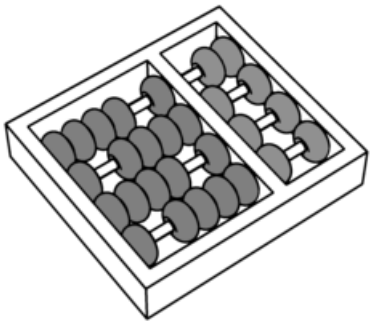
Universidade Estadual de Campinas

Instituto de Computação



Proposta de Dissertação de Mestrado

Heurísticas para Rearranjo de Genomas com Genes Duplicados

Candidato: Gabriel Henriques Siqueira

Orientador: Prof. Dr. Zanoni Dias

Resumo

Problemas de Rearranjo de Genomas buscam estimar a distância evolutiva entre genomas de diferentes organismos. Estes problemas lidam com eventos de rearranjo, que são mutações nos genomas capazes de alterar a sequência genética. Os diferentes problemas de rearranjo são definidos de acordo com os eventos considerados e a representação dos genomas utilizada. Quando os genomas apresentam repetição de genes eles são normalmente representados por strings. Neste trabalho, consideramos os eventos de rearranjo de reversão, que inverte a ordem de uma sequência de genes, e transposição, que troca a posição de duas sequências adjacentes de genes. Além disso, assumimos que os genomas podem ter genes duplicados, logo representamos os genomas por strings. Temos como objetivo o desenvolvimento de heurísticas para problemas de rearranjo de genomas com estas características.

1 Introdução

A comparação genômica é um dos estudos realizados no campo da Biologia Computacional. O objetivo desse estudo é a busca por similaridades e parentesco entre genomas de indivíduos. Uma das formas de realizar tal tarefa é estimando a distância evolutiva entre genomas. Esta estimativa pode ser feita através do número de eventos de rearranjo que afetaram o genoma de um indivíduo ao longo da evolução, transformando-o em outro. A distância de rearranjo entre dois genomas é o tamanho da menor sequência de eventos de rearranjo capaz de transformar um genoma em outro. O genoma consiste em uma sequência de genes, sendo que esses genes podem ou não apresentar repetições e suas orientações podem ser conhecidas ou desconhecidas.

Os eventos de rearranjo podem alterar a quantidade de genes (por exemplo: inserção, remoção e duplicação [34]) ou alterar sua ordem e/ou orientação (por exemplo: reversão, transposição e transreversão [3, 4, 26]). Existem também eventos que consideram múltiplos cromossomos (por exemplo: fissões, fusões e translocações [21]).

Dependendo das características do problema de rearranjo que é abordado, existem diferentes formas de se representar os genomas [16]. Quando consideramos que os genomas apresentam repetições de genes os representamos por strings, onde cada caractere corresponde a um gene ou a um bloco de genes conservado durante o processo evolutivo.

No caso particular em que é assumido que o genoma não apresenta repetição de genes, a string corresponde a uma permutação de números inteiros. Neste caso, podemos representar um dos genomas como a permutação identidade (a permutação com todos os números positivos e em ordem crescente) associando cada caractere a sua posição na string. Assim, o problema de rearranjo corresponde ao problema de ordenação de permutações, onde queremos transformar uma permutação na permutação identidade.

Um modelo de rearranjo $\mathcal{M}$ determina quais eventos de rearranjo são permitidos para transformar um genoma em outro. Estes modelos permitem o uso de um ou mais eventos de rearranjo [6, 33]. Um problema de rearranjo é definido por um modelo de rearranjo e pela forma em que representamos os genomas.

As próximas seções são organizadas da seguinte forma. A Seção 2 apresenta conceitos e definições importantes para modelarmos formalmente os problemas que serão abordados. Em seguida, a Seção 3 contém uma revisão de alguns resultados presentes na literatura. A Seção 4 apresenta os objetivos deste trabalho. A Seção 5 descreve a metodologia adotada e a Seção 6 contém o cronograma que será seguido no decorrer do trabalho. Por fim, a Seção 7 trata de alguns resultados preliminares deste projeto.

2 Fundamentação Teórica

Esta seção apresenta alguns conceitos importantes para os problemas de rearranjo de genomas.

2.1 Representação de Genomas por Strings

Em problemas de rearranjo de genomas, representamos um genoma $\mathcal{G}$ por uma string S, onde cada caractere corresponde a um bloco de genes conservados. No caso em que é conhecida a orientação dos genes, um sinal $+$ ou $-$ é associado a cada caractere, representando a orientação de cada bloco de genes dentro do genoma. Quando os genes apresentam orientação, temos uma **string com sinais**, caso contrário temos uma **string sem sinais**.

Dada uma string S, o alfabeto Σ_S é o conjunto dos caracteres que aparecem em S. Além disso, denotamos por $|S|$ o número de caracteres de S e por S_i o i-ésimo caractere de S. Chamamos S_1 e $S_{|S|}$ de extremidades da string S.

Exemplo 1. *Algumas das definições anteriores aplicadas em uma string sem sinais S e em uma strings com sinais S'.*

$$S = (5\ \ 2\ \ 1\ \ 3\ \ 4\ \ 5\ \ 5\ \ 4)$$

Extremidades de S: $S_1 = 5$ *e* $S_8 = 4$

$$S' = (+5\ \ -2\ \ -5\ \ +1\ \ +4\ \ +5\ \ +4\ \ -3)$$

Extremidades de S': $+S'_1 = +5$ *e* $-S'_8 = -3$

$\Sigma_S = \Sigma_{S'} = \{1, 2, 3, 4, 5\}$, $|S| = |S'| = 8$

Definição 1. *A ocorrência de um caractere α em uma string S, denotada por $occ(\alpha, S)$, representa o número de cópias do caractere α na string S. A maior ocorrência de um caractere em uma string S é denotada por $occ(S) = \max_{\alpha \in \Sigma_S}(occ(\alpha, S))$.*

Denotamos por $mult(S)$, o conjunto de caracteres que aparecem mais de uma vez em S e por $dup(S)$, o conjunto de caracteres que aparecem exatamente duas vezes em S ($mult(s) = \{\alpha : occ(\alpha, S) > 1, \forall \alpha \in S\}$ e $dup(s) = \{\alpha : occ(\alpha, S) = 2, \forall \alpha \in S\}$). Nas strings do Exemplo 1, temos $mult(S) = mult(S') = \{4, 5\}$ e $dup(S) = dup(S') = \{4\}$.

Definição 2. *Duas strings S e P são chamadas de balanceadas se fazem parte de um mesmo alfabeto Σ ($\Sigma = \Sigma_S = \Sigma_P$) e a ocorrência dos caracteres em ambas as strings é a mesma, ou seja, $occ(\alpha, S) = occ(\alpha, P)$, $\forall\ \alpha \in \Sigma$.*

Exemplo 2. *As strings S e P são balanceadas e as strings S e Q não são (os caracteres 1 e 5 tem ocorrências diferentes).*

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

Quando não adotamos eventos que alteram a quantidade de genes ou o conjunto de genes em um genoma, ao usar a representação por strings, precisamos que as strings sejam balanceadas.

2.2 Eventos de Rearranjo

Representamos um evento de rearranjo em um genoma $\mathcal{G}$ por uma operação na string S que o representa. Dizemos que um evento δ resulta em uma string $S \circ \delta$, que representa o genoma obtido após a aplicação do evento δ em $\mathcal{G}$.

Dado um modelo de rearranjo $\mathcal{M}$, a distância de rearranjo entre dois genomas representados pelas strings S e P, denotada por $d^{\mathcal{M}}(S, P)$, é o menor número de operações correspondentes a rearranjos de $\mathcal{M}$ necessárias para transformar S em P. Quando o modelo de rearranjo é composto apenas por reversões ($\mathcal{R}$) ou transposições ($\mathcal{T}$), usamos os termos distância de reversão ($d^{\mathcal{R}}(S, P)$) ou distância de transposição ($d^{\mathcal{T}}(S, P)$), respectivamente. Quando temos ambos os eventos de reversão e de transposição ($\mathcal{RT}$), usamos a distância de reversão e transposição ($d^{\mathcal{RT}}(S, P)$).

A seguir definimos algumas operações correspondentes aos eventos de rearranjo de reversão e transposição.

Definição 3. *Uma reversão $\rho(i, j)$, onde $1 \leq i \leq j \leq |S|$, é um evento que inverte a ordem dos elementos de um segmento do genoma. Quando a orientação dos genes do genoma é conhecida, os elementos do segmento afetado também têm a orientação invertida.*

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

Exemplo 3. *Uma reversão $\rho(2, 4)$ sendo aplicada em uma string com e sem sinais, respectivamente.*

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

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

Definição 4. *Uma transposição $\tau(i,j,j)$, onde $1 \leq i < j < k \leq |S|+1$, é um evento que troca a posição de dois segmentos consecutivos do genoma, mas sem alterar a ordem e a orientação dos genes nos segmentos.*

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

Exemplo 4. *Uma transposição $\tau(2,5,7)$ sendo aplicada em uma string com e sem sinais, respectivamente.*

$$\begin{aligned} S &= (+1\ \underline{+2\ +3\ -3}\ \underline{-2\ +1}\ -4) \\ S \circ \tau(2,5,7) &= (+1\ \underline{-2\ +1}\ \underline{+2\ +3\ -3}\ -4) \end{aligned}$$

$$\begin{aligned} P &= (1\ \underline{2\ 3\ 3}\ \underline{2\ 1}\ 4) \\ P \circ \tau(2,5,7) &= (1\ \underline{2\ 1}\ \underline{2\ 3\ 3}\ 4) \end{aligned}$$

2.3 Mapeamentos de Strings em Permutações

Uma forma de lidar com genes que apresentam múltiplas cópias é utilizar um mapeamento das strings em permutações. Dessa forma, podemos utilizar resultados de problemas para rearranjo de genomas em permutações, que são bastante estudados. Como neste projeto estamos interessados no caso de caracteres duplicados, para definir os mapeamentos que serão apresentados a seguir, assumimos que para qualquer string S, temos $occ(S) \leq 2$.

Queremos mapear a string S em uma permutação. Portanto, mantemos os caracteres não duplicados e, para cada caractere duplicado α, mapeamos as ocorrências de α em dois caracteres α' e α''. Note que temos duas formas de realizar este mapeamento. Uma possibilidade é mapear a primeira ocorrência de α em α' e a segunda ocorrência de α em α''. Por outro lado, podemos mapear a primeira e a segunda ocorrência de α em α'' e α', respectivamente.

