Heuristics for Reversal Distance Between Genomes with Duplicated Genes

Gabriel Siqueira[1], Klairton Lima Brito[1](✉), Ulisses Dias[2], and Zanoni Dias[1]

[1] Institute of Computing, University of Campinas, Campinas, Brazil
gabriel.siqueira@students.ic.unicamp.br,
{klairton,zanoni}@ic.unicamp.br

[2] School of Technology, University of Campinas, Limeira, Brazil
ulisses@ft.unicamp.br

Abstract. In comparative genomics, one goal is to find similarities between genomes of different organisms. Comparisons using genome features like genes, gene order, and regulatory sequences are carried out with this purpose in mind.

Genome rearrangements are mutational events that affect large extensions of the genome. They are responsible for creating extant species with conserved genes in different positions across genomes.

Close species—from an evolutionary point of view—tend to have the same set of genes or share most of them. When we consider gene order to compare two genomes, it is possible to use a parsimony criterion to estimate how close the species are. We are interested in the shortest sequence of genome rearrangements capable of transforming one genome into the other, which is named *rearrangement distance*.

Reversal is one of the most studied genome rearrangements events. This event acts in a segment of the genome, inverting the position and possibly the orientation of genes in it.

When the genome has no gene repetition, a common approach is to map it as a permutation such that each element represents a conserved block.

When genomes have replicated genes, this mapping is usually performed using strings. The number of replicas depends on the organisms being compared, but in many scenarios, it tends to be small. In this work, we study the reversal distance between genomes with duplicated genes considering that the orientation of genes is unknown. We present three heuristics that use techniques like genetic algorithms and local search. We conduct experiments using a database of simulated genomes and compared our results with other algorithms from the literature.

Keywords: Genome rearrangement · Reversal · Heuristics · Duplicated genes

C. Martín-Vide et al. (Eds.): AlCoB 2020, LNBI 12099, pp. 29–40, 2020.
https://doi.org/10.1007/978-3-030-42266-0_3

1 Introduction

The question that naturally arises when comparing the genomes of two organisms is how to estimate the sequence of mutational events that have occurred during evolution to transform one genome into another, or at least estimate the evolutionary distance between these genomes.

A way of estimating the evolutionary distance is to use a parsimony criterion and to compute a minimum sequence of events that transforms one genome into another. When large-scale mutational events are considered, the so-called genome rearrangements, this distance is called *rearrangement distance*.

A genome can be represented in different ways [6]. When the genome is treated as an ordered sequence of genes, it is possible to find scenarios where certain genes have multiple copies. In this case, it is common to adopt a representation in the form of a string, such that each character is associated with a specific gene. If each gene occurs only once, we can associate an integer number for each one and the representation is given in the form of a permutation. In both cases (string or permutation), if the orientation of the genes is known, a positive or negative sign is assigned to each element and the representation is called signed (signed string and signed permutation). Otherwise, the sign is omitted and the representation is called unsigned (unsigned string and unsigned permutation).

Reversal is a rearrangement event that breaks a chromosome at two locations and reassembles the middle piece in the reversed order. Several rearrangement problems considering only reversals were investigated over time [1–3,8,11]. When representing genomes as permutations, the goal is to determine the minimum number of reversals needed to sort any permutation.

In this case, we have the Sorting Signed Permutations by Reversals and Sorting Unsigned Permutations by Reversals problems. The former is solvable in polynomial time [8], whereas the latter is NP-hard [3]. The best algorithm for the latter has an approximation factor of 1.375 [2].

In genomes with replicated genes, when we have the same number of replicas of each gene in both genomes, the goal is to determine the minimum number of reversals needed to transform one genome into another.

The Reversal Distance for Unsigned Strings problem is NP-hard even if we consider a binary alphabet [5]. A binary alphabet means that all genes are replicas of just two types, so the genome can be mapped using only two values (e.g. $\{0,1\}$).

Unlike Sorting Signed Permutations by Reversals problem which has an exact polynomial-time algorithm, the Reversal Distance for Signed Strings problem is NP-hard [15]. Chen *et al.* [4] proved that Reversal Distance for Signed Strings is NP-hard even if we consider the simplest case where at most two replicas are allowed for each gene (duplicated genes). The authors also showed that the Reversal Distance for Signed Strings and the Minimum Common String Partition (MCSP) problems are related. The Reversal Distance for Unsigned Strings and Reverse MCSP [11] problems are also related. Based on this information, an approximation algorithm for the Reversal Distance in Signed and Unsigned

Strings problems were presented with a factor of $\Theta(k)$, where k represents the maximum number of copies of a character in the strings given as input to the algorithms [12].

