



Universidade Estadual de Campinas
Instituto de Computação



Gabriel Henrique Siqueira

Heurísticas para Problemas de Rearranjo de Genomas com Genes Multiplicados

CAMPINAS

2021

Gabriel Henriques Siqueira

**Heurísticas para Problemas de Rearranjo de Genomas
com Genes Multiplicados**

Dissertação apresentada ao Instituto de
Computação da Universidade Estadual de
Campinas como parte dos requisitos para a
obtenção do título de Mestre em Ciência da
Computação.

Orientador: Prof. Dr. Zanoni Dias

Coorientador: Dr. André Rodrigues Oliveira

Este exemplar corresponde à versão final da
Dissertação defendida por Gabriel Henriques
Siqueira e orientada pelo Prof. Dr. Zanoni
Dias.

CAMPINAS

2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

Si75h Siqueira, Gabriel Henriques, 1996-
Heurísticas para problemas de rearranjo de genomas com genes multiplicados / Gabriel Henriques Siqueira. – Campinas, SP : [s.n.], 2021.

Orientador: Zanoni Dias.
Coorientador: André Rodrigues Oliveira.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Biologia computacional. 2. Rearranjo de genomas. 3. Heurística (Computação). I. Dias, Zanoni, 1975-. II. Oliveira, André Rodrigues, 1990-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Heuristics for genome rearrangement problems with replicated genes

Palavras-chave em inglês:

Computational biology

Genome rearrangements

Heuristic (Computer science)

Área de concentração: Ciência da Computação

Titulação: Mestre em Ciência da Computação

Banca examinadora:

Zanoni Dias [Orientador]

Kelly Cristina Poldi

Rafael Crivellari Saliba Schouery

Data de defesa: 12-03-2021

Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)

- ORCID do autor: <https://orcid.org/0000-0001-5745-399X>

- Currículo Lattes do autor: <http://lattes.cnpq.br/3995613490047556>



Universidade Estadual de Campinas
Instituto de Computação



Gabriel Henriques Siqueira

Heurísticas para Problemas de Rearranjo de Genomas com Genes Multiplicados

Banca Examinadora:

- Prof. Dr. Zandoni Dias
Instituto de Computação – Universidade Estadual de Campinas
- Profa. Dra. Kelly Cristina Poldi
Instituto de Matemática, Estatística e Computação Científica – Universidade Estadual de Campinas
- Prof. Dr. Rafael Crivellari Saliba Schouery
Instituto de Computação – Universidade Estadual de Campinas

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 12 de março de 2021

Agradecimentos

Inicialmente, gostaria de agradecer aos meus pais Maria Cristina e Cesar por sempre apoiarem e incentivarem meus estudos. Da mesma forma, agradeço todo o apoio da minha irmã Caroline.

Também agradeço aos meus orientadores Zanoni e André por todas as críticas, ideias e sugestões que garantiram a qualidade deste trabalho. Quando procurei o professor Zanoni para o meu projeto final de graduação, que acabou evoluindo para o Mestrado, eu já sabia que ele era um ótimo professor, mas não imaginava a importância que ele teria como orientador na minha evolução durante os últimos dois anos.

Além dos meus orientadores, agradeço ao Alexsandro, Klairton e Ulisses pelo apoio, pelas colaborações e por todas as discussões muito produtivas durante nossas reuniões semanais.

Adicionalmente, agradeço aos professores e demais funcionários do IC que garantiram um ambiente ideal para a obtenção de conhecimento e desenvolvimento de pesquisa.

Por fim, agradeço o auxílio financeiro do CNPq e do Bradesco durante este período. O presente trabalho foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq) - 130624/2019-5.

Resumo

Problemas de rearranjo de genomas buscam estimar a distância evolutiva entre genomas de diferentes espécies. Esses problemas lidam com eventos de rearranjo, mutações capazes de alterar a sequência genética dos genomas. Diferentes problemas de rearranjo podem ser definidos de acordo com os eventos utilizados e as características dos genomas considerados.

O objetivo deste trabalho é o desenvolvimento de heurísticas para resolver problemas de rearranjo de genomas considerando genes multiplicados. A maioria dos trabalhos nessa área ignora a presença de múltiplas cópias de um gene, por esse ser um fator que complica a derivação de algoritmos eficientes. Dentro desse contexto, utilizamos como foco de estudo os eventos de reversão e transposição, onde o primeiro se caracteriza por inverter a ordem de uma sequência de genes, e o segundo por trocar a posição de duas sequências adjacentes de genes.

Os trabalhos iniciais sobre rearranjos de genomas consideravam que cada gene possui apenas uma cópia e representavam os genomas como permutações. Os trabalhos mais recentes consideram a repetição de genes como um fator relevante e representam genomas como strings. Nesta dissertação, adotamos a representação por strings. Como são conhecidos alguns resultados para problemas de rearranjos de genomas com características semelhantes às que propomos, utilizamos esses resultados para avaliar a qualidade das heurísticas desenvolvidas. Desenvolvemos heurísticas baseadas em técnicas normalmente adotadas para problemas de otimização combinatória, tais como Busca Local, Algoritmos Genéticos e GRASP. Nos testes realizados, verificamos que as distâncias obtidas pelas heurísticas propostas são menores que as distâncias obtidas por algoritmos da literatura.

Abstract

Genome rearrangement problems seek to estimate the evolutionary distance between genomes of different species. These problems deal with rearrangement events, which are mutations capable of altering the genetic sequence of genomes. Different rearrangement problems may be defined by the selected events and the characteristics of the considered genomes.

Our goal is to develop heuristics for solving genome rearrangement problems involving replicated genes. Most research in this area disregard multiple copies of a gene, since that characteristic may complicate the derivation of efficient algorithms that obtain high-quality solutions. In this context, we focus mainly on reversal and transposition events. The former corresponds to the inversion of a sequence of genes, and the latter corresponds to the exchange of two adjacent sequences of genes.

Initial research involving genome rearrangements assumed that each gene has a single copy and represented the genomes as permutation. More recent research considers the presence of gene copies as a relevant factor and represent genomes as strings. In this thesis we adopt the string representation. We evaluate the proposed heuristics using results from the literature for genome rearrangement problems with similar characteristics. We develop heuristics with common techniques used in combinatorial optimization problems, such as Local Search, Genetic Algorithms and GRASP. The experimental tests showed that the distances obtain with the proposed heuristics are inferior to the distances obtain with algorithms from the literature.

Lista de Figuras

3.1	Exemplo da heurística Mapeamentos Aleatórios	26
3.2	Exemplo da heurística Busca Local	28
3.3	Exemplo da heurística GRASP	31
3.4	Exemplo da heurística Algoritmo Genético	34
3.5	Exemplo da heurística Busca Tabu	36
3.6	Exemplo da heurística <i>Simulated Annealing</i>	39
3.7	Exemplo da heurística Busca Cuco	41
3.8	Exemplo da heurística Separação	44
3.9	Resultado da heurística MA com diferentes números de mapeamentos . . .	49
3.10	Ajuste dos parâmetros da heurística de Busca Local no problema <i>DT</i> . . .	50
3.11	Ajuste dos parâmetros da heurística GRASP no problema <i>DT</i>	51
3.12	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema <i>DT</i>	52
3.13	Ajuste dos parâmetros da heurística de Busca Tabu no problema <i>DT</i> . . .	53
3.14	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema <i>DT</i>	54
3.15	Ajuste dos parâmetros da heurística de Busca Cuco no problema <i>DT</i> . . .	55
5.1	Etapas do algoritmo para estimar a distância de reversão.	104
A.1	Ajuste dos parâmetros da heurística de Busca Local no problema <i>DR</i> . . .	111
A.2	Ajuste dos parâmetros da heurística GRASP no problema <i>DR</i>	112
A.3	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema <i>DR</i>	112
A.4	Ajuste dos parâmetros da heurística de Busca Tabu no problema <i>DR</i> . . .	113
A.5	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema <i>DR</i>	113
A.6	Ajuste dos parâmetros da heurística de Busca Cuco no problema <i>DR</i> . . .	114

A.7	Ajuste dos parâmetros da heurística de Busca Local no problema $D\bar{R}$	114
A.8	Ajuste dos parâmetros da heurística GRASP no problema $D\bar{R}$	115
A.9	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema $D\bar{R}$	115
A.10	Ajuste dos parâmetros da heurística de Busca Tabu no problema $D\bar{R}$	116
A.11	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema $D\bar{R}$	116
A.12	Ajuste dos parâmetros da heurística de Busca Cuco no problema $D\bar{R}$	117
A.13	Ajuste dos parâmetros da heurística de Busca Local no problema DT	117
A.14	Ajuste dos parâmetros da heurística GRASP no problema DT	118
A.15	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema DT	118
A.16	Ajuste dos parâmetros da heurística de Busca Tabu no problema DT	119
A.17	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema DT	119
A.18	Ajuste dos parâmetros da heurística de Busca Cuco no problema DT	120
A.19	Ajuste dos parâmetros da heurística de Busca Local no problema DRT	120
A.20	Ajuste dos parâmetros da heurística GRASP no problema DRT	121
A.21	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema DRT	121
A.22	Ajuste dos parâmetros da heurística de Busca Tabu no problema DRT	122
A.23	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema DRT	122
A.24	Ajuste dos parâmetros da heurística de Busca Cuco no problema DRT	123
A.25	Ajuste dos parâmetros da heurística de Busca Local no problema $D\bar{R}T$	123
A.26	Ajuste dos parâmetros da heurística GRASP no problema $D\bar{R}T$	124
A.27	Ajuste dos parâmetros da heurística de Algoritmo Genético no problema $D\bar{R}T$	124
A.28	Ajuste dos parâmetros da heurística de Busca Tabu no problema $D\bar{R}T$	125
A.29	Ajuste dos parâmetros da heurística de <i>Simulated Annealing</i> no problema $D\bar{R}T$	125
A.30	Ajuste dos parâmetros da heurística de Busca Cuco no problema $D\bar{R}T$	126

Lista de Tabelas

3.1	Comparação de algoritmos para o problema OR	47
3.2	Comparação de algoritmos para o problema OT	48
3.3	Comparação de algoritmos para o problema ORT	48
3.4	Comparação de algoritmos para o problema $\bar{O}RT$	48
3.5	Médias das distâncias obtidas para o problema DR com a base REV . . .	56
3.6	Médias das distâncias para o problema $\bar{D}R$ com a base SREV	56
3.7	Médias das distâncias para o problema DT com a base TRANS	57
3.8	Médias das distâncias para o problema DRT com a base REVTRANS . . .	58
3.9	Médias das distâncias para o problema $\bar{D}RT$ com a base SREVTRANS . .	59
3.10	Médias das distâncias para o problema DR com a base HARD	59
3.11	Médias das distâncias para o problema $\bar{D}R$ com a base SHARD	60
3.12	Médias das distâncias para o problema DT com a base HARD	61
3.13	Médias das distâncias para o problema DRT com a base HARD	61
3.14	Médias das distâncias para o problema $\bar{D}RT$ com a base SHARD	62
3.15	Tempo para resolver as instâncias de tamanho 100	62
3.16	Tempo para resolver as instâncias de tamanho 500	63
3.17	Tempo para resolver as instâncias de tamanho 1000	63
4.1	Resultados para o problema MCSP com a base T-O	93
4.2	Resultados para o problema SMCSPP com a base SR-O	94
4.3	Resultados para o problema RMCSP com a base R-O	94
4.4	Médias das distâncias para o problema DR com as strings da base RAND .	96
4.5	Médias das distâncias para o problema DR com as strings da base R-O . .	96
4.6	Médias das distâncias para o problema DR com as strings da base R-L . .	96
4.7	Médias das distâncias para o problema $\bar{D}R$ com as strings da base SRAND .	97
4.8	Médias das distâncias para o problema $\bar{D}R$ com as strings da base SR-O .	97

4.9	Médias das distâncias para o problema $D\bar{R}$ com as strings da base SR-L	. . 98
4.10	Médias das distâncias para o problema DT com as strings da base RAND	. 99
4.11	Médias das distâncias para o problema DT com as strings da base T-O	. . 99
4.12	Médias das distâncias para o problema DT com as strings da base T-L	. . 100
4.13	Médias das distâncias para o problema DRT com as strings da base RAND	100
4.14	Médias das distâncias para o problema DRT com as strings da base RT-O	101
4.15	Médias das distâncias para o problema DRT com as strings da base RT-L	101
4.16	Médias das distâncias para o problema $D\bar{R}T$ com as strings da base SRAND	102
4.17	Médias das distâncias para o problema $D\bar{R}T$ com as strings da base SRT-O	102
4.18	Médias das distâncias para o problema $D\bar{R}T$ com as strings da base SRT-L	103

Sumário

1	Introdução	14
2	Fundamentação Teórica	17
2.1	Representação de Genomas por Strings	17
2.2	Eventos de Rearranjo	18
2.3	Problemas de Rearranjo de Genomas	19
3	Heurísticas para Genes Duplicados	22
3.1	Mapeamento de Strings em Permutações	22
3.2	Mapeamentos Aleatórios (MA)	25
3.3	Busca Local (BL)	26
3.4	GRASP	27
3.5	Algoritmo Genético (AG)	30
3.6	Busca Tabu (BT)	35
3.7	<i>Simulated Annealing</i> (SA)	37
3.8	Busca Cuco (BC)	40
3.9	Separação (Sep)	42
3.10	Experimentos Práticos	44
3.10.1	Bases de Dados	44
3.10.2	Algoritmos para Permutações	47
3.10.3	Ajuste de Parâmetros	49
3.10.4	Resultados	55
4	Heurísticas para Genes Multiplicados	64
4.1	Mapeamento de Strings em Permutações	64
4.2	Adaptação das Heurísticas	66

4.3	Problemas de Partição de Strings Comuns Mínima	69
4.3.1	Partições de Permutações	74
4.3.2	Relação entre Partições e Distâncias de Rearranjo	76
4.3.3	Algoritmos da Literatura	80
4.3.4	Heurística de Combinação	86
4.4	O Problema de Empacotamento Máximo de Ciclos	88
4.5	Experimentos Práticos	90
4.5.1	Bases de Dados	90
4.5.2	Algoritmos para Partição	92
4.5.3	Resultados	94
5	Conclusão	104
A	Resultados Complementares para o Caso de Genes Duplicados	111
A.1	Problema DR	111
A.2	Problema $D\bar{R}$	114
A.3	Problema DT	117
A.4	Problema DRT	120
A.5	Problema $D\bar{R}T$	123

Capítulo 1

Introdução

A comparação genômica é uma área de estudo no campo da Biologia Computacional, onde o objetivo é a busca por similaridades e parentescos entre espécies utilizando informações genômicas. Parte dessa busca é realizada ao estimar a distância evolutiva entre genomas. Essa estimativa pode ser feita por meio da avaliação do número de mutações que afetaram o genoma de uma espécie ao longo da evolução, o que gera novas espécies com genomas distintos.

Para calcular essa estimativa, consideramos que o genoma consiste em uma sequência de genes, sendo que esses genes podem apresentar repetições, e suas orientações podem ser conhecidas. As mutações podem alterar a quantidade de genes, como a inserção, a remoção e a duplicação [46, 58, 61], ou alterar apenas a ordem ou a orientação dos genes, como a reversão, a transposição e a transreversão [5, 6, 45]. Existem também mutações que consideram múltiplos cromossomos, como a fissão, a fusão e a translocação [40].

Chamamos de *eventos de rearranjo conservativos* as mutações que podem afetar apenas a ordem e orientação dos genes. A *distância de rearranjo* entre dois genomas é definida pelo tamanho da menor sequência de eventos de rearranjo capaz de transformar um genoma em outro. Existem diferentes formas para representar os genomas, dependendo das características do problema de rearranjo abordado [30]. Quando consideramos que os genomas apresentam repetições de genes, representamos os genomas por strings, onde cada caractere corresponde a um gene ou a um bloco conservado de genes durante o processo evolutivo.

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

Um *modelo de rearranjo* \mathcal{M} representa o conjunto dos eventos de rearranjo permitidos para transformar um genoma em outro. Modelos podem conter apenas um ou múltiplos eventos de rearranjo [14, 56]. Um problema de rearranjo é então definido por um modelo de rearranjo e pela forma em que representamos os genomas.

Alguns modelos frequentemente utilizados em trabalhos de rearranjo de genomas são compostos pelos eventos de reversão [7, 8, 17, 35] e transposição [6, 15]. Existem trabalhos considerando um modelo composto por ambos os eventos [51, 56].

Os trabalhos iniciais sobre rearranjos de genomas envolviam genomas sem repetições de genes, ou seja, eles tratavam dos problemas de ordenação de permutações. Mesmo nesse contexto restrito, poucos problemas têm algoritmos exatos polinomiais conhecidos [35, 59]. De fato, foi provado que alguns desses problemas pertencem à classe de problemas NP-Difícil [17, 26, 48], o que nos fornece um indício de que pode não ser possível encontrar algoritmos eficientes para tais problemas.

Apesar dessa dificuldade em encontrar algoritmos exatos e eficientes, vários algoritmos com fatores de aproximação constantes foram propostos [8, 20, 36, 39, 51, 56]. Outra abordagem utilizada foi a busca por heurísticas e metaheurísticas para esses problemas que, embora não obtenham sempre uma solução exata, tendem a obter bons resultados na prática [3, 13, 25, 62].

Os trabalhos mais recentes passaram a considerar que os genes podem apresentar múltiplas cópias, ou seja, passaram a representar os genomas como strings. A principal abordagem com relação a aproximações envolve a relação entre os problemas de rearranjo e as variações do problema de Partição de Strings Comuns Mínima [19, 42, 53]. Já com relação à heurísticas, considerando os eventos abordados neste trabalho, encontramos apenas uma heurísticas baseada em Programação Linear Binária para o problema envolvendo o evento de reversão [55].

Existem duas variações dos problemas de rearranjo que dependem do conhecimento da orientação dos genes. Quando a orientação dos genes é conhecida temos um sinal associado a cada caractere das strings, portanto os problemas lidam com *strings com sinais*. Caso contrário, nenhum sinal é utilizado e os problemas lidam com *strings sem sinais*.

Esta dissertação tem como objetivo desenvolver e avaliar heurísticas para os problemas de rearranjo envolvendo os eventos de reversão e transposição. Adaptamos metaheurísticas clássicas da literatura com o intuito de obter bons resultados práticos. Consideramos os seguintes problemas de rearranjo envolvendo genomas com genes multiplicados:

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

A implementação das heurísticas desenvolvidas e de outros algoritmos tratados nessa dissertação está disponível em um repositório público¹.

¹<https://github.com/compbiogroup/Heuristics-for-Genome-Rearrangement-with-Replicated-Genes>

O restante deste documento segue a seguinte estrutura. No Capítulo 2, definimos formalmente os problemas tratados e realizamos uma revisão da literatura. No Capítulo 3, descrevemos as heurísticas propostas considerando um caso restrito, onde cada gene tem no máximo duas cópias. No Capítulo 4, generalizamos as heurísticas para considerar genes com múltiplas cópias e exploramos a relação dos problemas de rearranjo com as variações do problema de Partição de Strings Comuns Mínima para melhorar a qualidade das soluções fornecidas pelas heurísticas propostas. Por fim, encerramos com algumas considerações finais no Capítulo 5.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta alguns conceitos importantes para os problemas de rearranjo de genomas. Na Seção 2.1, descrevemos como representamos genomas por meio de strings e na Seção 2.2 formalizamos os eventos de rearranjo dos problemas estudados. Por fim, na Seção 2.3, apresentamos uma revisão da literatura sobre 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 representa um bloco conservado de genes. No caso em que a orientação dos genes é conhecida, um sinal (+ ou −) é associado a cada caractere, representando a orientação de cada bloco de genes dentro do genoma. Nesse caso, temos uma *string com sinais*, caso contrário, temos uma *string sem sinais*.

Dada uma string S com ou sem sinais, denotamos por $|S|$ o número de caracteres de S e por S_i o i -ésimo caractere de S . Além disso, o conjunto de caracteres distintos de S é chamado de *alfabeto* de S , denotado por Σ_S . Para evitar ambiguidade, utilizamos o termo *rótulo* quando falamos dos elementos de Σ_S e o termo *caractere* quando falamos dos elementos de S . Em strings com sinais, os caracteres incluem os sinais e os rótulos não, ou seja, os caracteres $-\alpha$ e $+\alpha$ são ambos correspondentes ao rótulo α .

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

$$\begin{aligned} S &= (E \ B \ A \ C \ D \ E \ E \ D) \\ S_1 &= E \text{ e } S_5 = D \\ P &= (+E \ -B \ -A \ +C \ +D \ +E \ +E \ -D) \\ P_4 &= +C \text{ e } P_8 = -D \\ \Sigma_S &= \Sigma_P = \{A, B, C, D, E\}, \ |S| = |P| = 8 \end{aligned}$$

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

O conjunto dos rótulos *multiplicados* de S , denotado por $mult(S)$, é composto por rótulos que aparecem mais de uma vez em S , ou seja, $mult(S) = \{\alpha : occ(\alpha, S) > 1, \forall \alpha \in \Sigma_S\}$. Já o conjunto dos rótulos *duplicados* de S , denotado por $dup(S)$, é composto por rótulos que aparecem exatamente duas vezes em S , ou seja, $dup(S) = \{\alpha : occ(\alpha, S) = 2, \forall \alpha \in \Sigma_S\}$. Usamos $|mult(S)|$ e $|dup(S)|$ para denotar o tamanho desses conjuntos. Quando $|mult(S)| = 0$, dizemos que a string S é uma *permutação*. Nas strings do Exemplo 1, temos $mult(S) = mult(P) = \{D, E\}$ e $dup(S) = dup(P) = \{D\}$.

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

Exemplo 2. Três strings S , P e Q , tais que S e P são balanceadas e S e Q não são balanceadas (os caracteres A e E têm ocorrências diferentes em S e Q).

$$\begin{aligned} S &= (E \quad B \quad A \quad C \quad D \quad E \quad D) \\ P &= (D \quad D \quad A \quad B \quad E \quad E \quad C) \\ Q &= (E \quad A \quad A \quad C \quad D \quad D \quad B) \end{aligned}$$

Quando um modelo \mathcal{M} possui apenas eventos de rearranjo conservativos, precisamos que as strings consideradas sejam balanceadas.

2.2 Eventos de Rearranjo

Representamos um evento de rearranjo em um genoma \mathcal{G} por uma operação aplicada 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 ocorrência do evento δ em \mathcal{G} . Nesse trabalho, consideramos os eventos de reversão e transposição definidos a seguir.

Definição 3. Dada uma string sem sinais S , uma *reversão* $\rho(i, j)$ aplicada em S , onde $1 \leq i < j \leq |S|$, é um evento que inverte a ordem dos elementos do segmento de S entre as posições i e j .

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

Definição 4. Dada uma string com sinais S , uma *reversão* $\rho(i, j)$ aplicada em S , onde $1 \leq i \leq j \leq |S|$, é um evento que inverte a ordem e orientação dos elementos do segmento de S entre as posições i e j .

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

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

$$\begin{aligned} S &= (A \text{ } \color{red}{B} \text{ } \color{red}{C} \text{ } \color{red}{C} \text{ } B \text{ } A \text{ } E) \\ S \circ \rho(2, 4) &= (A \text{ } \color{red}{C} \text{ } \color{red}{C} \text{ } \color{red}{B} \text{ } B \text{ } A \text{ } E) \\ P &= (+A \text{ } \color{red}{+B} \text{ } \color{red}{+C} \text{ } \color{red}{-C} \text{ } -B \text{ } +A \text{ } -E) \\ P \circ \rho(2, 4) &= (+A \text{ } \color{red}{+C} \text{ } \color{red}{-C} \text{ } \color{red}{-B} \text{ } -B \text{ } +A \text{ } -E) \end{aligned}$$

Definição 5. Dada uma string com ou sem sinais S , uma *transposição* $\tau(i, j, k)$ aplicada em S , onde $1 \leq i < j < k \leq |S| + 1$, é um evento que troca o segmento de S entre as posições i e $j - 1$ com o segmento de S entre as posições j e $k - 1$. Esse evento não altera a ordem e a orientação dos elementos em cada segmento.

$$\begin{aligned} S &= (S_1 \dots S_{i-1} \color{red}{S_i} \dots \color{red}{S_{j-1}} \color{blue}{S_j} \dots \color{blue}{S_{k-1}} S_k \dots S_{|S|}) \\ S \circ \tau(i, j, k) &= (S_1 \dots S_{i-1} \color{blue}{S_j} \dots \color{blue}{S_{k-1}} \color{red}{S_i} \dots \color{red}{S_{j-1}} S_k \dots S_{|S|}) \end{aligned}$$

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

$$\begin{aligned} S &= (A \text{ } \color{red}{B} \text{ } \color{red}{C} \text{ } \color{red}{C} \text{ } \color{blue}{B} \text{ } \color{blue}{A} \text{ } D) \\ S \circ \tau(2, 5, 7) &= (A \text{ } \color{blue}{B} \text{ } \color{blue}{A} \text{ } \color{red}{B} \text{ } \color{red}{C} \text{ } \color{red}{C} \text{ } D) \\ P &= (+A \text{ } \color{red}{+B} \text{ } \color{red}{+C} \text{ } \color{red}{-C} \text{ } -B \text{ } \color{blue}{+A} \text{ } -D) \\ P \circ \tau(2, 5, 7) &= (+A \text{ } \color{blue}{-B} \text{ } \color{blue}{+A} \text{ } \color{red}{+B} \text{ } \color{red}{+C} \text{ } \color{red}{-C} \text{ } -D) \end{aligned}$$

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 o termo distância de reversão e transposição ($d_{\mathcal{RT}}(S, P)$). Quando podemos utilizar qualquer uma das distância, utilizamos apenas o termo distância e omitimos o índice da notação ($d(S, P)$).

2.3 Problemas de Rearranjo de Genomas

Os primeiros problemas de rearranjo estudados consideravam genomas com uma única cópia de cada gene. O primeiro rearranjo estudado foi o de reversão, proposto em 1982 por Watterson *et al.* [57]. Em 1995, Kececioglu e Sankoff [39] apresentaram um algoritmo com fator de aproximação 2 para o problema de Ordenação de Permutações sem Sinais por Reversão (*OR*). Em 1996, Bafna e Pevzner [5] iniciaram o estudo de reversões em permutações com sinais. Nesse trabalho, eles apresentaram o grafo de breakpoints, amplamente utilizado na obtenção de algoritmos de aproximação para problemas de rearranjo.

Em 1997, Caprara [17] provou que o problema OR pertence à classe de problemas NP-Difícil. A melhor aproximação conhecida para esse problema é de 1,375, apresentada por Berman *et al.* [8] em 2002. Em 1999, Hannenhalli e Pevzner [35] apresentaram um algoritmo polinomial para o problema de Ordenação de Permutações com Sinais por Reversão (OR). Além disso, no caso quando desejamos apenas determinar o número de operações necessárias para ordenar uma permutação com sinais, existe um algoritmo de tempo linear [4].

O evento de rearranjo de transposição começou a ser estudado em 1998 por Bafna e Pevzner [6]. Eles introduziram a estrutura de grafo de ciclos, que é similar ao grafo de breakpoints utilizado para o evento de reversão. Em 2012, Bulteau *et al.* [15] mostraram que o problema de Ordenação de Permutações por Transposição (OT) pertence à classe de problemas NP-Difícil. A melhor aproximação conhecida para esse problema é de 1,375, apresentada em 2006 por Elias e Hartman [26].

Em 1998, Walter *et al.* [56] propuseram o modelo que considera ambos os eventos de reversão e transposição. Eles mostraram algoritmos com fator de aproximação 3 para o problema de Ordenação de Permutações sem Sinais por Reversão e Transposição (ORT) e com fator de aproximação 2 para o problema de Ordenação de Permutações com Sinais por Reversão e Transposição (ORT). Para o caso sem sinais, em 2008, Rahman *et al.* [51] apresentaram um algoritmo com fator de aproximação $2k$, onde k é a aproximação do algoritmo usado para a decomposição em ciclos do grafo de breakpoints. O menor valor de k conhecido até o momento é $1,4167 + \epsilon$ [18]. Em 2019, Oliveira *et al.* [48] provaram que os problemas ORT e ORT pertencem à classe de problemas NP-Difícil.

Após os primeiros problemas da área considerarem apenas genes distintos, novos trabalhos consideraram genomas com repetição de genes, passando a representar os genomas por meio de strings ao invés de permutações. Em 2001, Christie e Irving [21] provaram que o problema de Distância de Reversão em Strings sem Sinais (DR) pertence à classe de problemas NP-Difícil, mesmo quando consideramos um alfabeto binário, ou seja, quando existem apenas dois valores possíveis para os caracteres. Em 2005, Radcliffe *et al.* [50] mostraram que os problemas de Distância de Transposição em Strings sem Sinais (DT) e Distância de Reversão em Strings com Sinais ($D\bar{R}$) também pertencem à classe de problemas NP-Difícil, mesmo com um alfabeto binário. No mesmo ano, Chen *et al.* [19] provaram que o problema $D\bar{R}$ continua na classe NP-Difícil mesmo se considerarmos apenas genes duplicados.

A principal abordagem utilizada na literatura para obter fatores de aproximação para os problemas de distância de rearranjo envolvendo genes multiplicados é aproveitar a relação desses problemas com as diversas variações do problema de Partição de Strings Comuns Mínima (“Minimum Common String Partition” – MCSP), definidas no Capítulo 4. Em 2005, Chen e coautores [19] mostraram que uma aproximação com fator ℓ para a variação com sinais do problema MCSP garante uma aproximação de fator 2ℓ para o problema $D\bar{R}$. Um argumento análogo garante a mesma relação entre outra variação do problema, chamada Partição Reversa de Strings Comuns Mínima (“Reverse Minimum Common String Partition” – RMCSP) [42], e o problema DR . Em 2007, Shapira e Storer [53] mostraram que uma aproximação com fator ℓ para o problema MCSP garante uma aproximação de fator 3ℓ para o problema DT .

O problema MCSP e suas variações pertencem à classe de problemas NP-Difícil [33] mesmo se cada rótulo tiver no máximo duas cópias, e atualmente os melhores algoritmos de aproximação para esses problemas têm fatores $O(\log n \log^* n)$ [23], onde n é o tamanho das strings, e $\Theta(k)$ [43], onde k é o número máximo de cópias de um rótulo nas strings. Se restringirmos o número de cópias, podemos obter aproximações melhores. Goldstein e coautores [33] apresentam um algoritmo com fator de aproximação 1,1037 para o caso onde $k = 2$ e outro algoritmo com fator de aproximação 4 para o caso onde $k = 3$. Alguns algoritmos gulosos com aproximações piores, mas que retornam bons resultados práticos, foram propostos para esses problemas [22, 34, 37, 42].

Os resultados mais recentes tratam apenas do problema MCSP, deixando de lado suas variações. Em 2014, Blum e coautores [10] propuseram uma heurística baseada em uma estratégia aleatória para esse problema e, em 2017, Ferdous e Rahman [29] propuseram uma metaheurística utilizando colônia de formigas. Também foram propostos algoritmos exatos e heurísticas baseadas em programação linear inteira [9, 11, 12].

Capítulo 3

Heurísticas para Genes Duplicados

Neste capítulo, descrevemos detalhadamente o funcionamento das heurísticas desenvolvidas para os problemas de rearranjo de genomas considerando genes duplicados, ou seja, com no máximo duas cópias. Essas heurísticas utilizam a ideia de mapeamento, apresentada na Seção 3.1. Da Seção 3.2 até a Seção 3.9 apresentamos as heurísticas desenvolvidas. Ao final do capítulo, na Seção 3.10, apresentamos os resultados dos experimentos práticos.

Para as heurísticas propostas, utilizamos algoritmos da literatura para calcular a distância entre permutações. Nos modelos correspondentes a problemas da classe NP-Difícil, como não é conhecida uma solução polinomial exata para distâncias entre permutações, escolhemos algoritmos que garantem um fator de aproximação para os problemas.

3.1 Mapeamento de Strings em Permutações

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

Queremos mapear a string S em uma permutação. Portanto, mantemos os rótulos não duplicados e, para cada rótulo duplicado α , mapeamos os caracteres com rótulo α em dois caracteres α' e α'' . Note que temos duas formas de realizar esse 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 correspondentes a rótulos duplicados em uma string S , utilizamos um vetor binário \mathbf{x} que indica, para cada rótulo duplicado, qual dos dois possíveis mapeamentos é realizado. Usamos $\mathbf{x}[\alpha] = 0$ 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 correspondentes a rótulos duplicados, não havendo necessidade de mudar os sinais. Dessa forma, para strings com sinais, a orientação dos caracteres permanece a mesma.

Considerando essa 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 sinais S e em uma string sem sinais P .

$$\begin{aligned} S &= (+E +B -A -C +D -E -D), \quad dup(S) = \{D, E\} \\ S^{\mathbf{x}} &= (+E'' +B -A -C +D' -E' -D'') \\ P &= (E \ B \ A \ C \ D \ E \ D), \quad dup(P) = \{D, E\} \\ P^{\mathbf{x}} &= (E'' \ B \ A \ C \ D' \ E' \ D'') \\ \mathbf{x} &= \begin{array}{cc} D & E \\ \hline 0 & 1 \end{array} \end{aligned}$$

Definição 6. Sejam \mathbf{x} e \mathbf{w} dois mapeamentos para uma string S . Dizemos que \mathbf{x} e \mathbf{w} são *vizinhos* se eles diferem apenas para um rótulo duplicado. Ou seja, existe um rótulo $\alpha \in dup(S)$, tal que $\mathbf{x}[\alpha] \neq \mathbf{w}[\alpha]$ e para todo $\beta \neq \alpha$, temos que $\mathbf{x}[\beta] = \mathbf{w}[\beta]$.

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

$$\begin{aligned} S &= (+E +B -A -C +D -E -D), \quad dup(S) = \{D, E\} \\ S^{\mathbf{x}} &= (+E'' +B -A -C +D' -E' -D'') \\ S^{\mathbf{w}} &= (+E' +B -A -C +D' -E'' -D'') \\ S^{\mathbf{z}} &= (+E' +B -A -C +D'' -E'' -D') \\ \mathbf{x} &= \begin{array}{cc} D & E \\ \hline 0 & 1 \end{array} \quad \mathbf{w} = \begin{array}{cc} D & E \\ \hline 0 & 0 \end{array} \quad \mathbf{z} = \begin{array}{cc} D & E \\ \hline 1 & 0 \end{array} \end{aligned}$$

O lema a seguir nos permite usar mapeamentos para encontrar a distância de rearranjo entre duas strings S e P .

Lema 1. *Seja um modelo de rearranjo \mathcal{M} e duas strings S e P . Para dois mapeamentos \mathbf{x} e \mathbf{y} , temos $d_{\mathcal{M}}(S, P) \leq d_{\mathcal{M}}(S^{\mathbf{x}}, P^{\mathbf{y}})$.*

Demonstração. Se podemos obter $P^{\mathbf{y}}$ aplicando $d_{\mathcal{M}}(S^{\mathbf{x}}, P^{\mathbf{y}})$ eventos de rearranjos em $S^{\mathbf{x}}$, a mesma sequência de eventos nos permite transformar S em P . \square

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

$$\begin{array}{lcl}
 S^{\mathbf{x}} & = & (\underbrace{E'' \ B \ A}_{\rho(2,3)} \ C \ D' \ E' \ D'') \\
 & & (\ E'' \ A \ B \ \underbrace{C \ D' \ E' \ D''}_{\rho(4,7)}) \\
 & & (\ E'' \ A \ \underbrace{B \ D'' \ E'}_{\rho(3,5)} \ D' \ C) \\
 P^{\mathbf{y}} & = & (\ E'' \ A \ E' \ D'' \ B \ D' \ C) \\
 \hline
 S & = & (\underbrace{E \ B \ A}_{\rho(2,3)} \ C \ D \ E \ D) \\
 & & (\ E \ A \ B \ \underbrace{C \ D \ E \ D}_{\rho(4,7)}) \\
 & & (\ E \ A \ \underbrace{B \ D \ E}_{\rho(3,5)} \ D \ C) \\
 P & = & (\ E \ A \ E \ D \ B \ D \ C)
 \end{array}$$

$$\mathbf{x} = \begin{array}{|c|c|} \hline D & E \\ \hline 0 & 1 \\ \hline \end{array} \quad \mathbf{y} = \begin{array}{|c|c|} \hline D & E \\ \hline 1 & 1 \\ \hline \end{array}$$

Nas heurísticas que desenvolvemos, dada a string de origem S e a string de destino P , mapeamos P em uma permutação $P^{\mathbf{y}}$ utilizando um mapeamento arbitrário \mathbf{y} (o mapeamento correspondente ao vetor binário composto apenas por zeros) e é gerado um conjunto \mathbf{M} de mapeamentos da string S em permutações. Note que, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, temos um limitante superior para a distância entre S e P , ou seja, $d(S, P) \leq d(S^{\mathbf{x}}, P^{\mathbf{y}})$. O resultado dessas heurísticas é a menor distância entre $P^{\mathbf{y}}$ e $S^{\mathbf{x}}$, para todo mapeamento $\mathbf{x} \in \mathbf{M}$.

É importante ressaltar que não há necessidade de testar múltiplos mapeamentos para a string de destino P , conforme demonstrado no seguinte lema.

Lema 2. Dado um par de mapeamentos \mathbf{w} e \mathbf{x} para duas strings P e S , respectivamente, e um mapeamento \mathbf{y} para P , podemos obter um novo mapeamento \mathbf{x}' , tal que $d(S^{\mathbf{x}'}, P^{\mathbf{y}}) = d(S^{\mathbf{x}}, P^{\mathbf{w}})$.

Demonstração. Considere uma função f que mapeia cada elemento de $P^{\mathbf{w}}$ no elemento com a posição correspondente em $P^{\mathbf{y}}$, ou seja $f(P_i^{\mathbf{w}}) = P_i^{\mathbf{y}}$. Seja Q a string obtida ao aplicarmos f em cada elemento de $S^{\mathbf{x}}$. É possível ver que uma sequência de operações transforma Q em $P^{\mathbf{y}}$ se e somente se ela também transforma $S^{\mathbf{x}}$ em $P^{\mathbf{w}}$. Portanto, basta escolher o mapeamento \mathbf{x}' como o mapeamento que transforma S em Q . \square

Devido ao número exponencial de mapeamentos, \mathbf{M} pode não conter todos os mapeamentos possíveis para S . Dessa forma, as heurísticas utilizam diferentes estratégias para buscar mapeamentos que gerem distâncias próximas da distância ótima.

O critério de parada para essas heurísticas é o número de mapeamentos distintos obtidos. Como as heurísticas podem encontrar mapeamentos repetidos, é possível que nunca encontremos o número de mapeamentos buscado. Para evitar esse problema, utilizamos um segundo critério de parada baseado no número de iterações das heurísticas. Esse critério só é necessário quando o número de mapeamentos possíveis está próximo do número de mapeamentos gerados, o que só ocorre quando temos poucos rótulos duplicados.

3.2 Mapeamentos Aleatórios (MA)

Nesta heurística, os mapeamentos do conjunto \mathbf{M} são gerados de forma aleatória. Essa heurística serve como base para avaliarmos a qualidade das demais heurísticas, que utilizam estratégias mais sofisticadas.

Além das strings S e P , essa heurística também recebe como entrada um parâmetro $r \in \mathbb{N}$, que indica quantos mapeamentos são gerados.

Inicialmente, mapeamos P em uma permutação P^y com o mapeamento y . Em seguida, geramos mapeamentos da string S em permutações até obtermos r mapeamentos distintos. Cada bit dos vetores correspondentes a esses mapeamentos tem 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 é obtido.

Note que, embora possamos gerar mapeamentos repetidos, na prática temos um valor muito maior de mapeamentos possíveis ($2^{|dup(S)|}$) em relação ao número de mapeamentos gerados (r), garantindo que geramos mapeamentos distintos. Mais especificamente, se já temos c mapeamentos, o número esperado de iterações até gerarmos um mapeamento novo é:

$$\sum_{i=1}^{\infty} i \left(1 - \frac{c}{2^{|dup(S)|}}\right) \left(\frac{c}{2^{|dup(S)|}}\right)^{i-1} = \frac{2^{|dup(s)|}}{2^{|dup(s)|} - c}$$

Quando c é muito menor que $2^{|dup(S)|}$, esse número de iterações tende a 1.