Para mapear os caracteres duplicados em uma string S, utilizamos um vetor binário $\mathbf{x}$ que indica, para cada caractere duplicado, qual dos dois possíveis mapeamentos será realizado. Usaremos $\mathbf{x}[\alpha] = 0$ para representar a primeira possibilidade de mapeamento e $\mathbf{x}[\alpha] = 1$ para a segunda possibilidade. Denotamos por $S^{\mathbf{x}}$ a permutação gerada ao aplicarmos o mapeamento dado por $\mathbf{x}$ na string S.

Note que, para gerar uma permutação, o mapeamento precisa apenas atribuir novos valores para os caracteres duplicados, não havendo necessidade de mudar os sinais. Dessa forma, para strings com sinais, a orientação dos elementos permanece a mesma.

Considerando esta representação dos mapeamentos, é possível ver que existem $2^{|dup(S)|}$ mapeamentos possíveis de S em permutações.

Exemplo 5. *Aplicação de um mapeamento* $\mathbf{x}$ *em uma string com e sem sinais, respectivamente.*

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

$\mathbf{x} =$

4	*5*
0	*1*

A seguir apresentamos a definição de vizinhança para mapeamentos.

Definição 5. *Seja uma string* S *e dois mapeamentos* $\mathbf{x}$ *e* $\mathbf{w}$ *de* S*. Dizemos que* $\mathbf{x}$ *e* $\mathbf{w}$ *são vizinhos se eles diferem apenas para um caractere duplicado. Ou seja, para um caractere* $\alpha \in dup(S)$*, temos* $\mathbf{x}[\alpha] \neq \mathbf{w}[\alpha]$ *e* $\mathbf{x}[\beta] = \mathbf{w}[\beta], \forall_{\beta \neq \alpha}$.

Exemplo 6. *Aplicação em uma string com sinais* S *dos mapeamentos* $\mathbf{x}$*,* $\mathbf{w}$ *e* $\mathbf{z}$*. Note que* $\mathbf{x}$ *e* $\mathbf{w}$ *são vizinhos, assim como* $\mathbf{w}$ *e* $\mathbf{z}$*, mas* $\mathbf{w}$ *e* $\mathbf{z}$ *não são.*

$$\begin{aligned} S &= (+5\ +2\ -1\ -3\ +4\ -5\ -4), \quad dup(S) = \{4,5\} \\ S^{\mathbf{x}} &= (\underline{+5''}\ +2\ -1\ -3\ +4'\ \underline{-5'}\ -4'') \\ S^{\mathbf{w}} &= (\underline{+5'}\ +2\ -1\ -3\ \underline{\underline{+4'}}\ \underline{-5''}\ \underline{\underline{-4''}}) \\ S^{\mathbf{z}} &= (+5'\ +2\ -1\ -3\ \underline{\underline{+4''}}\ -5''\ \underline{\underline{-4'}}) \end{aligned}$$

$\mathbf{x} =$

4	*5*
0	*1*

$\mathbf{w} =$

4	*5*
0	*0*

$\mathbf{z} =$

4	*5*
1	*0*

O seguinte fato nos permite usar mapeamentos para encontrar a distância de rearranjo de duas strings S e P. Se utilizamos dois mapeamentos $\mathbf{x}$ e $\mathbf{y}$ para obter permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$, então $d^{\mathcal{M}}(S,P) \leq d^{\mathcal{M}}(S^{\mathbf{x}}, P^{\mathbf{y}})$. Isto ocorre porque se podemos obter $P^{\mathbf{y}}$ aplicando eventos de rearranjos em $S^{\mathbf{x}}$, o mesmo conjunto de eventos nos permite transformar S em P.

Exemplo 7. *A mesma sequência de reversões que transforma* $S^{\mathbf{x}}$ *em* $P^{\mathbf{y}}$ *sendo usada para transformar* S *em* P*.*

$$
\begin{array}{lll}
S^{\mathbf{x}} = & (5'' \underbrace{2\ \ 1}_{\rho(2,3)}\ 3\ \ 4'\ 5'\ 4'') & \\
& (5''\ 1\ \ 2\ \underbrace{3\ \ 4'\ 5'\ 4''}_{\rho(4,7)}) & \\
& (5''\ 1\ \underbrace{2\ \ 4''\ 5'}_{\rho(3,5)}\ 4'\ 3) & \\
P^{\mathbf{y}} = & (5''\ 1\ \ 5'\ 4''\ 2\ \ 4'\ 3) &
\end{array}
\qquad
\begin{array}{lll}
S = & (5\ \underbrace{2\ \ 1}_{\rho(2,3)}\ 3\ \ 4\ \ 5\ \ 4) & \\
& (5\ \ 1\ \ 2\ \underbrace{3\ \ 4\ \ 5\ \ 4}_{\rho(4,7)}) & \\
& (5\ \ 1\ \underbrace{2\ \ 4\ \ 5}_{\rho(3,5)}\ 4\ \ 3) & \\
P = & (5\ \ 1\ \ 5\ \ 4\ \ 2\ \ 4\ \ 3) &
\end{array}
$$

$$
\mathbf{x} = \begin{array}{|c|c|}\hline \scriptstyle 4 & \scriptstyle 5 \\ \hline 0 & 1 \\ \hline \end{array}
\qquad
\mathbf{y} = \begin{array}{|c|c|}\hline \scriptstyle 4 & \scriptstyle 5 \\ \hline 1 & 1 \\ \hline \end{array}
$$

2.4 Partição Mínima de Strings

A seguir apresentamos a definição de um problema de partição de strings que é relacionado a problemas de rearranjo de genomas. Este problema pertence a classe de problemas NP-Difícil [18].

Dizemos que uma string $inv(S)$ é a inversa de uma string S se ela consiste dos caracteres de S na ordem inversa, ou seja $inv(S)_i = S_{|S|-i+1}, \forall_{1 \leq i \leq |S|}$. No caso de strings com sinais os sinais são invertidos, ou seja $inv(S)_i = -S_{|S|-i+1}, \forall_{1 \leq i \leq |S|}$.

Definição 6. *Dadas duas strings balanceadas* S *e* P*, uma partição* $(\mathbb{S}, \mathbb{P}, \sigma)$ *de* S *e* P *é composta por duas sequências* $\mathbb{S}$ *e* $\mathbb{P}$ *de strings e uma permutação* σ*, tais que:*

- $\mathbb{S}$ *e* $\mathbb{P}$ *tem o mesmo número de elementos (*$|\mathbb{S}| = |\mathbb{P}|$*);*
- *Concatenando as strings de* $\mathbb{S}$ *obtemos* S*;*
- *Concatenando as strings de* $\mathbb{P}$ *obtemos* P*;*
- $\mathbb{P}_i = \mathbb{S}_{\sigma_i}$ *ou* $\mathbb{P}_i = inv(\mathbb{S}_{\sigma_i})$, $\forall_{1 \leq i \leq |\mathbb{S}|}$.

O tamanho de uma partição é igual ao número de strings em $\mathbb{S}$ ou $\mathbb{P}$ ($|(\mathbb{S}, \mathbb{P}, \sigma)| = |\mathbb{S}| = |\mathbb{P}|$). O problema de Partição Mínima de Strings consiste em encontrar uma partição de tamanho mínimo de duas strings S e P.

Exemplo 8. *Uma partição mínima entre duas strings S e P com sinais e entre duas strings S' e P' sem sinais.*

$$\begin{array}{lcl|lcl} S & = & (+5\ +2\ -1\ -3\ +4\ -5\ -4) & S' & = & (5\ 2\ 1\ 3\ 4\ 5\ 4) \\ P & = & (-4\ +3\ +1\ +5\ +2\ +4\ +5) & P' & = & (4\ 3\ 1\ 5\ 2\ 4\ 5) \\ \mathbb{S} & = & \langle(+5\ +2), (-1\ -3\ +4), (-5\ -4)\rangle & \mathbb{S}' & = & \langle(5\ 2), (1\ 3\ 4), (5\ 4)\rangle \\ \mathbb{P} & = & \langle(-4\ +3\ +1), (+5\ +2), (+4\ +5)\rangle & \mathbb{P}' & = & \langle(4\ 3\ 1), (5\ 2), (4\ 5)\rangle \\ \sigma & = & (2, 1, 3) & \sigma' & = & (2, 1, 3) \end{array}$$

Utilizaremos uma partição de strings para construir strings com sinais simplificadas que nos permitam aproximar a distância das strings originais. Dada uma partição $(\mathbb{S}, \mathbb{P}, \sigma)$ de duas strings S e P, associamos a cada string de $\mathbb{S}$ e de $\mathbb{P}$ um caractere de forma que:

- Duas strings iguais recebem o mesmo caractere com os sinais iguais;
- Duas strings inversas recebem o mesmo caractere com sinais trocados.

Usando estas atribuições de caracteres para strings, construímos as strings reduzidas $\hat{S}$ e $\hat{P}$ a partir das strings S e P, ao substituirmos as strings da partição pelos caracteres atribuídos a elas.

Exemplo 9. *As strings reduzidas correspondentes às partições do Exemplo 8. Note que, mesmo para strings originais sem sinais, devemos utilizar caracteres com sinais na string reduzida. Caso contrario não poderíamos diferenciar a string* (5 2) *do sua inversa* (2 5)*.*

$$\begin{array}{lcl} \hat{S} & = & \hat{S'} \quad = \quad (+1\ +2\ +3) \\ \hat{P} & = & \hat{P'} \quad = \quad (-2\ +1\ -3) \end{array}$$

Correspondência entre caracteres e strings:

$1 \iff (+5\ +2), (5\ 2)$

$2 \iff (-1\ -3\ +4), (1\ 3\ 4)$

$3 \iff (-5\ -4), (5\ 4)$

Note que, em strings com sinais, para um modelo de rearranjo $\mathcal{M}$, temos $d^{\mathcal{M}}(S, P) \leq d^{\mathcal{M}}(\hat{S}, \hat{P})$. Isso é verdade porque qualquer sequência de eventos de rearranjo que transforma $\hat{S}$ em $\hat{P}$, também transforma S em P.

Em strings sem sinais, podemos obter um resultado semelhante, mas temos que considerar operações com sinais ao lidar com as strings reduzidas.

