CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS **Diretoria de Pesquisa e Pós-Graduação**
Programa de Pós-graduação em Modelagem Matemática e Computacional

Algoritmos heurísticos para a solução do Problema da Cadeia de Caracteres Mais Próxima

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional, como parte dos requisitos exigidos para a obtenção do título de Mestre em Modelagem Matemática e Computacional.

Aluno: Augusto Flávio Morais Santos
Orientador: Prof. Dr. Marcone Jamilson Freitas Souza
Coorientador: Prof. Dr. Sérgio Ricardo de Souza

Belo Horizonte - MG
Agosto de 2018

Santos, Augusto Flávio Morais
S237a Algoritmos heurísticos para a solução do problema da cadeia de caracteres mais próxima / Augusto Flávio Morais Santos. – 2018.
71 f.

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Modelagem Matemática e Computacional.
Orientador: Marcone Jamilson Freitas Souza.
Coorientador: Sérgio Ricardo de Souza.
Dissertação (mestrado) – Centro Federal de Educação Tecnológica de Minas Gerais.

1. Algorítmos genéticos – Teses. 2. Programação heurística – Teses. 3. Conjunto de caracteres (Processamento de dados) – Teses. I. Souza, Marcone Jamilson Freitas. II. Souza, Sérgio Ricardo de. III. Centro Federal de Educação Tecnológica de Minas Gerais. IV. Título.

CDD 519.6

Elaboração da ficha catalográfica pela Biblioteca-Campus II / CEFET-MG

Agradecimentos

Agradeço, primeiramente, aos professores Dr. Marcone e Dr. Sérgio pelos conselhos, paciência, orientações e, sobretudo, por terem me concedido a oportunidade de concretizar este sonho. À CAPES pelo apoio financeiro, sem o qual não seria possível realizar esta pesquisa. À todos os funcionários e professores do CEFET-MG pela atenção e dedicação.

Agradeço eternamente à minha família, pelo amor, carinho e compreensão.

Sou grato aos meus amigos que me apoiaram nesta jornada e à minha companheira, Flávia, por compreender minha ausência nos momentos que foram dedicados a este trabalho.

"Avalia-se a inteligência de um indivíduo pela quantidade de incertezas que ele é capaz de suportar." (Immanuel Kant)

Resumo

O presente trabalho tem seu foco no Problema da Cadeia de Caracteres Mais Próxima (PCCP) – *Closest String Problem* (CSP). O objetivo deste problema é o de encontrar uma sequência de caracteres que mais se aproxime, segundo uma dada métrica, de um conjunto de cadeias de caracteres de mesma dimensão. Sendo o CSP da classe NP-difícil, justifica-se a utilização de métodos heurísticos para solucioná-lo. Propõe-se a implementação de três metaheurísticas, a saber: *Simulated Annealing* (SA), *Variable Neighborhood Search* (VNS) e *Discrete Particle Swarm Optimization* (DPSO). Todos os métodos foram implementados com características híbridas, combinados com um método de busca local, que é aplicado cada vez que é atualizada a melhor solução corrente. Para testá-los, foram utilizados conjuntos de problemas-teste amplamente usados na literatura. Os experimentos computacionais realizados indicaram que o método híbrido *Hybrid Discrete Particle Swarm Optimization for Closest String Problem* ($\text{HDPSO}_{\text{CSP}}$) obteve bom desempenho e, em alguns casos, chegou a encontrar as soluções ótimas.

Palavras-chave: Problema da Cadeia de Caracteres Mais Próxima. Simulated Annealing. Variable Neighborhood Search. Particle Swarm Optimization. Busca Local.

Abstract

This work addresses the *Closest String Problem* (CSP). The objective is to seek a string whose distance from a given set of strings is minimal and with the same dimension. The CSP is NP-Hard problem, in consequence, it is indicate to use heuristics methods to solve the problem. We proposed three metaheuristic-based algorithms: one based on the *Simulated Annealing* (SA), *Variable Neighborhood Search* (VNS) and, the last one based on *Discrete Particle Swarm Optimization* (DPSO). All methods were implemented with hybrid characteristics, combined with a local search method, that is applied every time that a better solution is updated. The instances used in this dissertation are known by literature and have big dimensions. The computational experiments performed showed that the hybrid method DPSO had good performance and, in some cases, found the global optimum.

Keywords: Closest String Problem. Simulated Annealing. Variable Neighborhood Search. Particle Swarm Optimization. Local Search.

Lista de Abreviaturas e Siglas

A *Adenina*

BI *Best Improvement*

CMSPWFBC *Close to Most String Problem with Few Bad Columns*

CMSP *Close to Most String Problem*

CSP *Closest String Problem*

CSSP *Closest Substring Problem*

C *Citosina*

DNA *Ácido Desoxirribonucleico*

DPSO *Discrete Particle Swarm Optimization*

DSSP *Distinguishing String Selection Problem*

EBI *Europe Bioinformatics Institute*

FFMSP *Far From Most String Problem*

FPT *Fixed-Parameter Tractable*

FSP *Farthest String Problem*

G *Guanina*

HDPSO$_{\mathbf{CSP}}$ *Hybrid Discrete Particle Swarm Optimization for Closest String Problem*

HSA$_{\mathbf{CSP}}$ *Hybrid Simulated Annealing for Closest String Problem*

HVNS$_{\mathbf{CSP}}$ *Hybrid Variable Neighborhood Search for Closest String Problem*

PCCP Problema da Cadeia de Caracteres Mais Próxima

PI *Programação Inteira*

PSO *Particle Swarm Optimization*

PTAS *Polynomial-Time Approximation Scheme*

RNA *Ácido Ribonucleico*

SA *Simulated Annealing*

$\mathbf{SA_{CSP}}$ *Simulated Annealing for Closest String Problem*

T *Timina*

U *Uracila*

VNS *Variable Neighborhood Search*

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Com a descoberta da estrutura do *Ácido Desoxirribonucleico* (DNA) por Watson e Crick (1953), a Biologia Molecular tem sofrido grandes avanços, permitindo, por exemplo, a manipulação de sequências biomoleculares e o acesso a uma grande quantidade de dados. Assim, o processamento dessas informações não somente ganhou pertinência, como abriu portas para uma nova gama de problemas ainda inexplorados à época e que puderam ser tratados sob a perspectiva interdisciplinar da Matemática aliada à Ciência da Computação (SETUBAL; MEDAINIS, 1997).

Segundo Festa (2007), houve uma intensificação nas pesquisas científicas voltadas à Biologia Molecular a partir do início do projeto Genoma Humano em 1986. Há uma clara divisão quanto as competências e aplicações no campo da Biologia Molecular e suas disciplinas afins antes e depois dos anos 80. Assim, as pesquisas começaram a ter interesses compartilhados com outras áreas do conhecimento, como a Ciência da Computação e a Física. A maior contribuição da área computacional veio da implementação de métodos eficientes para tratar uma grande quantidade de dados coletados pelas pesquisas. Já a contribuição dos físicos foi o de aprofundar o estudo molecular e suas estruturas tridimensionais.

Nos últimos anos, um grande número de problemas da Biologia Molecular têm sido formulados como problemas de Otimização Combinatória. Uma destas contribuições veio na abstração da estrutura tridimensional real do DNA e sua representação como uma sequência de caracteres unidimensional a partir de um alfabeto com quatro símbolos. Esta mesma proposição envolve a representação da proteína como sequência de caracteres a partir de um alfabeto de vinte símbolos. Como resultado dessa codificação linear do DNA e de proteínas, podemos citar, por exemplo: a reconstrução de sequências de DNA a partir da sobreposição de fragmentos; a comparação de duas ou mais sequências para

assim buscar suas similaridades; a busca por padrões que ocorrem em certa frequência no DNA e/ou sequências proteicas. Já no campo da otimização e programação matemática, em particular, podemos citar: problemas de sequências de alinhamento, problemas de reorganização do genoma; problemas de seleção e comparação de cadeias de caracteres; e reconhecimento e predição de estruturas proteicas (FESTA, 2007).

Este trabalho discute o Problema da Cadeia de Caracteres Mais Próxima (PCCP) – *Closest String Problem* (CSP), que tem como objetivo encontrar a maior similaridade entre duas ou mais cadeias de caracteres. Trata-se, portanto, de um problema de minimização, já que é preciso minimizar a máxima distância de uma sequência de caracteres frente a um conjunto estabelecido. Tal conjunto é formado por sequências de caracteres de dimensão finita e formado de acordo com um alfabeto dado. Este último – alfabeto, pode ainda ser representado pelas bases nitrogenadas do DNA, *Ácido Ribonucleico* (RNA), cadeias proteicas e ou alfabetos binários.

1.1 Objetivos

1.1.1 Objetivo Geral

Este trabalho tem como objetivo principal desenvolver e apresentar a efetividade da aplicação de métodos heurísticos híbridos, envolvendo metaheurísticas tanto de busca local quanto de busca populacional, para resolver o Problema da Cadeia de Caracteres Mais Próxima.

1.1.2 Objetivos Específicos

Com a finalidade de alcançar o objetivo geral, foram traçados os seguintes objetivos específicos:

- Realizar uma pesquisa dos trabalhos da literatura que tratam o problema abordado e problemas relacionados, assim como os métodos utilizados para resolvê-los;
- Desenvolver três algoritmos heurísticos híbridos para resolver o problema;
- Testar os algoritmos desenvolvidos em conjuntos de problemas-teste disponíveis na literatura;
- Realizar testes estatísticos, de modo a analisar os resultados encontrados.

1.2 Motivação

Uma das conquistas mais significativas dos últimos 20 anos foram os primeiros sequenciamentos do genoma humano. Isto possibilitou aprofundar o conhecimento não só da genética humana, como também das milhões de outras espécies no planeta. Contudo, decifrar a codificação dos genes para as proteínas ou até mesmo para o DNA tornou-se um grande desafio moderno, uma vez que tratar e analisar essa grande massa de dados só é possível por meio das abordagens de modelos matemáticos, computacionais e estatísticos (SCHATZ, 2015).

Para se ter uma ideia, o laboratório de Biologia Molecular do Instituto de Bioinformática Europeia – *Europe Bioinformatics Institute* (EBI) sedia um dos maiores repositórios de dados biológicos do mundo e, atualmente, possui 20 *petabytes* (1 *petabyte* são 10^{15} bytes) de dados, incluindo genes, proteínas e pequenas moléculas. Somente os dados do genoma são responsáveis por 2 *petabytes* e este número mais que dobra a cada ano (MARX, 2013).

O genoma humano, por exemplo, possui 140 *gigabytes* em dados e, às vezes, uma simples tarefa como compartilhar esses dados se torna complexa, já que muitos dos serviços *online* de compartilhamento de arquivos possuem limites de armazenagem: *Dropbox* e *Google Drive* (MARX, 2013).

Assim, o armazenamento e a interpretação dessa quantidade de dados requer, muitas vezes, processadores computacionais de alta performance, algoritmos avançados de busca e indexação, otimização numérica, técnicas de modelagem e diversas outras disciplinas fora da Biologia Genética.

O PCCP é um desses problemas que surgiram a partir do sequenciamento do genoma e que tem sido largamente estudado nos últimos anos, principalmente no campo da Biologia Molecular (MENESES et al., 2004; CHIMANI; WOSTE; BOCKER, 2011; FESTA, 2007; LIU et al., 2008; GOMES et al., 2008; LANCTOT et al., 1999).

Um exemplo prático, segundo Lanctot et al. (1999), é o de identificar e usar as informações genéticas de um conjunto de espécies fortemente relacionadas a favor de outro conjunto de espécies. Em outras palavras, uma vez identificado os genes responsáveis pela codificação das proteínas essenciais de um conjunto de bactérias patogênicas, poderia ser possível sintetizar uma nova droga (vacina) que atuaria a favor dos seres humanos.

O vírus da Dengue, por exemplo, apresenta taxas crescentes de mortalidade em países tropicais e subtropicais. Consequentemente, a identificação de sequências de *consensus* para cada sorotipo do vírus torna-se importante e possibilita o desenvolvimento de uma potencial vacina através da identificação dos agentes antivirais (AYUB et al., 2013).

Por fim, parte também como motivação deste trabalho, desenvolver métodos heurísticos eficientes para solução do PCCP.

1.3 Organização do Trabalho

O restante deste trabalho está estruturado como segue. No Capítulo 2 é apresentado detalhadamente o PCCP e os problemas relacionados. No Capítulo 3 é realizada uma revisão bibliográfica. No Capítulo 4 são apresentadas as heurísticas e metaheurísticas exploradas neste trabalho. No Capítulo 5 é apresentada a metodologia utilizada para resolver o problema, descrevendo os algoritmos desenvolvidos, suas características e estruturas. Os resultados computacionais obtidos com os métodos desenvolvidos são apresentados e discutidos no Capítulo 6. A conclusão e trabalhos futuros são encontrados no Capítulo 7.

Capítulo 2

Caracterização do Problema

Neste capítulo é discutido o Problema da Cadeia de Caracteres Mais Próxima (PCCP) – *Closest String Problem* (CSP) e suas características. Em seguida, são apresentados os problemas correlatos.

O capítulo está organizado da seguinte forma. Na Seção 2.1 é apresentado o PCCP. Na Seção 2.2 são apresentados os problemas relacionados: *Closest Substring Problem* (CSSP) na Subseção 2.2.1; *Farthest String Problem* (FSP) na Subseção 2.2.2; *Far From Most String Problem* (FFMSP) na Subseção 2.2.3; *Close to Most String Problem* (CMSP) na Subseção 2.2.4; e *Distinguishing String Selection Problem* (DSSP) na Subseção 2.2.5.

2.1 Problema da Cadeia de Caracteres Mais Próxima

Problemas de comparação e seleção de sequências de caracteres pertencem, de modo geral, à classe geral da Biologia Molecular e são conhecidos como sequências de consenso (*sequences consensus*). Neste problema, um grupo finito de sequências é dado e uma outra sequência é investigada de modo a encontrar uma similaridade. Em outras palavras, é preciso determinar uma sequência de caracteres, chamada de *consensus*, de modo que esta sequência represente de alguma forma todas as outras sequências fornecidas (FESTA, 2007).

O PCCP tem uma importante aplicação não somente no campo da Biologia Computacional, como também na Teoria dos Códigos. O desenvolvimento de novos medicamentos, testes para diagnósticos de doenças e identificação de padrões em sequências de proteínas são alguns exemplos práticos da Biocomputação. Já no campo da Teoria dos Códigos, existe o Teorema da Codificação de Ruídos, no qual o objetivo é reproduzir uma mensagem, de uma fonte de origem, utilizando apenas as informações de saída. Ao longo da

transmissão dessa mensagem, existe uma interferência, que faz com que esta mensagem seja codificada. Ao final da transmissão, é preciso decodificar a mensagem, de modo que a palavra recebida seja "mais próxima" da palavra enviada (ROMAN, 1992; LANCTOT et al., 1999).

Normalmente, as cadeias de caracteres usadas no PCCP são representadas pelas bases nitrogenadas do DNA: *Adenina* (A), *Timina* (T), *Guanina* (G) e *Citosina* (C). Para o RNA, usa-se a *Uracila* (U) no lugar da *Timina*. Há, ainda, a possibilidade de se utilizar outros alfabetos, como o binário, sequências proteicas ou números inteiros positivos.

Várias técnicas foram propostas para encontrar as regiões comuns de um conjunto de sequências de caracteres. A medida fundamental é a distância de *Hamming*. Segundo Li, Ma e Wang (2002b), existem duas funções objetivo mais utilizadas na literatura: na primeira é realizada a soma total das distâncias entre a sequência central (*consensus*) e cada uma das sequências fornecidas e; na segunda, é calculada a distância máxima entre a sequência central e a sequência fornecida. Vale ainda ressaltar que a primeira função objetivo para a distância de *Hamming*, bem como outras métricas, foram tratadas por Li, Ma e Wang (2002a).

O foco deste trabalho não é discutir as métricas das funções objetivo, uma vez que a segunda proposição de função objetivo para a distância de *Hamming* tem sido largamente utilizada nos trabalhos da literatura a partir de 2002. Entre eles, podemos citar Festa (2007), Meneses et al. (2004), Liu et al. (2008), Gomes et al. (2008), Lanctot et al. (1999), Chimani, Woste e Bocker (2011), Li, Ma e Wang (2002b). É importante reforçar portanto, que este trabalho utiliza da segunda métrica como função objetivo.

Segundo Festa (2007), o modelo da distância de *Hamming* é representado da seguinte forma:

- Seja um alfabeto $\mathscr{A} = \{c_1, c_2, \ldots, c_z\}$ formado por um grupo de elementos finito, normalmente definido com as bases nitrogenadas do DNA: $\mathscr{A} = \{A, C, G, T\}$;
- Seja $\mathscr{S} = \{s^1, s^2, \ldots, s^n\}$ um conjunto de sequências de caracteres de dimensão $n(|\mathscr{S}| = n)$, em que $\mathscr{S} = (s^i \in \mathscr{S}, i = 1, 2, \ldots, n)$;
- Seja $s = (s_1, s_2, \ldots, s_m)$ uma sequência de caracteres de dimensão $m(|s| = m)$, em que $s = (s_k \in \mathscr{A}, k = 1, 2, \ldots, m)$;

Assim, a distância de *Hamming* entre duas cadeias de caracteres s e t pode ser definida como:

$$d_H(s,t) = \sum_{i=1}^{|s|} \phi(s_i, t_i) \tag{1}$$

em que $\phi : \mathscr{A} \times \mathscr{A} \rightarrow \{0, 1\}$ tal que:

$$\phi(a, b) = \begin{cases} 0, & \text{se } a = b, \\ 1, & \text{caso contrário.} \end{cases} \tag{2}$$

A Figura 1 ilustra, de modo didático, duas cadeias de caracteres s^1 e s^2 e a respectiva distância de *Hamming*.

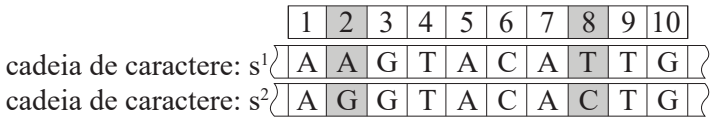


Figura 1 – Distância de *Hamming*

Pela Figura 1, percebe-se que as sequências s^1 e s^2 diferem entre si nas colunas 2 ($s^1_2 = A$ e $s^2_2 = G$) e 8 ($s^1_8 = T$ e $s^2_8 = C$). O valor da distância de *Hamming*, então, é calculado pela somatória dos caracteres díspares, o que, no caso, chega-se a dois.

Visto isso, o PCCP pode ser definido como: dado um conjunto finito de sequências $\mathscr{S} = \{s^1, s^2, \ldots, s^n\}$, em que $\mathscr{A}(s^i \in \mathscr{A}^m, i = 1, 2, \ldots, n)$, o problema tem como objetivo encontrar uma sequência central (*consensus*) $t \in \mathscr{A}^m$ tal que a distância de *Hamming* entre t e todas as outras sequências em $\mathscr{S}$ seja mínima. Desta forma, t é a sequência em que o menor valor d pode ser calculado a partir da seguinte expressão (FESTA, 2007):

$$d_H(t, s^i) \leq d, \quad \forall\ s^i \in \mathscr{A} \tag{3}$$

Assim, de forma sucinta: dado um conjunto $\mathscr{S}$ de sequências de caracteres com n elementos e formado pelo alfabeto $\mathscr{A}$, o objetivo é encontrar uma sequência t de dimensão m, minimizando d, tal que, para cada sequência s^i, $d_H(t, s^i) \leq d$ (LANCTOT et al., 1999).

A Figura 2 ilustra um exemplo do PCCP e calcula a distância de *Hamming* entre a sequência central e cada uma das cadeias de caracteres. Desta forma, dado o conjunto de sequências $\mathscr{S} = \{s^1, s^2, s^3\}$ de dimensão $n = 3$, em que s^i é a cadeia de caractere i, de dimensão $m = 10$ e o alfabeto $\mathscr{A} = \{A, C, G, T\}$ de dimensão 4, temos que a distância entre a sequência central t e s^1 é dada por: $d_{H,1}(t, s^1) = 7$; entre t e s^2: $d_{H,2}(t, s^2) = 8$ e entre t e s^3: $d_{H,3}(t, s^3) = 7$. Neste caso, a menor distância de *Hamming* é dada por 7 e a maior distância por 8.