In this paper, we investigate the Reversal Distance for Unsigned Strings problem considering duplicated genes. We propose three heuristics based on different techniques, and to verify the behavior of our heuristics we have created a database that simulates different scenarios. Our results were compared with others from the literature [4].

This manuscript is organized as follows. Section 2 provides definitions that are used throughout the paper. Section 3 describes the heuristics. Section 4 shows the experimental results, and Sect. 5 concludes the paper.

2 Basic Definitions

A genome $\mathcal{G}$ is represented by a string S, where each character in S corresponds to a gene or block of genes in $\mathcal{G}$. An alphabet Σ_S is the set of distinct characters of S. We denote by S_i the i-th character in S, and by $|S|$ the number of characters.

Example 1. A string S and some information we retrieve from it.

$$S = (5\ \ 2\ \ 1\ \ 3\ \ 4\ \ 5\ \ 4), \quad \Sigma_S = \{1, 2, 3, 4, 5\},\ \ |S| = 7,\ \ S_3 = 1\ ,\ \ S_5 = 4.$$

Definition 2. *The occurrence of a character α in a given string S, denoted by $occ(\alpha, S)$, represents the number of copies of α in S. The greatest occurrence of a character in S is denoted by $occ(S) = \max_{\alpha \in \Sigma_S}(occ(\alpha, S))$.*

Definition 3. *We denote by $dup(S)$ the set of duplicated characters of S, i.e. the characters that appear exactly twice in S. Therefore, $dup(S) = \{\alpha : occ(\alpha, S) = 2, \forall \alpha \in S\}$.*

In Example 1, we have $occ(S) = 2$, and $dup(S) = \{4, 5\}$.

Definition 4. *A pair of strings S and P are balanced if they have the same alphabet ($\Sigma_S = \Sigma_P = \Sigma$) and the occurrence of each character is the same for both strings. Therefore, $occ(\alpha, S) = occ(\alpha, P)$, $\forall\ \alpha \in \Sigma$.*

Example 5. Consider three strings S, P, and Q. Observe that S and P are balanced while S and Q are not, since the occurrences of character 1 in the strings S and Q are different ($occ(1, S) \neq occ(1, Q)$).

$$\begin{aligned} S &= (5\ \ 2\ \ 1\ \ 3\ \ 4\ \ 5\ \ 4) \\ P &= (4\ \ 4\ \ 1\ \ 2\ \ 5\ \ 5\ \ 3) \\ Q &= (5\ \ 1\ \ 1\ \ 3\ \ 4\ \ 5\ \ 4\ \ 2) \\ \Sigma_S &= \Sigma_P = \Sigma_Q \end{aligned}$$

We represent genome rearrangement events as operations applied to strings. This way, a rearrangement event ρ applied to a string S is denoted as $S \circ \rho$.

Definition 6. *A reversal $\rho(i,j)$, with $1 \leq i < j \leq |S|$, is an operation that inverts the order of elements in a segment of the string S.*

$$\begin{aligned} S &= (S_1 \dots S_{i-1}\ \underline{S_i \dots S_j}\ S_{j+1} \dots S_{|S|}) \\ S \circ \rho(i,j) &= (S_1 \dots S_{i-1}\ \underline{S_j \dots S_i}\ S_{j+1} \dots S_{|S|}) \end{aligned}$$

Example 7. A reversal $\rho(2,4)$ applied on a string S.

$$\begin{aligned} S &= (1\ \underline{2\ 3\ 3}\ 2\ 1\ 4) \\ S \circ \rho(2,4) &= (1\ \underline{3\ 3\ 2}\ 2\ 1\ 4) \end{aligned}$$

Definition 8. *Given two strings S and P, the reversal distance between S and P, denoted by $d(S,P)$, is the size of a shortest sequence of reversals capable of transforming S into P.*

From now on, we refer to the reversal distance only by distance. This work deals with balanced strings S such that $occ(S) \leq 2$. Every genome with multiple copies of a gene can be mapped into a string. We step forward and map the string into a permutation. To do that, we keep the characters without duplicates untouched and map each replica of duplicated characters into new values.

