UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

THIAGO FELIPE BASTOS DA SILVA

Problema da *String* mais próxima

Profª. Adria Ramos Lyra, D.Sc.
Orientador

Nova Iguaçu, Maio de 2022

Problema da *String* mais próxima

Thiago Felipe Bastos da Silva

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Thiago Felipe Bastos da Silva

Aprovado por:

_____

Profª. Adria Ramos Lyra, D.Sc.

_____

Profª. Fernanda Vieira Dias Couto, D.Sc.

_____

Profª. Juliana Mendes Nascente e Silva Zamith, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Maio de 2022

Emitido em 05/05/2022

DOCUMENTOS COMPROBATÓRIOS Nº 5952/2022 - CoordCGCC (12.28.01.00.00.98)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 12/05/2022 13:13)
ADRIA RAMOS DE LYRA
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: 1604994

(Assinado digitalmente em 16/05/2022 20:25)
FERNANDA VIEIRA DIAS COUTO
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: 1866457

(Assinado digitalmente em 11/05/2022 14:10)
JULIANA MENDES NASCENTE E SILVA ZAMITH
PROFESSOR DO MAGISTERIO SUPERIOR
DeptCC/IM (12.28.01.00.00.83)
Matrícula: 1673110

(Assinado digitalmente em 10/05/2022 12:58)
THIAGO FELIPE BASTOS DA SILVA
DISCENTE
Matrícula: 2018780393

Para verificar a autenticidade deste documento entre em https://sipac.ufrrj.br/documentos/ informando seu número: **5952**, ano: **2022**, tipo: **DOCUMENTOS COMPROBATÓRIOS**, data de emissão: **10/05/2022** e o código de verificação: **15c3d3d24f**

Agradecimentos

Agradeço primeiramente aos meus pais, Claudinei e Andréa por toda a dedicação dada a mim durante todos esses anos.

Também agradeço à minha orientadora Adria Lyra por ter me aceitado como orientando e por me ajudar na realização deste trabalho.

Agradeço a professora Fernanda Couto por ter me dado a chance de fazer a iniciação científica na área de grafos.

Agradeço também aos demais professores por toda a ajuda durante a graduação.

RESUMO

Problema da *String* mais próxima

Thiago Felipe Bastos da Silva

Maio/2022

Orientador: Adria Ramos Lyra, D.Sc.

O problema da string mais próxima tem aplicações na bioinformática e tem como objetivo encontrar uma string t, dado um conjunto S com cadeias de mesmo comprimento, tal que a distância de Hamming máxima entre t e todas S_i, $1 \leq$ i $\leq |S|$, seja mínima. Uma vez que pertence à classe np-difícil não é conhecido um algoritmo com complexidade polinomial para resolvê-lo. Assim para se obter uma solução se usa algum método exato com restrição ou se utiliza alguma heurística que não garante a otimalidade mas obtém-se a resposta em um tempo razoável. Com isso, este trabalho se propõe a criar algoritmos baseados em metaheurísticas já conhecidas como: Simulated Annealing, Iterated Local Search e o Algoritmo Genético e comparar os resultados com o que já foi obtido na literatura.

ABSTRACT

Problema da *String* mais próxima

Thiago Felipe Bastos da Silva

Maio/2022

Advisor: Adria Ramos Lyra, D.Sc.

The Closest String Problem has many applications in bioinformatics and your target is to find a string t, given a set S with strings of same length, such that maximum Hamming Distance between t and all S_i, $1 \leq i \leq |S|$, *be minimum. Since it belongs the np-hard class is no known a polynomial algorithm in order to solve it. Therefore, to get a solution uses some exact method with restriction or uses some heuristic that not guarantees optimality but the answer is obtained in a reasonable time. Therefore, this work proposes to create algorithms based on metaheuristics already known as: Simulated Annealing, Iterated Local Search and Genetic Algorithm and compare the results with what has already been obtained in the literature.*

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Abreviaturas e Siglas

GA	Algoritmo Genético
ILS	Iterated Local Search
lb	lower bound
ub	upper bound
SA	Simulated Annealing
SAP	Simulated Annealing Paralelo

Lista de Símbolos

Sumário

Capítulo 1

Introdução

O problema da string mais próxima pertence à classe dos np-difíceis (FRANCES; LITMAN, 1997) e tenta quantificar a semelhança entre um conjunto de cadeias ao encontrar a string que esteja a uma distância mínima de todas as outras. Consiste em receber uma sequência S com n strings, cada uma de comprimento m e encontrar uma string t em que a maior distância entre t e S_i, $1 \leq i \leq m$, seja a menor possível. Para medir a distância usamos a distância de Hamming em que dada duas strings calcula a quantidade de posições em que os caracteres não correspondem.

1.1 Objetivo do Trabalho

O intuito desse trabalho é realizar um estudo sobre o Problema da string mais próxima: apresentar heurísticas e estratégias ao implementar meta-heurísticas já existentes e combiná-las a fim de conseguir soluções de qualidade com baixo tempo de execução e por fim comparar os resultados obtidos com alguns presentes na literatura.

1.2 Organização do Trabalho

O capítulo 2 traz conceitos para o entendimento do problema e são apresentados: a distância de Hamming e o Problema da string mais próxima.

No capítulo 3 a nossa estratégia para alcançar bons resultados é construída através dos algoritmos de solução inicial 3.1 e das meta-heurísticas das quais são extraídas as nossas heurísticas. Além disso, todos os conceitos para se entender os algoritmos e inspirações de algumas das ideias são abordadas nesta parte.

O capítulo 4 referente aos experimentos realizados sobre instâncias encontradas na literatura apresenta os resultados das heurísticas elaboradas e do seus tempos de execução como também todos os detalhes relativos ao ambiente de testes, linguagem utilizada, etc.

O capítulo 5 apresenta uma comparação entre os algoritmos, as dificuldades encontradas e as possíveis extensões do trabalho.

1.3 Aplicações

Entre as aplicações relacionadas ao Problema da string mais próxima estão: identificar alvos de drogas genéticas (LI; MA; WANG, 2000), determinar um consenso imparcial de uma família de proteínas (BOUCHER, 2011), achar regiões conservadas em sequências não alinhadas (LI; MA; WANG, 2000), gerar sondas genéticas em biologia molecular (LI; MA; WANG, 2000), encontrar motifs (BOUCHER, 2011).

Capítulo 2

Fundamentação

2.1 Distância de Hamming

Dada duas cadeias s e t com |s| = |t| , a distância de Hamming d(s, t), é a quantidade de posições em que possuem carecteres diferentes.

$$d(s,t) = \sum_{i=1}^{n} f(s_i, t_i) = \begin{cases} 0, & s_i = t_i \\ 1, & s_i \neq t_i \end{cases}$$

2.1.1 Exemplos

Figura 2.1: distância de Hamming

Fonte: o autor.

Como as posições na figura 2.1, em vermelho, em que $S_i \neq T_i$ correspodem à quatro do total, $d(S,T)$ é igual a 4.

2.2 Problema da string mais próxima

Seja S uma sequência de cadeias $\{S_1, S_2, S_3, \ldots, S_n\}$ com $|S_i|$ = m que possuem caracteres de um alfabeto Σ. O problema da string mais próxima consiste em encontrar uma cadeia T também de comprimento m, em que a maior distância de Hamming entre T e todas as cadeias da entrada S é mínima, ou seja, T = $argmin_T\{max_{1\leq i\leq n}\{d(S_i, T)\}\}$

2.2.1 Exemplos

#	1	2	3
S_1	a	a	a
S_2	b	a	b
S_3	a	b	a
T	a	a	a

Tabela 2.1: strings de entrada S e uma candidata à solução T

#	$d(S_1, T)$	$d(S_2, T)$	$d(S_3, T)$	$max_{1\leq i\leq n}\{d(S_i, T)\}$
T	0	2	1	2

Tabela 2.2: Distâncias entre as strings de S e T

Em 2.2.1 é apresentado a entrada e uma candidata a solução T e em 2.2.1 calcula-se a distância entre T e todas as strings da sequência S.

Capítulo 3

Metodologia

Na seção 3.1 são apresentados os algoritmos de solução inicial, na seção 3.2 são apresentadas duas propostas do simulated annealing: serial e paralela, na seção 3.3 é apresentado o ILS com o simulated annealing funcionando como a busca local e em 3.4 o algoritmo genético é abordado.

3.1 Algoritmos de solução inicial

Na seção 3.1 são apresentados quatro versões de algoritmos de solução inicial e entre eles tem-se o algoritmo 1 que gera strings de forma aleatória se baseando nos caracteres da sequência da entrada, o algoritmo 2 que utiliza uma estratégia gulosa inspirada , o algoritmo 3 que tenta construir uma solução satisfatória a partir das strings de entrada e o algoritmo 4 que tenta aprimorar os resultados de 3.

3.1.1 Algoritmo de solução incial 1

O algoritmo 1 consiste em gerar uma string em que cada posição $1 \leq j \leq$ m contém um caractere escolhido aleatoriamente dentre os caracteres presentes na coluna j das cadeias de entrada.

Algoritmo 1: Algoritmo de solução inicial 1

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia T de comprimento m

```
1 para cada j ← 1 to m faça
2 |  Σ' ← {};
3 |  para cada i ← 1 to n faça
4 |  |  Σ' ← Σ' ∪ {S_{i,j}};
5 |  fim
6 |  T_j ← aleatorio(Σ');
7 fim
8 retorna T;
```

No algoritmo 1 para cada coluna $1 \leq j \leq m$, a sequência $\Sigma^{'}$, inicialmente vazia (linha 2), é preenchida com os caracteres dessa coluna (linhas 3 à 5) e a partir dela um elemento é escolhido aleatoriamente e atribuído a T_j (linha 6).

3.1.2 Algoritmo de solução incial 2

O algoritmo 2 que foi baseado no trabalho feito por (SANTOS, 2018) em seu mestrado segue uma estratégia gulosa para a construção da cadeia e para isso escolhe para cada posição um dos caracteres mais frequentes nessa posição de forma aleatória.

Algoritmo 2: Algoritmo de solução inicial 2

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia T de comprimento m

```
1  para cada j ← 1 to m faça
2  |  frq ← {0, 0, 0, ..., 0};
3  |  para cada i ← 1 to n faça
4  |  |  frq_{S_{i,j}} ← frq_{S_{i,j}} + 1;
5  |  fim
6  |  max_frq ← max_{c∈Σ}{frq_c};
7  |  caracteres ← {};
8  |  para cada c ∈ Σ faça
9  |  |  se max_frq = frq_c então
10 |  |  |  caracteres ← caracteres ∪ {c};
11 |  |  fim
12 |  fim
13 |  T_j ← aleatorio(caracteres);
14 fim
15 retorna T;
```

No algoritmo 2 para cada coluna $1 \leq j \leq m$, um contador para os caracteres do

alfabeto Σ , a variável frq, é inicializada com zeros (linha 2) e logo após a frequência de cada caractere é contabilizada (linhas 3 à 5) e com isso um dos caracteres que possuem a frequência máxima presentes em *caracteres* é escolhido aleatoriamente e atribuído a string T na posição j (linhas 8 à 13).

3.1.3 Exemplo

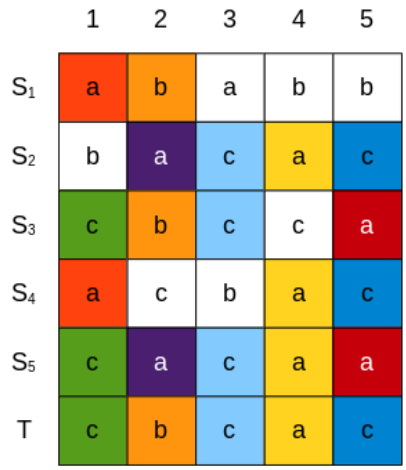

Figura 3.1: strings de entrada e uma possível solução

Na figura 3.1 atribui-se em T_i um dos caracteres mais frequentes presentes na coluna i, que por sua vez estão coloridos, na mesma posição onde se encontram.

