UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IANA MARY GOMES COSTA

UM ALGORITMO VND PARA O PROBLEMA DA PARTIÇÃO DE STRINGS COMUNS MÍNIMA

QUIXADÁ

2019

IANA MARY GOMES COSTA

UM ALGORITMO VND PARA O PROBLEMA DA PARTIÇÃO DE STRINGS COMUNS MÍNIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo Henrique Macêdo de Araújo

Coorientador: Prof. Dr. Fábio Carlos Sousa Dias

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C872a Costa, Iana Mary Gomes.
Um algoritmo vnd para o problema da partição de strings comuns mínima / Iana Mary Gomes Costa. – 2019.
43 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2019.
Orientação: Prof. Dr. Paulo Henrique Macêdo de Araújo.
Coorientação: Prof. Dr. Fábio Carlos Sousa Dias.

1. Heurística. 2. Otimização. 3. Biologia Computacional. I. Título.

CDD 004

IANA MARY GOMES COSTA

UM ALGORITMO VND PARA O PROBLEMA DA PARTIÇÃO DE STRINGS COMUNS MÍNIMA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: ___/___/___

BANCA EXAMINADORA

Prof. Dr. Paulo Henrique Macêdo de Araújo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio Carlos Sousa Dias (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araujo Tavares
Universidade Federal do Ceará (UFC)

A Deus,

Ao meus pais, Sheila e Francisco,

Ao meu namorado, Rodrigo.

AGRADECIMENTOS

Agradeço a Deus por seu Amor e por ter me dado sabedoria e força necessária para concluir mais essa etapa da minha vida.

Agradeço aos meus pais, Sheila e Francisco, pelo constante apoio, orientação, conselho, carinho e por todos os esforços feito pra que eu pudesse chegar até aqui. Obrigada por tudo mamãe e papai!

Agradeço meus irmãos, por sempre me incentivarem.

A todos os meus amigos, que de alguma forma me incentivaram. Em especial o meu amigo e namorado Rodrigo, pelo seu grande apoio e compreensão. Sou muito grato a todos!

Agradeço à minha família em geral, por sempre me incentivarem.

Ao professor Paulo Henrique, por ter me orientado no meu primeiro artigo acadêmico. Também por sua orientação, ensinamentos e disponibilidade neste nesse trabalho. Também sou grata pelos seus conselhos e ensinamentos. Obrigada, Paulo Henrique!

A professor Fábio Dias, por sua orientação, ensinamentos e paciência comigo durante esse trabalho, obrigada Fábio!

Ao professor Wladimir Tavares pelas suas valiosas contribuições nesse trabalho.

Aos demais professores do campus, em especial, Paulo de Tarso, Viviane Menezes, Diana Braga, Marcos Dantas, Régis e Aragão pelo seu esforço constante de tornar seus alunos não somente bons profissionais, mas boas pessoas.

A todos que compõe a UFC Quixadá, por tornarem o campus um dos melhores ambientes de aprendizagem que poderia ter.

Agradeço a todos os funcionários do campus, em especial a Gerlyson, Dias, Natália, Gilmário, Venício, Pipa e Kaká.

A todos que contribuíram positivamente para a minha formação.

"(..) Aprende que nunca se deve dizer a uma criança que sonhos são bobagens... Poucas coisas são tão humilhantes e seria uma tragédia se ela acreditasse nisso (..)"

(O Menestrel - William Shakespeare)

RESUMO

Neste trabalho, propomos uma implementação do algoritmo Variable Neighborhood Descent (VND) para o problema de Partição de Strings Comuns Mínima (em inglês, Minimum Common String Partition Problem ou MCSP) a fim de encontrar soluções de alta qualidade em tempo hábil. Para isso, definimos adaptações de uma heurística gulosa conhecida para esse problema para serem usadas no algoritmo de busca local. Os resultados são promissores em relação às adaptações da heurística gulosa, enquanto que o algoritmo VND demanda muito tempo de execução.

Palavras-chave: VND. Busca Local. Biologia Computacional.

ABSTRACT

In this work, we propose an implementation of the Variable Neighborhood Descent (VND) algorithm for the Minimum Common String Partition (MCSP) problem in order to find high quality solutions in a timely manner. To do so, we define adaptations of a known greedy heuristic for this problem to be used in the local search algorithm. The results are promising concerning the greedy heuristic adaptations, while the VND algorithm demands too much running time.

Keywords: VND. Local search. Computational Biology.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Na ciência da computação, existem vários problema de otimização que tem como objetivo encontrar uma solução ótima dentre um conjunto de várias soluções viáveis. Um caso especial de problema de otimização são os problemas de otimização combinatória, onde o conjunto de soluções viáveis é finito.

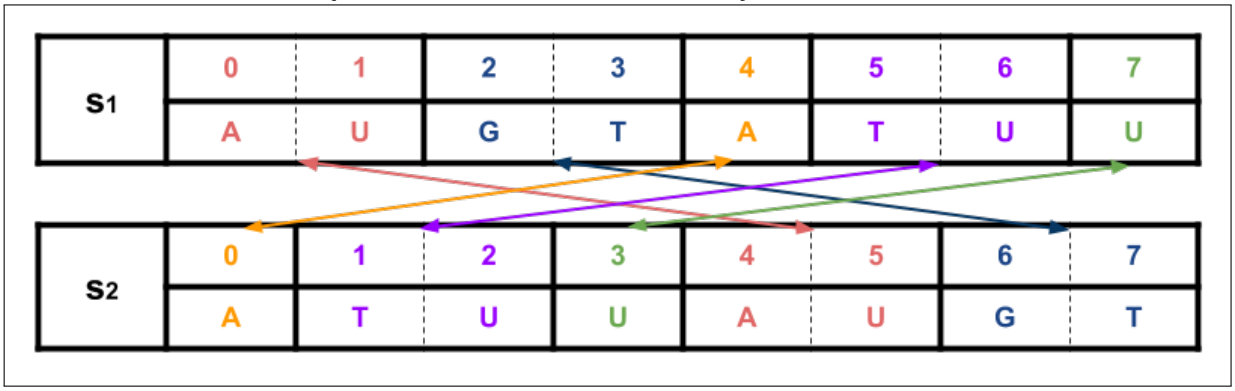
Na área de biologia computacional existem alguns problemas de otimização combinatória que consistem em comparar strings. Temos como exemplos o problema do Consenso de Strings (MENESES *et al.*, 2005; MOUSAVI *et al.*, 2012), o problema da Menor Subsequência Comum e suas variantes (HSU; DU, 1984; WATERMAN, 1981), o problema de Alinhamento (GUSFIELD, 1997) e o problema da Partição de Strings Comuns Mínima (BLUM; RAIDL, 2015; FERDOUS; RAHMAN, 2013).

Nosso trabalho tem como finalidade tentar encontrar uma solução de boa qualidade para o problema da Partição de Strings Comuns Mínima (Minimum Common String Partition ou MCSP, em inglês). Este é um problema de otimização combinatória, tipo de problema que têm como objetivo maximizar ou minimizar uma função definida sobre um certo domínio, conhecido ser NP-Difícil (GOLDSTEIN *et al.*, 2004). O MCSP tem suas aplicações nas áreas de processamento de texto e compressão, além de possuir aplicações em vários problemas na área da biologia computacional, como por exemplo o de rearranjamento de genomas: verificar se uma dada string de DNA pode ser obtida através da reordenação de subsequências de uma outra string de DNA.

O MCSP recebe como entrada duas strings, s_1 e s_2, de mesmo tamanho e de símbolos pertencentes a um alfabeto finito Σ. Além disso, cada símbolo do alfabeto aparece na mesma quantidade de vezes nas strings, sendo possível a quantidade ser igual a zero. Uma solução viável para o problema consiste nos conjuntos P_1 e P_2 que correspondem a uma partição de substrings não sobrepostas de s_1 e s_2, respectivamente, tais que existe uma bijeção entre os elementos da partição de modo que essa bijeção relacione strings iguais de s_1 e s_2. O valor de uma solução viável é dado por $z = |P_1| = |P_2|$. O problema consiste em encontrar uma solução viável em que z é mínimo.