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

O Algoritmo 1 apresenta um pseudocódigo da heurística Mapeamentos Aleatórios e a Figura 3.1 apresenta uma simulação do funcionamento da heurística utilizando as strings $S = (C \ B \ A \ B \ D \ C \ D)$ e $P = (A \ C \ D \ B \ B \ D \ C)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{x_1}, P^y) \leq 4$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 1: Mapeamentos Aleatórios

Entrada: Strings S e P e número de mapeamentos r

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $y \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $\mathbf{M} \leftarrow \emptyset$ 
4   enquanto  $|\mathbf{M}| < r$  faça
5      $x \leftarrow \text{gera\_mapeamento\_aleatório}(S)$ 
6      $\mathbf{M} \leftarrow \mathbf{M} \cup \{x\}$ 
7   retorna  $\text{menor\_distância}(\mathbf{M}, S, P^y)$ 

```

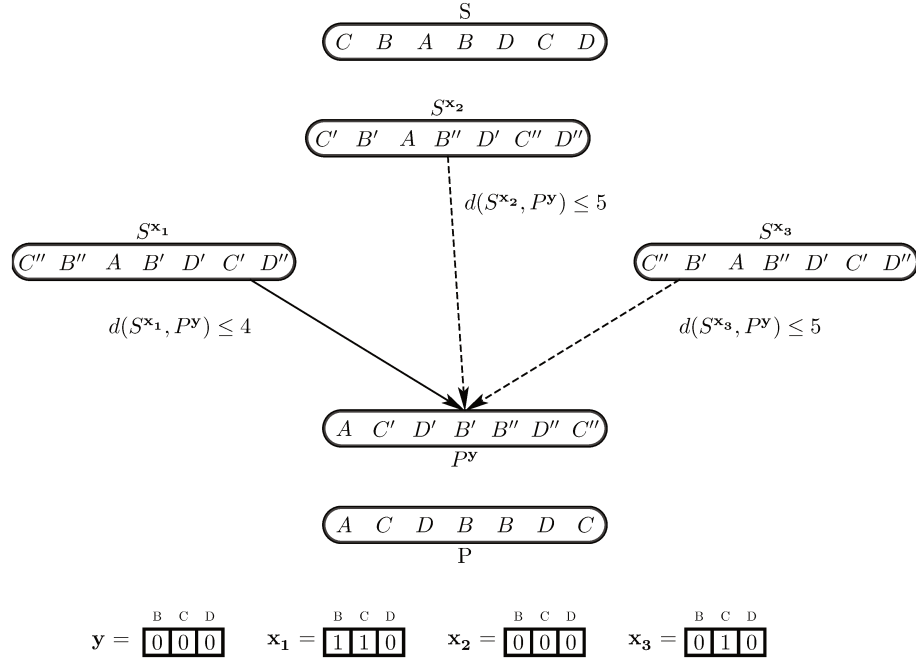


Figura 3.1: Exemplo da heurística Mapeamos Aleatórios com $r = 3$.

3.3 Busca Local (BL)

Nesta heurística, apenas os mapeamentos iniciais do conjunto \mathbf{M} são gerados de forma aleatória. Para gerar os demais mapeamentos, exploramos a vizinhança de alguns dos mapeamentos já conhecidos em busca de mapeamentos que gerem melhores resultados.

Além das strings S e P , essa heurística também recebe como entrada os parâmetros $r, c, \ell \in \mathbb{N}$. Esses parâmetros indicam, respectivamente, quantos mapeamentos são gerados no total, quantos mapeamentos são gerados de forma aleatória e qual o número máximo de vizinhos que podem ser explorados para cada mapeamento.

Novamente, mapeamos P em uma permutação $P^{\mathbf{y}}$ com o mapeamento \mathbf{y} . Em seguida, inicializamos \mathbf{M} com c mapeamentos aleatórios gerados usando a mesma estratégia da heurística MA. 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 um mesmo mapeamento, mantemos um conjunto \mathbf{E} de mapeamentos já explorados.

Em uma busca local, escolhemos um mapeamento $\mathbf{x} \in \mathbf{M} \setminus \mathbf{E}$ que gera a menor distância, e, em caso de empate, escolhemos de forma arbitrária um dos mapeamentos empatados. Seja \mathbf{V} o conjunto de vizinhos do mapeamento \mathbf{x} que não estão em \mathbf{M} . Seleccionamos 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 \mathbf{x} em \mathbf{E} e adicionamos os mapeamentos de \mathbf{V}' em \mathbf{M} . 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 o limitante superior da distância entre as strings S e P é a menor distância encontrada entre todos os mapeamentos de \mathbf{M} .

O Algoritmo 2 apresenta um pseudocódigo da heurística Busca Local e a Figura 3.2 apresenta uma simulação da heurística em um par de strings $S = (C\ B\ A\ B\ D\ C\ D\ A)$ e $P = (A\ C\ D\ B\ A\ B\ D\ C)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}'''}, P^{\mathbf{y}}) \leq 3$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 2: Busca Local

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

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $\mathbf{y} \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $\mathbf{M} \leftarrow \text{mapeamentos\_aleatórios}(c, S)$ 
4    $\mathbf{E} \leftarrow \emptyset$ 
5   enquanto  $|\mathbf{M}| < r$  faça
6      $\mathbf{a} \leftarrow \text{melhor\_mapeamento}(\mathbf{M} \setminus \mathbf{E}, S, P^{\mathbf{y}})$ 
7      $\mathbf{V} \leftarrow \text{vizinhos}(\mathbf{a}) \setminus \mathbf{M}$ 
8      $\mathbf{V}' \leftarrow \text{escolhe\_mapeamentos}(\mathbf{V}, \min(\ell, |\mathbf{V}|, r - |\mathbf{M}|))$ 
9      $\mathbf{E} \leftarrow \mathbf{E} \cup \{\mathbf{a}\}$ 
10     $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{V}'$ 
11  retorna  $\text{menor\_distância}(\mathbf{M}, S, P^{\mathbf{y}})$ 

```

3.4 GRASP

Esta heurística também parte de um conjunto de mapeamentos aleatórios iniciais, mas o restante dos mapeamentos é gerado através da estratégia GRASP (*Greedy Randomized Adaptive Search Procedure*) [28]. Essa é uma estratégia muito utilizada em problemas de otimização combinatória [1, 16], e já foi aplicada em problemas de rearranjo de genomas [2, 24]. A estratégia consiste em uma etapa de construção de soluções aleatórias, a partir de uma lista de candidatos RCL (*Restricted Candidate List*), seguida de uma busca local a partir dessas soluções.

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 são gerados no total e c indica quantos mapeamentos são gerados de forma aleatória. O parâmetro k representa o tamanho da RCL e o parâmetro g a quantidade máxima de mapeamentos gerados na etapa de construção. O parâmetro r_ℓ representa o número total de mapeamentos gerados na etapa de Busca Local. Essa etapa é uma adaptação da heurística BL.

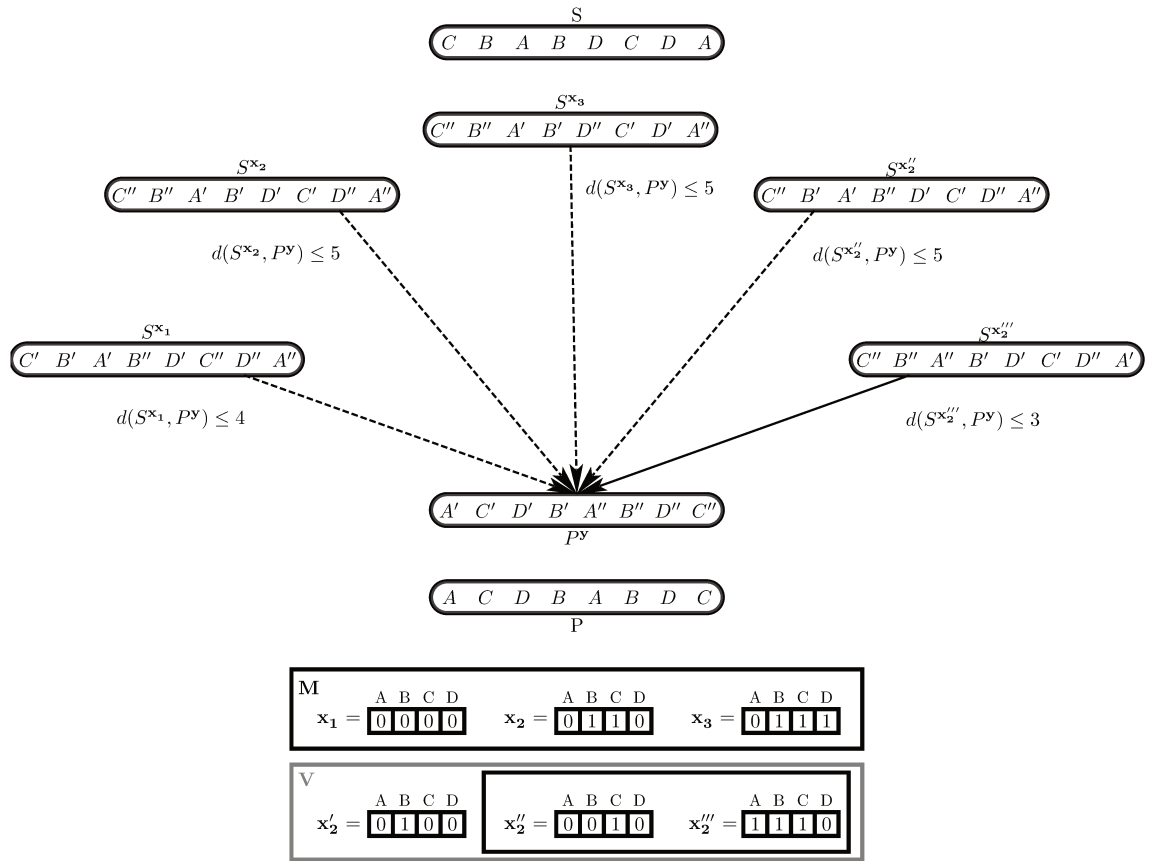


Figura 3.2: Exemplo da heurística Busca Local com $r = 5$, $c = 3$ e $\ell = 2$. O conjunto \mathbf{M} é composto dos 3 mapeamentos aleatórios gerados inicialmente, o conjunto \mathbf{V} contém os vizinhos do mapeamento \mathbf{x}_2 que não estão em \mathbf{M} e os mapeamentos \mathbf{x}_2'' e \mathbf{x}_2''' são os 2 mapeamentos selecionados para fazerem parte dos 5 mapeamentos finais.

Assim como nas heurísticas anteriores, mapeamos P em uma permutação P^y com o mapeamento y . 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 esses g mapeamentos para inicializar uma busca local. Nessa busca, geramos r_ℓ novos mapeamentos. A seguir explicamos essas duas etapas.

Na etapa de construção, para cada mapeamento $\mathbf{x} \in \mathbf{M}$, calculamos a distância entre as permutações S^x e P^y . Com essas distâncias, obtemos o conjunto $\mathbf{RCL} \subset \mathbf{M}$, composto dos k melhores mapeamentos de \mathbf{M} (ou seja, mapeamentos que geram as menores distâncias). Como \mathbf{M} pode ter mapeamentos iguais, é possível que o \mathbf{RCL} tenha menos de k mapeamentos.

Dado o conjunto \mathbf{RCL} , para cada rótulo duplicado α , calculamos o valor:

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

onde $freq(\mathbf{RCL}, \alpha, b)$, para $b \in \{0, 1\}$, é o número de mapeamentos da \mathbf{RCL} para os quais o bit correspondente a α no vetor binário tem valor b .

Exemplo 8. Cálculo dos valores de frequência e probabilidade para um conjunto \mathbf{RCL} de mapeamentos da string sem sinais S .

$$S = (B \quad A \quad C \quad D \quad C \quad D), \quad dup(S) = \{C, D\}$$

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

$$\mathbf{x}_1 = \begin{array}{cc} C & D \\ \boxed{1} & \boxed{0} \end{array} \quad \mathbf{x}_2 = \begin{array}{cc} C & D \\ \boxed{0} & \boxed{0} \end{array} \quad \mathbf{x}_3 = \begin{array}{cc} C & D \\ \boxed{1} & \boxed{1} \end{array}$$

$$\begin{aligned} freq(\mathbf{RCL}, C, 0) &= 1 & freq(\mathbf{RCL}, D, 0) &= 2 \\ freq(\mathbf{RCL}, C, 1) &= 2 & freq(\mathbf{RCL}, D, 1) &= 1 \\ prob(\mathbf{RCL}, C) &= \frac{1}{3} & prob(\mathbf{RCL}, D) &= \frac{2}{3} \end{aligned}$$

Com esses valores, geramos um conjunto \mathbf{N} com g novos mapeamentos. Cada mapeamento \mathbf{n} de \mathbf{N} é gerado aleatoriamente, sendo que, para um rótulo $\alpha \in dup(S)$, a probabilidade de termos $\mathbf{n}[\alpha] = 0$ é $prob(\mathbf{RCL}, \alpha)$ e a probabilidade de termos $\mathbf{n}[\alpha] = 1$ é $1 - prob(\mathbf{RCL}, \alpha)$. 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, que denotamos por BL_GRASP. Essa variação recebe como entrada os conjuntos \mathbf{M} e \mathbf{N} , a string S , a permutação P^y e os parâmetros r_ℓ e r . Como resultado, obtemos o conjunto \mathbf{M} com a adição de r_ℓ novos mapeamentos gerados a partir de uma busca local. Na última busca podem ser adicionados menos de r_ℓ mapeamentos caso o número total de r mapeamentos já tenha sido alcançado.

Nessa variação, utilizamos o conjunto \mathbf{N} no lugar dos mapeamentos aleatórios. Similarmente à heurística BL, mantemos o conjunto \mathbf{E} e geramos o conjunto \mathbf{V} . No lugar do conjunto \mathbf{V}' , escolhemos apenas um mapeamento e , ao final da busca, adicionamos esse 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 essas duas etapas de forma alternada 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 o limitante superior da distância entre as strings S e P é a menor distância encontrada entre todos os mapeamentos de \mathbf{M} .

Os algoritmos 3 e 4 apresentam pseudocódigos da heurística GRASP e da etapa BL_GRASP, respectivamente. A Figura 3.3 apresenta uma simulação da heurística em um par de strings $S = (C\ B\ A\ B\ D\ C\ D)$ e $P = (C\ C\ B\ D\ B\ D\ A)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}_2}, P^{\mathbf{y}}) \leq 3$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 3: GRASP

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

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $\mathbf{y} \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $\mathbf{M} \leftarrow \text{mapeamentos\_aleatórios}(c, S)$ 
4   enquanto  $|\mathbf{M}| < r$  faça
5      $\mathbf{RCL} \leftarrow \text{melhores\_mapeamentos}(\mathbf{M}, k, S, P^{\mathbf{y}})$ 
6      $\mathbf{N} \leftarrow \text{gera\_novos\_mapeamentos}(\mathbf{RCL}, g)$ 
7      $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{N}$ 
8      $\mathbf{M} \leftarrow \text{BL\_GRASP}(\mathbf{M}, \mathbf{N}, S, P^{\mathbf{y}}, r_\ell, r)$ 
9   retorna  $\text{menor\_distância}(\mathbf{M}, S, P^{\mathbf{y}})$ 

```

3.5 Algoritmo Genético (AG)

Esta heurística utiliza a estratégia de Algoritmo Genético [47] para gerar os mapeamentos. Essa estratégia também é comum em problemas de otimização combinatória [38, 49], e foi usada em problemas de rearranjo de genomas [31, 54]. Um algoritmo genético é composto das etapas de geração de uma população inicial, seleção de indivíduos para próxima geração e criação de novos indivíduos em cada geração por mutações e cruzamentos.

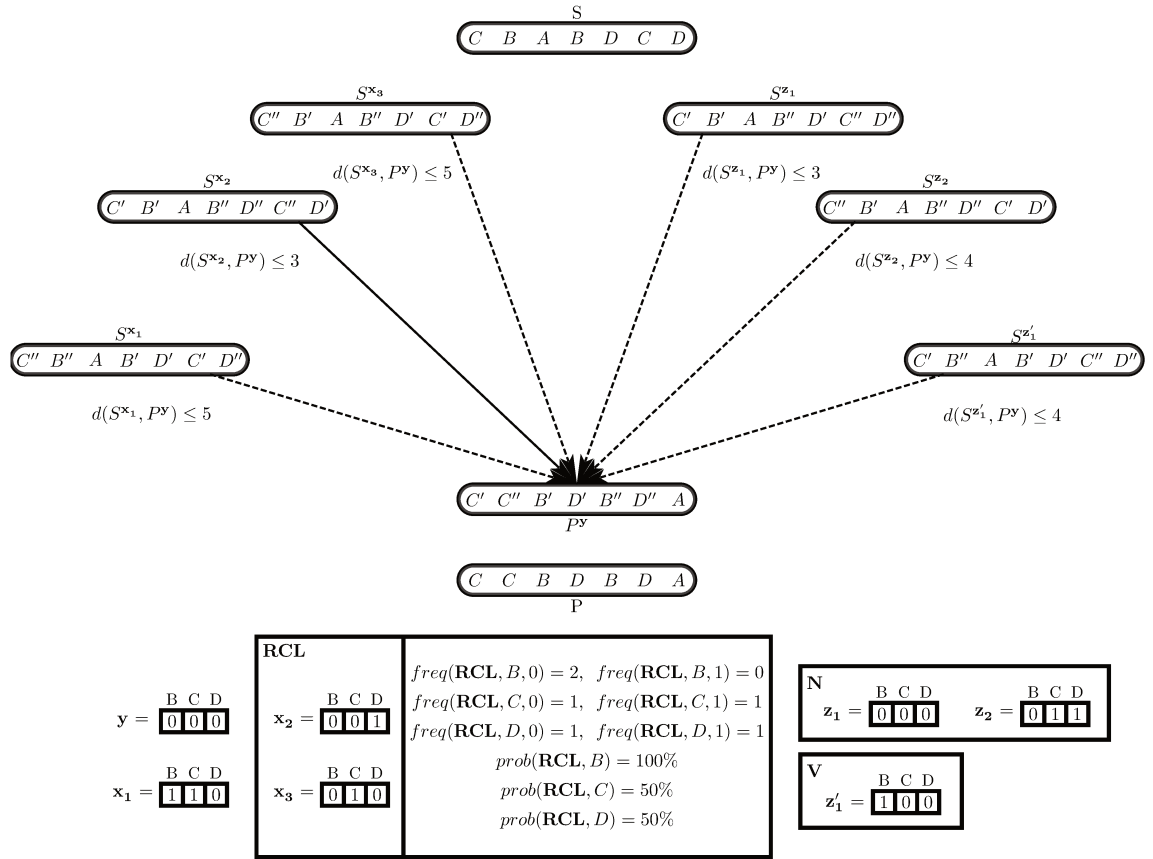


Figura 3.3: Exemplo da heurística GRASP com os parâmetros $r = 6$, $c = 3$, $k = 2$, $g = 2$ e $r_\ell = 1$. Os mapeamentos \mathbf{x}_1 , \mathbf{x}_2 e \mathbf{x}_3 são os 3 mapeamentos aleatórios gerados inicialmente. O mapeamento \mathbf{z}_1 foi escolhido na busca local; como \mathbf{z}'_1 era seu único vizinho desconhecido, ele foi o mapeamento gerado.

Algoritmo 4: BL_GRASP

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

Saída: Novo conjunto \mathbf{M}

```

1 início
2    $\mathbf{E} \leftarrow \emptyset$ 
3    $r_{antigo} \leftarrow |\mathbf{M}|$ 
4   enquanto  $|\mathbf{M}| < \min(r_{antigo} + r_\ell, r)$  faça
5      $\mathbf{a} \leftarrow \text{melhor\_mapeamento}(\mathbf{N} \setminus \mathbf{E}, S, P^y)$ 
6      $\mathbf{V} \leftarrow \text{vizinhos}(\mathbf{a}) - \mathbf{M}$ 
7      $\mathbf{a}' \leftarrow \text{escolhe\_mapeamento}(\mathbf{V})$ 
8     se todos os vizinhos de  $\mathbf{a}$  estão em  $\mathbf{M}$  então
9        $\mathbf{E} \leftarrow \mathbf{E} \cup \{\mathbf{a}\}$ 
10     $\mathbf{M} \leftarrow \mathbf{M} \cup \{\mathbf{a}'\}$ 
11     $\mathbf{N} \leftarrow \mathbf{N} \cup \{\mathbf{a}'\}$ 
12  retorna  $\mathbf{M}$ 

```

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 são gerados no total e c indica quantos mapeamentos são gerados para população inicial. O parâmetro k é o número de mapeamentos selecionados para a próxima geração e os parâmetros t_c e t_m são usados nas etapas de cruzamento e mutação, respectivamente.

Assim como nas heurísticas anteriores, mapeamos P em uma permutação P^y com o mapeamento \mathbf{y} . Em seguida, inicializamos \mathbf{M} com c mapeamentos aleatórios, gerados usando a mesma estratégia da heurística MA, sendo essa 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 início de uma geração, para cada mapeamento $\mathbf{x} \in \mathbf{G}$, calculamos a distância entre as permutações S^x e P^y . Com essas distâncias, selecionamos os k melhores mapeamentos de \mathbf{G} (mapeamentos que geram as menores distâncias), e o restante dos mapeamentos são removidos de \mathbf{G} . Em seguida, aumentamos a população de \mathbf{G} aplicando cruzamentos e mutações em seus elementos conforme explicado a seguir.

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

Exemplo 9. 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 . Os bits escolhidos para formar o mapeamento \mathbf{z} estão indicados em cinza.

$$\begin{aligned}
 S &= (C \ C \ A \ D \ B \ B \ D), \quad t_c = 2 \\
 S^{\mathbf{x}_1} &= (C'' \ C' \ A \ D'' \ B' \ B'' \ D') \\
 S^{\mathbf{x}_2} &= (C' \ C'' \ A \ D' \ B'' \ B' \ D'') \\
 S^{\mathbf{z}} &= (C' \ C'' \ A \ D'' \ B' \ B'' \ D') \\
 \mathbf{x}_1 &= \begin{array}{c} B \ C \ D \\ \boxed{0} \ \boxed{1} \ \boxed{1} \end{array} \quad \mathbf{x}_2 = \begin{array}{c} B \ C \ D \\ \boxed{1} \ \boxed{0} \ \boxed{0} \end{array} \quad \mathbf{z} = \begin{array}{c} B \ C \ D \\ \boxed{0} \ \boxed{0} \ \boxed{1} \end{array}
 \end{aligned}$$

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

Exemplo 10. Um mapeamento \mathbf{z} gerado através de uma mutação em um mapeamento \mathbf{x} de uma string sem sinais S . Os bits escolhidos para serem invertidos estão indicados em cinza.

$$\begin{aligned}
 S &= (C \ C \ A \ D \ B \ B \ D), \quad t_m = 2 \\
 S^{\mathbf{x}} &= (C'' \ C' \ A \ D'' \ B' \ B'' \ D') \\
 S^{\mathbf{z}} &= (C' \ C'' \ A \ D' \ B' \ B'' \ D'') \\
 \mathbf{x} &= \begin{array}{c} B \ C \ D \\ \boxed{0} \ \boxed{1} \ \boxed{1} \end{array} \quad \mathbf{z} = \begin{array}{c} B \ C \ D \\ \boxed{0} \ \boxed{0} \ \boxed{0} \end{array}
 \end{aligned}$$

Ao final da geração, adicionamos os mapeamentos gerados nos conjuntos \mathbf{G} e \mathbf{M} . Caso o número de mapeamentos em \mathbf{M} seja superior a r , ignoramos alguns dos mapeamentos gerados por mutações e, se necessário, gerados por cruzamentos. Note que geramos no máximo $\lfloor \frac{k}{2} \rfloor$ mapeamentos por cruzamentos e k por mutações.

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 o limitante superior da distância entre as strings S e P é a menor distância encontrada entre todos os mapeamentos de \mathbf{M} .

O Algoritmo 5 apresenta um pseudocódigo da heurística Algoritmo Genético e a Figura 3.4 apresenta um exemplo da aplicação da heurística em um par de strings $S = (+C \ -B \ -A \ -B \ -D \ -C \ +D)$ e $P = (+C \ +C \ +B \ +D \ +B \ +D \ +A)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}_2}, P^{\mathbf{y}}) \leq 5$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 5: Algoritmo Genético

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: Limitante superior para a distância entre S e P

```

1 início
2    $y \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $M \leftarrow \text{mapeamentos\_aleatórios}(c, S)$ 
4    $G \leftarrow M$ 
5   enquanto  $|M| < r$  faça
6      $G \leftarrow \text{seleciona\_melhores\_mapeamentos}(G, k, S, P^y)$ 
7      $G \leftarrow G \cup \text{cruzamentos}(G, t_c, \min(\lfloor \frac{k}{2} \rfloor, r - |M|)) \cup$ 
        $\text{mutações}(G, t_m, \min(k, \max(0, r - |M| - \lfloor \frac{k}{2} \rfloor)))$ 
8      $M \leftarrow M \cup G$ 
9   retorna  $\text{menor\_distância}(M, S, P^y)$ 
  
```

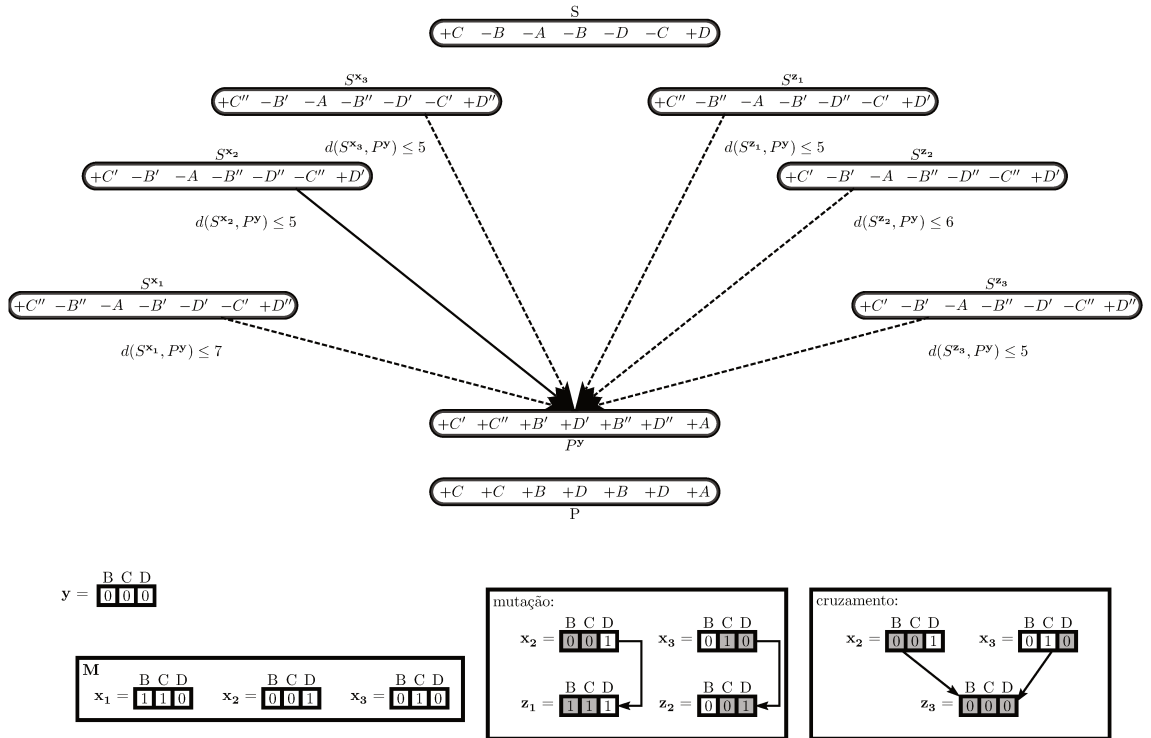


Figura 3.4: Exemplo da heurística Algoritmo Genético com os parâmetros $r = 6$, $c = 3$, $k = 2$, $t_m = 2$ e $t_c = 2$. Os mapeamentos x_1 , x_2 e x_3 são os 3 mapeamentos aleatórios gerados inicialmente. Os mapeamentos x_2 e x_3 foram escolhidos para sofrer mutações e cruzamentos.

3.6 Busca Tabu (BT)

Esta heurística gera um conjunto de mapeamentos através de uma Busca Tabu [32], uma variação da heurística BL. Diferente da heurística BL, as buscas são realizadas a partir de operações, chamadas movimentos, no último mapeamento escolhido e não no melhor mapeamento conhecido. Além disso, mantemos uma lista tabu com movimentos que não devem ser utilizados para gerar os novos mapeamentos.

A heurística recebe como entrada as strings S e P , assim como os parâmetros $r, t, \ell \in \mathbb{N}$. Como nas demais heurísticas, r é o número total de mapeamentos gerados. O parâmetro t indica o tamanho da lista tabu e ℓ indica o número máximo de vizinhos explorados.

Assim como nas heurísticas anteriores, mapeamos P em uma permutação P^y com o mapeamento y . Também escolhemos um mapeamento arbitrário de S para ser o primeiro mapeamento. O conjunto de mapeamentos \mathbf{M} é inicializado com esse mapeamento. A cada iteração exploramos até ℓ vizinhos do mapeamento atual e escolhemos o melhor deles para ser o novo mapeamento. Os vizinhos explorados são adicionados ao conjunto \mathbf{M} . A seguir descrevemos como esses vizinhos são escolhidos.

Chamamos de *movimento* a operação de inversão de um bit do vetor binário que representa um mapeamento. Note que, dado um mapeamento, podemos obter cada um dos seus vizinhos através de um movimento.

Exemplo 11. Um movimento aplicado no mapeamento \mathbf{x} para gerar um mapeamento \mathbf{z} . Note que \mathbf{x} e \mathbf{z} são vizinhos.

$$\begin{aligned}
 S &= (C \ C \ A \ D \ B \ B \ D) \\
 S^{\mathbf{x}} &= (C'' \ C' \ A \ D'' \ B' \ B'' \ D') \\
 S^{\mathbf{z}} &= (C'' \ C' \ A \ D' \ B' \ B'' \ D'') \\
 \mathbf{x} &= \begin{array}{|c|c|c|} \hline B & C & D \\ \hline 0 & 1 & 1 \\ \hline \end{array} \quad \mathbf{z} = \begin{array}{|c|c|c|} \hline B & C & D \\ \hline 0 & 1 & 0 \\ \hline \end{array}
 \end{aligned}$$

Durante o processo de execução da heurística, mantemos uma lista tabu T composta dos movimentos usados para obter os t últimos mapeamentos escolhidos. Em cada iteração da heurística escolhemos $\min(\ell, |dup(S)| - |T|, r - |\mathbf{M}|)$ novos mapeamentos. Esses mapeamentos são escolhidos aleatoriamente entre os vizinhos do mapeamento atual gerados por um movimento não pertencente a T . O melhor dos mapeamentos escolhidos é utilizado na próxima iteração. No final da iteração, adicionamos os mapeamentos gerados ao conjunto \mathbf{M} e atualizamos a lista T adicionando o movimento usado. Quando $|T| = t$, o movimento mais antigo da lista é removido.

Quando obtemos r mapeamentos em \mathbf{M} , encerramos a heurística. Como nas heurísticas anteriores, para cada mapeamento $\mathbf{x} \in \mathbf{M}$ obtemos a distância entre as permutações $S^{\mathbf{x}}$ e P^y . O resultado para o limitante superior da distância entre as strings S e P é a menor distância encontrada entre todos os mapeamentos de \mathbf{M} .

O Algoritmo 6 apresenta um pseudocódigo da heurística Busca Tabu e a Figura 3.5 apresenta uma simulação da heurística em um par de strings $S = (C \ B \ A \ B \ D \ C \ D)$ e $P = (A \ C \ D \ B \ B \ D \ C)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}^2}, P^y) \leq 4$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 6: Busca Tabu

Entrada: Strings S e P , número total de mapeamentos r , tamanho da lista tabu t e número máximo de vizinhos explorados ℓ

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $\mathbf{y} \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $\mathbf{a} \leftarrow \text{mapeamento\_padrão}(S)$ 
4    $\mathbf{M} \leftarrow \{\mathbf{a}\}$ 
5    $T \leftarrow \emptyset$ 
6   enquanto  $|\mathbf{M}| < r$  faça
7      $\mathbf{V} \leftarrow \text{vizinhos\_não\_tabus}(\mathbf{a}, T)$ 
8      $\mathbf{V}' \leftarrow \text{escolhe\_mapeamentos}(\mathbf{V}, \min(\ell, |\text{dup}(S)| - |T|, r - |\mathbf{M}|))$ 
9      $\mathbf{a}' \leftarrow \text{melhor\_mapeamento}(\mathbf{V}', S, P^{\mathbf{y}})$ 
10     $\mathbf{M} \leftarrow \mathbf{M} \cup \mathbf{V}'$ 
11     $m \leftarrow \text{movimento}(\mathbf{a}, \mathbf{a}')$ 
12     $T \leftarrow \text{atualiza\_tabu}(T, m, t)$ 
13     $\mathbf{a} \leftarrow \mathbf{a}'$ 
14  retorna  $\text{menor\_distância}(\mathbf{M}, S, P^{\mathbf{y}})$ 

```

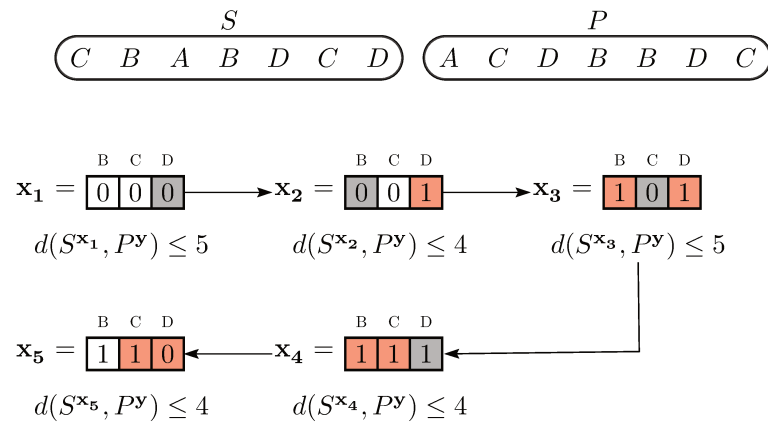


Figura 3.5: Exemplo da heurística Busca Tabu com os parâmetros $r = 5$, $t = 2$ e $\ell = 1$. Em cada mapeamento, estão indicados em vermelho os bits que não podem ser invertidos e em cinza o bit que foi invertido para gerar o próximo mapeamento.

3.7 *Simulated Annealing* (SA)

Esta heurística gera um conjunto de mapeamentos através da estratégia de *Simulated Annealing* [41], outra variação da heurística BL. Novamente, cada busca é feita a partir do último mapeamento escolhido. Diferente da heurística BT, exploramos um vizinho por vez e realizamos um teste para verificar se ele é escolhido. A forma como os mapeamentos são explorados nessa heurística é inspirada na técnica metalúrgica *annealing*, onde um metal é aquecido até uma temperatura elevada e resfriado de forma controlada. Interpretamos o resfriamento como uma diminuição na probabilidade de trocar o mapeamento atual por um mapeamento com resultado pior. Dessa forma, começamos com uma probabilidade alta que permite uma grande variação das possíveis soluções e diminuimos a temperatura para refinar a busca.

A heurística recebe as strings S e P , assim como os parâmetros $r, t_0, i, k, s \in \mathbb{N}$. Como nas demais heurísticas, r é o número total de mapeamentos gerados. Os parâmetros t_0 e s , usados para ajustar o valor da temperatura durante a heurística, representam a temperatura inicial e a taxa de variação da temperatura, respectivamente. O parâmetro k é uma constante multiplicativa para temperatura e i representa o número de iterações sem melhora do melhor mapeamento conhecido antes da atualização da temperatura.

Assim como nas outras heurísticas, mapeamos P em uma permutação P^y com o mapeamento y . Também escolhemos um mapeamento arbitrário de S para ser o primeiro mapeamento. O conjunto de mapeamentos \mathbf{M} é inicializado com esse mapeamento. A cada iteração, exploramos alguns dos vizinhos do mapeamento atual. Cada vizinho explorado é adicionado ao conjunto \mathbf{M} . A seguir descrevemos o teste realizado para determinar se o mapeamento atual é substituído pelo vizinho sendo explorado.

Sejam \mathbf{x} o mapeamento atual e \mathbf{z} o vizinho sendo explorado. Denotamos por $\Delta_{score} = d(S^z, P^y) - d(S^x, P^y)$ a diferença entre as distâncias obtidas por esses mapeamentos. Caso o novo mapeamento seja melhor que o anterior (ou seja $\Delta_{score} < 0$) ou os demais vizinhos de \mathbf{x} já tenham sido explorados, utilizamos \mathbf{z} como o novo mapeamento. Caso contrário, utilizamos \mathbf{z} apenas com probabilidade $p(\Delta_{score}, t, k)$ dada por:

$$p(\Delta_{score}, t, k) = \frac{\exp\left(-\frac{\Delta_{score}}{k \times t}\right)}{2}$$

O valor de t é inicialmente t_0 e, após gerarmos i mapeamentos sem melhorar a menor distância conhecida, atualizamos t para $s \times t$. Com a atualização o valor de t diminui no decorrer da execução e, caso t chegue a 0, 1, reiniciamos seu valor para t_0 . O denominador 2 na fórmula da probabilidade garante que, caso $\Delta_{score} = 0$, temos 50% de chance de trocar para o novo mapeamento.

Continuamos esse processo até obtermos r mapeamentos em \mathbf{M} . Como nas heurísticas anteriores, para cada mapeamento $\mathbf{x} \in \mathbf{M}$ obtemos a distância entre as permutações S^x e P^y . O resultado para o limitante superior da distância entre as strings é a menor distância encontrada entre todos os mapeamentos de \mathbf{M} .

