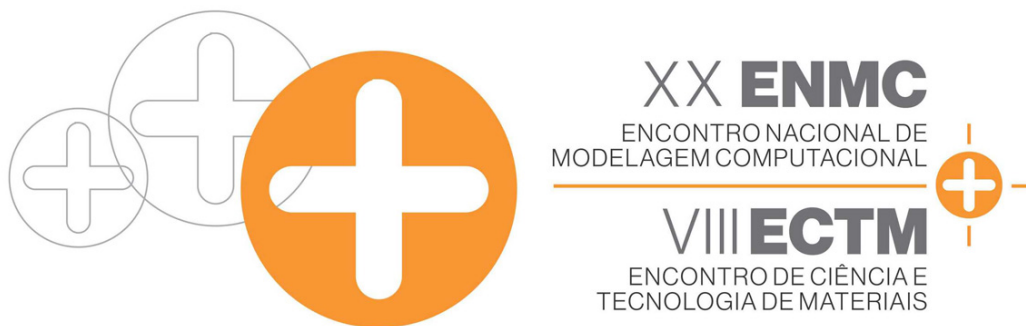




16 a 19 de Outubro de 2017
Instituto Politécnico - Universidade do Estado de Rio de Janeiro
Nova Friburgo - RJ

Um algoritmo Simulated Annealing com busca local para solução do Problema de Cadeia de Caracteres Mais Próxima

Augusto Flávio Morais Snatos¹ - aflavio@gmail.com

Marcone Jamilson Freitas Souza² - marcone@iceb.ufop.br

Sérgio Ricardo de Souza³ - sergio@dppg.cefetmg.br

¹Centro Federal de Educação Tecnológica de Minas Gerais - Belo Horizonte, MG, Brasil

²Universidade Federal de Ouro Preto, Ouro Preto, MG, Brasil

³Centro Federal de Educação Tecnológica de Minas Gerais - Belo Horizonte, MG, Brasil

Resumo. *O presente trabalho tem seu foco no Problema da Cadeia de Caracteres mais Próxima – PCCP (Closest String Problem - CSP). O objetivo no PCCP é o de encontrar uma sequência de caracteres que mais se aproxime, segundo uma dada métrica, de um dado conjunto de cadeias de caracteres de mesma dimensão. Sendo este problema da classe NP-difícil, é proposto um algoritmo híbrido para resolvê-lo. Em particular, propõe-se um algoritmo baseado na metaheurística Simulated Annealing, chamado HSALS, o qual é combinado com um método de busca local utilizando a estratégia Best Improvement. No algoritmo HSALS, essa busca local é aplicada cada vez que é atualizada a melhor solução. Para testá-lo, foram usados problemas-teste disponíveis na literatura e os resultados obtidos mostram que ele é competitivo com outro algoritmo heurística da literatura.*

Keywords: *Problema da Cadeia de Caracteres Mais Próxima. Simulated Annealing. Best Improvement.*

1. INTRODUÇÃO

Com a descoberta da estrutura de DNA em 1953, a biologia molecular tem sofrido grandes avanços, permitindo, por exemplo, a manipulação de sequências biomoleculares e o acesso a uma grande quantidade de dados. O processamento dessas informações não somente ganhou pertinência, como abriu portas para uma nova gama de problemas ainda inexplorados à época e que puderam ser tratados sob a perspectiva interdisciplinar da matemática aliada à ciência da computação (Setúbal, 1997).

Nos últimos anos, modelos de otimização têm sido analisados e propostos pela comunidade de Pesquisa Operacional mostrando que um grande número de problemas da biologia molecular podem ser formulados como problemas de Otimização Combinatória. A fundamental contribuição recai na abstração da real estrutura tridimensional do DNA e sua representação como uma sequência de caracteres unidimensional a partir de um alfabeto com quatro símbolos.

Como resultado dessa codificação linear do DNA e proteínas, podemos citar, por exemplo: a reconstrução de sequências de DNA a partir da sobreposição de fragmentos; a comparação de duas ou mais sequências para assim buscar suas similaridades; a busca por padrões que ocorrem com certa frequência no DNA e/ou sequências proteicas. Já no campo da otimização e programação matemática, em particular, podemos citar: problemas de sequências de alinhamento; problemas de reorganização do genoma; problemas de seleção e comparação de cadeias de caracteres; e reconhecimento e predição de estruturas proteicas (Festa, 2007).