Considere como exemplo as strings de entrada $s_1 =$ "$AUGTATUU$" e $s_2 =$ "$ATUUAUGT$". Sendo assim, podemos ter como exemplo de partições que definem uma solução viável, $P_1 = \{$"AU", "GT", "A", "TU", "U"$\}$ e $P_2 = \{$"A", "TU", "U", "AU", "GT"$\}$, veja a Figura 1.

Figura 1 – Partição $P_1 = \{$"AU", "GT", "A", "TU", "U"$\}$



Fonte: Elaborado pela autora (2019).

Na literatura, encontramos algumas abordagens de resolução para o problema MCSP. A maioria delas fazem uso de formulações de programação linear inteira ou heurísticas baseadas nas formulações apresentadas em (BLUM; RAIDL, 2015). No nosso conhecimento, existem poucas abordagens que fazem uso de heurísticas ou meta-heurísticas para encontrar uma solução de boa qualidade para o problema (por exemplo, o uso da meta-heurística de Colônia de Formigas (FERDOUS; RAHMAN, 2013)).

No nosso trabalho, mostraremos um algoritmo de busca local baseado no algoritmo Variable Neighborhood Descent (VND). Também implementaremos a heurística gulossa (CHROBAK *et al.*, 2005), e mais dos algoritmos *Guloso Estendido* e *Guloso Shift*.

1.1 Objetivos

1.1.1 Objetivo Geral

O nosso objetivo neste trabalho é propor e mostrar a implementação da heurística Gulosa (CHROBAK *et al.*, 2005), e de mais duas heurísticas a *Guloso Estendido* e *Guloso Shift*. Além do algoritmo *VND*.

1.1.2 Objetivos Específicos

- Apresentar uma representação das soluções do problema;
- Propor um método para encontrar a solução viável inicial (limite superior);
- Apresentar uma estrutura de vizinhança de uma solução;
- Apresentar uma comparação dos resultados com os resultados da literatura.

2 TRABALHOS RELACIONADOS

Nesta seção serão discutidos os trabalhos relacionados a este. No caso, alguns trabalhos que propõem soluções exatas ou aproximadas para o problema MCSP.

2.1 The Greedy Algorithm for the Minimum Common String Partition Problem

Em (CHROBAK *et al.*, 2005), os autores estudaram um algoritmo guloso para o MCSP, onde, a cada iteração, extrai-se a maior substring comum de s_1 e s_2 com elementos ainda não coberto. Além disso, eles mostraram que a razão de aproximação da heurística está entre $\Omega(n^{0.45})$ e $O(n^{0.69})$, no caso do 2-MCSP, caso onde cada caráter aparece no máximo duas vezes, a razão de aproximação é igual a 3, e no caso do 4-MCSP, caso onde cada caráter aparece no máximo quatro vezes, ela dá um limite inferior de Ω (log n).

As heurísticas sugeridas no nosso trabalho são adaptações dessa heurística gulosa apresentada em (CHROBAK *et al.*, 2005). Usaremos ela de três formas: (i) implementação original e compararemos seus resultados com as demais heurísticas sugeridas e com o (FERDOUS; RAHMAN, 2013); (ii) será usada como base na criação de mais duas heurísticas gulosas; (iii) fornecer a solução inicial do algoritmo *VND*.

2.2 Solving the Minimum Common String Partition Problem with the Help of Ants

Em (FERDOUS; RAHMAN, 2013), é mostrado como encontrar uma solução aproximada para o MCSP usando uma implementação da meta-heurística colônia de formigas. Tal trabalho superou os resultados do trabalho (CHROBAK *et al.*, 2005) na grande maioria das instâncias dos experimentos realizados.

Seus resultados serão comparados com os resultados das nossas heurísticas implementadas a fim de saber o quanto os resultados estão positivos ou não com base no gap.

2.3 Computational Performance Evaluation of Two Integer Linear Programming Models for the Minimum Common String Partition Problem

Em (BLUM; RAIDL, 2015), os autores mostram um novo modelo de programação linear inteira, denotado aqui por (M). Neste modelo existe a seguinte notação:

- T é conjunto de todas as strings (exclusivas) que aparecem como substrings pelo menos

uma vez em s_1 e s_2;

- Para cada $t \in T$, Q_t^1 e Q_t^2 é o conjunto de todas as posições entre 1 e n nas quais t começa nas strings de entrada s_1 e s_2, respectivamente;
- Para cada $t \in T$ e $k \in Q_t^i$, $i \in \{1,2\}$, a variável $y_{t,k}^i$ significa se a substring t foi escolhida na posição determinada por Q_t^i ($y_{t,k}^i = 1$) ou não ($y_{t,k}^i = 0$).

$$(M) \qquad \min \quad \sum_{t \in T} \sum_{k \in Q_t^1} y_{t,k}^1 \tag{2.1}$$

$$\text{s.t.} \quad \sum_{t \in T} \sum_{k \in Q_t^1 | k \leq j < k+|t|} y_{t,k}^1 = 1, \qquad j = 1, \ldots, n \tag{2.2}$$

$$\sum_{t \in T} \sum_{k \in Q_t^2 | k \leq j < k+|t|} y_{t,k}^2 = 1, \qquad j = 1, \ldots, n \tag{2.3}$$

$$\sum_{k \in Q_t^1} y_{t,k}^1 = \sum_{k \in Q_t^2} y_{t,k}^2, \qquad t \in T \tag{2.4}$$

$$y_{t,k}^1 \in \{0,1\} \qquad t \in T, k \in Q_t^1 \tag{2.5}$$

$$y_{t,k}^2 \in \{0,1\} \qquad t \in T, k \in Q_t^2$$

A função objetiva (2.1), diz o valor de quantas substrings foram escolhidas. Já as restrições (2.2) e (2.3) garantem que para cada posição $j = 1, \ldots n$ as strings de entrada, s_1 e s_2, exatamente uma substring de cobertura é escolhida. E as equações (2.4) garantem que cada substring $t \in T$ seja escolhida com o mesmo número de vezes dentro de s_1 e s_2.

A formulação (M) permite encontrar soluções exatas, de qualidade já razoável, em um tempo de execução relativamente baixo e também produz melhores soluções finais na maioria dos casos em que soluções ótimas comprovadas não puderam ser identificadas dentro do limite de tempo.

O trabalho (BLUM; RAIDL, 2015) tem como resultado o valor da solução ótima exato ou aproximado de cada instância. No nosso trabalho, uma das métricas de comparação de resultados foi o *gap*, no qual um dos fatores que entra no cálculo é valor do ótimo exato ou a melhor aproximação conhecida. Portanto, usaremos esses valores baseados nos valores fornecidos em (BLUM; RAIDL, 2015).

2.4 Comparação entre os trabalhos relacionados e o proposto

O Tabela 1 abaixo mostra uma comparação de métodos para encontrar uma solução para o problema MCSP. As informações na tabela são: se o trabalho usa ou não programação

inteira ou meta-heurística para resolver o problema, e se o trabalho encontra uma solução ótima ou não.

Tabela 1 – Comparação entre os trabalhos relacionados

Trabalhos	Solução Ótima	Programação Inteira	Heurística
(BLUM; RAIDL, 2015)	Sim	Sim	Sim
(CHROBAK *et al.*, 2005)	Não	Não	Sim
(FERDOUS; RAHMAN, 2013)	Não	Não	Sim
Trabalho Proposto	Não	Não	Sim

Fonte: Elaborada pela autora (2019).

3 FUNDAMENTAÇÃO TEÓRICA

3.1 Problema de otimização combinatória na área da biologia computacional

A biologia computacional é uma área de estudo que busca resolver problemas ligados a biologia com base em técnicas da ciência da computação, matemática aplicada e estatística. Vários desses problemas podem ser modelados como problemas de otimização combinatória, como é mostrado no trabalho (GREENBERG *et al.*, 2004). O problema tratado neste trabalho é um exemplo de problema que tem aplicações na área de biologia computacional e pode ser categorizado como um problema de otimização combinatória.

