(n-3)d+1}, \dots, s_{(n-2)d}, \\ & t_{2,2(n-1)-1},\ t_{2,2(n-1)}) \end{aligned}$$

To construct the genome $B = \{Y_1, Y_2\}$, consider the same integers $t_{1,k}, t_{2,l}$ and s_i, $1 \le k \le n-1$, $1 \le l \le 2(n-1)$, $1 \le i \le (n-2)d$, as calculated in A. The chromosome $Y_1 = Y = (1, 2, \dots, n)$ and Y_2 is built from X_2 inserting $t_{1,k}$ between $t_{2,2k-1}$ and $t_{2,2k}$ in X_2.

$$\begin{aligned} Y_2 \;=\; & (t_{2,1},\ t_{1,1},\ t_{2,2},\ s_1,\ s_2, \dots, s_d, \\ & t_{2,3},\ t_{1,2},\ t_{2,4},\ s_{d+1}, \dots, s_{2d}, \\ & \vdots \\ & t_{2,2(n-2)-1}, t_{1,n-2}, t_{2,2(n-2)}, s_{(n-3)d+1}, \dots, s_{(n-2)d}, \\ & t_{2,2(n-1)-1},\ t_{1,n-1},\ t_{2,2(n-1)}) \end{aligned}$$

At the end of the construction, each one of the genomes A and B has a total number of $4n - 3 + (n-2)d$ genes.

Example. Let $X = (1, 3, 4, 2, 5)$ and $Y = (1, 2, 3, 4, 5)$; Fig. 10 (a) illustrates the graph $G_u(X, Y)$. Consider $d = 4$. So, the genomes A and B are:

$$\begin{aligned} A = \;& \{X_1, X_2\}, \text{ where} \\ X_1 = \;& (1, 14, 3, 15, 4, 16, 2, 17, 5) \text{ and} \\ X_2 = \;& (6, 7, 18, 19, 20, 21, 8, 9, 22, 23, 24, \\ & 25, 10, 11, 26, 27, 28, 29, 12, 13) \end{aligned}$$

and

$$\begin{aligned} B = \;& \{Y_1, Y_2\}, \text{ where} \\ Y_1 = \;& (1, 2, 3, 4, 5) \text{ and} \\ Y_2 = \;& (6, 14, 7, 18, 19, 20, 21, 8, 15, 9, 22, 23, 24, \\ & 25, 10, 16, 11, 26, 27, 28, 29, 12, 17, 13) \end{aligned}$$

The graph $G_u(A, B)$ is shown in Fig. 10 (b). The Lemmas 2

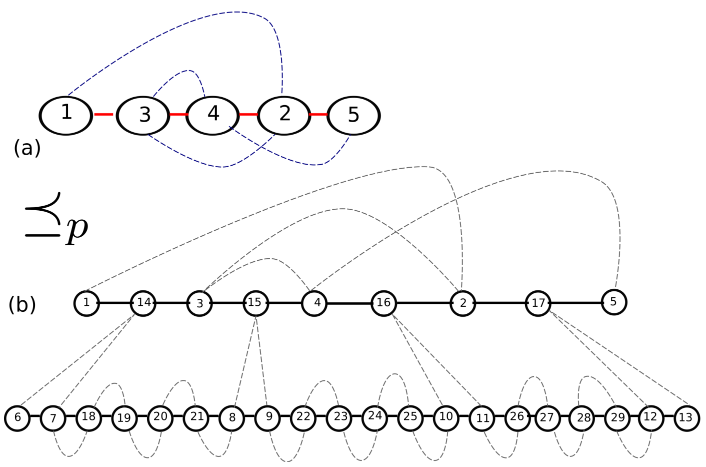

Fig. 10. Breakpoint graphs (a) $G_u(X, Y)$ is a graph with red (solid lines) and blue (dashed lines) edges. (b) $G_u(A, B)$ is a graph with black (solid lines) and gray (dashed lines) edges.

and 3 establish the relationship between a maximum decomposition into alternating cycles of $G_u(X, Y)$ and the translocation distance between genomes A and B.

Lemma 2 (Lemma 4 in [6]). *Assume $d \ge n - 1$. There is a decomposition of $G_u(X, Y)$ into J alternating cycles iff there is a decomposition of $G_u(A, B)$ into at least $(n-2)(d+1) + J$ alternating cycles.*

A brief idea of the proof of this relation given in [6] is described in the sequel.

Sufficiency: assume that there is a decomposition M of $G_u(X, Y)$ into J alternating cycles. The idea consists in associating univocally to each cycle $C \in M$ a cycle C' in $G_u(A, B)$, and with the remaining edges of $G_u(A, B)$ building $(n-2)(d+1)$ cycles.