O Algoritmo 7 apresenta um pseudocódigo da heurística *Simulated Annealing* e a Figura 3.6 apresenta uma simulação da heurística utilizando um par de strings $S = (C\ B\ A\ B\ D\ C\ D\ A)$ e $P = (A\ A\ C\ D\ B\ B\ C\ D)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{x}_3}, P^{\mathbf{y}}) \leq 4$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 7: Simulated Annealing

Entrada: Strings S e P , número total de mapeamentos r , temperatura inicial t_0 , número de iterações sem melhora i , constante multiplicativa k e taxa de variação da temperatura s

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $\mathbf{y} \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $\mathbf{x} \leftarrow \text{mapeamento\_padrão}(S)$ 
4    $\mathbf{M} \leftarrow \{\mathbf{x}\}$ 
5    $t \leftarrow t_0$ 
6   enquanto  $|\mathbf{M}| < r$  faça
7      $\mathbf{V} \leftarrow \text{vizinhos}(\mathbf{x})$ 
8     enquanto  $|\mathbf{M}| < r$  e  $\mathbf{x}$  não foi substituído faça
9        $\mathbf{z} \leftarrow \text{escolhe\_mapeamento}(\mathbf{V})$ 
10       $\mathbf{N} \leftarrow \mathbf{N} \setminus \{\mathbf{z}\}$ 
11       $\mathbf{M} \leftarrow \mathbf{M} \cup \{\mathbf{z}\}$ 
12       $\Delta_{score} \leftarrow d(S^{\mathbf{z}}, P^{\mathbf{y}}) - d(S^{\mathbf{x}}, P^{\mathbf{y}})$ 
13      se  $\Delta_{score} < 0$  ou  $\mathbf{V} = \{\mathbf{z}\}$  então
14         $\mathbf{x} \leftarrow \mathbf{z}$ 
15      senão
16         $\mathbf{x} \leftarrow \mathbf{z}$  com probabilidade  $p(\Delta_{score}, t, k)$ 
17      se a melhor distância conhecida não melhorou por  $i$  iterações então
18         $t \leftarrow s \times t$ 
19      se  $t \leq 0,1$  então
20         $t \leftarrow t_0$ 
21  retorna menor_distância( $\mathbf{M}, S, P^{\mathbf{y}}$ )

```

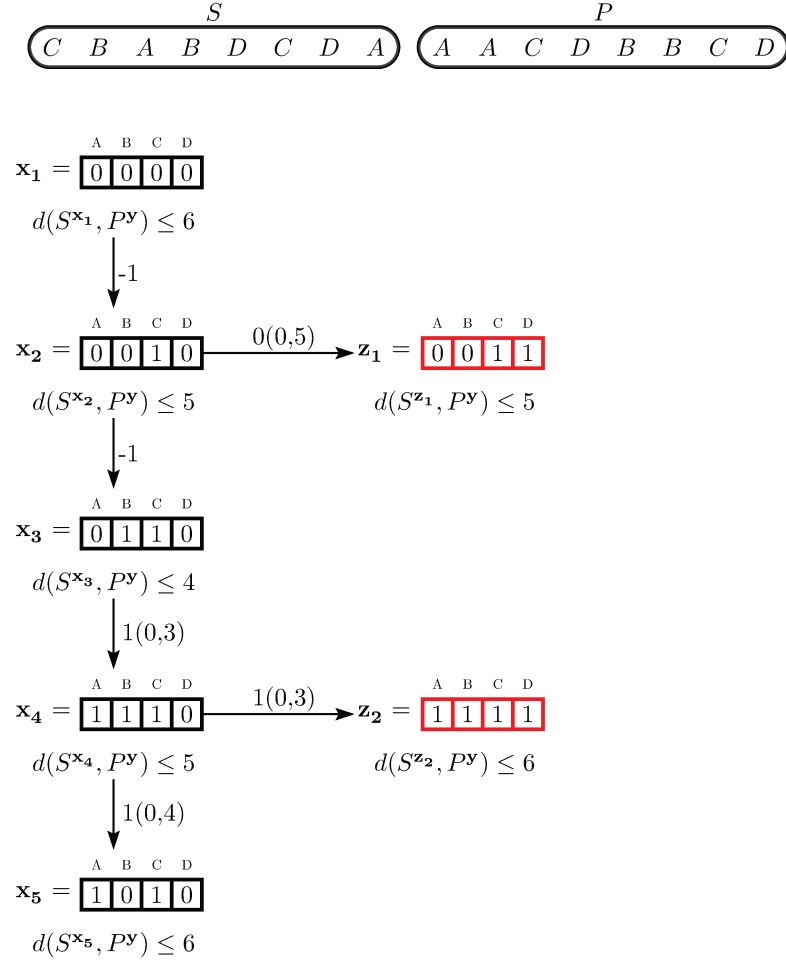


Figura 3.6: Exemplo da heurística *Simulated Annealing* com os parâmetros $r = 7$, $t_0 = 2$, $i = 2$, $k = 1$ e $s = 2$. Ao lado de cada seta indicamos o valor de Δ_{score} (com $p(\Delta_{score}, k, t)$ entre parênteses, caso $\Delta_{score} \geq 0$). Os mapeamentos \mathbf{z}_1 e \mathbf{z}_2 não foram escolhidos para substituir os mapeamentos atuais da busca (\mathbf{x}_2 e \mathbf{x}_4 , respectivamente). Note que, entre \mathbf{x}_4 e \mathbf{x}_5 , o valor de t foi atualizado, pois foram gerados dois mapeamentos sem melhorar a menor distância conhecida.

3.8 Busca Cuco (BC)

Esta heurística se baseia na Busca Cuco [60], que consiste em múltiplas buscas sendo realizadas de forma independente e sendo descartadas quando não geram bons resultados. Seguindo a terminologia da heurística, chamamos cada uma dessas buscas de um *ninho*.

A heurística recebe como entrada as strings S e P , assim como os parâmetros r , c , g e p . O parâmetro r correspondente ao número total de mapeamentos gerados. Os parâmetros c e g são o número de ninhos e o número de iterações até um ninho ser eliminado, respectivamente. O parâmetro p é uma probabilidade usada na geração dos novos mapeamentos.

Essa heurística se inspira no comportamento dos pássaros cuco. Esses pássaros colocam seus ovos em ninhos de outras espécies e em alguns casos replicam as características dos ovos dessa outra espécie. Se um cuco é descoberto no ninho é comum que o pássaro abandone o ninho e forme um novo.

Na heurística, cada ninho é uma busca representada pelo seu mapeamento atual. Os ovos de cucos correspondem a novos mapeamentos adicionados aos ninhos, possivelmente substituindo o atual. O comportamento dos pássaros abandonarem o ninho é simulado ao reiniciarmos as piores buscas.

Inicialmente, continuamos a mapear P em P^y e criamos um conjunto \mathbf{N} gerando c mapeamentos aleatórios, correspondentes aos c ninhos. Esses mapeamentos iniciais são adicionados ao conjunto \mathbf{M} .

A cada iteração da heurística, escolhemos aleatoriamente um ninho contendo um mapeamento \mathbf{x} . Em seguida, geramos um mapeamento \mathbf{z} a partir de \mathbf{x} . Para gerar esse novo mapeamento, invertemos os bits do vetor correspondente ao mapeamento \mathbf{x} . Para cada bit invertido, temos uma probabilidade p de encerrar esse processo e retornar o mapeamento \mathbf{z} . Dessa forma, temos uma probabilidade $(1 - p)^{n-1}p$ de inverter n bits de \mathbf{x} , o que garante uma chance maior de inverter poucos bits, mas permite que um número maior de bits possam ser invertidos. Com esse mapeamento, encontramos as distâncias entre as permutações $S^{\mathbf{x}}$ e P^y , e entre as permutações $S^{\mathbf{z}}$ e P^y . Caso o mapeamento \mathbf{z} corresponda a uma distância menor que o mapeamento \mathbf{x} , substituímos \mathbf{x} por \mathbf{z} em \mathbf{N} e, caso contrário, mantemos \mathbf{x} . Em ambos os casos, adicionamos \mathbf{z} no conjunto \mathbf{M} .

Após g iterações, selecionamos o ninho cujo mapeamento possui a maior distância e substituímos seu mapeamento por um novo mapeamento aleatório, e esse mapeamento aleatório é adicionado a \mathbf{M} . Essa etapa corresponde a eliminar o pior ninho e criar um ninho novo.

Quando \mathbf{M} contiver r mapeamentos, encontramos as distâncias entre $S^{\mathbf{x}}$ e P^y para todo mapeamento $\mathbf{x} \in \mathbf{M}$. O resultado para o limitante superior da distância entre as strings S e P é a menor distância encontrada.

O Algoritmo 8 apresenta um pseudocódigo da heurística Busca Cuco e a Figura 3.7 apresenta uma simulação da heurística em um par de strings $S = (C B A B D C D A)$ e $P = (A A C D B B C D)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{\mathbf{z}^3}, P^y) \leq 4$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 8: Busca Cuco

Entrada: Strings S e P , número total de mapeamentos r , número de ninhos c , número de iterações até eliminarmos um ninho g e probabilidade usada na geração de um novo mapeamento p

Saída: Limitante superior para a distância entre S e P

```

1 início
2    $y \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $N \leftarrow \text{mapeamentos\_aleatórios}(c, S)$ 
4    $M \leftarrow N$ 
5   enquanto  $|M| < r$  faça
6      $x \leftarrow \text{escolhe\_mapeamento}(N)$ 
7      $z \leftarrow \text{inverte\_elementos}(x, p)$ 
8     se  $d(S^z, P^y) < d(S^x, P^y)$  então
9        $N \leftarrow N \setminus \{x\} \cup \{z\}$ 
10     $M \leftarrow M \cup \{z\}$ 
11    se passaram  $g$  iterações após eliminarmos um ninho então
12       $x \leftarrow \text{pior\_mapeamento}(N, S, P^y)$ 
13       $z \leftarrow \text{mapeamento\_aleatório}(S)$ 
14       $N \leftarrow N \setminus \{x\} \cup \{z\}$ 
15       $M \leftarrow M \cup \{z\}$ 
16  retorna  $\text{menor\_distância}(M, S, P^y)$ 
  
```

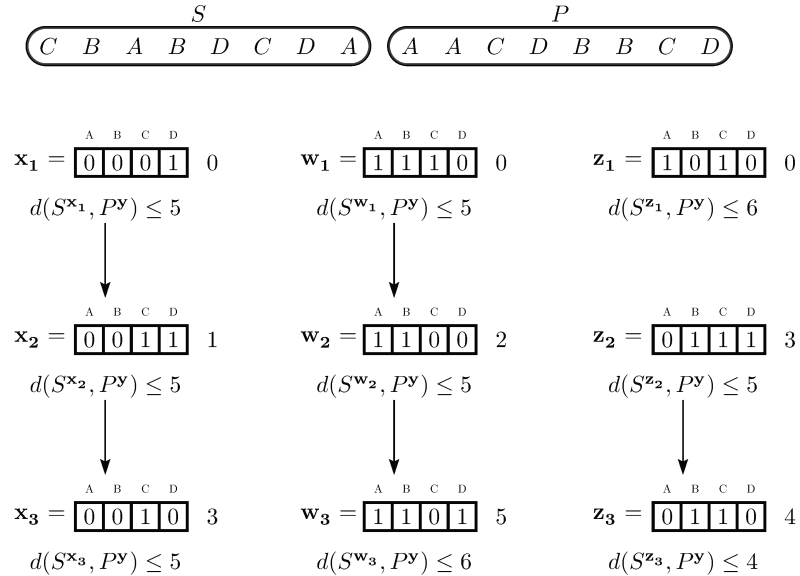


Figura 3.7: Exemplo da heurística Busca Cuco com os parâmetros $r = 9$, $c = 3$, $g = 3$ e $p = 0,9$. Cada ninho contém os mapeamentos com uma letra (x , w ou z). O número ao lado de cada mapeamento indica a iteração onde esse mapeamento foi gerado, onde o número 0 indica mapeamentos aleatórios gerados inicialmente. O mapeamento z_2 foi gerado aleatoriamente para substituir o mapeamento z_1 , que corresponde à pior distância entre os mapeamentos existentes na terceira iteração.

3.9 Separação (Sep)

Esta heurística utiliza uma estratégia mais inteligente para geração de mapeamentos aleatórios. Diferente da heurística MA, fixamos incrementalmente os bits dos mapeamentos de forma a melhorar a qualidade dos mapeamentos sendo gerados.

A heurística recebe como entrada as strings S e P , assim como o parâmetro r , correspondente ao número total de mapeamentos gerados.

Para P , realizamos o mapeamento em P^y e para S construímos um conjunto \mathbf{M} de mapeamentos. Mantemos uma lista L de bits fixados limitando os mapeamentos que podem ser gerados. Representamos a lista L por pares com o rótulo e o valor do bit, e o conjunto $R(L)$ é o conjunto de rótulos fixados por L . A cada iteração gera aproximadamente $c = \frac{r - |\mathbf{M}|}{|dup(S)| - |L|}$ mapeamentos ($\lceil c \rceil$, caso L seja vazia, e $\lfloor c \rfloor$, caso contrário). A seguir, descrevemos como essa lista é mantida e como ela é usada para gerar os mapeamentos

Inicialmente a lista L está vazia. A cada iteração, a lista L já está atualizada pelas iterações anteriores. Quando geramos os mapeamentos dessa iteração, escolhemos sempre o valor no qual um bit foi fixado, caso ele esteja presente na lista L , ou escolhemos os valores aleatoriamente, caso contrário. Note que se o número de mapeamentos possíveis ($2^{|dup(S)| - |L|}$) for menor ou igual ao número máximo de mapeamentos que podem ser gerados (c), podemos gerar todos os mapeamentos sem a necessidade de gerar mapeamentos aleatórios.

Ao final de cada iteração escolhemos um novo bit para fixar. Para realizar essa tarefa, para cada rótulo $\alpha \in dup(S)$, cujo bit correspondente não está fixado em L , separamos o conjunto \mathbf{M} em dois conjuntos \mathbf{M}_α^0 , composto pelos mapeamentos cujo bit corresponde a α é 0, e \mathbf{M}_α^1 , composto pelos mapeamentos cujo bit corresponde a α é 1. Em seguida, calculamos as médias das distâncias correspondentes aos mapeamentos do conjunto \mathbf{M}_α^0 (*média* $_\alpha^0$) e do conjunto \mathbf{M}_α^1 (*média* $_\alpha^1$). Fixamos o bit relativo ao rótulo β correspondente à maior diferença entre as médias calculadas, ou seja:

$$\beta = \arg \max_{\alpha \in dup(S) \setminus R(L)} (|média_\alpha^0 - média_\alpha^1|)$$

Caso a média correspondente a \mathbf{M}_β^0 seja menor que a média correspondente a \mathbf{M}_β^1 , fixamos o bit em 0, e fixamos em 1, caso contrário. Se, após fixarmos esse bit, L contiver $|dup(S)|$ bits fixados, esvaziamos L para que possamos gerar mais mapeamentos.

Após obtermos r mapeamentos no conjunto \mathbf{M} , escolhemos o mapeamento que gera a menor distância, como nas heurísticas anteriores.

O Algoritmo 9 apresenta um pseudocódigo da heurística Separação e a Figura 3.8 apresenta uma simulação da heurística em um par de strings $S = (C \ B \ A \ B \ D \ C \ D)$ e $P = (A \ C \ D \ B \ B \ D \ C)$. Nesse caso, a heurística determina que $d(S, P) \leq d(S^{x_2}, P^y) \leq 4$. A distância usada nesse exemplo é a distância de reversão.

Algoritmo 9: Separação

Entrada: Strings S e P e número total de mapeamentos r
Saída: Limitante superior para a distância entre S e P