Como acontece na maioria dos problemas reais, o MCSP é um problema é NP-Difícil, (GOLDSTEIN *et al.*, 2004). Para instâncias grandes, o tempo de execução para determinar uma solução ótima pode ser inviável. Por isso, algumas das estratégias utilizadas para contornar esse problema é tentar encontrar uma solução viável de boa qualidade através de heurísticas.

3.2 Heurísticas

Segundo (SANTOS, 2017), heurística é toda técnica que ajuda o processo de busca por uma solução de um problema, porém não garante encontrar a solução ótima para o problema, porém tende a encontrar uma solução de qualidade, em um tempo bom. Podemos classificar as heurística nos seguintes métodos:

- Construtivo: método iterativo que começa a partir de uma solução vazia e a cada nova iteração adiciona uma nova parte a solução, e isso se repete até que a solução esteja completa;
- Decomposição: método que divide o problema em vários problemas menores, de forma em que resolve esses problemas menores possa compor uma solução problema original;
- Redução: método que identifica algumas características que a solução ótima deverá possuir e simplifica o problema, fixando-se o valor de algumas várias, por exemplo variáveis;
- Manipulação do modelo: método que adaptam modelo de tal forma que ele fique mais fácil de resolver, por exemplo usando relaxação linear e lagrangeana;
- Busca: método onde pode ser encontrada a maioria das meta-heurísticas, método a qual partir de uma solução inicial, podendo ser obtida a partir de outra heurística), e busca uma solução melhor dentro da sua vizinhança.

Uma meta-heurística consiste em um framework para a criação de uma heurística para um problema específico. Elas possuem duas principais vantagens: i) fácil implementação e adaptação para um problema de otimização qualquer se comparado com uma criação de uma heurística do zero; ii) geralmente produz resultados de alta qualidade devido ao seu aperfeiçoamento por conta das várias aplicações e testes realizados com a mesma para diferentes problemas. As mesmas são caracterizadas também por apresentarem estratégias de alto nível que exploram o espaço de busca através da utilização de diferentes métodos.

3.3 Algoritmo Variable Neighborhood Descent (VND)

O algoritmo VND é um algoritmo de busca local simples que, por isso, será utilizado para implementação neste trabalho com o intuito de tentar encontrar uma solução de boa qualidade para o problema MCSP a partir de uma solução viável inicial. Esse algoritmo é geralmente usado como parte fundamental de uma conhecida meta-heurística chamada Variable Neighborhood Search (VNS). Nessa busca local realizada pelo VND, a vizinhança completa de uma solução x é dividida em k vizinhanças parciais, $N_1(x), \ldots, N_k(x)$, as quais serão exploradas iterativamente se necessário.

3.3.1 Pseudo-código do algoritmo VND

O Algoritmo 1 descreve o pseudo-código do algoritmo *VND*. Ele recebe como entrada uma solução inicial x. A execução do algoritmo segue da seguinte forma: na linha 2, k é inicializada com 1, pois se começa a vasculhar a primeira vizinhança; na linha 3, há um laço cuja condição de parada é $(k > k_{max})$, o que indica a finalização do algoritmo após não ser encontrada uma solução melhor que a atual dentro de todas as suas vizinhanças ($k = 1, \ldots, k_{max}$); dentro desse laço, nas linhas 4-11, acontece uma busca local pelo melhor vizinho de x, x', na k-ésima vizinhança. Se x' é melhor que x, então x é atualizado com o valor de x' e k retorna ao valor 1. Caso contrário, k será incrementado em 1 pois será vasculhada a próxima vizinhança de x na

iteração seguinte.

Algoritmo 1: Algoritmo VND

Input: Uma solução x
Result: Uma solução x

$k \leftarrow 1$
while $(k \leq k_{max})$ **do**
    Encontrar o melhor vizinho $x' \in N_k(x)$
    if $(f(x') < f(x))$ **then**
        $x \leftarrow x'$
        $k \leftarrow 1$
    end
    else
        $k \leftarrow k+1$
    end
end
Retorne x

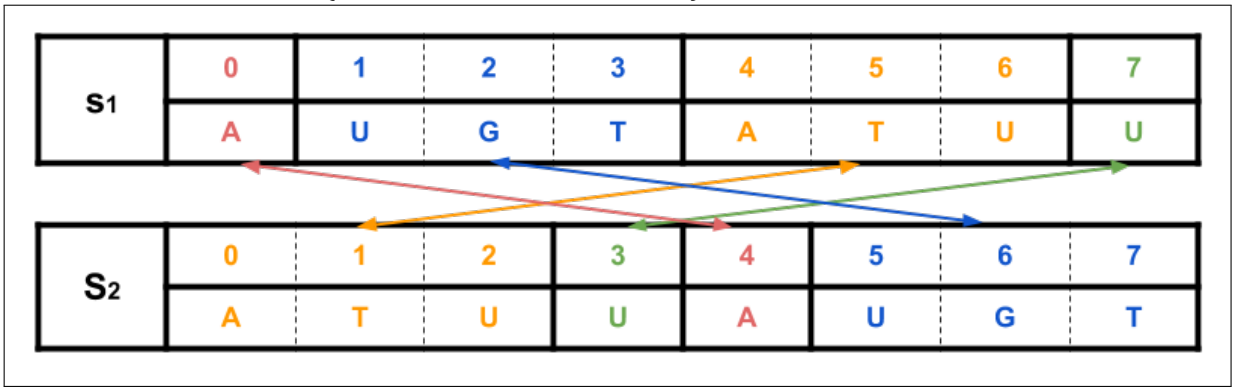
4 METODOLOGIA

4.1 Preliminares

O valor da função objetivo de uma solução x, representado por $f(x)$, será determinado pela cardinalidade de sua partição. Caso uma solução não seja viável, neste caso a partição não é válida, a função objetivo desta solução terá o valor definido por ∞.

Neste trabalho, para representar uma substring comum a s_1 e s_2, iremos usar o conceito de bloco. Um bloco é uma quádrupla (i_1, j_1, i_2, j_2), onde i_1 e j_1 são as posições inicial e final do bloco em s_1, e i_2 e j_2 são as posições inicial e final da bloco em s_2, que correspondem a uma substring comum em s_1 e s_2. O tamanho do bloco será o tamanho da substring comum que ele representa. Uma solução viável x é formada por um conjunto de blocos, que definem a partição, tal que a união das substrings correspondentes aos blocos que compõem todos os caracteres das strings s_1 e s_2 e não há sobreposição entre as substrings em cada uma das strings de entrada. Uma partição é considerada uma partição mínima se o valor da sua função objetivo, isto é, a quantidade de blocos, for a menor dentre todas as partições possíveis. Considere a Figura 2. Podemos ver a seguinte partição $P = \{$"A", "UGT", "ATU", "U"$\}$, que escrita usando a notação de blocos fica $P = \{(0,0,4,4),(1,3,5,7),(1,3,0,2),(7,7,3,3)\}$.

Figura 2 – Partição $P = \{$"A", "UGT", "ATU", "U"$\}$



Fonte: Elaborado pela autora (2019).

4.1.1 Função KMP

Em todos os algoritmos implementados neste trabalho, há momentos em que precisamos verificar se uma dada substring aparece em uma dada string. Para executar esse procedimento de uma maneira eficiente, usamos o algoritmo desenvolvido por Knuth, Morris e Pratt, denotado

por *KMP* (KNUTH *et al.*, 1977). Nos demais trabalhos da literatura aqui citados, não se tem certeza se os autores fazem uso do algoritmo *KMP* em suas implementações.

4.2 Heurística Gulosa

No Algoritmo 2, apresentamos a heurística gulosa de (CHROBAK *et al.*, 2004). É um processo iterativo, onde a cada iteração procura a maior substring comum a s_1 e s_2, considerando apenas os caracteres não cobertos pela solução parcial, inicialmente vazia, e então, adiciona a substring comum na solução parcial. Esse processo é repetido até que todos os caracteres de s_1 e s_2 sejam cobertos, onde teremos uma solução viável.