For each cycle $C \in M$, C can be represented as a list $C = u_1, u_2, \dots, u_{2k-1}, u_{2k}$, where (u_{2i-1}, u_{2i}) is a black edge and (u_{2i}, u_{2i+1}) is a gray edge, $1 \le i \le k$ and $u_{2k+1} = u_1$. Notice that if $u_{2i-1} = g_j$ then $u_{2i} = g_{j+1}$ or $u_{2i} = g_{j-1}$.

A new cycle C' in $G_u(A, B)$ can be obtained replacing each black edge (u_{2i-i}, u_{2i}) of C by the alternating path $P_{2i-i,2_i}$, where,

$$P_{2i-i,2i} = g_j, t_{1,j}, t_{2,2j-1}, t_{2,2j}, t_{1,j}, g_{j+1} \quad \text{if } u_{2i-1} = g_j, u_{2i} = g_{j+1};$$

$$P_{2i-i,2i} = g_j, t_{1,j-1}, t_{2,2j-3}, t_{2,2j-2}, t_{1,j-1}, g_{j-1} \quad \text{if } u_{2i-1} = g_j, u_{2i} = g_{j-1}.$$

So, to each cycle $C \in M$ a long cycle C' of $G_u(A, B)$ can be associated univocally (see Fig. 11). The only edges of $G_u(A, B)$ not used to build the J long cycles are the $d + 1$ black and gray edges

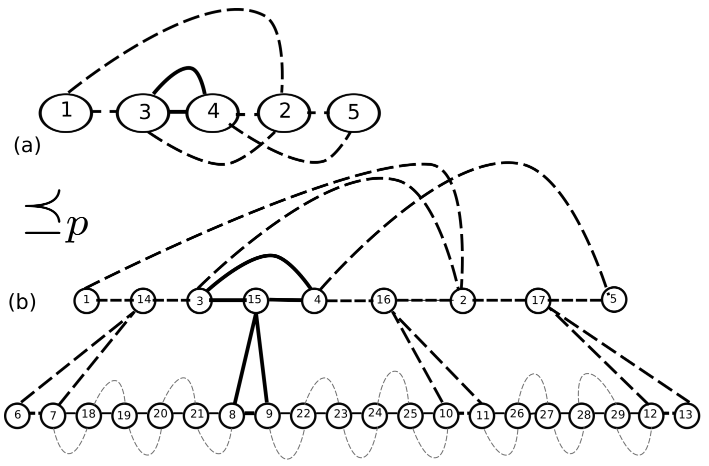

Fig. 11. Representing the same two cycles of $G_u(X,Y)$ in two long cycles into $G_u(A,B)$. The alternating cycles are distinguished by dashed and solid thick lines in $G_u(X,Y)$ and $G_u(A,B)$ respectively.

$(t_{2,2i}, s_{(i-1)d+1}), \ldots, (s_{id-1}, s_{id}), (s_{id}, t_{2,2i+1})$, whose vertices belong to X_2 and Y_2, for $i \in \{1, \ldots, n-2\}$. Such edges form $(n-2)(d+1)$ short cycles. Consequently, the decomposition M of $G_u(X,Y)$ into J alternating cycles induces a decomposition of $G_u(A,B)$ into $(n-2)(d+1)+J$ alternating cycles.

Necessity: consider M' a set of $(n-2)(d+1)+J$ alternating cycles forming a decomposition of $G_u(A,B)$. Note that, only the edges $(t_{2,2i}, s_{(i-1)d+1}), \ldots, (s_{id-1}, s_{id}), (s_{id}, t_{2,2i+1})$, for $i \in \{1, \ldots, n-2\}$, can form short cycles; consequently, there are at most $(n-2)(d+1)$ short cycles in M'. The idea consists in showing that it is always possible to build a decomposition of $G_u(A,B)$ with $(n-2)(d+1)$ short cycles; and, to the other J long cycles of $G_u(A,B)$ one can associate a decomposition of J alternating cycles of $G_u(X,Y)$. For more details see [6].

The Lemma 3 below provides an upper bound for the translocation distance between two unsigned genomes A and B, since the breakpoint graph of A and B can be decomposed into $(n-2)(d+1)+J$ alternating cycles.

Lemma 3 (Lemma 5 in [6]). *There is a decomposition of $G_u(A,B)$ into $(n-2)(d+1)+J$ alternating cycles iff $d(A,B) \leq 3n-3-J$.*

By Lemmas 2 and 3, there is a decomposition into J alternating cycles of $G_u(X,Y)$, if and only if, the translocation distance between A and B is at most $3n-3-J$.

Consider an instance of *MAX-ACD* containing n genes and $d = n-1$. So, the corresponding instance of unsigned translocation distance has $4n-3+(n-2).(n-1) = n^2+n-1$ genes. Consequently, there is a polynomial reduction from *MAX-ACD* to *Unsigned translocation distance problem* and the latter is $\mathcal{NP}$-hard.