3.1.4 Algoritmo de solução incial 3

O algoritmo 3 consiste em gerar uma string T ao associar cada uma de suas posições à uma das cadeias de entrada de tal forma que a diferença entre a quantidade recebida por uma string difira em no máximo uma unidade das demais. Com a associação realizada, os caracteres nas posições atribuídas às cadeias da entrada são atribuídos nas mesmas posições na string T.

Algoritmo 3: Algoritmo de solução inicial 3

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia T de comprimento m

```
1  S ← embaralha(S);
2  P ← embaralha({1,2,3,...,m});
3  fracao ← ⌊m/n⌋;
4  resto ← m mod n;
5  pos ← 1;
6  para cada i ← 1 to n faça
7  |  para cada k ← 1 to fracao faça
8  |  |  T_{P_pos} ← S_{i,P_pos};
9  |  |  pos ← pos + 1;
10 |  fim
11 |  se resto > 0 então
12 |  |  T_{Ppos} ← S_{i,P_pos};
13 |  |  pos ← pos + 1;
14 |  |  resto ← resto − 1;
15 |  fim
16 fim
17 retorna T
```

No algoritmo 3, a sequência S da entrada tem a ordem de suas strings embaralhadas (linha 1), a permutação com elementos de 1 até m também é embaralhada (linha) 2, e para cada $1 \leq i \leq n$, as posições que estão na permutação P no intervalo $[pos, pos + fracao - 1]$ são usadas para indexar os elementos de S_i e atribuí-los nas mesmas posições em T e caso $i \leq (m \bmod n)$ o intervalo usado para indexação é $[pos, pos + fracao]$ (linhas 7 à 15) e com isso pos é incrementado com o tamanho do intervalo.

3.1.5 Algoritmo de solução incial 4

3.1.5.1 Mesclar strings

Dada duas strings s e t, uma cadeia st é uma mescla de s e t se cada um de seus caracteres se originaram de uma dessas strings.

O algoritmo 4 consiste em gerar uma string T mesclando as strings da entrada em uma ordem pré-definida. Sendo que cada mesclagem destina $\frac{1}{i}$ das posições que possuem caracteres diferentes para uma das strings e o resto para a outra.

Algoritmo 4: Algoritmo de solução inicial 4

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia T de comprimento m

```
1  S ← embaralha(S);
2  T ← {};
3  T_1 ← S_1;
4  para cada i ← 2 to n faça
5  |  porcentagem ← 1/i;
6  |  posicoes ← {};
7  |  T_i ← T_{i-1};
8  |  para cada j ← 1 to m faça
9  |  |  se S_{i,j} ≠ T_{i-1,j} então
10 |  |  |  posicoes ← posicoes ∪ {j};
11 |  |  fim
12 |  fim
13 |  posicoes ← embaralha(posicoes);
14 |  q ← |posicoes|;
15 |  fracao ← ⌈q * porcentagem⌉;
16 |  para cada j ← 1 to fracao faça
17 |  |  T_{i,posicoes_j} ← S_{i,posicoes_j};
18 |  fim
19 fim
20 retorna T_n
```

No algoritmo 4 a sequência S tem a ordem de suas strings embaralhadas (linha 2) e então a sequência de strings T que pussui S_1 como primeiro elemento (linha 3) é construida: para cada $2 \leq i \leq n$ as posições j em que $S_{i,j} \neq T_{i-1,j}$ são inseridas em *posicoes* que é embaralhada aleatoriamente logo após isso e os primeiros $\lceil \frac{|posicoes|}{i} \rceil$ de seus elementos são usados para indexar os caracteres de S_i e atribuí-los nas mesmas posições em T_i.

3.2 Simulated Annealing

O Simulated Annealing (SA) é uma meta-heurística criada por (KIRKPATRICK; GELATT; VECCHI, 1983) baseada no algoritmo de Monte Carlo (METROPOLIS et al., 1953) e que tem como inspiração o processo de resfriamento de um metal submetido a uma alta temperatura.

Com uma temperatura T_0 inicialmente alta e que vai diminuindo lentamente,

uma solução s sofre alterações ocasionando em $s^{'}$ e acumula uma energia $\Delta = f(s^{'}) - f(s)$, com isso a candidata à solução se torna $s^{'}$ caso $f(s^{'})$ for melhor que $f(s)$ ou com uma probabilidade de $e^{\frac{-\Delta}{T}}$.

Algoritmo 5: Simulated Annealing

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia solução de tamanho m

```
1  solucao ← solucao_inicial();
2  candidato ← solucao;
3  maior_dist ← max_{1≤i≤n}{d(solucao,S_i)};
4  melhor_dist ← maior_dist;
5  T ← T_0;
6  enquanto T > ε faça
7      para cada i ← 1 to m faça
8          posicao ← aleatorio({1,2,3,...,m});
9          id_cadeia ← aleatorio({1,2,3,...,n});
10         vizinho ← candidato;
11         vizinho_posição ← S_{id_cadeia,posição};
12         dist ← max_{1≤i≤n}{d(vizinho, S_i)};
13         Δ ← dist − maior_dist;
14         se Δ ≤ 0 ou aleatorio([0,1]) < e^{-Δ/T} então
15             candidato ← vizinho;
16             maior_dist ← dist;
17             se dist < melhor_dist então
18                 melhor_dist ← dist;
19                 solucao ← vizinho;
20             fim
21         fim
22     fim
23     T ← T * ρ;
24 fim
25 retorna solucao
```

No algoritmo 5, uma string é gerada pelo algoritmo de solução inicial (linha 1), a maior distância de hamming entre a candidata à solução e uma das strings da entrada é encontrada (linha 3), uma posição (*posicao*) entre [1, m] é selecionada aleatoriamente (linha 8), uma posição (*id_cadeia*) entre [1, n] é selecionada aleatoriamente, a string *vizinho* é gerada ao se substituir $candidato_{posicao}$ por $S_{id\_cadeia,posicao}$ (linha 11), um número real entre [0, 1] é gerado (linha 14).

3.2.1 Parâmetros

- $\epsilon = 10^{-3}$ é a menor temperatura aceita.
- $\rho = 0.99$ é o coeficiente de redução da temperatura.
- $T_0 = 500$ é a temperatura inicial do sistema.

3.2.2 Vizinhança

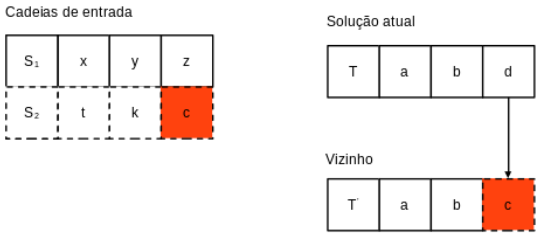

Figura 3.2: vizinhança

Escolhe-se uma string e uma de suas posições de forma aleatória e substitui-se o caractere da solução candidata na, mesma posição, pelo caractere da string selecionada.

3.2.3 Simulated Annealing Paralelo

O Simulated Annealing Paralelo (SAP) executa $NUM\_THREADS$ versões do algoritmo 5 cada uma delas identificada por um id inteiro único entre 1 e $NUM\_THREADS$ e funciona de maneira similar ao serial, contudo em um determinado momento cada thread envia a solução corrente para a de identificador $(id \ mod \ NUM\_THREADS) + 1$.

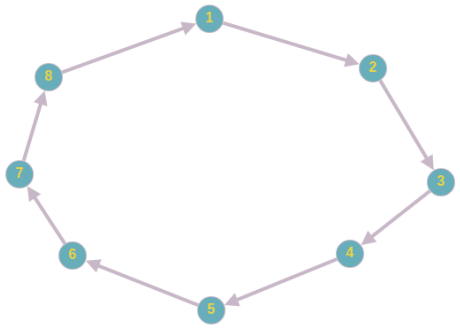

Figura 3.3: 8 threads trocando informação

3.2.3.1 Algoritmo

Algoritmo 6: Simulated Annealing Paralelo

Entrada: sequência S com n strings, comprimento m, alfabeto Σ, identificador id e número de threads $NUM\_THREADS$
Saída: string solução de comprimento m

```
1  solução ← solução_inicial();
2  candidato ← solucao;
3  maior_dist ← max_{1≤i≤n}{d(solucao,S_i)};
4  melhor_dist ← maior_dist;
5  T ← T_0;
6  enquanto T > ϵ faça
7  |  para cada i ← 1 to m faça
8  |  |  realiza movimentos;
9  |  |  critério de aceitação;
10 |  fim
11 |  envia_solucao_atual((id mod NUM_THREADS) + 1);
12 |  se recebeu_solução() então
13 |  |  critério de aceitação;
14 |  fim
15 |  T ← T * ρ;
16 fim
17 retorna solucao
```

3.3 ILS

O Iterated Local Search (ILS) é uma meta-heurística que tem como objetivo escapar de locais mínimos sem comprometer em demasia a qualidade da solução ao se realizar contínuas perturbações e posteriormente aprimorá-las em uma busca local.

Nossa abordagem é inspirada na estratégia de Smart ILS (REINSMA PUCA HUACHI VAZ PENNA, 2018) que propõe que o nível de perturbação cresça lentamente com o intuito de garantir que o espaço seja explorado devidamente, já que uma grande mudança pode levar a soluções distantes rapidamente.

Cada uma das soluções iniciais préviamente propostas 1, 2, 3 e 4 são utilizados para gerar primeira solução ocasionando em 4 versões do ILS e o Simulated Annealing 5 é usado a fim de realizar a busca local.

Algoritmo 7: ILS

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: string solução de comprimento m

```
1  solucao ← busca_local(solucao_inicial());
2  maior_dist ← max_{1≤i≤n}{d(solucao,S_i)};
3  nivel ← 1;
4  vezes ← 1;
5  enquanto i < MAX_ITERACOES faça
6      candidato ← solução;
7      para cada j ← 1 to nivel faça
8          posição ← aleatório({1, 2, 3, ..., m});
9          id_cadeia ← aleatório({1, 2, 3, ..., n});
10         candidato_posição ← S_id_cadeia,posição;
11     fim
12     candidato ← busca_local(candidato);
13     dist ← max_{1≤i≤n}{d(candidato,S_i)};
14     se maior_dist > dist então
15         i ← 0;
16         nivel ← 1;
17         vezes ← 1;
18         maior_dist ← dist;
19         solucao ← candidato;
20     senão
21         i ← i + 1;
22         se vezes < MAX_VEZES então
23             vezes ← vezes + 1;
24         senão
25             nivel ← nivel + 1;
26             vezes ← 1;
27         fim
28     fim
29 fim
30 retorna solucao
```

3.3.1 Parâmetros

- $MAX\_ITERACOES = 100$, indica o número máximo de passos que o algoritmo pode realizar sem aprimorar a solução corrente.
- $MAX\_VEZES = 10$, é a quantidade de iterações que um determinado nível de perturbação é aplicado á solução.

3.3.2 Busca local

O SA em 5 foi ligeiramente modificado para ser a busca local do ILS. A diferença entre os dois algoritmos é que nesta versão a temperatura se torna dinâmica a fim de minimizar a complexidade.

Algoritmo 8: Simulated Annealing

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia solução de comprimento m

```
1  candidato ← solucao;
2  maior_dist ← max_{1≤i≤n}{d(solucao,S_i)};
3  melhor_dist ← maior_dist;
4  T ← temperatura_inicial();
5  enquanto T > ε faça
6      para cada i ← 1 to m * β faça
7          posicao ← aleatorio({1,2,3,...,m});
8          id_cadeia ← aleatorio({1,2,3,...,n});
9          vizinho ← candidato;
10         vizinho_posio ← S_{id_cadeia,posicao};
11         dist ← max_{1≤i≤n}{d̄(vizinho,S_i)};
12         Δ ← dist − maior_dist;
13         se Δ ≤ 0 ou aleatorio([0, 1]) < e^{−Δ/T} então
14             candidato ← vizinho;
15             maior_dist ← dist;
16             se dist < melhor_dist então
17                 melhor_dist ← dist;
18                 solucao ← vizinho;
19             fim
20         fim
21     fim
22     T ← T * ρ;
23 fim
24 retorna solucao
```

3.3.2.1 Parâmetros

- $\rho = 0.99$ é o coeficiente de redução da temperatura.
- $\epsilon = 10^{-2}$ é o limite inferior que a temperatura pode alcançar.
- $\beta = 0.3$ é uma constante para reduzir o número de iterações.