Exemplo 10. *Uma sequência de reversões que transforma $\hat{S}$ em $\hat{P}$ e sequências correspondentes sendo usadas para transformar S em P e S' em P'.*

$$
\begin{array}{ll}
\hat{S} = & (\underbrace{+1\ +2}_{\rho(1,2)}\ +3) \\
& (-2\ -1\ \underbrace{+3}_{\rho(3,3)}) \\
& (-2\ \underbrace{-1}_{\rho(2,2)}\ -3) \\
\hat{P} = & (-2\ +1\ -3)
\end{array}
$$

$$
\begin{array}{ll}
S = & (\underbrace{+5\ +2\ -1\ -3\ +4}_{\rho(1,5)}\ -5\ -4) \\
& (-4\ +3\ +1\ -2\ -5\ \underbrace{-5\ -4}_{\rho(6,7)}) \\
& (-4\ +3\ +1\ \underbrace{-2\ -5}_{\rho(4,5)}\ +4\ +5) \\
P = & (-4\ +3\ +1\ +5\ +2\ +4\ +5) \\
S' = & (\underbrace{5\ \ 2\ \ 1\ \ 3\ \ 4}_{\rho(1,5)}\ \ 5\ \ 4) \\
& (4\ \ 3\ \ 1\ \ 2\ \ 5\ \ \underbrace{5\ \ 4}_{\rho(6,7)}) \\
& (4\ \ 3\ \ 1\ \ \underbrace{2\ \ 5}_{\rho(4,5)}\ \ 4\ \ 5) \\
P' = & (4\ \ 3\ \ 1\ \ 5\ \ 2\ \ 4\ \ 5)
\end{array}
$$

3 Revisão da Literatura

Para o caso em que o modelo de rearranjo é composto apenas pelo evento de reversão aplicado a genomas sem genes repetidos, em 1995, Kececioglu e Sankoff [22] apresentaram um algoritmo com fator de aproximação 2 para o problema de Ordenação de Permutações sem Sinais por Reversão. Em 1997, Caprara [9] provou que este problema pertence à classe de problemas NP-Difícil. A melhor aproximação conhecida para este problema é de 1.375, que foi apresentada por Berman *et al.* [5] em 2002.

Quando consideramos genes com orientação conhecida, em 1999, Hannenhalli e Pevzner [19] apresentaram um algoritmo polinomial para Ordenação de Permutações com Sinais por Reversão. Quando só estamos interessados no número de operações existe um algoritmo que fornece a solução em tempo linear [2].

No caso em que os genomas apresentam repetição de genes, foi provado que o problema de Distância de Reversão em Strings sem Sinais pertence à classe de problemas NP-Difícil [11], mesmo quando consideramos um alfabeto binário (quando existem apenas dois valores possíveis para os caracteres).

Diferente do caso sem genes repetidos, o problema de Distância de Reversão em Strings com Sinais pertence à classe de problemas NP-Difícil [30]. Chen *et al.* [10] provaram que este problema continua na classe NP-Difícil mesmo se considerarmos apenas genes duplicados. Eles também mostraram que este problema está relacionado com o problema de Partição Mínima de Strings com Sinais (definido na Subseção 2.4). Usando esta relação, eles apresentaram um algoritmo com fator de aproximação 3 para a Distância de Reversão em Strings com Sinais. Usando a mesma relação, Goldstein *et al.* [**?**] mostraram um algoritmo com fator de aproximação 2.2074 para este problema.

Uma relação similar existe entre os problemas "Distância de Reversão em Strings sem Sinais" e "Partição Mínima de Strings sem Sinais" [23] (definido na Subseção 2.4, os autores chamam este problema de "Reverse MCSP"). A partir dessas relações, foi apresentado um algoritmo com fator de aproximação $\Theta(k)$ para Distância de Reversão em Strings com e sem Sinais, onde k é o número máximo de cópias de um caractere nas strings consideradas [24].

Considerando um modelo de rearranjo composto apenas pelo evento de transposição, em 2012, Bulteau *et al.* [7] mostraram que o problema de Ordenação de Permutações por Transposição pertence à classe de problemas NP-Difícil. A melhor aproximação conhecida para este problema é de 1.375 [14]. Como no caso de reversão, o problema de Distância de Transposição em Strings sem Sinais pertence à classe de problemas NP-Difícil, mesmo com um alfabeto binário [30].

Quando consideramos um modelo de rearranjo composto pelos eventos de reversão e transposição, Oliveira *et al.* [28] provaram que o problema de Ordenação de Permutações por Reversão e Transposição pertence à classe de problemas NP-Difícil, para ambos os casos em que a orientação dos genes é conhecida ou desconhecida. Para o problema em que a orientação dos genes é desconhecida, existe um algoritmo com fator de aproximação $2k$ [31], onde k é a aproximação do algoritmo usado para decomposição em ciclos em uma etapa deste algoritmo de aproximação. Caso seja conhecida a orientação dos genes, existe um algoritmo com fator de aproximação 2 [33].

4 Objetivos

O projeto tem como objetivo estudar os seguintes problemas:

- Distância de Reversão em Strings com Sinais e com Genes Duplicados ($D\bar{R}$);
- Distância de Reversão em Strings sem Sinais e com Genes Duplicados (DR);
- Distância de Transposição em Strings sem Sinais e com Genes Duplicados (DT);
- Distância de Reversão e Transposição em Strings com Sinais e com Genes Duplicados ($D\bar{R}T$);
- Distância de Reversão e Transposição em Strings sem Sinais e com Genes Duplicados (DRT).

Pretendemos obter bons resultados para estes problemas a partir do desenvolvimento de heurísticas. Utilizaremos estratégias como Busca Local, GRASP [15] e Algoritmos Genéticos [27]. Com os resultados obtidos, esperamos ter um entendimento melhor destes problemas e construir uma base a partir da qual problemas mais gerais (considerando outros eventos ou genes com múltiplas cópias) possam ser estudados.

5 Metodologia

Para cada um dos problemas a serem estudados, iniciaremos realizando uma modelagem teórica. Utilizando os modelos construídos, desenvolveremos diferentes heurísticas para os problemas. Para comparar as heurísticas, iremos criar uma base de dados composta por genomas com diferentes características. Também iremos implementar algoritmos presentes na literatura. Dessa forma, podemos avaliar o desempenho das heurísticas quando comparadas com estes algoritmos.

Buscaremos trabalhar com modelos gerais que se apliquem a múltiplos problemas de rearranjo (como a ideia de mapeamento da Subseção 2.3), mas a cada etapa deste projeto focaremos em um problema específico. Assim, também podemos testar outros modelos que explorem as características específicas de cada problema.

6 Plano de Trabalho

A Tabela 1 mostra o cronograma das atividades a serem realizadas, seguida por uma breve descrição das mesmas.

Tabela 1: Cronograma das atividades.

	2019										2020												2021	
	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D	J	F
1	*	*	*	*	*				*				*				*							
2					*	*																		
3								*																
4						*	*	*	*	*														
5	*	*	*			*	*	*	*															
6			*	*	*	*	*	*	*															
7									*	*	*	*	*											
8													*	*	*	*	*							
9																	*	*	*	*	*			
10				*	*			*	*			*	*			*	*			*	*			
11					*				*				*				*				*	*		
12																						*	*	*
13																								*

1. Revisão da literatura;
2. Escrita da proposta de mestrado;
3. Exame de Qualificação de Mestrado (EQM);
4. Participação no Programa de Estágio Docente (PED);
5. Investigação do problema $D\bar{R}$;
6. Investigação do problema DR;
7. Investigação do problema DT;
8. Investigação do problema $D\bar{R}T$;
9. Investigação do problema DRT;
10. Execução dos experimentos e comparação dos resultados;
11. Escrita da dissertação;
12. Revisão da dissertação;
13. Defesa da dissertação.

Os créditos obrigatórios em disciplinas foram obtidos durante a graduação, antes do ingresso no mestrado, através do Programa Integrado de Formação (PIF). Vale ressaltar que os tempos alocados em algumas atividades podem sofrer alterações no decorrer do desenvolvimento da pesquisa, uma vez que alguns resultados obtidos podem ser mais promissores que outros, fazendo com que mais tempo seja despendido em uma atividade em detrimento de outra.

7 Resultados Preliminares

Já desenvolvemos e implementamos um conjunto de heurísticas para os problemas envolvendo apenas o evento de reversão ($D\bar{R}$ e DR).

Nestas heurísticas, utilizamos algoritmos da literatura para calcular a distância entre permutações. No caso de permutações com sinais usamos um algoritmo exato [32] e para permutações sem sinais utilizamos um algoritmo com fator de aproximação 2 [22].

Desenvolvemos um total de cinco heurísticas, sendo que quatro delas utilizam mapeamentos de strings em permutações. A qualidade desses mapeamentos é avaliada com a utilização dos algoritmos para ordenação de permutações. Note que existe uma quantidade exponencial de mapeamentos distintos de strings em permutações, sendo assim, as heurísticas usam diferentes estratégias para tentar obter um mapeamento que se aproxime ao máximo de uma solução ótima.

A primeira heurística é chamada de Mapeamentos Aleatórios (MA). Nessa abordagem, os mapeamentos são gerados de forma aleatória.

Nossa segunda heurística é chamada de Busca Local (BL). O funcionamento dessa heurística parte de um conjunto de mapeamentos aleatórios e utiliza buscas locais para gerar os novos mapeamentos.

A terceira heurística utiliza a estratégia GRASP (*Greedy Randomized Adaptive Search Procedure*) [15] para gerar os mapeamentos. Esta é uma estratégia muito utilizada em problemas de otimização combinatória [1, 8, 25] e já foi aplicada em problemas de rearranjo de genomas [12, 13].

A quarta heurística utiliza a estratégia de Algoritmos Genéticos (AG) [27] para gerar os mapeamentos. Esta estratégia também é comum em problemas de otimização combinatória [20, 29] e foi usada em problemas de rearranjo de genomas [17].

A última heurística, chamada Extremidades (Ext), não utiliza mapeamentos. Para encontrar a distância de reversão, a heurística encontra reversões que posicionam corretamente os caracteres nas extremidades da string. Estas reversões são selecionadas com base em um critério guloso.

Além das heurísticas, também implementamos o algoritmo SOAR apresentado por Chen *et al.* [10] que possui fator de aproximação 3. O mesmo algoritmo pode ser adaptado para o caso sem sinais, mas como ele utiliza a relação entre os problemas de Distância de Reversão em Strings e Partição Mínima de Strings, no caso sem sinais o fator de aproximação é 6.

O Apêndice A apresenta uma descrição detalhada destas heurísticas, assim como os resultados da execução das heurísticas e do algoritmo SOAR na base de dados desenvolvida.

Referências

[1] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29(4):393–430, Apr. 2003.

[2] D. A. Bader, B. M. E. Moret, and M. Yan. A Linear-Time Algorithm for Computing Inversion Distance Between Signed Permutations with an Experimental Study. *Journal of Computational Biology*, 8:483–491, 2001.