IV. 1.5+ε-Approximate solution for UTD

In the search for approximate solutions for the translocation distance problem between unsigned genomes, Zhu and Wang [6] noted that given an unsigned genome A (and the identity genome B), the minimum number of translocations necessary to order the signed versions of A can have different values, depending on the signs attributed to their genes. See a simple example of this fact in Fig. 12. Thus, the solutions known in the literature to the unsigned case exploit this statement, applying complex heuristics, in order to acquire good approximate solutions. Here details of our implementation of the algorithm

$$\{(+1, +\underline{3, +2, +4}), (+5, -\underline{6, +7, +8})\}$$
$$\{(+1, -6, \underline{+7, +8}), (\underline{+5, +3}, +2, +4)\}$$
$$\{(+1, -\underline{6, -3, -5}), (-8, -7, +\underline{2, +4})\}$$
$$\{(+1, +2, +4), (-\underline{8, -7, -6, -3, -5})\}$$
$$\{(+1, +2, \underline{+4}), (+5, +\underline{3, +6, +7, +8})\}$$
$$\{(+1, +2, +3, \underline{+6, +7, +8}), (+5, \underline{+4})\}$$
$$\{(+1, +2, +3, +4), (+5, +6, +7, +8)\}$$

(a)

$$\{(+1, +\underline{3, +2, +4}), (+5, +\underline{6, +7, +8})\}$$
$$\{(+1, +6, \underline{+7, +8}), (\underline{+5, +3}, +2, +4)\}$$
$$\{(+1, +2, \underline{+4}), (+5, +\underline{3, +6, +7, +8})\}$$
$$\{(+1, +2, +3, \underline{+6, +7, +8}), (+5, \underline{+4})\}$$
$$\{(+1, +2, +3, +4), (+5, +6, +7, +8)\}$$

(b)

Fig. 12. The dashed lines highlight regions of inversions of a chromosome and the solid lines of translocations. (a) Genomes $A = \{(+1, +3, +2, +4), (+5, -6, +7, +8)\}$ and $B = \{(+1, +2, +3, +4), (+5, +6, +7, +8)\}$. The translocation distance is 5. (b) Genomes $A = \{(+1, +3, +2, +4), (+5, +6, +7, +8)\}$ and same B. The translocation distance is 4.

introduced in [13] are given. This algorithm provides approximate solutions of ratio 1.5+ε for the unsigned translocation distance problem. Solutions given by this implementation are used as a quality control for the solutions provided by the proposed GA (see Algorithm 1). The strategy of this approximation algorithm consists in computing the decomposition in cycles of $G_u(A,B)$, and from this decomposition attribute signs for the genes in A obtaining a signed $\vec{A}$; then, we used the algorithm proposed in [5] for computing the translocation distance of $\vec{A}$.

A. Heuristics Used in the Approximation Algorithm

In the search for a decomposition in cycles of $G_u(A,B)$, all 1-cycles are maintained, where 1-cycles are formed by a black and a gray edge, such that appropriate signs are attributed to the genes involved in order to form an 1-cycle.

After obtaining the maximum number of 1-cycles, one seeks the maximum number of 2-cycles in polynomial time. A match graph F_{AB} of the breakpoint graph $G_u(A,B)$ is constructed as follows:
1 for each black edge in $G_u(A,B)$ with at least one unsigned vertex, create a vertex in F_{AB};

2 for each two vertices in F_{AB}, an edge is created connecting them if the two black edges in $G_u(A,B)$ form a 2-cycle.

Let V and E be the set vertices and edges of F_{AB} respectively. A maximum match of F_{AB} is a set $M \subseteq E$ such that: $\forall v \in V, v$ has at most one edge incident in M.

Each edge in M represents a 2-cycle in $G_u(A,B)$. A 2-cycle in M is isolated if it does not share any edge with any other 2-cycle. Otherwise, the 2-cycle is related. Since, a 2-cycle has two gray edges, it relates at most two 2-cycles.

A *related component* U consists of related 2-cycles $c_1, c_2, \cdots, c_k$, where c_i is related with c_{i-1} $(2 \leq i \leq k)$, and each 2-cycle is not related to any other 2-cycle out of U.

Also, a related component involves at most two chromosomes and can be only of one of the four types shown in Fig. 13.