3.3.3 Temperatura inicial

Caso a temperatura seja constante, a escolha de uma que seja agradável para várias instâncias pode ser difícil de encontrar, além disso em um cenário específico o valor pode estar muito acima do ideal. Por isso, um método que forneça a temperatura de acordo com cada instância é sempre uma boa opção e por isso optamos em usar o algoritmo proposto por (SOUZA, 2006) o 9 para realizar essa tarefa.

Algoritmo 9: Temperatura inicial do Simulated Annealing

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia solução de comprimento m

```
1  T ← T_0;
2  enquanto T > ε faça
3  |  aceitos ← 0;
4  |  para cada i ← 1 to m faça
5  |  |  posicao ← aleatorio({1,2,3,...,m});
6  |  |  id_cadeia ← aleatorio({1,2,3,...,n});
7  |  |  vizinho ← candidato;
8  |  |  vizinho_posicao ← S_id_cadeia,posicao;
9  |  |  dist ← max_{1≤i≤n}{d(vizinho, S_i)};
10 |  |  Δ ← dist − maior_dist;
11 |  |  se Δ ≤ 0 ou aleatorio([0,1]) < e^(−Δ/T) então
12 |  |  |  aceitos ← aceitos + 1;
13 |  |  fim
14 |  fim
15 |  se aceitos ≥ ζ * m então
16 |  |  pare;
17 |  fim
18 |  T ← T * α;
19 fim
20 retorna T
```

Dada uma temperatura e uma solução, m vizinhos são gerados (linhas 4-14) e os que são aceitos segundos os critérios do simulated annealing são contabilizados (linhas 11-13), com isso ao final das m iterações caso o percentual seja menor que

95% do total a temperatura aumenta em α (linha 18) e caso contrário é encerrado (linhas 15-17).

3.3.3.1 Parâmetros

- $\alpha = 1.1$ é o percentual de aumento em T.
- $\epsilon = 10^{-2}$ é o limite inferior que a temperatura pode alcançar.
- $\zeta = 0.95$ é o percentual mínimo de soluções aceitas.

3.4 Algoritmo Genético

Os algoritmos genéticos, propostos por John Henry, são meta-heurísticas que se inspiram nas teorias evolucionistas de Darwin e na genética para simular a interação de um conjunto de indivíduos entre si e o ambiente. Que é traduzido pelo algoritmo em uma população, com seus genes representados por strings, de soluções que em cada geração trocam informações através do crossover e modificam os seus genes esporadicamente.

A abordagem utilizada é inspirada no Algoritmo Genético (GA) de (FESTA; PARDALOS, 2012) que consiste em trabalhar com uma população dividida em três camadas a cada geração: a elite que possui as melhores soluções, a mediana que possui as melhores soluções que não estão na elite e que é preenchida por strings obtidas pelo crossover e a baixa que contém a parcela restante composta por strings aleatórias.

Algoritmo 10: Algoritmo Genético

Entrada: sequência S com n strings, comprimento m, alfabeto Σ
Saída: cadeia solução de comprimento m

$T \leftarrow \{\}$;
$k \leftarrow 0$;
$populacao \leftarrow n$;
$n\_elite \leftarrow \lfloor populacao * elite \rfloor$;
$n\_media \leftarrow \lfloor (populacao - n\_elite) * elite \rfloor$;
$n\_baixa \leftarrow populacao - n\_elite - n\_media$;
para cada $i \leftarrow 1$ **to** $populacao$ **faça**
    $t \leftarrow$ solucao_inicial_1();
    $max\_dist \leftarrow max_{1 \leq j \leq n}\{d(t, S_j\}$;
    $T_i \leftarrow (max\_dist, t)$;
fim
$solucao \leftarrow min_{1 \leq i \leq n} T_i$;
enquanto $k < m$ **faça**
    $ind \leftarrow \{\}$;
    $T \leftarrow$ ordena_crescente_pela_distancia(T);
    $T' \leftarrow T$;
    para cada $i \leftarrow 1$ **to** $populacao$ **faça**
        $ind_i \leftarrow$ (aleatorio($\{0, 1, ..., m$ - $T_{i,0}\}$), i);
    fim
    $ind \leftarrow$ ordena_decrescente_pela_distancia(ind);
    para cada $i \leftarrow 1$ **to** n_media **faça**
        $k \leftarrow ind_{(2*i-1) \bmod n + 1,1}$;
        $j \leftarrow ind_{(2*i) \bmod n + 1,1}$;
        $T'_{n\_elite+i,1} \leftarrow$ crossover($T_{k,1}$, $T_{j,1}$);
    fim
    para cada $i \leftarrow 1$ **to** $populacao - n\_baixa$ **faça**
        $T'_{i,1} \leftarrow$ mutacao($T'_{i,1}$);
    fim
    para cada $i \leftarrow populacao - n\_baixa + 1$ **to** $populacao$ **faça**
        $T'_{i,1} \leftarrow$ solucao_inicial_1();
    fim
    para cada $i \leftarrow 1$ **to** $populacao$ **faça**
        $T'_{i,0} \leftarrow max_{1 \leq j \leq n}$ {d($T'_{i,1}$, S_j)};
    fim
    $T \leftarrow T'$;
    $solucao \leftarrow min(solucao, min_{1 \leq j \leq n} T_j)$;
fim
retorna $solucao_1$

3.4.1 Parâmetros

- λ: representa a probabilidade de uma posição da string ser escolhida na mutação e tem valor igual a $\frac{\sqrt{m}}{m}$.
- *populacao*: representa a quantidade de soluções candidatas que são geradas pelo algoritmo e é composto pelo mesmo número de elementos que S possui.
- *elite*: é a porcentagem da população que possui as melhores soluções, tem valor constante de 30%.
- *media*: é a porcentagem da população que possui as soluções medianas e é composta pelo crossover de soluções, tem valor constante de 50%.
- *baixa*: é a porcentagem da população que possui os piores soluções e é composta por soluções aleatórias, tem valor constante de 20%.

3.4.2 Solução inicial

Como o GA é uma meta-heurística populacional é um requisito que sua população seja altamente diversa para que não haja uma estagnação em suas gerações futuras. Com isso, o único método anteriormente proposto que satisfaz essas necessidades é o 1.

3.4.3 Seleção

Cada indivíduo t da geração corrente recebe um número aleatório entre 0 e $max_{1 \leq i \leq n}\{d(t, S_i)\}$ e com isso são ordenados em ordem decrescente por esse valor. Após isso as melhores $2 * media$ soluções formam pares de tal forma que a $1^{\underline{a}}$ pareia com a $2^{\underline{a}}$, a 3 $^{\underline{a}}$ com a $4^{\underline{a}}$ e assim em diante e com isso essas duas strings realizam um crossover.

3.4.4 Crossover

O crossover entre dois indivíduos ao contrário do que é comumente adotado, a divisão da string em duas partes e a intercalação, adotamos uma divisão entre subconjuntos pelo fato dos caracteres serem independentes entre si, ou seja, um

caractere não afeta na contribuição de outro na distância de hamming.

Com isso, o crossover se resume em selecionar um subconjunto de posições de cada string de tal forma que sejam disjuntos e a união seja $\{1, 2, 3, ..., m\}$ e usar os elementos que estão nas posições selecionadas para gerar a nova solução.

Algoritmo 11: Crossover entre duas soluções

Entrada: strings s e t de comprimento m
Saída: string u de comprimento m

```
1 pos ← embaralha({1,2,3,...,m});
2 quantidade ← aleatorio({1,2,3,...,m});
3 u ← t;
4 para cada i ← 1 to quantidade faça
5 |   u_{pos_i} ← s_{pos_i};
6 fim
7 retorna u;
```

em 11 a permutação {1,2,3, ..., m} é embaralhada aleatoriamente e atribuída à *pos* (linha 1), um valor inteiro aleatório entre [1, m] é atribuído à variável *quantidade* (linha 2), u é inicializado como t (linha 3) e *quantidade* posições são atribuidas à u provenientes de s.

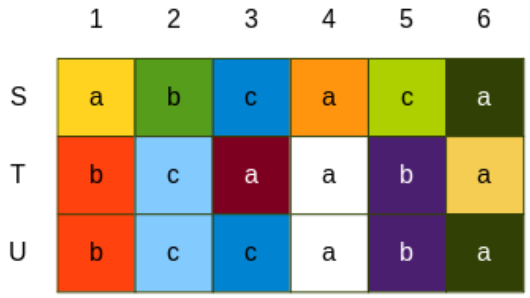

Figura 3.4: Crossover

Na figura 3.4 o crossover entre soluções gera uma nova candidata mesclando, aleatoriamente, as posições entre as strings.

3.4.5 Mutação

A mutação utilizada foi proposta em (COLEY, 1999) e ocorre individualmente em cada posição da string de acordo com uma probabilidade $\lambda = \frac{\sqrt{m}}{m}$ baixa e constante

e caracteriza-se como uma mudança entre caracteres feita de forma aleatória.

Algoritmo 12: Mutação na solução

Entrada: string s de comprimento m
Saída: string s de comprimento m

```
1  para cada j ← 1 to m faça
2  |  Σ' ← {};
3  |  para cada i ← 1 to n faça
4  |  |  Σ' ← Σ' ∪ {S_{i,j}};
5  |  fim
6  |  se aleatorio([0, 1]) < λ então
7  |  |  s_j ← aleatorio(Σ');
8  |  fim
9  fim
10 retorna s;
```

Em 12 um valor real aleatório entre [0, 1] é selecionado (linha 2), S_i recebe um caractere aleatório de $\Sigma^{'}$ que é a coleção de todos os caracteres da coluna j (linha 3).

Capítulo 4

Experimentos

4.1 Testes

Os testes foram realizados em um computador rodando o Ubuntu 18.04.3 LTS, com um processador Intel Core i7-3770 com CPU 3.40GHz e memória de 8Gb, a linguagem de programação utilizada foi a C++ versão 17, o compilador foi o g++ 7.4.0 e as flags de compilação usadas foram: -O3, -funroll-loops, -march=native, -mtune=native, -lpthread.

Cada instância foi executada três vezes para se obter a média aritmética dos resultados e dos tempos.

O projeto e as instâncias estão disponíveis em: <https://github.com/ThiagoFBastos/Closest-String-Problem>

4.2 Instâncias da literatura

As instâncias utilizadas foram utilizadas no artigo de (CHIMANI WOSTE, 2011) em que trabalha-se com um conjunto de três instâncias: McClure, Hufsky e as aleatória geradas pelo o próprio trabalho e apresenta resultados obtidos por algoritmos ILP que estão caracterizados na tabela como lower bound (lb) e upper bound (ub).

4.2.1 Instâncias Mcclure

As instâncias Mcclure (MCCLURE MARCELLA; VASI; FITCH WALTER, 1994) possuem um padrão de nome McClure-p-|Σ|-n-m.csp, em que p é a página do jornal em que foi publicado o artigo, Σ o alfabeto, n o número de strings da entrada e m é o comprimento das strings.

Tabela 4.1: Instâncias mcclure

instância	lb	ub
McClure-582-20-10-141.csp	97.000	97.000
McClure-582-20-12-141.csp	97.000	97.000
McClure-582-20-6-141.csp	88.000	88.000
McClure-586-20-10-98.csp	75.000	75.000
McClure-586-20-12-98.csp	76.000	76.000
McClure-586-20-6-100.csp	72.000	77.000

Instâncias Mcclure com o limite inferior da resposta lb e o limite superior ub.

4.2.2 Instâncias Hufsky

As instâncias Hufsky (HUFSKY et al., 2010) possuem um padrão de nome Hufsky-n-m-i.csp, em que n é o número de strings da entrada, m o comprimento das strings e i é um número que identifica a instância.