Algoritmo 2: Guloso

```
   Input: Duas strings s1 e s2
   Result: Uma solução x
 1 ts ← tamanhoString(s1)
 2 i ← tamanhoString(s1)
 3 j ← 1
 4 k ← 1
 5 while i ≥ 1 do
 6 |   k ← i
 7 |   while j ≤ ts do
 8 |   |   sub ← Substring com inicio em j e fim em j+k, na string s1
 9 |   |   if (kmp(sub, s2)) then
10 |   |   |   marca sub como coberto em s1 e s2
11 |   |   |   x ← x ∪ {sub}
12 |   |   end
13 |   |   j ← j+1
14 |   end
15 |   i ← i−1
16 end
17 Retorne x
```

Considere as strings $s_1 =$ "*ADBCACDD*" *e* $s_2 =$ "*DDADBCAC*". O algorimto *Guloso* vai executar da seguinte forma:

- iteração 1: a maior substring em s_1 é escolhida, no caso a própria string "*ADBCACDD*". Então, verifica-se se essa substring está em s_2. Como não está em s_2, o algoritmo executa a próxima iteração;
- iteração 2: a segunda maior substring é selecionada, no caso a substring "*ADBCACD*", pois note que o janelamento é feito da esquerda para direita; Como não pertence a s_2, segue-se para a próxima iteração.
- iteração 3: a terceira maior substring é selecionada, no caso a substring "*DBCACDD*", que

por acaso tem o mesmo tamanho da substring da iteração anterior; Como não pertence a s_2, segue-se para a próxima iteração.

- iteração 4: a quarta maior substring é selecionada, no caso a substring "*ADBCAC*". Agora, essa substring também ocorre em s_2, no caso, em posições diferentes. Dessa forma, adiciona-se o bloco $(1,6,3,8)$ (posições de início e fim da substring nas strings s_1 e s_2 respectivamente) à solução que está sendo criada e marca-se como cobertos os caracteres correspondentes da substring em cada uma das duas strings.
- seguindo essa ideia, depois de algumas iterações teremos como solução final a seguinte $x = \{$"*ADBCAC*", "*DD*"$\}$, de tamanho 2.

4.3 Heurística Gulosa Estendido

Nesse trabalho, desenvolvemos uma versão estendida da heurística gulosa de (CHROBAK *et al.*, 2004). Na heurística de (CHROBAK *et al.*, 2004) a solução parcial é inicialmente vazia. Nessa versão estendida, a solução parcial será iniciada com uma substring comum de s_1 e s_2. Após isso, aplicamos a heurística de (CHROBAK *et al.*, 2004) normalmente. Iremos chamar a heurística gulosa diversas vezes, em cada uma, passando uma substring comum diferente como solução parcial. A melhor solução encontrada será a devolvida pela heurística estendida. Portanto, nossa heurística será um processo iterativo, onde cada iteração iremos aplicar a heurística gulosa.

No Algoritmo 3, apresentamos o pseudo código da nossa heurística estendida. Ela recebe uma lista L de blocos, ordenada de forma decrescente pelo tamanho dos blocos. Na i-ésima iteração, selecionamos o bloco $b = L[i]$ como solução parcial e aplicamos o algoritmo Guloso.

A lista L é criada da seguinte forma: usando a técnica do algoritmo (CHROBAK *et al.*, 2005), de pegar da maior substring para menor substring, ou seja, pega todas as substrings de tamanho n, onde n é o tamanho das strings de entrada, depois as de tamanho $n-1$ até chegar nas substrings de tamanho igual a 1. Sendo que todas as substring que estão na lista pertence as

strings de entrada.

Algoritmo 3: Guloso Estendido

Input: Uma lista L de blocos e duas strings s_1 e s_2
Result: Uma solução x

```
1  i ← 1
2  f(x) ← ∞
3  while i ≤ tamanhoLista(L) do
4  |   marca L[K] como coberto em s1 e s2
5  |   x' ← Guloso(s1, s2)
6  |   if f(x') ≥ f(x) then
7  |   |   x ← x'
8  |   end
9  |   i ← i + 1
10 end
11 Retorne x
```

4.4 Heurística Gulosa Shift

A heurística chamada *Guloso Shift* pode ser vista no Algoritmo 4. Esse algoritmo cria uma lista L de blocos, ordenada de forma descrecente pelo tamanho dos blocos. Para cada bloco $b \in L$, é feito um shift, usando uma função chamada de *Shift*, para esquerda e direita sobre b, criando uma nova substring b', e a adicionamos em outra lista L'. Cada substring b', tem tamanho maior igual a 2 e menor igual ao tamanho de b. Depois de criado essa lista L', chamamos a heurística *Gulosa Estendido* passando a lista L' como parâmetro

A função *Shift* recebe uma lista L com k substrings que são comuns a s_1 e s_2. Ele funciona da seguinte forma: iterativamente, Pega uma substring $sub \in L$ e a partir dessa substring, cria-se uma lista L', adiciona *sub* a ela e depois adiciona todas as substrings em comum de tamanho maior ou igual a 2 que possuem interseção com sub.

Suponha as strings $s_1 =$ "*ADBCACDD*" e $s_2 =$ "*ACDDADBC*", os valores L e L' na primeira iteração para essa entrada seria:

- Considere que $n = 50$: Logo $|L| = 5$ e $|L'| = 45$
- $L =$ ["*ADBC*", "*ACDD*", "*DBC*", "*ACD*"];
- considere que seja $b =$ "*ADBC*", $L' =$ ["*ADBC*"];
- fazendo o shift para esquerda sobre b, temos: "*DBCA*", "*DBC*" e "*DB*", porém só entra em L' somente quem pertence a duas strings ao mesmo tempo, logo $L' =$ ["*ADBC*", "*DBC*", "*DB*"].
- fazendo o shift para direita temos sobre b: "*ADB*", "*DB*", porém só entra em L' somente quem pertence a duas strings ao mesmo tempo, logo $L' =$ ["*ADBC*", "*DBC*", "*DB*", "*ADB*"].

- ordenando a lista L', $L' = [$"$ADBC$", "ADB", "DBC", "DB"$]$.
- assim o processo se repete para o restante de L, até que todas as substrings que esteja em L tenha passado pelo shift.

Algoritmo 4: Guloso Shift

Input: Duas strings s_1, s_2
Result: Uma solução x
1 $L \leftarrow criarLista(s_1,\ (n\%10))$
2 $L' \leftarrow funcaoShift(L,\ n,\ 10)$
3 $x \leftarrow GulosoEstendido(L',\ s_1,\ s_2)$
4 *Retorne x*

4.5 Algoritmo VND para o problema MCSP

Nesta seção descrevemos nossa implementação do algoritmo *VND* para o problema MCSP.

4.5.1 Solução Inicial

Em nossa implementação, a solução inicial, x_0, é definida pela solução retornada pelo algoritmo *Guloso*.

4.5.2 Vizinhança

A i-ésima vizinhança $N_i(x)$ de x é determinada pelas soluções retornadas pela aplicação do algoritmo *Guloso Estendido* para todos os blocos da lista retornada pela função *Shift* passando como entrada o i-ésimo bloco da solução x. Portanto, no algoritmo VND que desenvolvemos, adaptado ao problema MCSP, a quantidade de vizinhanças de cada solução é variável e igual a quantidade de blocos em sua partição

4.5.3 Algoritmo final

O algoritmo *VND* que implementamos para o problema MCSP é mostrado no Algoritmo 5.