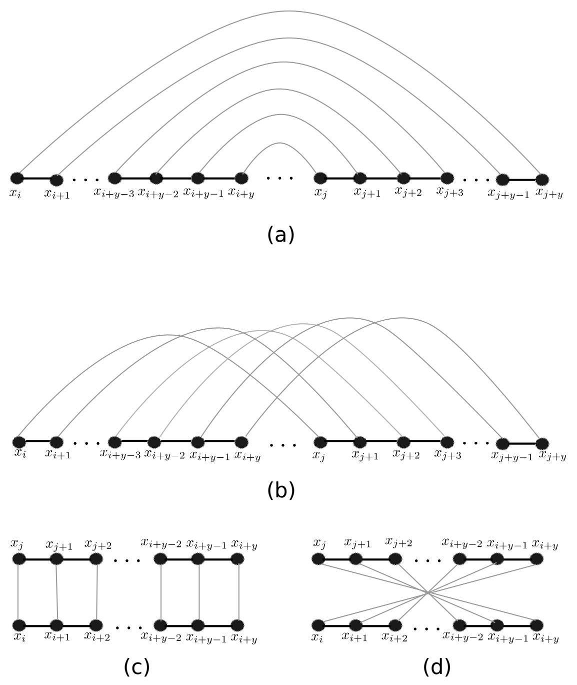
x_i x_{i+1} x_{i+y-3} x_{i+y-2} x_{i+y-1} x_{i+y} x_j x_{j+1} x_{j+2} x_{j+3} x_{j+y-1} x_{j+y}
(a)
(b)
(c)
(d)

Fig. 13. The four types of related components containing 2-cycles. (a) e (b) The component is only at one chromosome. (c) e (d) Two chromosomes are involved in the component.

After the components are built, the focus will be the isolated components. From isolated 2-cycles it is possible to identify a special type of 2-cycle, called *simple minSP* (here, *minSP* coincides with the notion defined in Section II) or in the *short form SMSP*, those 2-cycles appear with their gray edges at the extremity twisted, and with all internal cycles with their black edges involved in 1-cycles. Let $I_s = x_i, x_{i+1}, \cdots, x_j$ an *SMSP*. If there exists a gray edge (x_{i-1}, x_j) or (x_i, x_{j+1}) in $G_u(A, B)$, one can create a gray edge $(r(x_{i-1}), r(x_j))$ or $(l(x_i), l(x_{j+1}))$, transforming I_s into a *removable SMSP* (*RSMSP*). The *RSMSPs* are used to decrease the translocation distance by changing the signs of their end genes. For more details, see [13] and [4].

B. Analysis of the implementation

The Algorithm 1 is a high level abstraction of the implementation of the 1.5+ε algorithm that is also available at www.mat.unb.br/∼ayala/publications.html as part of the whole development. This algorithm was implemented using the C language. Here, we present an overview about the running time complexity, showing that the implementation has the same complexity given in [13]. Since, the Algorithm 1 produces as output a signed genome, the approximate solution of ratio $1.5 + \varepsilon$ is obtained by applying the algorithm in [5] as an auxiliary procedure.

Let n be the size of the genome A. At line 1, the breakpoint graph $G_u(A, B)$ is built using adjacency lists in time $O(n^2)$.

Both the processes of computing the 1-cycles at line 2, and building the graph F_{AB} at line 3 have time complexity $O(n)$ each one, since it is necessary to process n genes in A.

At line 4, the boost library was used for computing the maximum matching graph M of F_{AB}, this well-known library is implemented in C++ and is available at http://www.boost.org/. Computation of the matching graph has time complexity $O(V^2)$ with V representing the number of vertices in F_{AB}.

At line 5, the time complexity of finding isolated 2-cycles and 2-cycles in related components of M is $O(m^2)$, since it is necessary to compare if two cycles share the same gray edges in M, with m being the number of vertices of M.

At line 6, the procedure to identify the *RMSPs* has time complexity $O(mp)$, with p representing the number of genes of each isolated cycle.

At lines 7 and 8, the time complexity of distributing appropriate signs for both 2-cycles either isolated or related is $O(m^2)$.

At line 9, removing *RMSPs* is performed in $O(m)$, this procedure is very simple, because reverses only the extreme genes of each *RMSP*.

At line 10, getting the signs distributed for the 2-cycles in the previous steps and assigning it to the genes of A has time complexity $O(n)$.

At line 11, verifying if there exist genes without signs in A has time complexity $O(n)$.

At line 12, distributing arbitrarily signs to the genes of A has time complexity $O(n)$.

Thus, if one looks only to the procedure with highest complexity, that is the procedure that computes the graph $G_u(A, B)$ and has quadratic complexity, the implementation runs as proposed in ([12], [13]).

Algorithm 1: 1.5+ε approximate algorithm for UTD

```
Input: Unsigned genomes $A$ and $B$ (as identity genome)
Output: Signed genome $\vec{A}$
1  Build the breakpoint graph $G_u(A, B)$;
2  Compute all possible 1-cycles in $A$;
3  Build the graph $F_{AB}$;
4  Compute the maximum matching graph $M$ of $F_{AB}$;
5  Compute isolated 2-cycles and related components in $M$;
6  Build all possible RMSPs of isolated 2-cycles;
7  Distribute appropriate signs to isolated 2-cycles;
8  Distribute appropriate signs to related components;
9  Remove all RMSPs;
10 Get the signs distributed for the 2-cycles in the previous
   steps and put in genes of $A$;
11 if there exists genes without signs in $A$ then
12     Distribute arbitrarily signs to genes of $A$;
```

V. A Genetic Algorithm for UTD

Initially, necessary concepts about genetic algorithms are given. A GA is a searching technique used to solve optimization problems, that was introduced in 1975 by Holland in his book "*Adaptation in natural and artificial systems*". Such technique works with the hypothesis that the genetic information of a specific population contains a possible solution. This solution, possibly is not contained in a single individual. Thus, through techniques of genetic combination, new individuals can be obtained that improve the solution to the proposed problem, and after some generations the individuals converge to a good solution [16].