O objetivo geral deste trabalho é implementar um algoritmo híbrido envolvendo a metaheurística *Simulated Annealing* – HSALS combinada com um método de busca local utilizando a estratégia *Best Improvement* para resolver o problema da cadeia de caracteres mais próxima.

O restante deste artigo está estruturado como segue. Na Seção 2. o problema PCCP é caracterizado. Na Seção 3. é feita uma breve revisão de literatura sobre o problema da cadeia de caracteres. Na Seção 4. é apresentado o algoritmo proposto para resolver o problema. Na Seção 5. são apresentados os resultados encontrados pelo algoritmo proposto, comparando-o com outros algoritmos da literatura. Finalmente, na Seção 6., são apresentadas as conclusões deste trabalho.

2. O PROBLEMA DA CADEIA DE CARACTERES MAIS PRÓXIMA

De acordo com Festa (2007), problemas de comparação e seleção de sequências de caracteres pertencem, de modo geral, à classe geral da biologia molecular e são conhecidos como sequências de consenso (*sequences consensus*), em que um grupo finito de sequências é dado e uma outra sequência é investigada de modo a encontrar uma similaridade. Em outras palavras, é preciso determinar uma sequência de caracteres, chamada de *consensus* de modo que esta – sequência – represente de alguma forma todas as outras sequências fornecidas.

Várias técnicas foram propostas para encontrar as regiões comuns para um conjunto de sequências de caracteres. A mais popular entre elas é a distância de *Hamming*, em que é calculada a distância entre a sequência central e cada uma das sequências dadas. Lancot (1999) descreve e justifica as razões e motivações do uso desta métrica para a maioria dos problemas de *consensus*.

Festa (2007) formaliza a distância de *Hamming* de acordo com a seguinte notação: Seja o alfabeto $\mathcal{A} = \{c_1, c_2, \dots, c_z\}$ formado por um grupo de elementos finitos, normalmente definido com as bases nitrogenadas do DNA, ou seja: $\mathcal{A} = \{A, C, G, T\}$; $\mathcal{S} = \{s^1, s^2, \dots, s^n\}$ um conjunto de sequências de caracteres de dimensão n ($|\mathcal{S}| = n$) onde $\mathcal{S} = (s^i \in \mathcal{S}, i = 1, 2, \dots, n)$ e $s = \{s_1, s_2, \dots, s_m\}$ uma sequência de caracteres de dimensão m ($|s| = m$) onde $s = (s_k \in \mathcal{A}, k = 1, 2, \dots, m)$.

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

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

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

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

Para efeito didático, a Figura 1 ilustra uma representação para duas cadeias de caracteres s_1 e s_2 e a respectiva distância de *Hamming*.

	1	2	3	4	5	6	7	8	9	10
cadeia de carácter: s^1	A	A	G	T	A	C	A	T	T	G
cadeia de carácter: s^2	A	G	G	T	A	C	A	C	T	G

Figura 1- Cálculo da Distância de *Hamming*

Pela Figura 1 percebe-se que as sequências s^1 e s^2 diferem entre si nas colunas 2 ($s_2^1 = A$ e $s_2^2 = G$) e 8 ($s_8^1 = T$ e $s_8^2 = C$). O valor da distância de *Hamming*, então, é calculado pela somatória dos caracteres díspares, o que, no caso, chega-se a dois.