Assuming two replicas of a character α, there are two possible mappings of α into new values α' and α''. The mapping of all duplicated characters in S is represented by a vector $\mathbf{m}$ with size $|dup(S)|$. In $\mathbf{m}$ we place the value 0 or 1 to indicate for each duplicated character which of the two possible maps will be used. If the value associated with the character α in $\mathbf{m}$ is 0, the first and the second occurrences will be replaced by α' and α'', respectively. Otherwise, the first and second occurrences will be mapped as α'' and α', respectively.

We denote by $\mathbf{m}_\alpha$ the chosen map of the duplicate character α in $\mathbf{m}$ and by $S^{\mathbf{m}}$ the permutation generated by mapping S according to $\mathbf{m}$.

Example 9. A map $\mathbf{m}$ being applied to a string S and the permutation $S^{\mathbf{m}}$ obtained.

$$\begin{aligned} S &= (5\ \ 2\ \ 1\ \ 3\ \ 4\ \ 5\ \ 4), \quad dup(S) = \{4,5\} \\ S^{\mathbf{m}} &= (5''\ 2\ \ 1\ \ 3\ \ 4'\ 5'\ 4'') \end{aligned}$$

$$\mathbf{m} = \begin{array}{cc} 4 & 5 \\ \hline \multicolumn{1}{|c|}{0} & \multicolumn{1}{c|}{1} \\ \hline \end{array}, \quad \mathbf{m}_4 = 0, \quad \mathbf{m}_5 = 1$$

Definition 10. *Given a string S and two maps $\mathbf{m}$ and $\mathbf{v}$. We say that $\mathbf{m}$ and $\mathbf{v}$ are neighbors if they differ by exactly one duplicated character map. In other words, $\exists \alpha \in dup(S) : \mathbf{m}_\alpha \neq \mathbf{v}_\alpha$ and $\mathbf{m}_\beta = \mathbf{v}_\beta, \forall_{\beta \neq \alpha}$.*

Example 11. We obtain $S^{\mathbf{m}}$, $S^{\mathbf{v}}$, and $S^{\mathbf{z}}$ from S using maps $\mathbf{m}$, $\mathbf{v}$, and $\mathbf{z}$, respectively. Note that $(\mathbf{m}, \mathbf{v})$ and $(\mathbf{v}, \mathbf{z})$ are neighbors, but $(\mathbf{m}, \mathbf{z})$ are not.

$$\begin{aligned}
S &= (1\ \ 2\ \ 1\ \ 3\ \ 3\ \ 2), \quad dup(S) = \{1, 2, 3\} \\
S^{\mathbf{m}} &= (1'\ \ 2'\ \ 1''\ \ 3''\ \ 3'\ \ 2'') \\
S^{\mathbf{v}} &= (1'\ \ 2'\ \ 1''\ \ 3'\ \ 3''\ \ 2'') \\
S^{\mathbf{z}} &= (1''\ \ 2'\ \ 1'\ \ 3'\ \ 3''\ \ 2'')
\end{aligned}$$

$$\mathbf{m} = \begin{array}{|c|c|c|} 1 & 2 & 3 \\ \hline 0 & 0 & 1 \\ \hline \end{array} \quad \mathbf{v} = \begin{array}{|c|c|c|} 1 & 2 & 3 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \quad \mathbf{z} = \begin{array}{|c|c|c|} 1 & 2 & 3 \\ \hline 1 & 0 & 0 \\ \hline \end{array}$$

3 Heuristic Approaches

Although the task that models genomes as permutations does not allow duplicated genes, we can perform a mapping of the strings into permutations by assigning new values to replicas. We base our heuristics on the fact that if we obtain two permutations $S^{\mathbf{m}}$ and $P^{\mathbf{p}}$ from two strings S and P using the maps $\mathbf{m}$ and $\mathbf{p}$, respectively, then the sequence of reversals that turns $S^{\mathbf{m}}$ into $P^{\mathbf{p}}$ also transforms S into P. Therefore, $d(S, P) \leq d(S^m, P^p)$.

Note that, there exist maps such that $d(S, P) = d(S^m, P^p)$. Such maps could be derived from a shortest sequence of reversals transforming S in P.