In order to model a solution based in natural evolution, GAs emulate the evolutionary process that is done in the nature. The following concepts are necessary in order to understand GA:

Individual: an individual represents a unique solution, within a scenario of possible solutions.

Population: a population is a set of individuals constituting a scenario that contains a part of the search space, such population may contain potential solutions.

Fitness: it is used to measure how good an individual is in the population.

The process or cycle of reproduction is the central part of *GAs*, since during this process new individuals are created potentially with incremental quality. The reproduction cycle consist in 4 steps:

Selection: choose two parents to perform the reproduction. The objective is to find good individuals hoping the generation of descendants with better fitness.

Crossover: apply the crossover over the selected parents producing two news descendants. In the context of our problem this operation is performed by swapping the elements from a random point to the end of the string of two parents solutions.

Mutation: after the crossover, some of the individuals are subjected to mutation. The mutation has the roles of recovering the lost genetic material and also maintaining the genetic diversity. In the context of our problem this operation is performed by simply swapping the signs of a random element of an individual.

Replacement: consists in the replacement of the individuals with the worst fitness in the old population.

A. Fitness function in the GA for UTD

The purpose of the fitness in our algorithm is to calculate the translocation distance between the signed genome $\vec{A}$ and the identity genome, which is the genome with all its elements positive and sorted in increasing order. Thus, we can rank the best signed versions of the unsigned genome A according to this fitness. The linear algorithm proposed by Bergeron et al in [5] is used to calculate the translocation distance between two signed genomes. Originally, this linear algorithm was implemented in `Java` as a part of the system `UniMoG` [10], but we reimplemented it in the `C` language.

Finding an optimal solution for a given unsigned genome A is a hard task, since, the search space for such unsigned genome consists of 2^n signed genomes, that are all possible signed versions of A. Such signed genomes can be sorted in linear time. Additionally, it is easy to note that solutions that solve any signed genome in the search space also solve the initial unsigned genome, and of course, that all these solutions will require a number of translocations greater than or equal to the translocation distance of the given unsigned genome A. This fact will be used to guide the proposed GA.

B. Description of the GA

The GA works as follows. Initially, a random population of signed genomes is generated based on the unsigned genome input. After that, for each generation the reproduction is performed as follows: Select two individuals of the population, such individuals are part of the best current solutions for which crossover and mutation operations are applied producing two new individuals. Then, the new individuals are incorporated in the current population. The GA finishes after all the generations have been completed, the number of generations depends on the size of the input genome.

The pseudo-code of our proposed GA is shown in Algorithm 2.

Algorithm 2: GA for Calculating UTD

```
Input: Unsigned genomes $A$ and $B$ (as identity
       genome)
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 Length(A) 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;
```

Let n be the size of the genome A. The initial population size is defined as $n \log n$. Each individual in the population is generated from A in linear time, randomly assigning either a positive or negative sign to each gene. This step has time complexity of $O(n^2 \log n)$.

Since, for a single individual the fitness is computed in linear time, the process of computing the fitness value for all population has time complexity $O(n^2 \log n)$.

In the Selection step, the counting sort algorithm was used for increasingly sorting the population by their fitness values. This process has time complexity $O(n + n \log n)$, with the fitness value of each individual in the interval from 1 to n, and with population size being $n \log n$. Thus, the complexity of this step is $O(n \log n)$.

In the crossover step, the best individuals classified during the selection step are chosen to be the parents. For each pair of parents, the crossover was applied by interchanging the elements at the right side of a random point from one individual to the other. Clearly this takes linear time. Thus, the running time for executing the crossover over a maximum of $n \log n$ individuals is $O(n^2 \log n)$.

In the mutation step, this operator is applied to each new individual produced by the crossover. For each element of one individual, a check is made to verify whether to apply or not the mutation over a single element, this clearly takes linear time ($O(n)$). The total time taken by the mutation applied over a maximum of $n \log n$ individuals is $O(n^2 \log n)$.

In the replacement step, the replacement of each individual takes linear time, since all its n elements must be copied. Thus, the total time taken by the replacement of a maximum of $n \log n$ individuals is $O(n^2 \log n)$.

Finally, the genetic algorithm finishes after n generations and its total time complexity is $O(n^3 \log n)$.