[3] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[4] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[5] P. Berman, S. Hannenhalli, and M. Karpinski. 1.375-Approximation Algorithm for Sorting by Reversals. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'2002)*, ESA'2002, pages 200–210, London, UK, 2002. Springer-Verlag.

[6] L. Bulteau, G. Fertin, and C. Komusiewicz. (Prefix) Reversal Distance for (Signed) Strings with Few Blocks or Small Alphabets. *Journal of Discrete Algorithms*, 37:44–55, 2016.

[7] L. Bulteau, G. Fertin, and I. Rusu. Sorting by Transpositions is Difficult. *SIAM Journal on Computing*, 26(3):1148–1180, 2012.

[8] R. G. Cano, G. Kunigami, C. C. de Souza, and P. J. de Rezende. A Hybrid GRASP Heuristic to Construct Effective Drawings of Proportional Symbol Maps. *Computers & Operations Research*, 40(5):1435–1447, 2013.

[9] A. Caprara. Sorting Permutations by Reversals and Eulerian Cycle Decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[10] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of Orthologous Genes via Genome Rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[11] D. A. Christie and R. W. Irving. Sorting Strings by Reversals and by Transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[12] T. da Silva Arruda, U. Dias, and Z. Dias. A GRASP-based heuristic for the sorting by length-weighted inversions problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(2):352–363, 2018.

[13] U. Dias, C. Baudet, and Z. Dias. Greedy Randomized Search Procedure to Sort Genomes Using Symmetric, Almost-Symmetric and Unitary Inversions. In *Proceedings of the 4th International Conference on Bioinformatics, Computational Biology and Biomedical Informatics (BCB'2013)*, volume 8542 of *Lecture Notes in Computer Science*, pages 181–190, New York, NY, USA, 2013.

[14] I. Elias and T. Hartman. A 1.375-Approximation Algorithm for Sorting by Transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[15] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, Mar 1995.

[16] G. Fertin, A. Labarre, I. Rusu, É. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, London, England, 1st edition, 2009.

[17] N. Gao, N. Yang, and J. Tang. Ancestral Genome Inference Using a Genetic Algorithm Approach. *PLOS ONE*, 8(5):1–6, 2013.

[18] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. In *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'2004)*, pages 484–495, Berlin, Heidelberg, 2005.

[19] S. Hannenhalli and P. A. Pevzner. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *Journal of ACM*, 46(1):1–27, 1999.

[20] P. Jog, J. Y. Suh, and D. Van Gucht. Parallel genetic algorithms applied to the traveling salesman problem. *SIAM Journal on Optimization*, 1:515 – 529, 11 1991.

[21] J. D. Kececioglu and R. Ravi. Of Mice and Men: Algorithms for Evolutionary Distances Between Genomes with Translocation. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'1995)*, pages 604–613, Philadelphia, PA, USA, 1995.

[22] J. D. Kececioglu and D. Sankoff. Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. *Algorithmica*, 13(1):180–210, Feb 1995.

[23] P. Kolman and T. Waleń. Approximating Reversal Distance for Strings with Bounded Number of Duplicates. *Discrete Applied Mathematics*, 155(3):327–336, 2007.

[24] P. Kolman and T. Waleń. Reversal Distance for Strings with Duplicates: Linear Time Approximation Using Hitting Set. In *Proceedings of the 4th International Workshop on Approximation and Online Algorithms (WAOA'2006)*, pages 279–289, Berlin, Heidelberg, 2007.

[25] M. Laguna, T. A. Feo, and H. C. Elrod. A Greedy Randomized Adaptive Search Procedure for the Two-Partition Problem. *Operations Research*, 42(4):677–687, 1994.

[26] X. Lou and D. Zhu. Genome rearrangement algorithms for unsigned permutations with o(log n) singletons. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC'2008)*, pages 59–69, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[28] A. R. Oliveira, K. L. Brito, U. Dias, and Z. Dias. On the complexity of sorting by reversals and transpositions problems. *Journal of Computational Biology*. PMID: 31120331.

[29] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202 – 3212, 2008. Part Special Issue: Search-based Software Engineering.

[30] A. J. Radcliffe, A. D. Scott, and E. L. Wilmer. Reversals and Transpositions Over Finite Alphabets. *SIAM Journal on Discrete Mathematics*, 19(1):224–244, 2005.

[31] A. Rahman, S. Shatabda, and M. Hasan. An Approximation Algorithm for Sorting by Reversals and Transpositions. *Journal of Discrete Algorithms*, 6(3):449–457, 2008.

[32] G. Tesler. GRIMM: Genome Rearrangements Web Server. *Bioinformatics*, 18(3):492–493, 2002.

[33] M. E. M. T. Walter, Z. Dias, and J. Meidanis. Reversal and Transposition Distance of Linear Chromosomes. In *Proceedings of the 5th International Symposium*

on String Processing and Information Retrieval (SPIRE'1998), pages 96–102, Los Alamitos, CA, USA, 1998.

[34] J. Zhang. Evolution by gene duplication: an update. *Trends in Ecology & Evolution*, 18(6):292 – 298, 2003.

A Heurísticas Desenvolvidas

Neste apêndice, descrevemos detalhadamente o funcionamento das cinco heurísticas que foram desenvolvidas até a momento para os problemas $D\bar{R}$ e DR.

Durante a descrição das heurísticas, nos referimos à distância de reversão apenas como distância. Além disso, quando falamos da distância entre duas permutações, nos referimos ao resultado dos algoritmos utilizados (um algoritmo exato para o caso com sinais [32] e um com fator de aproximação 2 para o caso sem sinais [22]).

As primeiras quatro heurísticas (Mapeamentos Aleatórios, Busca Local, GRASP e Algoritmos Genéticos) utilizam a ideia de mapeamentos apresentada na Subseção 2.3. Nestas heurísticas, dada a string origem S e a string destino P, P será mapeada em uma permutação $P^{\mathbf{y}}$ por um mapeamento arbitrário $\mathbf{y}$ (o mapeamento correspondente ao vetor binário composto apenas por zeros) e será gerado um conjunto $\mathbf{M}$ de mapeamentos da string S em permutações. As heurísticas diferem pela forma como o conjunto $\mathbf{M}$ é gerado. Note que, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, temos um limitante superior para a distância entre S e P ($dist(S, P) \leq dist(S^{\mathbf{x}}, P^{\mathbf{y}})$). O resultado destas heurísticas é a menor distância entre $P^{\mathbf{y}}$ e algum mapeamento de $\mathbf{M}$.

A.1 Mapeamentos Aleatórios (MA)

Nesta heurística, os mapeamentos do conjunto $\mathbf{M}$ são gerados de forma aleatória. Além das strings S e P, essa heurística também recebe como entrada um parâmetro $r \in \mathbb{N}$, que indica quantos mapeamentos serão gerados.

Inicialmente mapeamos P em uma permutação $P^{\mathbf{y}}$ com um mapeamento $\mathbf{y}$ qualquer. Em seguida, um total de r mapeamentos em permutações da string S são gerados. Cada elemento dos vetores correspondentes a estes mapeamentos têm uma probabilidade de 50% de receber o valor 0 e 50% de receber o valor 1. Como resultado, o conjunto $\mathbf{M}$ com r mapeamentos distintos é obtido.

Por fim, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, obtemos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. Como resultado para a distância entre as strings, usamos a menor distância encontrada entre todos os mapeamentos de $\mathbf{M}$.

No Algoritmo 1, temos um pseudocódigo desta heurística. A Figura 1 apresenta uma simulação do funcionamento da heurística com $S = (3\ 2\ 1\ 2\ 4\ 3\ 4)$ e $P = (1\ 3\ 4\ 2\ 2\ 4\ 3)$. Neste caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}_1}, P^{\mathbf{y}}) \leq 4$.

Algoritmo 1: MA

Entrada: Strings S e P, número de mapeamentos r
Saída: Distância entre S e P

```
1 início
2     y ← mapeamento_padrão(P)
3     M ← ∅
4     enquanto |M| < r faça
5         y ← gera_mapeamento_aleatório(S)
6         M ← M ∪ {y}
7     retorna menor_distância(M, S, P^y)
```

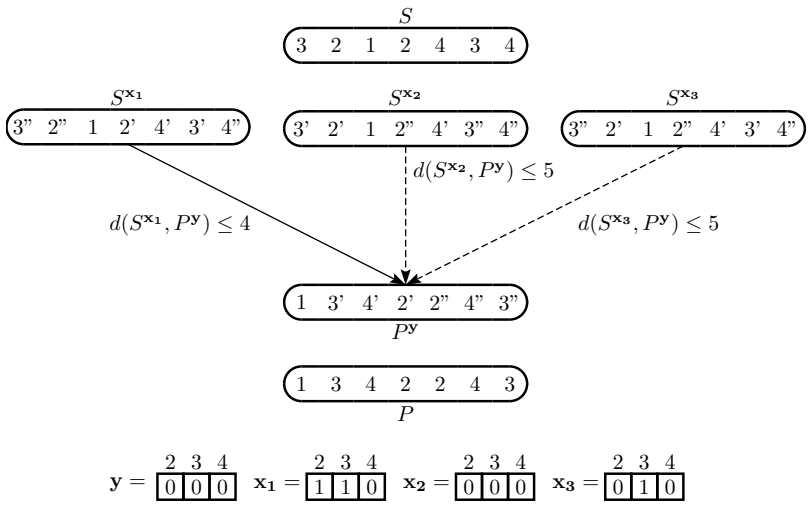


Figura 1: Exemplo da heurística de mapeamos aleatórios com $r = 3$.

A.2 Busca Local (BL)

Nesta heurística, apenas os mapeamentos iniciais do conjunto $\mathbf{M}$ são gerados de forma aleatória. Para gerar o restante dos mapeamentos, exploramos a vizinhança de alguns dos mapeamentos já conhecidos. Além das strings S e P, essa heurística também recebe como entrada os parâmetros $r, c, \ell \in \mathbb{N}$, que indicam respectivamente quantos mapeamentos serão gerados no total, quantos mapeamentos serão gerados de forma aleatória e qual o máximo de vizinhos que podem ser explorados para cada mapeamento.

Novamente, mapeamos P em uma permutação $P^{\mathbf{y}}$ com um mapeamento $\mathbf{y}$ qualquer. Em seguida, inicializamos $\mathbf{M}$ com c mapeamentos aleatórios, gerados usando a mesma estratégia da heurística anterior. Para obtermos os $r-c$ mapeamentos restantes, realizamos buscas locais. Em cada busca, adicionamos até ℓ novos mapeamentos em $\mathbf{M}$. Para garantir que não utilizamos duas vezes o mesmo mapeamento, mantemos um conjunto $\mathbf{E}$ de mapeamentos já explorados.