Tabela 4.2: Instâncias Hufsky

instância	*lb*	*ub*
Hufsky-30-250-0.csp	27.000	27.000
Hufsky-50-250-12.csp	30.000	30.000
Hufsky-40-250-25.csp	24.000	24.000
Hufsky-40-300-1.csp	29.000	29.000
Hufsky-30-300-16.csp	30.000	30.000
Hufsky-30-300-35.csp	30.000	30.000
Hufsky-30-350-28.csp	36.000	36.000
Hufsky-40-350-4.csp	38.000	38.000
Hufsky-50-350-33.csp	38.000	38.000
Hufsky-50-400-2.csp	40.000	40.000
Hufsky-30-400-24.csp	36.000	36.000
Hufsky-20-400-19.csp	42.000	42.000
Hufsky-40-450-7.csp	50.000	50.000
Hufsky-20-450-3.csp	47.000	47.000
Hufsky-30-450-4.csp	46.000	46.000
Hufsky-30-500-25.csp	44.000	44.000
Hufsky-50-500-12.csp	49.000	49.000
Hufsky-40-500-20.csp	54.000	54.000

Instâncias Hufsky com o limite inferior da resposta lb e o limite superior ub.

4.2.3 Instâncias Chimani

Tabela 4.3: Instâncias chimani

instância	*lb*	*ub*
20-50-250-3-5.csp	83.000	83.000
2-50-250-5-7.csp	50.000	50.000
2-30-250-2-5.csp	108.000	108.000
2-50-500-5-8.csp	100.000	100.000
2-10-500-3-5.csp	157.000	157.000
4-20-500-1-5.csp	317.000	317.000
2-30-750-4-5.csp	187.000	187.000
4-10-750-2-8.csp	358.000	358.000
20-30-750-3-0.csp	250.000	250.000
20-50-1000-4-9.csp	250.000	250.000
4-50-1000-4-4.csp	250.000	250.000
4-20-1000-2-4.csp	497.000	497.000
4-40-2000-2-2.csp	1000.000	1000.000
4-40-2000-5-0.csp	400.000	400.000
4-40-2000-5-1.csp	400.000	400.000
2-50-5000-4-5.csp	1250.000	1250.000
2-30-5000-2-5.csp	2136.000	2136.000
20-40-5000-5-6.csp	1000.000	1000.000
4-20-10000-2-6.csp	4960.000	4960.000
2-40-10000-3-2.csp	3327.000	3327.000
20-30-10000-4-8.csp	2500.000	2500.000

Instâncias Chimani com o limite inferior da resposta lb e o limite superior ub.

4.3 Simulated Annealing

Nesta seção exibimos os testes realizados sobre as instâncias com o SA usando os algoritmos solução inicial: 1 (sa_1), 2 (sa_2), 3 (sa_3) e 4 (sa_4).

4.3.1 Instâncias Mcclure

Tabela 4.4: Instâncias Mcclure - Respostas

instância	sa_1	sa_2	sa_3	sa_4	lb	ub	$ER\%$	σ
McClure-582-20-10-141.csp	**101.000**	**101.000**	101.333	101.333	97.000	97.000	4.124	0.167
McClure-582-20-12-141.csp	101.333	101.667	**101.000**	101.667	97.000	97.000	4.124	0.276
McClure-582-20-6-141.csp	91.667	91.667	**91.333**	**91.333**	88.000	88.000	3.788	0.167
McClure-586-20-10-98.csp	**79.333**	**79.333**	79.667	**79.333**	75.000	75.000	5.778	0.144
McClure-586-20-12-98.csp	**80.667**	81.000	81.000	81.000	76.000	76.000	6.140	0.144
McClure-586-20-6-100.csp	**74.000**	74.333	74.333	74.333	72.000	77.000	2.778	0.144

A tabela 4.4 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 1, a coluna sa_2 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 2, a coluna sa_3 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 3, a coluna sa_4 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do SA e a coluna σ com o desvio padrão entre as respostas das versões do SA.

Tabela 4.5: Instâncias Mcclure - Tempo de execução

instância	sa_1	sa_2	sa_3	sa_4
McClure-582-20-10-141.csp	0.0113016	0.0110665	**0.0110574**	0.0111395
McClure-582-20-12-141.csp	0.0115415	0.0137380	**0.0114596**	0.0115355
McClure-582-20-6-141.csp	0.0099606	0.0099889	**0.0098807**	0.0100720
McClure-586-20-10-98.csp	0.0076344	0.0076054	**0.0075870**	0.0076630
McClure-586-20-12-98.csp	0.0079171	0.0079612	**0.0079112**	0.0079702
McClure-586-20-6-100.csp	**0.0070076**	0.0070433	0.0070493	0.0071788

A tabela 4.5 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 1 sobre a instância, a coluna sa_2 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 2 sobre a instância, a coluna sa_3 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 3 sobre a instância e a coluna sa_4 que

contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 4 sobre a instância.

4.3.2 Instâncias Hufsky

Tabela 4.6: Instâncias Hufsky - Respostas

instância	sa_1	sa_2	sa_3	sa_4	lb	ub	$ER\%$	σ
Hufsky-30-250-0.csp	**27.333**	**27.333**	27.667	**27.333**	27.000	27.0	1.235	0.144
*Hufsky-50-250-12.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.0	0.000	0.000
*Hufsky-40-250-25.csp	**24.000**	**24.000**	**24.000**	**24.000**	24.000	24.0	0.000	0.000
*Hufsky-40-300-1.csp	**29.000**	**29.000**	**29.000**	**29.000**	29.000	29.0	0.000	0.000
*Hufsky-30-300-16.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.0	0.000	0.000
*Hufsky-30-300-35.csp	**30.000**	30.667	**30.000**	**30.000**	30.000	30.0	0.000	0.289
*Hufsky-30-350-28.csp	36.333	36.667	**36.000**	**36.000**	36.000	36.0	0.000	0.276
*Hufsky-40-350-4.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.0	0.000	0.000
*Hufsky-50-350-33.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.0	0.000	0.000
*Hufsky-50-400-2.csp	**40.000**	**40.000**	**40.000**	40.667	40.000	40.0	0.000	0.289
*Hufsky-30-400-24.csp	**36.000**	36.333	36.667	36.333	36.000	36.0	0.000	0.236
*Hufsky-20-400-19.csp	**42.000**	**42.000**	**42.000**	**42.000**	42.000	42.0	0.000	0.000
*Hufsky-40-450-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.0	0.000	0.000
Hufsky-20-450-3.csp	**47.333**	**47.333**	**47.333**	**47.333**	47.000	47.0	0.709	0.000
*Hufsky-30-450-4.csp	46.333	**46.000**	46.333	**46.000**	46.000	46.0	0.000	0.167
Hufsky-30-500-25.csp	**45.000**	**45.000**	45.333	**45.000**	44.000	44.0	2.273	0.144
*Hufsky-50-500-12.csp	**49.000**	49.333	**49.000**	49.667	49.000	49.0	0.000	0.276
*Hufsky-40-500-20.csp	54.333	54.667	54.333	**54.000**	54.000	54.0	0.000	0.236

A tabela 4.6 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 1, a coluna sa_2 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 2, a coluna sa_3 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 3, a coluna sa_4 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do

SA e a coluna σ com o desvio padrão entre as respostas das versões do SA. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.7: Instâncias Hufsky - Tempo de execução

instância	sa_1	sa_2	sa_3	sa_4
Hufsky-30-250-0.csp	0.0258946	0.0261186	**0.0256486**	0.0303428
Hufsky-50-250-12.csp	0.0298701	0.0301887	**0.0298700**	0.0300117
Hufsky-40-250-25.csp	0.0276565	0.0277259	**0.0275322**	0.0276816
Hufsky-40-300-1.csp	**0.0331615**	0.0334603	0.0332086	0.0332406
Hufsky-30-300-16.csp	0.0309524	0.0308443	**0.0306668**	0.0310366
Hufsky-30-300-35.csp	0.0308935	0.0308950	**0.0307928**	0.0310445
Hufsky-30-350-28.csp	**0.0360291**	0.0362311	0.0360582	0.0364421
Hufsky-40-350-4.csp	0.0387975	0.0387839	**0.0387287**	0.0388736
Hufsky-50-350-33.csp	**0.0419334**	0.0419463	0.0420559	0.0421862
Hufsky-50-400-2.csp	0.0479579	**0.0479380**	0.0481102	0.0495698
Hufsky-30-400-24.csp	**0.0412138**	0.0412269	0.0412244	0.0421934
Hufsky-20-400-19.csp	0.0316900	0.0318831	**0.0315902**	0.0321423
Hufsky-40-450-7.csp	0.0502597	0.0501445	**0.0499540**	0.0514325
Hufsky-20-450-3.csp	0.0358649	0.0358267	**0.0357727**	0.0365188
Hufsky-30-450-4.csp	0.0464562	0.0466440	**0.0458140**	0.0465943
Hufsky-30-500-25.csp	0.0570551	0.0516935	**0.0508910**	0.0517244
Hufsky-50-500-12.csp	0.0603314	0.0604139	**0.0596933**	0.0608342
Hufsky-40-500-20.csp	0.0557678	0.0559631	**0.0552084**	0.0556065

A tabela 4.7 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 1 sobre a instância, a coluna sa_2 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 2 sobre a instância, a coluna sa_3 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 3 sobre a instância e a coluna sa_4 que

contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 4 sobre a instância.

4.3.3 Instâncias Chimani

Tabela 4.8: Instâncias Chimani - Respostas

instância	sa_1	sa_2	sa_3	sa_4	lb	ub	ER	σ
*20-50-250-3-5.csp	**83.000**	**83.000**	**83.000**	**83.000**	83.000	83.0	0.000	0.000
*2-50-250-5-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.0	0.000	0.000
2-30-250-2-5.csp	116.000	116.000	116.333	**115.667**	108.000	108.0	7.099	0.236
*2-50-500-5-8.csp	103.000	**100.000**	**100.000**	101.667	100.000	100.0	0.000	1.258
2-10-500-3-5.csp	166.333	**164.000**	167.333	166.667	157.000	157.0	4.459	1.256
4-20-500-1-5.csp	331.000	331.000	**329.667**	330.667	317.000	317.0	3.996	0.546
2-30-750-4-5.csp	199.000	**188.333**	198.667	199.667	187.000	187.0	0.713	4.681
4-10-750-2-8.csp	370.667	**368.333**	369.667	369.000	358.000	358.0	2.886	0.862
*20-30-750-3-0.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.0	0.000	0.000
*20-50-1000-4-9.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.0	0.000	0.000
*4-50-1000-4-4.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.0	0.000	0.000
4-20-1000-2-4.csp	510.667	**508.667**	509.333	509.000	497.000	497.0	2.347	0.759
4-40-2000-2-2.csp	1024.667	**1006.333**	1029.667	1024.667	1000.000	1000.0	0.633	8.898
*4-40-2000-5-0.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.0	0.000	0.000
*4-40-2000-5-1.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.0	0.000	0.000
2-50-5000-4-5.csp	1340.333	**1250.667**	1335.333	1336.333	1250.000	1250.0	0.053	37.574
2-30-5000-2-5.csp	2289.667	**2204.000**	2285.667	2284.333	2136.000	2136.0	3.184	35.801
*20-40-5000-5-6.csp	**1000.000**	**1000.000**	**1000.000**	**1000.000**	1000.000	1000.0	0.000	0.000
4-20-10000-2-6.csp	5087.667	**4989.000**	5090.333	5090.667	4960.000	4960.0	0.585	43.557
2-40-10000-3-2.csp	3612.667	**3350.667**	3608.667	3611.667	3327.000	3327.0	0.711	112.737
*20-30-10000-4-8.csp	**2500.000**	**2500.000**	**2500.000**	**2500.000**	2500.000	2500.0	0.000	0.000

A tabela 4.8 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 1, a coluna sa_2 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 2, a coluna sa_3 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 3, a coluna sa_4 que contém o valor encontrado pelo SA usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do