VI. Experiments and Results

The GA and the 1.5+ε-approximation algorithm were implemented in `C` and tested on `OS X` and `Ubuntu Linux` platforms with Intel core I5 processors. The code is available at www.mat.unb.br/∼ayala/publications.html. For the experiments were used the same platforms.

In order to validate the proposed GA, several tests were performed. The tests were done for randomly generated genomes. These genomes were created as follows: Generate an identity genome containing n genes and N chromosomes. Then, over this identity genome apply a fixed number of random reversals and translocations. The pseudocode of this procedure is shown in Algorithm 3.

Algorithm 3: Construction of synthetic genomes

```
  Input: Number of genes $n$ with $N$ chromosomes
  Output: A synthetic genome $A$
1 Generate an identity genome $A$ with $n$ genes and $N$
  chromosomes;
2 $j \leftarrow 0$;
3 while $j \leq n$ do
4     Choose randomly a chromosome $C$ of $A$;
5     Select randomly an interval in $C$;
6     Apply a reversal over this interval;
7     Choose randomly two chromosomes $C$ and $C'$ of $A$;
8     Apply a Prefix-Prefix translocation between
      segments of $C$ and $C'$;
9     $j \leftarrow j + 1$;
```

In order to obtain good quality solutions, adjustments were performed in the parameters of the genetic operators. The fine-tuning was experimentally performed providing better solutions regarding GA solutions without adjustments in these parameters. The experiment was performed as follows: The GA was executed ten times for each genome contained in a group of hundred elements, with each group containing genomes with n genes, for $n \in \{20, 50, 100, 150\}$, and with 25% of each group having N chromosomes, with $N \in \{2, 3, 4, 5\}$. For each parameter to be adjusted its value was varied over a scenario of possible good values, and estimated values were fixed for the other parameters.

At the end of the experiment the parameters that provided the best results for the GA were taken. Those parameters are the following: single crossover point with probability of 90%, mutation probability of 2%, selection applied over 80% of the current population, and replacement applied over the 70% of the worst individuals of the current population.

TABLE I. Average results of the GA and the 1.5+ε-approximation algorithm for 2 and 3 chromosomes

	2 chromosomes		**3 chromosomes**	
n	GA	1.5+ε-Approx.	GA	1.5+ε-Approx.
10	3.394	3.540	2.750	2.900
20	9.738	10.770	9.244	10.360
30	16.513	18.690	15.891	18.110
40	23.600	27.080	22.442	25.730
50	29.9810	34.580	29.662	34.170
60	37.183	42.910	36.639	42.010
70	44.907	51.840	43.230	49.930
80	51.869	59.620	50.604	58.490
90	58.213	66.960	57.715	66.620
100	66.287	76.300	65.448	75.320
110	74.534	85.940	72.940	83.710
120	80.587	92.400	80.064	91.710
130	89.164	102.020	86.838	99.590
140	96.252	110.070	94.599	108.210
150	103.510	118.380	102.106	116.350

TABLE II. Average Results of the GA and the 1.5+ε-approximation algorithm for 4 and 5 chromosomes

	4 chromosomes		**5 chromosomes**	
n	GA	1.5+ε-Approx.	GA	1.5+ε-Approx.
10	1.670	1.740	0.980	0.980
20	8.442	9.170	7.320	7.890
30	14.861	16.710	13.650	15.330
40	21.058	23.860	20.078	22.420
50	27.545	31.450	26.334	30.020
60	34.314	39.380	32.111	36.880
70	41.488	47.640	38.935	44.430
80	47.589	54.440	45.134	51.490
90	55.020	63.110	52.609	60.310
100	61.539	70.830	58.815	67.260
110	68.687	78.460	65.027	74.450
120	75.462	86.140	72.028	82.270
130	82.452	94.020	78.792	89.680
140	89.948	102.750	86.133	98.230
150	97.686	111.080	92.506	105.330

Experiments were performed with the selected parameters for calculating the translocation distances for the GA and the 1.5+ε-approximation algorithm. For this purpose, genomes were generated using the Algorithm 3 with n genes, for $n \in \{10, 20, \cdots, 150\}$, and with N chromosomes, for $N \in \{2, 3, 4, 5\}$. For hundred genomes of length (n, N), the average of the results for the 1.5+ε-approximation algorithm was calculated. The same packages of hundred genomes used in the approximation algorithm were used to calculate the average in the GA. It is worth mentioning that for each genome of length (n, N) the GA was executed ten times and then, the average of the ten obtained results was calculated as the result for each genome of length (n, N).

The results (average translocation distances) of the experiment are shown in the Tables I and II. Also, experiments for calculating the running time (in seconds) for one execution were performed for both algorithms and the results are shown in the Tables III and IV.