```

1 início
2    $y \leftarrow \text{mapeamento\_padrão}(P)$ 
3    $M \leftarrow \emptyset$ 
4    $L \leftarrow \emptyset$ 
5   enquanto  $|M| < r$  faça
6      $c \leftarrow \frac{r - |M|}{|dup(S)| - |L|}$ 
7     se  $L = \emptyset$  então
8        $M \leftarrow M \cup \text{gera\_mapeamentos}(L, \lceil c \rceil, S)$ 
9     senão
10       $M \leftarrow M \cup \text{gera\_mapeamentos}(L, \lfloor c \rfloor, S)$ 
11     para cada  $\alpha \in dup(S) \setminus R(L)$  faça
12        $M_\alpha^0 \leftarrow \text{mapeamentos\_com\_bit\_0}(M, \alpha)$ 
13        $M_\alpha^1 \leftarrow \text{mapeamentos\_com\_bit\_1}(M, \alpha)$ 
14        $média_\alpha^0 \leftarrow \text{média\_das\_distâncias}(M_\alpha^0, S, P^y)$ 
15        $média_\alpha^1 \leftarrow \text{média\_das\_distâncias}(M_\alpha^1, S, P^y)$ 
16        $\beta \leftarrow \arg \max_{\alpha \in dup(S)} (|média_\alpha^0 - média_\alpha^1|)$ 
17       se  $média_\beta^0 < média_\beta^1$  então
18          $L = L \cup (\beta, 0)$ 
19       senão
20          $L = L \cup (\beta, 1)$ 
21       se  $|L| = |dup(S)|$  então
22          $L \leftarrow \emptyset$ 
23   retorna  $\text{menor\_distância}(M, S, P^y)$ 

```

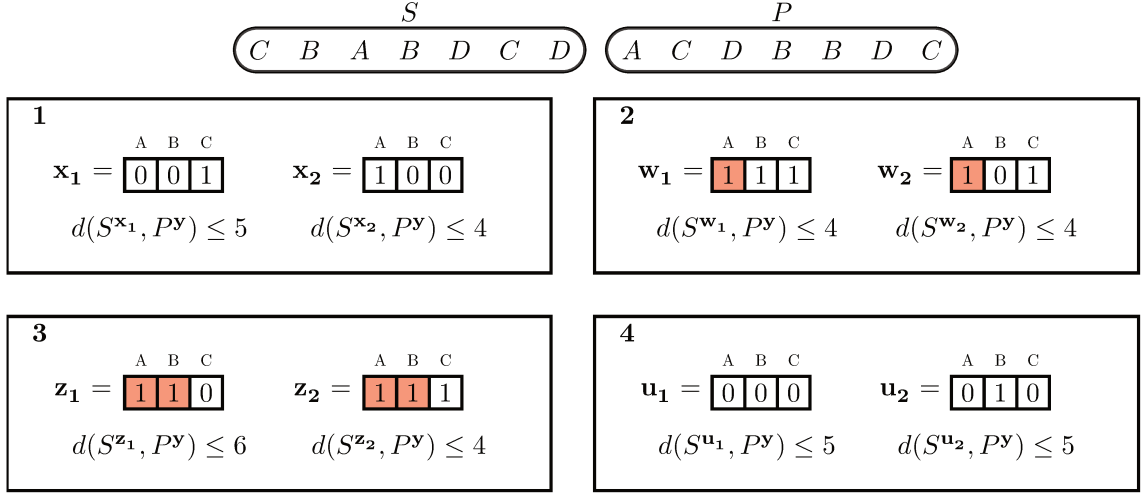


Figura 3.8: Exemplo da heurística Separação com o parâmetro $r = 7$. O número no canto superior esquerdo de cada retângulo indica a iteração em que os mapeamentos foram gerados. Em vermelho estão indicados os bits fixados na iteração anterior. Note que após a iteração 3 a lista de bits fixados foi reiniciada, pois todos os bits já haviam sido fixados e ainda não haviam sido gerados 7 mapeamentos distintos (w_1 e z_2 são iguais).

3.10 Experimentos Práticos

Nesta seção, apresentamos uma comparação entre as heurísticas propostas e o algoritmo SOAR [19]. Esse algoritmo foi proposto para o evento de reversão com sinais, mas simples adaptações permitem utilizá-lo com os demais modelos (no Capítulo 4 descrevemos essas adaptações). Para o caso em que os genes estão apenas duplicados, o algoritmo SOAR e suas adaptações garantem um fator de aproximação 3 para os problemas $D\bar{R}$ e DR , e um fator de aproximação 4,5 para os problemas DT , $D\bar{R}T$ e DRT .

Para verificar as heurísticas aplicadas à strings com diferentes quantidades de caracteres duplicados geramos bases de dados artificiais. A seguir, explicamos os processos para a geração dessas bases de dados, a escolha dos algoritmos para encontrar a distância entre permutações e o ajuste dos parâmetros para cada heurística. Ao fim dessa seção, apresentamos os resultados práticos obtidos.

3.10.1 Bases de Dados

Desenvolvemos uma base de dados específica para cada modelo, gerada a partir da aplicação de operações permitidas pelo modelo. Além disso, desenvolvemos duas bases de dados com pares de strings que requerem um número elevado de reversões ou transposições para serem transformadas umas nas outras. Para simplificar, consideramos que cada rótulo é representado por um número inteiro.

Base REV

Esta base de dados é composta por 10 conjuntos, cada um com 1000 pares de strings sem sinais (origem e destino). Cada conjunto é formado por strings de um mesmo tamanho. Os tamanhos variam de 100 até 1000 em intervalos de 100. Cada string S possui 25% de seu tamanho em rótulos duplicados, dessa forma $|dup(S)| = \frac{|S|}{4}$. O seguinte processo foi utilizado para a criação de cada par de strings de tamanho n :

1. Para produzir a string de origem, distribuimos de forma aleatória os caracteres $(1, 2, \dots, \frac{3n}{4}, 1, 2, \dots, \frac{n}{4})$.
2. Para a string de destino, nós aplicamos $\frac{n}{4}$ operações de reversão. Em cada reversão $\rho(i, j)$, os índices i e j são escolhidos de forma aleatória, sendo que $1 \leq i < j \leq n$.

Base SREV

Esta base foi construída da mesma forma que a base REV, mas aqui consideramos que temos strings com sinais e, ao aplicar uma reversão $\rho(i, j)$, podemos ter $i = j$. Os caracteres da string de origem têm apenas sinais positivos. Os caracteres da string de destino, por outro lado, podem possuir sinais negativos, pois as reversões aplicadas invertem os sinais dos caracteres afetados.

Base TRANS

Esta base foi construída da mesma forma que a base REV, mas, para gerar a string de destino, no lugar de operações de reversão aplicamos operações de transposição. Em cada transposição $\tau(i, j, k)$, os índices i , j e k são escolhidos de forma aleatória, sendo que $1 \leq i < j < k \leq n + 1$.

Base REVTRANS

Esta base foi construída da mesma forma que a base REV, mas, para gerar a string de destino, utilizamos tanto operações de reversão quanto operações de transposição. São aplicadas $\frac{n}{8}$ reversões e $\frac{n}{8}$ transposições, cujos índices são escolhidos de forma aleatória respeitando as mesmas restrições da base REV e TRANS, respectivamente. Escolhemos uma ordem aleatória para aplicar todas as $\frac{n}{4}$ operações.

Base SREVTRANS

Esta base foi construída da mesma forma que a base REVTRANS, mas aqui consideramos que temos strings com sinais e, ao aplicar uma reversão $\rho(i, j)$, podemos ter $i = j$. Os caracteres da string de origem têm apenas sinais positivos. Os caracteres da string de destino, por outro lado, podem possuir sinais negativos, pois as reversões aplicadas invertem os sinais dos caracteres afetados.

Base HARD

O objetivo desta base de dados é minimizar o número de adjacências comuns entre a string de origem e a string de destino. A redução no número de adjacências garante que o número de operações necessárias para transformar uma string em outra seja elevado. Nessa base, construímos novamente 10 conjuntos com 1000 pares de strings, cada um com um tamanho de string fixo. Os tamanhos das strings variam de 100 até 1000 em intervalos de 100. O seguinte processo foi utilizado para criação de cada par de strings de tamanho n :

1. Criamos duas strings aleatórias S e P , de tamanho $2 \lfloor \frac{n}{6} \rfloor$, utilizando os caracteres $(1, 2, \dots, \lfloor \frac{n}{6} \rfloor, 1, 2, \dots, \lfloor \frac{n}{6} \rfloor)$.
2. Para cada caractere S_i , com $1 \leq i \leq |S|$, adicionamos dois novos caracteres α_ℓ e α_r em S , antes e depois de S_i , respectivamente. Caso S_i seja a primeira ocorrência desse rótulo em S , temos $\alpha_\ell = \lfloor \frac{n}{3} \rfloor + 2S_i - 1$ e $\alpha_r = 2 \lfloor \frac{n}{3} \rfloor + 2S_i - 1$. Caso contrário, temos $\alpha_\ell = \lfloor \frac{n}{3} \rfloor + 2S_i$ e $\alpha_r = 2 \lfloor \frac{n}{3} \rfloor + 2S_i$.
3. Para cada caractere P_i , com $1 \leq i \leq |P|$, adicionamos dois novos caracteres α_ℓ e α_r em P , antes e depois de P_i , respectivamente. Caso P_i seja a primeira ocorrência desse rótulo em P , temos $\alpha_\ell = 2 \lfloor \frac{n}{3} \rfloor - 2P_i + 1$ e $\alpha_r = 3 \lfloor \frac{n}{3} \rfloor - 2P_i + 1$. Caso contrário, temos $\alpha_\ell = 2 \lfloor \frac{n}{3} \rfloor - 2P_i + 2$ e $\alpha_r = 3 \lfloor \frac{n}{3} \rfloor - 2P_i + 2$.
4. Para completar os n caracteres, adicionamos os caracteres $(3 \lfloor \frac{n}{3} \rfloor + 1, \dots, n)$, seguindo essa mesma ordem, no final das strings S e P .

Exemplo 12. O processo de construção da base HARD, para um par de strings de tamanho 20, a partir de duas strings aleatórias S e P obtidas no Passo 1.

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

Após a inclusão dos novos caracteres (em vermelho) nos passos 2 e 3, temos:

$$\begin{aligned} S &= (\textcolor{red}{11} \textcolor{red}{3} \textcolor{red}{17} \textcolor{red}{7} \textcolor{red}{1} \textcolor{red}{13} \textcolor{red}{12} \textcolor{red}{3} \textcolor{red}{18} \textcolor{red}{9} \textcolor{red}{2} \textcolor{red}{15} \textcolor{red}{10} \textcolor{red}{2} \textcolor{red}{16} \textcolor{red}{8} \textcolor{red}{1} \textcolor{red}{14}) \\ P &= (\textcolor{red}{9} \textcolor{red}{2} \textcolor{red}{15} \textcolor{red}{11} \textcolor{red}{1} \textcolor{red}{17} \textcolor{red}{7} \textcolor{red}{3} \textcolor{red}{13} \textcolor{red}{8} \textcolor{red}{3} \textcolor{red}{14} \textcolor{red}{12} \textcolor{red}{1} \textcolor{red}{18} \textcolor{red}{10} \textcolor{red}{2} \textcolor{red}{16}) \end{aligned}$$

Após a inclusão dos caracteres finais (em azul) no Passo 4, temos:

$$\begin{aligned} S &= (11 \textcolor{red}{3} \textcolor{red}{17} \textcolor{red}{7} \textcolor{red}{1} \textcolor{red}{13} \textcolor{red}{12} \textcolor{red}{3} \textcolor{red}{18} \textcolor{red}{9} \textcolor{red}{2} \textcolor{red}{15} \textcolor{red}{10} \textcolor{red}{2} \textcolor{red}{16} \textcolor{red}{8} \textcolor{red}{1} \textcolor{red}{14} \textcolor{blue}{19} \textcolor{blue}{20}) \\ P &= (\textcolor{red}{9} \textcolor{red}{2} \textcolor{red}{15} \textcolor{red}{11} \textcolor{red}{1} \textcolor{red}{17} \textcolor{red}{7} \textcolor{red}{3} \textcolor{red}{13} \textcolor{red}{8} \textcolor{red}{3} \textcolor{red}{14} \textcolor{red}{12} \textcolor{red}{1} \textcolor{red}{18} \textcolor{red}{10} \textcolor{red}{2} \textcolor{red}{16} \textcolor{blue}{19} \textcolor{blue}{20}) \end{aligned}$$

Base SHARD

Esta base foi gerada da mesma forma que a base HARD, mas foi escolhido um sinal aleatório para cada elemento de ambas as strings S e P .

3.10.2 Algoritmos para Permutações

As heurísticas propostas precisam de algoritmos para resolver as distâncias entre permutações. Como executamos os algoritmos várias vezes, estamos interessados em algoritmos rápidos que retornem boas soluções. Lembramos que, como estamos lidando com permutações, podemos renomear os elementos para que a permutação de destino esteja ordenada. Com essa modificação, podemos utilizar os algoritmos para ordenação de permutações.

O problema OR é o único para o qual conhecemos algoritmos exatos e eficientes. Em particular, utilizamos um algoritmo linear para esse problema [4]. Nos demais problemas utilizamos algoritmos que garantem uma fator de aproximação em relação a uma solução ótima.

Para escolher os demais algoritmos, realizamos alguns testes utilizando a base HARD (apenas com os tamanhos entre 100 e 500). Como esses algoritmos recebem permutações, aplicamos 10 mapeamentos aleatórios em cada string de origem e mapeamos as strings destino com o mapeamento padrão. Dessa forma, obtemos um conjunto de teste com 50000 pares de permutações.

Mostramos os resultados dos nossos testes comparativos nas tabelas 3.1, 3.2, 3.3 e 3.4. Em cada tabela indicamos, para cada conjunto de pares de strings de um mesmo tamanho, o valor mínimo (Min.), médio (Med.) e máximo (Max.) das distâncias encontradas, além do tempo médio (Tempo) em milissegundos para encontrar a distância entre um par de strings do conjunto.

Tabela 3.1: Comparação de algoritmos para o problema OR .

	1, 5 – OR [20]				2 – OR [39]			
Tamanho	Min.	Med.	Max.	Tempo	Min.	Med.	Max.	Tempo
100	91,00	95,62	99,00	7,01	71,00	79,34	88,00	0,93
200	188,00	193,87	198,00	18,87	136,00	151,39	167,00	2,59
300	289,00	296,88	300,00	33,37	213,00	233,27	253,00	5,53
400	389,00	395,20	399,00	41,26	314,00	342,44	362,00	10,99
500	486,00	492,67	498,00	55,03	368,00	392,28	416,00	13,50

Na Tabela 3.1, temos a comparação entre dois algoritmos para o problema OR . O primeiro é um algoritmo com fator de aproximação 1, 5 ($1, 5 - OR$), proposto por Christie [20]. O segundo é um algoritmo com fator de aproximação 2 ($2 - OR$), proposto por Kececioglu e Sankoff [39]. Podemos notar que o algoritmo $2 - OR$, embora tenha uma pior aproximação teórica, mostrou resultados práticos melhores que o algoritmo $1, 5 - OR$. Além disso, o tempo de execução médio do algoritmo $2 - OR$ foi menor que o do algoritmo $1, 5 - OR$. Por essa razão, decidimos utilizar o algoritmo $2 - OR$.

Na Tabela 3.2, temos a comparação entre dois algoritmos para o problema OT . O primeiro é um algoritmo com fator de aproximação 1, 375 ($1, 375 - OT$), proposto por Elias e Hartman [26]. O segundo é um algoritmo com fator de aproximação 1, 5 ($1, 5 - OT$), proposto por Hartman e Shamir [36], com a adaptação proposta por Feng e Zhu [27].

Tabela 3.2: Comparação de algoritmos para o problema OT .

	1, 375 – OT [26]				1, 5 – OT [36]			
Tamanho	Min.	Med.	Max.	Tempo	Min.	Med.	Max.	Tempo
100	46,00	48,39	51,00	6,59	47,00	51,03	56,00	0,68
200	95,00	98,21	101,00	17,41	99,00	104,72	112,00	1,34
300	148,00	149,98	152,00	33,26	152,00	160,30	168,00	2,31
400	196,00	198,27	201,00	40,62	205,00	214,34	223,00	3,51
500	244,00	247,83	251,00	52,36	256,00	266,61	283,00	4,23

Podemos notar que o algoritmo 1, 375 – OT mostrou resultados práticos melhores que o algoritmo 1, 5 – OT . Entretanto, ele apresenta um tempo de execução muito maior. Por essa razão, decidimos utilizar o algoritmo 1, 5 – OT .

Tabela 3.3: Comparação de algoritmos para o problema ORT .

	3 – ORT [56]				2k – ORT [51]			
Tamanho	Min.	Med.	Max.	Tempo	Min.	Med.	Max.	Tempo
100	48,00	52,15	58,00	0,93	57,00	71,72	85,00	6,81
200	98,00	103,81	110,00	2,72	116,00	144,27	159,00	17,85
300	151,00	157,28	166,00	5,69	170,00	213,76	245,00	31,80
400	202,00	207,11	215,00	9,71	236,00	286,34	316,00	40,11
500	249,00	256,96	264,00	13,62	292,00	357,64	408,00	50,59

Na Tabela 3.3, temos a comparação entre dois algoritmos para o problema ORT . O primeiro é um algoritmo com fator de aproximação 3 (3 – ORT), proposto por Walter e coautores [56]. O segundo é um algoritmo com fator de aproximação $2k$ ($2k$ – OR), proposto por Hahman e coautores [51], onde k é a aproximação para um algoritmo de decomposição de ciclos. No nosso caso utilizamos a decomposição de ciclos de Christie [20] onde $k = 1, 5$, resultando em um fator de aproximação 3. Como o algoritmo 3 – ORT é mais rápido e apresenta resultados melhores, ele foi escolhido para ser usado.

Tabela 3.4: Comparação de algoritmos para o problema ORT .

	3 – ORT [56]				2 – ORT [51]			
Tamanho	Min.	Med.	Max.	Tempo	Min.	Med.	Max.	Tempo
100	55,00	59,94	66,00	0,87	62,00	74,81	85,00	7,37
200	106,00	113,67	123,00	2,51	131,00	147,96	164,00	17,72
300	158,00	165,17	174,00	5,05	187,00	225,21	250,00	32,51
400	208,00	214,65	220,00	8,14	265,00	303,70	337,00	40,61
500	257,00	265,85	275,00	11,78	306,00	375,18	407,00	52,13

Os mesmos algoritmos para reversão e transposição do caso sem sinais podem ser adaptados para o caso com sinais. A adaptação do algoritmo 3 – ORT (3 – ORT) continua com a mesma aproximação.

Como a decomposição em ciclos para o caso com sinais é única, a adaptação do algoritmo $2k - ORT$ ($2 - O\bar{R}T$) possui um fator de aproximação 2. Na Tabela 3.4, podemos ver que o algoritmo $3 - O\bar{R}T$ apresentou resultados melhores em um tempo menor, logo escolhemos esse algoritmo para os nossos testes.

3.10.3 Ajuste de Parâmetros

Inicialmente, tivemos que decidir quantos mapeamentos utilizar em cada heurística. Como o tempo de execução depende do número de mapeamentos usados e queremos comparar os resultados obtidos, utilizamos a mesma quantidade de mapeamentos para todas as heurísticas e problemas. Decidimos permitir que esse valor varie com o tamanho das strings, pois com strings pequenas podemos obter bons resultados com menos mapeamentos. Para escolher esse valor, verificamos a distância de reversão obtida pela heurística MA na base REV (apenas com as strings de tamanho 500) com diferentes quantidades de mapeamentos. No gráfico da Figura 3.9, observamos que até $10|S|$ temos uma melhora significativa com o aumento dos mapeamentos, mas após esse ponto temos um ganho reduzido ao aumentar a quantidade de mapeamentos. Considerando esses testes, decidimos utilizar $10|S|$ mapeamentos.

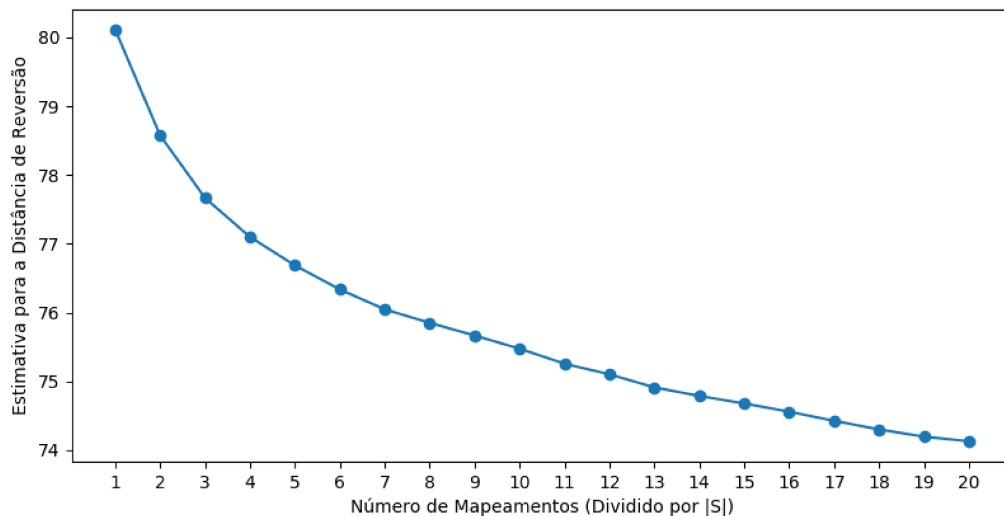


Figura 3.9: Resultado da heurística MA com diferentes números de mapeamentos para o conjunto com strings de tamanho 500 da base REV.

Para ajustar os demais parâmetros das heurísticas, realizamos testes em um subconjunto das nossas bases de dados, onde consideramos várias configurações de parâmetros distintas. Utilizamos as bases REV, SREV, TRANS, REVTRANS e SREVTRANS, mas apenas 300 pares de strings de cada base, sendo 100 pares de tamanho 400, 100 pares de tamanho 500 e 100 pares de tamanho 600. A seguir apresentamos os parâmetros escolhidos para cada heurística. Note que as heurísticas Mapeamentos Aleatórios e Separação não apresentam mais nenhum parâmetro para ser ajustado.

A seguir apresentamos, para cada heurística, os gráficos que mostram as escolhas dos parâmetros considerando o problema DT e, para melhorar a visualização, colocamos apenas parte dos valores testados. Gráficos similares para os ajustes dos parâmetros considerando os demais problemas se encontram no Anexo A.

Busca Local (BL)

Na heurística BL ainda é necessário ajustar os parâmetros c e ℓ . Testamos ambos os parâmetros no conjunto $\{10, 20, \dots, 100\}$. Para o parâmetro ℓ , notamos que nos problemas envolvendo o evento de transposição (DT , DRT e $D\bar{R}T$) o melhor valor foi $\ell = 10$ e, para verificar se é vantajoso diminuir o valor desse parâmetro, testamos também com o conjunto $\{1, 2, \dots, 10\}$. Na Figura 3.10, podemos visualizar parte dos valores testados para o problema DT .

A seguir apresentamos os melhores parâmetros encontrados para cada problema:

- DR : $c = 90$ e $\ell = 30$;
- $D\bar{R}$: $c = 40$ e $\ell = 40$;
- DT : $c = 60$ e $\ell = 7$;
- DRT : $c = 90$ e $\ell = 9$;
- $D\bar{R}T$: $c = 30$ e $\ell = 10$.

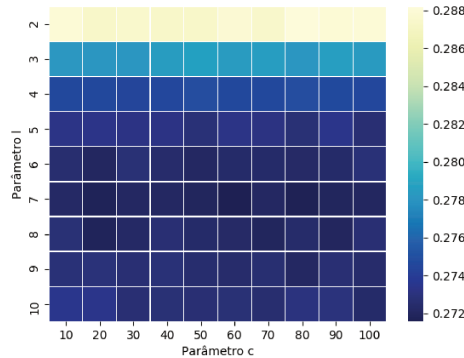


Figura 3.10: Ajuste dos parâmetros c e ℓ da heurística de Busca Local no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

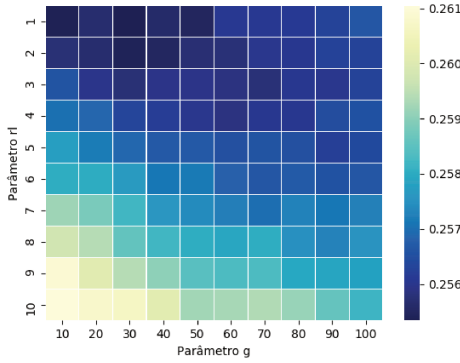
GRASP

Na heurística GRASP ainda é necessário ajustar os parâmetros c , k , g e r_ℓ . Inicialmente, fixamos $c = 100$ e $k = 100$, e testamos os parâmetros g e r_ℓ no conjunto $\{10, 20, \dots, 100\}$. Nos problemas onde o melhor valor para um desses parâmetros foi $\ell = 10$, verificamos se é vantajoso diminuir esse parâmetro testando também com o conjunto $\{1, 2, \dots, 10\}$. Com os parâmetros g e r_ℓ definidos, testamos os parâmetros c e k

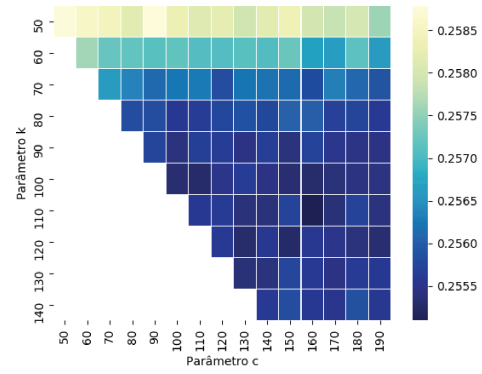
no conjunto $\{10, 20, \dots, 250\}$, respeitando a restrição $c \geq k$ para garantir que podemos popular a RCL no início da heurística. Na Figura 3.11, podemos visualizar parte dos valores testados para o problema *DT*.

A seguir apresentamos os melhores parâmetros encontrados para cada problema:

- *DR*: $c = 200$, $k = 150$, $g = 3$ e $r_\ell = 30$;
- *D \bar{R}* : $c = 200$, $k = 150$, $g = 3$ e $r_\ell = 30$;
- *DT*: $c = 160$, $k = 110$, $g = 30$ e $r_\ell = 1$;
- *DRT*: $c = 140$, $k = 120$, $g = 7$ e $r_\ell = 2$;
- *D $\bar{R}T$* : $c = 120$, $k = 90$, $g = 9$ e $r_\ell = 1$.



(a) Ajuste dos parâmetros g e r_ℓ .



(b) Ajuste dos parâmetros c e k .

Figura 3.11: Ajuste dos parâmetros da heurística GRASP no problema *DT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

Algoritmo Genético (AG)

Na heurística AG ainda é necessário ajustar os parâmetros c , k , t_c e t_m . Inicialmente, fixamos $c = 100$ e $k = 50$, e testamos os parâmetros t_c e t_m . Para o parâmetro t_c , escolhemos valores dependentes da quantidade d de rótulos duplicados, e o conjunto testado para esse parâmetro foi $\{[0, 1d], [0, 2d], \dots, d\}$. Para o parâmetro t_m investigamos o conjunto $\{1, 2, \dots, 20\}$. Com os parâmetros t_c e t_m definidos, testamos os parâmetros c e k no conjunto $\{10, 20, \dots, 100\}$, respeitando a restrição $c \geq k$ para garantir que a população inicial tenha mapeamentos suficientes para a primeira seleção. Na Figura 3.12, podemos visualizar parte dos valores testados para o problema *DT*.

A seguir, apresentamos os melhores parâmetros encontrados para cada problema:

- *DR*: $c = 90$, $k = 50$, $t_c = [0, 4d]$ e $t_m = 2$;
- *D \bar{R}* : $c = 10$, $k = 10$, $t_c = [0, 4d]$ e $t_m = 1$;

- DT : $c = 90$, $k = 40$, $t_c = [0, 5d]$ e $t_m = 8$;
- DRT : $c = 50$, $k = 40$, $t_c = [0, 6d]$ e $t_m = 7$;
- $D\bar{R}T$: $c = 40$, $k = 40$, $t_c = [0, 5d]$ e $t_m = 15$.

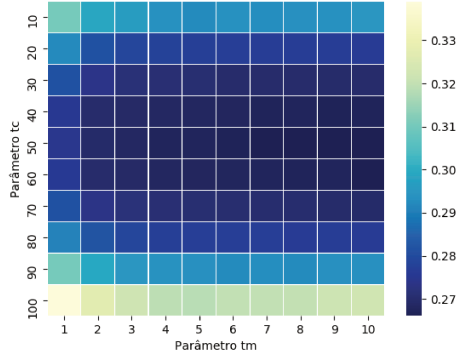
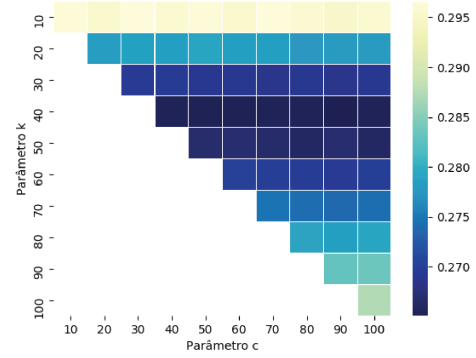
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura 3.12: Ajuste dos parâmetros da heurística Algoritmo Genético no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

Busca Tabu (BT)

Na heurística BT ainda é necessário ajustar os parâmetros t e ℓ . Testamos t no conjunto $\{1, 2, \dots, 10\}$ e ℓ no conjunto $\{10, 20, \dots, 100\}$. Na Figura 3.13, podemos visualizar parte dos valores testados para o problema DT .

A seguir, apresentamos os melhores parâmetros encontrados para cada problema:

- DR : $t = 1$ e $\ell = 30$;
- $D\bar{R}$: $t = 1$ e $\ell = 50$;
- DT : $t = 1$ e $\ell = 20$;
- DRT : $t = 1$ e $\ell = 20$;
- $D\bar{R}T$: $t = 2$ e $\ell = 20$.

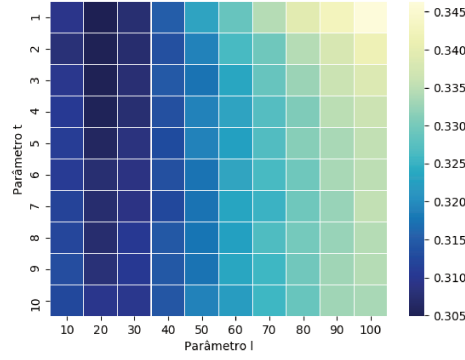


Figura 3.13: Ajuste dos parâmetros ℓ e t da heurística de Busca Tabu no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

Simulated Annealing (SA)

Na heurística SA escolhemos $k = 1$, pois esse valor está de acordo com a dimensão dos valores com os quais estamos trabalhando, e ajustamos os parâmetros t_0 , i e s . Inicialmente, fixamos $i = 10$ e testamos o parâmetro t_0 no conjunto $\{1, 2, \dots, 10\}$ e o parâmetro s no conjunto $\{0, 90, 0, 91, \dots, 0, 99\}$. Com os parâmetros t_0 e s definidos, testamos o parâmetro i no conjunto $\{1, 2, \dots, 10\}$. Na Figura 3.15, podemos visualizar parte dos valores testados para o problema DT .

A seguir, apresentamos os melhores parâmetros encontrados para cada problema:

- DR : $t_0 = 2$, $s = 0,98$ e $i = 6$;
- $D\bar{R}$: $t_0 = 2$, $s = 0,98$ e $i = 5$;
- DT : $t_0 = 7$, $s = 0,94$ e $i = 9$;
- DRT : $t_0 = 1$, $s = 0,99$ e $i = 6$;
- $D\bar{R}T$: $t_0 = 1$, $s = 0,98$ e $i = 10$.

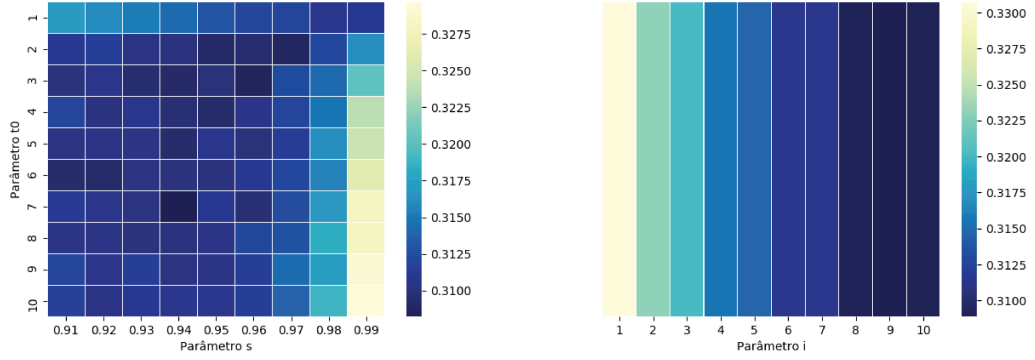
(a) Ajuste dos parâmetros s e t_0 .(b) Ajuste do parâmetro i .

Figura 3.14: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema *DT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

Busca Cuco (BC)

Na heurística BC ainda é necessário ajustar os parâmetros c , g e p . Inicialmente, fixamos $p = 50\%$ e testamos o parâmetro c no conjunto $\{1, 2, \dots, 10\}$ e o parâmetro g no conjunto $\{10, 20, \dots, 100\}$ (adicionando o conjunto $\{1, 2, \dots, 10\}$ caso o melhor valor encontrado seja 10). Com os parâmetros c e g definidos, testamos o parâmetro p no conjunto $\{10\%, 20\%, \dots, 100\%\}$. Na Figura 3.14, podemos visualizar parte dos valores testados para o problema *DT*.

A seguir, apresentamos os melhores parâmetros encontrados para cada problema:

- *DR*: $c = 3$, $g = 30$ e $p = 40\%$;
- $D\bar{R}$: $c = 4$, $g = 90$ e $p = 60\%$;
- *DT*: $c = 6$, $g = 3$ e $p = 10\%$;
- *DRT*: $c = 3$, $g = 60$ e $p = 20\%$;
- $D\bar{R}T$: $c = 3$, $g = 80$ e $p = 20\%$.

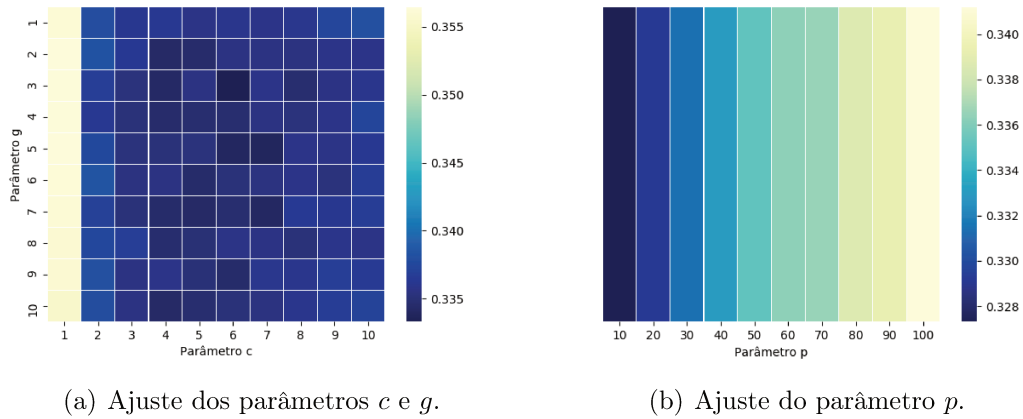


Figura 3.15: Ajuste dos parâmetros da heurística de Busca Cuco no problema *DT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

3.10.4 Resultados

Nesta seção, apresentamos os resultados dos testes experimentais para cada uma das bases de dados. Os valores apresentados são as médias das distâncias encontradas considerando as instâncias de cada conjunto de dados. Nas tabelas referentes as bases de dados geradas com um número específico de operações adicionamos uma coluna OP com o número de operações utilizadas para gerar cada conjunto de strings.

A Tabela 3.5 apresenta as médias das distâncias obtidas para o problema *DR* na base REV. Podemos ver que escolher os mapeamentos de forma aleatória resultou em valores piores que aplicar o algoritmo SOAR. Apesar disso, todas as heurísticas com estratégias mais sofisticadas obtiveram resultados melhores que o algoritmo SOAR.

Dentre as heurísticas, a que obteve os melhores resultados foi a GRASP, que apresentou valores menores ou iguais aos das demais em todos os conjuntos de instâncias. As heurísticas SA e AG obtiveram resultados próximos aos da heurística GRASP, sendo que SA superou a heurística AG nos conjuntos com strings de tamanhos 100 até 500 e AG superou a heurística SA nos demais conjuntos. A heurística BL obteve resultados próximos da SA, tendo resultados iguais para os conjuntos com strings de tamanhos 100 até 300 e sendo um pouco pior nos demais conjuntos. Vale notar que, para conjuntos com strings de tamanho 100 e 200, as heurísticas BL, SA e BT obtiveram resultados iguais aos da heurística GRASP. Nos conjuntos com strings de tamanhos maiores que 200, a heurística BT encontrou valores um pouco maiores que a BL. Os piores resultados foram obtidos pelas heurísticas BC e Sep, sendo que a BC obteve resultados melhores que a Sep para conjuntos com strings de tamanhos 200 e 300, enquanto o Sep obteve resultados melhores nos demais conjuntos.

Comparando os valores da heurística GRASP com os da coluna OP, notamos que ela encontrou valores bem próximos do número de operações aplicadas para gerar a string de destino. Com conjuntos de strings de tamanhos maiores, a diferença desses valores aumenta, mas ela não passa de 6.

Tabela 3.5: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema DR na base REV.

Tamanho	OP	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	25	32,62	24,19	24,19	24,21	24,19	24,23	24,19	24,22	29,59
200	50	75,42	49,69	49,69	49,75	49,69	49,90	49,69	50,01	61,37
300	75	120,88	75,40	75,39	75,49	75,45	76,29	75,40	76,41	94,12
400	100	167,27	101,00	100,92	101,10	101,28	103,79	100,98	103,30	126,31
500	125	214,54	126,74	126,51	126,80	127,59	132,91	126,67	130,91	158,95
600	150	262,36	152,67	152,12	152,44	154,91	166,18	152,57	158,95	191,18
700	175	310,53	179,43	178,00	178,38	183,84	202,38	179,20	188,00	224,10
800	200	359,49	207,65	203,88	204,55	216,33	244,43	206,76	217,48	256,40
900	225	408,32	239,27	229,63	230,52	250,48	285,93	236,35	247,31	289,05
1000	250	457,45	274,43	255,94	256,94	293,64	332,75	270,99	277,69	321,87

A Tabela 3.6 apresenta as médias das distâncias obtidas para o problema $D\bar{R}$ na base SREV. Novamente, escolher os mapeamentos de forma aleatória resultou em valores piores que aplicar o algoritmo SOAR, e as demais heurísticas são bem melhores que a heurística MA. Todas as heurísticas mais sofisticadas tiveram resultados muito próximos entre si e muito próximos dos resultados do algoritmo SOAR. Com exceção das heurísticas Sep e MA, os valores foram praticamente iguais. As heurísticas GRASP, AG e SA apresentaram os mesmos valores e foram as melhores em todos os conjuntos, com exceção do conjunto com strings de tamanho 100, onde a heurística SA foi a melhor, e do conjunto com strings de tamanho 1000, onde a heurística SA foi um pouco pior que as heurísticas GRASP e AG.

Observando a coluna OP, notamos que os valores obtidos pelas heurísticas mais sofisticadas e pelo algoritmo SOAR ficaram bem próximos do número de operações aplicadas.

Tabela 3.6: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema $D\bar{R}$ na base SREV.

Tamanho	OP	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	25	33,77	24,71	24,71	24,71	24,71	24,71	24,11	24,72	24,73
200	50	75,64	49,73	49,73	49,73	49,73	49,74	49,73	49,84	49,74
300	75	119,07	74,74	74,74	74,74	74,74	74,75	74,74	75,31	74,75
400	100	163,34	99,71	99,71	99,71	99,72	99,72	99,71	101,18	99,73
500	125	208,40	124,74	124,73	124,73	124,80	124,74	124,73	127,66	124,74
600	150	254,06	149,73	149,72	149,72	149,83	149,73	149,72	154,88	149,74
700	175	299,63	174,76	174,74	174,74	174,99	174,75	174,74	182,02	174,76
800	200	346,03	199,80	199,74	199,74	200,23	199,74	199,74	210,06	199,75
900	225	392,02	224,80	224,72	224,72	225,47	224,73	224,72	238,76	224,73
1000	250	438,65	249,81	249,73	249,73	250,70	249,74	249,75	267,23	249,74

A Tabela 3.7 apresenta as médias das distâncias obtidas para o problema *DT* na base TRANS. Como nos resultados anteriores, a heurística MA resultou em valores piores que o algoritmo SOAR e que as demais heurísticas. A heurística GRASP apresentou os melhores resultados em todos os conjuntos, obtendo valores consideravelmente menores que os demais, principalmente em conjuntos com strings de tamanhos maiores que 500.

Desconsiderando a heurística GRASP, o algoritmo SOAR ganhou das demais heurísticas em strings de tamanhos 800, 900 e 1000. Além disso, a heurística AG ganhou das demais em conjuntos com strings de tamanhos de 300 e 400. A heurística Sep também apresentou bons resultados, embora tenha perdido do algoritmo SOAR em strings com tamanhos superiores a 500. A heurística BL ainda apresentou bons resultados até strings de tamanho 400, mas piorou com o aumento do tamanho, perdendo do algoritmo SOAR nos demais conjuntos. As demais heurísticas (BT, BC e SA) apresentaram valores maiores em comparação com o algoritmo SOAR em quase todos os conjuntos. Dentre essas heurísticas, BC apresentou o pior resultado em conjuntos com strings de tamanhos superiores a 600 e SA nos demais conjuntos.

Comparando os valores com a coluna OP, vemos que as heurísticas mais sofisticadas e o algoritmo SOAR apresentaram resultados próximos do número de operações aplicadas em conjuntos com strings de tamanhos menores. Com o aumento no tamanho das strings os valores obtidos se afastam do número de operações aplicadas.

Tabela 3.7: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema *DT* na base TRANS.

Tamanho	OP	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	25	28,79	23,26	23,22	23,73	23,25	26,07	23,59	23,39	24,73
200	50	64,72	49,51	48,58	49,99	50,96	58,07	51,85	49,56	52,30
300	75	101,87	77,38	74,56	76,91	82,86	92,56	84,04	77,20	80,22
400	100	139,80	106,29	100,78	104,29	116,86	127,58	123,89	105,44	107,99
500	125	178,32	136,24	127,84	132,59	152,42	164,43	161,34	135,10	136,05
600	150	217,38	166,95	155,12	161,78	188,93	201,26	200,17	165,58	164,72
700	175	256,21	198,04	182,74	191,40	226,04	238,99	239,02	196,14	193,09
800	200	295,46	229,57	210,27	221,47	263,44	276,96	277,18	227,41	221,38
900	225	334,74	261,58	238,12	251,56	301,61	315,44	317,56	259,17	249,48
1000	250	374,35	293,87	266,38	282,19	339,38	353,67	357,63	291,30	278,53

A Tabela 3.8 apresenta as médias das distâncias obtidas para o problema *DRT* na base REVTRANS. Novamente, a heurística MA foi a pior de todas e perdeu do algoritmo SOAR. As outras heurísticas que apresentaram valores maiores que os do algoritmo SOAR foram a BT, BC e SA, em conjuntos com strings de tamanhos acima de 300 (no caso da BC isso também ocorre com strings do tamanho 300).

A heurística GRASP apresentou novamente os melhores resultados em todos os conjuntos. Desconsiderando a heurística GRASP, a heurística BL apresentou os melhores resultados no conjunto de strings de tamanho 100. A heurística Sep apresentou os melhores valores nos demais conjuntos, seguida pela heurística AG com resultados maiores.

Comparando os valores com a coluna OP, notamos que as melhores heurística (BL, GRASP, AG e Sep) conseguiram obter valores menores que o número de operações aplicadas em todos os conjuntos de dados.

Tabela 3.8: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema *DRT* na base REVTRANS.

Tamanho	OP	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	25	27,94	22,70	22,69	23,39	22,79	24,32	22,92	22,83	27,06
200	50	59,87	46,21	45,30	46,84	47,74	52,14	48,45	45,83	52,51
300	75	92,11	70,11	67,86	69,97	74,10	80,75	75,14	68,75	76,73
400	100	124,10	93,69	89,92	92,66	100,80	109,45	103,71	91,19	100,26
500	125	156,14	117,62	112,38	115,89	127,62	138,53	133,74	114,15	123,67
600	150	188,21	141,08	134,48	139,03	154,22	167,37	164,87	136,61	146,73
700	175	220,62	164,82	156,88	162,83	181,43	196,64	196,34	159,61	170,23
800	200	252,72	188,30	178,82	186,29	208,11	225,85	229,09	182,12	193,05
900	225	284,99	212,31	201,62	210,18	235,77	254,46	262,41	205,16	216,27
1000	250	317,06	235,62	223,81	234,37	261,94	284,11	294,30	227,85	239,12

A Tabela 3.9 apresenta as médias das distâncias obtidas para o problema *DRT* na base SREVTRANS. Novamente, a heurística MA foi a pior de todas, obtendo valores maiores que os do algoritmo SOAR. As heurísticas BC e SA também encontraram valores maiores que os do algoritmo SOAR em todos os conjuntos. As heurísticas BL, BT e AG apresentaram valores maiores que os do algoritmo SOAR em alguns conjuntos com tamanhos de strings mais elevados (tamanhos acima de 600, 200 e 700, respectivamente).

A heurística GRASP apresentou novamente os melhores resultados em todos os conjuntos. Desconsiderando a heurística GRASP, a heurística Sep apresentou os melhores valores nos conjuntos com strings de tamanhos acima de 100 e a heurística BL apresentou os melhores valores no conjunto de strings com tamanho 100.

Novamente, as melhores heurística (BL, GRASP e Sep) conseguiram obter valores menores que o número de operações aplicadas em todos os conjuntos de dados.

Tabela 3.9: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema $D\bar{R}T$ na base SREVTRANS.

Tamanho	OP	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	25	30,11	24,18	24,14	25,32	24,36	25,72	33,91	24,35	26,33
200	50	62,08	47,49	46,27	48,36	49,72	54,09	70,99	46,95	50,54
300	75	93,86	70,89	67,97	70,43	76,04	82,06	107,70	69,11	73,24
400	100	125,66	94,02	89,45	92,39	102,39	110,11	145,72	91,03	95,26
500	125	157,73	117,21	111,10	114,63	128,97	138,58	185,17	113,12	117,52
600	150	189,36	140,18	132,50	137,08	155,07	167,00	224,40	134,91	138,91
700	175	221,48	163,20	154,17	160,23	181,43	194,67	264,86	157,03	160,65
800	200	253,37	185,74	175,32	183,31	207,31	223,14	304,96	178,99	181,89
900	225	285,53	208,96	196,95	206,85	233,71	252,06	346,59	201,09	203,42
1000	250	317,69	231,52	218,76	230,63	260,38	279,53	389,67	223,21	224,73

A Tabela 3.10 apresenta as médias das distâncias obtidas para o problema DR na base HARD. Novamente a heurística MA apresentou os maiores valores, seguida pela heurística Sep que apresentou valores maiores que os do algoritmo SOAR em todos os conjuntos com strings de tamanhos superiores a 300. O algoritmo SOAR apresentou os melhores resultados para conjuntos com strings de tamanhos 400, 700 e 1000. As demais heurísticas apresentam resultados muito próximos, sendo que a heurística SA apresentou os melhores resultados em todos os conjuntos, com exceção dos conjuntos com strings de tamanhos 700 e 1000.

Tabela 3.10: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema DR na base HARD.

Tamanho	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	66,45	62,76	62,76	62,76	62,76	62,77	62,76	62,76	62,89
200	131,59	126,98	126,91	126,93	126,97	126,97	126,88	127,07	127,30
300	208,02	196,59	196,55	196,57	196,54	196,59	196,47	196,80	196,85
400	311,12	261,56	261,56	261,64	261,56	262,07	261,56	273,30	261,56
500	355,77	326,18	326,26	326,42	326,13	326,17	326,12	337,21	326,36
600	435,57	395,92	396,07	396,48	395,90	395,96	395,90	416,95	396,14
700	567,73	461,49	461,13	467,93	464,02	479,84	461,07	527,12	460,95
800	586,28	525,74	526,01	529,22	525,77	525,87	525,70	568,78	525,83
900	666,89	595,56	596,03	602,19	595,55	596,01	595,50	650,76	595,61
1000	827,51	670,57	669,90	698,76	696,11	723,71	661,72	786,83	660,65