Vista essa definição, o Problema da Cadeia de Caracteres Mais Próxima (PCCP) pode ser definido como: dado um conjunto finito de sequências $\mathcal{S} = \{s^1, s^2, \dots, s^n\}$, em que $\mathcal{A}(s^i \in \mathcal{A}^m, i = 1, 2, \dots, n)$, o problema tem como objetivo encontrar uma sequência central (*consensus*) $t \in \mathcal{A}^m$ tal que a distância de *Hamming* entre t e todas as outras sequências em $\mathcal{S}$ seja mínima. Desta forma, t é a sequência onde o menor valor d pode ser calculado a partir da seguinte expressão Festa (2007):

$$d_H(t, s^i) \leq d, \quad \forall s^i \in \Sigma \quad (3)$$

A Figura 2 ilustra um exemplo do PCCP e analisa a distância de *Hamming* entre a sequência central e cada uma das cadeias de caracteres. Desta forma, dado o conjunto de sequências $\mathcal{S} = \{s^1, s^2, s^3\}$ de dimensão $n = 3$, em que $s^i = \{s_1^i, s_2^i \dots s_{10}^i\}$ é a cadeia de carácter de dimensão $m = 10$ e $\mathcal{A} = \{A, C, G, T\}$ é o alfabeto de dimensão quatro, temos que a distância entre a sequência central t e s^1 é dada por: $d_{H,1}(t, s^1) = 7$; entre t e s^2 : $d_{H,2}(t, s^2) = 8$ e por fim: entre t e s^3 : $d_{H,3}(t, s^3) = 7$. Neste caso, a menor distância de *Hamming* é dada por 7 e a maior distância por 8.

cadeia de carácter: s^1	A	T	G	C	T	A	G	C	A	G
cadeia de carácter: s^2	T	T	A	G	T	A	G	C	T	G
cadeia de carácter: s^3	A	C	G	C	T	A	T	C	T	C

Sequência Central: t	A	A	G	T	A	C	A	T	T	G
------------------------	---	---	---	---	---	---	---	---	---	---

Figura 2- Exemplo do Problema da Cadeia de Caracteres

Normalmente o PCCP é representado pelas bases nitrogenadas do Ácido Dexoxirribonucléico, o DNA: Adenina (A), Timina (T), Guanina (G) e Citosina (C). Por outro lado, para o RNA – Ácido Ribonucléico, usa-se a Uracila (U) no lugar da Timina(T). Há, ainda, a possibilidade de se utilizar outros alfabetos, como o binário ou números inteiros positivos.

3. TRABALHOS RELACIONADOS

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

Roman (1992) foi o primeiro a estudar o PCCP, porém esse estudo ocorreu no campo da Teoria dos Códigos. A partir daí, somente no final dos anos noventa é que o problema foi provado como intratável computacionalmente, ou seja, pertence à classe NP-difícil (Frances, 1997; Lanctot, 1999).

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

Meneses (2004) conseguiu avanços significativos e propôs três formulações de programação inteira. O autor também utilizou uma heurística para determinar os limites superiores das soluções ótimas e, assim, agilizar o método *branch-and-bound*.

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

Faro (2010) propôs um algoritmo baseado na metaheurística Colônia de Formigas e o comparou com o trabalho de Liu (2008), o qual, por sua vez, baseou-se nos métodos *Simulated Annealing* e algoritmo Genético. Ambos os métodos de Liu (2008) foram implementados segundo as orientações clássicas da literatura. Os testes computacionais foram realizados com os seguintes problemas-teste: $n = \{10, 20, 30, 40, 50\}$ (número de sequências de caracteres), $m = \{10, 20, \dots, 50, 100, 200, \dots, 1000\}$ (dimensão de cada sequência de caracteres) e $\mathcal{A} = \{4\}$ (dimensão do alfabeto utilizado). O algoritmo de Colônia de Formigas superou quase a totalidade dos melhores resultados e mostrou-se bastante rápido computacionalmente.

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

Chimani (2011) trabalhou com o PCCP e o *Closest Substring Problem* e para isto utilizou a Programação Linear Inteira. Os experimentos computacionais revelaram que os métodos adotados tiveram resultados melhores à proposta de Meneses (2004). Por outro lado, a formulação proposta foi inspirada nas três formulações de Meneses (2004).

4. Algoritmo Proposto

Esta Seção está decomposta em duas etapas. Na primeira descreve-se o método utilizado para gerar uma solução inicial por meio de um algoritmo guloso – vide Seção 4.1. Na etapa

subsequente, 4.2, é apresentado o algoritmo de refinamento chamado HSALS (das iniciais em inglês *Hybrid Simulated Annealing with Local Search*), aplicado para melhorar a solução inicial.