VII. Discussion

A few considerations are necessary before discussing the results. On the way to build synthetic genomes, instead of applying just prefix-prefix translocations between the chromosomes, reversals over the chromosomes were also applied. By including reversals it was possible to obtain more complex instances of the problem. Prefix-suffix translocations were not

TABLE III. RUNNING TIME (IN SECONDS) OF THE GA AND THE 1.5+ε-APPROXIMATION ALGORITHM FOR 2 AND 3 CHROMOSOMES

	2 chromosomes		**3 chromosomes**	
n	GA	1.5+ε-Approx.	GA	1.5+ε-Approx.
10	0.008	0.010	0.009	0.010
20	0.030	0.010	0.032	0.011
30	0.082	0.010	0.091	0.012
40	0.184	0.009	0.199	0.011
50	0.356	0.010	0.377	0.010
60	0.605	0.010	0.646	0.019
70	0.962	0.010	1.029	0.010
80	1.427	0.010	1.535	0.011
90	2.076	0.010	2.186	0.010
100	2.877	0.010	3.016	0.010
110	3.839	0.010	4.032	0.011
120	5.011	0.011	5.246	0.010
130	6.498	0.011	6.811	0.011
140	8.123	0.011	8.519	0.011
150	10.071	0.011	10.489	0.011

TABLE IV. RUNNING TIME (IN SECONDS) OF THE GA AND THE 1.5+ε-APPROXIMATION ALGORITHM FOR 4 AND 5 CHROMOSOMES

	4 chromosomes		**5 chromosomes**	
n	GA	1.5+ε-Approx.	GA	1.5+ε-Approx.
10	0.011	0.013	0.012	0.012
20	0.037	0.015	0.042	0.011
30	0.101	0.014	0.110	0.011
40	0.217	0.011	0.234	0.011
50	0.405	0.010	0.434	0.010
60	0.685	0.009	0.737	0.010
70	1.082	0.010.	1.134	0.010
80	1.626	0.010	1.687	0.010
90	2.322	0.010	2.415	0.010
100	3.170	0.010	3.322	0.010
110	4.258	0.010	4.413	0.010
120	5.479	0.011	5.756	0.010
130	7.091	0.011	7.384	0.011
140	8.899	0.011	9.273	0.011
150	10.975	0.011	11.421	0.011

considered because they are analogous to apply a reversal and a prefix-prefix translocations, which is done by Algorithm 3.

It is important to emphasize that the algorithm proposed in [5], which is used as fitness function, has already been implemented in `Java` by their authors as a contribution of Jens Stoye. This implementation was translated into `C` in order to include it in the current work. Several test were performed to validate the results of this implementation.

There is a little variation in the running time of the 1.5+ε approximation algorithm even for genomes of length 150. This is explained because the steps of the algorithm are relatively simple, since the solutions of the approximation algorithm are based in the calculation of 2-cycles and the randomly generated inputs have a few number of 2-cycles. So, the execution of this algorithm is always fast.

In Tables III and IV can be observed that when the number of genes increases, the GA running time grows "as expected" (from the complexity analysis that gave $O(n^3 \log n)$ for the computation of inputs of length n). This can be clearly observed in the Fig 14, for inputs with 2 chromosomes. But also, the running time grows when the number of chromosomes increases. Indeed, this is explained since having more chromosomes implies more possibilities to apply prefix-prefix and prefix-suffix translocations over different pairs of them. Although this, it is necessary to stress here that the size of the population is not proportional to the size of the search

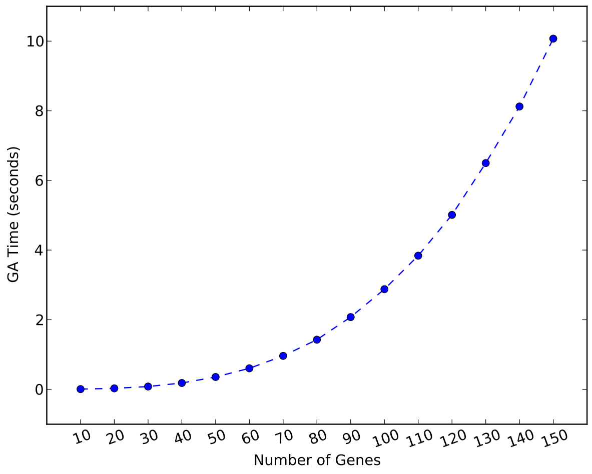

Fig. 14. Running time of the GA for different input length

space for genomes of different size as usual in combinatorics of permutations ($n \log n$ versus $n!$).

From Tables I and II, one can conclude that the GA computes better results on average than those obtained by the 1.5+ε-approximation algorithm. Also, it can be observed that for permutations of length greater than or equal to 50, the GA computes better solutions: indeed, the inputs are sorted with a number of translocations that are approximately at least 12% less than those computed by the approximation algorithm.

As can be seen in the Tables III and IV the running time of the GA is, as expected, greater when compared with the running time of the approximation algorithm and this difference is higher for larger inputs.