Em uma busca local, para cada mapeamento $\mathbf{x}$ de $\mathbf{M}$ ainda não explorado, calculamos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. Dentre estes mapeamentos, escolhemos o que gera a menor distância, em caso de empate escolhemos qualquer um dos mapeamentos empatados. Seja $\mathbf{V}$ o conjunto de vizinhos do mapeamento escolhido que não estão em $\mathbf{M}$, selecionamos aleatoriamente um conjunto $\mathbf{V'} \subset \mathbf{V}$, com $|\mathbf{V'}| = min(\ell, |\mathbf{V}|, r - |\mathbf{M}|)$ mapeamentos.

No fim da busca, adicionamos o mapeamento escolhido em $\mathbf{E}$ e adicionamos a $\mathbf{M}$ os mapeamentos de $\mathbf{V'}$. Note que o número de elementos escolhidos garante que geramos no máximo ℓ mapeamentos por busca e que após a última busca $\mathbf{M}$ terá exatamente r elementos.

Após completarmos $\mathbf{M}$ com r mapeamentos, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, obtemos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. Assim como na heurística anterior, o resultado para a distância entre as strings será a menor distância encontrada entre todos os mapeamentos de $\mathbf{M}$.

No Algoritmo 2, temos um pseudocódigo desta heurística. A Figura 2 apresenta uma simulação da heurística em um par de strings $S = (3\ 2\ 1\ 2\ 4\ 3\ 4\ 1)$ e $P = (1\ 3\ 4\ 2\ 1\ 2\ 4\ 3)$. Neste caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}_2''}, P^{\mathbf{y}}) \leq 3$.

Algoritmo 2: BL

Entrada: Strings S e P, número total de mapeamentos r, número de mapeamentos aleatórios c e número máximo de vizinhos explorados ℓ

Saída: Distância entre S e P

```
1  início
2      y ← mapeamento_padrão(P)
3      M ← mapeamentos_aleatórios(c, S)
4      E ← ∅
5      enquanto |M| < r faça
6          a ← melhor_mapeamento(M − E, S, P^y)
7          V ← vizinhos(a) − M
8          V' ← escolhe_mapeamentos(V, min(ℓ, |V|, r − |M|))
9          E ← E ∪ {a}
10         M ← M ∪ V'
11     retorna menor_distância(M, S, P^y)
```

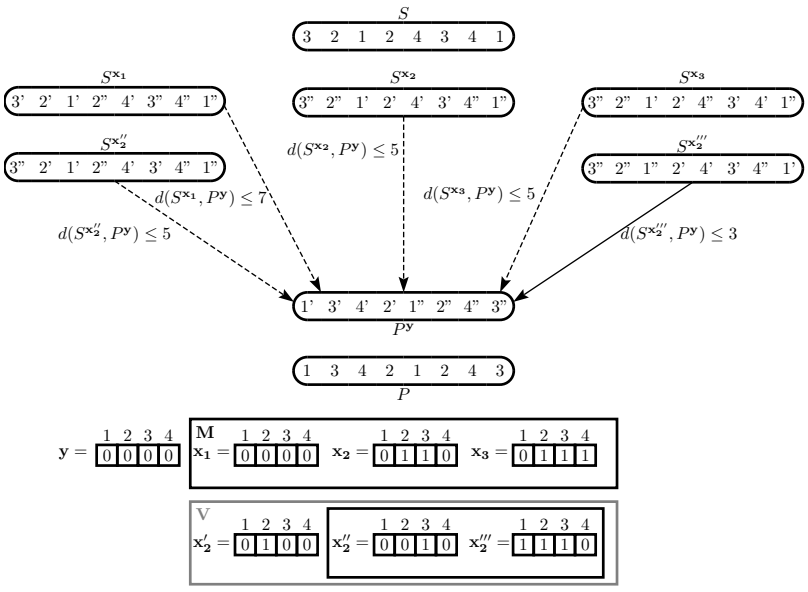


Figura 2: Exemplo da heurística de Busca Local com $r = 5$, $c = 3$ e $\ell = 2$. O conjunto $\mathbf{M}$ é composto dos 3 mapeamentos aleatórios gerados inicialmente, o conjunto $\mathbf{V}$ são os vizinhos do mapeamento $\mathbf{x}_2$ que não estão em $\mathbf{M}$ e os mapeamentos $\mathbf{x}''_2$ e $\mathbf{x}'''_2$ são os 2 mapeamentos selecionados para serem parte dos 5 mapeamentos finais.

A.3 GRASP

Esta heurística também parte de um conjunto de mapeamentos aleatórios iniciais, mas o restante dos mapeamentos é gerado através da estratégia GRASP. Esta estratégia consiste em uma etapa de construção de uma solução aleatória, a partir de uma lista de candidatos RCL (do inglês "Restricted Candidate List"), seguida de uma busca local a partir desta solução. Além das strings S e P, essa heurística também recebe como entrada os parâmetros $r, c, k, g, r_\ell \in \mathbb{N}$. Como na heurística BL, r indica quantos mapeamentos serão gerados no total e c indica quantos mapeamentos serão gerados de forma aleatória. O parâmetro k é o tamanho da RLC e o parâmetro g é a quantidade máxima de mapeamentos gerados na etapa de construção. O parâmetro r_ℓ é o número total de mapeamentos gerados na etapa de Busca Local. Esta etapa é uma adaptação do heurística BL.

Assim como nas heurísticas anteriores, mapeamos P em uma permutação $P^{\mathbf{y}}$ com um mapeamento $\mathbf{y}$ qualquer. Em seguida, inicializamos $\mathbf{M}$ com c mapeamentos aleatórios, gerados usando a mesma estratégia da heurística MA. Os $r - c$ mapeamentos restantes são gerados pelas etapas da estratégia GRASP. Na etapa de construção, geramos até g mapeamentos. Usamos estes g mapeamentos para inicializar uma busca local. Nesta busca, geramos r_ℓ novos mapeamentos. A seguir explicamos estas duas etapas.

Na etapa de construção, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, calculamos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. Com estas distâncias, obtemos o conjunto $\mathbf{RCL} \subset \mathbf{M}$, composto dos k melhores mapeamentos de $\mathbf{M}$ (mapeamentos que geram as menores distâncias).

Dada o conjunto $\mathbf{RCL}$, para cada caractere duplicado α calculamos o valor:

$$prob(\mathbf{RCL}, \alpha) = \frac{freq(\mathbf{RCL}, \alpha, 0)}{freq(\mathbf{RCL}, \alpha, 0) + freq(\mathbf{RCL}, \alpha, 1)}$$

Onde, $freq(\mathbf{RLC}, \alpha, b), b \in \{0, 1\}$ é o número de mapeamentos de $\mathbf{RLC}$ para os quais o elemento corresponde a α no vetor binário tem valor b.

Exemplo 11. *Cálculo dos valores apresentados anteriormente para um conjunto* **RLC** *de mapeamentos da string sem sinais* S*.*

$$S = (2\ 1\ 3\ 4\ 3\ 4), \quad dup(S) = dup(P) = \{3, 4\}$$

$$\mathbf{RLC} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$$

	3	4
$\mathbf{x}_1 =$	1	0

	3	4
$\mathbf{x}_2 =$	0	0

	3	4
$\mathbf{x}_3 =$	1	0

$freq(\mathbf{RLC}, 3, 0) = 1 \quad freq(\mathbf{RLC}, 4, 0) = 3$

$freq(\mathbf{RLC}, 3, 1) = 2 \quad freq(\mathbf{RLC}, 4, 1) = 0$

$prob(\mathbf{RLC}, 3) = \frac{1}{3} \quad prob(\mathbf{RLC}, 4) = 1$

Com estes valores, geramos um conjunto $\mathbf{N}$ com g novos mapeamentos. Cada mapeamento $\mathbf{n}$ de $\mathbf{N}$ é gerado aleatoriamente, sendo que, para um caractere $\alpha \in dup(S)$, a probabilidade de termos $\mathbf{n}[\alpha] = 0$ é $p = prob(\mathbf{RCL}, \alpha)$ e a probabilidade de termos $\mathbf{n}[\alpha] = 1$ é $1 - p$. Os mapeamentos de $\mathbf{N}$ são adicionados a $\mathbf{M}$ e passamos para etapa de busca local.

Na etapa de busca local, utilizamos uma variação da heurística BL (BL_GRASP). Esta variação recebe como entrada os conjuntos $\mathbf{M}$ e $\mathbf{N}$, a string S, a permutação $P^{\mathbf{y}}$ e o parâmetro r_ℓ. Como resultado obtemos o conjunto $\mathbf{M}$, com a adição de r_ℓ novos mapeamentos gerados a partir de buscas locais.

Nesta variação utilizamos o conjunto $\mathbf{N}$ no lugar dos mapeamentos aleatórios. Similarmente à heurística BL, mantemos o conjunto $\mathbf{E}$ e geramos o conjunto V. No lugar do conjunto $\mathbf{V}'$, escolhemos apenas um mapeamento e, ao final da busca, adicionamos este mapeamento aos conjuntos $\mathbf{M}$ e $\mathbf{N}$. Como essa variação lida com um conjunto menor de mapeamentos, apenas adicionamos o mapeamento selecionado ao conjunto $\mathbf{E}$ quando todos os vizinhos dele já estão em $\mathbf{M}$.

Repetimos estas duas etapas até obtermos r mapeamentos em $\mathbf{M}$. Por fim, como nas heurísticas anteriores, para cada mapeamento $\mathbf{x} \in \mathbf{M}$ obtemos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. O resultado para a distância entre as strings será a menor distância encontrada entre todos os mapeamentos de $\mathbf{M}$.

Nos algoritmos 3 e 4, temos um pseudocódigo desta heurística. A Figura 3 apresenta uma simulação da heurística em um par de strings $S = (+3\ -2\ -1\ -2\ -4\ -3\ +4)$ e $P = (+3\ +3\ +2\ +4\ +2\ +4\ +1)$. Neste caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{z}_1}, P^{\mathbf{y}}) \leq 5$.