A Tabela 3.11 apresenta as médias das distâncias obtidas para o problema $D\bar{R}$ na base SHARD. Novamente a heurística MA apresentou os maiores valores, embora tenha ganho do algoritmo SOAR para strings de tamanho 100. As demais heurísticas obtiveram valores melhores que o algoritmo SOAR.

A heurística SA apresentou os melhores resultados em todos os conjuntos, com exceção dos conjuntos com strings de tamanhos 300 (onde o melhor resultado foi da heurística Sep) e 400 (onde o melhor resultado foi da heurística GRASP). Desconsiderando a heurística SA, a heurística BL obteve o melhor resultado em strings de tamanho 100 e a heurística Sep obteve o melhor resultado em strings de tamanhos 200 e 300. Nos demais conjuntos, a heurística GRASP obteve o melhor resultado. A heurística AG apresentou resultados próximos das melhores heurísticas, embora um pouco piores. As heurísticas BT e BC apresentaram os piores resultados dentre as heurísticas mais sofisticadas, sendo que a BT superou os resultados da BC em todos os conjuntos.

Tabela 3.11: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema $D\bar{R}$ na base SHARD.

Tamanho	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	90,41	89,26	89,27	89,38	89,40	89,39	89,26	89,30	91,84
200	182,72	178,50	178,28	178,59	178,86	179,34	178,21	178,26	182,82
300	279,76	272,19	271,43	272,10	272,61	274,02	271,32	271,30	278,40
400	380,97	370,36	368,96	370,13	370,96	373,83	368,98	369,85	379,13
500	470,18	456,73	455,10	456,44	457,10	460,74	454,79	458,48	465,16
600	568,26	552,11	550,06	551,44	552,60	557,32	549,53	555,97	561,42
700	674,14	656,12	653,42	655,54	657,32	663,45	653,04	660,18	668,23
800	759,33	737,82	735,36	737,20	738,86	744,93	734,37	746,41	748,47
900	857,55	833,76	831,07	832,97	835,02	841,72	829,56	844,82	844,77
1000	968,05	944,49	940,48	943,15	946,65	954,39	938,84	952,89	957,79

A Tabela 3.12 apresenta as médias das distâncias obtidas para o problema DT na base HARD. A heurística MA continua sendo a pior, mas nessa base ele apresentou resultados melhores que o algoritmo SOAR em todos os conjuntos. As demais heurísticas obtiveram valores bem próximos e melhores que os da MA.

Nos conjuntos com strings de tamanhos acima de 200, a heurística Sep foi a melhor, seguida de perto pela heurística SA, que apresenta o melhor resultado para strings de tamanho 200. A heurística AG apresentou os melhores resultados para strings de tamanho 100 e nos demais conjuntos ficou bem próxima das heurísticas Sep e SA. Os resultados da heurística BC foram um pouco piores que os da AG, ambas apresentaram o mesmo valor em strings de tamanho 100. As heurísticas GRASP e BL encontraram valores praticamente iguais aos das anteriores, a heurística BL é melhor que a GRASP em strings de tamanhos 100, 200 e 300, a GRASP é melhor nos demais conjuntos. A heurística BT apresentou os piores resultados dentre as heurísticas mais sofisticadas, mas os valores continuam próximos das demais.

Tabela 3.12: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema *DT* na base HARD.

Tamanho	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	48,55	48,57	48,54	48,53	48,66	48,53	48,54	48,58	50,46
200	98,42	98,29	98,03	98,02	98,48	98,10	98,00	98,03	102,24
300	153,08	152,56	152,52	152,24	152,97	152,31	152,14	152,13	158,99
400	204,30	203,02	203,10	202,44	203,30	202,68	202,22	202,21	211,35
500	255,05	253,54	253,55	252,65	253,91	252,93	252,44	252,17	263,14
600	310,40	308,77	309,23	307,71	309,20	308,16	307,50	307,17	319,98
700	363,42	360,14	361,52	359,50	360,58	360,13	358,88	358,73	372,53
800	413,69	410,93	411,77	409,60	411,31	410,27	409,25	408,69	424,25
900	469,23	466,34	467,87	464,99	466,89	465,60	464,56	464,05	481,22
1000	523,80	518,93	521,55	518,31	519,49	519,04	517,21	516,97	533,91

A Tabela 3.13 apresenta as médias das distâncias obtidas para o problema *DRT* na base HARD. A heurística MA continua sendo a pior, mas apresentou resultados melhores que o algoritmo SOAR em todos os conjuntos. As demais heurísticas obtiveram valores melhores que os da MA.

Em todos os conjuntos, a melhor heurística foi a BL. Com relação as demais heurísticas, a heurística Sep foi melhor nos conjuntos com strings de tamanhos de 100 até 400, a heurística AG foi a melhor nos conjuntos com tamanhos de 500 até 900 e a heurística GRASP foi a melhor no conjunto com strings de tamanho 1000. As heurísticas BT, BC e SA, foram piores, mas apresentaram resultados bem próximos dos resultados das demais. Dentre essas, BT foi a melhor heurística considerando todas as strings com tamanhos maiores que 100.

Tabela 3.13: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema *DRT* na base HARD.

Tamanho	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	47,38	46,81	47,08	47,23	47,37	47,16	47,02	46,94	50,04
200	96,76	95,59	96,19	96,31	96,03	96,30	96,11	95,77	99,35
300	149,89	148,50	149,46	149,42	149,30	149,54	149,40	148,75	152,83
400	198,90	196,60	197,71	197,76	197,96	198,32	198,25	197,26	201,85
500	248,13	245,90	246,93	246,76	247,28	247,52	247,52	246,91	250,93
600	301,10	298,84	299,98	299,78	300,50	300,64	300,74	300,28	304,31
700	349,92	346,80	347,86	347,85	349,19	349,34	349,42	348,64	353,10
800	399,06	396,16	397,24	397,04	398,20	398,40	398,40	398,01	402,25
900	451,89	449,08	450,31	450,06	451,39	451,45	451,62	451,18	455,32
1000	500,56	496,95	497,85	497,89	500,00	500,00	500,21	499,47	503,97

A Tabela 3.14 apresenta as médias das distâncias obtidas para o problema $D\bar{R}T$ na base SHARD. Novamente, a heurística MA é a pior, mas apresenta resultados melhores que o algoritmo SOAR em todos os conjuntos. As demais heurísticas obtiveram valores melhores que os da MA. A melhor heurística foi a BL e as demais apresentaram quase os mesmos resultados.

Tabela 3.14: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR para o problema $D\bar{R}T$ na base SHARD.

Tamanho	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
100	53,55	52,90	53,24	53,33	53,06	53,37	53,17	53,11	59,77
200	104,84	103,90	104,53	104,49	104,24	104,55	104,34	104,16	112,90
300	156,05	154,86	155,71	155,55	155,43	155,70	155,50	155,25	164,22
400	206,07	204,68	205,71	205,54	205,53	205,59	205,65	205,41	213,25
500	256,71	254,81	256,25	255,96	255,96	255,90	256,07	255,91	264,96
600	307,55	305,49	307,14	306,85	306,90	306,70	307,05	306,93	315,52
700	357,26	354,94	356,85	356,55	356,70	356,36	356,80	356,74	363,45
800	407,78	404,98	407,14	406,77	407,02	406,55	407,25	407,02	415,81
900	458,57	455,58	458,01	457,49	457,81	457,29	458,13	457,90	466,50
1000	507,96	504,97	507,42	507,10	507,41	506,68	507,69	507,41	512,97

A seguir, apresentamos uma comparação dos tempos necessários para resolver as instâncias das bases REV, SREV, TRANS, REVTRANS e SREVTRANS. Os valores indicados correspondem ao tempo médio em segundos para resolver uma instância do conjunto considerado.

As tabelas 3.15, 3.16 e 3.17 apresentam os tempos necessários para resolver as instâncias de tamanhos 100, 500 e 1000, respectivamente. Podemos notar um aumento considerável no tempo de execução com o aumento no tamanho das instâncias. É possível perceber que o custo para obter resultados melhores que o algoritmo SOAR é um aumento no tempo de execução. Um fato interessante é que a heurística BL apresentou um tempo de execução menor que o das demais heurísticas na maioria dos casos, o que pode compensar o fato dela não ter obtido os melhores resultados em geral. O tempo de execução das heurísticas GRASP e AG, por outro lado, são os mais altos na maioria dos casos.

Tabela 3.15: Tempo médio, em segundos, necessário para resolver uma instância de tamanho 100 das bases de dados indicadas.

Base de Dados	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
REV	0,80	0,55	1,01	0,91	0,48	0,73	0,63	0,64	0,04
SREV	0,40	0,37	1,13	1,68	0,40	0,48	0,64	0,39	0,03
TRANS	0,98	0,82	1,33	1,20	0,85	0,75	0,92	0,95	0,03
REVTRANS	0,95	0,66	1,62	1,12	0,71	0,97	0,67	0,80	0,04
SREVTRANS	0,84	0,65	1,65	1,01	0,63	0,75	0,61	0,73	0,04

Tabela 3.16: Tempo médio, em segundos, necessário para resolver uma instância de tamanho 500 das bases de dados indicadas.

Base de Dados	MA	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
REV	72,70	37,37	48,00	50,72	40,50	57,65	35,58	50,58	0,37
SREV	9,33	9,38	18,49	70,23	9,24	12,86	9,82	9,74	0,34
TRANS	28,34	26,58	30,77	31,81	26,75	29,90	26,71	26,28	0,32
REVTRANS	66,26	48,70	62,64	59,03	53,22	64,05	52,63	54,38	0,39
SREVTRANS	54,46	41,69	53,62	50,59	44,27	54,45	44,64	44,98	0,37

Tabela 3.17: Tempo médio, em segundos, necessário para resolver uma instância de tamanho 1000 das bases de dados indicadas.

Base de Dados	Map	BL	GRASP	AG	BT	BC	SA	Sep	SOAR
REV	517,07	310,97	290,32	306,57	358,80	441,38	303,09	370,08	1,39
SREV	57,51	56,31	101,52	374,01	55,27	74,57	57,31	56,02	1,39
TRANS	131,79	120,55	131,21	137,73	125,02	136,46	126,34	122,45	1,29
REVTRANS	446,78	324,41	321,49	348,84	359,93	412,19	369,58	362,84	1,34
SREVTRANS	362,36	265,50	266,46	289,13	297,79	342,60	305,65	297,30	1,38

Considerando os resultados obtidos, vemos que em cada cenário uma heurística diferente é a mais indicada. A heurística GRASP é a melhor nas bases específicas de cada problema, mas não foi a melhor nas bases difíceis e apresenta um tempo de execução maior do que as demais. De forma similar, a heurística AG apresentou resultados consistentemente bons em todas as bases, mas tem um tempo de execução pior que as demais. A heurística BL pode não ter os melhores resultados, mas apresentou um tempo de execução menor que o das demais na maior parte das bases. A heurística SA também tem um tempo de execução mais baixo e apresentou bons resultados nas bases difíceis, mas não foi muito bem nas outras bases. As demais heurísticas também apresentam bons resultados em alguns casos, mas não tão bons em outros.

Capítulo 4

Heurísticas para Genes Multiplicados

Neste capítulo, generalizamos as heurísticas apresentadas no Capítulo 3, o que nos permite remover a restrição no número de cópias de cada gene. O foco é nas heurísticas GRASP, Algoritmo Genético e Busca Local, que apresentaram bons resultados nos testes com genes duplicados. Também generalizamos a heurística de Mapeamentos Aleatórios, utilizada como referência para comparação com as outras heurísticas. Note que, embora não apresentemos explicitamente as generalizações das outras heurísticas introduzidas no Capítulo 3, tais generalizações podem ser facilmente derivadas das ideias apresentadas neste capítulo.

Também discutimos variações do problema de Partição de Strings Comuns Mínima (MCSP), que são relacionadas aos problemas de distância de rearranjo. Ao final deste capítulo, mostramos que ao combinar as heurísticas generalizadas com os problemas de partição, conseguimos obter bons resultados para os problemas de distância de rearranjo com genes multiplicados.

4.1 Mapeamento de Strings em Permutações

Nesta seção, generalizamos o mapeamento de strings em permutações apresentado na Seção 3.1, considerando que cada rótulo pode possuir mais de duas cópias.

Dada uma string S , para construir um mapeamento \mathbf{x} de S em uma permutação $S^{\mathbf{x}}$, atribuímos um caractere distinto para cada ocorrência dos rótulos multiplicados de S . Cada um dos novos caracteres é denotado por S_i^p , onde S_i é o caractere original de S e $p \in \{1, \dots, occ(|S_i|, S)\}$ é um índice usado para diferenciar os caracteres de mesmo rótulo. Perceba que, no caso de strings com sinais, o mapeamento de caracteres não afeta os sinais que cada caractere já possui.

Para garantir que $S^{\mathbf{x}}$ seja uma permutação, dois caracteres de mesmo rótulo devem obrigatoriamente receber valores distintos de p (índices sobrescritos). Sendo assim, a lista com os valores de p para os caracteres de um rótulo α pode ser representada por uma permutação dos números de 1 a $occ(\alpha, S)$. Combinando todas as permutações de valores de p para cada rótulo multiplicado α , podemos representar um mapeamento \mathbf{x} por um vetor de permutações indexado pelos rótulos. O elemento $\mathbf{x}[\alpha]$ desse vetor contém a permutação correspondente à atribuição dos valores de p para os caracteres do rótulo α .

Para representar todas as atribuições possíveis de valores de p para os caracteres de um rótulo α , utilizamos o conjunto $Sym(\alpha, S)$, composto por todas as permutações dos números de 1 a $occ(\alpha, S)$. Com essa representação dos mapeamentos, é possível ver que existem $\prod_{\alpha \in \Sigma_S} occ(\alpha, S)!$ mapeamentos possíveis da string S em permutações.

Exemplo 13. Aplicação de um mapeamento \mathbf{x} em uma string com sinais S e em uma string sem sinais P .

$$\begin{aligned}
 S &= (+B +C -A -C -B +C -D +D +E +D), \quad mult(S) = \{B, C, D\} \\
 S^{\mathbf{x}} &= (+B^1 +C^2 -A -C^1 -B^2 +C^3 -D^2 +D^3 +E +D^1) \\
 P &= (B C A C B C D D E D), \quad mult(P) = \{B, C, D\} \\
 P^{\mathbf{x}} &= (B^1 C^2 A C^1 B^2 C^3 D^2 D^3 E D^1) \\
 \mathbf{x} &= \begin{array}{c} \text{B} \quad \quad \text{C} \quad \quad \text{D} \\ \boxed{1 \ 2} \mid \boxed{2 \ 1 \ 3} \mid \boxed{2 \ 3 \ 1} \end{array}
 \end{aligned}$$

Embora nos exemplos apresentados neste capítulo as permutações sejam representadas como uma sequência de números inteiros entre 1 e $occ(\alpha, S)$, computacionalmente representamos a permutação correspondente ao rótulo α como um único número inteiro entre 0 e $occ(\alpha, S)! - 1$. Essa representação é mais eficiente em termos de espaço e, dependendo da codificação escolhida, podemos simplificar algumas operações. No nosso caso, optamos pelo código de Lehmer [44], que nos permite acessar a próxima permutação na ordem lexicográfica meramente incrementando o inteiro correspondente, conforme definido a seguir.

Definição 7. Dada uma permutação ϕ dos números inteiros de 1 a n , denotamos por $next(\phi)$ a próxima permutação na ordem lexicográfica. Caso $\phi = (n \dots 2 \ 1)$ tomamos $next(\phi) = (1 \ 2 \dots n)$. Utilizando o código de Lehmer, temos $next(\phi) = \phi + 1 \pmod{n}$.

Definição 8. Sejam \mathbf{x} e \mathbf{w} dois mapeamentos para uma string S , dizemos que \mathbf{w} é um *vizinho* de \mathbf{x} caso satisfaça as seguintes condições:

1. Existe um rótulo $\alpha \in \Sigma_S$ tal que $\mathbf{w}[\alpha] = next(\mathbf{x}[\alpha])$;
2. Para qualquer outro rótulo $\beta \neq \alpha$ temos que $\mathbf{x}[\beta] = \mathbf{w}[\beta]$.

É relevante observar que, diferentemente do caso para rótulos duplicados, a relação de vizinhança não é necessariamente simétrica. Note também que, para rótulos duplicados, essa nova definição de mapeamento é equivalente à apresentada anteriormente.

Exemplo 14. Aplicação dos mapeamentos \mathbf{x} , \mathbf{w} e \mathbf{z} em uma string com sinais S . Note que \mathbf{w} é vizinho de \mathbf{x} , assim como \mathbf{z} é vizinho de \mathbf{w} , mas \mathbf{z} não é vizinho de \mathbf{x} . Além disso, a relação entre \mathbf{z} e \mathbf{w} é simétrica, pois \mathbf{w} também é vizinho de \mathbf{z} . O mesmo não ocorre com \mathbf{w} e \mathbf{x} , pois \mathbf{x} não é vizinho de \mathbf{w} .

$$\begin{aligned}
S &= (+E +B -A -C +D -E -D +E), \quad \text{mult}(S) = \{D, E\} \\
S^{\mathbf{x}} &= (+\textcolor{red}{E}^2 +B -A -C +D^1 -\textcolor{red}{E}^3 -D^2 +\textcolor{red}{E}^1) \\
S^{\mathbf{w}} &= (+\textcolor{red}{E}^3 +B -A -C +\textcolor{blue}{D}^1 -\textcolor{red}{E}^1 -\textcolor{blue}{D}^2 +\textcolor{red}{E}^2) \\
S^{\mathbf{z}} &= (+E^3 +B -A -C +\textcolor{blue}{D}^2 -E^1 -\textcolor{blue}{D}^1 +E^2)
\end{aligned}$$

$$\mathbf{x} = \begin{array}{cc|cc} \text{D} & & \text{E} & \\ \hline 1 & 2 & 2 & 3 & 1 \end{array} \quad \mathbf{w} = \begin{array}{cc|cc} \text{D} & & \text{E} & \\ \hline 1 & 2 & 3 & 1 & 2 \end{array} \quad \mathbf{z} = \begin{array}{cc|cc} \text{D} & & \text{E} & \\ \hline 2 & 1 & 3 & 1 & 2 \end{array}$$

O lema a seguir mostra que podemos usar mapeamentos para encontrar a distância de rearranjo entre duas strings S e P .

Lema 3. *Seja um modelo de rearranjo \mathcal{M} e duas strings S e P . Dados dois mapeamentos \mathbf{x} e \mathbf{y} para S e P , respectivamente, temos $d_{\mathcal{M}}(S, P) \leq d_{\mathcal{M}}(S^{\mathbf{x}}, P^{\mathbf{y}})$.*

Demonstração. Se podemos obter $P^{\mathbf{y}}$ aplicando $d_{\mathcal{M}}(S^{\mathbf{x}}, P^{\mathbf{y}})$ eventos de rearranjos em $S^{\mathbf{x}}$, a mesma sequência de eventos nos permite transformar S em P . \square

Exemplo 15. A mesma sequência de reversões que transforma $S^{\mathbf{x}}$ em $P^{\mathbf{y}}$ sendo usada para transformar S em P .

$$\begin{array}{lcl}
S^{\mathbf{x}} & = & (\quad D^2 \quad B \quad C^1 \quad D^1 \quad C^2 \quad \underbrace{D^3 \quad A}_{\rho(6,7)}) \\
& & (\quad D^2 \quad \underbrace{B \quad C^1 \quad D^1 \quad C^2}_{\rho(2,5)} \quad A \quad D^3) \\
& & (\quad D^2 \quad \underbrace{C^2 \quad D^1}_{\rho(2,3)} \quad C^1 \quad B \quad A \quad D^3) \\
P^{\mathbf{y}} & = & (\quad D^2 \quad D^1 \quad C^2 \quad C^1 \quad B \quad A \quad D^3)
\end{array} \quad \left| \quad \begin{array}{lcl}
S & = & (\quad D \quad B \quad C \quad D \quad C \quad \underbrace{D \quad A}_{\rho(6,7)}) \\
& & (\quad D \quad \underbrace{B \quad C \quad D \quad C}_{\rho(2,5)} \quad A \quad D) \\
& & (\quad D \quad \underbrace{C \quad D}_{\rho(2,3)} \quad C \quad B \quad A \quad D) \\
P & = & (\quad D \quad D \quad C \quad C \quad B \quad A \quad D)
\end{array}$$

$$\mathbf{x} = \begin{array}{cc|cc} \text{C} & & \text{D} & \\ \hline 2 & 1 & 3 & 1 & 2 \end{array} \quad \mathbf{y} = \begin{array}{cc|cc} \text{C} & & \text{D} & \\ \hline 2 & 1 & 3 & 2 & 1 \end{array}$$

4.2 Adaptação das Heurísticas

Utilizando a nova definição de mapeamentos da Seção 4.1, generalizamos as heurísticas de Mapeamentos Aleatórios (MA), Busca Local (BL), GRASP e Algoritmo Genético (AG). As heurísticas que descrevemos aqui são similares às heurísticas desenvolvidas para strings cujos rótulos possuem no máximo duas cópias, descritas no Capítulo 3, e seguem os seguintes passos:

1. A string de destino P é mapeada em uma permutação P^y com um mapeamento y qualquer. Decidimos utilizar $y[\alpha] = (1\ 2\ \dots\ occ(\alpha, P))$, $\forall \alpha \in \Sigma_P$. Na codificação que utilizamos, esse mapeamento é representado por um vetor de zeros.
2. Um conjunto \mathbf{M} de mapeamentos de S em permutações é gerado para uma determinada heurística.
3. Para cada mapeamento $\mathbf{x} \in \mathbf{M}$, calculamos uma estimativa para o valor de $d(S^x, P^y)$, utilizando algoritmos existentes para permutações.
4. O resultado da heurística é o valor da menor estimativa encontrada no passo anterior.

A generalização da heurística MA é direta, basta utilizar a nova definição de mapeamentos. Para gerar um mapeamento \mathbf{x} para S , escolhemos de forma aleatória e uniforme um número inteiro no intervalo $[0, occ(\alpha, S)! - 1]$ para cada elemento $\mathbf{x}[\alpha]$ correspondente a um rótulo $\alpha \in \Sigma_S$.

A generalização da heurística BL também é simples. É importante lembrar que a heurística BL gera o conjunto \mathbf{M} a partir de buscas locais. A cada passo, a heurística avalia um subconjunto da vizinhança do melhor mapeamento conhecido cuja vizinhança ainda não foi explorada. Para generalizar essa heurística, basta utilizar os novos mapeamentos com a nova definição de vizinhança.

Agora explicamos como foi realizada a generalização da heurística GRASP. Conforme explicado anteriormente, essa heurística é composta por uma fase de construção de soluções aleatórias seguida por uma fase de busca local.

Para generalizar a fase de construção, definimos uma nova função de probabilidade $prob(\mathbf{RCL}, \alpha, \phi)$, que indica qual a chance da permutação ϕ ser escolhida para a posição correspondente ao rótulo α nos mapeamentos a serem gerados:

$$prob(\mathbf{RCL}, \alpha, \phi) = \frac{freq(\mathbf{RCL}, \alpha, \phi)}{\sum_{\xi \in Sym(\alpha, S)} freq(\mathbf{RCL}, \alpha, \xi)} = \frac{freq(\mathbf{RCL}, \alpha, \phi)}{|\mathbf{RCL}|}$$

Lembramos que \mathbf{RCL} é um conjunto dos melhores mapeamentos selecionados para participar da etapa de construção. Além disso, $freq(\mathbf{RCL}, \alpha, \phi)$ é o número de vezes em que a permutação ϕ aparece na posição correspondente ao rótulo α nos mapeamentos em \mathbf{RCL} .

Exemplo 16. Cálculo dos valores de frequência e probabilidade para um conjunto **RCL** de mapeamentos da string sem sinais S . Os valores de $freq$ e $prob$ iguais a 0 foram omitidos.

$$S = (C \ B \ A \ C \ D \ C \ D), \quad mult(S) = \{C, D\}$$

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

$$\mathbf{x}_1 = \begin{array}{cc|cc} & C & & D \\ 1 & 2 & 3 & 1 & 2 \end{array} \quad \mathbf{x}_2 = \begin{array}{cc|cc} & C & & D \\ 3 & 1 & 2 & 1 & 2 \end{array} \quad \mathbf{x}_3 = \begin{array}{cc|cc} & C & & D \\ 2 & 1 & 3 & 2 & 1 \end{array}$$

$$freq(\mathbf{RCL}, C, 1 \ 2 \ 3) = 1 \quad freq(\mathbf{RCL}, D, 1 \ 2) = 2$$

$$freq(\mathbf{RCL}, C, 3 \ 1 \ 2) = 1 \quad freq(\mathbf{RCL}, D, 2 \ 1) = 1$$

$$freq(\mathbf{RCL}, C, 2 \ 1 \ 3) = 1$$

$$prob(\mathbf{RCL}, C, 1 \ 2 \ 3) = \frac{1}{3} \quad prob(\mathbf{RCL}, D, 1 \ 2) = \frac{2}{3}$$

$$prob(\mathbf{RCL}, C, 3 \ 1 \ 2) = \frac{1}{3} \quad prob(\mathbf{RCL}, D, 2 \ 1) = \frac{1}{3}$$

$$prob(\mathbf{RCL}, C, 2 \ 1 \ 3) = \frac{1}{3}$$

Para generalizar a fase de busca local, utilizamos a nova definição de vizinhança. Os outros passos da heurística seguem o que foi descrito para o caso de rótulos duplicados.

Por fim, detalhamos a generalização da heurística AG. É importante lembrar que essa heurística mantém uma população de mapeamentos e, a cada geração, seleciona os melhores mapeamentos para criar uma nova população. Essa nova população é criada através de cruzamentos e mutações dos mapeamentos selecionados. Essas operações de cruzamentos e mutações são os únicos conceitos que precisam ser redefinidos. Os outros passos da heurística seguem o mesmo procedimento explicado para o caso em que temos até duas cópias de cada rótulo.

- **Cruzamentos:** Para cada par \mathbf{x}_1 e \mathbf{x}_2 escolhidos para cruzamento, geramos um novo mapeamento, tal que t_c rótulos multiplicados de S (escolhidos aleatoriamente) são mapeados de acordo com \mathbf{x}_1 e os $|mult(S)| - t_c$ rótulos restantes são mapeados de acordo com \mathbf{x}_2 .

Exemplo 17. Um mapeamento \mathbf{z} gerado através do cruzamento de dois mapeamentos \mathbf{x}_1 e \mathbf{x}_2 de uma string sem sinais S .

$$S = (B \ C \ C \ A \ D \ B \ B \ D), \quad t_c = 2$$

$$S^{\mathbf{x}_1} = (\textcolor{red}{B}^3 \ \textcolor{red}{C}^2 \ \textcolor{red}{C}^1 \ A \ \textcolor{red}{D}^2 \ \textcolor{red}{B}^1 \ \textcolor{red}{B}^2 \ \textcolor{red}{D}^1)$$

$$S^{\mathbf{x}_2} = (B^2 \ \textcolor{blue}{C}^1 \ \textcolor{blue}{C}^2 \ A \ D^1 \ B^3 \ B^1 \ D^2)$$

$$S^{\mathbf{z}} = (\textcolor{red}{B}^3 \ \textcolor{blue}{C}^1 \ \textcolor{blue}{C}^2 \ A \ \textcolor{red}{D}^2 \ \textcolor{red}{B}^1 \ \textcolor{red}{B}^2 \ \textcolor{red}{D}^1)$$

$$\mathbf{x}_1 = \begin{array}{cc|cc} & B & & C & & D \\ 3 & 1 & 2 & 2 & 1 & 2 & 1 \end{array} \quad \mathbf{x}_2 = \begin{array}{cc|cc} & B & & C & & D \\ 2 & 3 & 1 & 1 & 2 & 1 & 2 \end{array} \quad \mathbf{z} = \begin{array}{cc|cc} & B & & C & & D \\ 3 & 1 & 2 & 1 & 2 & 2 & 1 \end{array}$$

- **Mutações:** Para cada mapeamento \mathbf{x} selecionado para sofrer mutação, aplicamos a operação *next* em t_m elementos (escolhidos aleatoriamente) do vetor correspondente.

Exemplo 18. Um mapeamento \mathbf{z} gerado através de uma mutação em um mapeamento \mathbf{x} de uma string sem sinais S .

$$\begin{aligned}
 S &= (B \ C \ C \ A \ D \ B \ B \ D), \quad t_m = 2 \\
 S^{\mathbf{x}} &= (\textcolor{red}{B}^3 \ \textcolor{red}{C}^2 \ \textcolor{red}{C}^1 \ A \ D^1 \ \textcolor{red}{B}^1 \ \textcolor{red}{B}^2 \ D^2) \\
 S^{\mathbf{z}} &= (\textcolor{red}{B}^3 \ \textcolor{red}{C}^1 \ \textcolor{red}{C}^2 \ A \ D^1 \ \textcolor{red}{B}^2 \ \textcolor{red}{B}^1 \ D^2)
 \end{aligned}$$

	B	C	D		B	C	D
$\mathbf{x} =$	3	1	2	2	1	2	

B	C	D
3	2	1

$$\mathbf{z} =$$

O principal problema das heurísticas generalizadas é o aumento considerável na quantidade de mapeamentos possíveis. Para contornar parcialmente essa dificuldade, discutimos a seguir o problema de Partição de Strings Comuns Mínima, que nos permite simplificar strings e, conseqüentemente, reduzir o número de mapeamentos possíveis.

4.3 Problemas de Partição de Strings Comuns Mínima

Nesta seção, definimos os problemas de Partição de Strings Comuns Mínima que estão relacionados aos problemas de rearranjo de genomas. Começamos com a definição de diferentes tipos de partições de strings.

Definição 9. Dadas duas strings balanceadas sem sinais S e P , uma *partição direta* $(\mathbb{S}, \mathbb{P}, \phi)$ de S e P é composta por duas sequências \mathbb{S} e \mathbb{P} de strings e uma permutação ϕ , tais que:

- \mathbb{S} e \mathbb{P} têm o mesmo número de elementos ($|\mathbb{S}| = |\mathbb{P}|$);
- É possível obter S concatenando as strings de \mathbb{S} ;
- É possível obter P concatenando as strings de \mathbb{P} ;
- $\mathbb{P}_i = \mathbb{S}_{\phi_i}, \forall 1 \leq i \leq |\mathbb{S}|$.

Exemplo 19. Uma partição direta entre duas strings S e P sem sinais.

$$\begin{aligned}
 S &= (E \ B \ A \ C \ D \ E \ D) \\
 P &= (D \ C \ A \ E \ B \ D \ E) \\
 \mathbb{S} &= \langle (E \ B), (A), (C), (D \ E), (D) \rangle \\
 \mathbb{P} &= \langle (D), (C), (A), (E \ B), (D \ E) \rangle \\
 \phi &= (\ 5 \ 3 \ 2 \ 1 \ 4 \)
 \end{aligned}$$

Em strings com sinais, precisamos levar em consideração os rótulos que podem aparecer em caracteres com orientações distintas em ambas as strings. Sendo assim, outras partições levam em consideração strings inversas, definidas a seguir.

Definição 10. Dada uma string S , dizemos que uma string $Q = \text{inv}(S)$ é a *inversa* de uma string S se ela consiste dos caracteres de S na ordem inversa, ou seja $Q_i = S_{|S|-i+1}, \forall 1 \leq i \leq |S|$. No caso de strings com sinais, o sinal de cada caractere é invertido, ou seja $Q_i = -S_{|S|-i+1}, \forall 1 \leq i \leq |S|$.

Definição 11. Dadas duas strings balanceadas com sinais S e P , uma *partição com sinais* $(\mathbb{S}, \mathbb{P}, \phi)$ de S e P é composta por duas sequências \mathbb{S} e \mathbb{P} de strings e uma permutação ϕ , tais que:

- \mathbb{S} e \mathbb{P} têm o mesmo número de elementos ($|\mathbb{S}| = |\mathbb{P}|$);
- É possível obter S concatenando as strings de \mathbb{S} ;
- É possível obter P concatenando as strings de \mathbb{P} ;
- $\mathbb{P}_i = \mathbb{S}_{\phi_i}$ ou $\mathbb{P}_i = \text{inv}(\mathbb{S}_{\phi_i}), \forall 1 \leq i \leq |\mathbb{S}|$.

Exemplo 20. Uma partição com sinais entre duas strings S' e P' com sinais.

$$\begin{aligned} S' &= (+E +B -A -C +D -E -D) \\ P' &= (-D +C +A +E +B +D +E) \\ \mathbb{S}' &= \langle (+E +B), (-A -C +D), (-E -D) \rangle \\ \mathbb{P}' &= \langle (-D +C +A), (+E +B), (+D +E) \rangle \\ \phi' &= \begin{pmatrix} 2 & 1 & 3 \end{pmatrix} \end{aligned}$$

Por último, definimos um segundo tipo de partição para strings sem sinais, chamada de partição reversa.

Definição 12. Dadas duas strings balanceadas sem sinais S e P , uma *partição reversa* $(\mathbb{S}, \mathbb{P}, \phi)$ de S e P é composta por duas sequências \mathbb{S} e \mathbb{P} de strings e uma permutação ϕ , tais que:

- \mathbb{S} e \mathbb{P} têm o mesmo número de elementos ($|\mathbb{S}| = |\mathbb{P}|$);
- É possível obter S concatenando as strings de \mathbb{S} ;
- É possível obter P concatenando as strings de \mathbb{P} ;
- $\mathbb{P}_i = \mathbb{S}_{\phi_i}$ ou $\mathbb{P}_i = \text{inv}(\mathbb{S}_{\phi_i}), \forall 1 \leq i \leq |\mathbb{S}|$.

Note que, por definição, toda partição direta é uma partição reversa, mas existem partições reversas que não são partições diretas.

Exemplo 21. Uma partição reversa entre duas strings S e P sem sinais. Essa partição reversa não é uma partição direta.

$$\begin{aligned} S &= (E \ B \ A \ C \ D \ E \ D) \\ P &= (D \ C \ A \ E \ B \ D \ E) \\ \mathbb{S} &= \langle (E \ B), (A \ C \ D), (E \ D) \rangle \\ \mathbb{P} &= \langle (D \ C \ A), (E \ B), (D \ E) \rangle \\ \phi &= (\ 2 \ 1 \ 3) \end{aligned}$$

Chamamos as strings de \mathbb{S} e \mathbb{P} de *blocos*. O tamanho de uma partição é igual ao número de quebras necessárias para separar os blocos, ou seja $|(\mathbb{S}, \mathbb{P}, \phi)| = |\mathbb{S}| - 1 = |\mathbb{P}| - 1$. O problema de Partição de Strings Comuns Mínima consiste em encontrar uma partição de tamanho mínimo de duas strings S e P . As partições dos exemplos 22, 20 e 21 são todas mínimas.

Considerando as diferentes definições para partições, temos as seguintes variações desse problema:

- Partição de Strings Comuns Mínima (“Minimum Common String Partition” – MCSP): Considerando partições diretas;
- Partição com Sinais de Strings Comuns Mínima (“Signed Minimum Common String Partition” – SMCSPP): Considerando partições com sinais;
- Partição Reversa de Strings Comuns Mínima (“Reverse Minimum Common String Partition” – RMCSP): Considerando partições reversas.

Goldstein e coautores [33] provaram que os problemas MCSP e SMCSPP pertencem à classe de problemas NP-Difícil, mesmo quando $occ(S) = 2$. O seguinte teorema mostra que o mesmo ocorre com o problema RMCSP.

Teorema 1. *Sejam S e P duas strings balanceadas tal que $occ(S) = occ(P) = 2$. O problema de encontrar uma partição reversa de tamanho mínimo de S e P pertence à classe de problemas NP-Difícil.*

Demonstração. Considere a versão de decisão dos problemas MCSP e RMCSP. Dado um inteiro k , queremos saber se existe uma partição direta para o problema MCSP ou partição reversa para o problema RMCSP de tamanho menor ou igual a k . A seguir descrevemos uma redução do problema MCSP para o problema RMCSP.

Seja uma instância de MCSP composta por duas strings S e P , e um inteiro k . Construamos uma instância do problema RMCSP composta pelo mesmo inteiro k e por duas strings S' e P' tais que $\forall i \in [1, |S|], S'_{2i-1} = S_i^\ell, S'_{2i} = S_i^r$ e $\forall i \in [1, |P|], P'_{2i-1} = P_i^\ell, P'_{2i} = P_i^r$. A seguir, provamos que a instância de MCSP é satisfeita se e somente se a instância de RMCSP é satisfeita.

(\implies) Seja uma partição direta $(\mathbb{S}, \mathbb{P}, \phi)$ de S e P com tamanho $k' \leq k$. Obtemos \mathbb{S}' substituindo, em cada bloco de \mathbb{S} , o caractere S_i pelo par $(S_i^\ell \ S_i^r)$, e obtemos \mathbb{P}' substituindo, em cada bloco de \mathbb{P} , o caractere P_i pelo par $(P_i^\ell \ P_i^r)$. Note que $(\mathbb{S}', \mathbb{P}', \phi)$ é uma partição direta com tamanho $k' \leq k$ de S' e P' , e, por definição, toda partição direta é também uma partição reversa.

(\impliedby) Seja uma partição reversa $(\mathbb{S}', \mathbb{P}', \phi)$ de S' e P' com tamanho $k' \leq k$. Inicialmente, realizamos uma transformação na partição reversa para garantir que todo par $(S'_{2i-1} \ S'_{2i})$, com $i \in [1, |S|]$, pertença a um mesmo bloco.

Seja $(S'_{2i-1} \ S'_{2i})$, com $i \in [1, |S|]$, um par separado em dois blocos A e B , tal que S'_{2i-1} esteja no bloco A e S'_{2i} esteja no bloco B . Note que A e B devem ser consecutivos, S'_{2i-1} deve estar no final de A , e S'_{2i} deve estar no começo de B . Seja A' um novo bloco formado ao adicionarmos S'_{2i} no bloco A (ou seja, $A' = A \ S'_{2i}$) e B' um novo bloco formado ao removermos S'_{2i} do bloco B (ou seja, $B = S'_{2i} \ B'$). Substituímos os blocos A e B pelos blocos A' e B' . Aplicamos essa transformação para todos os pares $(S'_{2i-1} \ S'_{2i})$, com $i \in [1, |S|]$, separados em blocos distintos. Após as transformações, obtemos uma nova sequência \mathbb{S}'' , onde $|\mathbb{S}'| = |\mathbb{S}''|$.

Note que, para cada bloco A de \mathbb{S}' mencionado acima, sabemos que em \mathbb{P}' deve existir um bloco A ou $inv(A)$, mas como todas as ocorrências do rótulo S'_{2i-1} aparecem procedidas de uma ocorrência do rótulo S'_{2i} , \mathbb{P}' não pode conter $inv(A)$, logo existe um bloco A em \mathbb{P}' . Como, logo após o bloco A , temos um caractere com o rótulo S'_{2i} , podemos adicioná-lo em A para obter A' . Um raciocínio análogo garante que existe um bloco B em \mathbb{P}' , e que podemos remover S'_{2i} de B para obter B' . Sendo assim, ao aplicarmos transformações similares as aplicados em \mathbb{S} para garantir que em \mathbb{P} todos os pares $(S'_{2i-1} \ S'_{2i})$, com $i \in [1, |S|]$, estejam em um mesmo bloco, obtemos uma sequência \mathbb{P}'' , tal que $(\mathbb{S}'', \mathbb{P}'', \phi)$ ainda é uma partição reversa de tamanho k' .

Considere agora a partição reversa $(\mathbb{S}'', \mathbb{P}'', \phi)$. A seguir, mostramos que essa partição reversa é também uma partição direta, ou seja, cada bloco X de \mathbb{S}'' está também em \mathbb{P}'' . Para isso, note que todo bloco X de \mathbb{S}'' possui o par $(S'_{2i-1} \ S'_{2i})$, para algum $i \in [1, |S|]$. Além disso, \mathbb{P}'' deve conter o bloco X ou o bloco $inv(X)$ mas, por construção, \mathbb{P}'' contém o par $(S'_{2i-1} \ S'_{2i})$, portanto \mathbb{P}'' contém X .

Obtemos \mathbb{S} substituindo, em cada bloco de \mathbb{S}'' , o par $(S'_{2i-1} \ S'_{2i})$ pelo elemento S_i , e obtemos \mathbb{P} substituindo, em cada bloco de \mathbb{P}'' , o par $(P'_{2i-1} \ P'_{2i})$ pelo elemento P_i . Note que $(\mathbb{S}, \mathbb{P}, \phi)$ é uma partição direta com tamanho $k' \leq k$ de S e P , e o teorema segue. \square

Exemplo 22. A redução do Teorema 1 aplicada em uma instância do problema MCSP formada por duas strings S e P . As strings S' e P' formam uma instância para o problema RMCS, a partição $(\mathbb{S}', \mathbb{P}', \phi)$ é a solução desse problema e a partição $(\mathbb{S}, \mathbb{P}, \phi)$ é a solução correspondente para o problema MCSP.

$$\begin{aligned}
S &= (E \ B \ A \ C \ D \ E \ D) \\
P &= (D \ C \ A \ E \ B \ D \ E) \\
S' &= (E^l \ E^r \ B^l \ B^r \ A^l \ A^r \ C^l \ C^r \ D^l \ D^r \ E^l \ E^r \ D^l \ D^r) \\
P' &= (D^l \ D^r \ C^l \ C^r \ A^l \ A^r \ E^l \ E^r \ B^l \ B^r \ D^l \ D^r \ E^l \ E^r) \\
\mathbb{S}' &= \langle (E^l \ E^r \ B^l \ B^r), (A^l \ A^r), (C^l \ C^r), (D^l \ D^r \ E^l \ E^r), (D^l \ D^r) \rangle \\
\mathbb{P}' &= \langle (D^l \ D^r), (C^l \ C^r), (A^l \ A^r), (E^l \ E^r \ B^l \ B^r), (D^l \ D^r \ E^l \ E^r) \rangle \\
\mathbb{S} &= \langle (E \ B), (A), (C), (D \ E), (D) \rangle \\
\mathbb{P} &= \langle (D), (C), (A), (E \ B), (D \ E) \rangle \\
\phi &= (5 \ 3 \ 2 \ 1 \ 4)
\end{aligned}$$

Podemos utilizar uma partição de strings para construir strings simplificadas que nos permitam aproximar a distância das strings originais. Dada uma partição direta $(\mathbb{S}, \mathbb{P}, \phi)$ de duas strings S e P , associamos a cada bloco de \mathbb{S} e de \mathbb{P} um caractere de forma que dois blocos iguais recebam caracteres correspondentes ao mesmo rótulo.

Para o caso de strings com sinais ou de strings sem sinais com uma partição reversa, associamos a cada bloco de \mathbb{S} e de \mathbb{P} um caractere com sinal de tal forma que:

- Dois blocos iguais recebam caracteres correspondentes ao mesmo rótulo e com sinais iguais;