Algoritmo 5: Implementação do VND para o MCSP

```
   Input: Duas strings s1 e s2
   Result: Uma solução x
1  x ← guloso(s1, s2)
2  k ← 1
3  k_max ← funcaoObjetivo(x)
4  L ← criarLista(x)
5  while ((k ≤ k_max) E (tamanhoString(L[k]) ≥ 2)) do
6  |  L' ← funcaoShift(L[k])
7  |  i ← 1
8  |  while (i ≤ tamanhoLitsa(L')) do
9  |  |  x' ← gulosoEstendido(L', s1, s2)
10 |  |  if (f(x') < f(x)) then
11 |  |  |  x ← x'
12 |  |  |  k ← 0
13 |  |  |  k_max ← funcaoObjetivo(x)
14 |  |  |  L ← criarLista(x)
15 |  |  end
16 |  |  i ← i + 1
17 |  end
18 |  k ← k + 1
19 end
20 Retorne x
```

Na linha 4, cria-se uma lista L composta por todas as substrings dos blocos da partição que define a solução x. Em cada iteração do algoritmo, é explorada uma nova vizinhança, a k-ésima vizinhança da solução atual x, a partir da função *Shift*, que irá gerar uma lista de blocos L' (linha 7). Cada solução vizinha é então determinada pela aplicação da heurística *GulosoEstendido* a um novo bloco da lista L' (linha 9). Se foi encontrada uma solução melhor que a atual na k-ésima vizinhança, então deve-se atualizar a melhor solução encontrada e criar a lista dos blocos referentes a essa nova solução (linhas 10-14). Caso contrário, se não foi encontrada uma solução melhor na k-ésima vizinhança, então deve-se incrementar o valor de k a fim de explorar a a vizinhança seguinte. Se k ultrapassa o valor de k_{max} (número de blocos na solução atual), o algoritmo termina pois o laço da linha 6 é finalizado.

5 EXPERIMENTOS COMPUTACIONAIS

Nossos experimentos foram realizados em um computador com processador Intel Core i5 7a Geração 2.5 GHz, memória RAM de 8 GB e sistema operacional Ubuntu 18.04 LTS. Os algoritmos foram implementados usando a linguagem C.

5.1 Conjunto de instâncias

Usamos as instâncias que foram disponibilizados pelos autores de (FERDOUS; RAHMAN, 2013), as quais são divididas em dois tipos de sequências de DNA: geradas aleatoriamente e sequências genéticas reais.

Para as sequências de DNA aleatórias, eles geraram 30 sequências de DNA de comprimento no máximo 600 aleatoriamente (STOTHARD, 2000). A ocorrência das bases A, T, G e C são iguais em cada instância. Cada sequência de DNA é embaralhada para criar uma nova sequência de DNA que corresponde a segunda string de entrada. O embaralhar é feito usando a caixa de ferramentas on-line (VILLESEN, 2007). A sequência de DNA aleatória original e seu par embaralhado constituem uma única entrada (X, Y) para o experimento. Este conjunto de dados é dividido em três grupos. As primeiras 10 instâncias têm comprimento menor ou igual a 200 bps (pares de bases) e caracterizam o Grupo 1. As 10 instâncias seguintes têm comprimento entre 201 e 400 bps e caracterizam o Grupo 2. As últimas 10 instâncias tem comprimento entre 401 e 600 bps e correspondem ao Grupo 3. Já para as sequências genéticas reais, as quais foram o *grupo real*, (Grupo 4), eles coletaram os dados da sequência gênica do NCBI GenBank1. Nós usamos em nossos experimentos 11 sequências genéticas cujos comprimentos estão entre 200 e 600.

5.2 Parâmetros

Executamos o algoritmo *Guloso Estendido* com três tamanhos diferentes da lista de blocos: $|L| = 200$; $|L| = 400$; e $|L| = 800$. Dentre estes três valores, o valor 800 foi o que deu o melhor resultado em relação de qualidade de solução por tempo de execução, empiricamente. Portanto, esse foi o valor escolhido para o tamanho das listas de blocos utilizadas na implementação dos algoritmos.

Para o algoritmo *Guloso Shift*, a soma dos tamanhos das listas é igual a 800, ou seja, $|L| + |L'| = 800$, onde esse valor ficou determinado com base no resultado do *Guloso Estendido*.

Sendo assim, testamos as três opções seguintes de tamanhos: $|L| = 750$ e $|L'| = 50$; $|L| = 700$ e $|L'| = 100$; $|L| = 650$ e $|L'| = 150$. Dentre essas três opções, $|L| = 650$ e $|L'| = 150$ foi a que deu o melhor resultado em relação de qualidade de solução por tempo de execução.

Seguindo os resultados dos testes acima, também definimos o tamanho da lista L' para nosso algoritmo *VND* em $|L'| = 150$. Já o número máximo de iterações, k_{max}, foi definido com o valor do tamanho da solução atual x (ver Tabela 2).

Tabela 2 – Valores usados na execução dos algoritmos

ALGORITMOS	PARÂMETROS		
	k_{max}	$\|L\|$	$\|L'\|$
GULOSO	-	-	-
GULOSO ESTENDIDO	$\|x\|$	800	-
GULOSO SHIFT	$\|x\|$	150	650
VND	$\|x\|$	$\|x\|$	150

Fonte: Elaborada pela autora (2019).

5.3 Resultados

Os 4 algoritmos foram executados para os 4 grupos de instâncias descritos na seção anterior. Determinamos a qualidade das soluções retornadas utilizando um gap em relação à solução ótima, se conhecida. Além do gap, utilizamos o tempo de execução em segundos para determinar a viabilidade de usar tais algoritmos. As informações para cada instância analisada foram:

- Uma solução ótima ou próximo do ótimo de uma instância i, será representa da seguinte forma x_o e x_o*, respectivamente. As soluções ótimas consideradas foram reportados por (BLUM; RAIDL, 2015).
- Para cada instância i: $gap_i \leftarrow (x_i - x_{io})/x_{io}$, onde x_i é a solução x de uma instância i encontrada pelo algoritmo.
- Para cada instância i: $tempo_i$, é o tempo de duração do algoritmo para encontrar uma solução x_i de uma instância i;
- Para cada grupo g: $x_{mg} \leftarrow x_{gi}/10$, onde x_{gi} é a soma de todos os x_i pertencente ao mesmo grupo g;
- Para cada grupo g: $gap_{mg} \leftarrow gap_{gi}/10$, onde gap_{gi} é a soma de todos os gap_i pertencente ao mesmo grupo g;
- Para cada grupo i: $tempo_{mg} \leftarrow tempo_{gi}/10$, onde $tempo_{gi}$ é a soma de todos os $tempo_i$ pertencente ao mesmo grupo g.

Para uma melhor análise, usaremos as cores:

- cor para indicar o valor de x_o e x_o*;
- cor para indicar o melhor gap;
- cor para indicar o melhor $tempo$ (ignorando o tempo da heurística *Guloso*);

Como os algoritmos *Guloso Estendido*, *Guloso Shift* e *VND* executam várias vezes o algoritmo *Guloso* então podemos afirma que o tempo *Guloso* vai melhor que os do *Guloso Estendido*, *Guloso Shift* e *VND*.Dessa maneira só comparemos em questão de tempo os algoritmos *Guloso Estendido*, *Guloso Shift* e *VND*.

5.3.1 Resultados Para o Grupo 1

Em relação ao gap, podemos ver que as heurísticas *Guloso Estendido* e *Guloso Shift* foram melhores em todas as instâncias do Grupo 1, como visto nas Tabelas 4 e 5, respectivamente. Já o *VND* teve o melhor gap em cinco instâncias, no caso as instâncias instâncias 2, 3, 4 7 e 8, como visto na Tabela 6.

Em relação ao $tempo$, o algoritmo *VND* são foi ruim em três instâncias, no caso nas instâncias 5, 6 e 8 como visto na Tabela 6. Já o *Guloso Estendido* só teve o melhor tempo em três instâncias, no caso as instâncias 5, 6 e 8 como visto na Tabela 4.

Quanto ao algoritmo *VND*, nota-se que o mesmo não obteve os melhores resultados, no geral. Porém, seu tempo de execução foi menor que os das heurísticas *Guloso Estendido* e *Guloso Shift*. Sendo assim, podemos concluir que no geral as heurísticas *Guloso Estendido* e *Guloso Shift* tiveram um resultado melhor, em relação ao gap, no Grupo 1, e o algoritmo *VND* em relação ao $tempo$, teve o melhor desempenho como mostrado na Tabela 7.