4.1 Construção da Solução Inicial

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

Algoritmo 1: CONSTROISOLUÇÃOALEATÓRIA

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

Saída : s

```

1 para  $i = 1$  até  $m$  faça
2    $s_i \leftarrow$  Gere um caracter qualquer de  $\mathcal{A}$ 
3 fim
4 retorne  $s$ 

```

Uma vez formada a cadeia de caracteres aleatoriamente, esta, por sua vez, é submetida a um procedimento de Busca Local para, assim garantir que seja encontrado um ótimo local – vide algoritmo 2.

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

Com o melhor vizinho determinado, é possível agora encontrar o ótimo local usando uma Busca Local com a estratégia *Best Improvement*. Assim, o algoritmo 3 se inicia alocando um valor infinito positivo na variável f_{vizinho}^* – vide linha 1. Depois, toda a vizinhança é avaliada, através do algoritmo 2, de modo a encontrar a melhor solução local – vide 5–13

Ao final dos algoritmos 1, 2 e 3, a solução solução inicial é construída de acordo com o algoritmo 4. A figura 3 exemplifica a maneira de como são realizadas todas as possíveis trocas nas posições s_1 e s_{10} da sequência central (*consensus*) t no alfabeto $\mathcal{A} = \{A, C, T, G\}$.

Sequência Central: t	{	A	A	A	A	A	A	A	A	A	}
1º vizinho	{	C	A	A	A	A	A	A	A	A	}
2º vizinho	{	T	A	A	A	A	A	A	A	A	}
3º vizinho	{	G	A	A	A	A	A	A	A	A	}
...											
28º vizinho	{	A	A	A	A	A	A	A	A	C	}
29º vizinho	{	A	A	A	A	A	A	A	A	T	}
30º vizinho	{	A	A	A	A	A	A	A	A	G	}

Figura 3- Movimento de troca utilizado na Busca Local

Algoritmo 2: MELHORVIZINHO

Entrada: s - solução inicial

Saída : s - melhor vizinho

i_{melhor} - índice do vetor do melhor vizinho

k_{melhor} - melhor caracter do alfabeto

f_{vizinho}^* - valor da função de avaliação do melhor vizinho

```

1   $f_{\text{vizinho}}^* \leftarrow +\infty$ ;
2   $i \leftarrow 1$ ;
3   $s' \leftarrow s$ ;
4  enquanto  $i \leq m$  faça
5       $k \leftarrow 1$ ;
6      enquanto  $k \leq |\mathcal{A}|$  faça
7          se  $(s_i \neq \mathcal{A}_k)$  então
8               $s'_i \leftarrow \text{troca}(s_i, \mathcal{A}_k)$ ;
9               $f(s') \leftarrow \text{avalia}(s'_i)$ ;
10             se  $(f(s') < f_{\text{vizinho}})$  então
11                  $i_{\text{melhor}} \leftarrow i$ ;
12                  $k_{\text{melhor}} \leftarrow k$ ;
13                  $f_{\text{vizinho}}^* \leftarrow f(s')$ ;
14                  $s \leftarrow s'$ ;
15             fim
16         fim
17     fim
18 fim
19 retorne  $s, i_{\text{melhor}}, k_{\text{melhor}}, f_{\text{melhor}}^*$ 

```

Algoritmo 3: BUSCA LOCAL

Entrada: s - solução inicial

Saída : s - melhor solução local

```

1   $f_{\text{vizinho}}^* \leftarrow +\infty$ ;
2   $\text{melhora} \leftarrow \text{TRUE}$ ;
3   $s \leftarrow s_i$ ;
4  enquanto  $\text{melhora} == \text{TRUE}$  faça
5       $f_{\text{vizinho}}, i_{\text{melhor}}, k_{\text{melhor}} \leftarrow \text{melhorVizinho}(s)$ ;
6      se  $(f_{\text{vizinho}} < f(s))$  então
7           $s[i_{\text{melhor}}] \leftarrow \mathcal{A}[k_{\text{melhor}}]$ ;
8           $f(s) \leftarrow f_{\text{vizinho}}$ ;
9      senão
10          $\text{melhora} \leftarrow \text{FALSE}$ ;
11     fim
12 fim
13 retorne  $s$ 