- Dois blocos inversos recebam caracteres correspondentes ao mesmo rótulo e com sinais distintos.

Construímos as strings reduzidas \hat{S} e \hat{P} a partir das strings S e P , ao substituímos os blocos da partição pelos caracteres atribuídos a eles.

Exemplo 23. Duas strings reduzidas \hat{S} e \hat{P} obtidas a partir da partição do Exemplo 22.

$$\begin{aligned}
\mathbb{S} &= \langle (E \ B), (A), (C), (D \ E), (D) \rangle \\
\mathbb{P} &= \langle (D), (C), (A), (E \ B), (D \ E) \rangle \\
\phi &= (5 \ 3 \ 2 \ 1 \ 4) \\
\hat{S} &= (A \ B \ C \ D \ E) \\
\hat{P} &= (E \ C \ B \ A \ D)
\end{aligned}$$

Correspondência entre caracteres e blocos

$$\begin{aligned}
A &\iff (E \ B) & D &\iff (D \ E) \\
B &\iff (A) & E &\iff (D) \\
C &\iff (C)
\end{aligned}$$

Exemplo 24. Duas strings reduzidas \hat{S} e \hat{P} obtidas a partir das partições dos exemplos 20 e 21. Note que, mesmo para strings originais sem sinais, devemos utilizar caracteres com sinais nas strings reduzidas, pois, caso contrário, não seria possível diferenciar o bloco $(E \ B)$ de seu inverso $(B \ E)$.

$$\begin{aligned}\mathbb{S}' &= \langle (+E \ +B), (-A \ -C \ +D), (-E \ -D) \rangle \\ \mathbb{P}' &= \langle (-D \ +C \ +A), (+E \ +B), (+D \ +E) \rangle \\ \mathbb{S} &= \langle (E \ B), (A \ C \ D), (E \ D) \rangle \\ \mathbb{P} &= \langle (D \ C \ A), (E \ B), (D \ E) \rangle \\ \phi' &= \phi = \begin{pmatrix} & 2 & 1 & 3 \end{pmatrix} \\ \hat{S} &= (+A \ +B \ +C) \\ \hat{P} &= (-B \ +A \ -C)\end{aligned}$$

Correspondência entre caracteres e blocos:

$$\begin{aligned}A &\iff (+E \ +B), (E \ B) \\ B &\iff (-A \ -C \ +D), (A \ C \ D) \\ C &\iff (-E \ -D), (E \ D)\end{aligned}$$

4.3.1 Partições de Permutações

Embora no caso geral os problemas de partição pertençam à classe de problemas NP-Difícil, quando estamos lidando com permutações (strings sem rótulos multiplicados) esses problemas podem ser resolvidos em tempo linear. Para isso, utilizamos o conceito de breakpoints, que é habitualmente utilizado em problemas de rearranjo em permutações.

As abordagens que consideram uma permutação π geralmente utilizam o alfabeto $\Sigma_\pi = \{1, \dots, |\pi|\}$. Além disso, para simplificar as definições, é suposto que a permutação é estendida com a adição de dois novos elementos $\pi_0 = 0$ e $\pi_{|\pi|+1} = |\pi| + 1$, no início e fim da permutação, respectivamente. Em permutações com sinais, esses novos elementos possuem sinais positivos. No restante desta seção, consideramos que todas as permutações estão estendidas.

Exemplo 25. Uma permutação π sem sinais, estendida com a adição dos elementos 0 e 6, e uma permutação π' com sinais, estendida com a adição dos elementos +0 e +6.

$$\begin{aligned}\pi &= \begin{pmatrix} & \mathbf{0} & 4 & 3 & 1 & 2 & 5 & \mathbf{6} \end{pmatrix} \\ \pi' &= \begin{pmatrix} & +\mathbf{0} & -4 & -3 & +1 & +2 & -5 & +\mathbf{6} \end{pmatrix}\end{aligned}$$

Definição 13. Dada uma permutação π sem sinais, um par de elementos π_i e π_{i+1} , com $0 \leq i \leq |\pi|$, é um *breakpoint do tipo 1* se $|\pi_{i+1} - \pi_i| \neq 1$. Denotamos o número de breakpoints do tipo 1 em π por $b_1(\pi)$.

Definição 14. Dada uma permutação π com ou sem sinais, um par de elementos π_i e π_{i+1} , com $0 \leq i \leq |\pi|$, é um *breakpoint do tipo 2* se $\pi_{i+1} - \pi_i \neq 1$. Denotamos o número de breakpoints do tipo 2 em π por $b_2(\pi)$.

Chamamos de *strip* uma sequência de elementos que se encontra entre dois breakpoints, que precede o primeiro breakpoint, ou que sucede o último breakpoint de uma permutação. As strips são do tipo 1 caso sejam separadas por breakpoints do tipo 1 e do tipo 2 caso sejam separadas por breakpoints do tipo 2.

Exemplo 26. A permutação π do Exemplo 25 possui 3 breakpoints do tipo 1 e 4 breakpoints do tipo 2. A permutação π' possui 4 breakpoints do tipo 2. Abaixo ilustramos esses breakpoints utilizando um ponto (\bullet).

Breakpoints tipo 1:

$$\pi = (0 \bullet 4 \ 3 \bullet 1 \ 2 \bullet 5 \ 6), \ b_1(\pi) = 3$$

Breakpoints tipo 2:

$$\begin{aligned} \pi &= (0 \bullet 4 \bullet 3 \bullet 1 \ 2 \bullet 5 \ 6), \ b_2(\pi) = 4 \\ \pi' &= (+0 \bullet -4 \ -3 \bullet +1 \ +2 \bullet -5 \bullet +6), \ b_2(\pi') = 4 \end{aligned}$$

A seguir, mostramos como utilizar breakpoints para encontrar uma partição mínima entre duas permutações π e σ , de tamanho n . Primeiro, mostramos que, sem perda de generalidade, pode-se considerar que a permutação σ é a permutação identidade $\iota^n = (1 \ 2 \ \dots \ n)$ (ou $\iota^n = (+1 \ +2 \ \dots \ +n)$, no caso de permutações com sinais).

Lema 4. Uma partição entre duas permutações π e σ de tamanho n corresponde a uma partição entre uma nova permutação π' e a permutação ι^n .

Demonstração. Construimos π' substituindo cada elemento π_i por $\sigma^{-1}(\pi_i)$, onde σ^{-1} leva cada caractere em sua posição na permutação σ (ou seja, $\sigma_{\sigma^{-1}(x)} = x$). Note que a mesma substituição aplicada em σ gera a permutação ι^n . Portanto, a correspondência entre elemento de π e σ é a mesma que a correspondência entre elemento de π' e ι^n , diferindo apenas nos rótulos dos elemento. Sendo assim, qualquer partição entre π e σ corresponde a uma partição entre π' e ι^n . \square

Lema 5. A partição direta mínima $(\mathbb{S}, \mathbb{P}, \phi)$ das permutações sem sinais π' e ι^n é composta pelas strips do tipo 2 de π' .

Demonstração. Para provar que os blocos pertencentes a partição correspondem as strips de π' , provamos que os blocos de partição direta mínima $(\mathbb{S}, \mathbb{P}, \phi)$, de π' e ι^n , não contém breakpoints, e que os blocos consecutivos são separados por breakpoints.

Note que, caso um bloco $Q \in \mathbb{S}$ contenha um breakpoint (π'_i, π'_{i+1}) , sabemos que $\pi'_{i+1} - \pi'_i \neq 1$. Portanto, o par (π'_i, π'_{i+1}) não está presente em ι^n . Sendo assim, essa partição não é válida, pois o bloco Q não pode existir em ι^n .

Agora, considere que dois blocos consecutivos $Q, R \in \mathbb{S}$ não são separados por um breakpoint. Sabemos que o último elemento de Q aparece em ι^n seguido do primeiro elemento de R . Consequentemente, como sabemos que Q e R devem estar presentes em ι^n , Q e R devem ser consecutivos em ι^n . Com isso, é possível juntar os blocos Q e R em \mathbb{S} e em \mathbb{P} , reduzindo o tamanho da partição $(\mathbb{S}, \mathbb{P}, \phi)$, o que contradiz a hipótese de que a partição era mínima. \square

Lema 6. *A partição com sinais mínima $(\mathbb{S}, \mathbb{P}, \phi)$ das permutações com sinais π' e ι^n é composta pelas strips do tipo 2 de π' , que podem aparecer invertidas em \mathbb{P} .*

Demonstração. Para provar que os blocos pertencentes a partição correspondem a strips de π' , provamos que os blocos da partição com sinais mínima $(\mathbb{S}, \mathbb{P}, \phi)$, de π' e ι^n , não contém breakpoints, e que os blocos consecutivos são separados por breakpoints.

Note que, caso um bloco $Q \in \mathbb{S}$ contenha um breakpoint (π'_i, π'_{i+1}) , sabemos que $\pi'_{i+1} - \pi'_i \neq 1$. Portanto, os pares (π'_i, π'_{i+1}) e $(-\pi'_{i+1}, -\pi'_i)$ não estão presentes em ι^n . Sendo assim, essa partição não é válida, pois os blocos Q e $inv(Q)$ não podem existir em ι^n .

Considere agora que dois blocos consecutivos $Q, R \in \mathbb{S}$ não são separados por um breakpoint. Dessa forma, seja π'_i o último elemento de Q e seja π'_{i+1} o primeiro elemento de R . Por definição de breakpoints, sabemos que π'_i precede π'_{i+1} em ι^n ou $-\pi'_{i+1}$ precede $-\pi'_i$ em ι^n . Portanto, caso Q esteja em ι^n , R também deve estar, e ambos devem ser consecutivos em ι^n . Por outro lado, caso $inv(Q)$ esteja em ι^n , $inv(R)$ também deve estar, e ambos devem ser consecutivos em ι^n . Note que obrigatoriamente um dos casos acima deve ser verdade, dado que Q ou $inv(Q)$ deve estar presente em ι^n .

Com isso, podemos juntar os blocos Q e R (ou seus inversos) em \mathbb{S} e em \mathbb{P} , o que reduz o tamanho da partição $(\mathbb{S}, \mathbb{P}, \phi)$ e contradiz a hipótese de que a partição era mínima. \square

Lema 7. *A partição reversa mínima $(\mathbb{S}, \mathbb{P}, \phi)$ das permutações com sinais π' e ι^n é composta pelas strips do tipo 1 de π' , que podem aparecer invertidas em \mathbb{P} .*

Demonstração. Análoga à prova do Lema 6. \square

Com essas relações, e dado que os breakpoints podem ser calculados em tempo linear, as partições mínimas também podem ser obtidas em tempo linear.

4.3.2 Relação entre Partições e Distâncias de Rearranjo

A partir de aproximações para os problemas de partição, é possível obter aproximações para os problemas de distâncias de rearranjo. Nesta seção, apresentamos teoremas que demonstram essas relações. O Teorema 2 é uma adaptação do Teorema 4.7 apresentado por Chen e coautores [19]. Uma prova alternativa para o Teorema 4 foi apresentada por Shapira e Storer [53, Teorema 2]. Em seu trabalho, Shapira e Storer chamam as transposições de *movimentos*, e classificam os movimentos em recursivos, quando não afetam o tamanho da partição mínima, e não recursivos, quando aumentam o tamanho da partição mínima. O teorema provado por eles garante que uma sequência qualquer de n movimentos pode ser transformada em uma sequência de $3n$ movimentos não recursivos.

Supomos, que as strings S e P foram estendidas com a adição de dois caracteres γ_i e γ_f não pertencentes a Σ_S , γ_i é colocado no início de S e P e γ_f é colocado no final de S e P . Caso S seja uma string com sinais γ_i e γ_f têm sinais positivos.

Exemplo 27. Uma string S sem sinais, estendida com a adição dos caracteres I e F , e uma string S' com sinais, estendida com a adição dos caracteres $+I$ e $+F$.

$$\begin{aligned} S &= (\text{ I } \text{ A } \text{ B } \text{ B } \text{ D } \text{ A } \text{ C } \text{ C } \text{ B } \text{ F }) \\ S' &= (+\text{I } +\text{A } +\text{B } -\text{B } -\text{D } +\text{A } -\text{C } +\text{C } -\text{B } +\text{F }) \end{aligned}$$

Definição 15. Uma partição $(\mathbb{S}, \mathbb{P}, \phi)$ de um par de strings S e P é *compatível* com um par de mapeamentos \mathbf{x} e \mathbf{y} de S e P , respectivamente, se $(\mathbb{S}^{\mathbf{x}}, \mathbb{P}^{\mathbf{y}}, \phi)$ é uma partição de $S^{\mathbf{x}}$ e $P^{\mathbf{y}}$, onde $\mathbb{S}^{\mathbf{x}}$ (resp. $\mathbb{P}^{\mathbf{y}}$) corresponde a sequência \mathbb{S} (resp. \mathbb{P}) após a aplicação das atribuições de caracteres dadas pelo mapeamento \mathbf{x} (resp. \mathbf{y}).

Exemplo 28. Duas strings S e P sem sinais, uma partição direta entre elas e um par de mapeamentos \mathbf{x} e \mathbf{y} compatíveis com essa partição.

$$\begin{aligned} S &= (\text{ I } \text{ A } \text{ B } \text{ B } \text{ D } \text{ A } \text{ C } \text{ C } \text{ B } \text{ F }) \\ P &= (\text{ I } \text{ C } \text{ B } \text{ C } \text{ A } \text{ B } \text{ B } \text{ D } \text{ A } \text{ F }) \\ \mathbb{S} &= \langle (I), (A \text{ B } \text{ B}), (D \text{ A}), (C), (C \text{ B}), (F) \rangle \\ \mathbb{P} &= \langle (I), (C \text{ B}), (C), (A \text{ B } \text{ B}), (D \text{ A}), (F) \rangle \\ \phi &= (\text{ 1 } \text{ 5 } \text{ 4 } \text{ 2 } \text{ 3 } \text{ 6 }) \\ S^{\mathbf{x}} &= (\text{ I } \text{ A}^2 \text{ B}^1 \text{ B}^3 \text{ D } \text{ A}^1 \text{ C}^1 \text{ C}^2 \text{ B}^2 \text{ F }) \\ P^{\mathbf{y}} &= (\text{ I } \text{ C}^2 \text{ B}^2 \text{ C}^1 \text{ A}^2 \text{ B}^1 \text{ B}^3 \text{ D } \text{ A}^1 \text{ F }) \\ \mathbb{S} &= \langle (I), (A^2 \text{ B}^1 \text{ B}^3), (D \text{ A}^1), (C^1), (C^2 \text{ B}^2), (F) \rangle \\ \mathbb{P} &= \langle (I), (C^2 \text{ B}^2), (C^1), (A^2 \text{ B}^1 \text{ B}^3), (D \text{ A}^1), (F) \rangle \\ \mathbf{x} &= \begin{array}{c} \text{A} \quad \text{B} \quad \text{C} \\ \boxed{2 \ 1} \mid \boxed{1 \ 3 \ 2} \mid \boxed{1 \ 2} \end{array} \quad \mathbf{y} = \begin{array}{c} \text{A} \quad \text{B} \quad \text{C} \\ \boxed{2 \ 1} \mid \boxed{2 \ 1 \ 3} \mid \boxed{2 \ 1} \end{array} \end{aligned}$$

As provas a seguir utilizam mapeamentos das strings em permutações, aproveitando os resultados conhecidos para permutações.

Lema 8. *Seja $(\mathbb{S}, \mathbb{P}, \phi)$ uma partição com sinais mínima de duas strings com sinais S e P . Qualquer sequência de reversões que transforma S em P deve ter tamanho maior ou igual a $\frac{|\mathbb{S}, \mathbb{P}, \phi|}{2}$.*

Demonstração. Considere uma sequência qualquer de k reversões que transforma S em P . Sejam \mathbf{x} e \mathbf{y} um par de mapeamentos de S e P , respectivamente, tais que a permutação $S^{\mathbf{x}}$ pode ser transformada na permutação $P^{\mathbf{y}}$ utilizando essa sequência de k reversões. Considere uma partição $(\mathbb{R}, \mathbb{Q}, \xi)$ compatível com os mapeamentos \mathbf{x} e \mathbf{y} . Pelos lemas 4 e 6, existe uma permutação π , tal que $|(\mathbb{R}, \mathbb{Q}, \xi)| = |(\mathbb{R}^{\mathbf{x}}, \mathbb{Q}^{\mathbf{y}}, \xi)| = |(\pi, \iota^{|\pi|}, \xi)| = b_2(\pi)$. Pela forma como a permutação π é obtida, a sequência de k reversões também transforma π em $\iota^{|\pi|}$.

Sabemos que $\frac{b_2(\pi)}{2} \leq k$, pois cada reversão remove no máximo 2 breakpoints [5]. Como $(\mathbb{S}, \mathbb{P}, \phi)$ é uma partição mínima, temos $\frac{|\mathbb{S}, \mathbb{P}, \phi|}{2} \leq \frac{|\mathbb{R}, \mathbb{Q}, \xi|}{2} \leq k$. \square

Teorema 2. *Uma ℓ -aproximação para o problema SMCSF garante uma 2ℓ -aproximação para o problema DR.*

Demonstração. Sejam S e P duas strings com sinais e seja p o tamanho da partição com sinais mínima de S e P . Um algoritmo para o problema SMCSF com fator de aproximação ℓ retorna uma partição com sinais $(\mathbb{S}, \mathbb{P}, \phi)$, tal que $p \leq |(\mathbb{S}, \mathbb{P}, \phi)| \leq \ell p$.

Considere as strings reduzidas \hat{S} e \hat{P} obtidas a partir da partição $(\mathbb{S}, \mathbb{P}, \phi)$. Mapeie \hat{S} e \hat{P} em permutações com dois mapeamentos quaisquer \mathbf{x} e \mathbf{y} . Note que sempre é possível transformar a permutação $\hat{S}^{\mathbf{x}}$ na permutação $\hat{P}^{\mathbf{y}}$ com $k = |\hat{S}^{\mathbf{x}}| - 1 = |(\mathbb{S}, \mathbb{P}, \phi)|$ reversões (uma consequência da fórmula da distância de Hannenhalli e Pevzner [35], e do fato de que as extremidades de S e P são iguais). Como temos uma sequência de k reversões que transforma \hat{S} em \hat{P} , podemos obter uma sequência correspondente de k reversões que transforma S em P .

Pelo Lema 8 sabemos que $d_{\mathcal{R}}(S, P) \geq \frac{p}{2}$. Como $k = |(\mathbb{S}, \mathbb{P}, \phi)|$, temos que $d_{\mathcal{R}}(S, P) \leq k \leq 2\ell d_{\mathcal{R}}(S, P)$. \square

Exemplo 29. Uma sequência de reversões que transforma uma string reduzida com sinais \hat{S} em uma string reduzida com sinais \hat{P} e uma sequência de reversões correspondente que transforma a string original com sinais S na string original com sinais P .

$$\begin{array}{lcl}
 \hat{S} & = & (+A \underbrace{+B +C}_{\rho(2,3)} +D +E) \\
 & & (+A -C -B \underbrace{+D}_{\rho(4,4)} +E) \\
 & & (+A -C \underbrace{-B}_{\rho(3,3)} -D +E) \\
 \hat{P} & = & (+A -C +B -D +E)
 \end{array}
 \quad \left| \quad
 \begin{array}{lcl}
 S & = & (+E \underbrace{+C +B -A}_{\rho(2,4)} -C -D +F) \\
 & & (+E +A -B -C \underbrace{-C -D}_{\rho(5,6)} +F) \\
 & & (+E +A \underbrace{-B -C}_{\rho(3,4)} +D +C +F) \\
 P & = & (+E +A +C +B +D +C +F)
 \end{array}$$

Correspondência entre caracteres e strings:

$$\begin{array}{lcl}
 A & \iff & (+E) \\
 B & \iff & (+C +B) \\
 C & \iff & (-A) \\
 D & \iff & (-C -D) \\
 E & \iff & (+F)
 \end{array}$$

Lema 9. *Seja $(\mathbb{S}, \mathbb{P}, \phi)$ uma partição reversa mínima de duas strings sem sinais S e P . Qualquer sequência de reversões que transforma S em P deve ter tamanho maior ou igual a $\frac{|(\mathbb{S}, \mathbb{P}, \phi)|}{2}$.*

Demonstração. Similar à prova do Lema 8, mas considerando breakpoints do tipo 1. Nesse caso, cada reversão também remove no máximo 2 breakpoints [39]. \square

Teorema 3. *Uma ℓ -aproximação para o problema RMCSP garante uma 2ℓ -aproximação para o problema DR.*

Demonstração. Similar à prova do Teorema 2, substituindo o Lema 8 pelo Lema 9. \square

Exemplo 30. Uma sequência de reversões que transforma uma string reduzida com sinais \hat{S} em uma string reduzida com sinais \hat{P} e uma sequência de reversões correspondente que transforma a strings original sem sinais S na string original sem sinais P .

$$\begin{array}{lcl}
 \hat{S} & = & (+A \underbrace{+B +C}_{\rho(2,3)} +D +E) \\
 & & (+A -C -B \underbrace{+D}_{\rho(4,4)} +E) \\
 & & (+A -C \underbrace{-B}_{\rho(3,3)} -D +E) \\
 \hat{P} & = & (+A -C +B -D +E)
 \end{array}
 \quad \left| \quad
 \begin{array}{lcl}
 S & = & (E \underbrace{C B A}_{\rho(2,4)} C D F) \\
 & & (E A B C \underbrace{C D}_{\rho(5,6)} F) \\
 & & (E A \underbrace{B C}_{\rho(3,4)} D C F) \\
 P & = & (E A C B D C F)
 \end{array}$$

Correspondência entre caracteres e strings:

$$\begin{aligned}
 A &\iff (E) \\
 B &\iff (C B) \\
 C &\iff (A) \\
 D &\iff (C D) \\
 E &\iff (F)
 \end{aligned}$$

Lema 10. Seja $(\mathbb{S}, \mathbb{P}, \phi)$ uma partição direta mínima de duas strings sem sinais S e P . Qualquer sequência de transposições que transforma S em P deve ter tamanho maior ou igual a $\frac{|\mathbb{S}, \mathbb{P}, \phi|}{3}$.

Demonstração. Similar à prova do Lema 8, levando em conta que cada transposição remove no máximo 3 breakpoints do tipo 2 [6]. \square

Teorema 4. Uma ℓ -aproximação para o problema MCSP garante uma 3ℓ -aproximação para o problema DT.

Demonstração. Similar à prova do Teorema 2, substituído o Lema 8 pelo Lema 10. Note que, para transformar a string reduzida sem sinais \hat{S} na string reduzida sem sinais \hat{P} , basta utilizar a i -ésima transposição para posicionar corretamente o elemento na posição $i + 1$. Como o primeiro e último elemento já estão posicionados corretamente, são necessárias apenas $|\hat{S}| - 2$ transposições. \square

Exemplo 31. Uma sequência de transposições que transforma uma string reduzida sem sinais \hat{S} em uma string reduzida sem sinais \hat{P} e uma sequência de transposições correspondente que transforma a strings original sem sinais S na string original sem sinais P .

$$\begin{array}{lcl}
 \hat{S} & = & (A \underbrace{\textcolor{red}{D} \textcolor{blue}{B} \textcolor{blue}{C}}_{\tau(2,5,6)} E) \\
 & & (A C \underbrace{\textcolor{red}{D} \textcolor{blue}{B}}_{\tau(2,3,4)} E) \\
 \hat{P} & = & (A C B D E)
 \end{array}
 \quad \left| \quad
 \begin{array}{lcl}
 \hat{S} & = & (E \underbrace{\textcolor{red}{C} \textcolor{red}{D} \textcolor{red}{C} \textcolor{blue}{B} \textcolor{blue}{A}}_{\tau(2,5,6)} F) \\
 & & (E A \underbrace{\textcolor{red}{C} \textcolor{red}{D} \textcolor{blue}{C} \textcolor{blue}{B}}_{\tau(2,3,4)} F) \\
 \hat{P} & = & (E A C B C D F)
 \end{array}$$

Correspondência entre caracteres e strings:

$$\begin{aligned} A &\iff (E) \\ B &\iff (C \ B) \\ C &\iff (A) \\ D &\iff (C \ D) \\ E &\iff (F) \end{aligned}$$

Lema 11. *Seja $(\mathbb{S}, \mathbb{P}, \phi)$ uma partição com sinais mínima de duas strings com sinais S e P . Qualquer sequência de reversões e transposições que transforma S em P deve ter tamanho maior ou igual a $\frac{|\mathbb{S}, \mathbb{P}, \phi|}{3}$.*

Demonstração. Análoga à prova do Lema 8, e pelo fato de que cada reversão remove no máximo 2 breakpoints do tipo 2 e cada transposição remove no máximo 3 breakpoints do tipo 2. \square

Teorema 5. *Uma ℓ -aproximação para o problema SMCSP garante uma 3ℓ -aproximação para o problema $D\bar{R}T$.*

Demonstração. Análoga à prova do Teorema 4, substituindo o Lema 10 pelo Lema 11. \square

Lema 12. *Seja $(\mathbb{S}, \mathbb{P}, \phi)$ uma partição reversa mínima de duas strings sem sinais S e P . Qualquer sequência de reversões e transposições que transforma S em P deve ter tamanho maior ou igual a $\frac{|\mathbb{S}, \mathbb{P}, \phi|}{3}$.*

Demonstração. Análoga à prova do Lema 11, utilizando breakpoints do tipo 1. \square

Teorema 6. *Uma ℓ -aproximação para o problema RMCSP garante uma 3ℓ -aproximação para o problema DRT .*

Demonstração. Análoga à prova do Teorema 4, substituindo o Lema 10 pelo Lema 12. \square

4.3.3 Algoritmos da Literatura

Nesta seção, descrevemos brevemente algoritmos da literatura para os problemas de partição, incluindo as modificações necessárias para que solucionem todas as variações do problema.

Partição do SOAR (PSOAR)

O primeiro algoritmo que descrevemos é o algoritmo de partição presente no algoritmo SOAR, desenvolvido por Chen e coautores [19]. Esse trabalho foi o primeiro a apresentar o problema SMCSP e sua relação com o problema $D\bar{R}$. Parte do algoritmo SOAR resolve o problema $D\bar{R}$ decompondo-o em dois subproblemas, o SMCSP e o problema de Empacotamento Máximo de Ciclos (EMC). Daqui em diante, falamos sobre a solução proposta para o problema SMCSP, e a descrição do problema EMC é apresentada na Seção 4.4. Chamamos o algoritmo para partição utilizado no SOAR de PSOAR.

Definição 16. Dado um par de strings com sinais S e P , um *match com sinais* de S em P é um par de caracteres (α, β) , tal que:

1. (α, β) está presente em S ;
2. (α, β) ou $(-\beta, -\alpha)$ está presente em P .

A primeira etapa do algoritmo consiste em aplicar algumas atribuições de caracteres para reduzir o número de rótulos multiplicados nas strings. São utilizadas três estratégias para determinar essas atribuições, e todas se baseiam na ideia de encontrar dois caracteres α e β tais que:

- α corresponde a um rótulo multiplicado;
- β corresponde a um rótulo que não é multiplicado;
- (α, β) é um *match* com sinais.

Quando o par (α, β) é encontrado, ambos os caracteres α , de S e de P , são substituídos por um novo caractere $\gamma \notin \Sigma_S$.

As primeiras duas estratégias são versões mais restritas dessa ideia. Na primeira estratégia, α deve ser adjacente a dois caracteres sem cópias, e não apenas um. Na segunda estratégia, o rótulo $|\alpha|$ deve ser apenas duplicado, e cada uma de suas ocorrências deve pertencer a um *match* com sinais. Os autores provaram que essas estratégias têm um impacto baixo na distância de reversão entre as strings, em particular, eles provaram que a aplicação da segunda estratégia não altera a distância de reversão. A terceira estratégia encontra os *matches* com sinais sem nenhuma restrição adicional.

Exemplo 32. Aplicações das atribuições de caracteres em duas strings com sinais S e P . As strings S e P são transformadas em duas strings S' e P' após a aplicação da primeira estratégia: o caractere A das triplas $(-F +A -H)$ e $(+H -A +F)$, de S e P respectivamente, é trocado pelo caractere M . Em seguida, S' e P' são transformadas em S'' e P'' com a aplicação da segunda estratégia: o caractere D dos pares $(-G -D)$ e $(+D +G)$, de S' e P' respectivamente, é substituído pelo caractere I e o caractere D dos pares $(-K +D)$ de ambas as strings é substituído pelo caractere J . Por fim, S'' e P'' são transformadas em S''' e P''' utilizando a terceira estratégia: o caractere A do par $(+E +A)$ de ambas as strings é substituído pelo caractere L .

$$\begin{aligned}
S &= (-A +C +B -G -D \underline{-F +\mathbf{A} -H} -K +D +A -C +E +A) \\
P &= (-K +D \underline{+H -\mathbf{A} +F} +D +G +E +A -B -C +A -C +A)
\end{aligned}$$

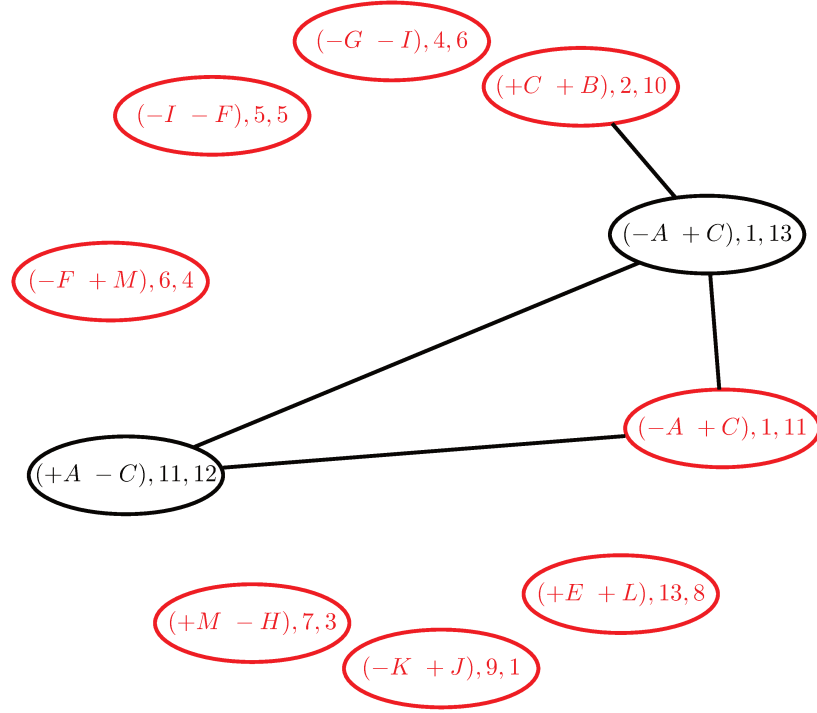
$$\begin{aligned}
S' &= (-A +C +B \underline{-G -\mathbf{D} -F} +M \underline{-H -K +\mathbf{D}} +A -C +E +A) \\
P' &= (\underline{-K +\mathbf{D} +H -M +F} \underline{+\mathbf{D} +G} +E +A -B -C +A -C +A)
\end{aligned}$$

$$\begin{aligned}
S'' &= (-A +C +B \underline{-G -I -F} +M \underline{-H -K +J} +A -C \underline{+E +\mathbf{A}}) \\
P'' &= (\underline{-K +J +H -M +F} \underline{+I +G} \underline{+E +\mathbf{A}} -B -C +A -C +A)
\end{aligned}$$

$$\begin{aligned}
S''' &= (-A +C +B -G -I -F +M -H -K +J +A -C \underline{+E +L}) \\
P''' &= (-K +J +H -M +F +I +G \underline{+E +L} -B -C +A -C +A)
\end{aligned}$$

Após as atribuições de caracteres iniciais, o algoritmo constrói um *grafo de matches*. Nesse grafo, o conjunto de vértices é composto pelos *matches* com sinais de S em P , e existe uma aresta entre dois vértices se os *matches* correspondentes têm o par de S ou de P em comum, ou se não for possível criar uma partição com sinais de S e P de forma que os dois *matches* correspondentes não sejam separados pela partição. Os autores provaram que um conjunto de vértices independentes desse grafo (ou seja, um conjunto de vértices onde nenhum par é ligado por uma aresta) corresponde a uma partição com sinais entre S e P . Nessa correspondência, um conjunto independente de tamanho máximo implica em uma partição de tamanho mínimo. Para o problema de encontrar um conjunto independente é utilizado um algoritmo guloso, que garante um fator de aproximação 2 para o problema dual (problema de cobertura de vértices).

Exemplo 33. Um grafo de *matches* das strings S''' e P''' obtida após a aplicação das estratégias no Exemplo 32. Cada vértice é rotulado com o par de caracteres do *match* corresponde e as posições onde esse par aparece em S''' e em P''' (em P''' o par pode estar invertido). Os vértices vermelhos formam um conjunto independente de tamanho máximo e os vértices pretos formam uma cobertura de vértices de tamanho mínimo. A partição com sinais $(\mathbb{S}, \mathbb{P}, \phi)$ é a partição de S''' e P''' correspondente ao conjunto independente indicado.



$$\begin{aligned}
 \mathbb{S} &= \langle (-A + C + B), (-G - I - F + M - H), (-K + J), (+A), (-C), (+E + L) \rangle \\
 \mathbb{P} &= \langle (-K + J), (+H - M + F + I + G), (+E + L), (-B - C + A), (-C), (+A) \rangle \\
 \phi &= (\begin{array}{cccccc} 3 & 2 & 6 & 1 & 5 & 4 \end{array})
 \end{aligned}$$

Para utilizar o algoritmo do PSOAR nas outras variações de problemas de partição, basta considerar as seguintes definições alternativas de *match*, sendo o *match* direto utilizado no problema MCSP e o *match* reverso utilizado no problema RMCSP.

Definição 17. Dado um par de strings sem sinais S e P , um *match direto* de S em P é um par de caracteres (α, β) que aparece em S e em P .

Definição 18. Dado um par de strings sem sinais S e P , um *match reverso* de S em P é um par de caracteres (α, β) , tal que:

1. (α, β) está presente em S ;
2. (α, β) ou (β, α) está presente em P .

Hitting Set (HS)

A seguir, descrevemos o algoritmo baseado em *hitting set* (HS), desenvolvido por Kolman e Waleń [42]. Um ponto interessante desse algoritmo é que ele garante um fator de aproximação $4k$, onde $k = occ(S)$, para o problema MCSP considerando as strings S e P . Esse é o melhor fator de aproximação conhecido até o momento. O algoritmo foi descrito para o problema MCSP, mas os próprios autores explicam como adaptá-lo para os problemas SMCS e RMCSP. Com a adaptação, o fator de aproximação sobe para $8k$.

O objetivo do algoritmo é garantir que todas as substrings que ocorrem com frequências diferentes em S e em P (ou, seja que aparecem mais vezes em S ou em P) são separadas em pelo menos dois blocos diferentes nas sequências \mathbb{S} e \mathbb{P} . De fato, essa propriedade é suficiente para garantir que as sequências formam uma partição.

O seguinte processo é utilizado pelo algoritmo. Seja T o conjunto de substrings de S e P com frequências diferentes, o algoritmo obtém um conjunto $T_{min} = \{X \in T : Y \not\subset X, \forall Y \in T\}$. Ou seja, T_{min} é o conjunto de strings minimais de T . Em seguida, o algoritmo gera uma partição separando, em S e em P , todos os pares de caracteres α e β que aparecem no início (primeiros dois caracteres) ou no fim (últimos dois caracteres) de alguma strings de T_{min} .

Por questão de eficiência o algoritmo utiliza um superconjunto de T_{min} , mas a ideia é a mesma. Para generalizar o algoritmo para os problemas RMCSP e SMCSP basta considerar que as strings X e $inv(X)$ são iguais. O artigo original [42] dá mais detalhes sobre como a adaptação é feita.

Exemplo 34. Obtenção de uma partição com sinais $(\mathbb{S}, \mathbb{P}, \phi)$ de um par de strings S e P a partir do HS (adaptado para o problema SMCSP).

$$S = (-A +C +B -D -F +A -K +D +A -C +A +E +A)$$

$$P = (-K +D -A -A +F +D +E +A -B -C +A -C +A)$$

$$T_{min} = \{(+B -D), (-A -K), (+D +A), (+A +E), (+D -A), \\ (-A -A), (+D +E), (+A -B), (-C +A -C)\}$$

$$\mathbb{S} = \langle (-A), (+C +B), (-D -F +A), (-K +D), (+A), (-C), (+A), (+E +A) \rangle$$

$$\mathbb{P} = \langle (-K +D), (-A), (-A +F +D), (+E +A), (-B -C), (+A), (-C), (+A) \rangle$$

$$\phi = (\begin{array}{cccccccc} 4 & 1 & 3 & 8 & 2 & 5 & 6 & 7 \end{array})$$

Algoritmos Gulosos (GREEDY e GREEDY*)

Vários algoritmos gulosos foram propostos para o problema MCSP e aqui descrevemos dois deles. O primeiro deles, que chamamos de GREEDY, é descrito em detalhes por Chrobak e coautores [22], que provaram que o fator de aproximação desse algoritmo está entre $\Omega(n^{0,43})$ e $O(n^{0,69})$. Já o segundo, que chamamos de GREEDY*, é uma nova versão do primeiro proposta por He [37].

Esses dois algoritmos foram desenvolvidos para o problema MCSP mas, assim como no caso do algoritmo HS, é possível generalizá-los também para os problemas RMCSP e SMCSP se considerarmos que as strings X e $inv(X)$ são iguais.

Explicamos a forma como o algoritmo GREEDY encontra a partição mínima entre duas strings S e P de forma recursiva.

Inicialmente, temos as sequências $\mathbb{S} = \langle S \rangle$ e $\mathbb{P} = \langle P \rangle$. Em cada passo, o algoritmo encontra a maior substring comum X entre um bloco $A = A'XA''$ de \mathbb{S} e um bloco $B = B'XB''$ de \mathbb{P} , nesse ponto é necessário que pelo menos uma das strings A' ou A'' e pelo menos uma das strings B' ou B'' não seja vazia. Em seguida, o algoritmo troca o bloco A da sequência \mathbb{S} pelos blocos A' , X e A'' . Além disso, o algoritmo troca o bloco B da sequência \mathbb{P} pelos blocos B' , X e B'' . Note que, caso algum desses blocos seja vazio ele não é considerado. Esse processo se repete até que as sequências \mathbb{S} e \mathbb{P} sejam uma partição.

O algoritmo GREEDY* é similar ao GREEDY, mas ele aproveita a existência de rótulos com apenas uma cópia para melhorar a escolha dos blocos. Qualquer substring de S que contenha um caractere cujo rótulo possua apenas uma cópia pode ter no máximo uma substring correspondente em P . Para aproveitar esse fato, o algoritmo GREEDY* escolhe um rótulo arbitrário que possui apenas uma cópia. A maior substring comum de S e P , que contenha um caractere com o rótulo escolhido, é então transformada em um bloco. Strings que possuem apenas caracteres cujos rótulos são multiplicados são escolhidas apenas quando todos os rótulos com apenas uma cópia já foram selecionados.

Exemplo 35. Obtenção de uma partição com sinais $(\mathbb{S}_5, \mathbb{P}_5, \phi)$ de um par de strings S e P a partir do algoritmo GREEDY* (adaptado para o problema SMCSP). Os índices nas sequências \mathbb{S}_i e \mathbb{P}_i indicam em qual iteração elas foram geradas (as sequências \mathbb{S}_0 e \mathbb{P}_0 são as iniciais). Em cada passo intermediário o carácter com apenas uma cópia escolhido está em negrito (quando ele existir) e a maior substrings comum selecionada está sublinhada.

$$\begin{aligned} S &= (-A +C +B -G -D -F +A -H -K +D +A -C +E +A) \\ P &= (-K +D +H -A +F +D +G +E +A -B -C +A -C +A) \end{aligned}$$

$$\begin{aligned} \mathbb{S}_0 &= \langle (-A +C +B -G -D -F +A -H -K +D +A -C +\underline{\mathbf{E}} +A) \rangle \\ \mathbb{P}_0 &= \langle (-K +D +H -A +F +D +G +\underline{\mathbf{E}} +A -B -C +A -C +A) \rangle \end{aligned}$$

$$\begin{aligned} \mathbb{S}_1 &= \langle (-A +C +B -\underline{G} -D -\mathbf{F} +A -H -K +D +A -C), (+E +A) \rangle \\ \mathbb{P}_1 &= \langle (-K +D +\underline{H} -A +\mathbf{F} +D +G), (+E +A), (-B -C +A -C +A) \rangle \end{aligned}$$

$$\begin{aligned} \mathbb{S}_2 &= \langle (-A +C +B), (-G -D -F +A -H), (-\mathbf{K} +D +A -C), (+E +A) \rangle \\ \mathbb{P}_2 &= \langle (-\mathbf{K} +D), (+H -A +F +D +G), (+E +A), (-B -C +A -C +A) \rangle \end{aligned}$$

$$\mathbb{S}_3 = \langle (-A + C + B), (-G - D - F + A - H), (-K + D), (+A - C), (+E + A) \rangle$$

$$\mathbb{P}_3 = \langle (-K + D), (+H - A + F + D + G), (+E + A), (-B - C + A - C + A) \rangle$$

$$\mathbb{S}_4 = \langle (-A + C + B), (-G - D - F + A - H), (-K + D), (+A - C), (+E + A) \rangle$$

$$\mathbb{P}_4 = \langle (-K + D), (+H - A + F + D + G), (+E + A), (-B - C + A), (-C + A) \rangle$$

$$\mathbb{S}_5 = \langle (-A + C + B), (-G - D - F + A - H), (-K + D), (+A), (-C), (+E + A) \rangle$$

$$\mathbb{P}_5 = \langle (-K + D), (+H - A + F + D + G), (+E + A), (-B - C + A), (-C), (+A) \rangle$$

$$\phi = \begin{pmatrix} 3 & 2 & 6 & 1 & 5 & 4 \end{pmatrix}$$

4.3.4 Heurística de Combinação

Desenvolvemos também uma heurística que combina blocos de uma partição, reduzindo seu tamanho. Essa heurística pode tanto ser utilizada em strings originais para gerar uma solução, bem como para melhorar as soluções dos algoritmos HS e PSOAR. Isso se deve ao fato das soluções desses algoritmos não serem minimais, ou seja, é possível obter uma partição ainda menor sem quebrar os blocos já definidos.

Inicialmente, explicamos a heurística para o problema MCSP e, em seguida, descrevemos as modificações necessárias para os outros problemas de partição. A heurística inicia com duas sequências de strings \mathbb{S} e \mathbb{P} , correspondentes a uma partição. São aplicadas várias operações de combinação de blocos até que as duas sequências sejam uma partição minimal, onde não é possível mais aplicar essa operação.

A operação de combinação de blocos consiste em encontrar um par de blocos adjacentes em ambas as sequências \mathbb{S} e \mathbb{P} , e combiná-los em um único bloco. Como a ordem em que os blocos são combinados afeta quais blocos poderão ser combinados mais tarde, temos que utilizar alguma estratégia que faça boas escolhas. Para essas escolhas, utilizamos uma ideia similar à do algoritmo GREEDY*, privilegiando blocos que possuam caracteres cujos rótulos não sejam multiplicados.

Inicialmente, procuramos pares de blocos consecutivos tais que ambos possuam um caractere cujo rótulo não seja multiplicado. Em seguida, escolhemos pares onde apenas um dos blocos contém um caractere cujo rótulo não é multiplicado. Finalmente, juntamos os demais blocos. Em cada etapa, dentre os blocos disponíveis, escolhemos os primeiros que aparecem na sequência \mathbb{S} . É importante ressaltar que, no problema MCSP, os blocos combinados devem aparecer na mesma ordem em \mathbb{S} e em \mathbb{P} .

O Algoritmo 10 apresenta um pseudocódigo dessa heurística. Podemos escolher, como bloco das sequências iniciais \mathbb{S} e \mathbb{P} , os caracteres de S e P , respectivamente, ou os blocos de uma partição resultante de algum outro algoritmo.

Algoritmo 10: Combinação

Entrada: Sequências de strings \mathbb{S} e \mathbb{P} correspondentes a uma partição

Saída: Sequências correspondentes a uma partição minimal