Tabela 3 – Resultado Guloso para o Grupo 1

Grupo 1					
ID	TAM STRING	x_o	GULOSO		
			x_i	gap_i	$tempo_i$
1	114	41*	46	12,20%	0,02387
2	137	47*	56	19,15%	0,01202
3	158	52*	62	19,23%	0,01670
4	113	41*	46	12,20%	0,02283
5	119	40*	44	10,00%	0,01278
6	115	40*	46	15,00%	0,03004
7	162	55*	65	18,18%	0,01659
8	132	43*	52	20,93%	0,00852
9	118	42*	51	21,43%	0,01403
10	170	54*	65	20,37%	0,01944

Fonte: Elaborada pela autora (2019).

Tabela 4 – Resultado Guloso Estendido para o Grupo 1

Grupo 1					
ID	TAM STRING	x_o	GULOSO ESTENDIDO		
			x_i	gap_i	$tempo_i$
1	114	41*	45	9,76%	1,77060
2	137	47*	53	12,77%	3,86870
3	158	52*	59	13,46%	7,70080
4	113	41*	45	9,76%	11,62370
5	119	40*	42	5,00%	2,28250
6	115	40*	44	10,00%	1,85840
7	162	55*	63	14,55%	8,23000
8	132	43*	49	13,95%	2,27080
9	118	42*	46	9,52%	1,98030
10	170	54*	63	16,67%	10,01600

Fonte: Elaborada pela autora (2019).

Tabela 5 – Resultado Guloso Shift para o Grupo 1

Grupo 1					
ID	TAM STRING	x_o	GULOSO SHIFT		
			x_i	gap_i	$tempo_i$
1	114	41*	45	9,76%	4,22210
2	137	47*	53	12,77%	9,48810
3	158	52*	59	13,46%	14,93400
4	113	41*	45	9,76%	4,02940
5	119	40*	42	5,00%	6,15890
6	115	40*	44	10,00%	5,29170
7	162	55*	63	14,55%	16,67300
8	132	43*	49	13,95%	6,50570
9	118	42*	46	9,52%	5,67610
10	170	54*	63	16,67%	19,21600

Fonte: Elaborada pela autora (2019).

Tabela 6 – Resultado VND para o Grupo 1

Grupo 1					
ID	TAM STRING	x_o	VND		
			x_i	gap_i	$tempo_i$
1	114	41*	46	12,20%	1,09640
2	137	47*	53	12,77%	2,96960
3	158	52*	59	13,46%	5,68800
4	113	41*	45	9,76%	2,09090
5	119	40*	43	7,50%	2,32810
6	115	40*	45	12,50%	1,87400
7	162	55*	63	14,55%	7,93060
8	132	43*	49	13,95%	2,27230
9	118	42*	47	11,90%	1,18300
10	170	54*	64	18,52%	6,43270

Fonte: Elaborada pela autora (2019).

Tabela 7 – Comparação entre as heurísticas do trabalho proposto em relação ao Grupo 1

Grupo 1			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	53,30	16,87%	0,01768
GULOSO ESTENDIDO	50,9	11,54%	4,16018
GULOSO SHIFT	50,9	11,54%	9,21950
VND	51,40	12,71%	3,38656

Fonte: Elaborada pela autora (2019).

5.3.2 Resultados Para o Grupo 2

Em relação ao Grupo 2, podemos ver que a heurística *Guloso Estendido* foi melhor em todas as instâncias, como visto na Tabela 9. Já a heurística *Guloso Shift* só não foi melhor em duas instâncias do Grupo 2, no caso nas instâncias 2 e 10. A heurística *VND* só foi melhor em duas instâncias do Grupo 2, no caso nas instâncias 3, 4, 5 e 8, como visto na Tabela 11. Além disso, a heurística *Guloso Estendido* também foi a que apresentou menor tempo de execução no geral (ver Tabela 12), o que demonstra sua superioridade em relação às demais heurísticas no geral para o Grupo 2.

Tabela 8 – Resultado Guloso para o Grupo 2

Grupo 2					
ID	TAM STRING	x_o	GULOSO		
			x_i	gap_i	$tempo_i$
1	337	98	120	22,45%	0,16903
2	376	103	124	20,39%	0,23573
3	334	96*	114	18,75%	0,16520
4	351	100	119	19,00%	0,18905
5	398	114	134	17,54%	0,28581
6	327	93	108	16,13%	0,15654
7	303	87	111	27,59%	0,12527
8	358	104	122	17,31%	0,20738
9	360	103	125	21,36%	0,20834
10	306	88*	109	23,86%	0,12406

Fonte: Elaborada pela autora (2019).

Tabela 9 – Resultado Guloso Estendido para o Grupo 2

Grupo 2					
ID	TAM STRING	x_o	GULOSO ESTENDIDO		
			x_i	gap_i	$tempo_i$
1	337	98	114	16,33%	113,30000
2	376	103	119	15,53%	160,36000
3	334	96*	111	15,63%	110,16000
4	351	100	114	14,00%	129,30000
5	398	114	132	15,79%	187,75000
6	327	93	103	10,75%	104,58000
7	303	87	103	18,39%	83,74400
8	358	104	117	12,50%	137,04000
9	360	103	120	16,50%	141,76000
10	306	88*	104	18,18%	85,11900

Fonte: Elaborada pela autora (2019).

Tabela 10 – Resultado Guloso Shift para o Grupo 2

Grupo 2					
ID	TAM STRING	x_o	GULOSO SHIFT		
			x_i	gap_i	$tempo_i$
1	337	98	114	16,33%	147,75000
2	376	103	120	16,50%	182,27000
3	334	96*	111	15,63%	148,97000
4	351	100	114	14,00%	139,41000
5	398	114	132	15,79%	205,84000
6	327	93	103	10,75%	140,09000
7	303	87	103	18,39%	118,61000
8	358	104	117	12,50%	136,41000
9	360	103	120	16,50%	160,05000
10	306	88*	105	19,32%	120,05000

Fonte: Elaborada pela autora (2019).

Tabela 11 – Resultado VND para o Grupo 2

Grupo 2					
ID	TAM STRING	x_o	VND		
			x_i	gap_i	$tempo_i$
1	337	98	115	17,35%	173,28000
2	376	103	120	16,50%	241,36000
3	334	96*	111	15,63%	201,73000
4	351	100	114	14,00%	313,86000
5	398	114	132	15,79%	321,37000
6	327	93	105	12,90%	130,87000
7	303	87	105	20,69%	86,16500
8	358	104	119	14,42%	152,34000
9	360	103	120	16,50%	249,80000
10	306	88*	106	20,45%	90,82300

Fonte: Elaborada pela autora (2019).

Tabela 12 – Comparação entre as heurísticas do trabalho proposto em relação ao Grupo 2

Grupo 2			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	118,6	20,44%	0,18664
GULOSO ESTENDIDO	113,7	15,36%	125,31130
GULOSO SHIFT	113,9	15,57%	149,94500
VND	114,70	16,42%	196,15980

Fonte: Elaborada pela autora (2019).

5.3.3 Resultados Para o Grupo 3

Para o Grupo 3, podemos ver que a heurística *Guloso Estendido* só não foi melhor em uma instância do Grupo 3 em relação ao *gap*, no caso a instância 4, como visto na Tabela 14. Já a heurística *Guloso Shift* só não foi melhor em duas instâncias, no caso nas instâncias 3 e 8, como visto na Tabela 15. A heurística *VND* só foi melhor em seis instâncias e novamente apresentou o maior tempo de execução em média, como visto na Tabela 17, o que mostra que quanto maior a instância pior fica seu tempo de execução em relação às demais heurísticas.

Novamente, podemos concluir que no geral a heurística *Guloso Estendido* teve o melhor resultado em relação ao *gap* no Grupo 3, como mostrado na Tabela 17.