SA e a coluna σ com o desvio padrão entre as respostas das versões do SA. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.9: Instâncias Chimani - Tempo de execução

instância	sa_1	sa_2	sa_3	sa_4
20-50-250-3-5.csp	0.0298808	0.0301319	**0.0297255**	0.0303122
2-50-250-5-7.csp	**0.0300669**	0.0301165	0.0300699	0.0317519
2-30-250-2-5.csp	0.0270367	0.0270306	**0.0268579**	0.0274246
2-50-500-5-8.csp	0.0605350	0.0610886	**0.0601147**	0.0610313
2-10-500-3-5.csp	0.0383930	0.0391153	**0.0383869**	0.0384686
4-20-500-1-5.csp	0.0428100	0.0432778	**0.0425932**	0.0432729
2-30-750-4-5.csp	0.0799509	0.0811420	**0.0794311**	0.0807950
4-10-750-2-8.csp	0.0587245	0.0608833	**0.0583105**	0.0644082
20-30-750-3-0.csp	0.0795779	0.0866238	**0.0791281**	0.0808207
20-50-1000-4-9.csp	0.1212607	0.1326939	**0.1203236**	0.1214208
4-50-1000-4-4.csp	0.1217383	0.1207370	**0.1204110**	0.1234542
4-20-1000-2-4.csp	0.0865790	0.0890619	**0.0855999**	0.0886612
4-40-2000-2-2.csp	0.2304503	0.2320211	**0.2286775**	0.2314127
4-40-2000-5-0.csp	0.2292698	0.2331635	**0.2280634**	0.2325096
4-40-2000-5-1.csp	**0.2286320**	0.2344150	0.2308694	0.2295755
2-50-5000-4-5.csp	**0.6477175**	0.6561502	0.6491384	0.6496024
2-30-5000-2-5.csp	0.5761071	0.5791858	**0.5743028**	0.5801621
20-40-5000-5-6.csp	0.6052132	0.6118199	**0.6028082**	0.6043198
4-20-10000-2-6.csp	0.9679311	0.9768637	**0.9610051**	0.9758389
2-40-10000-3-2.csp	1.2682653	1.2736829	**1.2672072**	1.2787791
20-30-10000-4-8.csp	**1.1655843**	1.1720967	1.1666275	1.1739950

A tabela 4.9 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sa_1 que contém o tempo em segundos da execução

do SA usando o algoritmo de solução inicial 1 sobre a instância, a coluna sa_2 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 2 sobre a instância, a coluna sa_3 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 3 sobre a instância e a coluna sa_4 que contém o tempo em segundos da execução do SA usando o algoritmo de solução inicial 4 sobre a instância.

4.3.4 Simulated Annealing Paralelo

Nesta seção exibimos os testes realizados sobre as instâncias com o SAP usando os algoritmos de solução inicial: 1 (sap_1), 2 (sap_2), 3 (sap_3) e 4 (sap_4) com $NUM\_THREADS = 8$.

4.3.4.1 Mcclure

Tabela 4.10: Instâncias Mcclure - Respostas

instância	sap_1	sap_2	sap_3	sap_4	lb	ub	$ER\%$	σ
McClure-582-20-10-141.csp	100.333	100.667	**100.000**	**100.000**	97.000	97.0	3.093	0.276
McClure-582-20-12-141.csp	100.333	100.333	100.333	**100.000**	97.000	97.0	3.093	0.144
McClure-582-20-6-141.csp	90.667	90.667	**90.333**	91.333	88.000	88.0	2.652	0.363
McClure-586-20-10-98.csp	78.667	**78.000**	78.667	79.000	75.000	75.0	4.000	0.363
McClure-586-20-12-98.csp	80.000	**79.667**	80.000	80.000	76.000	76.0	4.825	0.144
McClure-586-20-6-100.csp	**73.333**	**73.333**	73.667	**73.333**	72.000	77.0	1.852	0.144

A tabela 4.10 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 1, a coluna sap_2 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 2, a coluna sap_3 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 3, a coluna sap_4 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do SAP e a coluna σ com o desvio padrão entre as respostas das versões do SAP.

Tabela 4.11: Instâncias Mcclure - Tempo de execução

instância	sap_1	sap_2	sap_3	sap_4
McClure-582-20-10-141.csp	0.0556148	0.0533406	**0.0404300**	0.0406899
McClure-582-20-12-141.csp	0.0545150	0.0532483	**0.0412410**	0.0433831
McClure-582-20-6-141.csp	0.0510945	0.0525831	0.0380217	**0.0379292**
McClure-586-20-10-98.csp	0.0372250	0.0369256	**0.0278991**	0.0280586
McClure-586-20-12-98.csp	0.0371769	0.0393188	0.0297388	**0.0292034**
McClure-586-20-6-100.csp	0.0376788	0.0275086	**0.0273414**	0.0275086

A tabela 4.11 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 1 sobre a instância, a coluna sap_2 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 2 sobre a instância, a coluna sap_3 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 3 sobre a instância e a coluna sap_4 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 4 sobre a instância.

4.3.4.2 Hufsky

Tabela 4.12: Instâncias Hufsky - Respostas

instância	sap_1	sap_2	sap_3	sap_4	lb	ub	$ER\%$	σ
*Hufsky-30-250-0.csp	**27.000**	**27.000**	**27.000**	**27.000**	27.000	27.000	0.000	0.000
*Hufsky-50-250-12.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-40-250-25.csp	**24.000**	**24.000**	**24.000**	**24.000**	24.000	24.000	0.000	0.000
*Hufsky-40-300-1.csp	**29.000**	**29.000**	**29.000**	**29.000**	29.000	29.000	0.000	0.000
*Hufsky-30-300-16.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-30-300-35.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-30-350-28.csp	**36.000**	**36.000**	**36.000**	**36.000**	36.000	36.000	0.000	0.000
*Hufsky-40-350-4.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.000	0.000	0.000
*Hufsky-50-350-33.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.000	0.000	0.000
*Hufsky-50-400-2.csp	**40.000**	**40.000**	**40.000**	**40.000**	40.000	40.000	0.000	0.000
*Hufsky-30-400-24.csp	**36.000**	**36.000**	**36.000**	**36.000**	36.000	36.000	0.000	0.000
*Hufsky-20-400-19.csp	**42.000**	**42.000**	**42.000**	**42.000**	42.000	42.000	0.000	0.000
*Hufsky-40-450-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.000	0.000	0.000
*Hufsky-20-450-3.csp	**47.000**	**47.000**	**47.000**	**47.000**	47.000	47.000	0.000	0.000
*Hufsky-30-450-4.csp	**46.000**	**46.000**	**46.000**	**46.000**	46.000	46.000	0.000	0.000
*Hufsky-30-500-25.csp	**44.000**	44.333	44.333	44.667	44.000	44.000	0.000	0.236
*Hufsky-50-500-12.csp	**49.000**	**49.000**	**49.000**	**49.000**	49.000	49.000	0.000	0.000
*Hufsky-40-500-20.csp	**54.000**	**54.000**	54.333	**54.000**	54.000	54.000	0.000	0.144

A tabela 4.12 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 1, a coluna sap_2 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 2, a coluna sap_3 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 3, a coluna sap_4 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do SAP e a coluna σ com o desvio padrão entre as respostas das versões do SAP. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.13: Instâncias Hufsky - Tempo de execução

instância	sap_1	sap_2	sap_3	sap_4
Hufsky-30-250-0.csp	0.0973859	0.0942833	0.0789745	**0.0770080**
Hufsky-50-250-12.csp	0.1039809	0.1032635	0.0927869	**0.0890228**
Hufsky-40-250-25.csp	0.1030702	0.0978809	0.0891456	**0.0875680**
Hufsky-40-300-1.csp	0.1138784	0.1152578	0.1010777	**0.1000626**
Hufsky-30-300-16.csp	0.1154104	0.1144481	0.0994590	**0.0958930**
Hufsky-30-300-35.csp	0.1131289	0.1130004	0.0991728	**0.0973637**
Hufsky-30-350-28.csp	0.1333509	0.1338981	**0.1123771**	0.1152560
Hufsky-40-350-4.csp	0.1407396	0.1324677	**0.1169882**	0.1218763
Hufsky-50-350-33.csp	0.1481965	0.1420124	0.1301174	**0.1265377**
Hufsky-50-400-2.csp	0.1663657	0.1632584	**0.1430618**	0.1454156
Hufsky-30-400-24.csp	0.1487933	0.1462703	**0.1230851**	0.1270625
Hufsky-20-400-19.csp	0.1389608	0.1384533	**0.1091865**	0.1105483
Hufsky-40-450-7.csp	0.1748479	0.1767382	**0.1485881**	0.1522861
Hufsky-20-450-3.csp	0.1538108	0.1528018	0.1236722	**0.1214285**
Hufsky-30-450-4.csp	0.1696484	0.1719913	**0.1392272**	0.1445916
Hufsky-30-500-25.csp	0.1907305	0.1822294	0.1608924	**0.1567194**
Hufsky-50-500-12.csp	0.1990460	0.1961464	**0.1725636**	0.1792651
Hufsky-40-500-20.csp	0.1929209	0.1948891	0.1688838	**0.1640891**

A tabela 4.13 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 1 sobre a instância, a coluna sap_2 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 2 sobre a instância, a coluna sap_3 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 3 sobre a instância e a coluna sap_4 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 4 sobre a instância.

4.3.4.3 Chimani

Tabela 4.14: Instâncias Chimani - Respostas

instância	sap_1	sap_2	sap_3	sap_4	lb	ub	$ER\%$	σ
*20-50-250-3-5.csp	**83.000**	**83.000**	**83.000**	**83.000**	83.000	83.000	0.000	0.000
*2-50-250-5-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.000	0.000	0.000
2-30-250-2-5.csp	**114.667**	115.000	115.333	115.667	108.000	108.000	6.173	0.373
*2-50-500-5-8.csp	**100.000**	**100.000**	**100.000**	**100.000**	100.000	100.000	0.000	0.000
2-10-500-3-5.csp	**164.000**	164.333	**164.000**	**164.000**	157.000	157.000	4.459	0.144
4-20-500-1-5.csp	**327.333**	328.000	328.333	328.333	317.000	317.000	3.260	0.408
2-30-750-4-5.csp	196.333	**188.333**	195.000	196.000	187.000	187.000	0.713	3.261
4-10-750-2-8.csp	367.000	**365.333**	366.333	367.333	358.000	358.000	2.048	0.764
*20-30-750-3-0.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
*20-50-1000-4-9.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
*4-50-1000-4-4.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
4-20-1000-2-4.csp	507.333	**507.000**	**507.000**	507.333	497.000	497.000	2.012	0.167
4-40-2000-2-2.csp	1018.000	**1005.000**	1021.667	1018.667	1000.000	1000.000	0.500	6.405
*4-40-2000-5-0.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.000	0.000	0.000
*4-40-2000-5-1.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.000	0.000	0.000
2-50-5000-4-5.csp	1310.667	**1250.000**	1313.333	1309.333	1250.000	1250.000	0.000	26.501
2-30-5000-2-5.csp	2271.667	**2187.333**	2268.333	2274.000	2136.000	2136.000	2.403	36.429
*20-40-5000-5-6.csp	**1000.000**	**1000.000**	**1000.000**	**1000.000**	1000.000	1000.000	0.000	0.000
4-20-10000-2-6.csp	5064.333	**4978.333**	5063.667	5065.333	4960.000	4960.000	0.370	37.292
2-40-10000-3-2.csp	3564.000	**3347.667**	3572.000	3566.667	3327.000	3327.000	0.621	95.258
*20-30-10000-4-8.csp	**2500.000**	**2500.000**	**2500.000**	**2500.000**	2500.000	2500.000	0.000	0.000