```

1 início
2   enquanto existe um par de blocos consecutivos em  $\mathbb{S}$  e em  $\mathbb{P}$  faça
3     se existe um par de blocos consecutivos em  $\mathbb{S}$  e em  $\mathbb{P}$ , tal que ambos
4       contêm um caractere cujo rótulo não é multiplicado então
5          $A, B \leftarrow$  primeiro par de blocos de  $\mathbb{S}$  satisfazendo essa propriedade
6       senão se existe um par de blocos consecutivos em  $\mathbb{S}$  e em  $\mathbb{P}$ , tal que um
7         deles contenha um caractere cujo rótulo não é multiplicado então
8            $A, B \leftarrow$  primeiro par de blocos de  $\mathbb{S}$  satisfazendo essa propriedade
9         senão
10           $A, B \leftarrow$  primeiro par de blocos de  $\mathbb{S}$ , consecutivos em  $\mathbb{S}$  e em  $\mathbb{P}$ 
11           $\mathbb{S} \leftarrow \mathbb{S}$  com os blocos  $A$  e  $B$  combinados
12           $\mathbb{P} \leftarrow \mathbb{P}$  com os blocos  $A$  e  $B$  combinados
13   retorna  $\mathbb{S}$  e  $\mathbb{P}$ 

```

Exemplo 36. A heurística de Combinação sendo aplicada em duas sequências de strings \mathbb{S} e \mathbb{P} , formadas inicialmente pelos caracteres de duas strings sem sinais S e P .

$$\begin{aligned}
 S &= (C \ E \ F \ B \ A \ A \ C \ D \ G \ B) \\
 P &= (A \ C \ C \ B \ D \ G \ E \ F \ B \ A) \\
 \mathbb{S} &= \langle (C), (E), (F), (B), (A), (A), (C), (D), (G), (B) \rangle \\
 \mathbb{P} &= \langle (A), (C), (C), (B), (D), (G), (E), (F), (B), (A) \rangle
 \end{aligned}$$

A seguir, apresentamos as sequências \mathbb{S} e \mathbb{P} , após a combinação de dois pares de blocos, onde ambos os blocos possuem um caractere cujo rótulo não é multiplicado. Inicialmente são combinados os blocos (E) e (F) . Em seguida, são combinados os blocos (D) e (G) .

$$\begin{aligned}
 \mathbb{S} &= \langle (C), (E \ F), (B), (A), (A), (C), (D \ G), (B) \rangle \\
 \mathbb{P} &= \langle (A), (C), (C), (B), (D \ G), (E \ F), (B), (A) \rangle
 \end{aligned}$$

A seguir, estão as sequências \mathbb{S} e \mathbb{P} , após duas combinações de blocos, a primeira juntando os blocos $(E \ F)$ e (B) , e a segunda juntando o bloco resultante com o bloco (A) . Essas combinações foram escolhidas pois $(E \ F)$ possui um caractere cujo rótulo tem apenas uma ocorrência.

$$\begin{aligned}
 \mathbb{S} &= \langle (C), (E \ F \ B \ A), (A), (C), (D \ G), (B) \rangle \\
 \mathbb{P} &= \langle (A), (C), (C), (B), (D \ G), (E \ F \ B \ A) \rangle
 \end{aligned}$$

Por fim, apresentamos as sequências \mathbb{S} e \mathbb{P} , após a combinação dos blocos (A) e (C) . Esses são os únicos blocos que podem ser combinados nessa iteração.

$$\begin{aligned}\mathbb{S} &= \langle (C), (E F B A), (\textcolor{red}{A} \textcolor{red}{C}), (D G), (B) \rangle \\ \mathbb{P} &= \langle (\textcolor{red}{A} \textcolor{red}{C}), (C), (B), (D G), (E F B A) \rangle\end{aligned}$$

Para que essa heurística possa ser utilizada também nos problemas SMCSP e RMCSP, basta adaptar a condição para combinar os blocos. Nessa adaptação, dois blocos A e B , consecutivos em \mathbb{S} , podem ser combinados se A e B também forem consecutivos em \mathbb{P} ou se $\text{inv}(B)$ e $\text{inv}(A)$ forem consecutivos em \mathbb{P} . Quando combinamos os blocos de \mathbb{S} , também combinamos os blocos correspondentes em \mathbb{P} .

4.4 O Problema de Empacotamento Máximo de Ciclos

Nesta seção, descrevemos o problema de Empacotamento Máximo de Ciclos (EMC), um dos subproblemas resolvidos pelo algoritmo SOAR [19] (no artigo os autores chamam esse problema de “Maximum Cycle Decomposition”). Conforme explicado na Seção 4.3.3, o algoritmo SOAR começa gerando um par de permutações reduzidas \hat{S} e \hat{P} através de uma partição entre as strings originais S e P . Após essa etapa, uma solução do problema EMC é usada para obter um mapeamento de \hat{S} e \hat{P} em uma permutação. Como nas heurísticas propostas, a distância pode ser obtida utilizando algoritmos existentes para permutações.

Aqui consideramos que as strings S e P estão estendidas conforme explicado na Seção 4.3.2 e, conseqüentemente, podemos considerar que \hat{S} e \hat{P} também estão estendidas. Além disso, utilizamos duas strings sem sinais \tilde{S} e \tilde{P} obtidas a partir de \hat{S} e \hat{P} , respectivamente. Para gerar \tilde{S} , substituímos cada caractere com sinal $\alpha \in \hat{S}, \alpha \notin \{\hat{S}_1, \hat{S}_{|\hat{S}|}\}$, por um par de caracteres sem sinais $(\alpha^h \alpha^t)$, caso α tenha sinal positivo, ou $(\alpha^t \alpha^h)$, caso contrário. Também removemos os sinais dos caracteres \hat{S}_1 e $\hat{S}_{|\hat{S}|}$. Aplicamos as mesmas substituições em \hat{P} para gerar \tilde{P} .

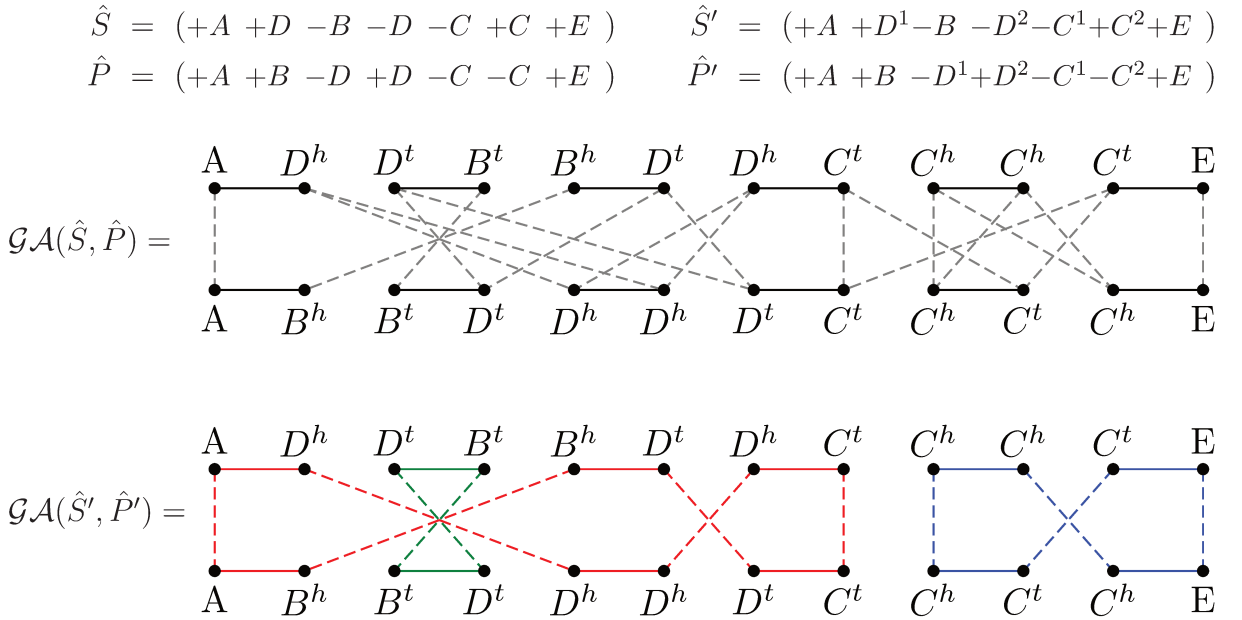
O *grafo de adjacências* $\mathcal{GA}(\hat{S}, \hat{P}) = (V, E)$ de um par de strings \hat{S} e \hat{P} , é formado pelo conjunto de vértices V , composto pelos caracteres de \tilde{S} e \tilde{P} , e pelo conjunto de arestas não direcionadas $E = E_b \cup E_g$, composto pelos conjuntos de arestas pretas (E_b) e arestas cinzas (E_g). Cada par de caracteres α e β consecutivos em \tilde{S} e não correspondentes ao mesmo caractere em \hat{S} são ligados por uma aresta preta. O mesmo ocorre com pares de caracteres consecutivos em \tilde{P} que não são correspondentes ao mesmo caractere de \hat{P} . Além disso, um caractere α de \tilde{S} e um caractere β de \tilde{P} são ligados por uma aresta cinza se seus rótulos são iguais. Dizemos que duas arestas cinzas são *gêmeas* se elas conectam vértices originados de um mesmo caractere em \hat{S} a vértices originados de um mesmo caractere em \hat{P} .

Um *ciclo alternado* do grafo de adjacências $\mathcal{GA}(\hat{S}, \hat{P})$ é um ciclo composto por arestas alternadas entre pretas e cinzas. Um *empacotamento em ciclos alternados* do grafo $\mathcal{GA}(\hat{S}, \hat{P})$ é um conjunto disjunto de ciclos alternados, tal que:

1. cada vértice pertence a exatamente um ciclo;
2. uma aresta cinza pertence a um ciclo se e somente se ela não tem uma aresta gêmea ou sua aresta gêmea também pertença a algum ciclo.

Dado um empacotamento em ciclos alternados do grafo $\mathcal{GA}(\hat{S}, \hat{P})$ e um mapeamento qualquer para \hat{P} , podemos obter um mapeamento para \hat{S} . Note que, devido às condições de empacotamento em ciclos alternados, os vértices correspondentes a um caractere α de \hat{S} estão ligados por arestas cinzas apenas aos vértices correspondentes a um único caractere β de \hat{P} . Para obtermos o mapeamento de \hat{S} basta mapearmos seus caracteres de acordo com o mapeamento dos caracteres correspondentes (pelas conexões por arestas cinzas) em \hat{P} .

Exemplo 37. Dadas duas strings com sinais \hat{S} e \hat{P} , temos o grafo de adjacências $\mathcal{GA}(\hat{S}, \hat{P})$ e um empacotamento desse grafo em 3 ciclos de arestas alternadas (indicados por cores distintas). As permutações com sinais \hat{S}' e \hat{P}' são as correspondentes a esse empacotamento. Nesse exemplo as arestas pretas são linhas contínuas e as arestas cinzas são linhas tracejadas.



O problema de Empacotamento Máximo de Ciclos corresponde a encontrar um empacotamento em ciclos alternados do grafo \mathcal{GA} que possua o maior número de ciclos possível.

Como esse é um problema NP-Difícil [52], no algoritmo SOAR os autores optaram por utilizar uma heurística gulosa. A heurística consiste em escolher um vértice arbitrário que ainda não seja coberto pelo empacotamento e adicionar no empacotamento o menor ciclo que contenha este vértice. Tal ciclo pode ser obtido por uma busca em largura no grafo \mathcal{GA} .

Note que o mapeamento correspondente ao empacotamento obtido pode ser usado como um dos mapeamentos iniciais das heurísticas propostas. No problema *DT*, onde as strings reduzidas não tem sinais, utilizamos a mesma definição de grafo de adjacências ao supor que todos os caracteres possuem sinais positivos.

4.5 Experimentos Práticos

Nesta seção, apresentamos os testes realizados para avaliar as heurísticas propostas. Assim como no Capítulo 3, utilizamos o algoritmo SOAR [19] e suas adaptações para comparação.

Os parâmetros utilizados em cada heurística são os mesmos que foram escolhidos no Capítulo 3. A Seção 4.5.1 descreve o processo realizado para a geração das bases de dados utilizadas. A Seção 4.5.2 apresenta uma comparação de diferentes algoritmos para os problemas de partição. Por fim, a Seção 4.5.3 apresenta os resultados das heurísticas generalizadas.

4.5.1 Bases de Dados

Para testar as heurísticas generalizadas, desenvolvemos novas bases de dados. Geramos duas bases aleatórias com poucos rótulos multiplicados. Além dessas duas bases, foram geradas duas bases específicas para cada modelo, totalizando 12 bases de dados. Cada uma das duas bases geradas para um modelo específico utiliza uma estratégia diferente para aumentar o número de ocorrências de alguns rótulos. Para simplificar, consideramos que cada rótulo é representado por um número inteiro.

Base RAND

Esta base de dados foi gerada de forma totalmente aleatória, e é composta por 10 conjuntos com 1000 pares de strings, sendo cada conjunto com um tamanho de string fixo. Os tamanhos das strings variam de 100 até 1000 em intervalos de 100. O seguinte processo foi utilizado para criação de cada par de strings de tamanho n :

1. Geramos a string de origem S escolhendo aleatoriamente cada caractere no alfabeto $\Sigma = \{1, \dots, n\}$. As escolhas aleatórias são realizadas com repetição para que um rótulo possa ter mais de uma ocorrência.
2. Geramos a string de destino P escolhendo aleatoriamente uma nova ordem para os caracteres de S .

Base SRAND

Esta base foi gerada da mesma forma que a base RAND, mas foi escolhido um sinal aleatório para cada elemento de ambas as strings S e P .

Base R-O

Esta base de dados é composta por 9 conjuntos, cada um com 1000 pares de strings sem sinais. Todos os conjuntos são formados por strings de tamanho 500 e possuem 250 rótulos com apenas uma ocorrência. Os conjuntos se distinguem pela ocorrências máxima dos demais rótulos, que varia de 2 até 10 em intervalos de 1. O seguinte processo foi utilizado para a criação de cada par de strings com uma ocorrência máxima o :

1. Para gerar a string de origem, distribuímos de forma aleatória caracteres correspondentes aos rótulos de 1 até $250 + \lceil \frac{250}{o} \rceil$. Temos apenas um caractere para os rótulos de 1 até 250, mas temos até o caracteres correspondentes aos rótulos de 251 até $250 + \lceil \frac{250}{o} \rceil$. Quando necessário, apenas um dos rótulos multiplicados possui menos de o ocorrências e, nesse caso, ele terá $250 \bmod o$ ocorrências.
2. A string de destino é gerada aplicando 250 operações de reversão. Os valores de i e j de cada reversão $\rho(i, j)$ são escolhidos de forma aleatória, sendo que $1 \leq i < j \leq 500$.

Base SR-O

Esta base foi construída da mesma forma que a base R-O, mas na geração da string de origem atribuímos aleatoriamente um sinal positivo ou negativo para cada um de seus caracteres. Também permitimos a aplicação de reversões $\rho(i, i)$.

Base T-O

As strings de origem desta base foram construídas da mesma forma que as strings de origem da base R-O e, para gerar as strings de destino, aplicamos 250 operações de transposição. Os valores de i , j e k de cada transposição $\tau(i, j, k)$ são escolhidos de forma aleatória, sendo que $1 \leq i < j < k \leq 501$.

Base RT-O

As strings de origem desta base também foram construídas da mesma forma que as strings de origem da base R-O e, para gerar as strings de destino, aplicamos tanto operações de reversão quanto operações de transposição. São aplicadas 125 reversões e 125 transposições, cujos índices são escolhidos de forma aleatória respeitando as mesmas restrições das bases R-O e T-O, respectivamente. Além disso, escolhemos uma ordem aleatória para aplicar todas as 250 operações.

Base SRT-O

Esta base foi construída da mesma forma que a base RT-O, mas na geração da string de origem atribuímos aleatoriamente um sinal positivo ou negativo para cada um de seus caracteres. De modo similar à base SR-O, aqui permitimos a aplicação de reversões $\rho(i, i)$.

Base R-L

Esta base de dados é composta por 10 conjuntos, cada um com 1000 pares de strings sem sinais, e todos os conjuntos são formados por strings de tamanho 500. Os conjuntos se distinguem pelo tamanho do alfabeto utilizado na geração das strings, que varia de 50 até 500 em intervalos de 50. O seguinte processo foi utilizado para a criação de cada par de strings considerando um alfabeto de tamanho ℓ :

1. Geramos a string de origem S escolhendo aleatoriamente cada caractere no alfabeto $\Sigma = \{1, \dots, \ell\}$. De modo similar à base RAND, as escolhas aleatórias são realizadas com repetição para que um caractere possa ter mais de uma cópia.
2. A string de destino é gerada aplicando 250 operações de reversão. Os valores de i e j de cada reversão $\rho(i, j)$ são escolhidos de forma aleatória, sendo que $1 \leq i < j \leq 500$.

Bases SR-L, T-L, RT-L e SRT-L

Assim como as bases SR-O, T-O, RT-O e SRT-O, que foram geradas a partir de modificações no processo de geração da base R-O, as bases SR-L, T-L, RT-L e SRT-L foram geradas a partir de modificações similares no processo de geração da base R-L.

4.5.2 Algoritmos para Partição

Como o número de mapeamentos possíveis para strings com rótulos multiplicados é elevado, as strings são simplificadas antes de executar as heurísticas propostas. Essas simplificações são realizadas utilizando algoritmos para os problemas de partição. Quando estamos levando em conta o evento de transposição, as strings são simplificadas utilizando um algoritmo para o problema MCSP. No caso em que tratamos do evento de reversão ou de ambos os eventos de reversão e transposição, as strings com sinais são simplificadas utilizando um algoritmo para o problema SMCS, e as strings sem sinais são simplificadas utilizando um algoritmo para o problema RMCS. Nesta seção, comparamos os algoritmos para escolher aqueles que são utilizados em cada variação do problema de partição.

Os algoritmos para o problema MCSP foram testados utilizando as strings da base T-O. Essa base foi escolhida por nos permitir ter uma ideia de como os resultados variam com o aumento no número de cópias e por ser gerada a partir do evento de transposição, que está relacionado ao problema MCSP. Por razões similares, os algoritmos para o problema SMCS foram testados com as strings da base SR-O, e os algoritmos para o problema RMCS foram testados com as strings da base R-O.

As tabelas 4.1, 4.2 e 4.3 apresentam as médias dos tamanhos das strings reduzidas encontradas por cada um dos algoritmos, para os problemas MCSP, SMCSP e RMCSP, respectivamente. As colunas PSOAR, HS, COMBI, GREEDY e GREEDY* apresentam os resultados para os algoritmos descritos na Seção 4.3.3. Além disso, as colunas HS* e PSOAR* apresentam os resultados dos algoritmos PSOAR e HS, respectivamente, após a aplicação da heurística de combinação de blocos.

Pela Tabela 4.1, vemos que os algoritmos COMBI, GREEDY, GREEDY*, HS* e PSOAR* obtiveram resultados bem próximos entre si, e melhores que os resultados dos algoritmos PSOAR e HS. Dentre esses, o algoritmo PSOAR* apresentou os melhores resultados, portanto ele foi escolhido para ser usado na simplificação das strings quando levamos em conta o problema MCSP. É importante ressaltar que a heurística de combinação de blocos melhorou consideravelmente os resultados dos algoritmos PSOAR e HS. Além disso, o algoritmo HS* garante as mesmas aproximações teóricas do algoritmo HS, o que o torna uma alternativa interessante para esse problema.

Tabela 4.1: Média dos tamanhos das strings reduzidas obtidas com diferentes algoritmos para o problema MCSP com a base de dados T-O.

$occ(S)$	PSOAR	HS	COMBI	GREEDY	GREEDY*	HS*	PSOAR*
2	378,27	432,38	378,11	378,11	378,11	378,11	378,10
3	377,92	431,90	377,41	377,43	377,40	377,42	377,40
4	377,21	431,54	376,17	376,19	376,14	376,15	376,10
5	377,43	431,47	375,80	375,83	375,74	375,78	375,69
6	377,55	431,46	375,12	375,19	375,03	375,07	374,96
7	377,60	431,94	374,19	374,21	374,00	374,10	373,93
8	378,00	432,33	373,61	373,69	373,40	373,50	373,28
9	378,40	432,25	372,82	372,92	372,59	372,72	372,45
10	378,03	432,72	371,23	371,37	370,95	371,11	370,76

Pela Tabela 4.2, vemos que os resultados dos algoritmos COMBI, GREEDY, GREEDY*, HS* e PSOAR* continuam bem próximos entre si, e melhores que os resultados dos algoritmos PSOAR e HS. O algoritmo PSOAR* apresentou novamente os melhores resultados, e foi escolhido para ser usado na simplificação das strings quando levamos em conta o problema SMCSP. Assim como ocorreu com os testes do problema MCSP, a heurística de combinação de blocos melhorou os resultados dos algoritmos PSOAR e HS, principalmente quando as strings apresentam uma quantidade maior de ocorrências de cada rótulo.

Observando os resultados da Tabela 4.3, vemos que os algoritmos GREEDY, GREEDY* e PSOAR* obtiveram resultados bem próximos entre si. O algoritmo GREEDY* foi o melhor dentre eles, e foi escolhido para ser usado na simplificação das strings quando levamos em conta o problema RMCSP. Assim como nos dois casos anteriores, a heurística de combinação de blocos melhorou consideravelmente os resultados dos algoritmos PSOAR e HS, mas mesmo com essa heurística o tamanho médio das strings simplificadas geradas pelo algoritmo HS* foram bem maiores do que o tamanho médio das strings geradas pelos demais algoritmos.

Tabela 4.2: Média dos tamanhos das strings reduzidas obtidas com diferentes algoritmos para o problema SMCSF com a base de dados SR-O.

$occ(S)$	PSOAR	HS	COMBI	GREEDY	GREEDY*	HS*	PSOAR*
2	312,51	391,76	312,46	312,45	312,44	312,45	312,44
3	312,49	391,54	312,24	312,25	312,23	312,25	312,23
4	313,00	392,18	312,48	312,47	312,45	312,46	312,43
5	313,58	392,49	312,83	312,82	312,78	312,81	312,76
6	313,12	392,70	311,97	311,96	311,90	311,94	311,87
7	313,66	393,36	312,03	312,00	311,90	311,96	311,87
8	313,96	393,89	311,87	311,83	311,71	311,76	311,65
9	314,32	394,14	311,70	311,66	311,51	311,57	311,41
10	314,72	394,69	311,31	311,25	311,01	311,14	310,91

Tabela 4.3: Média dos tamanhos das strings reduzidas obtidas com diferentes algoritmos para o problema RMCSP com a base de dados R-O.

$occ(S)$	PSOAR	HS	COMBI	GREEDY	GREEDY*	HS*	PSOAR*
2	310,53	434,24	404,40	310,09	310,11	371,10	310,21
3	310,87	434,40	402,59	309,55	309,53	369,69	309,95
4	311,80	434,68	402,98	309,34	309,29	370,22	310,10
5	312,41	435,26	400,56	308,63	308,52	368,67	309,81
6	313,83	436,03	399,55	308,57	308,37	368,39	310,15
7	314,81	436,25	398,69	307,79	307,52	367,51	309,96
8	315,50	437,07	396,63	306,76	306,42	366,08	309,45
9	317,63	437,85	395,31	307,12	306,58	365,63	310,40
10	318,70	438,38	393,09	306,29	305,69	363,99	310,07

4.5.3 Resultados

Nesta seção, apresentamos os resultados das heurísticas propostas generalizadas. Como temos strings com uma quantidade elevada de ocorrências, aplicamos os algoritmos escolhidos na Seção 4.5.2 para simplificá-las. Para o algoritmo SOAR, como as strings já estão simplificadas, executamos apenas a etapa do empacotamento em ciclos, o que seria equivalente a executar o algoritmo SOAR substituindo seu próprio algoritmo de partição pelo novo algoritmo de partição. Denotamos essa adaptação do algoritmo SOAR por SOAR*.

Além de utilizar um conjunto inicial de mapeamentos totalmente aleatório, também apresentamos os resultados das heurísticas propostas levando em conta que o mapeamento obtido pelo empacotamento em ciclos do algoritmo SOAR foi adicionado ao conjunto de mapeamentos iniciais. Nos referimos às heurísticas AG, BL e GRASP com essa adaptação por AG*, BL* e GRASP*, respectivamente. É importante observar que as distâncias obtidas pelas heurísticas AG*, BL* e GRASP* são necessariamente menores ou iguais às distâncias obtidas pelo algoritmo SOAR*.

Nas tabelas 4.4 a 4.18, apresentamos as médias das distâncias obtidas por cada algoritmo testado em cada base de dados descritas na Seção 4.5.1.

Os conjuntos de cada base de dados são indexados por um valor diferente: nas bases RAND e SRAND usamos o tamanho das strings originais; nas bases R-O, SR-O, T-O, RT-O e SRT-O usamos o número máximo de ocorrências de um rótulo nas strings originais ($occ(S)$); nas bases R-L, SR-L, T-L, RT-L e SRT-L usamos o tamanho do alfabeto utilizado para a geração das strings originais ($|\Sigma_S|$).

As tabelas 4.4, 4.5 e 4.6 apresentam as médias das distâncias obtidas para o problema *DR* nas bases RAND, R-O e R-L, respectivamente. Podemos ver que em todos os conjuntos as heurísticas mais sofisticadas obtiveram resultados melhores que os resultados da heurística de mapeamentos aleatórios e que os resultados do algoritmo SOAR*.

Para a base RAND nos conjuntos com strings de tamanhos maiores que 100, a melhor heurística foi a BL*, que também foi a melhor heurística na base R-O para o conjunto onde a ocorrência máxima de rótulos nas strings é 2 e na base R-L para os conjuntos correspondentes a alfabetos de tamanhos 50 e 100. Na base RAND, para o conjunto com strings de tamanho 100, a melhor heurística foi a GRASP*. Na base R-O, a heurística AG foi a melhor nos conjuntos de strings em que as ocorrências máximas de rótulos são iguais a 3, 4 e 5. Nos demais conjuntos dessa base, a heurística GRASP* foi a que retornou os melhores resultados. Na base R-L, a heurística GRASP* foi a melhor em conjuntos correspondentes a alfabetos de tamanhos 150 até 300, e a heurística AG foi a melhor em conjuntos correspondentes a alfabetos de tamanhos 350 até 500.

Note também que, na base RAND, as variações das heurísticas que utilizam o mapeamento do algoritmo SOAR* em seu conjunto inicial foram melhores que as heurísticas sem esse mapeamento. Nas bases R-O e R-L existem vários casos onde as heurísticas que não consideram o mapeamento do algoritmo SOAR* encontraram distâncias menores do que as que utilizam o mapeamento, mas nesses casos a diferença das distâncias é normalmente pequena.

As tabelas 4.7, 4.8 e 4.9 apresentam as médias das distâncias obtidas para o problema *D \bar{R}* nas bases SRAND, SR-O e SR-L, respectivamente. É possível ver novamente que, em todos os conjuntos, as heurísticas mais sofisticadas obtiveram resultados melhores que os resultados da heurística de mapeamentos aleatórios e que os resultados do algoritmo SOAR*.

A melhor heurística para a base SRAND foi a BL*, que também foi a melhor heurística na base SR-O para os conjuntos onde as ocorrências máximas dos rótulos das strings são 2 e 3, e na base SR-L para os conjuntos correspondentes a alfabetos de tamanhos 50, 400, 450 e 500. Nos demais conjuntos das bases SR-O e SR-L, a heurística GRASP* foi a melhor.

Como nos testes para o problema *DR*, observamos que na base RAND as variações das heurísticas que utilizam o mapeamento do algoritmo SOAR* em seu conjunto inicial foram melhores que as heurísticas sem esse mapeamento. Além disso, nos casos onde a distância considerando o mapeamento do algoritmo SOAR é maior que a distância sem considerar esse mapeamento, a diferença da média entre as distâncias é bem pequena.

Também podemos ver que, para os conjuntos da base SR-O onde as ocorrências máximas dos rótulos das strings estão abaixo de 6 e para conjuntos da base SR-L com alfabetos de tamanhos acima de 200, as melhores heurísticas encontraram distâncias muito próximas das 250 reversões aplicadas para gerar essas bases de dados.

Tabela 4.4: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DR* com as strings da base RAND simplificadas com o algoritmo GREEDY*.

Tamanho	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
100	85,86	83,44	83,36	83,66	83,50	83,35	83,32	85,50
200	183,37	177,47	177,70	177,72	177,52	176,66	177,19	178,87
300	281,96	273,00	273,99	273,56	272,24	270,88	272,36	273,23
400	380,92	369,32	371,01	370,21	367,13	365,61	367,67	368,10
500	480,36	466,45	468,94	467,64	462,61	460,86	463,23	463,51
600	579,74	563,62	566,21	565,61	557,90	556,15	558,54	558,62
700	679,31	661,24	663,77	663,65	653,51	651,71	654,22	654,32
800	778,94	759,24	762,10	762,20	749,44	747,58	750,16	750,18
900	878,70	857,04	859,84	860,93	845,05	843,19	845,77	845,79
1000	978,25	955,14	957,30	959,73	941,11	939,17	941,75	941,78

Tabela 4.5: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DR* com as strings da base R-O simplificadas com o algoritmo GREEDY*.

$occ(S)$	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
2	292,92	291,77	291,83	291,82	291,82	291,77	291,82	294,76
3	292,73	288,94	288,80	288,67	288,80	288,96	288,81	293,24
4	293,05	288,01	287,44	287,33	287,71	288,23	287,42	292,33
5	292,93	287,26	286,33	286,19	287,06	287,36	286,26	290,93
6	292,91	286,97	285,73	285,63	286,86	286,88	285,60	289,86
7	292,59	286,58	285,08	285,07	286,35	286,20	285,00	288,68
8	291,78	285,80	284,32	284,34	285,50	285,06	284,03	287,10
9	292,22	286,33	284,72	284,71	285,58	285,10	284,20	286,74
10	291,58	285,73	284,05	284,05	284,53	283,98	283,37	285,35

Tabela 4.6: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DR* com as strings da base R-L simplificadas com o algoritmo GREEDY*.

$ \Sigma_S $	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
50	286,77	278,90	278,12	278,34	270,86	270,52	271,01	271,05
100	292,62	284,00	282,72	283,17	282,67	281,53	281,71	283,42
150	292,92	284,69	283,35	283,69	285,19	284,15	283,24	287,17
200	293,38	285,82	284,72	285,03	286,51	285,95	284,72	289,70
250	293,26	286,49	285,56	285,68	286,77	286,65	285,55	290,90
300	293,16	287,16	286,48	286,50	287,17	287,34	286,46	291,85
350	293,00	287,62	287,10	287,07	287,48	287,73	287,08	292,31
400	292,83	287,99	287,69	287,63	287,83	288,15	287,68	292,71
450	292,78	288,51	288,35	288,21	288,33	288,63	288,33	293,05
500	292,82	288,98	288,88	288,79	288,87	289,04	288,90	293,46

Tabela 4.7: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}$ com as strings da base SRAND simplificadas com o algoritmo PSOAR*.

Tamanho	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
100	91,56	88,73	88,92	88,89	88,82	88,43	88,65	90,12
200	190,22	184,85	185,64	184,86	183,24	182,97	184,19	184,68
300	289,53	281,77	283,70	282,09	278,52	278,19	279,87	280,02
400	388,91	379,12	382,16	380,00	374,10	373,73	375,60	375,64
500	488,52	476,93	480,62	478,05	469,94	469,57	471,54	471,56
600	588,15	574,95	579,31	576,38	565,92	565,42	567,40	567,40
700	687,85	672,99	678,15	674,87	662,14	661,69	663,60	663,60
800	787,75	771,57	777,32	773,81	758,39	757,86	759,92	759,92
900	887,41	869,87	876,14	872,41	854,60	854,03	856,00	856,00
1000	987,31	968,54	974,75	971,55	950,99	950,46	952,40	952,40

Tabela 4.8: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}$ com as strings da base SR-O simplificadas com o algoritmo PSOAR*.

$occ(S)$	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
2	251,03	247,81	247,81	247,81	247,82	247,81	247,81	250,41
3	259,31	248,38	248,50	250,21	249,93	248,23	248,44	257,36
4	265,73	251,73	251,37	254,33	254,15	251,43	251,13	262,28
5	271,37	256,91	255,31	259,55	259,58	256,34	254,87	266,55
6	274,33	261,23	258,90	263,80	263,69	260,52	258,15	268,52
7	276,82	264,95	261,94	267,67	266,93	264,13	261,36	270,21
8	278,81	268,23	264,91	270,85	269,05	266,51	263,89	271,00
9	280,46	270,47	267,16	273,61	270,64	268,41	265,93	271,92
10	281,33	272,10	268,74	275,33	271,41	269,41	267,56	272,15

Tabela 4.9: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}$ com as strings da base SR-L simplificadas com o algoritmo PSOAR*.

$ \Sigma_S $	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
50	294,67	286,20	281,95	288,01	275,19	274,88	275,33	275,45
100	290,24	277,56	272,05	277,77	276,09	275,51	271,12	279,05
150	283,50	267,27	263,11	267,28	268,08	267,37	262,67	275,84
200	277,89	260,00	257,49	260,62	261,13	260,03	257,24	272,30
250	272,90	255,27	253,94	256,18	256,44	255,02	253,71	268,50
300	269,12	252,65	251,88	253,93	253,80	252,26	251,72	265,19
350	266,18	250,97	250,57	252,32	252,22	250,61	250,48	262,95
400	263,46	249,83	249,77	251,31	250,93	249,53	249,61	260,41
450	261,32	249,01	249,10	250,49	250,26	248,82	248,96	258,60
500	259,30	248,50	248,72	249,94	249,56	248,34	248,55	256,81

As tabelas 4.10, 4.11 e 4.12 apresentam as médias das distâncias obtidas para o problema DT nas bases RAND, T-O e T-L, respectivamente. Novamente, em todos os conjuntos, as heurísticas mais sofisticadas obtiveram resultados melhores que os resultados da heurística de mapeamentos aleatórios e que os resultados do algoritmo SOAR*.

Em todos os conjuntos, a heurística BL* apresentou os melhores resultados. As heurísticas AG e GRASP, assim como suas adaptações AG* e GRASP*, obtiveram resultados bem similares, e encontraram distâncias em média maiores que as distâncias encontradas pelas heurísticas BL e BL*.

É possível verificar mais uma vez que utilizar o mapeamento do algoritmo SOAR* diminui as distâncias na maioria dos casos, e que mesmo quando isso não ocorre as distâncias ainda são bem próximas das distâncias obtidas ao não considerarmos esse mapeamento. Também notamos que, nas bases T-O e T-L, as distâncias encontradas foram consideravelmente inferiores às 250 transposições que foram aplicadas para gerar esses conjuntos de dados.

Tabela 4.10: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DT* com as strings da base RAND simplificadas com o algoritmo PSOAR*.

Tamanho	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
100	48,49	47,03	48,29	48,33	48,27	46,93	48,28	52,71
200	102,91	100,47	102,75	102,54	102,49	100,19	102,69	108,58
300	158,48	155,45	158,16	157,92	157,73	154,74	158,18	164,78
400	214,33	211,05	214,01	213,65	212,98	209,80	214,00	221,47
500	270,51	267,04	270,26	269,78	268,36	265,10	270,12	278,24
600	326,80	323,24	326,56	326,17	323,50	320,43	326,37	335,02
700	383,31	379,71	383,10	382,55	378,58	375,74	382,98	391,53
800	440,06	436,27	439,75	439,29	434,42	431,50	439,50	448,59
900	496,82	493,09	496,38	496,00	490,01	487,29	496,27	505,61
1000	553,48	549,67	553,15	552,69	545,63	542,83	552,94	562,41

Tabela 4.11: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DT* com as strings da base T-O simplificadas com o algoritmo PSOAR*.

$occ(S)$	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
2	202,50	199,78	201,87	202,06	201,96	199,76	201,84	214,10
3	202,45	199,34	201,91	201,75	201,69	199,25	201,78	212,95
4	201,84	198,69	201,34	201,20	201,07	198,54	201,31	211,15