Algoritmo 3: GRASP

Entrada: Strings S e P, número total de mapeamentos r, número de mapeamentos aleatórios c, tamanho da RLC k, número de mapeamentos gerados na etapa de construção g e número de mapeamentos gerados na busca local r_ℓ

Saída: Distância entre S e P

```
1  início
2      y ← mapeamento_padrão(P)
3      M ← mapeamentos_aleatórios(c, S)
4      enquanto |M| < r faça
5          RCL ← melhores_mapeamentos(M, k, S, P^y)
6          N ← gera_novos_mapeamentos(RCL, g)
7          M ← M ∪ N
8          M ← BL_GRASP(M, N, S, P^y, r_ℓ)
9      retorna menor_distância(M, S, P^y)
```

Algoritmo 4: BL_GRASP

Entrada: Conjuntos $\mathbf{M}$ e $\mathbf{N}$, String S, Permutação $P^{\mathbf{y}}$ e número total de mapeamentos r_ℓ

Saída: Novo conjunto $\mathbf{M}$

```
1  início
2      E ← ∅
3      r_antigo ← |M|
4      enquanto |M| < r_antigo + r_ℓ faça
5          a ← melhor_mapeamento(N − E, S, P^y)
6          V ← vizinhos(a) − M
7          a' ← escolhe_mapeamento(V)
8          se Todos os vizinhos de a estão em M então
9              E ← E ∪ {a}
10         M ← M ∪ {a'}
11         N ← N ∪ {a'}
12     retorna M
```

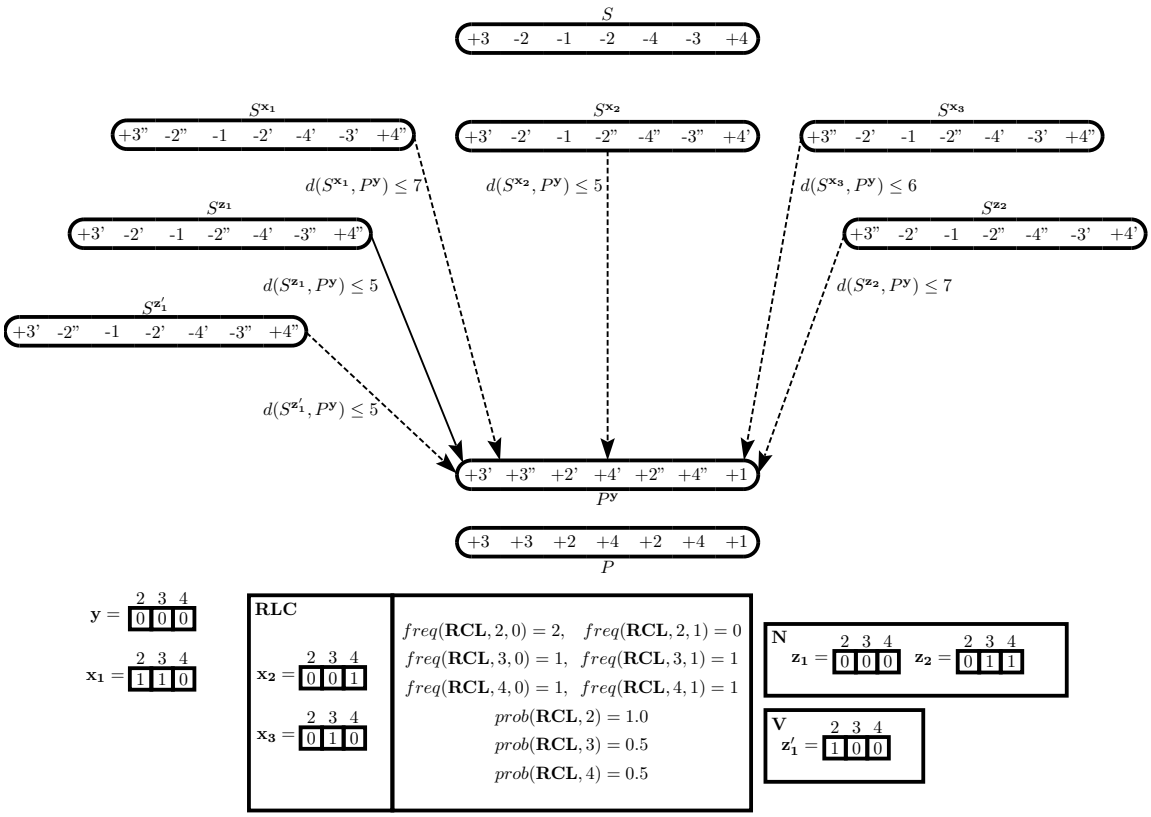


Figura 3: Exemplo da heurística GRASP com os parâmetros $r = 6$, $c = 3$, $k = 2$, $g = 2$ e $r_\ell = 1$. Os mapeamentos $\mathbf{x}_1$, $\mathbf{x}_2$ e $\mathbf{x}_3$ são os 3 mapeamentos aleatórios gerados inicialmente. O mapeamento $\mathbf{z}_1$ foi escolhido na busca local, como $\mathbf{z}'_1$ era seu único vizinho desconhecido ele foi o mapeamento gerado.

A.4 Algoritmo Genético (AG)

Esta heurística utiliza um algoritmo genético para gerar os mapeamentos. O algoritmo genético é composto das etapas de geração de uma população inicial, seleção de mapeamentos para próxima geração e criação de novos mapeamentos em cada geração por mutações e cruzamentos. Além das strings S e P, essa heurística também recebe como entrada os parâmetros $r, c, k, t_c, t_m \in \mathbb{N}$. Como nas heurísticas BL e GRASP, r indica quantos mapeamentos serão gerados no total e c indica quantos mapeamentos serão gerados para população inicial. O parâmetro k é o número de mapeamentos que serão selecionados para a próxima geração e os parâmetros t_c e t_m são usados nas operações de cruzamento e mutação, respectivamente.

Assim como nas heurísticas anteriores, mapeamos P em uma permutação $P^{\mathbf{y}}$ com um mapeamento $\mathbf{y}$ qualquer. Em seguida, inicializamos $\mathbf{M}$ com c mapeamentos aleatórios, gerados usando a mesma estratégia da heurística MA. Esta é a população inicial. Geramos os $r - c$ mapeamentos restantes através de mutações e cruzamentos dos mapeamentos selecionados em cada geração. O conjunto $\mathbf{G}$ representa os mapeamentos presentes na população em cada geração. Inicialmente $\mathbf{G}$ é composto dos mapeamentos do conjunto $\mathbf{M}$.

No inicio de uma geração, para cada mapeamento $\mathbf{x} \in \mathbf{G}$, calculamos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. Com estas distâncias, selecionamos os k melhores mapeamentos de $\mathbf{G}$ (mapeamentos que geram as menores distâncias), o restante dos mapeamentos são removidos do conjunto $\mathbf{G}$.

Em seguida, aumentamos a população de $\mathbf{G}$ aplicando os seguintes cruzamentos e mutações em seus elementos.

- **Cruzamentos:** pareamos todos os k mapeamentos do conjunto $\mathbf{G}$. Em seguida, para cada par $\mathbf{x}_1$ e $\mathbf{x}_2$ geramos um novo mapeamento, onde t_c caracteres duplicados de S (escolhidos aleatoriamente) serão mapeados de acordo com $\mathbf{x}_1$ e os $|S| - t_c$ restantes de acordo com $\mathbf{x}_2$.

Exemplo 12. *Um mapeamento* $\mathbf{z}$ *gerado através de um cruzamento de dois mapeamentos* $\mathbf{x}_1$ *e* $\mathbf{x}_2$ *de uma string sem sinais* S*.*

$$
\begin{aligned}
S &= (3\ 3\ 1\ 4\ 2\ 2\ 4), \quad t_c = 2 \\
S^{\mathbf{x}_1} &= (3''\ 3'\ 1\ \underline{4''\ 2'\ 2''\ 4'}) \\
S^{\mathbf{x}_2} &= (\underline{\underline{3'\ 3''}}\ 1\ 4'\ 2''\ 2'\ 4'') \\
S^{\mathbf{z}} &= (\underline{\underline{3'\ 3''}}\ 1\ \underline{4''\ 2'\ 2''\ 4'})
\end{aligned}
$$

$\mathbf{x}_1 =$

2	*3*	*4*
0	1	1

$\mathbf{x}_2 =$

2	*3*	*4*
1	0	0

$\mathbf{z} =$

2	*3*	*4*
0	0	1

- **Mutações:** para cada mapeamento $m \in \mathbf{G}$, criamos um novo mapeamento invertendo t_m elementos (escolhidos aleatoriamente) do vetor correspondente ao mapeamento $\mathbf{M}$.

  Exemplo 13. *Um mapeamento* $\mathbf{z}$ *gerado através de uma mutação em um mapeamento* $\mathbf{x}$ *de uma string sem sinais* S*.*

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

$\mathbf{x} =$

2	*3*	*4*
0	1	1

$\mathbf{z} =$

2	*3*	*4*
0	0	0

Ao final da geração, adicionamos os mapeamentos gerados nos conjuntos $\mathbf{G}$ e $\mathbf{M}$. Caso o número de mapeamentos no conjunto $\mathbf{M}$ passe de r, ignoramos alguns dos mapeamentos gerados por mutações e, se necessário, gerados por cruzamentos.

Após todas as gerações, temos um conjunto $\mathbf{M}$ com r mapeamentos. Como nas heurísticas anteriores, para cada mapeamento $\mathbf{x} \in \mathbf{M}$ obtemos a distância entre as permutações $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$. O resultado para a distância entre as strings será a menor distância encontrada entre todos os mapeamentos de $\mathbf{M}$.

No Algoritmo 5, temos um pseudocódigo desta heurística. A Figura 4 apresenta uma simulação da heurística em um par de strings $S = (+3\ -2\ -1\ -2\ -4\ -3\ +4)$ e $P = (+3\ +3\ +2\ +4\ +2\ +4\ +1)$. Neste caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{z}_1}, P^{\mathbf{y}}) \leq 5$.