```

Algoritmo 4: CONSTROI SOLUÇÃO INICIAL

Entrada: $\mathcal{A}$ - alfabeto utilizado
- dimensão da cadeia de caracteres

Saída : s

- 1 $s' \leftarrow \text{ConstroiSoluçãoAleatória}(s)$
 - 2 $s \leftarrow \text{BuscaLocal}(s')$
 - 3 retorne s
-

4.2 Fase de Refinamento

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

Este nome veio de uma analogia com o processo de recozimento físico de materiais sólidos, como por exemplo, uma chapa de aço, no qual o material é aquecido e depois resfriado muito lentamente, até atingir a sua configuração mais regular possível, isto é, o seu estado mínimo de energia. Deste modo, é possível eliminar os defeitos de cristalização, evitando que o material se torne quebradiço.

A característica chave deste algoritmo é permitir movimentos que pioram o valor da função objetivo. Assim, à medida que o parâmetro temperatura diminui, os movimentos de escalada ocorrem com menos frequência e é possível convergir para espaços de busca onde existe uma probabilidade maior de se encontrar o ótimo global. (Nikolaev, 2010)

O algoritmo 5 implementado é baseado na metaheurística *Simulated Annealing* e inclui uma fase de busca local. O método inicia-se com uma solução inicial – vide algoritmo 4 e linha 1. A cada nova iteração é gerada uma solução vizinha – linha 8. Avalia-se então a solução corrente e o vizinho. Caso este vizinho seja melhor, então este passa a ser a solução corrente – linhas 9-12. Caso contrário, há a probabilidade de aceitar a solução de piora – linhas 13-16. Esta probabilidade de aceitar ou não a solução de piora é dada pela expressão $e^{-\Delta/T}$, em que T é a temperatura atual.

O método começa com uma temperatura alta (T) e a cada iteração vai diminuindo de acordo com o parâmetro α – linha 18, que regula a taxa de resfriamento. Desta forma, há uma probabilidade maior das soluções de piora serem aceitas no início do processo; porém, essa probabilidade vai reduzindo à medida que a temperatura vai diminuindo. São estas soluções de piora que permitem que o algoritmo escape das armadilhas dos ótimos locais. A adaptação feita no método clássico consiste em fazer uma busca local usando a estratégia *Best Improvement* (algoritmo 3)

sempre que é encontrada uma solução de melhora – vide linha 12 do algoritmo 5.

Algoritmo 5: MÉTODO HSALS

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

Saída : s^*

```

1  $s \leftarrow s_0$  // Solução corrente
2  $s' \leftarrow s$  // Melhor solução obtida até então
3  $T \leftarrow T_0$  // Temperatura Corrente
4  $IterT \leftarrow 0$  // Número de iterações na temperatura T
5 enquanto  $T > T_{final}$  faça
6   enquanto  $IterT < SA_{max}$  faça
7      $IterT \leftarrow IterT + 1$ ;
8     Gere um vizinho qualquer  $s' \in N(s)$ ;
9      $\Delta \leftarrow f(s') - f(s)$ ;
10    se (  $\Delta < 0$  ) então
11       $s \leftarrow s'$ ;
12      se (  $f(s') < f(s^*)$  ) então  $s^* \leftarrow BuscaLocal(s')$ ;
13    senão
14      Tome  $X \in [0,1]$ ;
15      se (  $X < e^{-\Delta/T}$  ) então  $s \leftarrow s'$ ;
16    fim
17  fim
18   $T \leftarrow \alpha \times T$ ;
19   $IterT \leftarrow 0$ ;
20 fim
21 Retorne  $s^*$ ;
```

5. EXPERIMENTOS E RESULTADOS

Nesta seção são apresentados os resultados computacionais obtidos pelo método HSALS na resolução do Problema da Cadeia de Caracteres Mais Próxima. O método foi comparado com o trabalho de Faro (2010) e com a heurística de Programação Inteira de Chimani (2011).

As instâncias utilizadas no trabalho foram as mesmas de Chimani (2011). Os métodos foram desenvolvidos na linguagem de programação C++ e compiladas no GCC versão 7.1.1., processador Intel Core i7 – 3517u – 1.9GHz, com 8GB de memória RAM, no sistema operacional Linux – distribuição Arch, 64 bits.