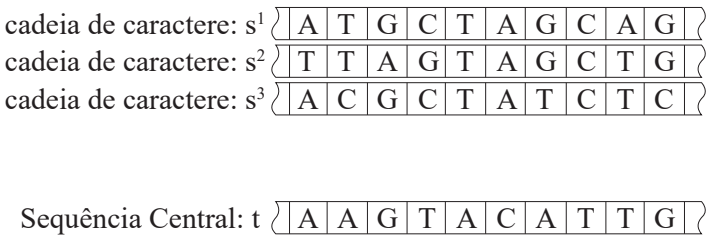


Figura 2 – Exemplo do PCCP com 3 sequências de caracteres

A próxima seção apresenta uma representação matemática para o PCCP.

2.1.1 Formulação Matemática

O objetivo desta seção é apresentar uma formulação matemática para a resolução do PCCP. Meneses et al. (2004), Kelsey e Lars (2011), Soleimani-damaneh (2011), Chimani, Woste e Bocker (2011) são alguns dos autores que propuseram métodos exatos para resolver esse problema. Contudo, não é proposta deste trabalho discutir os detalhes dessas formulações, uma vez que o objetivo deste trabalho é abordar o PCCP pela perspectiva dos métodos heurísticos.

Uma das formulações matemáticas mais conhecidas e citadas na literatura é o de Meneses et al. (2004). O modelo de *Programação Inteira* (PI) deste autor possibilitou resolver o PCCP em sua otimalidade com o uso de problemas-teste de tamanho moderado. O modelo é representado pelas equações (4) e (5a)–(5g):

Seja z_k^i a variável de decisão do problema:

$$z_k^i = \begin{cases} 1, & \text{se } t_k \neq x_k^i \\ 0, & \text{caso contrário.} \end{cases} \tag{4}$$

Onde k é o caractere da sequencia central t e x_k^i é o caractere k da i-ésima sequencia de caractere x^i. Então, o Problema da Cadeia de Caracteres Mais Próxima pode ser formulado pelas seguintes equações:

$$\min \quad d \tag{5a}$$

sujeito a:

$$\sum_{k=1}^{m} z_k^i \leq d \qquad\qquad i = 1, \ldots, n \tag{5b}$$

$$t_k - x_k^i \leq k z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{5c}$$

$$x_k^i - t_k \leq k z_k^i \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{5d}$$

$$z_k^i \in \{0, 1\} \qquad i = 1, \ldots, n;\ k = 1, \ldots, m \tag{5e}$$

$$d \in \mathbb{Z}_+ \tag{5f}$$

$$t_k \in \mathbb{Z}_+ \qquad k = 1, \ldots, m \tag{5g}$$

em que $k = |\mathscr{A}|$ é a dimensão do alfabeto usado pelo conjunto de problemas-teste $\mathscr{S}$.

A função objetivo d, dada pela Expressão (5a), busca minimizar a máxima diferença entre t e as sequências de caracteres em $\mathscr{S}$.

As restrições (5b) determinam o limite inferior para d, o qual deve ser maior ou igual ao número total de diferenças para cada sequência de caractere $s^i \in \mathscr{S}$. As restrições (5c) e (5d) estabelecem os limites inferior e superior para a diferença $t_k - x_k^i$. As restrições (5e), (5f) e (5g) definem os domínios das variáveis do problema.

2.2 Problemas Relacionados a Cadeias de Caracteres

Nas próximas seções são discutidos problemas correlatos ao PCCP: *Closest Substring Problem* (CSSP), *Farthest String Problem* (FSP), *Far From Most String Problem* (FFMSP) e o *Distinguishing String Selection Problem* (DSSP).

2.2.1 Closest Substring Problem

Segundo Meneses (2005), o CSSP compara sequências que não têm uma dimensão definida e tem como finalidade encontrar regiões similares de um dado conjunto. Esse problema parte de uma generalização do PCCP e, na área da Biogenética, procura encontrar regiões no material genético de diferentes espécies.

O problema pode ser formalizado da seguinte forma: determinar uma subsequência de caracteres s, minimizando d, tal que cada sequência s^i de um conjunto de problemas-teste (sequências de caracteres) $\mathscr{S}$ possua uma subsequência $r \subseteq s^i$ de dimensão $|r| = |s|$ e $d_H(r, s) \leq d$.

O CSSP é um problema pertencente à classe NP-Difícil, segundo Frances e Litman (1997). Li, Ma e Wang (2002a) propuseram um algoritmo de Aproximação em Tempo Polinomial – *Polynomial-Time Approximation Scheme* (PTAS) com desempenho $O((k^2n)^{O(\log|\mathscr{A}|/\epsilon^4)})$. Ma e Sun (2008) desenvolveram um algoritmo Tratável por Parâmetro-Fixo – *Fixed-Parameter Tractable* (FPT) com desempenho $O(k\ell+kd\cdot 2^{4d}|\mathscr{A}|^d\cdot n^{\lfloor\log_2 d\rfloor+1})$. Marx (2005) desenvolveu 2 algoritmos FPT com desempenhos $O(|\mathscr{A}|^{d(\log d+2)})$ e $O((|\mathscr{A}|d)^{O(kd)}\cdot n^{O(\log\log k)})$ respectivamente. Meneses et al. (2006) aprimoraram um algoritmo de programação inteira desenvolvido por Zaslavsky e Singh (2006). Já Chimani, Woste e Bocker (2011) propuseram um algoritmo de programação inteira para resolver o CSSP, porém o desempenho só foi satisfatório para problemas-teste de pequena dimensão.

De acordo com Chimani, Woste e Bocker (2011), o CSSP admite a seguinte formulação matemática:

$$\min \quad d \tag{6a}$$

sujeito a:

$$\sum_{\sigma\in\mathscr{A}} x_{\sigma,j} = 1 \qquad \forall \quad 1\le j\le \ell \tag{6b}$$

$$\sum_{i=1}^{n-\ell+1} p_{s,i} = 1 \qquad \forall \quad s\in S \tag{6c}$$

$$\sum_{j=i}^{i+\ell-1} x_{s_j,j-i+1} + d \ge p_{s,i}\cdot\ell \qquad \forall \quad s\in S;\quad 1\le i\le n-\ell+1 \tag{6d}$$

$$x_{\sigma,j},\quad p_{s,i}\in\{0,1\} \qquad \forall \quad 1\le j\le\ell,\quad 1\le i\le n-\ell+1,\quad s\in S,\quad \sigma\in\mathscr{A} \tag{6e}$$

Seja $\mathscr{S}$ um grupo de k cadeias de caracteres formadas pelo alfabeto $\mathscr{A}$, em que cada cadeia de caracteres possui dimensão entre ℓ e n. O objetivo do CSSP é, então, encontrar um número inteiro mínimo d (que corresponde a uma sequência de caracteres t de dimensão ℓ), de modo que, para cada cadeia de caracteres $s\in\mathscr{S}$, exista uma subcadeia de caracteres de dimensão ℓ com distância de *Hamming* d mais próxima de t, sendo t, por sua vez, a subcadeia de caracteres mais próxima de $\mathscr{S}$.

A definição matemática acima define uma variável binária $x_{\sigma,j}$, para cada caractere $\sigma\in\mathscr{A}$ e para cada posição $1\le j\le n$ da solução corrente t. Ou seja, $x_{\sigma,i}$ deve ser igual a 1 se, e somente se, $t_i=\sigma$. Logo após, para cada cadeia de caracteres $s\in\mathscr{S}$,

inicia-se a procura por uma subcadeia de caracteres s_i^ℓ, iniciando em i, $1 \leq n - \ell + 1$, tal que $d_H(s_i^\ell, t) \leq d$. Assim, é atribuído o valor 1 à variável binária $p_{s,i}$, após escolhida a subcadeia de caracteres s_i^ℓ.

As restrições (6b) garantem que cada posição de t seja ocupada por somente um caractere. Já as restrições (6c) asseguram que somente uma subcadeia de caracteres seja encontrada para cada cadeia de caractere. Por fim, as restrições (6d) garantem que seja aplicada uma distância limite inferior às subcadeias de caracteres e essa condição só é possível atribuindo o valor 0 (zero) à variável $p_{s,i}$.

2.2.2 Farthest String Problem

O *Farthest String Problem* (FSP) é um problema semelhante ao PCCP, porém, tem por objetivo encontrar a sequência de caracteres mais distante de um dado conjunto $\mathscr{S}$. Este problema, como grande parte dos problemas de *consensus*, tem aplicação na indústria farmacêutica para desenvolvimento de novos medicamentos.

Lanctot et al. (1999) provaram que o FSP pertence à classe NP-difícil e a transformação usada para esta prova veio por meio do problema 3-SAT.

Festa (2007) caracteriza o FSP como: dado um conjunto de cadeias de caracteres $\mathscr{S} = \{s^1, s^2, \ldots, s^n\}$ com $|s^i| = m$ para $i = \{1, 2, \ldots, n\}$, formado pelo alfabeto $\mathscr{A}$. O objetivo é encontrar a sequência de caracteres s, tal que a soma de todas as distâncias do conjunto de caracteres $\mathscr{S}$ seja máxima.

O problema pode então ser resolvido da seguinte forma: tome V_k como um conjunto de caracteres que aparecem na posição k do conjunto de cadeias de caracteres $\mathscr{S}$. Assim, é preciso definir as seguintes variáveis:

$$v_{j,k} = \begin{cases} 1, & \text{se o j-ésimo caractere em } V_k \text{ é usado na posição } k \\ 0, & \text{caso contrário.} \end{cases} \tag{7}$$

A formulação de programação inteira é então dada por:

$$\max \quad d \tag{8a}$$

sujeito a:

$$\sum_{j \in V_k} v_{j,k} = 1 \qquad k = 1, \ldots, m \tag{8b}$$

$$m - \sum_{j=1}^{m} v_{s_j^i, j} \geq d \qquad i = 1, \ldots, n; \tag{8c}$$

$$d \in \mathbb{N}^+ \qquad\qquad v_{j,k} \in \{0,1\}, \quad k = 1, \ldots, m, \quad \forall \quad j \in V_k \tag{8d}$$

Já a Expressão (8b) garante que somente um caractere em cada conjunto V_k, para $k \in \{1 \ldots m\}$ seja selecionado. A Expressão (8c), diz que se um caractere da cadeia de caracteres s^i não estiver na solução v, então este caractere vai contribuir para diminuir a distância de *Hamming* de v para s^i. Por fim, as restrições (8d) definem os domínios das variáveis.

2.2.3 Far From Most String Problem

O *Far From Most String Problem* (FFMSP) é muito semelhante ao FSP e, como tal, tem por objetivo encontrar uma sequência de caracteres que seja distante do maior número de sequências possíveis de um dado conjunto $\mathscr{S}$. Já o FSP tem por objetivo encontrar uma sequência que seja mais distante de todas as sequências do conjunto.

Dessa forma, o FFMSP pode ser representado da seguinte maneira.

Dado um limitante t, uma sequência s deve ser encontrada de modo a maximizar a variável ω, tal que:

$$d_H(s, s^i) \geq t, \quad \text{para} \quad s^i \in P \subseteq \mathscr{S} \quad \text{e} \quad |P| = \omega \tag{9}$$

É importante ressaltar que P é um subconjunto de $\mathscr{S}$ de cardinalidade ω.

2.2.4 Close to Most String Problem

O *Close to Most String Problem* (CMSP) é também um dos muitos problemas da área da Biocomputação. Seu objetivo também é encontrar uma sequência de caracteres que represente uma aproximação de uma coleção de sequências dadas.

O problema pode ser caracterizado da seguinte forma: dado um conjunto de sequências de mesma dimensão n e um parâmetro d, é preciso encontrar uma sequência x que maximize o número de "*non-outliers*" para a distância de *hamming* d de x (BOUCHER et al., 2012).

Assim, dado um conjunto $\mathscr{S} = \{s^1, \cdots, s^n\}$ de dimensão m e formado pelo alfabeto $\mathscr{A}$, em que $d \in \mathbb{Z}^+$, o objetivo neste problema é maximizar o número de sequências $s^i \in \mathscr{S}$ que satisfaçam à condição $d_H = \{s, s^i\} \leq d$. A solução deste problema é, então, encontrar uma sequência s de dimensão m.

Boucher et al. (2012) provaram que não existe um algoritmo de aproximação em tempo polinomial para o CMSP e seu trabalho também tratou de uma variação deste mesmo problema chamado *Close to Most String Problem with Few Bad Columns* (CMSPWFBC). Por fim, os autores provaram a impossibilidade de existir um algoritmo PTAS para resolver o CMSPWFBC.

2.2.5 Distinguishing String Selection Problem

Deng, Li e Wang (2002) definem o *Distinguishing String Selection Problem* (DSSP) da seguinte forma: dados dois conjuntos de sequências de caracteres S_c e S_f de dimensões m, formados a partir de um alfabeto $\mathscr{A}$ e definido um raio superior K_c e inferior K_f, tal que $K_c \leq K_f$, em que $K_c, K_f \in \mathbb{Z}^+$. O objetivo é encontrar uma sequência $x \in \mathscr{A}$, tal que para cada sequência $c_i \in S_c$, $d_H(x, c_i) \leq K_c$ e para cada sequência $f_i \in S_f$, $d_H(x, f_i) \geq K_f$.

Assim como os problemas descritos anteriormente, o DSSP também pertence à classe NP-difícil e esta prova está detalhada no trabalho de Lanctot et al. (1999). Esses autores propuseram, ainda, um algoritmo de aproximação de complexidade $O(max_{s \in S_b}|s|n^{d_b}(|A|-1)^{d_b})$.

Capítulo 3

Trabalhos Relacionados

O recente desenvolvimento de aplicações da biocomputação tem feito uso de métodos matemáticos, estatísticos e, principalmente, de métodos de otimização combinatória para solucionar diversos problemas de sequenciamento de caracteres, dentre eles o Problema da Cadeia de Caracteres Mais Próxima.

Roman (1992) foi o primeiro a estudar o PCCP, porém, esse estudo ocorreu no campo da Teoria dos Códigos. Somente no final dos anos noventa é que o problema foi provado como intratável computacionalmente, ou seja, pertence à classe NP-difícil (FRANCES; LITMAN, 1997; LANCTOT et al., 1999).

Ben-Dor et al. (1997) fazem uso da definição de sequência de consenso para identificar alinhamentos integrais ou parciais com o uso de um algoritmo de arredondamento randômico baseado em uma formulação de programação inteira.

Já Gasieniec, Jansson e Lingas (1999) trabalharam com o *Hamming Center problem* – que é análogo ao PCCP – e propuseram vários algoritmos de aproximação em tempo polinomial, com desempenho $(\frac{4}{3} + \epsilon)$. Lanctot et al. (1999) trabalharam com o PCCP e o *Farthest String Problem* – FSP – e provou que os dois problemas pertencem à classe NP-Difícil. Mais ainda, foi desenvolvido um algoritmo de aproximação em tempo polinomial para resolver o FSP e que se baseava em técnicas de relaxação em programação linear. Quanto ao PCCP, foi proposto um outro algoritmo de aproximação em tempo polinomial (*Polinomial Time Approximation Schemes - PTAS*), de desempenho semelhante ao de Gasieniec, Jansson e Lingas (1999), ou seja: $(\frac{4}{3} + \epsilon)$ para valores pequenos, sendo $\epsilon > 0$.

Gramm, Niedermeier e Rossmanith (2001) demonstraram que é possível resolver o problema em tempo polinomial usando técnicas de complexidade parametrizada (DOWNEY; FELLOWS, 2012) para um valor fixo da distância de *Hamming*.

Li, Ma e Wang (2002b) trabalharam também com a proposta FTAS; contudo, o foco

está no estudo do PCCP e do *Closest Substring Problem*. Neste algoritmo houve um avanço e foi possível garantir um desempenho de $1 + \epsilon$ para quaisquer valores pequenos de ϵ.

Um modelo de programação não-linear foi utilizado por Ecker et al. (2002) para encontrar similaridades em subsequências de caracteres de DNA. Contudo, é importante destacar que foram utilizados problemas-teste de pequena dimensão.

Deng, Li e Wang (2002) estudaram três problemas relacionados com as sequências de consenso. O objetivo principal do trabalho foi encontrar um padrão, com alguma taxa de erro, que ocorreria em uma das sequências de um determinado conjunto de problemas-teste no PCCP, porém não ocorreria nos problemas *Farthest String Problem* e/ou *Distinguishing String Problem*. Os autores também utilizaram um modelo PTAS para resolverem os problemas.

Em 2003, surgiu uma das primeiras propostas de algoritmos evolutivos para solucionar o PCCP. Mauch, Melzer e Hu (2003) utilizaram um algoritmo genético que apresentou vantagens para encontrar boas soluções em um tempo computacional reduzido.

Meneses et al. (2004) apresentaram um estudo geral do PCCP, de modo a solucionar os problemas-teste em sua otimalidade. O trabalho concentrou-se na formulação de três algoritmos de Programação Inteira. Foi, ainda, utilizado uma relaxação linear para o algoritmo *branch-and-bound* em uma das três formulações propostas. De modo a agilizar este último algoritmo, desenvolveu-se uma heurística para encontrar os limites superiores para os ótimos valores do PCCP. Os testes computacionais mostraram que foi possível solucionar o problema em sua otimalidade. Foram utilizados problemas-teste de tamanho moderado: $n = \{10, 15, 20, 25, 30\}$ (número de sequências de caracteres) e $m = \{300, 400, 500, 600, 700, 800\}$ (dimensão de cada sequência de caracteres), $\mathscr{A} = \{2, 4, 20\}$ (dimensão do alfabeto utilizado).

Meneses (2005) propôs um algoritmo paralelo *Multistart* para resolver problemas-teste complexos. Em geral, os resultados computacionais não foram bons e tiveram, em média, um *gap* de $2,3\%$, $4,9\%$ e $2,9\%$ dos ótimos valores dos problemas-teste utilizados. O tempo de CPU máximo utilizado no algoritmo exato foi de uma hora e o foco do estudo estava na qualidade das soluções e não em seu tempo de processamento. A dimensão dos problemas-teste gerados foram de: $n = \{10, 20, 30\}$ (número de sequências de caracteres) e $m = \{1000, 2000, 3000, 4000, 5000\}$ (dimensão de cada sequência de caracteres), $\mathscr{A} = \{2, 4, 20\}$ (dimensão do alfabeto utilizado).

Festa (2007) tratou um problema semelhante ao PCCP, conhecido como *Far From Most String Problem*, e fez uso do método *Greedy Randomized Adaptive Search Procedure* – GRASP para resolvê-lo. Os testes realizados mostraram que o algoritmo desenvolvido é

competitivo com a heurística de Meneses, Oliveira e Pardalos (2005).

Faro e Pappalardo (2010) propuseram um algoritmo baseado na metaheurística Colônia de Formigas, comparando-o com o trabalho de Liu et al. (2008), o qual, por sua vez, se baseou nos métodos *Simulated Annealing* e Algoritmo Genético. O Algoritmo Genético fez uso de um operador *crossover* adaptativo de dois pontos. Os testes computacionais foram feitos em problemas-teste com as seguintes características: $n = \{10, 20, 30, 40, 50\}$ (número de sequências de caracteres), $m = \{10, 20, \ldots, 50, 100, 200, \ldots, 1000\}$ (dimensão de cada sequência de caracteres) e $\mathscr{A} = \{4\}$ (dimensão do alfabeto utilizado). O algoritmo de Colônia de Formigas superou quase a totalidade dos melhores resultados e se mostrou bastante rápido computacionalmente.

Gomes et al. (2008) desenvolveram um algoritmo *multistart* paralelo, com problemas-teste de dimensões moderados e grandes, para resolver o PCCP. Os resultados computacionais demonstraram soluções com valores próximos a $2,0\%$, $2,3\%$ e $4,9\%$ dos valores ótimos e com as respectivas dimensões dos alfabetos: $\mathscr{A} = \{2, 4, 20\}$.

Já Tanaka (2012) aplicou uma técnica de relaxação lagrangeana e de programação inteira para resolver o PCCP. Foi possível decompor o problema em subproblemas triviais, o que permitiu encontrar a região de factibilidade facilmente. O método de Busca Tabu foi utilizado como combinação do multiplicador lagrangiano. Os resultados encontrados foram próximos dos valores ótimos em um tempo computacional reduzido.