Tabela 13 – Resultado Guloso para o Grupo 3

Grupo 3					
ID	TAM STRING	x_o	GULOSO		
			x_i	gap_i	$tempo_i$
1	577	154	182	18,18%	0,86792
2	556	152	180	18,42%	0,78495
3	599	160	199	24,38%	0,97783
4	588	159	192	20,75%	0,91702
5	547	147	174	18,37%	0,73648
6	517	143	172	20,28%	0,62071
7	535	145	174	20,00%	0,69837
8	542	149	179	20,13%	0,72010
9	559	148	176	18,92%	0,79193
10	543	145	170	17,24%	0,72566

Fonte: Elaborada pela autora (2019).

Tabela 14 – Resultado Guloso Estendido para o Grupo 3

Grupo 3					
ID	TAM STRING	x_o	GULOSO ESTENDIDO		
			x_i	gap_i	$tempo_i$
1	577	154	176	14,29%	589,75000
2	556	152	176	15,79%	530,38000
3	599	160	189	18,13%	671,29000
4	588	159	183	15,09%	622,30000
5	547	147	168	14,29%	506,87000
6	517	143	165	15,38%	421,24000
7	535	145	167	15,17%	479,01000
8	542	149	174	16,78%	496,21000
9	559	148	172	16,22%	538,71000
10	543	145	164	13,10%	496,35000

Fonte: Elaborada pela autora (2019).

Tabela 15 – Resultado Guloso Shift para o Grupo 3

Grupo 3					
ID	TAM STRING	x_o	GULOSO SHIFT		
			x_i	gap_i	$tempo_i$
1	577	154	176	14,29%	603,83000
2	556	152	176	15,79%	618,46000
3	599	160	190	18,75%	702,51000
4	588	159	182	14,47%	656,39000
5	547	147	168	14,29%	521,87000
6	517	143	165	15,38%	431,96000
7	535	145	167	15,17%	496,82000
8	542	149	175	17,45%	527,66000
9	559	148	172	16,22%	592,18000
10	543	145	164	13,10%	530,35000

Fonte: Elaborada pela autora (2019).

Tabela 16 – Resultado VND para o Grupo 3

Grupo 3					
ID	TAM STRING	x_o	VND		
			x_i	gap_i	$tempo_i$
1	577	154	176	14,29%	1594,60000
2	556	152	177	16,45%	1162,60000
3	599	160	190	18,75%	1702,10000
4	588	159	182	14,47%	1990,90000
5	547	147	168	14,29%	1341,30000
6	517	143	167	16,78%	1256,70000
7	535	145	169	16,55%	865,28000
8	542	149	174	16,78%	1195,10000
9	559	148	172	16,22%	1581,10000
10	543	145	164	13,10%	1388,50000

Fonte: Elaborada pela autora (2019).

Tabela 17 – Comparação entre as heurísticas do trabalho proposto em relação ao Grupo 3

Grupo 3			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	179,8	19,67%	0,78410
GULOSO ESTENDIDO	173,4	15,42%	535,21100
GULOSO SHIFT	173,5	15,49%	568,20300
VND	173,90	15,77%	1400,81800

Fonte: Elaborada pela autora (2019).

5.3.4 Resultados Para o Grupo 4

Em relação ao Grupo 4, *gap*, a heurística *Guloso Estendido* apresentou o melhor valor de *gap* em todas as instâncias do grupo, como visto na Tabela 19. A heurística *Guloso Shift* foi melhor em cinco instâncias, no caso nas instâncias 1, 3, 5, 6 e 11, como visto na Tabela 20. O algoritmo *VND* só foi melhor em três instâncias, no caso as instâncias 1, 2 e 5.

Novamente, em relação ao *tempo*,a heurística *Guloso Estendido* também apresentou o menor tempo de execução, mostrando mais uma vez sua superioridade em relação às demais heurísticas (ver Tabela 22).

Tabela 18 – Resultado Guloso para o Grupo 4

Grupo 4					
ID	TAM STRING	x_o	GULOSO		
			x_i	gap_i	$tempo_i$
1	252	78*	100	28,21%	0,06683
2	487	134	163	21,64%	0,52043
3	363	102	122	19,61%	0,21336
4	513	141	165	17,02%	0,61974
5	559	148	174	17,57%	0,80205
6	451	124	153	23,39%	0,42326
8	433	115	137	19,13%	0,36897
11	400	109	126	15,60%	0,29007
12	449	122	142	16,39%	0,41202
15	510	135	158	17,04%	0,62497

Fonte: Elaborada pela autora (2019).

Tabela 19 – Resultado Guloso Estendido para o Grupo 4

Grupo 4					
ID	TAM STRING	x_o	GULOSO ESTENDIDO		
			x_i	gap_i	$tempo_i$
1	252	78*	91	16,67%	49,50100
2	487	134	156	16,42%	350,91000
3	363	102	117	14,71%	145,54000
4	513	141	162	14,89%	413,52000
5	559	148	168	13,51%	543,53000
6	451	124	148	19,35%	280,91000
8	433	115	131	13,91%	250,36000
11	400	109	123	12,84%	195,92000
12	449	122	139	13,93%	278,54000
15	510	135	152	12,59%	414,59000

Fonte: Elaborada pela autora (2019).

Tabela 20 – Resultado Guloso Shift para o Grupo 4

Grupo 4					
ID	TAM STRING	x_o	GULOSO SHIFT		
			x_i	gap_i	$tempo_i$
1	252	78*	91	16,67%	48,32700
2	487	134	157	17,16%	369,66000
3	363	102	117	14,71%	147,10000
4	513	141	163	15,60%	436,60000
5	559	148	168	13,51%	564,83000
6	451	124	148	19,35%	297,26000
8	433	115	132	14,78%	258,18000
11	400	109	123	12,84%	203,12000
12	449	122	140	14,75%	290,14000
15	510	135	155	14,81%	432,41000

Fonte: Elaborada pela autora (2019).

Tabela 21 – Resultado VND para o Grupo 4

Grupo 4					
ID	TAM STRING	x_o	VND		
			x_i	gap_i	$tempo_i$
1	252	78*	91	16,67%	46,46700
2	487	134	156	16,42%	1057,20000
3	363	102	118	15,69%	260,05000
4	513	141	163	15,60%	817,10000
5	559	148	168	13,51%	2099,20000
6	451	124	149	20,16%	510,02000
8	433	115	132	14,78%	490,32000
11	400	109	124	13,76%	330,54000
12	449	122	140	14,75%	457,63000
15	510	135	153	13,33%	1559,60000

Fonte: Elaborada pela autora (2019).

Tabela 22 – Comparação entre as heurísticas do trabalho proposto em relação ao Grupo 4

Grupo 4			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	144	19,56%	0,43417
GULOSO ESTENDIDO	138,7	14,88%	292,33210
GULOSO SHIFT	139,4	15,42%	304,76270
VND	139,4	15,47%	762,81270

Fonte: Elaborada pela autora (2019).

5.3.5 Comparação com o trabalho de (FERDOUS; RAHMAN, 2013)

No trabalho (FERDOUS; RAHMAN, 2013), os autores desenvolveram uma aplicação da meta-heurística de Colônia de Formigas para o problema MCSP, resultando em um algoritmo não-determinístico. Usaremos aqui a notação *MAX_MIN* para denotar esse algoritmo. Os resultados desse trabalho são apresentados da seguinte forma: para cada instância *i* de cada grupo *g* o experimento foi realizado 4 vezes e a média dos resultados é relatada. Dessa forma, calculamos x_{gi} e gap_{gi} da seguinte forma:

- Para cada instância *i*: $gap_i \leftarrow (MAX\_MIN - x_o)/x_o$;
- Para cada grupo *g*: $x_{mg} \leftarrow x_{gi}/10$, onde x_{gi} é a soma de todos os *MAX_MIN*. pertencentes ao mesmo grupo *g*;
- Para cada grupo *g*: $gap_{mg} \leftarrow gap_{gi}/10$, onde gap_{gi} é a soma de todos os gap_i pertencentes ao mesmo grupo *g*.