Example 12. A sequence of reversals transforming S^m into P^p and S into P.

$$\begin{aligned}
S^m &= (5''\ \underbrace{2\ \ 1}_{\rho(2,3)}\ 3\ \ 4'\ 5'\ 4'') \\
&\ (5''\ 1\ \ 2\ \underbrace{3\ \ 4'\ 5'\ 4''}_{\rho(4,7)}) \\
&\ (5''\ 1\ \underbrace{2\ \ 4''\ 5'}_{\rho(3,5)}\ 4'\ 3) \\
P^p &= \ (5''\ 1\ \ 5'\ 4''\ 2\ \ 4'\ 3)
\end{aligned}
\qquad
\begin{aligned}
S &= (5\ \underbrace{2\ \ 1}_{\rho(2,3)}\ 3\ \ 4\ \ 5\ \ 4) \\
&\ (5\ \ 1\ \ 2\ \underbrace{3\ \ 4\ \ 5\ \ 4}_{\rho(4,7)}) \\
&\ (5\ \ 1\ \underbrace{2\ \ 4\ \ 5}_{\rho(3,5)}\ 4\ \ 3) \\
P &= (5\ \ 1\ \ 5\ \ 4\ \ 2\ \ 4\ \ 3)
\end{aligned}$$

$$\mathbf{m} = \begin{array}{|c|c|} 4 & 5 \\ \hline 0 & 1 \\ \hline \end{array} \quad \mathbf{p} = \begin{array}{|c|c|} 4 & 5 \\ \hline 1 & 1 \\ \hline \end{array}$$

To find the distance between two strings we must find the map of the strings in the permutations with the smallest distance. We use as a distance estimator for the Sorting Unsigned Permutations by Reversals problem an approximation algorithm with factor 2 developed by Kececioglu and Sankoff [10], which we will call KS95. We chose this algorithm because its results in practice are good and its execution time is fast, which serves our purpose of creating simple heuristics that provide solutions in a fast way. However, another much more complicated algorithm with a better approximation ratio is known [2].

Our heuristics share a common goal: find a map of strings into permutations that results in a good solution. Sections 3.1, 3.2, and 3.3 present heuristics using Random Maps, Local Search, and Genetic Algorithms, respectively.

3.1 Random Maps (RM)

This heuristic randomly generates several maps of the source string into permutations, and a single random map of the target string. After that, the heuristic estimates the distance between each source permutation and the target permutation using KS95. In the end, the heuristic selects the solution with smallest distance. In case of a tie, the heuristic selects one of the best solutions randomly.

The inputs are the strings S and P, and a parameter $r \in \mathbb{N}$ that define the total number of random maps for the source string. The random maps are generated as follows: for each position of the map, the values 0 and 1 are assigned with same probability. Initially, the target string P is mapped into a permutation $P^{\mathbf{p}}$ using a random map $\mathbf{p}$. Next, r random maps of the source string S are generated and stored in a set $\mathbf{M}$. For each map $\mathbf{m} \in \mathbf{M}$, the distance between the permutations $S^{\mathbf{m}}$ and $P^{\mathbf{p}}$ is computed using KS95. The result for S and P is the shortest distance between the permutations.

Figure 1 shows a simulation of the heuristic for $S = (3\ 2\ 1\ 2\ 4\ 3\ 4)$ and $P = (1\ 3\ 4\ 2\ 2\ 4\ 3)$. The heuristic finds that $d(S,P) \leq d(S^m, P^p) \leq 4$.

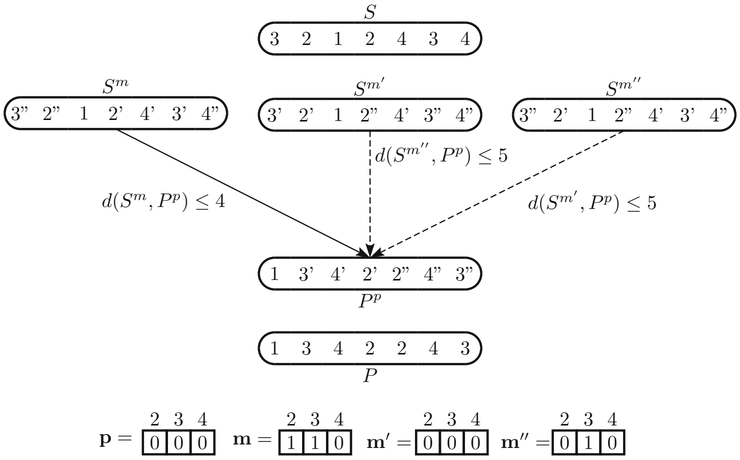

Fig. 1. Example of random maps heuristic with $r = 3$.

The random mapping process described in this heuristic will be used as subroutines by other heuristics.