Chimani, Woste e Bocker (2011) trabalharam com o PCCP e o CSSP, para isto, utilizaram Programação Linear Inteira. Os experimentos computacionais revelaram que os métodos desenvolvidos foram aplicados a problemas-teste de tamanho real e obtiveram melhores resultados quando comparados à proposta de Meneses et al. (2004). Por outro lado, vale destacar que a formulação proposta pelos autores foi inspirada nas três formulações de Meneses et al. (2004). As características dos problemas-teste testados foram as seguintes: $n = \{10, 20, 30, 40, 50\}$ (número de sequências de caracteres), $m = \{250, 500, 750, 1000, 2000, 5000, 10000\}$ (dimensão de cada sequência de caracteres) e $\mathscr{A} = \{2, 4, 20\}$ (dimensão do alfabeto utilizado).

Mousavi e Esfahani (2012) fizeram uso da metaheurística GRAP para resolver o PCCP e foi proposta uma função heurística de avaliação inspirada na teoria das probabilidades. Esta função permite avaliar os benefícios de duas ou mais soluções diferentes com a mesma distância de *Hamming* e com custo computacional menor. Os resultados obtidos pelo métodos foram comparados com as propostas de Liu et al. (2008), Faro e Pappalardo (2010). Para as cadeias de caracteres de dimensão $m \leq 50$, o algoritmo obteve soluções com valores iguais. Para os demais casos, o algoritmo teve um desempenho melhor e com um tempo de processamento otimizado.

A Tabela 1, a seguir, resume os trabalhos relacionados. Nesta Tabela, a primeira coluna mostra o ano do trabalho, a segunda os autores, na terceira o problema tratado e na quarta e última coluna, os métodos usados para resolver o problema.

Tabela 1 – Resumos dos trabalhos

Ano	Autor	Problema	Metodologia
1997	Ben-Dor	CSP e DSSP	Algoritmo de aproximação
1999	Gasieniec et al.	Hamming Center Problem	Algoritmo de aproximação de tempo polinomial
2002	Li et at.	CSP e CSSP	Algoritmo de aproximação de tempo polinomial
2002	Deng et al.	CSP, FSP e DSP	Esquemas de Aproximação em tempo polinomial
2002	Ecker et at.	CSSP	Programação não linear
1999/2003	Lanctot et al.	CSP, FSP, DSSP	Técnicas de relaxamento de Programação Linear
2003	Mauch	CSP	Algoritmo Genético
2004	Menezes et al.	CSP	Otimização Inteira/ branch-and-bound
2005	Liu et al.	CSP	Algoritmo Genético e Simulated Annealing (ambos paralelos)
2006	Gramm	DSSS	Algoritmo exato de parâmetro-fixo
2007	Festa	CSP, FSP e FFMSP	Otimização combinatória e GRASP
2008	Gomes et al.	CSP	Multistart (paralelo)
2010	Faro and Pappalardo	CSP	Colônia de Formigas
2011	Chimani	CSP, CSSP	Programação linear inteira
2012	Tanaka	CSP	Relaxação Lagrangeana, Algoritmo Heurístico e Busca Tabu
2012	Mousavi e Esfahani	CSP	GRASP

Capítulo 4

Fundamentação Teórica

Este capítulo apresenta o referencial teórico sobre as técnicas adotadas nesta dissertação. Todavia, cabe ressaltar que os métodos aqui apresentados são descritos em sua forma genérica para solucionarem qualquer problema de otimização. Assim, o objetivo é conhecer tais modelos para, posteriormente, compreender as adaptações desenvolvidas no presente trabalho.

4.1 Heurísticas Convencionais e Metaheurísticas

Esta seção se inicia descrevendo os conceitos fundamentais do que são as heurísticas convencionais (Subseção 4.1.1) e as metaheurísticas (Subseção 4.1.2). Em seguida, de forma detalhada, são explicados cada um dos métodos utilizados neste trabalho.

4.1.1 Heurísticas Convencionais

Problemas de Otimização Combinatória podem ser encontrados em diferentes áreas da engenharia, gestão da informação, medicina, biologia e da indústria. Nestes problemas, temos um conjunto finito ou infinito de soluções, a partir das quais é preciso minimizar ou maximizar uma função de custo. O mais conhecido entre eles é o Problema do Caixeiro Viajante (*Traveling Salesman Problem* – TSP). Nele, um vendedor precisa visitar cada cidade a partir de um conjunto de n cidades e, ao final, retornar à primeira. Desse modo, é preciso encontrar a distância mínima entre todas as possíveis permutações de visita.

Algoritmos heurísticos permitem a solução de problemas em Otimização Combinatória e podem ser classificados em duas categorias: Construtivos e de busca local (BLUM; ROLI, 2008).

Algoritmos Construtivos geram uma solução em várias etapas e, em cada etapa, uma solução parcial é obtida. O método termina quando se atinge o último passo e então é possível chegar a uma solução completa. A ordem em que as etapas são realizadas e o que acontece em cada etapa depende do problema tratado (BLUM; ROLI, 2008).

Os algoritmos de busca local, por sua vez, procuram encontrar soluções de alta qualidade no espaço de soluções por meio de modificações na solução corrente. Esses algoritmos começam com uma solução inicial e geram iterativamente uma nova solução que está "mais próxima" da solução corrente (MICHIELS; AARTS; KORST, 2010).

A representação dessas soluções pode ser dada por estruturas discretas como sequências, permutações, grafos e partições. Os algoritmos de busca local normalmente usam essas representações para definir o tipo de vizinhança que deve ser explorado. Movimentos como rearranjos, trocas e/ou substituição de itens podem ser aplicados a uma solução, de modo a obter uma solução vizinha (MICHIELS; AARTS; KORST, 2010).

Exitem diversas estratégias que podem ser usadas nas heurísticas de busca local, entre elas *Best Improvement* e *First Improvement*.

4.1.2 Metaheurísticas

Assim como as heurísticas, as metaheurísticas procuram solucionar problemas de nosso dia-a-dia. Problemas como o de agendamento de voos, balanceamento de cargas em redes de telecomunicações, alocação de centros de distribuição nas cidades, otimização de rotas de veículos e otimização de tabela de horários de transporte público são alguns dos exemplos de problemas em Otimização Combinatória (BLUM; ROLI, 2008).

A origem das metaheurísticas se deu nas comunidades de Inteligência Artificial e de Pesquisa Operacional e, geralmente, se refere a algoritmos de aproximação para resolver problemas que podem exigir tempo exponencial. As metaheurísticas sacrificam a garantia de encontrar uma solução ótima em favor das boas soluções, mas com tempo computacional significativamente menor (BLUM et al., 2011).

Dessa forma, a metaheurística é um método de solução de um dado problema que orquestra uma iteração entre procedimentos de busca local com altos níveis de estratégias para, assim, criar um processo capaz de ir além da otimalidade local e permitir uma busca robusta no espaço de soluções (GLOVER; KOCHENBERGER, 2010).

Colônia de Formigas, algoritmos evolucionários, *Iterated Local Search*, *Simulated Annealing* e Busca Tabu são alguns exemplos de metaheurísticas (DORIGO; Di Caro, 1999; CASTRO, 2007; LOURENÇO; MARTIN; STÜTZLE, 2003; KIRKPATRICK; GELATT; VECCHI, 1983; GLOVER; MARTÍ, 1986).

4.2 Métodos Heurísticos Convencionais

4.2.1 Busca Local - Best Improvement

A heurística de busca local com a estratégia *Best Improvement*, ou Método de Descida, consiste em analisar todas as soluções vizinhas e se mover para aquela que tiver a melhor avaliação e que represente uma melhora em relação à solução corrente. Se não houver solução de melhora, então o método para e retorna a solução corrente como ótimo local em relação à estrutura de vizinhança utilizada (HANSEN; MLADENOVIC, 2006).

O pseudocódigo desse método é apresentado no Algoritmo 1.

Algoritmo 1: Busca Local com a estratégia *Best Improvement*

Entrada : s' - solução inicial
Saída : s

1 $V = \{s' \in N(s) \mid f(s') < f(s)\ \}$
2 **enquanto** $|V| > 0$ **faça**
3 | *Selecione* $s' \in V$ *, em que* $s' = min\{\ f(s') \mid s' \in V\ \}$
4 | $s \leftarrow s'$
5 | $V = \{\ s' \in N(s) \mid f(s') < f(s)\ \}$
6 **fim**
7 **retorna** s

4.2.2 Busca Local - First Improvement

Este método difere do anterior no sentido de que a decisão para mover-se para uma nova solução vizinha não ocorre após a análise de toda a vizinhança. Especificamente, sempre que um vizinho melhor é encontrado, move-se para esse vizinho. O método também retorna um ótimo local em relação à vizinhança utilizada, situação em que toda a vizinhança é analisada (HANSEN; MLADENOVIC, 2006).

O pseudocódigo desse método é apresentado no Algoritmo 2

Algoritmo 2: BUSCA LOCAL COM A ESTRATÉGIA *First Improvement*

Entrada : s' - solução inicial
Saída : s

1 $V = \{s' \in N(s) \mid f(s') < f(s)\ \}$
2 **enquanto** $|V| > 0$ **faça**
3 | *Selecione* $s' \in V$
4 | **se** *f(s') < f(s)* **então**
5 | | $s \leftarrow s'$
6 | | **retorna** s
7 | **senão**
8 | | $V = \{\ s' \in N(s) \mid f(s') < f(s)\ \}$
9 | **fim**
10 **fim**
11 **retorna** s

4.3 Métodos Metaheurísticos

4.3.1 Simulated Annealing

O *Simulated Annealing* foi proposto por Kirkpatrick, Gelatt e Vecchi (1983) e é um algoritmo que tem, como principal propriedade, explorar os espaços de busca de modo a fazer o uso de movimentos de escalada (*hill-climbing*) na solução corrente, para escapar das armadilhas dos ótimos locais e, assim, tentar encontrar o ótimo global.

Este nome veio de uma analogia com o processo de recozimento físico de materiais sólidos, a exemplo de uma chapa de aço, na qual o material é aquecido e depois resfriado muito lentamente até atingir a sua configuração mais regular possível, isto é, o seu estado mínimo de energia. Dessa maneira, é possível eliminar os defeitos de cristalização, evitando que o material se torne quebradiço.

A característica chave deste algoritmo é permitir movimentos de escaladas que pioram o valor da função objetivo. Ou seja, à medida que o parâmetro temperatura diminui, os movimentos de escalada ocorrem com menor frequência e, portanto, é possível convergir para espaços de busca onde existe probabilidade maior de se encontrar o ótimo global (NIKOLAEV; JACOBSON, 2010).

O Algoritmo 3 ilustra o *Simulated Annealing* para a solução de um problema de minimização, sendo s_0 uma solução inicial, T_0 a temperatura inicial, α a taxa de resfriamento e SA_{max} o número máximo de iterações para se atingir o equilíbrio térmico em uma dada temperatura.

O método se inicia com uma solução gerada aleatoriamente – linha 1. A cada nova

Algoritmo 3: SIMULATED ANNEALING

Entrada : S_0 - Solução inicial
T_0 - Temperatura inicial
α - Taxa de resfriamento
SA_{max} - Número máximo de iterações em uma determinada temperatura

Saída : s^*

```
1  s ← s_0 // Solução corrente
2  s' ← s // Melhor solução obtida até então
3  T ← T_0 // Temperatura Corrente
4  IterT ← 0 // Número de iterações na temperatura T
5  enquanto T > 0 faça
6      enquanto IterT < SA_max faça
7          IterT ← IterT + 1;
8          Gere um vizinho qualquer s' ∈ N(s);
9          Δ ← f(s') − f(s);
10         se ( Δ < 0 ) então
11             s ← s';
12             se ( f(s') < f(s*) ) então s* ← s' ;
13         senão
14             Tome X ∈ [0,1];
15             se ( X < e^{−Δ/T} ) então s ← s';
16         fim
17     fim
18     T ← α× T;
19     IterT ← 0;
20 fim
21 Retorne s*;
```

iteração, é gerada uma solução vizinha – linha 8. Avalia-se, então, a solução corrente e o vizinho. Caso este vizinho seja melhor que a solução corrente, então este passa a ser a nova solução corrente – linhas 10-12. Caso contrário, há a probabilidade de se aceitar uma solução de piora – linhas 13-16.

Esta probabilidade de aceitar ou não a solução de piora é dada pela expressão $e^{-\Delta/T}$, em que T é a temperatura atual.

O método começa com uma temperatura alta (T) e a cada iteração essa temperatura vai diminuindo de acordo com o parâmetro α – linha 18. Dessa forma, há uma probabilidade maior das soluções de piora serem aceitas no início do processo; porém, essa probabilidade vai reduzindo à medida que a temperatura vai diminuindo. São estas soluções de piora que permitem que o algoritmo escape das armadilhas dos ótimos locais.

4.3.2 Variable Neighborhood Search

O *Variable Neighborhood Search* (VNS) (HANSEN; MLADENOVIC, 2001) é um método que tem por objetivo varrer o espaço de busca por meio de trocas sistemáticas de estruturas de vizinhança. O algoritmo procura explorar, inicialmente, as vizinhanças mais próximas da solução inicial e intensifica a busca em torno de uma nova solução sempre quando houver um movimento de melhora. Quando não é mais possível melhorar a solução com a vizinhança corrente, passa-se para uma outra vizinhança mais distante, retornando-se à primeira vizinhança sempre que uma solução de melhora é encontrada. O método termina quando não é mais possível melhorar a solução em todas as estruturas de vizinhança exploradas, situação que caracteriza um ótimo local com relação a todas as vizinhanças adotadas.

É importante salientar que a intensificação em torno de uma solução corrente é realizada por um método de busca local. A estratégia utilizada pelo VNS para não ficar preso às armadilhas dos ótimos locais está na modificação das estruturas de vizinhança.

O pseudocódigo do método é apresentado pelo Algoritmo 4. Neste algoritmo, a solução s' corresponde a um vetor de dimensão fixa, como por exemplo, $m = 100$, e cada posição deste vetor corresponde a um caractere do alfabeto utilizado pelo conjunto de problema-teste, dado, por exemplo: $\mathscr{A} = \{A, G, T, C\}$. A solução s' é então gerada aleatoriamente a partir da solução s na vizinhança $N^k(s)$ – linha 5. Logo depois, a solução s' é submetida a uma busca local, s" – linha 6. Caso o ótimo local de s" seja melhor que a solução corrente s, a busca continua a partir da solução s" e recomeça a exploração na primeira estrutura de vizinhança $N^1(s)$, linhas 8–9. Caso contrário, a busca é realizada na próxima estrutura de vizinhança $N^{(k+1)}(s)$, linha 11. O método termina quando o critério de parada é atingido, que pode ser definido como: tempo máximo de processamento, número máximo de iterações ou número máximo de iterações entre duas soluções de melhora – linha 2.

4.3.3 Discrete Particle Swarm Optimization

Otimização por Nuvem de Partículas ou *Particle Swarm Optimization* (PSO) é um algoritmo de busca populacional inspirado no comportamento social de um conjunto de pássaros em voo com movimentos localmente aleatórios, mas globalmente direcionados. Os indivíduos, conhecidos como partículas, exploram o espaço de soluções sob a influência dos componentes cognitivos e sociais do seu conjunto.

O método foi proposto por Kennedy e Eberhart (1995) para explorar problemas contínuos para otimizar funções não-lineares continuas.

Algoritmo 4: VARIABLE NEIGHBORHOOD SEARCH

Entrada : t_{max} - critério de parada
r_{max} - número máximo de estrutura de vizinhanças
s_0 - solução inicial
Saída : s - solução final

$s \leftarrow s_0$
enquanto t_{max} *não atendido* **faça**
  $k \leftarrow 1$ `// estrutura de vizinhança`
  enquanto $k \leq r_{max}$ **faça**
    $s' \leftarrow$ *Gere um vizinho qualquer* $s' \in N^k(s)$
    $s'' \leftarrow$ *buscaLocal(s')*
    se $(\ f(s'') < f(s)\ )$ **então**
      $s \leftarrow s''$
      $k \leftarrow 1$
    senão
      $k \leftarrow k + 1$
    fim
  fim
fim
Retorne s

Trata-se, pelo exposto, de uma técnica de otimização estocástica, em que cada partícula é uma solução candidata e a sua posição é representada por um vetor de números reais. Cada partícula tem sua própria posição e velocidade dentro do espaço de soluções.

Para determinar as trajetórias, mantem-se para cada partícula sua melhor posição e a melhor posição dentre todas as partículas do enxame, variáveis denotadas por *pbest* e *gbest*. Essas duas melhores posições são usadas para determinar o vetor velocidade. Ou seja, dada uma partícula i, a velocidade (v_i) é determinada de acordo com a Equação (10):

$$v_{id}^{t+1} = w \cdot v_{id}^{t} + c_1 \cdot r_{1i} \cdot (pbest_{id} - x_{id}^{t}) + c_2 \cdot r_{2i} \cdot (gbest_d - x_{id}^{t}) \tag{10}$$

em que v_{id}^{t+1} é a velocidade da partícula i na dimensão d na iteração t. A posição da partícula na dimensão d é dada por x_{id}^{t}. O coeficiente de inércia é dado por w, e c_1 e c_2 representam os coeficientes cognitivo e social, respectivamente. O componente estocástico do método é representado pelas constantes r_{1i} e $r_{2i} \sim U(0, 1)$. Por fim, a melhor posição da partícula i na dimensão d durante toda a busca é dada por $pbest_{id}$ e a melhor posição de todas as partículas na dimensão d é armazenada na variável $gbest_d$ (HARRISON; OMBUKI-BERMAN; P., 2017).

Uma vez calculado o vetor velocidade v_i, a partícula atualiza sua posição para a

próxima iteração de acordo com a Equação (11):

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \tag{11}$$

O Algoritmo *Discrete Particle Swarm Optimization* (DPSO) mantém as mesmas características do PSO, isto é, não há diferenciação nas equações (10) e (11) propostas por Kennedy e Eberhart (1995), Kennedy e Eberhart (1997).

Contudo, é importante ressaltar que, para o PSO, as variáveis devem ser contínuas; caso sejam discretas é preciso usar uma outra representação que, muitas vezes, depende do problema a ser estudado. Kennedy e Eberhart (1997), Maurice (2004) e Strasser et al. (2016) são alguns dos trabalhos que propuseram versões discretas do PSO e foge ao escopo deste trabalho representar as várias formulações propostas.

O pseudocódigo do DPSO, dado pelo Algoritmo 5, se inicia após a criação da solução inicial, ou nuvem de partículas – linha 2. Depois, enquanto o critério de parada não for atendido, isto é, enquanto o número de iterações corrente for menor que a quantidade máxima de iterações – linha 3, cada partícula passa por uma avaliação – linha 5.

A melhor posição de uma partícula é armazenada na posição na variável *pbest*, linhas 6–7, enquanto a melhor das posições já encontradas por todas as partículas do enxame é armazenada na variável *gbest*, linhas 8–9. Na linha 10 é realizada uma busca local na melhor solução até então encontrada, *gbest*. Por fim, atualizam-se os vetores velocidade e posição de cada partícula de acordo com o critério estabelecido pelos operadores no espaço de busca, linhas 14–15.

Algoritmo 5: Discrete Particle Swarm Optimization

Entrada : número de partículas e dimensão; velocidade máxima; número máximo de iterações; peso de inércia; fitness inicial; fitness alvo

Saída : SoluçãoFinal

```
1  DPSO ()
2      NuvemPartículas()
3      enquanto Núm. Iterações ≤ Máx. Iterações faça
4          para cada Partícula faça
5              f'_p ← AvaliaFitness()
6              se f'_p é melhor que f_pbest então
7                  pbest ← s^i
8                  se f'_p é melhor que gbest então
9                      gbest ← s^i
10                     gbest ← buscaLocal(gbest)
11                 fim
12             fim
13             para cada Dimensão faça
14                 Atualiza velocidade da Partícula i
15                 Atualiza posição da Partícula i
16             fim
17         fim
18     fim
19     retorna gbest
```

Capítulo 5

Metodologia

Neste capítulo é descrita a metodologia proposta nesta dissertação para resolver o Problema da Cadeia de Caracteres Mais Próxima. A Seção 5.1 descreve como uma solução para o PCCP é representada. Na Seção 5.2 é apresentada a função de avaliação utilizada. Na Seção 5.3 é descrita a estrutura de vizinhança utilizada para explorar o espaço de soluções do problema. Na Seção 5.4 são explicados dois métodos de construção da solução inicial do problema. Por fim, na Seção 5.5 são apresentados os métodos propostos para resolver o PCCP.