Algoritmo 5: GenAlg

Entrada: Strings S e P, número total de mapeamentos r, número de mapeamentos aleatórios c, número de mapeamentos selecionados k, parâmetro dos cruzamentos t_c e parâmetro das mutações t_m

Saída: Distância entre S e P

início
  $\mathbf{y} \leftarrow mapeamento\_padrão(P)$
  $\mathbf{M} \leftarrow mapeamentos\_aleatórios(c, S)$
  $\mathbf{G} \leftarrow \mathbf{M}$
  enquanto $|\mathbf{M}| < r$ **faça**
    $\mathbf{G} \leftarrow seleciona\_melhores\_mapeamentos(\mathbf{G}, k, S, P^{\mathbf{y}})$
    $\mathbf{G} \leftarrow \mathbf{G} \cup cruzamentos(\mathbf{G}, t_c, min(\lfloor \frac{k}{2} \rfloor, r - |M|)) \cup$
     $mutacões(\mathbf{G}, t_m, min(k, max(0, r - |M| - \lfloor \frac{k}{2} \rfloor)))$
    $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{G}$
  retorna $menor\_distância(M, S, P^{\mathbf{y}})$

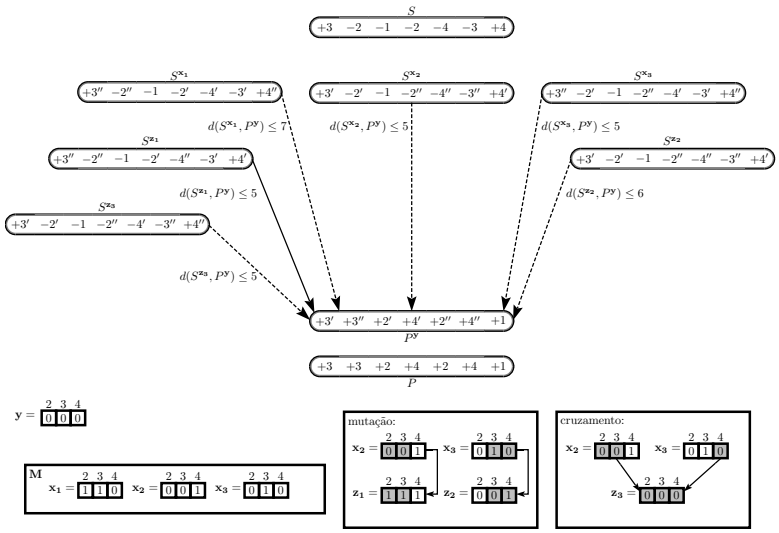


Figura 4: Exemplo da heurística de Algoritmos Genéticos com os parâmetros $r = 6$, $c = 3$, $k = 2$, $t_m = 2$ e $t_c = 2$. Os mapeamentos $\mathbf{x}_1$, $\mathbf{x}_2$ e $\mathbf{x}_3$ são os 3 mapeamentos aleatórios gerados inicialmente. Os mapeamentos $\mathbf{x}_2$ e $\mathbf{x}_3$ foram escolhidos para sofrer mutações e cruzamentos.

A.5 Extremidades (Ext)

Essa heurística recebe como entrada as strings S e P. Para obter a distância de reversão, aplicamos reversões que posicionam um caractere corretamente em alguma extremidade da string e aplicamos a heurística recursivamente para posicionar o restante. Como existem várias reversões que podem ser usadas, escolhemos uma delas usando uma estratégia gulosa.

Dada duas strings S e P, inicialmente a heurística realiza um processamento que remove os caracteres das extremidades das strings que estão na posição correta, com isso as strings S' e P' são obtidas de forma que $|S'| \leq |S|$, $|P'| \leq |P|$ e $|S'| = |P'|$.

Exemplo 14. *Processamento para remover os caracteres posicionados corretamente em strings sem sinais S e P.*

$$\begin{aligned} S &= (5\ 2\ 1\ 3\ 4\ 5\ 4) \\ P &= (5\ 1\ 4\ 2\ 3\ 5\ 4) \\ S' &= (2\ 1\ 3\ 4) \\ P' &= (1\ 4\ 2\ 3) \end{aligned}$$

Exemplo 15. *Processamento para remover os caracteres posicionados corretamente em strings com sinais S e P.*

$$\begin{aligned} S &= (-5\ -1\ +2\ -3\ +4\ +5\ +4) \\ P &= (-5\ +1\ +4\ -2\ +3\ +5\ +4) \\ S' &= (-1\ +2\ -3\ +4) \\ P' &= (+1\ +4\ -2\ +3) \end{aligned}$$

Note que se obtivermos uma sequência de reversões que transforme S' em P', então é possível obter uma sequência que transforma S em P, bastando manter os caracteres removidos na mesma posição.

Caso $occ(S') = 1$, temos uma permutação e utilizamos um algoritmo para ordenação de permutações. Caso contrário, aplicamos reversões para posicionar corretamente um caractere em uma das extremidades de S'. Escolhemos quais reversões usar através de uma função gulosa $rank$.

Dada uma sequência R, com $|R| \in \{1, 2\}$ reversões, que posicionam corretamente em S' uma das extremidades de P'. Denotamos por $corretos(R, S', P')$ o número de caracteres posicionados corretamente em um prefixo ou sufixo de S', ou seja, o comprimento do maior prefixo ou sufixo comum entre S' (após aplicarmos as reversões de R) e P'. Definimos o função $rank$ por:

$$rank(R, S', P') = \frac{corretos(R, S', P')}{|R|}$$

Para strings com sinais, no cálculo da função $rank$, comparamos também os sinais. Além disso, caso o elemento posicionado na extremidade tenha o sinal errado, adicionamos uma reversão extra na sequência R que atua somente revertendo este elemento (neste caso R pode ter até 3 reversões).

Exemplo 16. *Cálculo de alguns ranks em strings sem sinais.*

$S' = \quad (5\ 2\ 3\ 4\ 5\ 4\ 3\ 1)$

$P' = \quad (3\ 4\ 2\ 5\ 1\ 5\ 3\ 4)$

$rank(\langle \rho(1,7) \rangle, S', P') = 2:$ $\quad S' = (\underbrace{5\ 2\ 3\ 4\ 5\ 4\ 3}_{\rho(1,7)} 1) \rightarrow (3\ 4\ 5\ 4\ 3\ 2\ 5\ 1)$

$rank(\langle \rho(6,8) \rangle, S', P') = 2:$ $\quad S' = (5\ 2\ 3\ 4\ 5\ \underbrace{4\ 3\ 1}_{\rho(6,8)}) \rightarrow (5\ 2\ 3\ 4\ 5\ 1\ 3\ 4)$

$rank(\langle \rho(1,4), \rho(1,2) \rangle, S', P') = 2:$ $\quad S' = (\underbrace{5\ 2\ 3\ 4}_{\rho(1,4)}\ 5\ 4\ 3\ 1) \rightarrow (\underbrace{4\ 3}_{\rho(1,2)}\ 2\ 5\ 5\ 4\ 3\ 1)$
$\rightarrow (3\ 4\ 2\ 5\ 5\ 4\ 3\ 1)$

$rank(\langle \rho(3,8), \rho(7,8) \rangle, S', P') = 1,5:$ $\quad S' = (5\ 2\ \underbrace{3\ 4\ 5\ 4\ 3\ 1}_{\rho(3,8)}) \rightarrow (5\ 2\ 1\ 3\ 4\ 5\ \underbrace{4\ 3}_{\rho(7,8)})$
$\rightarrow (5\ 2\ 1\ 3\ 4\ 5\ 3\ 4)$

Exemplo 17. *Cálculo de alguns ranks em strings com sinais.*

$$
\begin{aligned}
S' &= (-1\ -2\ +3\ +4\ +1\ -4\ +3)\\
P' &= (+3\ +4\ +2\ +1\ -1\ +3\ -4)
\end{aligned}
$$

$rank(\langle \rho(1,7), \rho(1,1) \rangle, S', P') = 1:$

$S' = (\underbrace{-1\ -2\ +3\ +4\ +1\ -4\ +3}_{\rho(1,7)}) \rightarrow (\underbrace{-3}_{\rho(1,1)}\ +4\ -1\ -4\ -3\ +2\ +1)$

$\rightarrow (+3\ +4\ -1\ -4\ -3\ +2\ +1)$

$rank(\langle \rho(1,4), \rho(1,2) \rangle, S', P') = 2:$

$S' = (\underbrace{-1\ -2\ +3\ +4}_{\rho(1,4)}\ +1\ -4\ +3) \rightarrow (\underbrace{-4\ -3}_{\rho(1,2)}\ +2\ +1\ +1\ -4\ +3)$

$\rightarrow (+3\ +4\ +2\ +1\ +1\ -4\ +3)$

$rank(\langle \rho(3,8), \rho(7,8), \rho(8,8) \rangle, S', P') = 1:$

$S' = (-1\ -2\ \underbrace{+3\ +4\ +1\ -4\ +3}_{\rho(3,8)}) \rightarrow (-1\ -2\ -3\ +4\ -1\ \underbrace{-4\ -3}_{\rho(7,8)})$

$\rightarrow (-1\ -2\ -3\ +4\ -1\ +3\ \underbrace{+4}_{\rho(8,8)}) \rightarrow (-1\ -2\ -3\ +4\ -1\ +3\ -4)$

Para o caso em que temos um empate nestes valores, damos preferência para a sequência que utiliza menos reversões. Se o empate persistir, escolhemos a sequência que posiciona um elemento no começo da string.

Por fim, a heurística aplica a(s) operação(ões) com melhor custo-benefício segundo a função *rank*. Com isso, obtemos uma string S'', onde sabemos que pelo menos um caractere foi posicionado corretamente em uma das extremidades e recursivamente a heurística é aplicada em S'' e P'.

Note que a heurística é capaz de transformar uma string em outra, pois uma vez que um ou mais caracteres são posicionados corretamente nas extremidades eles não mudam mais de posição e a cada interação pelo menos um caractere é posicionado corretamente em uma das extremidades da string.

Exemplo 18. *Uma sequência de reversões fornecida pela heurística sendo aplicada em strings com sinais.*

$$P = (-2\ +2\ -1\ -1\ -3)$$
$$S = (\underline{+1\ -2\ -2}\ -1\ -3)$$
$$S \circ \rho(1,3) = (\underline{+2}\ +2\ -1\ -1\ -3)$$
$$S \circ \rho(1,3) \circ \rho(1,1) = (-2\ +2\ -1\ -1\ -3)$$

Exemplo 19. *Uma sequência de reversões fornecida pela heurística sendo aplicada em strings sem sinais.*

$$P = (3\ 4\ 5\ 2\ 5\ 3\ 1\ 4)$$
$$S = (\underline{5\ 4\ 3}\ 3\ 5\ 2\ 1\ 4)$$
$$S \circ \rho(1,3) = (3\ 4\ 5\ \underline{3\ 5\ 2}\ 1\ 4)$$
$$S \circ \rho(1,3) \circ \rho(4,6) = (3\ 4\ 5\ 2\ 5\ 3\ 1\ 4)$$

No Algoritmo 6, temos um pseudocódigo desta heurística.

Algoritmo 6: Extremidades

Entrada: Strings S e P
Saída: Distância entre S e P

```
1 início
2     (S', P') ← Processamento(S, P)
3     se |occ(S')| = 1 então
4         retorna distância_entre_permutacões(S', P')
5     R ← operacões_com_melhor_rank(S', P')
6     S'' ← aplica_operacões(R, S')
7     retorna |R| + Extremidades(S'', P')
```

A.6 Resultados Obtidos

Para comparar os resultados das heurísticas, criamos uma base de dados composta por 19 conjuntos com 1000 pares de strings (origem e destino). Cada um dos primeiros 10 conjuntos possui strings de um mesmo tamanho e os tamanhos variam de 100 até 1000 em intervalos de 100. Todas as strings desses conjuntos possuem 25% de caracteres duplicados, ou seja, para uma string S temos $|dup(S)| = \frac{|S|}{4}$. Os 9 conjuntos restantes são compostos por strings com tamanho 500. A porcentagem de caracteres duplicados nestes conjuntos variam de 10% até 50% em intervalos de 5%.