3.2 Local Search (LS)

This heuristic enhances the Random Maps by generating maps through local search technique and not exclusively in a random way. The idea is to create new maps by selecting some of the current ones to have their neighborhoods explored.

Inputs are strings S and P, and the parameters $r, c, l \in \mathbb{N}$. Parameters r, c, and l determine the total number of maps that will be created, the number of maps randomly generated, and the maximum number of neighbors explored in each local search, respectively.

Initially, the heuristic behaves like the Random Maps heuristic: (i) the target string P is mapped into a permutation $P^{\mathbf{p}}$ using a random map $\mathbf{p}$, and (ii) a set $\mathbf{M}$ is generated with c random maps of the source string S into permutations. Note that $\mathbf{M}$ is composed of only c randomly generated maps, but this set must contain r distinct maps. To generate $r - c$ maps the heuristic performs a local search on the best solutions found so far as follows:

1. In each iteration, the heuristic ranks the maps using the algorithm KS95.
2. The heuristic selects the best map and explores up to l neighbor maps, adding them to $\mathbf{M}$. This process ends when the set $\mathbf{M}$ has r distinct maps.
3. The heuristic keeps a list of the maps that have already been explored. Thus, it ensures that a map is explored only once. This behavior is important to explore the neighborhood of other maps that are also good.

Note that a map created through local search in one iteration can have the neighborhood explored in future iterations.

Similarly to Random Maps heuristic, the result for the distance between the strings S and P is the shortest distance calculated between the permutations resulting from maps in $\mathbf{M}$ and the permutation $P^{\mathbf{p}}$.

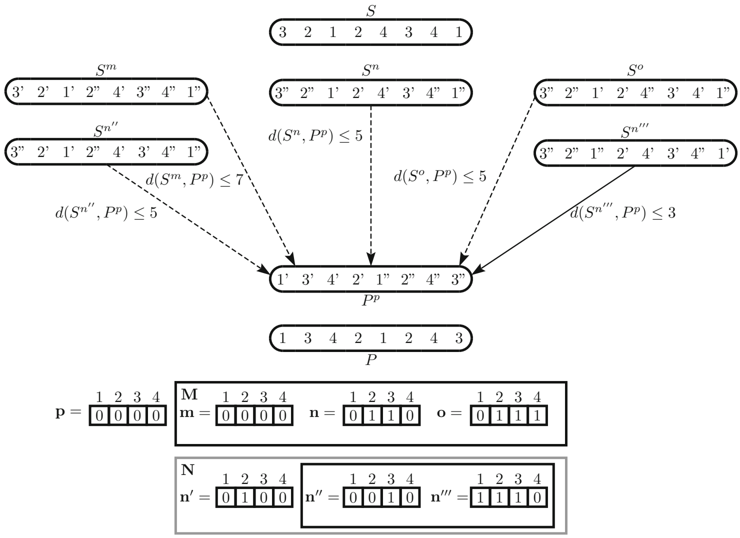

Fig. 2. Example of local search heuristic with $r = 5$, $c = 3$, and $l = 2$.

Figure 2 illustrates this heuristic in a pair of strings $S = (3\ 2\ 1\ 2\ 4\ 3\ 4\ 1)$ and $P = (1\ 3\ 4\ 2\ 1\ 2\ 4\ 3)$. The initial set $\mathbf{M} = \{\mathbf{m}, \mathbf{n}, \mathbf{o}\}$ is compose of 3 randomly generated maps. From $\mathbf{M}$, the map $\mathbf{n}$ is selected to have the neighborhood explored. Set $\mathbf{N} = \{\mathbf{n}', \mathbf{n}'', \mathbf{n}'''\}$ represents all neighboring maps of $\mathbf{n}$ that are not yet in $\mathbf{M}$. Adopting the parameter $l = 2$, the heuristic chooses the maps $\mathbf{n}''$ and $\mathbf{n}'''$ to add into $\mathbf{M}$, filling the set with $r = 5$ distinct maps. In that case, the heuristic finds that $d(S, P) \leq d(S^{n'''}, P^p) \leq 3$.

3.3 Genetic Algorithm (GA)

This heuristic is modeled using Genetic Algorithm metaheuristic [13] to generate the maps. This meta-heuristic is widely used on combinatorial optimization problems [9,14] and has already been used in problems of genome rearrangement [7].

A genetic algorithm is a search heuristic inspired by the theory of evolution. It uses features like mutations, inheritance of parents characteristics, and selection of fittest individuals for reproduction, to name a few.