5.1 Representação de uma Solução

Ao longo da Seção 2.1 discutiu-se o PCCP. A fim de recapitular alguns conceitos relacionados a este problema, introduzimos a seguinte notação e definição de acordo com Meneses et al. (2004). Um alfabeto $\mathscr{A} = \{c_1, c_2, \ldots, c_z\}$ é um conjunto finito de elementos, chamado de caracteres, e é daí que serão formadas as cadeias de caracteres. Cada cadeia s é uma sequência de caracteres $(s_1, \ldots, s_m)$, $s_i \in \mathscr{A}$. A dimensão de uma cadeia $|s|$ é o número de elementos na sequência de caractere que compõe a cadeia. Dessa forma, se $s = (s_1 \ldots, s_m)$, então $|s| = m$.

Cada solução do PCCP é representada de acordo com o alfabeto e dimensões do problema-teste. Mais precisamente, dado um problema-teste formado por sequências de caracteres de dimensão n $\mathscr{S} = \{s^1, s^2, \ldots s^n\}$, em que cada sequência de caracteres de dimensão m é constituído por um alfabeto $\mathscr{A}$, tal que $\mathscr{A}(s^i \in \mathscr{A}^m, i = 1, 2, \ldots, n)$, uma solução é então representada por um vetor de m posições, em que cada índice do vetor está sujeito a um caractere qualquer do alfabeto $\mathscr{A}$. Assim, por exemplo, uma solução de dimensão $m = 10$, submetida ao alfabeto $\mathscr{A} = \{A, G, C, T\}$ pode ser representada como:

$s = \{AAGTACATTG\}$.

5.2 Função de Avaliação

O objetivo do problema estudado é encontrar uma sequência alvo (*consensus*), de modo a minimizar a distância máxima de um conjunto de sequências de caracteres. Sendo assim, a função de avaliação utilizada para esse cálculo é a distância de *Hamming*, que pode ser definida da seguinte forma: dado $d_H = (a, b)$, a diferença entre as cadeias a e b tal que $|a| = |b|$ é o número de posições em que se diferem. Então, é possível definir a seguinte função de predição: $\phi : \mathscr{A} \times \mathscr{A} \rightarrow \{0, 1\}$ como $\phi(x, y) = 1$ se e somente se $x \neq y$. Tem-se, então, que:

$$d_H(a, b) = \sum_{i=1}^{|a|} \phi(a_i, b_i) \tag{12}$$

Dessa forma, o PCCP, bem como a função objetivo, são definidos como: dado um grupo finito de $\mathscr{S} = \{s^1, s^2, \ldots, s^n\}$ de cadeias, com $s^i \in \mathscr{A}^m$, $1 \leq i \leq n$, é preciso encontrar uma cadeia central $t \in \mathscr{A}^m$ minimizando d tal que para cada cadeia $s^i \in \mathscr{S}$, $d_H(t, s^i) \leq d$.

Para efeito didático, considere o seguinte exemplo: seja o conjunto $\mathscr{S} = \{$differ, median, length, medium$\}$ de dimensões $n = 4$ e $m = 5$, em que $\mathscr{A} = \{$ a, d, e, f, g, h, i, l, m, n, r, t, u $\}$ é o alfabeto utilizado de dimensão $z = 13$. Então a sequência alvo (*consensus*) t que constitui a solução ótima para este problema é: $t = \{$menfar$\}$ e a distância de *Hamming* é $d = 4$.

5.3 Estrutura de Vizinhança

Com o objetivo de explorar o espaço de soluções do problema proposto, utiliza-se neste trabalho somente um tipo de estrutura de vizinhança, chamado *swap*. Dada, então, uma solução corrente s, a vizinhança é o conjunto de soluções $N(s)$, tal que cada solução $s' \in N(s)$ é obtida a partir de um movimento realizado na solução corrente s. A "caminhada" de s em direção à solução s' é chamada de "movimento" e tal movimento consiste na troca de um caractere do índice i da sequência de caracteres s, por um outro caractere do alfabeto $\mathscr{A}$.

Para ilustrar, seja o exemplo da Figura 3:

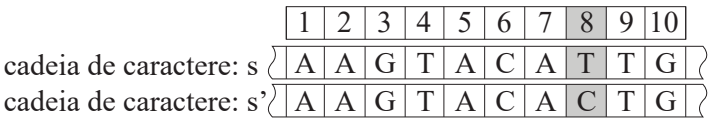


Figura 3 – Exemplo de movimento *Swap*

A Figura 3 ilustra o movimento *Swap* realizado na solução corrente s. Assim, no índice $i = 8$ é realizada a troca do caractere $s_8 = T$ para $s_8 = C$. O vizinho resultante desse movimento é solução s'.

5.4 Construção da Solução Inicial

Nesta seção são apresentados dois métodos de construção de soluções iniciais. No primeiro, Subseção 5.4.1, a solução é construída a partir de um método de busca local; no segundo, Subseção 5.4.2, constrói-se a solução inicial de acordo com a frequência com que os caracteres aparecem no conjunto de problema-teste dado.

5.4.1 Solução Inicial com busca local

A solução inicial é gerada, num primeiro momento, de forma aleatória – vide Algoritmo 6. Desta forma, dada uma cadeia de caracteres s, de dimensão m e sujeita ao alfabeto $\mathscr{A}$, para cada posição de s_i é realizado um sorteio aleatório entre os caracteres que compõe $\mathscr{A}$ – linhas 1–3. Assim, são realizados sorteios de caracteres a cada iteração até se atingir a dimensão m necessária.

Algoritmo 6: Constroi Solução Aleatória

Entrada: $\mathscr{A}$ - alfabeto utilizado
m - dimensão da cadeia de caracteres
Saída : s

1 **para** $i = 1$ *até* m **faça**
2 $\quad s_i \leftarrow$ Gere um caractere aleatório de $\mathscr{A}$
3 **fim**
4 retorne s

Uma vez formada a cadeia de caracteres aleatoriamente, esta, por sua vez, é submetida à heurística de busca local para, assim, garantir que seja encontrado o melhor vizinho – vide Algoritmo 7.

O método se inicia alocando um valor infinito positivo na variável f^*_{vizinho} – vide linha 1. Depois, são realizadas todas as possíveis trocas de caracteres que compõem o alfabeto $\mathscr{A}$ na posição s_i – vide linhas 4–9. Caso esta nova solução seja melhor que todas as soluções até então avaliadas, então redefine-se as variáveis k_{melhor}, i_{melhor} e f^*_{melhor} de acordo com este novo vizinho – vide linhas 10–15. O método finaliza retornando o melhor vizinho encontrado – vide linha 19.

Algoritmo 7: MELHOR VIZINHO

Entrada : s - solução inicial
Saída : s - melhor vizinho
i_{melhor} - indice do vetor do melhor vizinho
k_{melhor} - melhor caractere do alfabeto
f^*_{vizinho} - valor da função de avaliação do melhor vizinho

```
1  f*_vizinho ← +∞;
2  i ← 1;
3  s' ← s;
4  enquanto i ≤ m faça
5      k ← 1;
6      enquanto k ≤ |𝒜| faça
7          se (s_i ≠ 𝒜_k) então
8              s'_i ← troca(s_i, 𝒜_k);
9              f(s') ← avalia(s'_i);
10             se (f(s') < f_vizinho) então
11                 i_melhor ← i;
12                 k_melhor ← k;
13                 f*_vizinho ← f(s');
14                 s ← s';
15             fim
16         fim
17     fim
18 fim
19 retorne s, i_melhor, k_melhor, f*_melhor
```

Com o melhor vizinho determinado, é possível agora encontrar o ótimo local usando uma busca local com a estratégia *Best Improvement*. Assim, o Algoritmo 8 se inicia alocando um valor infinito positivo na variável f^*_{vizinho} – vide linha 1. Depois, toda a vizinhança é avaliada, através do Algoritmo 7, de modo a encontrar a melhor solução local – vide 5–13

Ao final dos Algoritmos 6, 7 e 8, a solução inicial é construída de acordo com o Algoritmo 9.

Algoritmo 8: Busca Local

Entrada : s - solução inicial
Saída : s - melhor solução local

$f^*_{\text{vizinho}} \leftarrow +\infty$;
$melhora \leftarrow$ TRUE;
$s \leftarrow s_i$;
enquanto $melhora == TRUE$ **faça**
  $f_{\text{vizinho}}, i_{\text{melhor}}, k_{\text{melhor}} \leftarrow melhorVizinho(s)$;
  se $(f_{vizinho} < f(s))$ **então**
    $s[i_{melhor}] \leftarrow \mathscr{A}[k_{\text{melhor}}]$;
    $f(s) \leftarrow f_{\text{vizinho}}$;
  senão
    melhora $\leftarrow$ FALSE;
  fim
fim
retorne s

Algoritmo 9: Solução Inicial Com Busca Local

Entrada : $\mathscr{A}$ - alfabeto utilizado
- dimensão da cadeia de caracteres
Saída : s

$s' \leftarrow$ SoluçãoAleatória(s)
$s \leftarrow BuscaLocal(s')$
retorne s

A Figura 4 exemplifica a maneira como são realizadas todas as possíveis trocas nas posições s_1 e s_{10} da sequência central t no alfabeto $\mathscr{A} = \{A, C, T, G\}$.

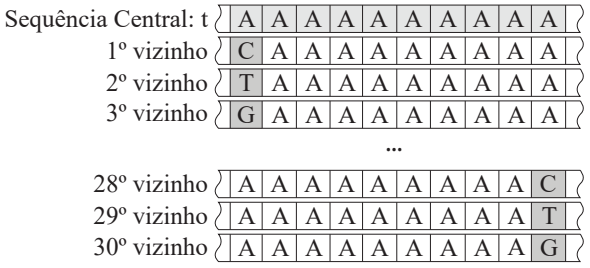


Figura 4 – Movimento de troca utilizado na busca local

5.4.2 Solução Inicial com Tabela de Frequência

Neste método a solução inicial é construída a partir de uma análise prévia do problema-teste dado e foi inspirada na abordagem semelhante a dos autores Kelsey e Lars (2011). Esta análise é mostrada pelo Algoritmo 10 e o método se inicia percorrendo todos os índices j de todas as cadeias de caracteres s^i, vide linhas 2–3. Depois, é realizada a contagem de cada caractere na posição j para todo o conjunto de sequências do problema-teste em estudo, linha 4.

Algoritmo 10: TABELA DE FREQUÊNCIA

Entrada : problemasTeste
Saída : tabelaFrequencia

```
1 SolucaoInicialTabelaFrequencia ()
2     para cada Sequência de Caracteres s^i faça
3         para cada Índice da Seq. de Caractere s^i_j faça
4             Incrementa a ocorrência do caractere s^i_j na tabela de frequência
5         fim
6     fim
7     retorna tabelaFrequencia
```

Assim, dado um conjunto de cadeias de caracteres $\mathscr{S} = (s^i \in \mathscr{S}, i = 1, 2, \ldots, 4)$, em que cada cadeia de caracteres (s^i) de dimensão $m = 10$, é possível então calcular a frequência de cada ocorrência s^i_j.

A Figura 5 elucida de forma mais simples este procedimento. Tome como exemplo a posição do índice 1. Nela, aparece 1 ocorrência do caractere "A", 1 ocorrência do caractere "C"e 2 ocorrências do caractere "G". Tal contagem é realizada para cada uma das posições dos índices do conjunto do problema-teste.

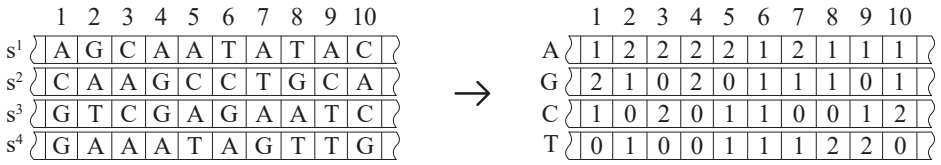


Figura 5 – Exemplo de um Problema-teste e a Tabela de Frequência

Ao final, a solução inicial é construída a partir dos caracteres que possuem maior frequência. Em caso de empate, é realizado um sorteio com os caracteres que empataram. Assim, uma solução inicial para o exemplo acima é $s = \{GAAGAGATTC\}$. Os índices que apresentaram ocorrências iguais e foram aleatoriamente selecionados são $3, 4, 6$.

5.5 Algoritmos para a solução do PCCP

Com o objetivo de resolver o PCCP, neste capítulo são propostos os algoritmos heurísticos que combinam os procedimentos *Simulated Annealing* com busca local; *Variable Neighborhood Search* com a construção da solução inicial a partir da tabela de frequência e; *Discrete Particle Swarm Optimization* com busca local e solução inicial utilizando a tabela de frequência.

5.5.1 Hybrid Simulated Annealing

O algoritmo implementado é uma adaptação da metaheurística clássica *Simulated Annealing*, descrita na Subseção 4.3.1. Trata-se de um algoritmo híbrido, denominado *Hybrid Simulated Annealing for Closest String Problem* (HSA_{CSP}), que incorpora um mecanismo auto-adaptativo para determinar a temperatura inicial de acordo com Souza (2006). Além disso, o método utiliza um procedimento de busca local que é executado sempre que uma nova solução de melhora é encontrada.

O algoritmo 11 parte então de uma solução inicial gerada conforme a Subseção 5.4.1, depois o método entra em um procedimento iterativo, em que, a cada iteração, um nova solução s' é gerada a partir da solução corrente s, linhas 6–9. Posto então que Δ é a variação do valor da função objetivo, temos $\Delta = f(s') - f(s)$, linha 10. Caso $\Delta < 0$, o método aceita o movimento e a solução vizinha passa a ser a corrente, linhas 11–12. Caso contrário, $\Delta \geq 0$, a solução vizinha s' também pode ser aceita, porém com o critério de probabilidade $e^{-\Delta/T}$, sendo T é a temperatura corrente do método e é aquela que regula a probabilidade de aceitar soluções de piora, linhas 14–16.

A temperatura T_0, num primeiro momento, assume um valor alto e é obtida conforme o procedimento auto-adaptativo descrito na Subsubseção 5.5.1.1. Ao longo das interações, a temperatura gradativamente diminui, de acordo com uma razão de resfriamento α, onde $T \leftarrow \alpha \times T$, sendo $0 < \alpha < 1$. Dessa forma, no início do procedimento, há uma maior chance de escapar das armadilhas dos ótimos locais e a medida que T vai se aproximando de zero, o algoritmo tende a ter um comportamento de descida, já que a probabilidade de aceitar soluções de piora diminui.

Por fim, o método utiliza um procedimento de busca local, descrita na Subseção 4.2.1,

que é executado sempre que uma nova solução de melhora é encontrada, linha 13.

Algoritmo 11: HSA_{CSP}

Entrada: S_0 - Solução inicial
T_0 - Temperatura inicial
T_{final} - Temperatura final
α - Taxa de resfriamento
SA_{max} - Número máximo de iterações em uma determinada temperatura

Saída : s^*

```
1  HSAforCSP ()
2      s ← s_0 // Solução corrente
3      s' ← s // Melhor solução obtida até então
4      T ← T_0 // Temperatura Corrente
5      IterT ← 0 // Número de iterações na temperatura T
6      enquanto T > T_final faça
7          enquanto IterT < SA_max faça
8              IterT ← IterT + 1;
9              Gere um vizinho qualquer s' ∈ N(s);
10             Δ ← f(s') − f(s);
11             se ( Δ < 0 ) então
12                 s ← s';
13                 se ( f(s') < f(s*) ) então s* ← BuscaLocal(s') ;
14             senão
15                 Tome X ∈ [0,1];
16                 se ( X < e^(−Δ/T) ) então s ← s';
17             fim
18         fim
19         T ← α × T;
20         IterT ← 0;
21     fim
22     Retorne s*;
```

5.5.1.1 Temperatura Inicial

Para se calcular a temperatura inicial do algoritmo proposto, foi utilizado o método descrito por Souza (2006). O procedimento, dado pelo Algoritmo 12, inicia com uma solução s qualquer e com uma baixa temperatura $T_0 = 2$. Em seguida, é calculada a

quantidade de vizinhos da solução s que são aceitas em um determinado número de interações $SA_{max} = 100$ e em uma dada temperatura, linhas 5–14. Caso o número de vizinhos aceitos seja maior que $\gamma = 95\%$, então o método finaliza e retorna à temperatura inicial que deve ser utilizada no método HSA_{CSP}, linhas 15–17. Caso o número de vizinhos aceitos não seja suficiente, então a temperatura aumenta de acordo com a taxa $\beta = 1,1$ e o processo de contagem de vizinhos repete-se novamente, linha 18. O método somente finaliza quando o mínimo de vizinhos forem aceitos. Ao final, a temperatura retornada por este procedimento é a temperatura inicial utilizada pelo algoritmo HSA_{CSP}.

Algoritmo 12: TEMPERATURA INICIAL

Entrada : β - Solução inicial
γ - Número máximo de iterações em uma determinada temperatura
SA_{max} - Taxa de resfriamento
T_0 - Temperatura inicial
s - Temperatura inicial

Saída : T_0

```
1  T ← T_0 // Temperatura Corrente
2  Continua ← True
3  enquanto Continua == True faça
4      aceitos ← 0 // Número de vizinhos aceitos na Temperatura T
5      para IterT = 1 até SA_max faça
6          Gere um vizinho qualquer s' ∈ N(s);
7          Δ ← f(s') − f(s);
8          se ( Δ < 0 ) então
9              aceitos ← aceitos + 1;
10         senão
11             Tome X ∈ [0,1];
12             se ( X < e^{−Δ/T} ) então aceitos ← aceitos + 1;
13         fim
14     fim
15     se (aceitos ≥ (γ × SA_max) ) então
16         Continua ← False;
17     senão
18         T ← β × T;
19     fim
20     T_0 ← T;
21 fim
22 Retorne T_0;
```

5.5.2 Hybrid Variable Neighborhood Search

O algoritmo VNS, como dito na Subseção 4.3.2, explora o espaço de soluções primeiramente em torno da vizinhança mais próxima da solução corrente. Caso não encontre uma solução de melhora, o método vai gradativamente explorando as vizinhanças mais distantes. Caso seja encontrado uma solução de melhora, o método aceita e se move para essa nova vizinhança. Depois, o algoritmo itera novamente, explorando a vizinhança mais próxima da nova solução corrente. Por fim, o procedimento é finalizado quando o critério de parada é atendido.

O método *Hybrid Variable Neighborhood Search for Closest String Problem* (HVNS_{CSP}) proposto neste trabalho funciona da seguinte maneira. Em cada uma das *iter* iterações, a solução inicial é construída de acordo com o método da tabela de frequência explicada na Subseção 5.4.2.

Para explorar a vizinhança em torno da solução corrente, foi utilizado o movimento de troca (*swap*). Assim, a 1^a vizinhança a ser explorada seleciona aleatoriamente uma posição do vetor da cadeia de caracteres alvo e troca por um outro caractere, escolhido aleatoriamente do alfabeto $\mathscr{A}$. Na 2^a vizinhança, são selecionados aleatoriamente duas posições da solução corrente e são efetuadas duas trocas de caracteres do alfabeto $\mathscr{A}$ de maneira aleatória. O mesmo acontece, respectivamente para as 3^a e 4^a vizinhanças.

O pseudocódigo do método HVNS_{CSP} é apresentado pelo Algoritmo 13 e o procedimento parte de uma solução inicial construída a partir da tabela de frequência (Algoritmo 10), linha 2. Depois, é gerado um novo vizinho a partir da 1^a estrutura de vizinhança, linha 6. Em seguida, é aplicado uma busca local de acordo com o Algoritmo 8 na solução corrente s', linha 7. Caso a solução encontrada s" seja melhor que a melhor solução até então encontrada s, o método move para essa nova vizinhança e volta a explorar o espaço de soluções, linhas 9–10. Caso contrário, $f(s") > f(s)$, o método passa a explorar uma vizinhança mais distante do espaço de busca N^{k+1}, linha 12. O método termina quando

o critério de parada é atendido, t_{max} iterações.