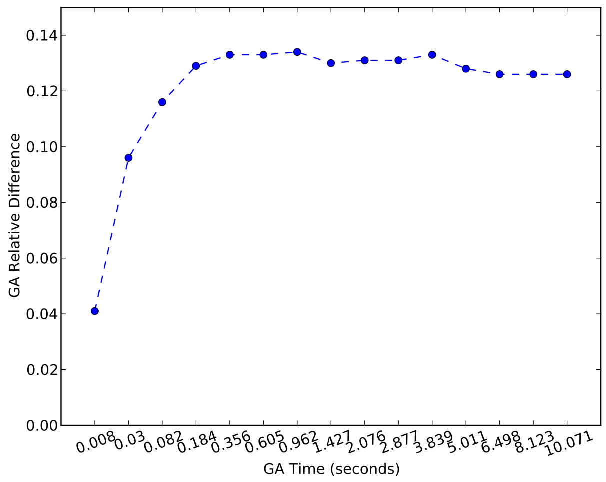

Fig. 15. Trade-off between the running time of the GA and the relative difference of the GA

Finally, the relative difference of the GA was calculated for measuring the improvement of the GA over the 1.5+ε approximation algorithm. The Fig. 15 shows the trade-off between the running time and the relative difference of the

GA for inputs with 2 chromosomes (the data was taken from Table I and II), one has similar results for inputs with 3, 4, and 5 chromosomes. In this figure it can be observed that when the time increases also the relative difference increases until it remains in the interval from 0.12 to 0.14. These values can be interpreted in the following way: as the GA running time increases the results of the GA are 12% to 14% better than those computed by the 1.5+ε approximation algorithm.

VIII. Conclusions and Future work

In the search for good solutions for the $\mathcal{NP}$-hard problem of translocation distance for unsigned genomes, a GA was proposed in this paper. This GA acts on a population of signed genomes generated from a given unsigned genome, and after each generation the population evolves to the signed genomes with the best translocation distance. Indeed, the distinguishing feature of our GA is that it uses as (linearly computable) fitness function the translocation distance of signed genomes.

Experiments showed that results obtained by the GA outperform those obtained by the 1.5+ε-approximation algorithm. Regarding running time, the 1.5+ε-approximation algorithm, as expected, is faster than the GA. However this difference is tolerable, since, the experiments with the GA have running time of approximately 10 seconds for genomes with 150 genes.

As an immediate further step we will perform experiments with data generated from the *GeneBank* sequence database. This data would be generated by assigning an integer number to each gene of a real genome; these integer numbers are mapped from an identity genome with the same genes. Also, as future work we are planning to improve our GA by including other useful heuristics as done for other GA approaches that deal with reversal distance. This will include memetic GA approaches as in [17] and parallel GA approaches as in [18] with the aim of improving the quality of the solutions while the running time is reduced. Also, it would be of great interest performing experiments with opposition based learning with the aim of exploring the search space using opposite solutions [19]. Work in progress proposes improvements which include memetic and opposition-based approaches [20].

Acknowledgment

The authors would like to thank the Brazilian agencies FAPDF and CNPq for funding this work. The first two authors are funded by CAPES M.Sc. and Ph.D. scholarships, respectively; the last author, is partially funded by a CNPq research grant.

References

[1] V. Bafna and P. A. Pevzner, "Sorting by transpositions," *SIAM J. of Disc. Math.*, vol. 11, no. 2, pp. 224–240, 1998.

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

[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," *J. of Comp. Biology*, vol. 8, no. 5, pp. 483–491, 2001.

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

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

[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] I. Holyer, "The NP-completeness of some edge-partition problems," *SIAM J. of Comp.*, vol. 10, no. 4, pp. 713–717, 1981.

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

[9] 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.

[10] R. Hilker, C. Sickinger, C. N. Pedersen, and J. Stoye, "UniMoG—a unifying framework for genomic distance calculation and sorting based on DCJ," *Bioinformatics*, vol. 28, no. 19, pp. 2509–2511, 2012.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer, 2007. [Online]. Available: http://books.google.com.br/books?id=wonrLjj2GagC

[17] J. L. Soncco-Álvarez and M. Ayala-Rincón, "Memetic algorithm for sorting unsigned permutations by reversals," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 2770–2777.

[18] J. L. Soncco-Álvarez, G. M. Almeida, J. Becker, and M. Ayala-Rincón, "Parallelization of genetic algorithms for sorting permutations by reversals over biological data," *Int. J. of Hybrid Intelligent Systems*, vol. 12, no. 1, pp. 53–64, 2015.

[19] Q. Xu, L. Wang, N. Wang, X. Hei, and L. Zhao, "A review of opposition-based learning from 2005 to 2012," *Engineering Applications of Artificial Intelligence*, vol. 29, pp. 1–12, Mar. 2014. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0952197613002388

[20] 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," Universidade de Brasília, Tech. Rep., submitted 2015, available: `www.mat.unb.br/ayala/publications.html`.