The inputs are the strings S and P, and the parameters $c, k, r, t_m, t_c \in \mathbb{N}$. The parameter c is the initial population size, k is the number of individuals selected in a given population, and r is the total number of maps created for the source string. The parameters t_m and t_c are used in the mutations and crossovers, respectively. The population size, in each generation, is $\frac{5k}{2}$. Once r is reached, the algorithm stops and the best result so far is returned.

We describe our genetic algorithm considering five features: initial population, fitness function, selection, crossover, and mutation.

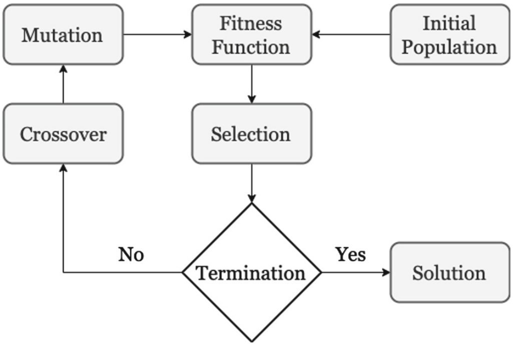

Fig. 3. Flowchart of genetic algorithm phases.

Figure 3 shows a flowchart of the interactions between each phase to obtain a solution using the genetic algorithm technique. We developed a heuristic where each individual in the population is represented by the mapping of the source string S into a permutation. We also used a single random map $\mathbf{p}$ of the target string P into a permutation $P^{\mathbf{p}}$.

- **Initial Population:** is generated c random maps of the source string S into distinct permutations. The process of generating random maps is the same as that used in the Random Maps heuristic.

- **Fitness Function:** each individual receives a score based on a fitness function. The fitness function assigns a score to an individual $\mathbf{m}$ of the population as follows: $\mathbf{m}_{score} = \frac{1}{d(S^{\mathbf{m}}, P^{\mathbf{p}})+1}$, such that $d(S^{\mathbf{m}}, P^{\mathbf{p}})$ is computed using the KS95 algorithm. Maps that result in solutions with smaller number of reversals receive higher scores and tend to transmit their characteristics to future generations.
- **Selection:** at this stage, the top k highest scored individuals are selected to the next generation, and the others are discarded. In cases of ties, the heuristic arbitrarily selects the required number of individuals among the tied ones.
- **Crossover:** at this phase, the population has the k highest scored individuals. To perform the re-population, individuals are arbitrarily paired for crossovers. Given two maps $\mathbf{m}$ and $\mathbf{m}'$ with $x = dup(S) = dup(P)$ bits, the crossover generates a new individual with t_c bits randomly selected from $\mathbf{m}$ and the remaining $x - t_c$ from $\mathbf{m}'$.

Example 13. Maps $\mathbf{m}$ and $\mathbf{m}'$ of a string $S = (3\ 3\ 1\ 4\ 2\ 2\ 4)$ adopting $t_c = 2$ and resulting in a new individual $\mathbf{n}$.

$$\begin{aligned} S &= (3\ 3\ 1\ 4\ 2\ 2\ 4), \quad t_c = 2 \\ S^m &= (3''\ 3'\ 1\ \underline{4''\ 2'\ 2''\ 4'}) \\ S^{m'} &= (\underline{\underline{3'\ 3''}}\ 1\ 4'\ 2''\ 2'\ 4'') \\ S^n &= (\underline{\underline{3'\ 3''}}\ 1\ \underline{4''\ 2'\ 2''\ 4'}) \end{aligned}$$

$$\mathbf{m} = \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} \\ \hline 0 & 1 & 1 \\ \hline \end{array} \quad \mathbf{m}' = \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} \\ \hline 1 & 0 & 0 \\ \hline \end{array} \quad \mathbf{n} = \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} \\ \hline 0 & 0 & 1 \\ \hline \end{array}$$

- **Mutation:** after crossover, each of the k individuals selected from the past generation gives rise to a new individual by inverting the values of t_m bits chosen randomly.