Algoritmo 13: $HVNS_{CSP}$

Entrada : t_{max} - critério de parada
r_{max} - número máximo de estrutura de vizinhanças

Saída : s - solução final

```
1  HVNSforCSP ()
2      s ← SolucaoInicialTabelaFrequencia()
3      enquanto t_max não atendido faça
4          k ← 1 // estrutura de vizinhança
5          enquanto k ≤ r_max faça
6              s' ← Gere um vizinho qualquer s' ∈ N^k(s)
7              s'' ← buscaLocal(s')
8              se ( f(s'') < f(s) ) então
9                  s ← s''
10                 k ← 1
11             senão
12                 k ← k + 1
13             fim
14         fim
15     fim
16     retorne s
```

5.5.3 Hybrid Discrete Particle Swarm Optimization

Como dito na Subseção 4.3.3, o algoritmo DPSO ou Otimização Dicreta por Nuvem de Partículas é a abstração de um processo natural, o qual é inspirada na simulação de um sistema social do comportamento de um bando de pássaros em voo com seu movimento localmente aleatório, porém globalmente direcionado.

Do ponto de vista da posição da partícula, sua representação consiste em um vetor em que cada dimensão são os caracteres do alfabeto. A mudança de posição desse vetor consiste na troca (*swap*) de posições no vetor da partícula. Já para a velocidade, existe uma troca (*swap*) de informações nos índices dos vetores das partícula e sua soma é uma concatenação de listas de *swaps*. Para o PSO contínuo, a velocidade de uma partícula é obtida por um número escalar que será adicionado à posição corrente da partícula para, assim, obter a nova posição da partícula.

O algoritmo proposto, $HDPSO_{CSP}$, mantém as mesmas características do PSO clássico,

isto é, não há diferenciação nas equações (10) e (11) propostas por Kennedy e Eberhart (1995).

O pseudocódigo do $HDPSO_{CSP}$, mostrado no Algoritmo 14, se inicia após a criação da solução inicial, ou nuvem de partículas, que é descrito na Subsubseção 5.5.3.1 – linha 2. Depois, enquanto o critério de parada não for atendido, isto é, enquanto o número de iterações corrente for menor que a quantidade máxima de iterações – linha 3, cada partícula passa por uma avaliação – linha 5.

A melhor posição de uma partícula é armazenada na posição na variável *pbest*, linhas 6–7, enquanto a melhor das posições já encontradas por todas as partículas do enxame é armazenada na variável *gbest*, linhas 8–9. Na linha 10 é realizada uma busca local, de acordo com o Algoritmo 1, na melhor solução até então encontrada, *gbest*. Por fim, atualiza-se os vetores velocidade e posição de cada partícula de acordo com o critério estabelecido pelos operadores no espaço de busca, linhas 14–15.

Algoritmo 14: $HDPSO_{CSP}$

Entrada : TabelaFrequencia, SoluçãoInicial

Saída : SoluçãoFinal

```
1  HDPSOforCSP ()
2      NuvemPartículas()
3      enquanto Núm. Iterações ≤ Máx. Iterações faça
4          para cada Partícula faça
5              f'_p ← AvaliaFitness()
6              se f'_p é melhor que f_pbest então
7                  pbest ← s^i
8                  se f'_p é melhor que gbest então
9                      gbest ← s^i
10                     gbest ← buscaLocal(gbest)
11                 fim
12             fim
13             para cada Dimensão faça
14                 Atualiza velocidade da Partícula i
15                 Atualiza posição da Partícula i
16             fim
17         fim
18     fim
19     retorna gbest
```

Como dito no início desta Seção, o $\text{HDPSO}_{\text{CSP}}$ apresenta diferença significativa somente nas linhas 14 e 15 e, portanto, será dado ênfase a este procedimento de forma didática, de acordo com o exemplo a seguir:

Dada uma partícula $s^i = \{AAACCC\}$, $pbest^i = \{CAAGCC\}$ e $gbest = \{ACGCAA\}$, é preciso identificar a diferença entre os componentes cognitivos e sociais. Esta diferença se dá pelos caracteres diferentes em cada um dos índices descritos. Desta forma, para o componente cognitivo, temos que os índices de número 1 e 4 são diferentes um do outro, ou seja: $s^i \neq pbest^i$ em $\{(1, C, A), (4, G, C)\}$. Já para o componente social, temos que os índices de número 2, 3, 5 e 6 são diferentes um do outro, ou seja: $s^i \neq gbest$ em $\{(2, C, A), (3, G, A), (5, A, C), (6, A, C)\}$. Uma vez identificados estas diferenças, é possível então construir a Equação (10) da seguinte forma:

$$\begin{aligned} v_{id}^{t+1} = w \cdot v_{id}^{t} + \\ c_1 \cdot r_{1i} \cdot (\{(1, C, A), (4, G, C)\}) + \\ c_2 \cdot r_{2i} \cdot (\{(2, C, A), (3, G, A), (5, A, C), (6, A, C)\}) \end{aligned} \tag{13}$$

Prosseguindo com o cálculo do vetor velocidade, as variáveis c_1 e c_2 são os pesos dados para os componentes cognitivo e social respectivamente. Caso haja um peso maior para c_1 em relação a c_2, a partícula tende a se direcionar no espaço de busca em relação a seu próprio caminho (*pbest*), já o contrário, a partícula tende a seguir a melhor partícula da nuvem (*gbest*). Dado, como exemplificação didática, que $c_1 = c_2 = 3$, $r_{1i} = 0,64$ e $r_{2i} = 0,77$, temos que: $c_1 \cdot r_{1i} = 1,92 \approx 2$ e $c_2 \cdot r_{2i} = 2,31 \approx 2$. Considerando, um valor qualquer como exemplificação, que a variável de inércia é $w = 0,60$ e a velocidade atual da partícula é $v_{id}^{t} = \{(1, C, T), (4, C, G)\}$, temos:

$$\begin{aligned} v_{id}^{t+1} = 0,60 \cdot \{(1, C, T), (4, C, G)\} + \\ \mathbf{2} \cdot (\{(1, C, A), (4, G, C)\}) + \\ \mathbf{2} \cdot ((2, C, A), (3, G, A), (5, A, C), (6, A, C)) \end{aligned} \tag{14}$$

Neste momento, o método escolhe **2** posições aleatórias do componente cognitivo: {(1, C, A), (4, G, C)} e **2** posições aleatórias do componente social: {(2, C, A), (5, A, C)}, por exemplo. Ao final, temos a seguinte velocidade:

$$\begin{aligned} v_{id}^{t+1} = 0,60 \cdot \{(1, C, T), (4, C, G)\} + \\ \{(1, C, A), (2, C, A), (4, G, C), (5, A, C)\}) \end{aligned} \tag{15}$$

A componente inércia da Equação 15 diz que 60% do conjunto $\{(1, C, T), (4, C, G)\}$ será selecionado aleatoriamente para compor a nova velocidade v_{id}^{t+1}. Como há somente 2 elementos neste conjunto, então é realizado um arredondamento e, assim, 60% deste conjunto é o mesmo que 50% e, portanto, será escolhido o elemento $\{(4, C, G)\}$. Ou seja:

$$v_{id}^{t+1} = \{(4, C, G), (1, C, A), (2, C, A), (4, G, C), (5, A, C)\} \tag{16}$$

Caso haja mais de um índice em comum, um destes é escolhido aleatoriamente, por exemplo: $\{4, C, G\}$ e $\{(4, G, C)\}$. Por fim, o vetor velocidade fica definido como:

$$v_{id}^{t+1} = \{(1, C, T), (2, C, A), (4, G, C), (5, A, C)\} \tag{17}$$

Agora, aplica-se esta nova velocidade v_{id}^{t+1} na partícula s^i, ou seja, são realizados os seguintes movimentos de troca (*swap*): $s_1^i = T$, $s_2^i = A$, $s_4^i = C$, $s_5^i = C$, ou $s^i = \{TAACCC\}$.

Vale ainda ressaltar que o vetor velocidade carrega três informações importantes, a saber, $(2, A, T)$: Índice a ser modificado na partícula – **2**; origem do caractere: inercial, cognitivo ou social – **A**; para qual caractere será realizada a troca (*swap*) – **T**.

5.5.3.1 Construção da Nuvem de Partículas

O DPSO é um algoritmo populacional e assim, o conjunto de soluções iniciais, também chamado de nuvem de partículas, é criada a partir da construção da tabela de frequência, vide Subseção 5.4.2.

Uma vez construída a tabela de frequência, na linha 2 do algoritmo 15, é estabelecido o critério de parada e logo depois, o algoritmo cria uma partícula vazia – linha 3. Em seguida, foi implementado um método de sorteio probabilístico construído a partir da tabela de frequência, ou seja, realiza-se um sorteio para cada índice da partícula e ganha aquele que tiver maior probabilidade de ocorrência.

Algoritmo 15: NUVEM DE PARTÍCULAS

Entrada : TabelaFrequencia
Saída : s - nuvem de partículas

```
1  NuvemDePartículas ()
2      enquanto Qte. part. na nuvem ≤ Total Partículas faça
3          s^i ←CriaUmaPartículaVazia()
4          para cada Indice da Partícula (j) faça
5              s^i_j ← FazUmSorteioNaTabelaFrequencia()
6          fim
7          pbest^i ← s^i
8          v_i ← Velocidade Inicial
9      fim
10     retorna s
```

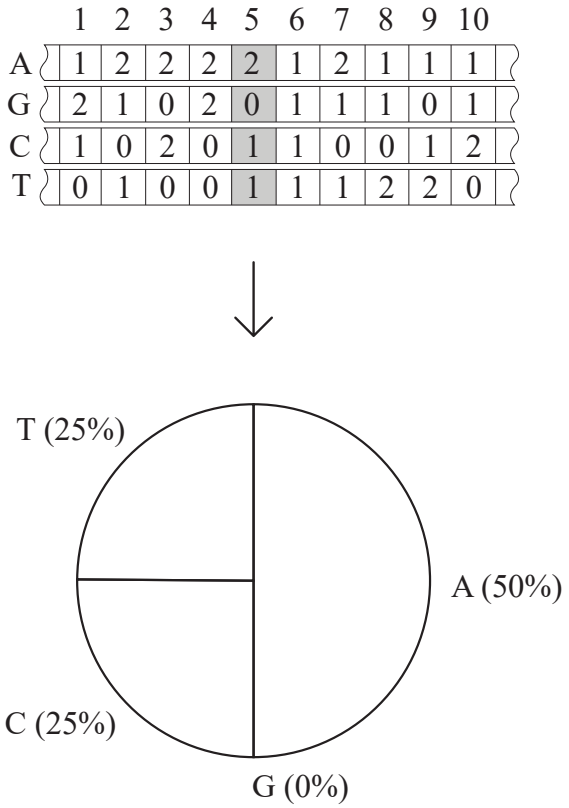


Figura 6 – Exemplo sorteio probabilístico

Por exemplo, na Figura 6, o caractere A, que aparece no índice 5, tem maior probabilidade de ser sorteado que os caracteres G, C e T e as probabilidades, neste caso, são

dadas por: A (50%), G (0%), C (25%) e T (25%). O método encerra atribuindo uma velocidade inicial, no caso 0, a cada uma das partículas criadas – linha 8.

Capítulo 6

Resultados Computacionais

Neste capítulo são apresentados os resultados computacionais obtidos pelos algoritmos HSA_{CSP}, $HVNS_{CSP}$ e $HDPSO_{CSP}$ na resolução do CSP. Primeiramente, são apresentados os conjuntos de problemas-teste utilizados. Na Seção 6.2 são apresentados os resultados obtidos com os três métodos desenvolvidos e o procedimento utilizado para determinar os parâmetros de cada algoritmo. Por fim, na Seção 6.3, foram realizados testes estatísticos de modo a encontrar diferenças entre os métodos.

Os algoritmos foram desenvolvidos na linguagem C++ e compilados no GCC, versão 8.1. Todos os experimentos computacionais foram executados em um computador com processador Intel Core i7 – 3517u – 1.9GHz, com 8GB de memória RAM, no sistema operacional Linux – distribuição Arch, 64 bits. Não foi utilizado otimização para multiprocessamento apesar de o processador possuir mais de um núcleo.

6.1 Problemas-teste

Foram utilizados 36 conjuntos de problemas-teste divididos em 2 grupos, de acordo com a classificação de Chimani, Woste e Bocker (2011). Esses conjuntos têm por base o trabalho de Meneses et al. (2004).

O primeiro grupo contém problemas-teste de dimensões mais elevadas, definidos da seguinte forma: alfabeto utilizado $\mathscr{A} = \{4, 20\}$, quantidade de cadeias de caracteres em cada conjunto de sequências $n = \{40, 50\}$ e a dimensão de cada sequência de caracteres $m = \{5.000, 10.000\}$. Cada conjunto contém 4 problemas-teste, totalizando, assim, 32 problemas-teste.

O segundo grupo contém problemas-teste de dimensões pequenas e médias. O motivo para esta segunda bateria de testes foi comparar os resultados obtidos com aqueles

do trabalho de Faro e Pappalardo (2010), que utilizaram a metaheurística Colônia de Formigas para resolver o CSP e que também foi citado no trabalho de Chimani, Woste e Bocker (2011). Este grupo é então caracterizado como: alfabeto utilizado $=\{4, 20\}$, quantidade de cadeias de caracteres em cada conjunto de sequências $n = \{40, 50\}$ e a dimensão de cada sequência de caracteres $m = \{500, 1.000\}$. São, assim, 4 problemas-teste no total.

6.2 Resultados obtidos com os algoritmos HSA_{CSP}, $HVNS_{CSP}$ e $HDPSO_{CSP}$

Os ótimos globais são conhecidos na literatura para todos os problemas-teste utilizados. Assim, tais valores são exibidos nas tabelas de resultados, na coluna "Literatura", com o objetivo de facilitar a comparação.

Para medir a qualidade da solução retornada por cada um dos métodos em relação ao método de programação inteira (ILP) de Chimani, Woste e Bocker (2011), foi utilizado a Equação (18):

$$gap = \frac{(MétodoProposto)_{média} - ILP}{ILP \times 100} \tag{18}$$

Na Equação (18), a variável *gap*, mensura a diferença percentual entre o valor retornado pelo método proposto com o algoritmo de Chimani, Woste e Bocker (2011). A variável denotada por $MétodoProposto_{média}$ é a média dos resultados de 30 execuções do algoritmo proposto e *ILP* é o resultado do método de Chimani, Woste e Bocker (2011).

6.2.1 Definição dos parâmetros

Os parâmetros dos algoritmos HSA_{CSP}, $HVNS_{CSP}$ e $HDPSO_{CSP}$, utilizados durante as baterias de testes, foram calibrados com a ajuda do pacote *iRace* da linguagem R (LóPEZ-IBáñEZ et al., 2016). Vale mencionar que para calibrar o *iRace*, foram utilizadas todas os problemas-teste utilizados nas simulações.

Irace é um configurador automático de parâmetros baseado no algoritmo *Iterated Racing* e tem sido amplamente utilizado em trabalhos científicos devido ao seu bom resultado prático nas configurações de parâmetros em diferentes tipos de algoritmos. O pacote é distribuído livremente e não requer conhecimento específico sobre seu funcionamento interno.

As próximas subseções detalham as configurações utilizadas pelo *Irace* para cada um dos métodos desenvolvidos.

6.2.1.1 Método HSA_{CSP}

Para o HSA_{CSP}, foram calibrados somente os parâmetros de taxa de resfriamento (α) e número máximo de iterações (SA_{max}) no *Irace*. Os intervalos utilizados nesse procedimento foram:

- $\alpha = \{0.80, 0.90, 0.95, 0.98, 0.99\}$;
- $SA_{max} = \{50, 100, 150, 200, 300\}$.

O parâmetro temperatura inicial foi calculado de acordo com o método da Subsubseção 5.5.1.1. Já a temperatura final, foi fixada em $0,001$. Por fim, todos os parâmetros utilizados pelo método proposto foram:

- $\text{temp}_0 = \{891\}$ - Temperatura inicial;
- $\text{temp}_{\text{final}} = \{0.001\}$ - Temperatura final;
- $\alpha = \{0.98\}$ - Taxa de resfriamento*;
- $SA_{max} = \{100\}$ - Número máximo de iterações em uma determinada temperatura*.

Os parâmetros assinalados por "*" foram determinados pelo *Irace*.

6.2.1.2 Método $HVNS_{CSP}$

Foram calibrados os parâmetros: critério de parada ($t_{\max}$), ou número máximo de soluções de melhora não aceitas consecutivamente; e número máximo de estruturas de vizinhanças ($r_{\max}$). Os intervalos para cada um dos parâmetros calibrados foram:

- $t_{\max} = \{30, 50, 70, 90, 100\}$ - Critério de parada;
- $r_{\max} = \{1, 2, 3, 4, 5, 6\}$ - Número máximo de estrutura de vizinhanças.

A melhor combinação de parâmetros retornada pelo *Irace* foi:

- $t_{\max} = \{70\}$ - Critério de parada;
- $r_{\max} = \{3\}$ - Número máximo de estruturas de vizinhança.

6.2.1.3 Método $HDPSO_{CSP}$

Foram calibrados os parâmetros: número de partículas (tp) na nuvem, componente cognitivo (c_1), componente social (c_2), componente inercial (w) e critério de parada, ou número máximo de soluções de melhora não aceitas consecutivamente (cp). Os intervalos com cada um dos parâmetros calibrados foram:

- $tp = \{20, 40, 60, 100\}$ - Número de partículas;
- $c_1 = \{2, 4, 6, 8, 10\}$ - Componente cognitivo;
- $c_2 = \{2, 4, 6, 8, 10\}$ - Componente social;
- $w = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ - Componente inercial;
- $cp = \{50, 100, 200, 500\}$ - Número máximo de soluções não-aceitas.

A melhor combinação de parâmetros retornada pelo *Irace* foi:

- $tp = \{60\}$ - Número de partículas;
- $c_1 = \{8\}$ - Componente cognitivo;
- $c_2 = \{4\}$ - Componente social;
- $w = \{0.8\}$ - Componente inercial;
- $cp = \{100\}$ - Número máximo de soluções não-aceitas.

6.2.2 Primeira bateria de Testes

A primeira bateria de testes foi realizada com os conjuntos de problemas-teste de grande dimensão propostos por Chimani, Woste e Bocker (2011) que, até o momento, de nosso conhecimento, não haviam sido comparados com outros métodos da literatura.

Os resultados encontrados são resumidos nas tabelas a seguir para cada problema-teste indicado na primeira coluna. As colunas "Resultado" mostram as médias de resultados dos métodos, as colunas "gap" mostram as médias dos "gaps" retornados pelos respectivos modelos e as colunas "Tempo" mostram as médias dos tempos (em segundos) demandados por cada algoritmo. Essa última coluna levou em consideração a quantidade de *GFlops*, número de operações de ponto flutuante por segundo, do processador no qual os métodos propostos foram executados com o processador utilizado pelos autores Chimani, Woste e Bocker (2011) - (Intel Xeon E5540, 8 GB Ram). Há uma eficiência superior de 19% do processador utilizado por estes autores em comparação com o processador utilizado neste trabalho (INTEL, 2018a; INTEL, 2018c).

A Tabela 2 resume o resultado de 30 simulações dos algoritmos HSA_{CSP}, $HVNS_{CSP}$ e

HDPSO$_{\text{CSP}}$ com o alfabeto de 4 dimensões. A Tabela 3 mostra a média dos resultados de 30 simulações, porém com o alfabeto de 20 dimensões.