Cada par de strings é criado da seguinte forma. Geramos uma permutação aleatória da sequência $(1, 2, \ldots, |S| - |dup(S)|, 1, 2, \ldots, |dup(S)|)$ para a string origem S e aplicamos $\frac{|S|}{4}$ reversões aleatórias (com índices i e j escolhidos aleatoriamente) em S para obter a string destino P.

Utilizando esta base de dados, realizamos os testes das heurísticas. Para as heurísticas que utilizam mapeamentos fixamos o parâmetro $r = 10|S|$. Assim, estas heurísticas produzem o mesmo número de mapeamentos. Realizamos alguns testes preliminares para ajustar os demais parâmetros. Nestes testes, utilizamos apenas os conjuntos com 25% de caracteres duplicados e com strings de tamanho 400, 500 e 600. No caso com sinais, como a variação dos resultados foi menor, adicionamos o conjunto com strings de tamanho 700. Para o caso sem sinais utilizamos apenas 100 pares de strings por conjunto e para o caso com sinais utilizamos apenas 200 pares. A seguir descrevemos as combinações de parâmetros que foram testadas e os valores escolhidos para cada heurística.

- Busca Local: testamos todas as combinações dos paramentos $\ell \in \{10, 20..., 100\}$ e $c \in \{10, 20, ..., 100\}$. Para o caso com sinais os valores escolhidos foram $\ell = 40$ e $c = 40$. No caso sem sinais os valores foram $\ell = 30$ e $c = 90$.
- GRASP: inicialmente fixamos $c = 100$ e $k = 100$ e testamos todas as combinações dos paramentos $r_\ell \in \{10, 20, ..., 100\}$ e $g \in \{1, 2, ..., 10\}$. Em seguida, com os melhores valores encontrados para r_ℓ e g, testamos todas as combinações dos paramentos $k \in \{10, 20..., 250\}$ e $c \in \{k, k + 10, ..., 250\}$. Para ambos os casos, com e sem sinais, os valores escolhidos foram $k = 150$, $c = 200$, $r_\ell = 30$ e $g = 3$.

- Algoritmos Genéticos: inicialmente fixamos $c = 100$ e $k = 50$ e testamos todas as combinações dos paramentos $t_c \in \{\lfloor 10\%|dup(S)|\rfloor, \lfloor 20\%|dup(S)|\rfloor, ..., |dup(S)|\}$ e $t_m \in \{1, 2, ..., 10\}$. Note que o parâmetro t_c varia de acordo com o número de caracteres duplicados na string origem S. Em seguida, com os melhores valores encontrados para t_c e t_m, testamos todas as combinações dos paramentos $k \in \{10, 20, ..., 100\}$ e $c \in \{k, k + 10, ..., 100\}$. Para o caso com sinais os valores escolhidos foram $k = 10$, $c = 10$, $t_c = \lfloor 40\%|dup(S)|\rfloor$ e $t_m = 1$. No caso sem sinais os valores escolhidos foram $k = 50$, $c = 90$, $t_c = \lfloor 40\%|dup(S)|\rfloor$ e $t_m = 2$.

As tabelas 2, 3, 4 e 5 apresentam os resultados dos testes realizados. Para efeito de comparação, adicionamos uma coluna com o resultado do algoritmo SOAR e a coluna OP com o número de operações de reversão aplicadas para gerar a string destino. Também adicionamos uma linha com a média dos estimadores dos erros das distâncias (EED_{med}). Para um par de strings S e P e um algoritmo A, o estimador do erro da distância é dado por $\frac{|D_A - OP|}{OP}$, onde D_A é a estimativa da distância entre S e P obtida pelo algoritmo A. Note que este estimador indica, em porcentagem, a distância entre a estimativa obtida e o número de operações usadas para gerar a string destino.

Tabela 2: Média das distância obtidas pelas nossas heurísticas e pelo algoritmo SOAR para o problema DR. Variando o tamanho das strings e fixando a porcentagem de caracteres duplicados em 25%.

Tamanho da String	**Ext**	**GRASP**	**AG**	**BL**	**MA**	**SOAR**	**OP**
100	40.263	24.221	24.330	24.221	32.622	29.649	25
200	88.928	49.609	49.754	49.609	75.310	61.423	50
300	137.788	75.370	75.602	75.388	120.937	94.158	75
400	187.646	100.902	101.300	100.989	167.302	126.339	100
500	237.501	126.564	127.132	126.789	214.613	158.971	125
600	288.311	152.050	152.907	152.743	262.344	191.212	150
700	338.460	177.967	179.171	179.640	310.991	224.106	175
800	388.726	203.833	205.327	207.730	359.602	256.414	200
900	438.124	229.581	231.809	238.634	408.307	289.041	225
1000	488.634	255.768	258.538	274.032	457.218	321.878	250
$\mathbf{EED_{med}(\%)}$	87.043	2.402	2.878	3.872	67.812	26.195	-

Como podemos ver na Tabela 2, utilizando mapeamentos aleatórios, obtemos médias de distâncias superiores as do SOAR. Conforme o tamanho das strings aumenta, os resultados da heurística MA se afastam dos do SOAR, mas se mantém melhores do que os resultados da heurística EXT. Neste caso, a estratégia de utilizar mapeamentos é superior a de posicionar os elementos nas extremidades da string.

Nas heurísticas GRASP, AG e BL, em que utilizamos estratégias adicionais para a escolha dos mapeamentos, temos resultados melhores que os do SOAR em todos os casos. Além disso, as distâncias encontradas são muito próximas do número de operações aplicadas. Entre estas heurísticas, a heurística GRASP apresentou os melhores resultados para todos os tamanhos. A heurística BL apresentou médias das distâncias inferiores as da heurística AG nas strings de tamanhos 100 até 600, mas apresentou médias das distâncias superiores para os tamanhos 700 até 1000.

Tabela 3: Média das distância obtidas pelas nossas heurísticas e pelo algoritmo SOAR para o problema $D\bar{R}$. Variando o tamanho das strings e fixando a porcentagem de caracteres duplicados em 25%.

Tamanho da String	**Ext**	**GRASP**	**AG**	**BL**	**MA**	**SOAR**	**OP**
100	46.510	24.713	24.713	24.713	33.887	24.729	25
200	107.418	49.735	49.735	49.735	75.538	49.745	50
300	173.157	74.742	74.744	74.742	119.025	74.757	75
400	237.756	99.712	99.712	99.712	163.337	99.726	100
500	307.454	124.732	124.732	124.733	208.529	124.741	125
600	373.993	149.720	149.722	149.730	254.094	149.740	150
700	441.690	174.742	174.742	174.758	299.666	174.770	175
800	517.942	199.737	199.737	199.786	345.839	199.748	200
900	589.635	224.721	224.721	224.786	392.036	224.734	225
1000	655.677	249.731	249.731	249.841	438.758	249.740	250
$\mathbf{EED_{med}(\%)}$	140.050	0.322	0.323	0.332	63.878	0.334	-

Observamos na Tabela 3 que as relações entre as heurísticas são semelhantes para o caso com sinais. Neste caso, as heurísticas GRASP, AG, BL e o algoritmo SOAR apresentam praticamente os mesmos resultados.

Tabela 4: Média das distância obtidas pelas nossas heurísticas e pelo algoritmo SOAR para o problema DR. Variando a porcentagem de caracteres duplicados e fixando o tamanho das strings em 500.

Porcentagem de Caracteres Duplicados	**Ext**	**GRASP**	**AG**	**BL**	**MA**	**SOAR**	**OP**
10%	226.814	126.719	126.795	126.722	152.601	159.023	125
15%	231.217	126.619	126.793	126.650	172.652	158.640	125
20%	235.371	126.486	126.784	126.610	193.402	159.100	125
25%	237.777	126.539	127.052	126.737	214.370	158.622	125
30%	240.985	126.546	128.112	127.136	235.559	158.621	125
35%	243.008	126.855	131.490	128.233	256.117	158.987	125
40%	247.298	127.535	138.476	130.624	275.992	159.089	125
45%	249.715	129.276	147.987	136.610	295.152	158.916	125
50%	252.684	133.305	158.778	149.244	313.842	159.083	125
$\mathbf{EED_{med}(\%)}$	83.190	2.373	7.195	4.579	78.775	24.406	-

Na Tabela 4, vemos que a heurística Ext é melhor que a heurística MA para strings com mais de 30% de caracteres duplicados. Os resultados de todas as heurísticas pioram conforme o número de caracteres duplicados aumentam, mas as heurísticas GRASP, AG e BL têm uma variação menor. Estas três heurísticas continuam superiores ao SOAR.

Tabela 5: Média das distância obtidas pelas nossas heurísticas e pelo algoritmo SOAR para o problema $D\bar{R}$. Variando a porcentagem de caracteres duplicados e fixando o tamanho das strings em 500.

Porcentagem de Caracteres Duplicados	**Ext**	**GRASP**	**AG**	**BL**	**MA**	**SOAR**	**OP**
10%	311.776	124.731	124.731	124.731	149.082	124.732	125
15%	311.453	124.714	124.714	124.714	168.042	124.718	125
20%	306.794	124.723	124.723	124.723	188.064	124.732	125
25%	306.136	124.770	124.770	124.772	208.490	124.777	125
30%	307.132	124.725	124.727	124.770	229.110	124.733	125
35%	306.883	124.740	124.739	124.927	250.342	124.753	125
40%	306.229	124.725	124.735	125.371	271.571	124.742	125
45%	307.330	124.753	124.810	126.253	293.122	124.765	125
50%	310.325	124.818	124.908	128.189	313.862	124.766	125
$\mathbf{EED_{med}(\%)}$	131.925	0.207	0.216	0.593	75.734	0.207	-

Na Tabela 5, observamos que a heurística Ext supera a heurística MA apenas com 50% de caracteres duplicados. A heurística MA tem uma piora considerável nos resultados com o aumento dos caracteres duplicados. Além disso, notamos que as heurísticas GRASP, AG e o algoritmo SOAR quase não são afetados pelo aumento do número de caracteres duplicados. A heurística BL, por outro lado, apresentou uma piora nos resultados com o aumento dos caracteres duplicados.