Example 14. A map $\mathbf{n}$ generated by a mutation in a map $\mathbf{m}$ of a string $S = (3\ 3\ 1\ 4\ 2\ 2\ 4)$ adopting $t_m = 2$.

$$\begin{aligned} S &= (3\ 3\ 1\ 4\ 2\ 2\ 4), \quad t_m = 2 \\ S^m &= (\mathbf{3''}\ \mathbf{3'}\ 1\ \mathbf{4''}\ 2'\ 2''\ \mathbf{4'}) \\ S^n &= (\mathbf{3'}\ \mathbf{3''}\ 1\ \mathbf{4'}\ 2'\ 2''\ \mathbf{4''}) \end{aligned}$$

$$\mathbf{m} = \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} \\ \hline 0 & 1 & 1 \\ \hline \end{array} \quad \mathbf{n} = \begin{array}{|c|c|c|} \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} & \multicolumn{1}{c}{4} \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

The heuristic repeats this process until r maps are generated. Afterwards, the individual $\mathbf{m}$ with the highest score is selected to obtain a solution computing the distance between $S^{\mathbf{m}}$ and $P^{\mathbf{p}}$ using the KS95 algorithm.

4 Experimental Results

We present our test methodology and the results obtained by our heuristics. In the end, we compare our results with others from the literature.

4.1 Database

Our database comprises 10 sets of 1000 pairs of strings (source and target). Each set has strings of different sizes ranging from 100 to 1000 in intervals of 100. Strings have 25% of duplicated characters, so $|dup(S)| = \frac{|S|}{4}$. Each pair of source and target strings was created as follows: we randomly distributed the values $\{1, 2, \ldots, \frac{3|S|}{4}, 1, 2, \ldots, \frac{|S|}{4}\}$ to create the source string S, and we applied a total of $\frac{|S|}{4}$ random reversals (the parameters i and j being chosen randomly) to generate the target string P.

4.2 Model Tuning

Random Maps, Local Search, and Genetic Algorithm heuristics share a common r parameter, which represents the total number of maps that are generated. This parameter should have the same number set for us to be able to compare the heuristics. In other words, heuristics must explore the same number of maps, so we may estimate the gain of each model in choosing good maps to investigate. We assigned $r = 10|S|$ for each source string S during the experiments.

Other parameters of Local Search and Genetic Algorithm heuristics were selected using a grid search. For Local Search heuristic parameters c and l, we swept through the set $\{10, 20, ..., 100\}$. For the Genetic Algorithm heuristic, k was investigated in $\{10, 20, ..., 100\}$, c in $\{10, 20, ..., 100\}$, t_m in $\{1, 2, ..., 10\}$, and t_c in $\{\lfloor 0.1d \rfloor, \lfloor 0.2d \rfloor, ..., d\}$, where $d = |dup(S)| = |dup(P)|$. The grid search was performed in the sets of strings of sizes 400, 500, and 600. We obtained the following parameters:

- Local Search: $c = 90$ and $l = 30$;
- Genetic Algorithm: $c = 90$, $k = 50$, $t_m = 2$, and $t_c = \lfloor 0.4d \rfloor$.

4.3 Results

For comparison purposes, we implemented the Kolman and Waleń algorithm [12] (called **HS**) and an adaptation of the **SOAR** algorithm [4]. This adaptation was performed to consider the Reverse MCSP [11] instead of MCSP problem [4] in order to address the unsigned version of the Reversal Distance for Strings problem.

The abbreviations **RM**, **LS**, and **GA** refer to Random Maps, Local Search, and Genetic Algorithm heuristics, respectively. Table 1 shows the average distance provided by the heuristics, **HS**, and **SOAR** using our database as input.

The **OP** column shows the number of random reversals used to create each instance. The line ($\mathbf{DEE_{avg}}$) represents the average distance estimation error. For an instance (S, P), the distance estimation error is calculated as follows: $\frac{|D_H - \mathbf{OP}|}{\mathbf{OP}}$, such that D_H is the distance estimation for the instance (S, P) computed by our heuristics and **SOAR** algorithm. The distance estimation error shows, as a percentage of the number of reversals applied to create the instance,

Table 1. Results provided by our heuristics and by **SOAR**.

String size	RM	LS	GA	SOAR	HS	OP
100	32.62	24.22	24.33	29.65	50.27	25
200	75.31	49.61	49.75	61.42	104.71	50
300	120.94	75.39	75.60	94.16	160.41	75
400	167.30	100.99	101.30	126.34	214.97	100
500	214.36	126.75	126.89	158.97	270.93	125
600	262.34	152.74	152.91	191.21	326.75	150
700	310.99	179.64	179.17	224.11	382.77	175
800	359.60	207.73	205.33	256.41	438.87	200
900	408.31	238.63	231.81	289.04	494.33	225
1000	457.22	274.03	258.54	321.88	549.91	250
$\mathrm{DEE}_{\mathrm{avg}}$	67.81%	3.87%	2.88%	26.20%	115.17%	–

how far the heuristics and **SOAR** algorithm have distanced from **OP**, either for more or less.