Tabela 2 – Comparação dos métodos propostos com a literatura - Alfabeto de 4 dimensões

Problema	HSA_{CSP}			$HVNS_{CSP}$			$HDPSO_{CSP}$			ILP	
teste	GAP	Resultado	Tempo	GAP	Resultado	Tempo	GAP	Resultado	Tempo	Resultado	Tempo
4-40-10000-1-0	8.71	7240.90	49.30	1.70	6774.55	462.52	1.58	6766.53	8.26	**6661.00**	1800.00
4-40-10000-1-1	8.66	7242.20	46.32	1.75	6781.95	462.24	1.78	6783.77	8.26	**6665.00**	1800.00
4-40-10000-2-0	34.08	6701.53	48.00	0.45	5020.55	572.66	0.37	5016.63	9.70	**4998.00**	1800.00
4-40-10000-2-1	34.15	6704.57	49.72	0.28	5011.95	572.09	0.28	5011.80	9.78	**4998.00**	1800.00
4-50-10000-1-0	7.72	7273.33	63.25	1.70	6867.00	572.76	1.51	6854.07	10.28	**6752.00**	1800.00
4-50-10000-1-1	7.64	7279.83	63.11	1.62	6872.25	571.28	1.61	6871.80	10.25	**6763.00**	1800.00
4-50-10000-2-0	34.52	6725.93	62.95	0.15	5007.95	721.09	0.18	5009.23	12.15	**5000.00**	1800.00
4-50-10000-2-1	34.52	6726.20	59.52	0.26	5012.90	720.80	0.21	5010.73	12.13	**5000.00**	1800.00
4-40-5000-1-0	5.16	3551.23	17.20	0.96	3409.40	414.07	0.88	3406.67	4.19	**3377.00**	1800.00
4-40-5000-1-1	5.15	3551.03	17.20	0.85	3405.80	388.25	0.79	3403.73	4.17	**3377.00**	1800.00
4-40-5000-2-0	23.81	3095.13	15.89	0.41	2510.20	397.73	0.34	2508.47	4.92	**2500.00**	36.06
4-40-5000-2-1	23.89	3097.37	17.92	0.37	2509.20	409.67	0.35	2508.63	4.91	**2500.00**	6.90
4-50-5000-1-0	5.79	3572.67	21.36	1.97	3443.50	484.37	1.70	3434.53	5.20	**3377.00**	1800.00
4-50-5000-1-1	6.41	3593.37	19.24	3.86	3507.65	519.41	3.82	3506.13	5.21	**3377.00**	1800.00
4-50-5000-2-0	24.10	3102.60	21.25	0.33	2508.15	477.18	0.28	2507.10	6.12	**2500.00**	2.26
4-50-5000-2-1	24.01	3100.37	21.64	0.31	2507.85	487.23	0.27	2506.63	6.13	**2500.00**	2.62

Tabela 3 – Comparação dos métodos propostos com a literatura - Alfabeto de 20 dimensões

Problema	HSA_{CSP}			$HVNS_{CSP}$			$HDPSO_{CSP}$			ILP	
teste	GAP	Resultado	Tempo	GAP	Resultado	Tempo	GAP	Resultado	Tempo	Resultado	Tempo
20-40-10000-1-0	7.30	9385.47	190.06	0.57	8797.00	181.88	0.73	8810.57	3.83	**8747.00**	**81.48**
20-40-10000-1-1	7.27	9385.47	206.78	0.83	8822.00	181.81	0.90	8827.63	4.16	**8749.00**	**101.06**
20-40-10000-2-0	3.95	5197.27	190.10	0.02	5001.00	363.46	0.00	**5000.00**	**4.37**	5000.00	11.64
20-40-10000-2-1	3.78	5188.90	179.92	0.02	5001.00	362.16	0.00	**5000.00**	**4.43**	5000.00	9.92
20-50-10000-1-0	6.47	9402.47	236.54	1.19	8936.00	237.37	0.85	8905.90	12.24	**8831.00**	**287.16**
20-50-10000-1-1	6.42	9402.20	233.38	0.79	8905.00	229.23	0.78	8903.63	12.23	**8835.00**	**287.16**
20-50-10000-2-0	4.04	5201.93	227.83	0.02	5001.00	456.85	0.00	**5000.00**	**3.43**	5000.00	9.92
20-50-10000-2-1	4.07	5203.53	238.97	0.02	5001.00	456.13	0.00	**5000.00**	**3.42**	5000.00	7.78
20-40-5000-1-0	6.42	4651.67	236.54	1.44	4434.00	229.27	1.15	4421.37	12.24	**4371.00**	**38.18**
20-40-5000-1-1	6.35	4653.03	233.38	1.01	4419.00	229.23	1.19	4426.93	12.23	**4375.00**	**1006.14**
20-40-5000-2-0	78.43	4460.63	227.83	0.04	2501.00	456.85	0.00	**2500.00**	**3.43**	2500.00	5.38
20-40-5000-2-1	78.48	4461.97	238.97	0.04	2501.00	456.13	0.00	**2500.00**	**2.42**	2500.00	2.66
20-50-5000-1-0	5.58	4662.60	61.94	1.20	4469.00	507.64	1.16	4467.27	31.03	**4416.00**	**1800.00**
20-50-5000-1-1	5.58	4662.60	62.48	0.84	4453.00	506.50	1.01	4460.63	29.71	**4416.00**	**58.76**
20-50-5000-2-0	78.71	4467.80	63.32	0.04	2501.00	107.39	0.00	**2500.00**	**1.94**	2500.00	3.90
20-50-5000-2-1	78.79	4469.73	63.96	0.04	2501.00	107.12	0.00	**2500.00**	**2.05**	2500.00	4.72

Na Tabela 2, observa-se que o método HSA_{CSP} foi o que obteve pior desempenho. Os métodos $HVNS_{CSP}$ e $HDPSO_{CSP}$ obtiveram faixas de *gaps* semelhantes, ou seja: em 10 problemas-teste obtiveram um $gap < 1\%$, em 5 problemas-teste obtiveram um *gap* entre 1% e 2% e somente em 1 problema-teste obtiveram *gap* superior a 3%. Contudo, o método $HDPSO_{CSP}$ apresentou melhores resultados quando comparado ao $HVNS_{CSP}$. Somente em 3 problemas-teste o $HVNS_{CSP}$ superou o $HDPSO_{CSP}$ e ambos empataram em 1 problema-teste. Vale mencionar que, mesmo os métodos $HVNS_{CSP}$ e $HDPSO_{CSP}$ não encontrando os ótimos globais, em ambos os procedimentos o tempo computacional foi significantemente inferior quando comparado ao da literatura.

Na Tabela 3, observa-se que o método HSA_{CSP} foi o que obteve pior desempenho. O método $HVNS_{CSP}$ obteve 12 $gaps < 1\%$, sendo que, destes, 8 ficaram próximos ao ótimo global e em 4 problemas-teste *gaps* entre 1% e 2%. O método $HDPSO_{CSP}$ encontrou o ótimo global em 8 problemas-teste, em 4 obteve $gaps < 1\%$ e em 4 problemas-teste, *gaps* entre 1% e 2%. Destaca-se, nesse ponto, que os ótimos globais encontrados pelo método $HDPSO_{CSP}$ foram obtidos em um tempo computacional inferior ao método de programação inteira de Chimani, Woste e Bocker (2011). Em alguns casos, essa diferença chegou a 289%.

6.2.3 Segunda bateria de Testes

A segunda bateria de testes foi realizada com os conjuntos de problemas-teste de pequena e média dimensões propostos por Chimani, Woste e Bocker (2011). A motivação desta segunda bateria de testes foi comparar os métodos propostos com o algoritmo Colônia de Formigas de Faro e Pappalardo (2010).

Os resultados encontrados são resumidos nas tabelas a seguir para cada problema-teste indicado na primeira coluna. As colunas "Resultado" mostram as médias de resultados dos métodos, as colunas "gap" mostram as médias dos "gaps" retornados pelos respectivos modelos e as colunas "Tempo" mostram as médias dos tempos (em segundos) demandados por cada algoritmo. Essa última coluna levou em consideração a quantidade de *GFlops*, número de operações de ponto flutuante por segundo, do processador no qual os métodos propostos foram executados com o processador utilizado pelos autores Chimani, Woste e Bocker (2011). Essa equivalência de tempo, processador cerca de 7 vezes inferior, foi também ajustada para o algoritmo de Colônia de Formigas de Faro e Pappalardo (2010) - Intel Pentium M750 (INTEL, 2018a; INTEL, 2018c; INTEL, 2018b).

A Tabela 4 resume o resultado de 30 simulações dos algoritmos HSA_{CSP}, $HVNS_{CSP}$ e $HDPSO_{CSP}$ com o alfabeto de 4 dimensões. Os tempos de execuções do método ILP não

foi mencionado pelo autor.

Tabela 4 – Comparação dos métodos propostos com a literatura - Alfabeto de 4 dimensões - Pequena dimensão

Problema	HSA_{CSP}			$HVNS_{CSP}$			$HDPSO_{CSP}$			ANT			ILP	
teste	GAP	Resultado	Tempo	GAP	Resultado	Tempo	GAP	Resultado	Tempo	GAP	Resultado	Tempo	Resultado	Tempo
4-40-500-1-0	5.90	354.46	0.66	5.77	354.00	3.09	5.83	354.20	0.72	6.96	358.00	7.33	**334.70**	-
4-50-500-1-0	6.59	362.10	0.83	7.74	366.00	3.91	6.77	362.70	0.85	6.56	362.00	16.00	**339.70**	-
4-40-1000-1-0	5.03	701.30	1.04	4.24	696.00	11.87	4.80	699.73	1.15	8.13	722.00	12.90	**667.70**	-
4-50-1000-1-0	4.84	710.00	1.38	5.58	715.00	14.74	4.55	708.00	1.38	7.64	729.00	18.30	**677.20**	-

Na Tabela 4, observa-se que os métodos propostos foram melhores que a abordagem de Faro e Pappalardo (2010). No entanto, os *gaps* dos resultados das soluções são altos, maiores que 4%.

6.3 Análise Estatística

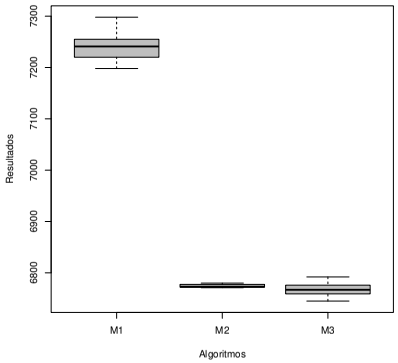
Este capítulo tem por objetivo realizar testes estatísticos com os resultados dos métodos propostos, a fim de classificar e encontrar diferenças estatísticas entre os algoritmos desenvolvidos. Como dito anteriormente, para cada um dos três algoritmos foram realizadas trinta execuções para garantir o teorema do limite central de normalidade.

6.3.1 Comparação estatística com gráfico de caixa

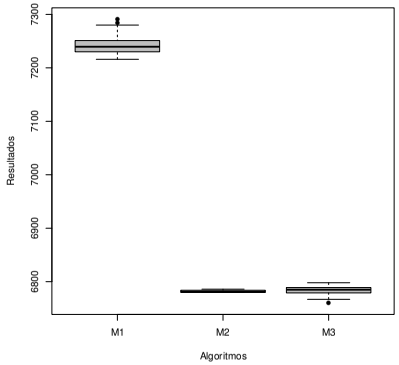
O gráfico *boxplot* é a representação gráfica dos dados numéricos usando os quartis. É um procedimento não paramétrico que tem por objetivo mostrar a variação nas amostras de uma determinada população sem que haja nenhuma hipótese sobre a distribuição dos dados. O topo e a base das caixas representam o primeiro e terceiro quartil, respectivamente, já a linha horizontal dentro da caixa representa a mediana. Os espaços dentro da caixa representam a dispersão dos dados. De modo que, caso não haja sobreposição das caixas, é possível afirmar que existe diferença estatística entre as duas amostras; caso contrário, é preciso realizar uma análise mais detalhada, por exemplo, uma análise de variância (ANOVA). (MONTGOMERY, 2016).

6.3.1.1 Primeira bateria de testes

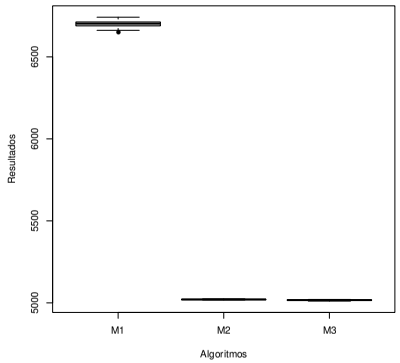
Partindo deste conceito, foram gerados os gráficos *boxplot*, gerados a partir da linguagem R, que comparam os resultados da função objetivo obtidos em cada execução dos algoritmos da primeira bateira de testes: HSA_{CSP} (M1), HVNS_{CSP} (M2) e $\text{HDPSO}_{\text{CSP}}$ (M3) .



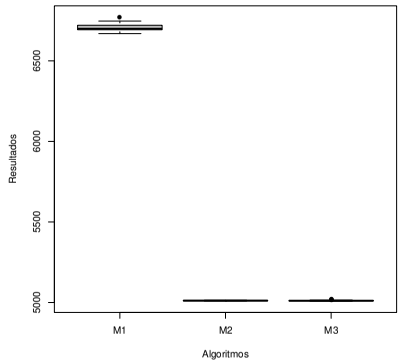
(a) 4-40-10000-1-0*



(b) 4-40-10000-1-1*

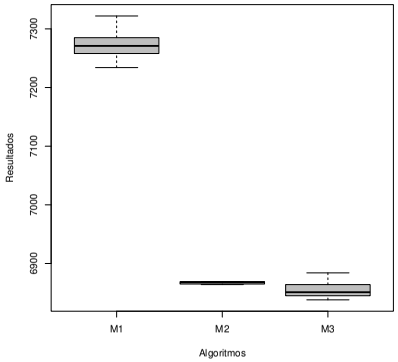


(c) 4-40-10000-2-0*

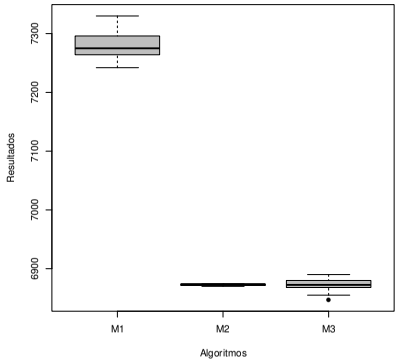


(d) 4-40-10000-2-1*

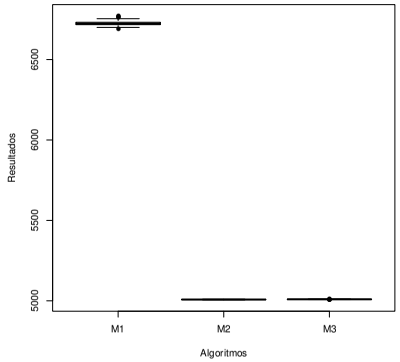
Figura 7 – Boxsplot - Problemas-teste 4-40-10000



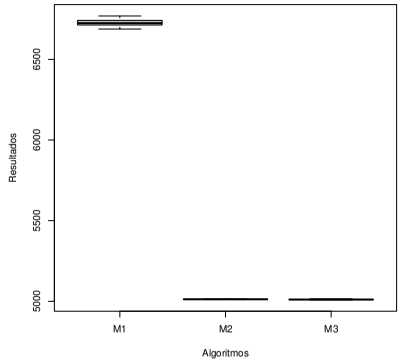
(a) 4-50-10000-1-0*



(b) 4-50-10000-1-1*

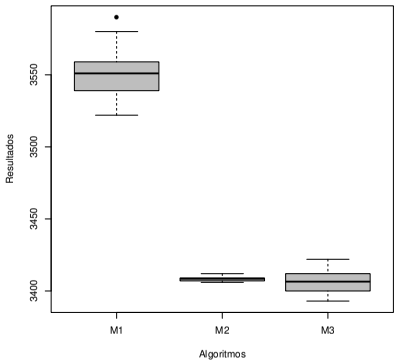


(c) 4-50-10000-2-0*

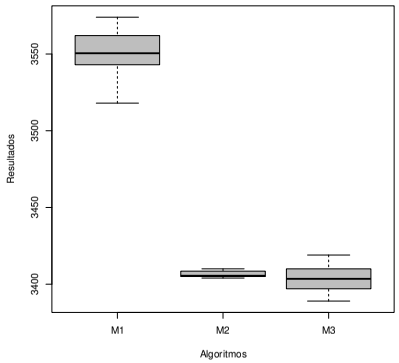


(d) 4-50-10000-2-1*

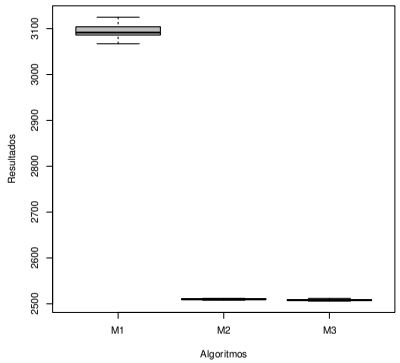
Figura 8 – Boxsplot - Problemas-teste 4-50-10000



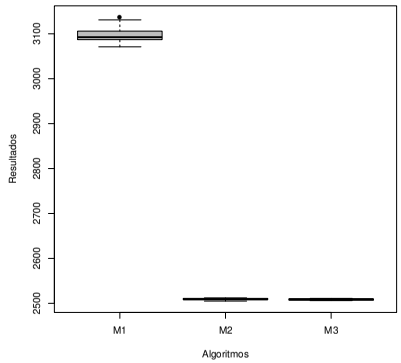
(a) 4-40-5000-1-0*



(b) 4-40-5000-1-1*

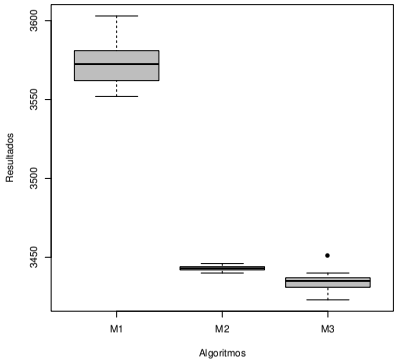


(c) 4-40-5000-2-0*

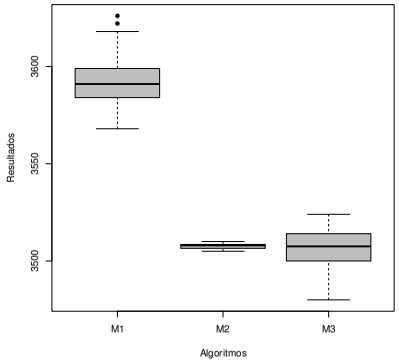


(d) 4-40-5000-2-1*

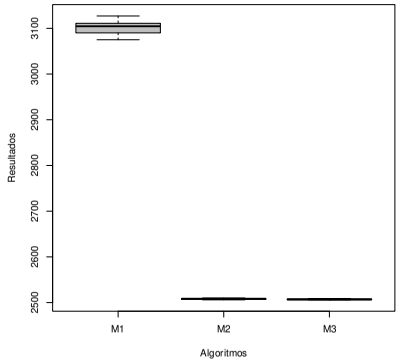
Figura 9 – Boxsplot - Problemas-teste 4-40-5000