Os parâmetros utilizados no algoritmo foram: Temperatura inicial: $t_0 = 1.000$; Temperatura final: $t_{final} = 0,001$; Taxa de resfriamento: $\alpha = 0,99$; Número máximo de iterações em uma determinada temperatura: $SA_{max} = 100$. A temperatura inicial do algoritmo foi calculada de forma empírica após algumas simulações preliminares.

A fim de validar o algoritmo proposto, tal como em Chimani (2011), foram utilizados os problemas-teste de acordo com os seguintes conjuntos: O alfabeto utilizado $\mathcal{A}$ é composto

por quatro caracteres, que são representados pelas bases nitrogenadas do DNA, ou seja $\mathcal{A} = \{A, C, G, T\}$; a quantidade de cadeias de caracteres em cada conjunto de sequências são $n = \{10, 20, 30, 40, 50\}$; já a dimensão de cada sequência de caracteres é dada por $m = \{500, 1000\}$. Cada uma das cem instâncias foram executadas cinquenta vezes, totalizando então cinco mil execuções.

Na Tabela 1 são apresentados os resultados obtidos pelo algoritmo HSALS e pelos métodos de Faro (2010) e Chimani (2011). Nessa tabela, o alfabeto $\mathcal{A}$ é composto por quatro caracteres, que são representados pela base nitrogenada do DNA, ou seja $\mathcal{A} = \{A, C, G, T\}$; a quantidade de cadeias de caracteres em cada conjunto de sequências são $n = \{10, 20, 30, 40, 50\}$; já a dimensão de cada sequência de caracteres é dada por $m = \{500, 1000\}$. Nas colunas HSALS têm-se o resultado do algoritmo *Hybrid Simulated Annealing with Local Search*; na coluna Ant, o método de Faro (2010) e, por último, na coluna ILP são apresentados os resultados do método exato de Chimani (2011). A coluna GAP mostra a diferença percentual entre o algoritmo HSALS e Ant com o método exato de Chimani (2011) e é calculada da seguinte forma: $((\text{Resultado do Algoritmo HSALS ou Ant}) - \text{ILS}) / \text{ILS} \times 100$. Observa-se que nos problemas-teste com $n > 40$, Chimani (2011) não informou o tempo de execução.

Tabela 1- Comparação com os métodos da literatura

Instância			HSALS			Ant			ILP	
A	n	m	Média	Tempo	GAP	Média	Tempo	GAP	Média	Tempo
4	10	500	308,75	0,06	6,17	317,00	2,11	9,00	290,80	0,08
4	10	1000	611,26	0,22	5,42	652,00	7,85	12,45	579,80	0,17
4	20	500	335,59	0,22	5,99	341,00	3,51	7,70	316,60	0,33
4	20	1000	665,42	0,55	5,13	695,00	11,8	9,81	632,90	0,77
4	30	500	348,98	0,38	6,23	351,00	6,76	6,84	328,50	0,60
4	30	1000	686,70	0,89	4,75	713,00	10,7	8,77	655,50	1,32
4	40	500	356,88	0,53	6,62	358,00	7,33	6,96	334,70	—
4	40	1000	698,60	1,21	4,62	722,00	16,00	8,13	667,70	—
4	50	500	364,21	0,69	7,21	362,00	12,90	6,56	339,70	—
4	50	1000	712,57	1,52	5,22	729,00	18,30	7,64	677,20	—

6. CONCLUSÕES

Este trabalho teve seu foco no Problema da Cadeia de Caracteres Mais Próxima (PCCP). Para resolvê-lo, foi desenvolvido um algoritmo baseado na metaheurística *Simulated Annealing*, denominado HSALS. Para testá-lo, foram utilizados os mesmos problemas-teste de Chimani (2011), porém foram consideradas somente o alfabeto das bases nitrogenadas do DNA.