From the results, we can see that although the **HS** algorithm guarantees an approximation to the solution by a multiplicative factor $\Theta(k)$, in practice, the results of the other algorithms were better. This result was probably caused by the fact that the constant associated with function $\Theta(k)$ has a high value.

Genetic Algorithm (**GA**) and Local Search (**LS**) are better than **SOAR** and **RM** in all sets, which clearly indicates that the more sophisticated rules added in **GA** and **LS** succeed. On average, these strategies provide values very close to the number of reversals applied to generate the instances. This can be evidenced by observing the average distance estimation error ($\mathbf{DEE_{avg}}$) which, considering all instances, shows **GA** and **LS** heuristics with a percentage of 2.88% and 3.87%, respectively. We also can note that **GA** generates the best distance estimator (closest to **OP**) for all sets tested.

The heuristics **LS** and **GA** presented a far better performance than the previous know methods for estimate the evolutionary distance between genomes with duplicated genes, considering the reversal event.

We also performed tests with different values for the parameter r ranging from $5|S|$ up to $50|S|$, but the best trade-off between solution quality and runtime was observed adopting $r = 10|S|$.

5 Conclusion

We presented three heuristics for the Reversal Distance for Unsigned Strings with Duplicated Genes problem. We performed experiments with a database created to simulate genomes with different characteristics. We compared the results obtained by our heuristics with results from the literature. The comparison shows that our heuristic based on local search and genetic algorithms techniques tend to produce very good solutions.

As future works, we plan to extend the heuristics by considering other genome rearrangement events (e.g., transposition, insertion, and deletion), other metaheuristics, and by investigating the problem having more than two copies for each character.

Acknowledgments. This work was supported by the National Council for Scientific and Technological Development - CNPq (grants 400487/2016-0, 425340/2016-3, 304380/2018-0, and 140466/2018-5), the São Paulo Research Foundation - FAPESP (grants 2015/11937-9, 2017/12646-3, and 2017/16246-0), the Brazilian Federal Agency for the Support and Evaluation of Graduate Education - CAPES, and the CAPES/COFECUB program (grant 831/15).

References

1. Bergeron, A.: A very elementary presentation of the Hannenhalli-Pevzner theory. Discrete Appl. Math. **146**(2), 134–145 (2005)
2. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation algorithm for sorting by reversals. In: Möhring, R., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 200–210. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45749-6_21
3. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discrete Math. **12**(1), 91–110 (1999)
4. Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., Jiang, T.: Assignment of orthologous genes via genome rearrangement. IEEE/ACM Trans. Comput. Biol. Bioinform. **2**(4), 302–315 (2005)
5. Christie, D.A., Irving, R.W.: Sorting strings by reversals and by transpositions. SIAM J. Discrete Math. **14**(2), 193–206 (2001)
6. Fertin, G., Labarre, A., Rusu, I., Tannier, É., Vialette, S.: Combinatorics of genome rearrangements. Computational Molecular Biology. The MIT Press, London (2009)
7. Gao, N., Yang, N., Tang, J.: Ancestral genome inference using a genetic algorithm approach. PLOS ONE **8**(5), 1–6 (2013)
8. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. ACM **46**(1), 1–27 (1999)
9. Jog, P., Suh, J., Van Gucht, D.: Parallel genetic algorithms applied to the traveling salesman problem. SIAM J. Optimization **1**, 515–529 (1991)
10. Kececioglu, J.D., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica **13**, 180–210 (1995)
11. Kolman, P., Waleń, T.: Approximating reversal distance for strings with bounded number of duplicates. Discrete Appl. Math. **155**(3), 327–336 (2007)
12. Kolman, P., Waleń, T.: Reversal distance for strings with duplicates: linear time approximation using hitting set. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 279–289. Springer, Heidelberg (2007). https://doi.org/10.1007/11970125_22
13. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
14. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Comput. Oper. Res. **35**(10), 3202–3212 (2008)
15. Radcliffe, A.J., Scott, A.D., Wilmer, E.L.: Reversals and transpositions over finite alphabets. SIAM J. Discrete Math. **19**(1), 224–244 (2005)