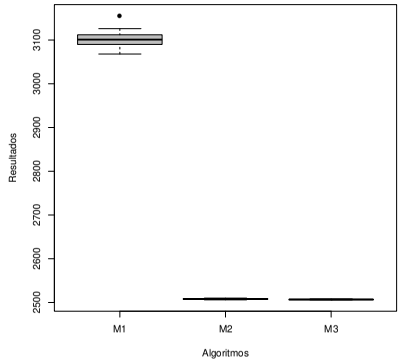
(a) 4-50-5000-1-0



(b) 4-50-5000-1-1*

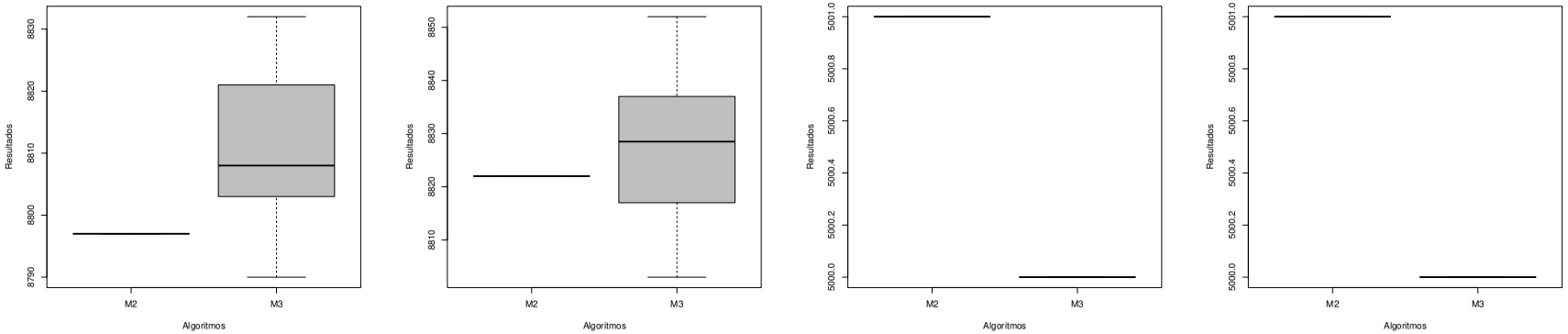


(c) 4-50-5000-2-0*



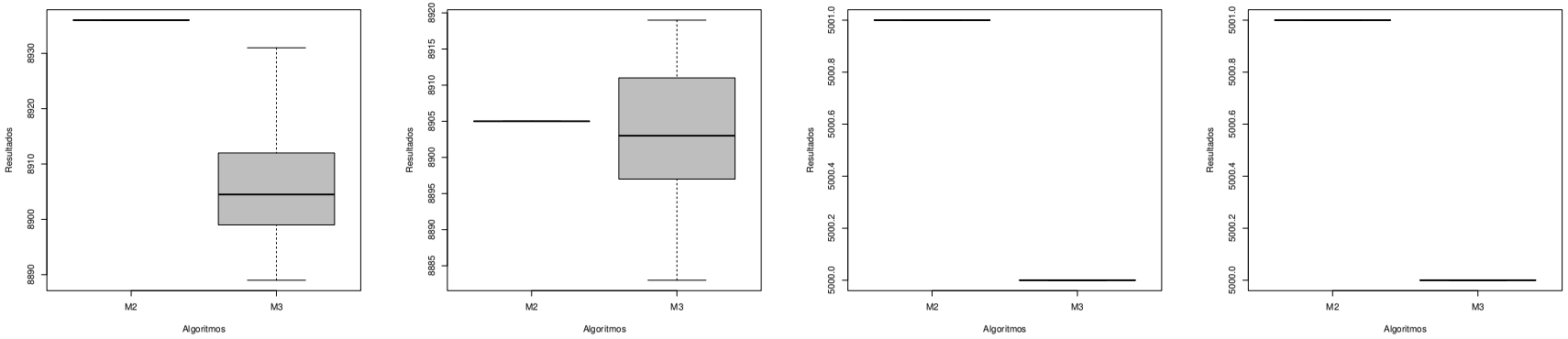
(d) 4-50-5000-2-1*

Figura 10 – Boxsplot - Problemas-teste 4-50-5000



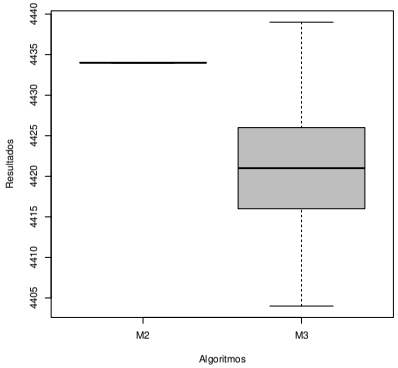
(a) 20-40-10000-1-0 (b) 20-40-10000-1-1* (c) 20-40-10000-2-0 (d) 20-40-10000-2-1

Figura 11 – Boxsplot - Problemas-teste 20-40-10000

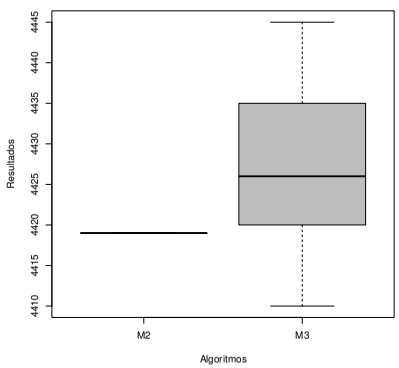


(a) 20-50-10000-1-0 (b) 20-50-10000-1-1* (c) 20-50-10000-2-0 (d) 20-50-10000-2-1

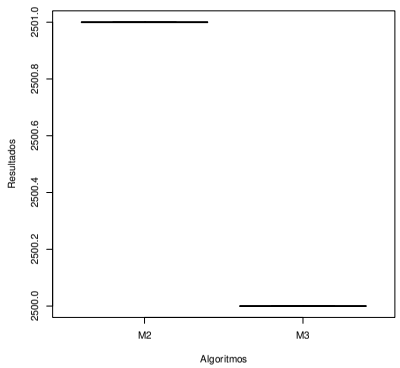
Figura 12 – Boxsplot - Problemas-teste 20-50-10000



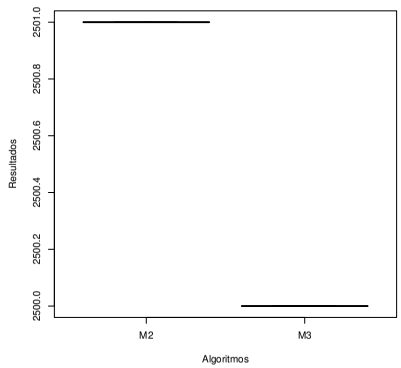
(a) 20-40-5000-1-0



(b) 20-40-5000-1-1

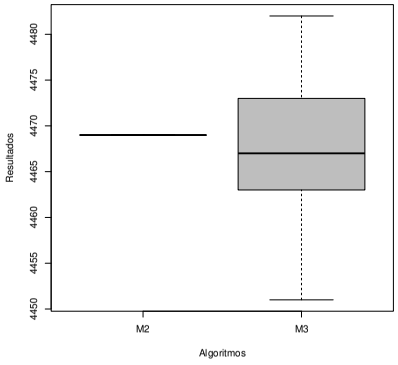


(c) 20-40-5000-2-0

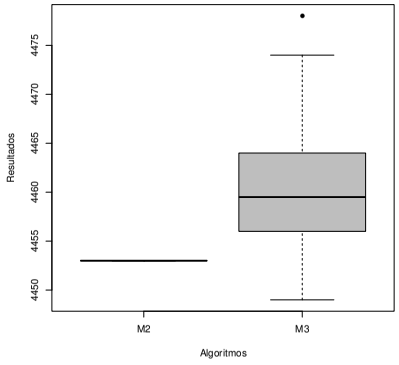


(d) 20-40-5000-2-1

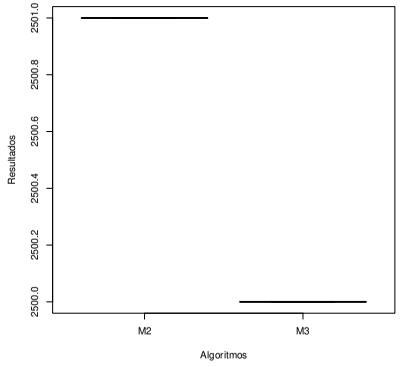
Figura 13 – Boxsplot - Problemas-teste 20-40-5000



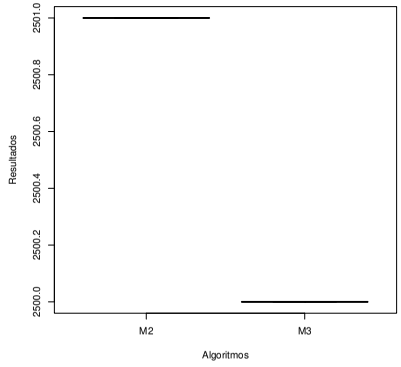
(a) 20-50-5000-1-0*



(b) 20-50-5000-1-1



(c) 20-50-5000-2-0



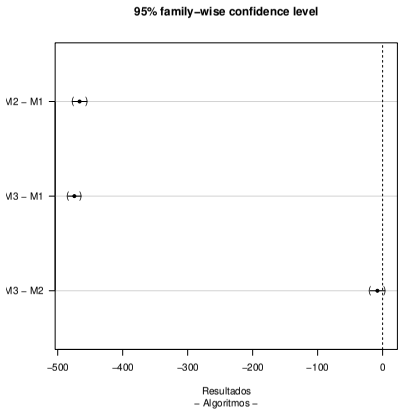
(d) 20-50-5000-2-1

Figura 14 – Boxsplot - Problemas-teste 20-50-5000

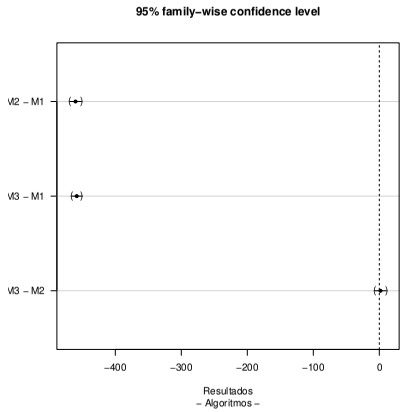
Percebe-se, claramente, que na primeira bateria de testes o algoritmo HSA_{CSP} obteve o pior desempenho quando comparado com os métodos HVNS_{CSP} e $\text{HDPSO}_{\text{CSP}}$, vide Figuras 7, 8, 9 e 10. Este comportamento do HSA_{CSP} permanece o mesmo em todos os problemas-teste da primeira bateria de testes. Sendo assim, para facilitar a análise dos gráficos *boxplot*, optou-se por suprimir esse algoritmo e focar a análise somente nos métodos HVNS_{CSP} e $\text{HDPSO}_{\text{CSP}}$, vide Figuras 11, 12, 13 e 14.

O algoritmo HVNS_{CSP} foi o método que apresentou menor dispersão dos dados e teve desempenho superior ao $\text{HDPSO}_{\text{CSP}}$ em 3 problemas-testes: 11-(a), 13-(b) e 14-(b). O $\text{HDPSO}_{\text{CSP}}$ apresentou uma dispersão de resultados maior que o HVNS_{CSP}, porém foi superior em desempenho em 8 problemas-teste: 10-(a), 11-(c,d), 12-(c,d), 13-(a,c,d). Para os demais 18 problemas-teste assinalados com "*", foi preciso realizar teste estatístico para encontrar nova informação capaz de diferenciar os algoritmos HVNS_{CSP} e $\text{HDPSO}_{\text{CSP}}$.

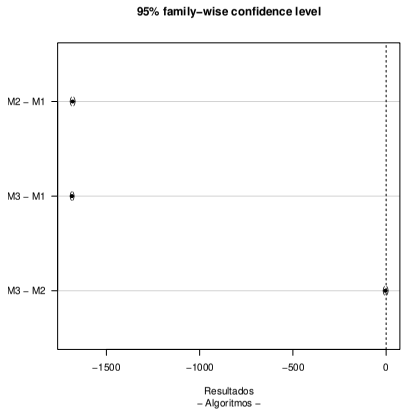
O teste ANOVA com o teste de *Tukey* é um método estatístico que compara médias entre diferentes amostras, com a variância entre todos os indivíduos dentro dessas amostras. A estratégia do teste de *Tukey* é definir a menor diferença significativa entre as médias. O intervalo de confiança utilizado para esta análise foi de 95%.



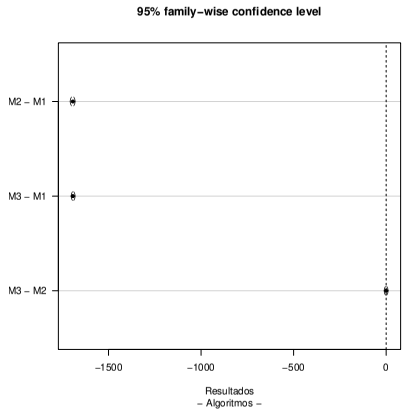
(a) 4-40-10000-1-0



(b) 4-40-10000-1-1

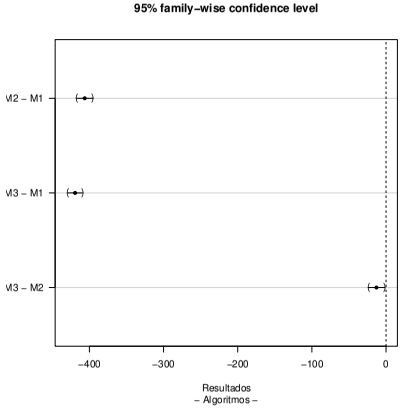


(c) 4-40-10000-2-0

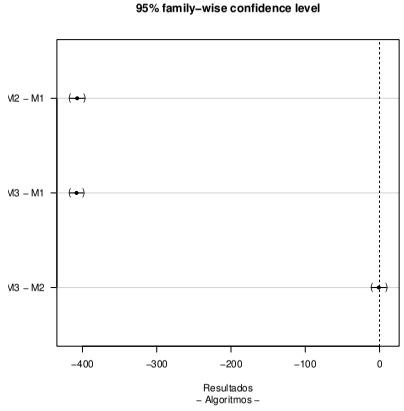


(d) 4-40-10000-2-1

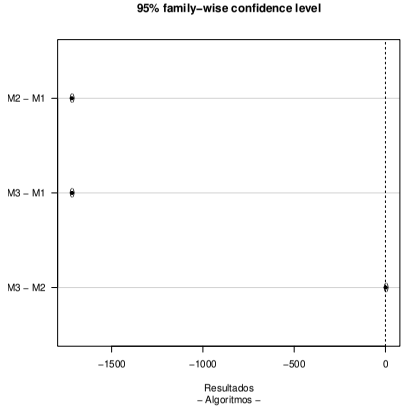
Figura 15 – Tukey - Problemas-teste 4-40-10000



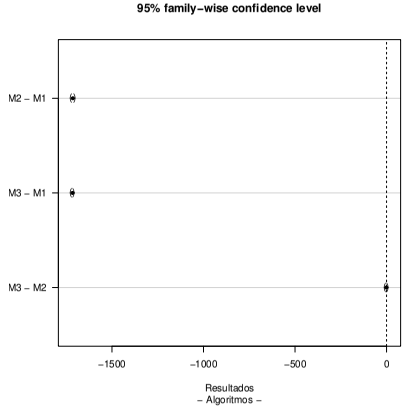
(a) 4-50-10000-1-0



(b) 4-50-10000-1-1

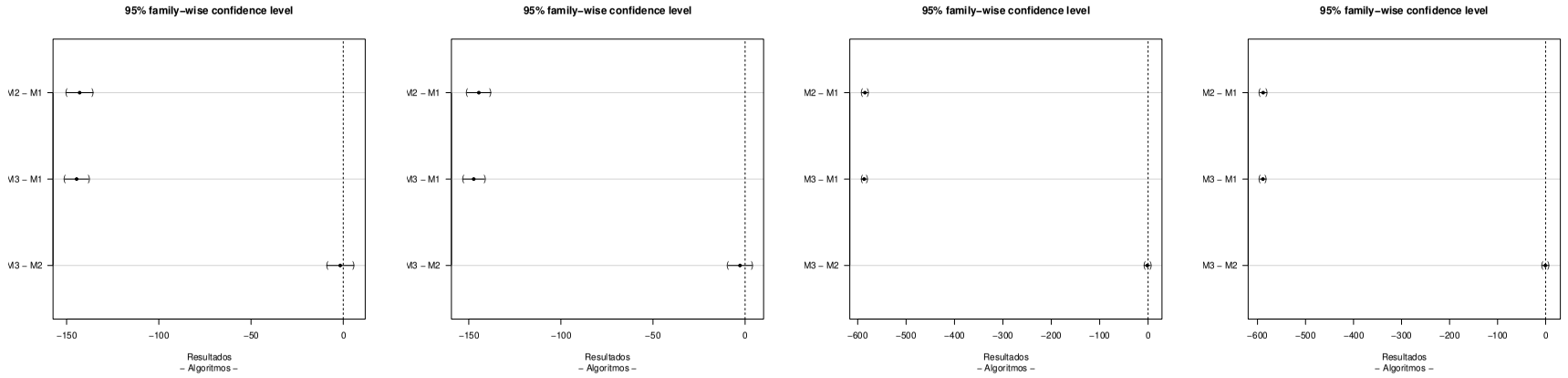


(c) 4-50-10000-2-0



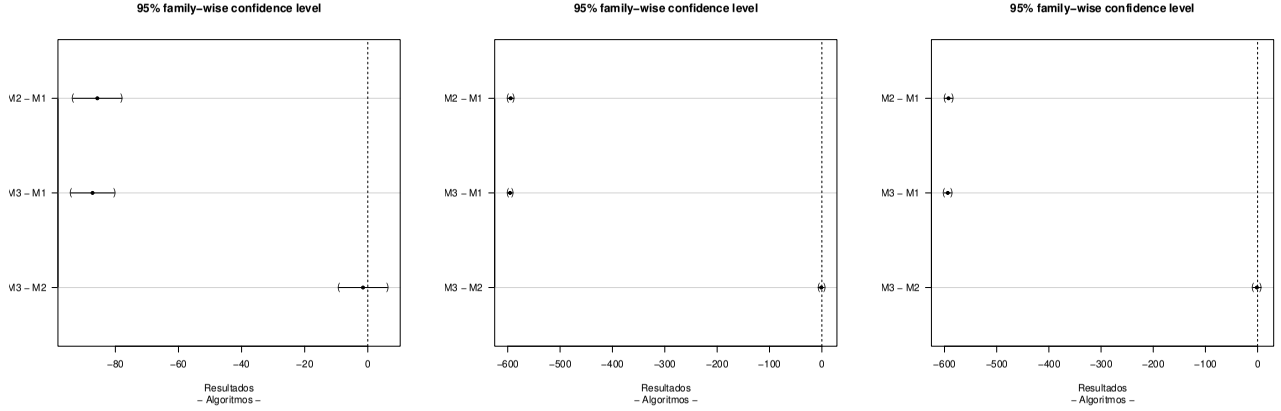
(d) 4-50-10000-2-1

Figura 16 – Tukey - Problemas-teste 4-50-10000



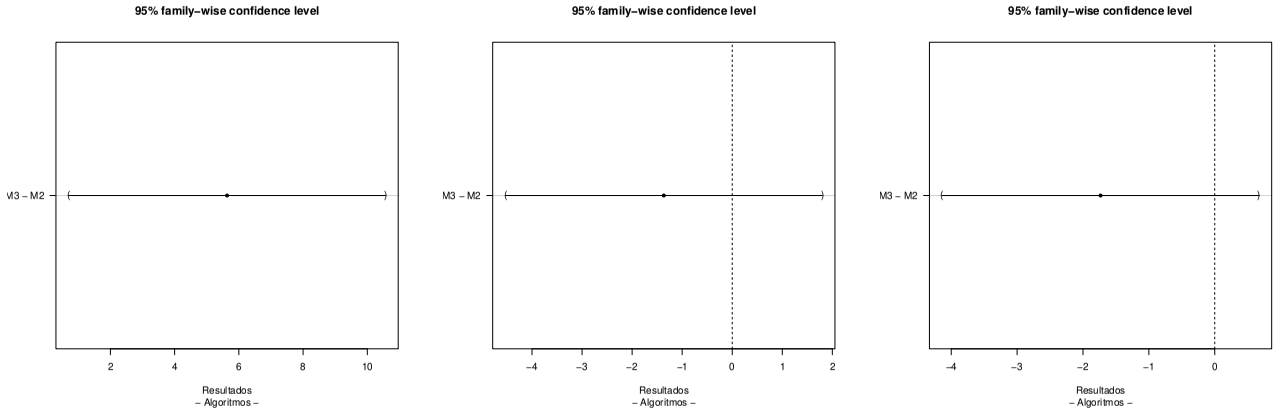
(a) 4-40-5000-1-0 (b) 4-40-5000-1-1 (c) 4-40-5000-2-0 (d) 4-40-5000-2-1

Figura 17 – Tukey - Problemas-teste 4-40-5000



(a) 4-50-5000-1-1 (b) 4-50-5000-2-0 (c) 4-50-5000-2-1

Figura 18 – Tukey - Problemas-teste 4-50-5000



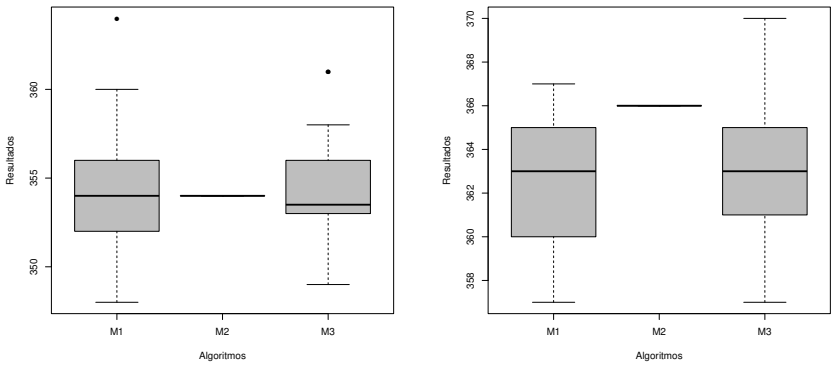
(a) 20-40-10000-1-1 (b) 20-50-10000-1-1 (c) 20-50-5000-1-0

Figura 19 – Tukey - Problemas-teste 20-50-10000 e 20-50-5000

Considerando um nível de significância de 95%, a saída gráfica indica que não é rejeitada a hipótese de igualdade entre as médias dos métodos $HVNS_{CSP}$ e $HDPSO_{CSP}$ para 16 problemas-teste: Figuras 15 (a,b,c,d), 16 (b,c,d), 17 (a,b,c,d) e 18 (a,b,c) e 19 (b,c). Somente houve diferença estatística, ao nível de 95% de confiança, nos problemas-teste 16(a) e 19(a), onde as melhores médias foram para o algoritmo $HVNS_{CSP}$.