A tabela 4.14 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 1, a coluna sap_2 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 2, a coluna sap_3 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 3, a coluna sap_4 que contém o valor encontrado pelo SAP usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do SAP e a coluna σ com o desvio padrão entre as respostas das versões do SAP. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.15: Instâncias Chimani - Tempo de execução

instância	sap_1	sap_2	sap_3	sap_4
20-50-250-3-5.csp	0.1104805	0.1080763	0.0945350	**0.0923067**
2-50-250-5-7.csp	0.1077881	0.1078395	0.0924257	**0.0920405**
2-30-250-2-5.csp	0.1063342	0.1037945	**0.0882581**	0.0882862
2-50-500-5-8.csp	0.2112819	0.2131016	**0.1822495**	0.1824872
2-10-500-3-5.csp	0.1804389	0.1774246	0.1364063	**0.1337936**
4-20-500-1-5.csp	0.1935495	0.1888295	**0.1497651**	0.1501983
2-30-750-4-5.csp	0.2964913	0.2963312	**0.2477085**	0.2494484
4-10-750-2-8.csp	0.2696472	0.2751258	0.2051902	**0.2003384**
20-30-750-3-0.csp	0.2947495	0.2936942	0.2478850	**0.2460059**
20-50-1000-4-9.csp	0.4094509	0.4061589	0.3545075	**0.3530020**
4-50-1000-4-4.csp	0.4038036	0.4173318	0.3514387	**0.3512115**
4-20-1000-2-4.csp	0.3721831	0.3692764	**0.2889038**	0.2933672
4-40-2000-2-2.csp	0.8172393	0.8099725	**0.6720557**	0.6821544
4-40-2000-5-0.csp	0.7718842	0.7632208	**0.6423539**	0.6618025
4-40-2000-5-1.csp	0.7790213	0.7768188	**0.6481500**	0.6488791
2-50-5000-4-5.csp	2.0137676	2.0197551	**1.7230834**	1.7501691
2-30-5000-2-5.csp	2.0076625	1.9857979	1.6195047	**1.6031866**
20-40-5000-5-6.csp	1.9399607	1.9554308	1.6547025	**1.6234552**
4-20-10000-2-6.csp	3.9074330	3.9148180	3.0194407	**3.0115460**
2-40-10000-3-2.csp	4.4136266	4.3532037	3.7896223	**3.6740887**
20-30-10000-4-8.csp	**3.0220315**	3.9820183	3.3388761	3.2272939

A tabela 4.15 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna sap_1 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 1 sobre a instância, a coluna sap_2 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 2 sobre a instância, a coluna sap_3 que contém o tempo em segundos da execução do SAP usando o algoritmo de solução inicial 3 sobre a instância e a coluna sap_4 que contém o tempo em segundos da execução do SAP usando o algoritmo de

solução inicial 4 sobre a instância.

4.4 ILS

Nesta seção exibimos os testes realizados sobre as instâncias com o ILS usando os algoritmos de solução inicial: 1 (ils_1), 2 (ils_2), 3 (ils_3) e 4 (ils_4).

4.4.1 Instâncias Mcclure

Tabela 4.16: Instâncias Mcclure - Respostas

instância	ils_1	ils_2	ils_3	ils_4	lb	ub	$ER\%$	σ
McClure-582-20-10-141.csp	101.000	**100.000**	**100.000**	**100.000**	97.000	97.000	3.093	0.433
McClure-582-20-12-141.csp	101.000	100.333	**100.000**	**100.000**	97.000	97.000	3.093	0.408
McClure-582-20-6-141.csp	91.000	91.000	**90.333**	**90.333**	88.000	88.000	2.652	0.333
McClure-586-20-10-98.csp	79.000	**77.667**	78.333	78.000	75.000	75.000	3.556	0.493
McClure-586-20-12-98.csp	80.667	**79.667**	80.000	80.000	76.000	76.000	4.825	0.363
McClure-586-20-6-100.csp	73.667	**73.000**	**73.000**	**73.000**	72.000	77.000	1.389	0.289

A tabela 4.16 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 1, a coluna ils_2 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 2, a coluna ils_3 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 3, a coluna ils_4 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do ILS e a coluna σ com o desvio padrão entre as respostas das versões do ILS. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.17: Instâncias Mcclure - Tempo de execução

instância	ils_1	ils_2	ils_3	ils_4
McClure-582-20-10-141.csp	**0.1683167**	0.7270298	0.6960941	0.7377443
McClure-582-20-12-141.csp	**0.2242524**	0.5960839	0.8021944	0.7406816
McClure-582-20-6-141.csp	**0.1953785**	0.5100284	0.6592149	0.7097359
McClure-586-20-10-98.csp	**0.1134596**	0.4412381	0.4366631	0.4010897
McClure-586-20-12-98.csp	**0.1364169**	0.3913550	0.3961606	0.3982533
McClure-586-20-6-100.csp	**0.1292943**	0.4195519	0.4256844	0.4166404

A tabela 4.17 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 1 sobre a instância, a coluna ils_2 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 2 sobre a instância, a coluna ils_3 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 3 sobre a instância e a coluna ils_4 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 4 sobre a instância.

4.4.2 Instâncias Hufsky

Tabela 4.18: Instâncias Hufsky - Respostas

instância	ils_1	ils_2	ils_3	ils_4	lb	ub	$ER\%$	σ
*Hufsky-30-250-0.csp	**27.000**	**27.000**	**27.000**	**27.000**	27.000	27.000	0.000	0.000
*Hufsky-50-250-12.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-40-250-25.csp	**24.000**	**24.000**	**24.000**	**24.000**	24.000	24.000	0.000	0.000
*Hufsky-40-300-1.csp	**29.000**	**29.000**	**29.000**	**29.000**	29.000	29.000	0.000	0.000
*Hufsky-30-300-16.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-30-300-35.csp	**30.000**	**30.000**	**30.000**	**30.000**	30.000	30.000	0.000	0.000
*Hufsky-30-350-28.csp	**36.000**	**36.000**	**36.000**	**36.000**	36.000	36.000	0.000	0.000
*Hufsky-40-350-4.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.000	0.000	0.000
*Hufsky-50-350-33.csp	**38.000**	**38.000**	**38.000**	**38.000**	38.000	38.000	0.000	0.000
*Hufsky-50-400-2.csp	**40.000**	**40.000**	**40.000**	**40.000**	40.000	40.000	0.000	0.000
*Hufsky-30-400-24.csp	**36.000**	**36.000**	**36.000**	**36.000**	36.000	36.000	0.000	0.000
*Hufsky-20-400-19.csp	**42.000**	**42.000**	**42.000**	**42.000**	42.000	42.000	0.000	0.000
*Hufsky-40-450-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.000	0.000	0.000
*Hufsky-20-450-3.csp	**47.000**	**47.000**	**47.000**	**47.000**	47.000	47.000	0.000	0.000
*Hufsky-30-450-4.csp	**46.000**	**46.000**	**46.000**	**46.000**	46.000	46.000	0.000	0.000
*Hufsky-30-500-25.csp	**44.000**	**44.000**	**44.000**	**44.000**	44.000	44.000	0.000	0.000
*Hufsky-50-500-12.csp	**49.000**	**49.000**	**49.000**	**49.000**	49.000	49.000	0.000	0.000
*Hufsky-40-500-20.csp	**54.000**	**54.000**	**54.000**	**54.000**	54.000	54.000	0.000	0.000

A tabela 4.18 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 1, a coluna ils_2 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 2, a coluna ils_3 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 3, a coluna ils_4 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do ILS e a coluna σ com o desvio padrão entre as respostas das versões do ILS. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.19: Instâncias Hufsky - Tempo de execução

instância	ils_1	ils_2	ils_3	ils_4
Hufsky-30-250-0.csp	**0.3017282**	0.9571239	0.9797029	0.9882600
Hufsky-50-250-12.csp	**0.3556066**	1.1530398	1.1638598	1.1731155
Hufsky-40-250-25.csp	**0.3233834**	1.0352402	1.0581029	1.0482474
Hufsky-40-300-1.csp	**0.4011730**	1.2309708	1.2816911	1.2759129
Hufsky-30-300-16.csp	**0.3633931**	1.1592750	1.1818844	1.1999242
Hufsky-30-300-35.csp	**0.3607466**	1.1511153	1.1847542	1.1767374
Hufsky-30-350-28.csp	**0.4340814**	1.3545910	1.3771626	1.3742904
Hufsky-40-350-4.csp	**0.4519801**	1.4424213	1.4729192	1.4677520
Hufsky-50-350-33.csp	**0.4961953**	1.6070410	1.6381127	1.6399103
Hufsky-50-400-2.csp	**0.5866695**	1.8326568	1.8673882	1.8769658
Hufsky-30-400-24.csp	**0.4905159**	1.5469891	1.5717032	1.5843602
Hufsky-20-400-19.csp	**0.4215230**	1.3315531	1.3731403	1.3707236
Hufsky-40-450-7.csp	**0.5903293**	1.8676401	1.8973861	1.9016295
Hufsky-20-450-3.csp	**0.4974275**	1.5195491	1.5744058	1.5626015
Hufsky-30-450-4.csp	**0.5617291**	1.7488795	1.7740260	1.7869172
Hufsky-30-500-25.csp	**0.9980077**	1.9772462	2.3502975	2.2116359
Hufsky-50-500-12.csp	**0.7167379**	2.2994829	2.3565630	2.3740874
Hufsky-40-500-20.csp	**0.6577525**	2.1240671	2.1269353	2.1876092

A tabela 4.19 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 1 sobre a instância, a coluna ils_2 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 2 sobre a instância, a coluna ils_3 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 3 sobre a instância e a coluna ils_4 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 4 sobre a instância.

4.4.3 Instâncias Chimani

Tabela 4.20: Instâncias Chimani - Respostas

instância	ils_1	ils_2	ils_3	ils_4	lb	ub	$ER\%$	σ
*20-50-250-3-5.csp	**83.000**	**83.000**	**83.000**	**83.000**	83.000	83.000	0.000	0.000
*2-50-250-5-7.csp	**50.000**	**50.000**	**50.000**	**50.000**	50.000	50.000	0.000	0.000
2-30-250-2-5.csp	114.667	114.333	**113.333**	114.000	108.000	108.000	4.938	0.493
*2-50-500-5-8.csp	**100.000**	**100.000**	**100.000**	**100.000**	100.000	100.000	0.000	0.000
2-10-500-3-5.csp	166.333	**162.000**	163.667	163.333	157.000	157.000	3.185	1.572
4-20-500-1-5.csp	330.333	**328.000**	329.000	329.000	317.000	317.000	3.470	0.829
2-30-750-4-5.csp	198.667	**188.667**	193.667	194.667	187.000	187.000	0.891	3.562
4-10-750-2-8.csp	369.333	**366.000**	366.667	**366.000**	358.000	358.000	2.235	1.374
*20-30-750-3-0.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
*20-50-1000-4-9.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
*4-50-1000-4-4.csp	**250.000**	**250.000**	**250.000**	**250.000**	250.000	250.000	0.000	0.000
4-20-1000-2-4.csp	513.333	**507.667**	508.667	508.000	497.000	497.000	2.146	2.290
4-40-2000-2-2.csp	1037.333	**1006.333**	1025.000	1025.667	1000.000	1000.000	0.633	11.101
*4-40-2000-5-0.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.000	0.000	0.000
*4-40-2000-5-1.csp	**400.000**	**400.000**	**400.000**	**400.000**	400.000	400.000	0.000	0.000
2-50-5000-4-5.csp	1358.667	**1251.333**	1317.333	1316.333	1250.000	1250.000	0.107	38.408
2-30-5000-2-5.csp	2297.667	**2178.667**	2275.000	2275.333	2136.000	2136.000	1.998	45.961
*20-40-5000-5-6.csp	**1000.000**	**1000.000**	**1000.000**	**1000.000**	1000.000	1000.000	0.000	0.000
4-20-10000-2-6.csp	5152.000	**4980.667**	5100.333	5099.000	4960.000	4960.000	0.417	62.828
2-40-10000-3-2.csp	3678.333	**3344.667**	3588.333	3589.667	3327.000	3327.000	0.531	124.171
*20-30-10000-4-8.csp	**2500.000**	**2500.000**	**2500.000**	**2500.000**	2500.000	2500.000	0.000	0.000

A tabela 4.20 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 1, a coluna ils_2 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 2, a coluna ils_3 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 3, a coluna ils_4 que contém o valor encontrado pelo ILS usando o algoritmo de solução inicial 4, a coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima, a coluna ER com o erro relativo entre o lower bound e a melhor resposta entre as versões do ILS e a coluna σ com o desvio padrão entre as respostas das versões do ILS. Além disso, o símbolo * indica quais das instâncias em que se foi possível alcançar a solução ótima.