5	201,72	198,70	201,30	201,07	200,93	198,19	201,27	210,27
6	201,36	198,54	200,92	200,74	200,46	197,65	200,89	209,09
7	200,84	198,14	200,43	200,16	199,71	196,97	200,38	208,19
8	200,50	197,93	200,10	199,79	199,27	196,53	200,04	207,27
9	200,08	197,48	199,60	199,38	198,64	195,81	199,48	206,63
10	199,20	196,70	198,71	198,47	197,59	194,78	198,60	205,26

Tabela 4.12: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DT* com as strings da base T-L simplificadas com o algoritmo PSOAR*.

$ \Sigma_S $	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
50	191,08	188,07	190,85	190,39	182,85	181,41	189,37	191,41
100	198,60	195,30	198,31	197,89	194,21	191,75	197,92	202,42
150	200,80	197,29	200,46	200,01	198,27	195,23	200,23	206,35
200	201,51	198,08	201,13	200,75	200,09	196,84	201,05	208,58
250	201,74	198,27	201,34	200,96	200,65	197,53	201,20	209,82
300	202,21	198,82	201,72	201,40	201,21	198,29	201,74	210,81
350	202,35	198,91	201,75	201,56	201,44	198,57	201,78	211,69
400	202,37	199,08	201,79	201,56	201,54	198,75	201,85	212,20
450	202,54	199,15	201,92	201,81	201,78	199,06	201,92	212,62
500	202,17	199,02	201,49	201,45	201,46	198,88	201,59	212,29

As tabelas 4.13, 4.14 e 4.15 apresentam as médias das distâncias obtidas para o problema *DRT* nas bases RAND, RT-O e RT-L, respectivamente. Na maioria dos casos, as heurísticas mais sofisticadas obtiveram resultados melhores que os resultados da heurística de mapeamentos aleatórios e que os resultados do algoritmo SOAR*. Entretanto, nos conjuntos da base RAND com strings de tamanhos acima de 600, a heurística GRASP encontrou distâncias em média maiores que as distâncias encontradas pelo algoritmo SOAR*.

A melhor heurística para todas as bases foi a BL*. As heurísticas AG e GRASP, assim como suas adaptações AG* e GRASP*, obtiveram resultados bem similares, embora os resultados das heurísticas baseadas em Algoritmo Genético tenham sido um pouco melhores na maioria dos conjuntos.

Como nos problemas anteriores, adicionar o mapeamento do algoritmo SOAR* no conjunto inicial das heurísticas tende a diminuir as distâncias, ou a pelo menos não aumentá-las muito.

Tabela 4.13: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema *DRT* com as strings da base RAND simplificadas com o algoritmo GREEDY*.

Tamanho	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
100	46,41	45,50	46,22	46,22	46,21	45,48	46,21	49,73
200	95,77	93,92	95,46	95,23	95,15	93,76	95,37	98,66
300	145,41	142,80	145,08	144,54	144,04	142,46	144,90	147,69
400	195,13	191,91	194,78	193,84	192,56	191,19	194,40	196,66
500	244,95	241,24	244,49	243,21	240,94	240,06	243,83	245,82
600	294,74	290,58	294,26	292,67	289,60	288,95	293,31	294,57
700	344,67	340,05	344,09	342,13	338,22	337,80	342,60	343,85
800	394,55	389,56	394,01	391,68	387,08	386,79	391,95	393,04
900	444,47	439,12	443,91	441,24	435,88	435,67	441,44	442,01
1000	494,30	488,66	493,74	490,77	484,92	484,75	490,74	491,02

Tabela 4.14: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema DRT com as strings da base RT-O simplificadas com o algoritmo GREEDY*.

$occ(S)$	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
2	178,06	177,27	177,79	177,86	177,85	177,29	177,78	185,12
3	177,78	176,65	177,50	177,43	177,44	176,63	177,41	184,12
4	177,17	175,95	176,91	176,81	176,76	175,85	176,86	182,73
5	176,71	175,47	176,46	176,33	176,25	175,18	176,39	181,87
6	176,30	175,09	176,01	175,97	175,76	174,60	175,99	181,03
7	175,95	174,76	175,65	175,58	175,32	174,08	175,59	180,18
8	175,21	174,05	174,90	174,81	174,43	173,19	174,82	179,17
9	174,82	173,68	174,57	174,45	173,97	172,62	174,41	178,38
10	174,42	173,33	174,17	174,09	173,46	172,18	173,96	177,98

Tabela 4.15: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema DRT com as strings da base RT-L simplificadas com o algoritmo GREEDY*.

$ \Sigma_S $	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
50	166,33	164,93	166,16	165,92	161,09	160,55	164,91	166,60
100	173,53	172,04	173,32	173,11	170,68	169,61	172,88	175,57
150	175,40	173,89	175,17	174,96	173,94	172,61	174,94	178,73
200	176,34	174,85	176,03	175,86	175,54	174,20	175,94	180,61
250	176,86	175,50	176,60	176,44	176,25	175,03	176,57	181,56
300	177,31	175,90	176,98	176,90	176,81	175,71	176,96	182,58
350	177,17	175,86	176,87	176,80	176,71	175,73	176,87	182,74
400	177,40	176,14	177,12	177,06	177,01	176,02	177,10	183,22
450	177,44	176,24	177,10	177,12	177,09	176,16	177,12	183,51
500	177,66	176,50	177,35	177,32	177,31	176,48	177,31	183,78

As tabelas 4.16, 4.17 e 4.18 apresentam as médias das distâncias obtidas para o problema DRT nas bases SRAND, SRT-O e SRT-L, respectivamente. Na maioria dos casos, as heurísticas mais sofisticadas obtiveram resultados melhores que os resultados da heurística de mapeamentos aleatórios e que os resultados do algoritmo SOAR*. Entretanto, nos conjuntos da base SRAND com strings de tamanhos acima de 600, as heurísticas GRASP e AG encontraram distâncias em média maiores que as distâncias encontradas pelo algoritmo SOAR*. O mesmo ocorreu com o conjunto correspondente ao alfabeto de tamanho 50 da base SRT-L.

Considerando todos os conjuntos, a melhor heurística foi a BL*. Como nos problemas anteriores, utilizar o mapeamento do algoritmo SOAR* no conjunto inicial das heurísticas tende a gerar resultados melhores.

Tabela 4.16: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}T$ com as strings da base SRAND simplificadas com o algoritmo PSOAR*.

Tamanho	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
100	52,68	51,94	52,51	52,65	52,62	51,87	52,52	57,56
200	103,89	103,01	103,74	103,75	103,70	102,52	103,67	108,28
300	154,54	153,48	154,37	154,36	154,18	152,17	154,23	158,16
400	204,97	203,76	204,79	204,77	204,14	201,22	204,43	207,53
500	255,25	253,91	255,13	254,97	253,14	250,10	254,42	256,66
600	305,51	303,92	305,39	305,20	300,88	298,96	304,00	305,76
700	355,69	354,00	355,57	355,37	348,94	347,82	353,70	354,74
800	405,93	404,10	405,82	405,56	397,42	396,70	403,30	403,82
900	456,07	454,04	455,93	455,62	446,09	445,51	452,33	452,82
1000	506,23	504,06	506,06	505,74	495,03	494,42	501,78	501,93

Tabela 4.17: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}T$ com as strings da base SRT-O simplificadas com o algoritmo PSOAR*.

$occ(S)$	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
2	178,17	176,97	177,52	177,95	177,95	176,94	177,53	185,66
3	178,67	177,16	178,04	178,33	178,31	177,06	178,04	185,39
4	178,64	177,15	178,01	178,22	178,23	176,90	178,01	184,34
5	179,01	177,57	178,44	178,64	178,55	177,13	178,41	184,12
6	178,77	177,46	178,28	178,42	178,31	176,64	178,20	183,28
7	178,59	177,43	178,22	178,34	178,15	176,25	178,08	182,62
8	178,54	177,42	178,18	178,25	178,03	176,11	178,00	182,48
9	178,48	177,42	178,11	178,18	177,88	175,68	177,86	181,67
10	178,34	177,28	177,93	178,02	177,61	175,08	177,69	181,04

Tabela 4.18: Médias das distâncias obtidas com as heurísticas propostas e com o algoritmo SOAR* para o problema $D\bar{R}T$ com as strings da base SRT-L simplificadas com o algoritmo PSOAR*.

$ \Sigma_S $	MA	BL	GRASP	AG	AG*	BL*	GRASP*	SOAR*
50	175,86	174,72	175,71	175,69	169,53	167,79	173,30	174,08
100	178,50	177,16	178,32	178,25	176,43	173,38	177,50	179,49
150	178,92	177,45	178,67	178,65	178,16	175,50	178,37	181,98
200	179,17	177,58	178,78	178,75	178,57	176,53	178,61	183,16
250	178,99	177,43	178,57	178,64	178,53	176,81	178,48	183,92
300	179,12	177,54	178,64	178,76	178,69	177,19	178,56	184,56
350	178,94	177,31	178,37	178,57	178,52	177,08	178,34	184,66
400	178,64	177,01	177,99	178,26	178,22	176,90	177,95	184,67
450	178,62	177,01	177,98	178,23	178,21	176,95	177,97	185,06
500	178,58	177,06	177,99	178,25	178,22	176,99	177,94	184,98

Com os resultados obtidos, percebemos que a heurística BL foi a que melhor generalizou para o caso em que permitimos rótulos com mais de duas cópias. As heurísticas GRASP e AG também obtiveram bons resultados, e são alternativas válidas principalmente para o caso do modelo composto apenas pelo evento de reversão. Também notamos que utilizar o mapeamento do SOAR* entre os mapeamentos iniciais tende a melhorar os resultados.

Capítulo 5

Conclusão

Neste trabalho, apresentamos uma nova abordagem baseada em metaheurísticas para os problemas de distância de rearranjo entre genomas (representados por strings) envolvendo os eventos de reversão e transposição. Inicialmente, descrevemos e testamos nossas heurísticas considerando um cenário restrito onde cada gene dos genomas pode ter no máximo duas cópias. Posteriormente, estendemos algumas heurísticas e adaptamos outras ideias da literatura para tratar do caso geral. O algoritmo desenvolvido para o caso geral pode ser decomposto em quatro etapas, representadas na Figura 5.1.

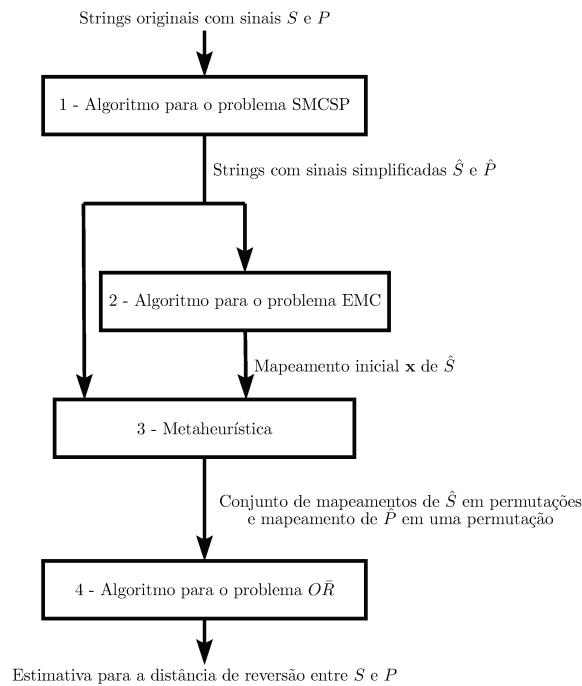


Figura 5.1: Etapas do algoritmo para estimar a distância de reversão em strings com sinais. As etapas são similares para os demais modelos, apenas os problemas de partição (SM CSP) e de distância entre permutações (OR) devem ser substituídos pelas variações correspondentes.

O foco deste trabalho foi no desenvolvimento de metaheurísticas para a etapa 3. Para as demais etapas, utilizamos algoritmos existentes na literatura, realizando testes experimentais para determinar o melhor algoritmo para cada problema. Na etapa 1, adaptamos alguns dos algoritmos para utilizá-los em todas as variações dos problemas de partição, e conseguimos melhorar os resultados de alguns algoritmos com uma nova heurística baseada em combinação de blocos.

Dentre as metaheurísticas desenvolvidas para a etapa 3, verificamos que nos experimentos práticos a heurística Busca Local obteve bons resultados no caso geral. Quando estamos lidando apenas com o evento de reversão ou com um limite de duas cópias para cada gene, as heurísticas GRASP e Algoritmo Genético também se mostraram boas alternativas.

Em trabalhos futuros seria interessante explorar o problema EMC, para melhorar a qualidade do mapeamento inicial. Além disso, as decomposições utilizadas para o problema EMC também podem ser usadas como uma representação alternativa para os mapeamentos de strings em permutações, o que pode resultar em novas metaheurísticas para a etapa 3. Outro caminho para trabalhos futuros é a busca por uma melhor aproximação para os problemas de partição da etapa 1, o que garantiria uma melhor aproximação para os problemas de distância de rearranjo.

Bibliografia

- [1] Renata M. Aiex, S. Binato e Mauricio G. C. Resende. “Parallel GRASP with path-relinking for job shop scheduling”. Em: *Parallel Computing* 29.4 (abr. de 2003), pp. 393–430.
- [2] Thiago da Silva Arruda, Ulisses Dias e Zanoni Dias. “A GRASP-Based Heuristic for the Sorting by Length-Weighted Inversions Problem”. Em: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 15.2 (mar. de 2018), pp. 352–363.
- [3] Andy Auyeung e Ajith Abraham. “Estimating genome reversal distance by genetic algorithm”. Em: *The 2003 Congress on Evolutionary Computation, 2003. CEC '03*. Vol. 2. IEEE, jan. de 2004, 1157–1161 Vol.2.
- [4] David A. Bader, Bernard M.E. Moret e Mi Yan. “A Linear-Time Algorithm for Computing Inversion Distance between Signed Permutations with an Experimental Study”. Em: *Journal of Computational Biology* 8.5 (out. de 2001), pp. 483–491.
- [5] Vineet Bafna e Pavel A. Pevzner. “Genome Rearrangements and Sorting by Reversals”. Em: *SIAM Journal on Computing* 25.2 (abr. de 1996), pp. 272–289.
- [6] Vineet Bafna e Pavel A. Pevzner. “Sorting by Transpositions”. Em: *SIAM Journal on Discrete Mathematics* 11.2 (mai. de 1998), pp. 224–240.
- [7] Anne Bergeron. “A very elementary presentation of the Hannenhalli–Pevzner theory”. Em: *Discrete Applied Mathematics* 146.2 (mar. de 2005), pp. 134–145.
- [8] Piotr Berman, Sridhar Hannenhalli e Marek Karpinski. “1.375-Approximation Algorithm for Sorting by Reversals”. Em: *Proceedings of the 10th Annual European Symposium on Algorithms (ESA '2002)*. London, UK: Springer-Verlag, 2002, pp. 200–210.
- [9] Christian Blum, José A. Lozano e Pinacho Davidson. “Mathematical programming strategies for solving the minimum common string partition problem”. Em: *European Journal of Operational Research* 242.3 (mai. de 2015), pp. 769–777.
- [10] Christian Blum, José A. Lozano e Pedro Pinacho Davidson. “Iterative Probabilistic Tree Search for the Minimum Common String Partition Problem”. Em: *Hybrid Metaheuristics*. Cham: Springer International Publishing, 2014, pp. 145–154.
- [11] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez e José A. Lozano. “Construct, Merge, Solve & Adapt A new general algorithm for combinatorial optimization”. Em: *Computers & Operations Research* 68 (abr. de 2016), pp. 75–88.

- [12] Christian Blum e Günther R. Raidl. “Computational performance evaluation of two integer linear programming models for the minimum common string partition problem”. Em: *Optimization Letters* 10.1 (jul. de 2015), pp. 189–205.
- [13] Klairton Lima Brito, Andre Rodrigues Oliveira, Ulisses Dias e Zanoni Dias. “Heuristics for the Sorting Signed Permutations by Reversals and Transpositions Problem”. Em: *Algorithms for Computational Biology*. Vol. 10849. Heidelberg, Germany: Springer International Publishing, 2018, pp. 65–75.
- [14] Laurent Bulteau, Guillaume Fertin e Christian Komusiewicz. “(Prefix) reversal distance for (signed) strings with few blocks or small alphabets”. Em: *Journal of Discrete Algorithms* 37 (mar. de 2016), pp. 44–55.
- [15] Laurent Bulteau, Guillaume Fertin e Irena Rusu. “Sorting by Transpositions Is Difficult”. Em: *SIAM Journal on Discrete Mathematics* 26.3 (jan. de 2012), pp. 1148–1180.
- [16] Rafael G. Cano, Guilherme Kunigami, Cid C. de Souza e Pedro J. de Rezende. “A hybrid GRASP heuristic to construct effective drawings of proportional symbol maps”. Em: *Computers & Operations Research* 40.5 (mai. de 2013), pp. 1435–1447.
- [17] Alberto Caprara. “Sorting Permutations by Reversals and Eulerian Cycle Decompositions”. Em: *SIAM Journal on Discrete Mathematics* 12.1 (jan. de 1999), pp. 91–110.
- [18] Xin Chen. “On sorting unsigned permutations by double-cut-and-joins”. Em: *Journal of Combinatorial Optimization* 25.3 (dez. de 2010), pp. 339–351.
- [19] Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, Stefano Lonardi e Tao Jiang. “Assignment of Orthologous Genes via Genome Rearrangement”. Em: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2.4 (out. de 2005), pp. 302–315.
- [20] David A. Christie. “A $3/2$ -Approximation Algorithm for Sorting by Reversals”. Em: *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’1998)*. Philadelphia, PA, USA, 1998, pp. 244–252.
- [21] David A. Christie e Robert W. Irving. “Sorting Strings by Reversals and by Transpositions”. Em: *SIAM Journal on Discrete Mathematics* 14.2 (jan. de 2001), pp. 193–206.
- [22] Marek Chrobak, Petr Kolman e Jiří Sgall. “The Greedy Algorithm for the Minimum Common String Partition Problem”. Em: *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX’2004), and 8th International Workshop on Randomization and Computation (RANDOM’2004)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 84–95.
- [23] Graham Cormode e S. Muthukrishnan. “The string edit distance matching problem with moves”. Em: *ACM Transactions on Algorithms* 3.1 (fev. de 2007), pp. 1–19.

- [24] Ulisses Dias, Christian Baudet e Zanoni Dias. “Greedy Randomized Search Procedure to Sort Genomes using Symmetric, Almost-Symmetric and Unitary Inversions”. Em: *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics (BCB'2013)*. Vol. 8542. Lecture Notes in Computer Science. New York, NY, USA: ACM Press, 2007, pp. 181–190.
- [25] Ulisses Dias, Gustavo Rodrigues Galvão, Carla Négri Lintzmayer e Zanoni Dias. “A general heuristic for genome rearrangement problems”. Em: *Journal of Bioinformatics and Computational Biology* 12.03 (jun. de 2014), p. 1450012.
- [26] Isaac Elias e Tzvika Hartman. “A 1.375-Approximation Algorithm for Sorting by Transpositions”. Em: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 3.4 (out. de 2006), pp. 369–379.
- [27] Jianxing Feng e Daming Zhu. “Faster algorithms for sorting by transpositions and sorting by block interchanges”. Em: *ACM Transactions on Algorithms* 3.3 (ago. de 2007), p. 25.
- [28] Thomas A. Feo e Mauricio G.C. Resende. “Greedy Randomized Adaptive Search Procedures”. Em: *Journal of Global Optimization* 6.2 (mar. de 1995), pp. 109–133.
- [29] S.M. Ferdous e Mohammad Sohel Rahman. “Solving the Minimum Common String Partition Problem with the Help of Ants”. Em: *Mathematics in Computer Science* 11.2 (fev. de 2017), pp. 233–249.
- [30] Guillaume Fertin, Anthony Labarre, Irena Rusu, Eric Tannier e Stéphane Vialette. *Combinatorics of Genome Rearrangements*. 1^a ed. Computational Molecular Biology. London, England: The MIT Press, 2009.
- [31] Nan Gao, Ning Yang e Jijun Tang. “Ancestral Genome Inference Using a Genetic Algorithm Approach”. Em: *PLoS ONE* 8.5 (mai. de 2013), e62156.
- [32] Fred Glover. “Tabu Search—Part I”. Em: *ORSA Journal on Computing* 1.3 (ago. de 1989), pp. 190–206.
- [33] Avraham Goldstein, Petr Kolman e Jie Zheng. “Minimum Common String Partition Problem: Hardness and Approximations”. Em: *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'2004)*. Berlin, Heidelberg, 2005, pp. 484–495.
- [34] Isaac Goldstein e Moshe Lewenstein. “Quick greedy computation for minimum common string partition”. Em: *Theoretical Computer Science* 542 (jul. de 2014), pp. 98–107.
- [35] Sridhar Hannenhalli e Pavel A. Pevzner. “Transforming cabbage into turnip. polynomial algorithm for sorting signed permutations by reversals”. Em: *Journal of ACM* 46.1 (jan. de 1999), pp. 1–27.
- [36] Tzvika Hartman e Ron Shamir. “A simpler and faster 1.5-approximation algorithm for sorting by transpositions”. Em: *Information and Computation* 204.2 (fev. de 2006), pp. 275–290.

- [37] Dan He. “A Novel Greedy Algorithm for the Minimum Common String Partition Problem”. Em: *Bioinformatics Research and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 441–452.
- [38] Prasanna Jog, Jung Y. Suh e Dirk Van Gucht. “Parallel Genetic Algorithms Applied to the Traveling Salesman Problem”. Em: *SIAM Journal on Optimization* 1.4 (nov. de 1991), pp. 515–529.
- [39] John Kececioğlu e David Sankoff. “Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement”. Em: *Algorithmica* 13.1-2 (fev. de 1995), pp. 180–210.
- [40] John D. Kececioğlu e R. Ravi. “Of Mice and Men: Algorithms for Evolutionary Distances Between Genomes with Translocation”. Em: *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'1995)*. Philadelphia, PA, USA, 1995, pp. 604–613.
- [41] Scott Kirkpatrick, Daniel Gelatt e Mario P. Vecchi. “Optimization by Simulated Annealing”. Em: *Science* 220.4598 (mai. de 1983), pp. 671–680.
- [42] Petr Kolman e Tomasz Waleń. “Approximating reversal distance for strings with bounded number of duplicates”. Em: *Discrete Applied Mathematics* 155.3 (fev. de 2007), pp. 327–336.
- [43] Petr Kolman e Tomasz Waleń. “Reversal Distance for Strings with Duplicates: Linear Time Approximation Using Hitting Set”. Em: *Proceedings of the 4th International Workshop on Approximation and Online Algorithms (WAOA'2006)*. Vol. 4368. Lecture Notes in Computer Science. Berlin, Heidelberg, 2007, pp. 279–289.
- [44] Derrick Lehmer. “Teaching combinatorial tricks to a computer”. Em: *Proceedings of Symposia in Applied Mathematics*. 1960, pp. 179–193.
- [45] Xiaowen Lou e Daming Zhu. “Genome Rearrangement Algorithms for Unsigned Permutations with $O(\log N)$ Singletons”. Em: *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC'2008)*. Xi'an, China: Springer-Verlag, 2008, pp. 59–69.
- [46] Nadia El-Mabrouk e David Sankoff. “Analysis of Gene Order Evolution Beyond Single-Copy Genes”. Em: *Evolutionary Genomics. Methods in Molecular Biology (Methods and Protocols)* 855 (mai. de 2012), pp. 397–429.
- [47] Melanie Mitchell. *Introduction to Genetic Algorithms*. Cambridge, MA, USA: Springer Berlin Heidelberg, 2008.
- [48] Andre Rodrigues Oliveira, Klairton Lima Brito, Ulisses Dias e Zanoni Dias. “On the Complexity of Sorting by Reversals and Transpositions Problems”. Em: *Journal of Computational Biology* 26.11 (nov. de 2019), pp. 1223–1229.
- [49] Ferdinando Pezzella, Gianluca Morganti e Gianfranco Ciaschetti. “A genetic algorithm for the Flexible Job-shop Scheduling Problem”. Em: *Computers & Operations Research* 35.10 (out. de 2008), pp. 3202–3212.

- [50] Andrew J. Radcliffe, Alex D. Scott e Elizabeth Wilmer. “Reversals and Transpositions Over Finite Alphabets”. Em: *SIAM Journal on Discrete Mathematics* 19.1 (jan. de 2005), pp. 224–244.
- [51] Atif Rahman, Swakkhar Shatabda e Masud Hasan. “An approximation algorithm for sorting by reversals and transpositions”. Em: *Journal of Discrete Algorithms* 6.3 (set. de 2008), pp. 449–457.
- [52] Mingfu Shao, Yu Lin e Bernard M.E. Moret. “An Exact Algorithm to Compute the Double-Cut-and-Join Distance for Genomes with Duplicate Genes”. Em: *Journal of Computational Biology* 22.5 (mai. de 2015), pp. 425–435.
- [53] Dana Shapira e James A. Storer. “Edit distance with move operations”. Em: *Journal of Discrete Algorithms* 5.2 (jun. de 2007), pp. 380–392.
- [54] José Luis Soncco-Álvarez e Mauricio Ayala-Rincón. “Sorting Permutations by Reversals through a Hybrid Genetic Algorithm based on Breakpoint Elimination and Exact Solutions for Signed Permutations”. Em: *Electronic Notes in Theoretical Computer Science* 292 (mar. de 2013). Proceedings of the XXXVIII Latin American Conference in Informatics (CLEI), pp. 119–133.
- [55] Jakkarin Suksawatchon, Chidchanok Lursinsap e Mikael Bodén. “Computing The Reversal Distance Between Genomes In The Presence Of Multi-Gene Families Via Binary Integer Programming”. Em: *Journal of Bioinformatics and Computational Biology* 05.01 (fev. de 2007), pp. 117–133.
- [56] Maria Emília M.T. Walter, Zanoni Dias e João Meidanis. “Reversal and transposition distance of linear chromosomes”. Em: *Proceedings of the String Processing and Information Retrieval: A South American Symposium (SPIRE'1998)*. Los Alamitos, CA, USA, 1998, pp. 96–102.
- [57] Geoffrey A. Watterson, Warren J. Ewens, Thomas E. Hall e Alexander Morgan. “The chromosome inversion problem”. Em: *Journal of Theoretical Biology* 99.1 (nov. de 1982), pp. 1–7.
- [58] Eyla Willing, Simone Zaccaria, Marília D.V. Braga e Jens Stoye. “On the inversion-indel distance”. Em: *BMC Bioinformatics* 14.Suppl 15 (out. de 2013), S3.
- [59] Sophia Yancopoulos, Oliver Attie e Richard Friedberg. “Efficient sorting of genomic permutations by translocation, inversion and block interchange”. Em: *Bioinformatics* 21.16 (jun. de 2005), pp. 3340–3346.
- [60] Xin-She Yang e Suash Deb. “Cuckoo Search via Lévy flights”. Em: *Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC'2009)*. IEEE, 2009, pp. 210–214.
- [61] Jianzhi Zhang. “Evolution by gene duplication: An update”. Em: *Trends in Ecology & Evolution* 18.6 (jun. de 2003), pp. 292–298.
- [62] Mo Zhongxi e Zeng Tao. “An improved genetic algorithm for problem of genome rearrangement”. Em: *Wuhan University Journal of Natural Sciences* 11.3 (mai. de 2006), pp. 498–502.

Apêndice A

Resultados Complementares para o Caso de Genes Duplicados

Esse anexo apresenta gráficos para a visualização dos testes realizados na escolha dos parâmetros de cada heurística.

A.1 Problema DR

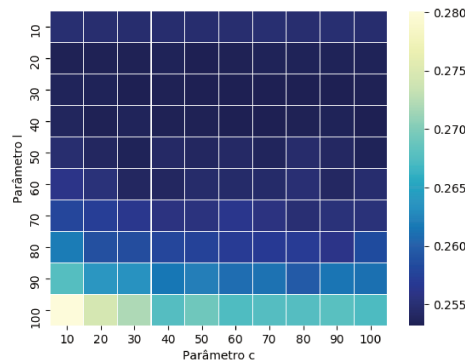


Figura A.1: Ajuste dos parâmetros c e l da heurística de Busca Local no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

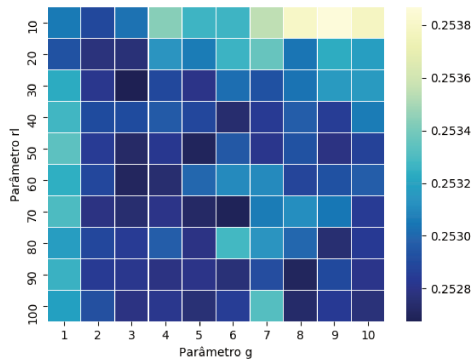
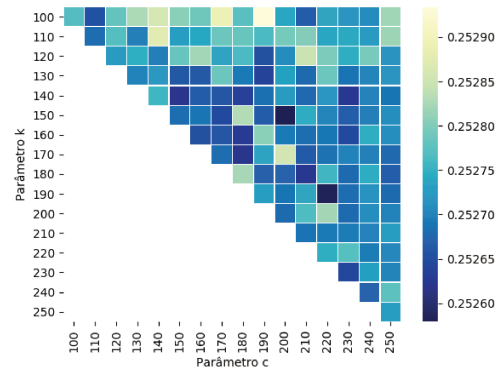
(a) Ajuste dos parâmetros g e r_ℓ .(b) Ajuste dos parâmetros c e k .

Figura A.2: Ajuste dos parâmetros da heurística GRASP no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

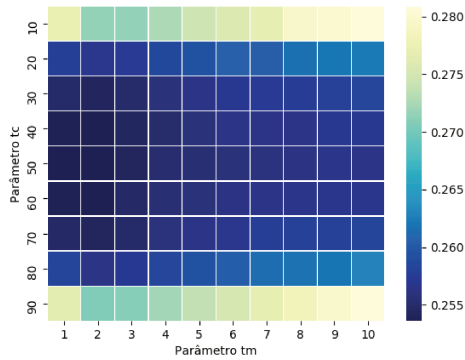
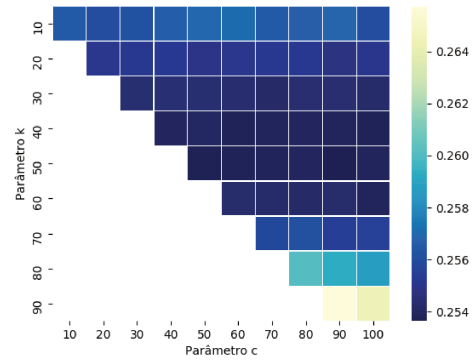
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura A.3: Ajuste dos parâmetros da heurística Algoritmo Genético no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

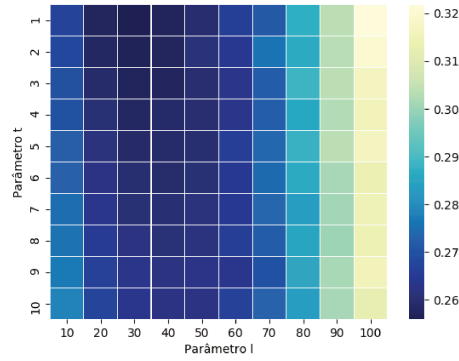
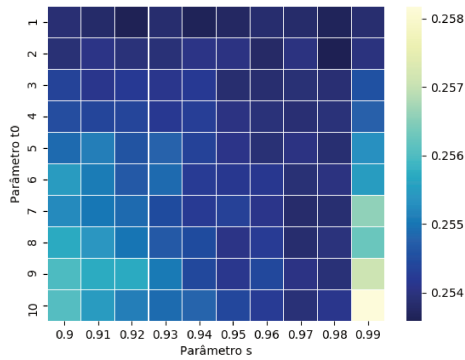
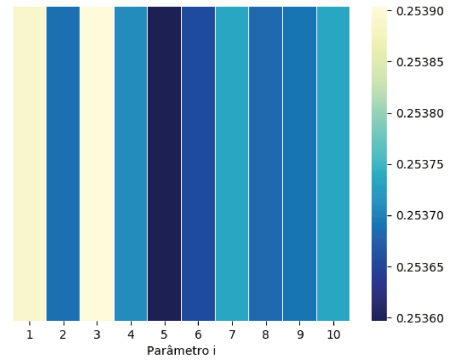


Figura A.4: Ajuste dos parâmetros l e t da heurística de Busca Tabu no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).



(a) Ajuste dos parâmetros s e t_0 .



(b) Ajuste do parâmetro i .

Figura A.5: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

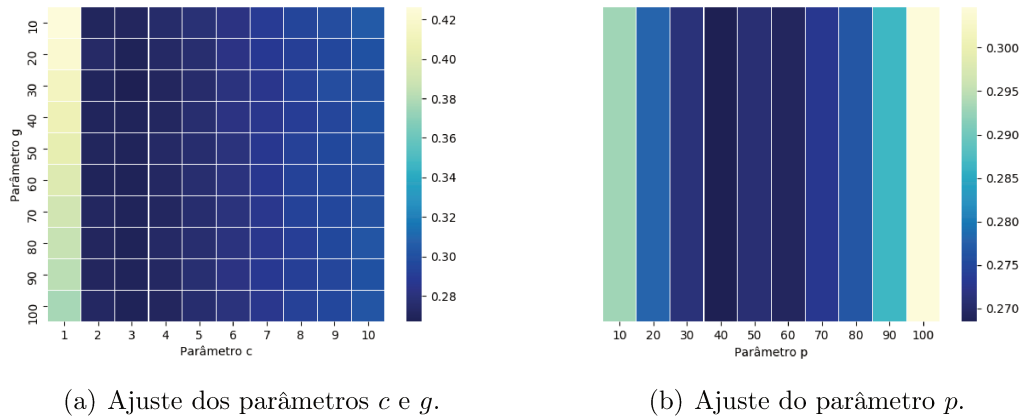


Figura A.6: Ajuste dos parâmetros da heurística de Busca Cuco no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

A.2 Problema $D\bar{R}$

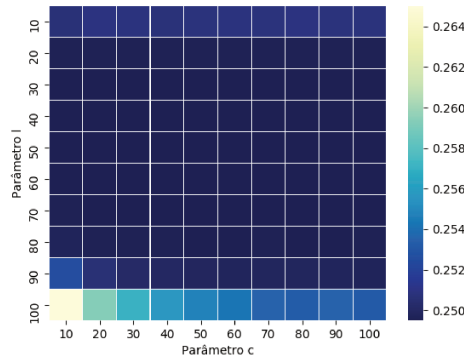


Figura A.7: Ajuste dos parâmetros c e ℓ da heurística de Busca Local no problema $D\bar{R}$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

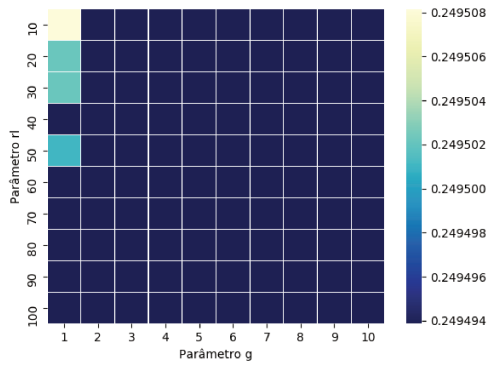
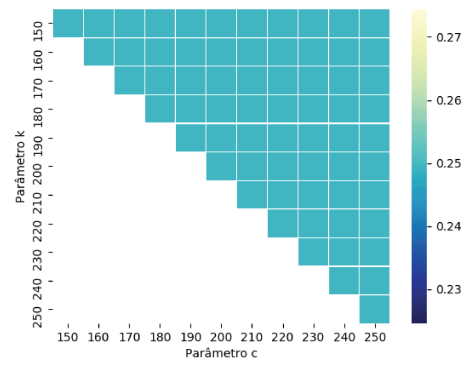
(a) Ajuste dos parâmetros g e r_ℓ .(b) Ajuste dos parâmetros c e k .

Figura A.8: Ajuste dos parâmetros da heurística GRASP no problema $D\bar{R}$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

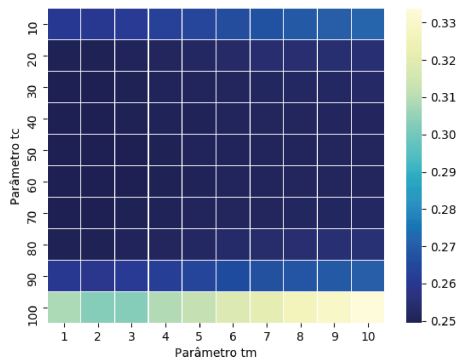
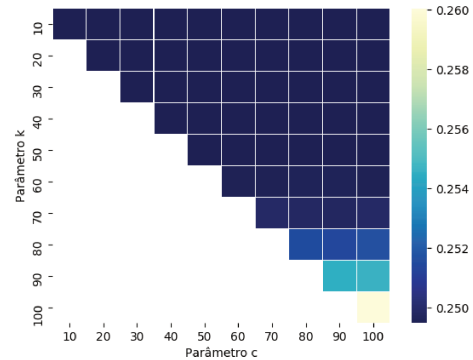
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura A.9: Ajuste dos parâmetros da heurística Algoritmo Genético no problema $D\bar{R}$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

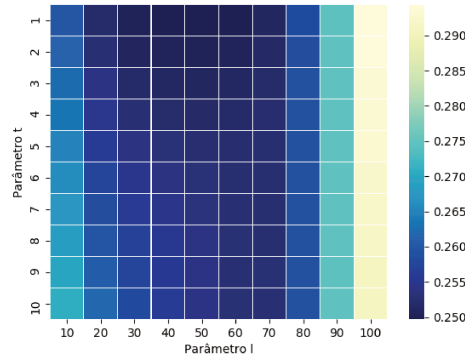
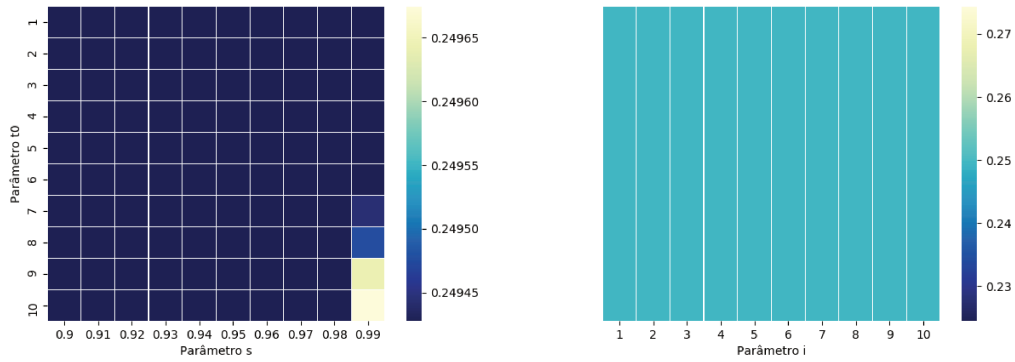


Figura A.10: Ajuste dos parâmetros ℓ e t da heurística de Busca Tabu no problema $D\bar{R}$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).



(a) Ajuste dos parâmetros s e t_0 .

(b) Ajuste do parâmetro i .

Figura A.11: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema $D\bar{R}$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

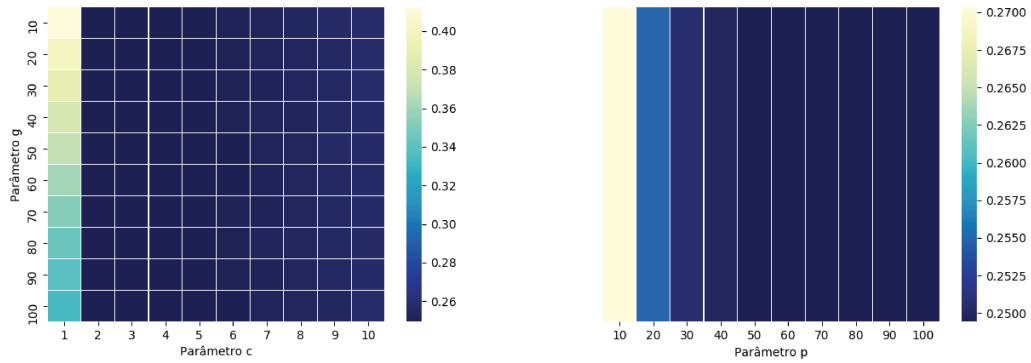
(a) Ajuste dos parâmetros c e g .(b) Ajuste do parâmetro p .

Figura A.12: Ajuste dos parâmetros da heurística de Busca Cuco no problema DR . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

A.3 Problema DT

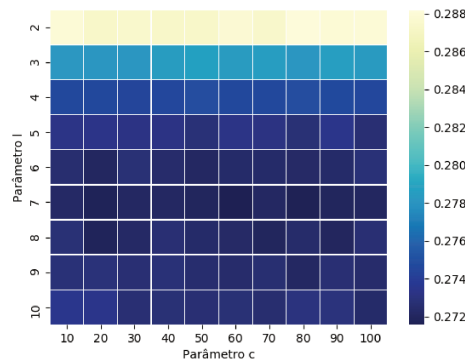


Figura A.13: Ajuste dos parâmetros c e l da heurística de Busca Local no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

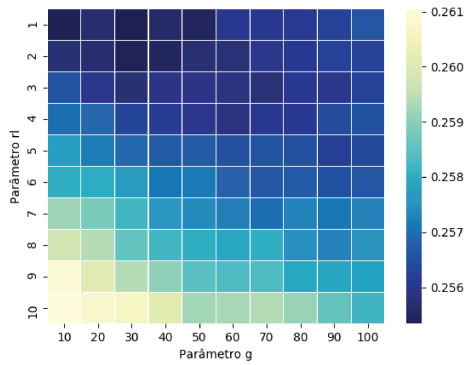
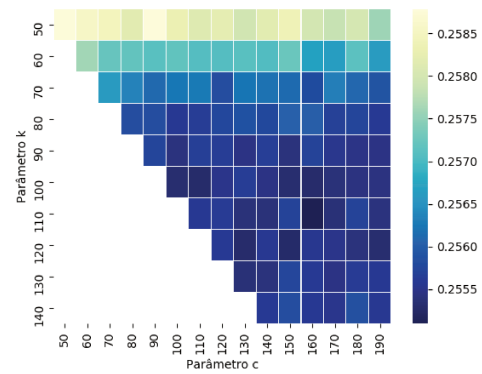
(a) Ajuste dos parâmetros g e r_ℓ .(b) Ajuste dos parâmetros c e k .

Figura A.14: Ajuste dos parâmetros da heurística GRASP no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

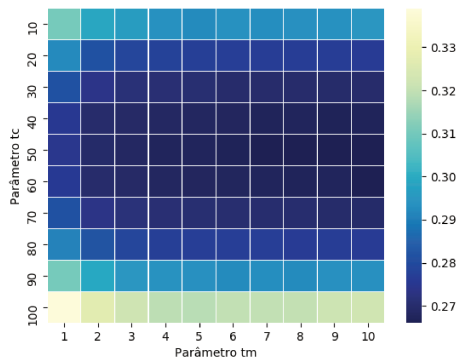
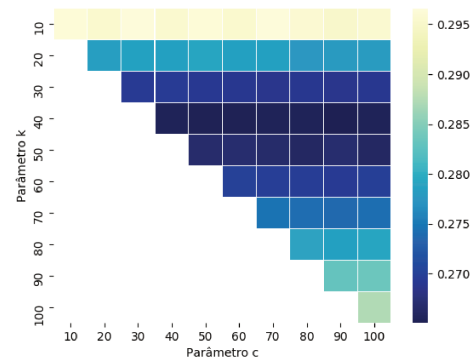
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura A.15: Ajuste dos parâmetros da heurística Algoritmo Genético no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

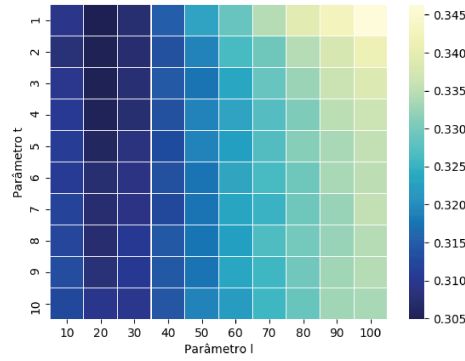
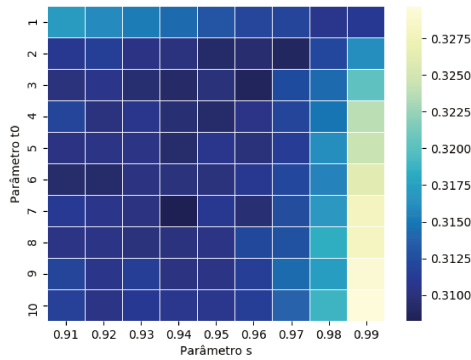
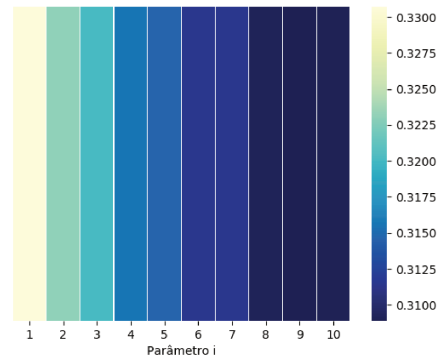


Figura A.16: Ajuste dos parâmetros l e t da heurística de Busca Tabu no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).



(a) Ajuste dos parâmetros s e t_0 .



(b) Ajuste do parâmetro i .

Figura A.17: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

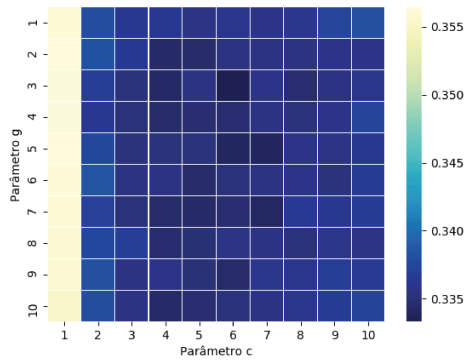
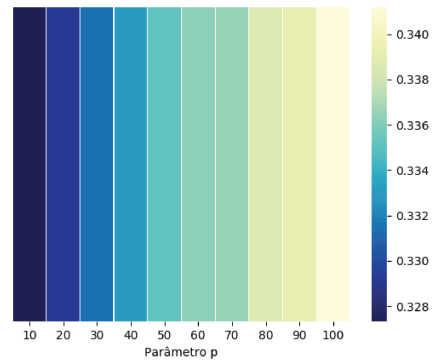
(a) Ajuste dos parâmetros c e g .(b) Ajuste do parâmetro p .

Figura A.18: Ajuste dos parâmetros da heurística de Busca Cuco no problema DT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

A.4 Problema DRT

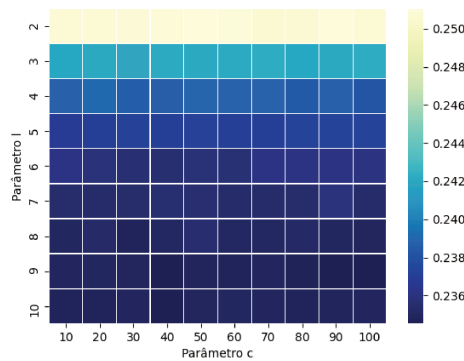


Figura A.19: Ajuste dos parâmetros c e ℓ da heurística de Busca Local no problema DRT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

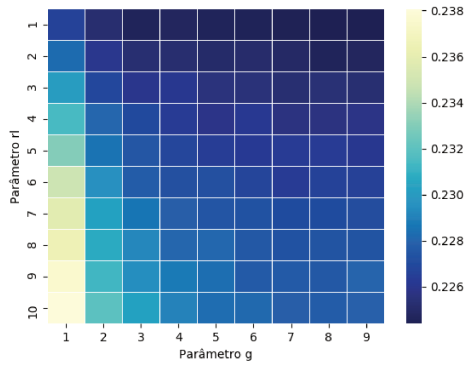
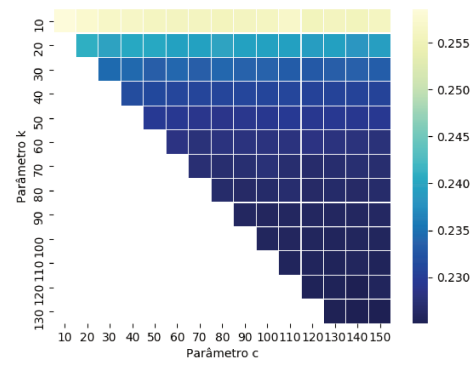
(a) Ajuste dos parâmetros g e r_ℓ .(b) Ajuste dos parâmetros c e k .

Figura A.20: Ajuste dos parâmetros da heurística GRASP no problema *DRT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

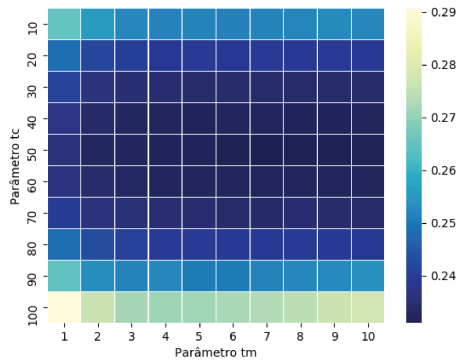
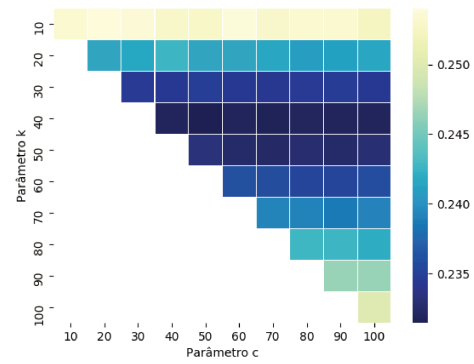
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura A.21: Ajuste dos parâmetros da heurística Algoritmo Genético no problema *DRT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

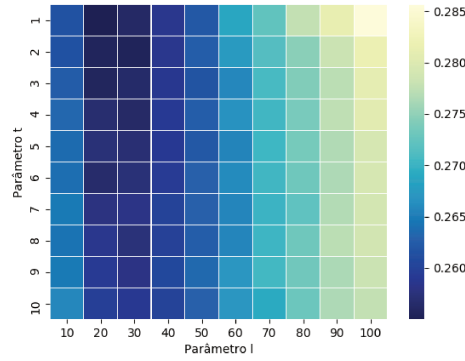
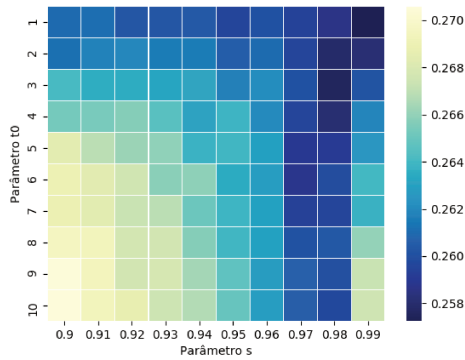
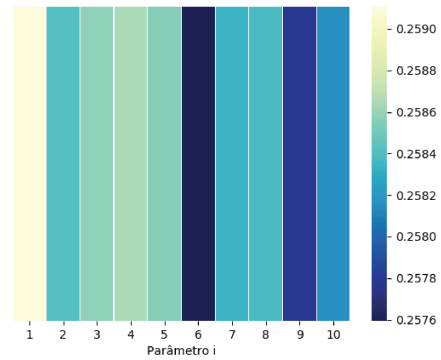


Figura A.22: Ajuste dos parâmetros ℓ e t da heurística de Busca Tabu no problema *DRT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).



(a) Ajuste dos parâmetros s e t_0 .



(b) Ajuste do parâmetro i .

Figura A.23: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema *DRT*. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

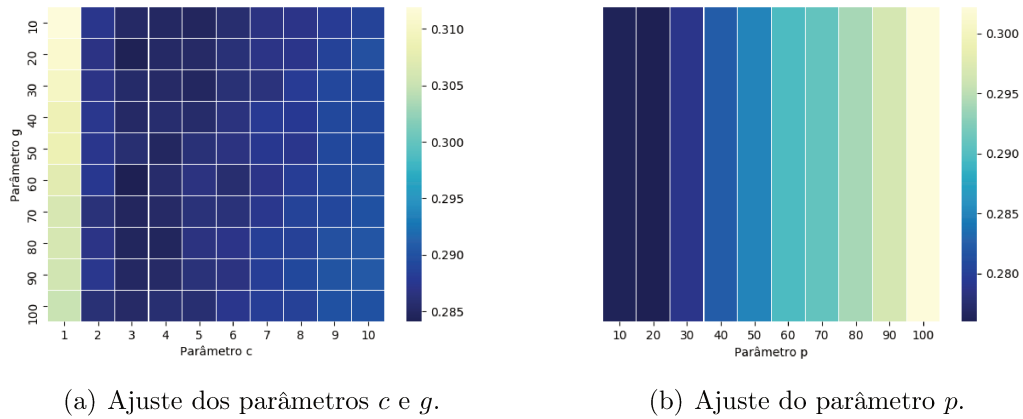


Figura A.24: Ajuste dos parâmetros da heurística de Busca Cuco no problema DRT . Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

A.5 Problema $D\bar{R}T$

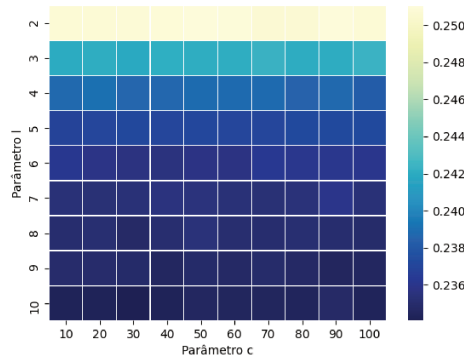


Figura A.25: Ajuste dos parâmetros c e ℓ da heurística de Busca Local no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).

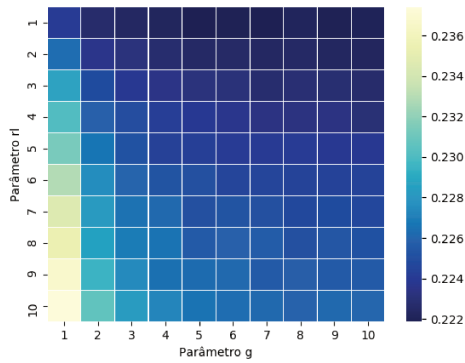
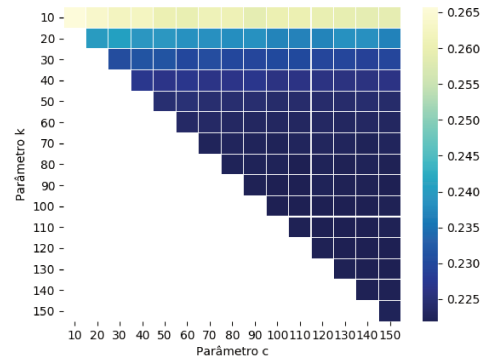
(a) Ajuste dos parâmetros g e r_ℓ .(b) Ajuste dos parâmetros c e k .

Figura A.26: Ajuste dos parâmetros da heurística GRASP no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

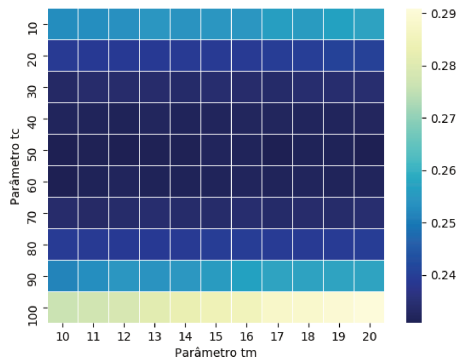
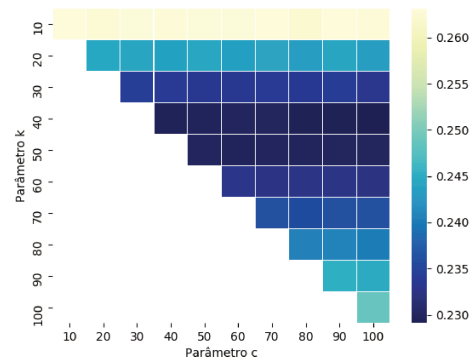
(a) Ajuste dos parâmetros t_m e t_c .(b) Ajuste dos parâmetros c e k .

Figura A.27: Ajuste dos parâmetros da heurística Algoritmo Genético no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

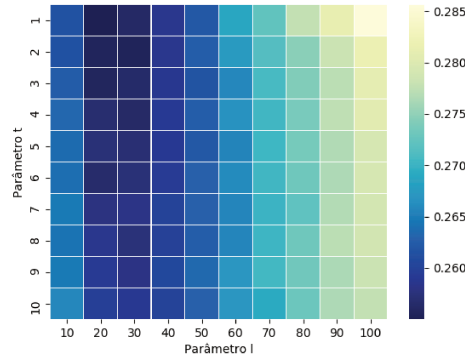
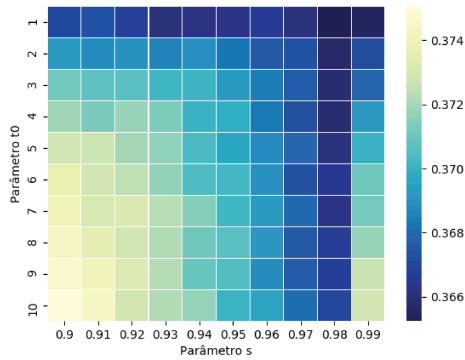
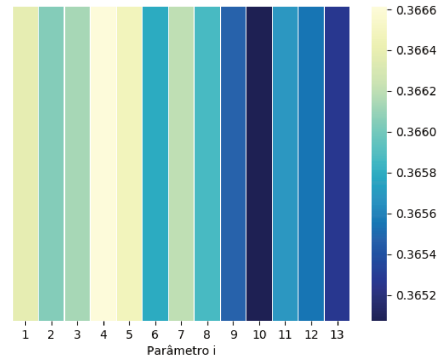


Figura A.28: Ajuste dos parâmetros l e t da heurística de Busca Tabu no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral).



(a) Ajuste dos parâmetros s e t_0 .



(b) Ajuste do parâmetro i .

Figura A.29: Ajuste dos parâmetros da heurística de *Simulated Annealing* no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).

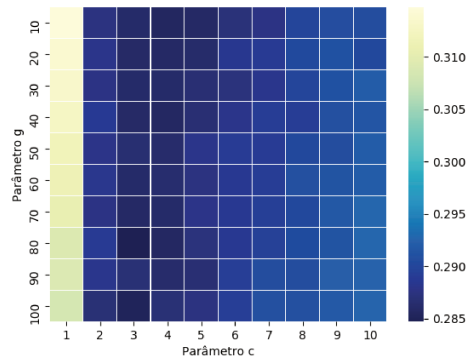
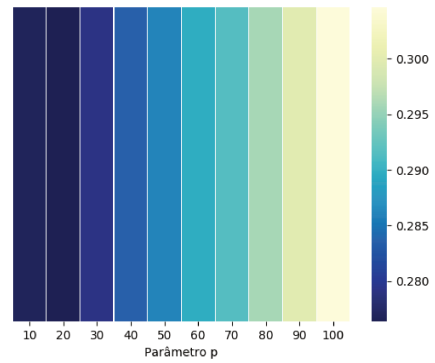
(a) Ajuste dos parâmetros c e g .(b) Ajuste do parâmetro p .

Figura A.30: Ajuste dos parâmetros da heurística de Busca Cuco no problema $D\bar{R}T$. Os valores correspondem à média das distâncias divididas pelo tamanho das strings (os valores correspondentes às cores são indicados na barra lateral de cada figura).