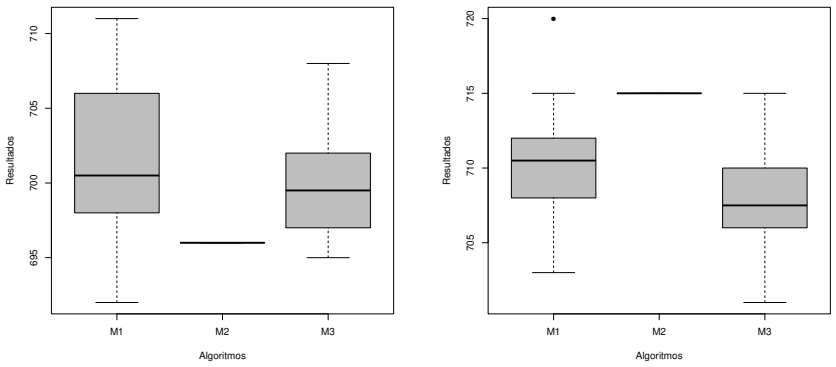
6.3.1.2 Segunda bateria de testes

Partindo do mesmo pressuposto anterior, foram gerados os gráficos *boxplot*, que comparam os resultados da função objetivo obtidos em cada execução dos algoritmos da segunda bateira de testes: HSA_{CSP} (M1), $HVNS_{CSP}$ (M2) e $HDPSO_{CSP}$ (M3).



(a) 4-40-500-1-0* (b) 4-50-500-1-0*

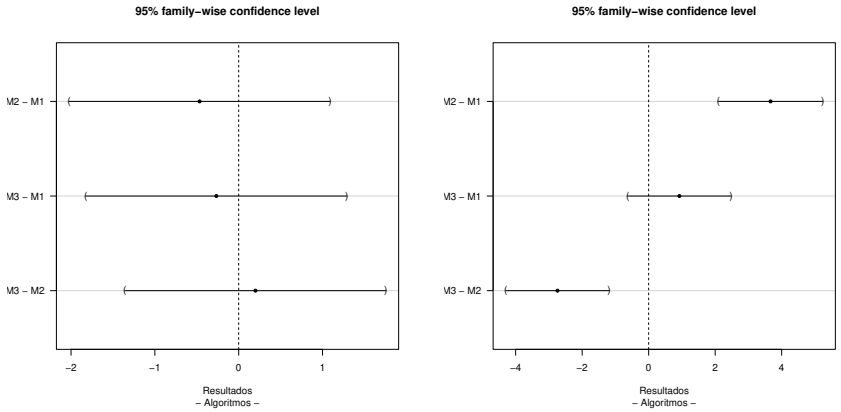
Figura 20 – Boxplot - Problemas-teste 500



(a) 4-40-1000-1-0 (b) 4-50-1000-1-0*

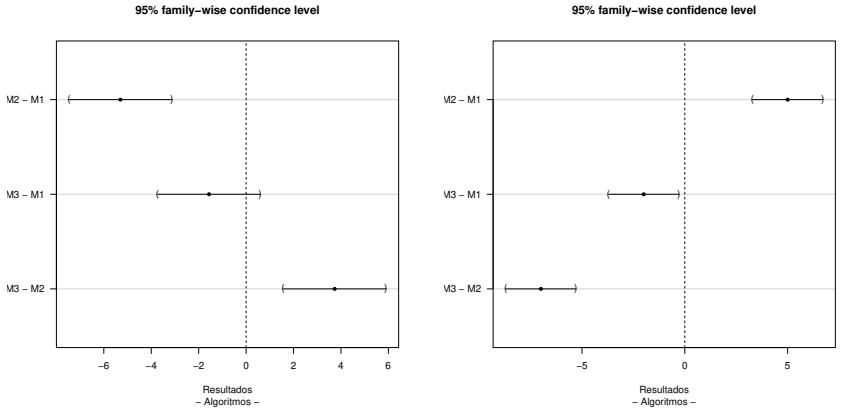
Figura 21 – Boxplot - Problemas-teste 1000

Assim como na primeira bateira de testes, o algoritmo $HVNS_{CSP}$ foi o método que apresentou menor dispersão dos dados e teve desempenho superior aos outros dois métodos somente em um problema-teste: Figura 20(c). De toda forma, foi preciso realizar um teste estatístico para encontrar diferença entre as médias dos métodos já que houve sobreposição das caixas em todos os cenários.



(a) 4-40-500-1-0 (b) 4-50-500-1-0

Figura 22 – Tukey - Problemas-teste Segunda Bateria



(a) 4-40-1000-1-0 (b) 4-50-1000-1-0

Figura 23 – Tukey - Problemas-teste Segunda Bateria

Considerando um nível de significância de 95%, a saída gráfica indica que não é rejeitada a hipótese de igualdade entre as médias dos métodos HSA_{CSP}, $HVNS_{CSP}$ e $HDPSO_{CSP}$ para um problema-teste: Figura 22(a). O algoritmo $HVNS_{CSP}$ foi superior aos métodos HSA_{CSP} e $HDPSO_{CSP}$ em dois problemas-teste: Figuras 22(b) e 23(b). O

algoritmo HSA_{CSP} foi superior aos outros dois métodos somente em um problema-teste: 23(a).

Capítulo 7

Conclusão e Trabalhos Futuros

Este trabalho teve foco no Problema da Cadeia de Caracteres Mais Próxima (PCCP), que tem várias aplicações, como por exemplo no desenvolvimento de novas vacinas e na teoria dos códigos.

Para solucioná-lo, foram propostos três algoritmos híbridos: *Hybrid Simulated Annealing for Closest String Problem*, *Variable Neighborhood Search for Closest String Problem* e *Hybrid Discrete Particle Swarm Optimization for Closest String Problem*.

Para testar os algoritmos de uma forma justa, foi utilizado o pacote *IRace*, Subseção 6.2.1, para encontrar a melhor configuração de parâmetros desses algoritmos. Encontrar a parametrização certa para os algoritmos foi um processo que demandou tempo uma vez que o objetivo do pacote é executar o algoritmos dezenas de vezes sob diferentes parâmetros. Em alguns casos, esse tempo chegou a mais de 120 horas.

Com os algoritmos devidamente configurados, foi preciso validar os métodos por meio de experimentos computacionais, descritos na Seção 6.2.

A seguir foram realizados experimentos estatísticos, descritos na Seção 6.3, para verificar se existem diferenças estatísticas entre os métodos implementados. Foram gerados gráficos caixa e realizados vários testes de *Tukey* para encontrar diferenças estatísticas entre cada um dos métodos.

Após esta análise, foi possível destacar o algoritmo *Hybrid Discrete Particle Swarm Optimization for Closest String Problem* ($\text{HDPSO}_{\text{CSP}}$) como o melhor para os problemas-teste de dimensões mais elevadas. Em alguns casos, esse algoritmo chegou a ter ganho, em tempo computacional, de até 289% quando comparado ao método de Chimani, Woste e Bocker (2011). É importante ressaltar que esta comparação de tempo levou em consideração a quantidade de *Gflops*, número de operações de ponto flutuante por segundo, do processador de Chimani, Woste e Bocker (2011), que era um Intel Xeon

E5540, e do processador utilizado neste trabalho, um Intel i7 3517u (INTEL, 2018a; INTEL, 2018c).

O objetivo geral deste trabalho foi parcialmente atingido. De fato, um dos algoritmos desenvolvidos superou um algoritmo estado-da-arte quando aplicado em instâncias de dimensões elevadas. Entretanto, em instâncias menores, esse algoritmo desenvolvido não foi capaz de encontrar todas as soluções ótimas.

7.1 Trabalhos Futuros

Como trabalhos futuros sugere-se:

- Incorporar a heurística ILS de Chimani, Woste e Bocker (2011) como estratégia de pós-otimização do método $\text{HDPSO}_{\text{CSP}}$;
- Testar novos tipos de movimentos;
- Estudar novas estratégias para otimizar o algoritmo HVNS_{CSP};
- Aplicar os algoritmos desenvolvidos em outros problemas-teste;
- Implementar paralelismo tanto em CPU quanto em GPU para os métodos desenvolvidos;
- Fazer um estudo comparativo com outras metaheurísticas de modo a evidenciar ganhos de desempenho.

Referências

AYUB, A. et al. Global consensus sequence development and analysis of dengue ns3 conserved domains. **BioResearch Open Access**, PMC Web, n. 2(5), p. 392–396, 2013. Citado na página 3.

BEN-DOR, A. et al. **Banishing bias from consensus sequences**. [S.l.]: Springer Berlin Heidelberg, 1997. 247–261 p. Citado na página 13.

BLUM, C. et al. Hybrid metaheuristics in combinatorial optimization: A survey. **Applied Soft Computing**, v. 11, n. 6, p. 4135–4151, 2011. Citado na página 18.

BLUM, C.; ROLI, A. **Hybrid Metaheuristics: An Introduction**. [S.l.]: Springer Berlin Heidelberg, 2008. 1–30 p. Citado 2 vezes nas páginas 17 e 18.

BOUCHER, C. et al. On approximating string selection problems with outliers. Springer Berlin Heidelberg, p. 427–438, 2012. Citado na página 12.

CASTRO, L. N. de. **Fundamentals of natural computing: an overview**. 2007. 1–674 p. Citado na página 18.

CHIMANI, M.; WOSTE, M.; BOCKER, S. A closer look at the closest string and closest substring problem. Society for Industrial and Applied Mathematics, p. 13–24, 2011. Citado 12 vezes nas páginas 3, 6, 8, 9, 10, 15, 41, 42, 44, 47, 61 e 62.

DENG, X.; LI, G.; WANG, L. Center and distinguisher for strings with unbounded alphabet. **Journal of Combinatorial Optimization**, v. 6, n. 4, p. 383–400, 2002. Citado 2 vezes nas páginas 12 e 14.

DORIGO, M.; Di Caro, G. Ant colony optimization: a new meta-heuristic. **Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)**, p. 1470–1477, 1999. Citado na página 18.

DOWNEY, R. G.; FELLOWS, M. R. **Parameterized Complexity**. [S.l.]: Springer Publishing Company, Incorporated, 2012. Citado na página 13.

ECKER, J. G. et al. An application of nonlinear optimization in molecular biology. **European Journal of Operational Research**, v. 138, n. 2, p. 452 – 458, 2002. Citado na página 14.

FARO, S.; PAPPALARDO, E. Ant-csp: An ant colony optimization algorithm for the closest string problem. Springer Berlin Heidelberg, p. 370–381, 2010. Citado 5 vezes nas páginas 14, 15, 41, 47 e 49.

FESTA, P. On some optimization problems in molecular biology. **Mathematical Biosciences**, p. 219–234, 2007. Citado 8 vezes nas páginas 1, 2, 3, 5, 6, 7, 11 e 14.

FRANCES, M.; LITMAN, A. On covering problems of codes. **Theory of Computing Systems**, v. 30, n. 2, p. 113–119, 1997. Citado 2 vezes nas páginas 9 e 13.

GASIENIEC, L.; JANSSON, J.; LINGAS, A. Efficient approximation algorithms for the hamming center problem. p. 905–906, 1999. Citado na página 13.

GLOVER, F.; KOCHENBERGER, G. A. **Handbook of Metaheuristics**. 2nd. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. vii p. Citado na página 18.

GLOVER, F.; MARTÍ, R. Tabu search. p. 1–16, 1986. Citado na página 18.

GOMES, F. C. et al. A parallel multistart algorithm for the closest string problem. **Computers & Operations Research**, v. 35, n. 11, p. 3636–3643, 2008. Citado 3 vezes nas páginas 3, 6 e 15.

GRAMM, J.; NIEDERMEIER, R.; ROSSMANITH, P. **Exact Solutions for Closest String and Related Problems**. [S.l.]: Springer Berlin Heidelberg, 2001. 441–453 p. Citado na página 13.

HANSEN, P.; MLADENOVIC, N. Variable neighborhood search: Principles and applications. **European Journal of Operational Research**, v. 130, n. 3, p. 449–467, 2001. Citado na página 22.

HANSEN, P.; MLADENOVIC, N. First vs. best improvement: An empirical study. **Discrete Applied Mathematics**, v. 154, n. 5 SPEC. ISS., p. 802–817, 2006. Citado na página 19.

HARRISON, K. R.; OMBUKI-BERMAN, B. M.; P., E. A. Optimal parameter regions for particle swarm optimization algorithms. In: **2017 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2017. p. 349–356. Citado na página 23.

INTEL. **Intel Core Processor Series**. 2018. Disponível em: <https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf>. Acesso em: 13 de Maio de 2018. Citado 3 vezes nas páginas 44, 47 e 61.

INTEL. **Intel Legacy Processors Series**. 2018. Disponível em: <https://ark.intel.com/products/27593/Intel-Pentium-M-Processor-750-2M-Cache-1_86-GHz-533-MHz-FSB>. Acesso em: 13 de Maio de 2018. Citado na página 47.

INTEL. **Intel Xeon Processors Series**. 2018. Disponível em: <https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Xeon-Processors.pdf>. Acesso em: 13 de Maio de 2018. Citado 3 vezes nas páginas 44, 47 e 61.

KELSEY, T.; LARS, K. Exact closest string as a constraint satisfaction problem. **Proceedings of the International Conference on Computational Science**, v. 4, p. 1062–1071, 2011. Citado 2 vezes nas páginas 8 e 31.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: **Proceedings IEEE International Conference on Neural Networks**. [S.l.: s.n.], 1995. p. 1942–1948. Citado 3 vezes nas páginas 22, 24 e 37.

KENNEDY, J.; EBERHART, R. C. A discrete binary version of the particle swarm algorithm. In: **IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation**. [S.l.: s.n.], 1997. p. 4104–4108. Citado na página 24.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. **Science**, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. Citado 2 vezes nas páginas 18 e 20.

LANCTOT, J. K. et al. Distinguishing string selection problems. **Information and Computation**, Society for Industrial and Applied Mathematics, p. 633–642, 1999. Citado 6 vezes nas páginas 3, 6, 7, 11, 12 e 13.

LI, M.; MA, B.; WANG, L. Finding similar regions in many sequences. v. 65, p. 73–96, 2002. Citado 2 vezes nas páginas 6 e 9.

LI, M.; MA, B. i.; WANG, L. On the closest string and substring problems. **J. ACM**, ACM, v. 49, n. 2, p. 157–171, 2002. Citado 2 vezes nas páginas 6 e 13.

LIU, X. et al. Compounded genetic and simulated annealing algorithm for the closest string problem. **Bioinformatics and Biomedical Engineering**, p. 702–705, 2008. Citado 4 vezes nas páginas 3, 6, 14 e 15.

LOURENÇO, H.; MARTIN, O.; STÜTZLE, T. Iterated Local Search. **Handbook of Metaheuristics SE - 11**, v. 57, p. 320–353, 2003. Citado na página 18.

LóPEZ-IBáñEZ, M. et al. The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, p. 43–58, 2016. Citado na página 42.

MA, B.; SUN, X. More efficient algorithms for closest string and substring problems. **SIAM J. Comput.**, Society for Industrial and Applied Mathematics, v. 4955, p. 396–409, 2008. Citado na página 9.

MARX, D. The closest substring problem with small distances. v. 2005, p. 63–72, 2005. Citado na página 9.

MARX, V. The big challenges of big data. **Nature**, Nature Publishing Group, a division of Macmillan Publishers Limited. All Rights Reserved. SN -, v. 498, p. 255–260, 2013. Citado na página 3.

MAUCH, H.; MELZER, M. J.; HU, J. S. Genetic algorithm approach for the closest string problem. p. 560–561, 2003. Citado na página 14.

MAURICE, C. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: _____. **New Optimization Techniques in Engineering**. [S.l.]: Springer Berlin Heidelberg, 2004. p. 219–239. Citado na página 24.

MENESES, C. et al. Modeling and solving string selection problems. **BIOMAT 2005**, p. 54–64, 2006. Citado na página 9.

MENESES, C. N. **Combinatorial Approaches for Problems in Bioinformatics**. 105 p. Tese (Doutorado em Industrial and Systems Engineering) — University of Florida, 2005. Citado 2 vezes nas páginas 9 e 14.

MENESES, C. N. et al. Optimal solutions for the closest-string problem via integer programming. **INFORMS Journal on Computing**, v. 16, n. 4, p. 419–429, 2004. Citado 7 vezes nas páginas 3, 6, 8, 14, 15, 26 e 41.

MENESES, C. N.; OLIVEIRA, C. A. S.; PARDALOS, P. M. Optimization techniques for string selection and comparison problems in genomics. **IEEE Engineering in Medicine and Biology Magazine**, v. 24, n. 3, p. 81–87, 2005. Citado na página 14.

MICHIELS, W.; AARTS, E.; KORST, J. **Theoretical Aspects of Local Search**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. 1–10 p. Citado na página 18.

MONTGOMERY, D. C. Estatística aplicada e probabilidade para engenheiros. In: _____. **Estatística aplicada e probabilidade para engenheiros**. [S.l.]: LTC, 2016. p. 217–218. Citado na página 49.

MOUSAVI, S. R.; ESFAHANI, N. N. A grasp algorithm for the closest string problem using a probability-based heuristic. **Computers and Operations Research**, Elsevier Science Ltd., v. 39, n. 2, p. 238–248, 2012. Citado na página 15.

NIKOLAEV, A. G.; JACOBSON, S. H. **Simulated Annealing**. [S.l.]: Springer US, 2010. 1–39 p. Citado na página 20.

ROMAN, S. **Coding and Information Theory**. [S.l.]: Springer-Verlag New York, Inc., 1992. Citado 2 vezes nas páginas 6 e 13.

SCHATZ, M. C. Biological data sciences in genome research. **Genome Res**, Cold Spring Harbor Laboratory Press, v. 25, n. 10, p. 1417–1422, 2015. Citado na página 3.

SETUBAL, J. C.; MEDAINIS, J. **Introduction to Computational Molecular Biology**. [S.l.]: PWS Pub, 1997. 4 p. Citado na página 1.

SOLEIMANI-DAMANEH, M. An optimization modelling for string selection in molecular biology using pareto optimality. **Applied Mathematical Modelling**, v. 35, n. 8, p. 3887–3892, 2011. Citado na página 8.

SOUZA, M. J. F. Departamento de Computação, Universidade Federal de Ouro Preto, **Inteligência Computacional para Otimização**. 2006. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/SA.PPT>. Acesso em: 15 de Maio de 2018. Citado 2 vezes nas páginas 32 e 33.

STRASSER, S. et al. A new discrete particle swarm optimization algorithm. In: **Proceedings of the Genetic and Evolutionary Computation Conference 2016**. [S.l.: s.n.], 2016. (GECCO '16), p. 53–60. Citado na página 24.

TANAKA, S. A heuristic algorithm based on lagrangian relaxation for the closest string problem. **Computers & Operations Research**, p. 709–717, 2012. Citado na página 15.

WATSON, J. D.; CRICK, F. H. C. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. **Nature**, Nature Publishing Group, v. 171, n. 4356, p. 737–738, 1953. Citado na página 1.

ZASLAVSKY, E.; SINGH, M. A combinatorial optimization approach for diverse motif finding applications. **Algorithms for Molecular Biology**, v. 1, n. 1, 2006. Citado na página 9.