Tabela 4.21: Instâncias Chimani - Tempo de execução

instância	ils_1	ils_2	ils_3	ils_4
20-50-250-3-5.csp	**0.3661275**	1.1828335	1.2140000	1.2133131
2-50-250-5-7.csp	**0.3649707**	1.1681084	1.1970000	1.1961843
2-30-250-2-5.csp	**0.4671990**	1.2705096	1.6280000	1.6871994
2-50-500-5-8.csp	**0.7542652**	2.3495524	2.4210000	2.4403514
2-10-500-3-5.csp	**0.7208440**	2.3609401	2.6330000	3.7646297
4-20-500-1-5.csp	**0.9718775**	3.3554796	2.1240000	2.4610513
2-30-750-4-5.csp	**1.5706579**	3.0873925	4.2140000	4.4771006
4-10-750-2-8.csp	**1.1870004**	4.5670134	3.8240000	4.0408979
20-30-750-3-0.csp	**0.9593597**	3.1251354	3.1800000	3.1821912
20-50-1000-4-9.csp	**1.4873883**	4.8241718	4.9480000	4.9176634
4-50-1000-4-4.csp	**1.4971330**	4.9207485	4.9660000	4.9467842
4-20-1000-2-4.csp	**1.7629049**	5.6189571	5.9670000	7.1264287
4-40-2000-2-2.csp	**3.5360809**	11.1577528	11.8600000	11.2637181
4-40-2000-5-0.csp	**2.7870550**	9.0166348	9.2330000	9.2430652
4-40-2000-5-1.csp	**2.7885469**	8.9915192	9.2330000	9.2331401
2-50-5000-4-5.csp	**13.0917307**	26.3418982	41.2960000	55.2414392
2-30-5000-2-5.csp	**11.7723529**	49.3450932	38.0000000	39.3642855
20-40-5000-5-6.csp	**7.2138023**	23.2728669	23.9370000	23.9611738
4-20-10000-2-6.csp	**20.9722594**	66.5595695	61.7050000	65.6082970
2-40-10000-3-2.csp	**26.9323438**	93.8815302	109.6740000	68.6704955
20-30-10000-4-8.csp	**13.9987764**	45.1365810	46.4130000	46.3554583

A tabela 4.21 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ils_1 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 1 sobre a instância, a coluna ils_2 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 2 sobre a instância, a coluna ils_3 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução inicial 3 sobre a instância e a coluna ils_4 que contém o tempo em segundos da execução do ILS usando o algoritmo de solução

inicial 4 sobre a instância.

4.5 GA

Nesta seção exibimos os testes realizados sobre as instâncias com o GA .

4.5.1 Mcclure

Tabela 4.22: Instâncias Mcclure - Respostas

instância	*ga*	*t*	*lb*	*ub*	*ER*%
McClure-582-20-10-141.csp	111.000	0.0102807	97.000	97.000	14.433
McClure-582-20-12-141.csp	112.333	0.0109104	97.000	97.000	15.808
McClure-582-20-6-141.csp	101.333	0.0071720	88.000	88.000	15.152
McClure-586-20-10-98.csp	83.333	0.0044131	75.000	75.000	11.111
McClure-586-20-12-98.csp	85.000	0.0054671	76.000	76.000	11.842
McClure-586-20-6-100.csp	79.000	0.0020046	72.000	77.000	9.722

A tabela 4.22 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna *ga* com a resposta encontrada pelo GA, a coluna *t* com o tempo em segundos da execução do algoritmo sobre a instância, coluna *lb* com o lower bound da solução ótima, a coluna *ub* com o upper bound da solução ótima e a coluna *ER* com o erro relativo entre o lower bound e a resposta do GA.

4.5.2 Hufsky

Tabela 4.23: Instâncias Hufsky

instância	ga	t	lb	ub	$ER\%$
Hufsky-30-250-0.csp	29.000	0.1304349	27.000	27.000	7.407
Hufsky-50-250-12.csp	32.000	0.2855641	30.000	30.000	6.667
Hufsky-40-250-25.csp	26.000	0.1909848	24.000	24.000	8.333
Hufsky-40-300-1.csp	32.000	0.2702980	29.000	29.000	10.345
Hufsky-30-300-16.csp	32.333	0.2459922	30.000	30.000	7.778
Hufsky-30-300-35.csp	32.667	0.1780787	30.000	30.000	8.889
Hufsky-30-350-28.csp	40.667	0.2760628	36.000	36.000	12.963
Hufsky-40-350-4.csp	41.333	0.4179562	38.000	38.000	8.772
Hufsky-50-350-33.csp	41.667	0.6397809	38.000	38.000	9.649
Hufsky-50-400-2.csp	44.667	0.8187508	40.000	40.000	11.667
Hufsky-30-400-24.csp	40.667	0.3659019	36.000	36.000	12.963
Hufsky-20-400-19.csp	47.000	0.1575064	42.000	42.000	11.905
Hufsky-40-450-7.csp	56.000	0.6729163	50.000	50.000	12.000
Hufsky-20-450-3.csp	51.000	0.2657714	47.000	47.000	8.511
Hufsky-30-450-4.csp	51.667	0.4869150	46.000	46.000	12.319
Hufsky-30-500-25.csp	51.000	0.5710899	44.000	44.000	15.909
Hufsky-50-500-12.csp	59.667	1.1117149	49.000	49.000	21.769
Hufsky-40-500-20.csp	62.333	0.8805755	54.000	54.000	15.432

A tabela 4.23 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna ga com a resposta encontrada pelo GA, a coluna t com o tempo em segundos da execução do algoritmo sobre a instância, coluna lb com o lower bound da solução ótima, a coluna ub com o upper bound da solução ótima e a coluna ER com o erro relativo entre o lower bound e a resposta do GA.

4.5.3 Chimani

Tabela 4.24: Instâncias Chimani

instância	*ga*	*t*	*lb*	*ub*	*ER*%
20-50-250-3-5.csp	122.000	0.6087183	83.000	83.000	46.988
2-50-250-5-7.csp	71.000	0.3268519	50.000	50.000	42.000
2-30-250-2-5.csp	124.667	0.1461762	108.000	108.000	15.432
2-50-500-5-8.csp	148.333	0.9260298	100.000	100.000	48.333
2-10-500-3-5.csp	194.333	0.1233138	157.000	157.000	23.779
4-20-500-1-5.csp	359.667	0.2927815	317.000	317.000	13.460
2-30-750-4-5.csp	262.333	1.5405195	187.000	187.000	40.285
4-10-750-2-8.csp	444.333	0.2354658	358.000	358.000	24.115
20-30-750-3-0.csp	380.333	1.3478352	250.000	250.000	52.133
20-50-1000-4-9.csp	397.333	5.5540772	250.000	250.000	58.933
4-50-1000-4-4.csp	390.667	5.6712732	250.000	250.000	56.267
4-20-1000-2-4.csp	628.667	1.3465089	497.000	497.000	26.492
4-40-2000-2-2.csp	1300.667	16.0591473	1000.000	1000.000	30.067
4-40-2000-5-0.csp	644.667	14.1106179	400.000	400.000	61.167
4-40-2000-5-1.csp	643.667	15.8444092	400.000	400.000	60.917
2-50-5000-4-5.csp	1814.333	133.8316215	1250.000	1250.000	45.147
2-30-5000-2-5.csp	2431.000	72.6037410	2136.000	2136.000	13.811
20-40-5000-5-6.csp	1675.667	105.8297699	1000.000	1000.000	67.567
4-20-10000-2-6.csp	6304.667	106.1364112	4960.000	4960.000	27.110
2-40-10000-3-2.csp	4326.000	628.7202966	3327.000	3327.000	30.027
20-30-10000-4-8.csp	4108.667	258.2336644	2500.000	2500.000	64.347

A tabela 4.24 apresenta uma coluna instância que possui os nomes dos arquivos com os dados da entrada, a coluna *ga* com a resposta encontrada pelo GA, a coluna *t* com o tempo em segundos da execução do algoritmo sobre a instância, coluna *lb* com o lower bound da solução ótima, a coluna *ub* com o upper bound da solução ótima e a coluna *ER* com o erro relativo entre o lower bound e a resposta do GA.

Capítulo 5

Conclusão

Na seção 5.1 apresentamos resultados extraídos dos experimentos como o erro relativo e desvio padrão que nos permite realizar uma comparação entre os algoritmos a cada instância e de maneira global.

Na seção 5.2 descrevemos tudo o que foi construído e o que foi obtido a partir dos experimentos.

Já na seção 5.3 mostramos as principais limitações encontradas, o que foi realizado para suavizá-las e possíveis extensões.

5.1 Comparação entre os algoritmos

Tabela 5.1: Erros relativos: Instâncias McClure

instância	sa%	sap%	ils%	ga%
McClure-582-20-10-141.csp	4.124	**3.093**	**3.093**	14.433
McClure-582-20-12-141.csp	4.124	**3.093**	**3.093**	15.808
McClure-582-20-6-141.csp	3.788	**2.652**	**2.652**	15.152
McClure-586-20-10-98.csp	5.778	4.000	**3.556**	11.111
McClure-586-20-12-98.csp	6.140	**4.825**	**4.825**	11.842
McClure-586-20-6-100.csp	2.778	1.852	**1.389**	9.722

A tabela 5.1 apresenta uma coluna instância que identifica o arquivo e as informações da instância e as colunas *sa*, *sap*, *ils* e *ga* com o erro relativo entre o lower bound da solução ótima e a melhor resposta dada por uma de suas versões.

Tabela 5.2: erro relativo máximo e médio: Instâncias McClure

#	*sa*%	*sap*%	*ils*%	*ga*%
ER_{max}	6.140	4.825	4.825	15.808
ER_{medio}	4.455	3.252	3.101	13.011

A tabela 5.2 apresenta o erro relativo máximo ER_{max} encontrado em uma das execuções do algoritmo e a média entre os erros relativos ER_{medio} de todas as instâncias McClure testadas.

Tabela 5.3: Desvio padrão máximo e médio: Instâncias McClure

#	*sa*	*sap*	*ils*
σ_{max}	0.276	0.363	0.493
σ_{medio}	0.174	0.239	0.387

A tabela 5.3 apresenta o desvio padrão máximo σ_{max} e a médias dos desvios padrão σ_{medio} das quatro versões de cada algoritmo considerando todas as instâncias testadas.

Tabela 5.4: Erro relativo: Instâncias Hufsky

instância	sa%	sap%	ils%	ga%
Hufsky-30-250-0.csp	1.235	**0.000**	**0.000**	7.407
Hufsky-50-250-12.csp	**0.000**	**0.000**	**0.000**	6.667
Hufsky-40-250-25.csp	**0.000**	**0.000**	**0.000**	8.333
Hufsky-40-300-1.csp	**0.000**	**0.000**	**0.000**	10.345
Hufsky-30-300-16.csp	**0.000**	**0.000**	**0.000**	7.778
Hufsky-30-300-35.csp	**0.000**	**0.000**	**0.000**	8.889
Hufsky-30-350-28.csp	**0.000**	**0.000**	**0.000**	12.963
Hufsky-40-350-4.csp	**0.000**	**0.000**	**0.000**	8.772
Hufsky-50-350-33.csp	**0.000**	**0.000**	**0.000**	9.649
Hufsky-50-400-2.csp	**0.000**	**0.000**	**0.000**	11.667
Hufsky-30-400-24.csp	**0.000**	**0.000**	**0.000**	12.963
Hufsky-20-400-19.csp	**0.000**	**0.000**	**0.000**	11.905
Hufsky-40-450-7.csp	**0.000**	**0.000**	**0.000**	12.000
Hufsky-20-450-3.csp	0.709	**0.000**	**0.000**	8.511
Hufsky-30-450-4.csp	**0.000**	**0.000**	**0.000**	12.319
Hufsky-30-500-25.csp	2.273	**0.000**	**0.000**	15.909
Hufsky-50-500-12.csp	**0.000**	**0.000**	**0.000**	21.769
Hufsky-40-500-20.csp	**0.000**	**0.000**	**0.000**	15.432

A tabela 5.4 apresenta uma coluna instância que identifica o arquivo e as informações da instância e as colunas *sa*, *sap*, *ils* e *ga* com o erro relativo entre o lower bound da solução ótima e a melhor resposta dada por uma de suas versões.