O algoritmo HSALS, apesar de produzir soluções com qualidade inferior ao do método exato, foi capaz de gerar soluções melhores do que o algoritmo heurístico Ant, de Faro (2010), em quase todos os problemas-teste. Além disso, o tempo computacional requerido pelo HSALS foi muito inferior ao do algoritmo de Faro (2010), mesmo considerando que a quantidade de *GFlops* – número de operações de ponto flutuante por segundo – do processador no qual

o algoritmo HSALS foi testado é cerca de sete vezes superior ao do processador no qual o algoritmo Ant foi testado (Intel, 2017).

Por outro lado, é importante ressaltar a ineficiência dos métodos exatos em resolver problemas de dimensão mais elevada. Nesse sentido, justifica-se a utilização do algoritmo HSALS proposto.

Como trabalhos futuros, pretende-se testar o algoritmo proposto em problemas-teste de dimensões mais elevadas, assim como testar outras estratégias de busca local, geração de solução inicial e ajustes de parâmetros para melhorar seu desempenho.

Referências

- Chimani, M.; Woste, M.; Bocker, S. (2011), “A Closer Look at the Closest String and Closest Substring Problem”, *Proceedings of the Meeting on Algorithm Engineering & Experiments*, 13–24.
- Faro, S.; Pappalardo, E. (2010), “Ant-CSP: An Ant Colony Optimization Algorithm for the Closest String Problem”, *SOFSEM 2010: Theory and Practice of Computer Science: 36th Conference on Current Trends in Theory and Practice of Computer Science*, 370–381.
- Festa, P. (2007), “On some optimization problems in molecular biology”, *Journal Mathematical Biosciences*, 219 - 234.
- Frances, M.; Litman, A. (1997), “On covering problems of codes”, *Journal Theory of Computing Systems*, 113–119.
- Gasieniec, L.; Jansson, J.; Lingas, A. (1999), “Efficient Approximation Algorithms for the Hamming Center Problem”, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 905–906.
- Gomes, F. C.; Meneses, C. N.; Pardalos, P. M.; Viana, G. V. R. (2008), “A Parallel Multistart Algorithm for the Closest String Problem”, *Journal Computers & Operations Research* 3636–3643.
- “Intel Compare Processors”, <https://www.intel.com/content/www/us/en/products/processors/core.html>, Acessado em: 19 de julho de 2017.
- Kelsey, T.; Kotthoff, L. (2010), “The Exact Closest String Problem as a Constraint Satisfaction Problem”, *Procedia Computer Science*, 1062–1071.
- Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P., “Optimization by Simulated Annealing”, *American Association for the Advancement of Science*, 671–680.
- Lancot, J. K.; Li, M.; Ma, B.; Wang, S.; Zhang, L. (1999), Distinguishing String Selection Problems, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 633–642.
- Deng, X.; Li, G.; Wang, L. (2002), “Center and Distinguisher for Strings with Unbounded Alphabet”, *Journal of Combinatorial Optimization*, 383–400.
- Liu, X.; Holger, M.; Hao, Z.; Wu, G. (2008), *Compounded Genetic and Simulated Annealing Algorithm for the Closest String Problem*, 2008 2nd International Conference on Bioinformatics and Biomedical Engineering.
- Meneses, C. N. (2005), “*Combinatorial Approaches for Problems in Bioinformatics*”, Tese de Doutorado, University of Florida.
- Meneses, C. N.; Lu, Z.; Oliveira, C. A. S.; Pardalos, P. M. (2004), “Optimal Solutions for the Closest-String Problem via Integer Programming”, *INFORMS Journal on Computing*, 419-429.
- Nikolaev, A. G.; Jacobson, S. H. (2010), “Simulated Annealing”, in *Handbook of Metaheuristics*, 1–39, Springer US.
- Roman, S. (1992), “*Coding and Information Theory*”, Springer-Verlag, New York.
- Setubal, J.; Medainis, J. (1997), “Introduction to Computational Molecular Biology”, in *PWS Boston*.
- Soleimani-damaneh, M. (2011), “An optimization modelling for string selection in molecular biology using Pareto optimality”, *Journal Applied Mathematical Modelling*, 3887 - 3892.
- Tanaka, S. (2012), “A heuristic algorithm based on Lagrangian relaxation for the closest string problem”, *Computers & Operations Research*, 709–717.