Para fins de análise, compararemos somente o gap_{mg}, pois o trabalho em (FERDOUS; RAHMAN, 2013) não informa o tempo de execução.

Analisando a Tabela 23 podemos ver que o algoritmo *MAX_MIN* foi melhor que as heurísticas propostas pelo nosso trabalho para o Grupo 1. Na Tabela 24, podemos ver que os algoritmos *MAX_MIN* e *Guloso Estendido* tiveram os melhores resultados no Grupo 2. Analisando as Tabelas 25 e 26 podemos ver que a heurística *Guloso Estendido* obteve o melhor resultado de *gap* nos grupos C e D.

Tabela 23 – Comparação entre trabalho proposto e o (FERDOUS; RAHMAN, 2013) em relação ao grupo 1

GRUPO 1			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	53,30	16,87%	0,01768
GULOSO ESTENDIDO	50,90	11,54%	4,16018
GULOSO SHIFT	50,90	11,54%	9,21950
VND	51,40	12,71%	3,38656
(FERDOUS; RAHMAN, 2013)	49,13	7,80%	-

Fonte: Elaborada pela autora (2019).

Tabela 24 – Comparação entre trabalho proposto e o (FERDOUS; RAHMAN, 2013) em relação ao grupo 2

GRUPO 2			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	118,60	20,44%	0,18664
GULOSO ESTENDIDO	113,70	15,36%	125,31130
GULOSO SHIFT	113,90	15,57%	149,94500
VND	114,70	16,42%	196,15980
(FERDOUS; RAHMAN, 2013)	113,78	15,36%	-

Fonte: Elaborada pela autora (2019).

Tabela 25 – Comparação entre trabalho proposto e o (FERDOUS; RAHMAN, 2013) em relação ao grupo 3

GRUPO 3			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	179,80	19,67%	0,78410
GULOSO ESTENDIDO	173,40	15,42%	535,21100
GULOSO SHIFT	173,50	15,49%	568,20300
VND	173,90	15,77%	1400,81800
(FERDOUS; RAHMAN, 2013)	174,83	16,38%	-

Fonte: Elaborada pela autora (2019).

Tabela 26 – Comparação entre trabalho proposto e o (FERDOUS; RAHMAN, 2013) em relação ao grupo 4

GRUPO 4			
ALGORITMOS	x_{mg}	gap_{mg}	$tempo_{mg}$
GULOSO	144,0000	19,56%	0,43417
GULOSO ESTENDIDO	138,7000	14,88%	292,33210
GULOSO SHIFT	139,4000	15,42%	304,76270
VND	139,4	15,47%	762,81270
(FERDOUS; RAHMAN, 2013)	144,77500	20,09%	-

Fonte: Elaborada pela autora (2019).

Para uma melhor análise geral e comparação do *gap*, criamos o seguinte critério:

- 10 pontos por cada vez que um algoritmo ficou em primeiro;
- 8 pontos por cada vez que um algoritmo ficou em segundo;
- 6 pontos por cada vez que um algoritmo ficou em terceiro;
- 4 pontos por cada vez que um algoritmo ficou em quarto;
- 2 pontos por cada vez que um algoritmo ficou em quinto.

Observando a Tabela 27, os melhores algoritmos aparecem na seguinte ordem: *Guloso Estendido* com 38 pontos, *VND* com 36 pontos, *Guloso Shift* com 32 pontos, (FERDOUS; RAHMAN, 2013) com 26 pontos e *Guloso* com 14 pontos.

Tabela 27 – Melhor Algoritmo

ALGORITMOS	GRUPO 1	GRUPO 2	GRUPO 3	GRUPO 4	TOTAL
GULOSO	4	4	2	4	14
GULOSO ESTENDIDO	8	10	10	10	38
GULOSO SHIFT	8	8	8	8	32
VND	6	6	6	6	24
(FERDOUS; RAHMAN, 2013)	10	10	4	2	26

Fonte: Elaborada pela autora (2019).

6 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, estudamos o problema MCSP (Minimum Common String Partition) e, por se tratar de um problema NP-Difícil, apresentamos heurísticas para o mesmo. Esse problema possui diversas aplicações, incluindo as áreas de biologia computacional e processamento de textos.

Implementamos a heurística *Gulosa* (CHROBAK *et al.*, 2004) e também desenvolvemos as heurísticas *Guloso Adaptado* e *Guloso com Shift*, que são adaptações da heurística gulosa, para o problema MCSP. Além disso, implementamos um algoritmo de busca local *VND* que faz uso das demais heurísticas desenvolvidas a fim de obter uma melhor solução viável.

Como analisado no Capítulo 5, os resultados dos experimentos em geral mostram que a heurística *Guloso Estendido* proposta apresentou os melhores resultados, inclusive batendo os resultados da meta-heurística de Colonia de Formigas apresentada em (FERDOUS; RAHMAN, 2013). Além disso, pode-se observar que quanto maior o tamanho das strings de entrada, melhor é o resultado da heurística *Guloso Estendido* em relação às demais heurísticas comparadas. Nesse caso, o algoritmo *VND* não apresentou resultados satisfatórios devido ao enorme tempo de execução exigido quando o tamanho da entrada cresce.

Como trabalho futuro, sugerimos a implementação de meta-heurísticas para o problema MCSP que podem fazer uso das heurísticas e algoritmos aqui propostos.

REFERÊNCIAS

BLUM, C.; RAIDL, G. R. Computational performance evaluation of two integer linear programming models for the minimum common string partition problem. **arXiv preprint arXiv:1501.02388**, [S.l], 2015.

CHROBAK, M.; KOLMAN, P.; SGALL, J. The greedy algorithm for the minimum common string partition problem. In: **Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques**. [S.l.]: Springer, 2004. p. 84–95.

CHROBAK, M.; KOLMAN, P.; SGALL, J. The greedy algorithm for the minimum common string partition problem. **ACM Transactions on Algorithms (TALG)**, ACM, v. 1, n. 2, p. 350–366, 2005.

FERDOUS, S.; RAHMAN, M. S. Solving the minimum common string partition problem with the help of ants. In: SPRINGER. **International Conference in Swarm Intelligence**. [S.l.], 2013. p. 306–313.

GOLDSTEIN, A.; KOLMAN, P.; ZHENG, J. Minimum common string partition problem: Hardness and approximations. In: SPRINGER. **International Symposium on Algorithms and Computation**. [S.l.], 2004. p. 484–495.

GREENBERG, H. J.; HART, W. E.; LANCIA, G. Opportunities for combinatorial optimization in computational biology. **INFORMS Journal on Computing**, INFORMS, v. 16, n. 3, p. 211–231, 2004.

GUSFIELD, D. **Algorithms on strings, trees and sequences**: computer science and computational biology. [S.l.]: Cambridge university press, 1997.

HSU, W.; DU, M. Computing a longest common subsequence for a set of strings. **BIT Numerical Mathematics**, Springer, v. 24, n. 1, p. 45–59, 1984.

KNUTH, D. E.; Morris Jr, H, J.; PRATT, V. R. Fast pattern matching in strings. **j-SIAM-J-COMPUT**, v. 6, n. 2, p. 323–350, jun. 1977.

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

MOUSAVI, S. R.; BABAIE, M.; MONTAZERIAN, M. An improved heuristic for the far from most strings problem. **Journal of Heuristics**, Springer, v. 18, n. 2, p. 239–262, 2012.

SANTOS, I. M. Algoritmos heurísticos. Disponível em: http://computacao.cua.ufmt.br/ ivairton/courses/pl/anotacoes_de_aulas3.pdf. 2017. Acesso em: 27.nov.2019.

STOTHARD, P. The sequence manipulation suite: Javascript programs for analyzing and formatting protein and dna sequences. **Biotechniques**, v. 28, p. 1102–1104, 2000.

VILLESEN, P. **FaBox**: An online fasta sequence toolbox. 2007. Disponível em: http://http://www.birc.au.dk/software/fabox. Acesso em: 27.nov.2019.

WATERMAN, M. Identification of common molecular subsequence. **Mol. Biol**, v. 147, p. 195–197, 1981.