Tabela 5.5: Erro relativo máximo e médio: Instâncias Hufsky

#	*sa*%	*sap*%	*ils*%	*ga*%
ER_{max}	2.278	0.000	0.000	21.769
ER_{medio}	0.234	0.000	0.000	11.293

A tabela 5.5 apresenta o erro relativo máximo ER_{max} encontrado em uma das execuções do algoritmo e a média entre os erros relativos ER_{medio} de todas as

instâncias Hufsky testadas.

Tabela 5.6: Desvio padrão máximo e médio: Instâncias Hufsky

#	*sa*	*sap*	*ils*
σ_{max}	0.289	0.236	0.000
σ_{medio}	0.114	0.021	0.000

A tabela 5.6 apresenta o desvio padrão máximo σ_{max} e a médias dos desvios padrão σ_{medio} das quatro versões de cada algoritmo considerando todas as instâncias testadas.

Tabela 5.7: Erro relativo: Instâncias Chimani

instância	sa%	sap%	ils%	ga%
20-50-250-3-5.csp	**0.000**	**0.000**	**0.000**	46.988
2-50-250-5-7.csp	**0.000**	**0.000**	**0.000**	42.000
2-30-250-2-5.csp	7.099	6.173	**4.938**	15.432
2-50-500-5-8.csp	**0.000**	**0.000**	**0.000**	48.333
2-10-500-3-5.csp	4.459	4.459	**3.185**	23.779
4-20-500-1-5.csp	3.996	**3.260**	3.470	13.460
2-30-750-4-5.csp	**0.713**	**0.713**	0.891	40.285
4-10-750-2-8.csp	2.886	**2.048**	2.235	24.115
20-30-750-3-0.csp	**0.000**	**0.000**	**0.000**	52.133
20-50-1000-4-9.csp	**0.000**	**0.000**	**0.000**	58.933
4-50-1000-4-4.csp	**0.000**	**0.000**	**0.000**	56.267
4-20-1000-2-4.csp	2.347	**2.012**	2.146	26.492
4-40-2000-2-2.csp	0.633	**0.500**	0.633	30.067
4-40-2000-5-0.csp	**0.000**	**0.000**	**0.000**	61.167
4-40-2000-5-1.csp	**0.000**	**0.000**	**0.000**	60.917
2-50-5000-4-5.csp	0.053	**0.000**	0.107	45.147
2-30-5000-2-5.csp	3.184	2.403	**1.998**	13.811
20-40-5000-5-6.csp	**0.000**	**0.000**	**0.000**	67.567
4-20-10000-2-6.csp	0.585	**0.370**	0.417	27.110
2-40-10000-3-2.csp	0.711	0.621	**0.531**	30.027
20-30-10000-4-8.csp	**0.000**	**0.000**	**0.000**	64.347

A tabela 5.7 apresenta uma coluna instância que identifica o arquivo e as informações da instância e as colunas *sa*, *sap*, *ils* e *ga* com o erro relativo entre o lower bound da solução ótima e a melhor resposta dada por uma de suas versões.

Tabela 5.8: Erro relativo máximo e médio: Instâncias Chimani

#	*sa*%	*sap*%	*ils*%	*ga*%
ER_{max}	7.099	6.179	4.938	67.567
ER_{medio}	1.270	1.074	0.979	40.399

A tabela 5.8 apresenta o erro relativo máximo ER_{max} encontrado em uma das execuções do algoritmo e a média entre os erros relativos ER_{medio} de todas as instâncias Chimani testadas.

Tabela 5.9: Desvio padrão médio e máximo: Instâncias Chimani

#	*sa*	*sap*	*ils*
σ_{max}	112.737	95.258	124.171
σ_{medio}	11.817	9.857	13.933

A tabela 5.9 apresenta o desvio padrão máximo σ_{max} e a médias dos desvios padrão σ_{medio} das quatro versões de cada algoritmo considerando todas as instâncias testadas.

5.2 Considerações finais

Construimos 4 algoritmos de solução inicial que em seguida foram usados em outros algoritmos, criamos um algoritmo usando simulated annealing com uma versão serial e outra paralela e usamos como solução inicial: 1, 2, 3 e 4, construímos também um algoritmo usando ILS que tinha como busca local o SA e onde as soluções iniciais eram: 1, 2, 3 e 4 e além disso elaboramos um algoritmo genético simples.

5.2.1 Simulated Annealing

Em relação às instâncias McClure, as 4 abordagens de soluções iniciais não apresentaram diferenças significativas entre si 5.3 e o melhor resultado conseguiu no pior caso uma solução que é aproximadamente 6.141% maior que a solução ótima 5.2.

As instâncias Hufsky, por sua vez, também não apresentaram grandes disparidades entre os algoritmos de solução inicial 5.6, contudo foi capaz de alcançar em 15 entradas a solução ótima da instância. Ademais, o melhor resultado conseguiu no pior caso uma solução que é aproximadamente 2.273% maior que a solução ótima 5.5.

Já nas instâncias Chimani, o algoritmo de solução inicial 2 conseguiu os melhores

resultados em 19 das 21 instâncias testadas. Além disso, em 10 instâncias foram encontradas as soluções ótimas. Ademais, o melhor resultado conseguiu no pior caso uma solução que é aproximadamente 7.099% maior que a solução ótima 5.8.

5.2.2 Simulated Annealing Paralelo

Em relação às instâncias McClure, as 4 abordagens de soluções iniciais não apresentaram diferenças significativas 5.3 e o melhor resultado conseguiu no pior caso uma solução que é 4.825% maior que a solução ótima 5.2.

As instâncias Hufsky, por sua vez, também não apresentaram grandes disparidades entre os algoritmos de solução inicial 5.9, contudo foi capaz de alcançar a solução ótima em todas as instâncias por pelo menos um dos algoritmos de solução inicial. Outra observação é que o algoritmo de solução inicial 1 obteve os melhores resultados ao chegar sempre no resultado ótimo.

Já nas instâncias Chimani, o algoritmo de solução inicial 2 conseguiu os melhores resultados em 18 das 21 instâncias testadas. Além disso, em 10 instâncias foram encontradas as soluções ótimas. Ademais, o melhor resultado conseguiu no pior caso uma solução que é aproximadamente 6.173% maior que a solução ótima 5.8.

5.2.3 ILS

Em relação às instâncias McClure, as 4 abordagens de soluções iniciais não apresentaram diferenças significativas 5.3 e o melhor resultado consegui no pior caso uma solução que é 4.825% maior que a solução ótima.

As instâncias Hufsky, por sua vez, também não apresentaram grandes disparidades entre os algoritmos de solução inicial, contudo foi capaz de alcançar a solução ótima em todas as instâncias por todos os algoritmos de solução inicial.

Já nas instâncias Chimani, o algoritmo de solução inicial 2 conseguiu os melhores resultados em 20 das 21 instâncias testadas. Além disso, em 10 instâncias foram encontradas as soluções ótimas. Ademais, o melhor resultado conseguiu no pior caso uma solução que é aproximadamente 4.938% maior que a solução ótima 5.8.

5.2.4 Algoritmo Genético

Em relação às instâncias McClure, o GA com conseguiu no pior caso uma resposta que estava 15.807% maior que a resposta da solução ótima 5.2.

Com as instâncias Hufsky conseguiu também boas soluções e que no pior caso obteve um erro de 8 unidades da resposta ótima. Ademais, conseguiu no pior caso uma solução que é aproximadamente 21.769% maior que a solução ótima 5.5.

Já nas instâncias Chimani, as respostas foram boas para instâcias pequenas, mas à medida que o comprimento das strings foram aumentando a qualidade foi decaindo o que levou ao algoritmo encontrar no pior caso uma solução que é aproximadamente 67.567% maior que a solução ótima 5.8.

5.2.5 Conclusão

Com isso, em relação à qualidade das soluções, vemos que o ILS por usar o SA como uma busca local obteve resultados melhores que o SA, o SAP com a estratégia de troca de informações também conseguiu resultados melhores que a sua versão serial: SA, já o GA por não usar nenhuma busca local para aprimorar as soluções, obteve respostas inferiores aos demais algoritmos.

5.3 Limitações e trabalhos futuros

5.3.1 Simulated Annealing

Uma das maneiras de se melhorar a qualidade das soluções é aumentar o número de iterações do SA, mas com isso o tempo aumenta o que não é desejável. Para equilibrar uma boa qualidade e uma execução rápida uma das alternativas seria otimizar o código e posteriormente elevar o número de iterações, o que pode ser alcançado trocando as strings por vetores e aplicando vetorização nas operações que os envolvem.

5.3.2 Simulated Annealing Paralelo

Além da estratégia adotada, o SAP foi testado com uma tática em que cada thread também executava o SA de maneira independente, mas ao contrário da que foi utilizada, cada uma das threads enviava a solução candidata para todas as outras e por isso acarretava em uma convergência prematura. Logo, a estratégia implementada que causa uma convergência mais lenta foi usada, porém outras que não foram abordadas: sem ciclo direcionado ou com mais vizinhos que poderiam melhorar a troca de informações seriam possíveis alternativas.

5.3.3 ILS

A principal limitação do ILS é em relação ao tempo de execução que para ser atenuada foi necessário adotar: uma temperatura dinâmica na busca local, utilizar uma constante real entre 0 e 1 para diminuir o número de iterações por temperatura, substituir as strings por vetores e usar instruções SSE em suas operações. Com isso, uma possível melhoria seria tentar substituir as instruções SSE por AVX. Além disso, uma versão do SAP poderia ser usada como busca local afim de melhorar a qualidade das respostas.

5.3.4 Algoritmo Genético

Um dos problemas do GA está na qualidade das soluções que em comparação com os demais algoritmos obteve respostas bem inferiores. Com isso, uma possível extensão seria aplicar um algoritmo de busca local nos indivíduos da população já que estes são gerados apenas pela combinação do crossover e da mutação. Além disso, os tempos de execução estão bem altos e para contornar isso poderia-se vetorizar e/ou paralelizar algumas das operações.

Referências

BOUCHER, C. Closest string with outliers. *BMC Bioinformatics*, v. 12, p. 1471–2105, 02 2011.

CHIMANI WOSTE, B. A closer look at the closest string and closest substring problem. *ALENEX 11: Proceedings of the Meeting on Algorithm Engineering & Expermiments*, Society for Industrial and Applied Mathematics, p. 13–24, 2011.

COLEY, D. A. *An Introduction to Genetic Algorithms for Scientists and Engineers*. 1. ed. [S.l.]: World Scientific Publishing Co. Pte. Ltd, 1999.

FESTA, P.; PARDALOS, P. Efficient solutions for the far from most string problem. *Ann Oper Res*, v. 196, p. 663–682, 2012.

FRANCES, M.; LITMAN, A. On covering problems of codes. *Theory of Computing Systems*, v. 30(2), p. 113–119, 1997.

HUFSKY, F. et al. Swiftly computing center strings. *LNBI*, Spring, v. 6293, p. 325–336, 2010.

KIRKPATRICK, S.; GELATT, C.; VECCHI, M. Optimization by simulated annealing. *Science (New York, N.Y.)*, v. 220, p. 671–80, 06 1983.

LI, M.; MA, B.; WANG, L. *On The Closest String and Substring Problems*. arXiv, 2000. Disponível em: <https://arxiv.org/abs/cs/0002012>.

MCCLURE MARCELLA, A.; VASI, T.; FITCH WALTER, M. Comparative analysis of multiple protein-sequence alignment methods. *Molecular biology and evolution*, v. 11 4, p. 571–92, 1994.

METROPOLIS, N. et al. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, v. 21, p. 1087–1092, June 1953.

REINSMA PUCA HUACHI VAZ PENNA, M. J. F. S. J. Um algoritmo simples e eficiente para resolução do problema do caixeiro viajante generalizado. *L SIMPóSIO BRASILEIRO DE PESQUISA OPERACIONAL*, 2018.

SANTOS, A. F. M. *Algoritmos heurísticos para a solução do Problema da Cadeia de Caracteres Mais Próxima*. Dissertação (Mestrado) — CEFET-MG, 2018.

SOUZA, M. J. F. *Inteligência Computacional para Otimização*. [S.l.], 2006. Disponível em: <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/SA.PPT>.