METAHEURÍSTICA *GRASP* NA RESOLUÇÃO DO PROBLEMA DA CADEIA DE CARACTERES MAIS PRÓXIMA

Eduardo Ribas Pinto
Universidade Federal da Paraíba
Departamento de Informática
João Pessoa, PB - Brasil
ducaribas@gmail.com

Lucídio dos Anjos Formiga Cabral
Universidade Federal da Paraíba
Departamento de Estatística
João Pessoa, PB - Brasil
lucidio@de.ufpb.br

Elder Magalhães Macambira
Universidade Federal da Paraíba
Departamento de Estatística
João Pessoa, PB - Brasil
elder@de.ufpb.br

Resumo: Este artigo apresenta algoritmos heurísticos de construção e busca local para o *Greedy Randomized Adaptive Search Procedures* (*GRASP*) com o intuito de resolver o Problema da Cadeia de Caracteres mais Próxima (PCCP). No PCCP deseja-se encontrar uma seqüência de caracteres que mais se aproxime, segundo uma dada métrica, de um dado conjunto de cadeias de caracteres de mesmo tamanho. Em outras palavras, desejamos minimizar a maior distância desta cadeia de caracteres às demais cadeias do conjunto. Nós reportamos experimentos computacionais e comparações entre cada versão proposta para o PCCP.

Palavras-chaves: Problema da Cadeia de Caracteres mais Próxima, Metaheurísticas, *GRASP*.

Abstract: This paper presents several heuristic algorithms for construction and local search phases of Greedy Randomized Adaptive Search (GRASP) to solve the Closest String Problem (CSP). In the CSP the objective is to find a sequence of characters that is closer, according to a given metric, to a set of sequences of the same size. In another words, the objective is to minimize the major distance between this sequence and the other sequences of the set. We report computational results and comparisons between each version proposed for the CSP.

Keywords: Closest String Problem, Metaheuristics, GRASP.

1. Introdução

Na área de biologia molecular pode-se verificar a presença de vários problemas de otimização combinatória. Geralmente, estes problemas apresentam a necessidade de se comparar e encontrar determinadas características presentes em uma dada cadeia de caracteres (ou seqüências) de DNA, RNA ou proteínas. Por exemplo, quando se trabalha com genoma, ou com outra seqüência de aminoácidos, um dos aspectos principais é como determinar as similaridades e as diferenças que ocorrem em duas seqüências quaisquer (veja [12]).

Neste trabalho estudamos um problema de otimização combinatória, denominado Problema da Cadeia de Caracteres mais Próxima (*Closest String Problem*) - PCCP, no qual desejamos determinar uma cadeia de caracteres que mais se aproxime, segundo alguma métrica, de um dado conjunto de

cadeias de caracteres. Em outras palavras, desejamos minimizar a maior distância desta cadeia de caracteres às demais cadeias do conjunto.

O Problema da Cadeia de Caracteres mais Próxima tem sido bastante estudado na literatura. Em [5], os autores provaram que o PCCP é um problema NP-difícil. Em [2], os autores apresentaram um algoritmo exato polinomial de uma versão parametrizada. Neste caso, a distância entre a seqüência a ser determinada e o conjunto dado de seqüências será no máximo uma constante. Em [1] e [6], os autores apresentaram um algoritmo aproximativo, com razão de performance próxima do valor ótimo quando este é suficientemente grande. Em [8], os autores obtiveram um algoritmo $(4/3+\varepsilon)$-aproximado (para $\varepsilon > 0$). Já em [10], os autores apresentaram um esquema de aproximação polinomial para o problema. Finalmente, em [11] os autores propuseram métodos exatos baseados em *branch-and-bound* e três novas formulações de programação linear inteira para o PCCP.

Neste trabalho apresentamos algoritmos heurísticos *GRASP* para resolver o Problema da Cadeia de Caracteres mais Próxima. Foram propostas uma estratégia gulosa para a fase de construção e duas estratégias de perturbação para a fase de busca local do *GRASP*.

Este artigo está organizado em cinco seções. Os tópicos a serem cobertos em cada uma delas são descritos a seguir. Na seção 2, apresentamos uma definição formal do Problema da Cadeia de Caracteres mais Próxima e alguns conceitos básicos importantes. Na seção 3 são descritos os aspectos de implementação relacionados à metaheurística *GRASP*. Por último, na seção 4 apresentamos os resultados computacionais obtidos com o emprego desta metaheurística. Nestes experimentos, testamos várias instâncias geradas de forma aleatória disponíveis na literatura. Algumas considerações finais e trabalhos futuros são discutidos na seção 5.

2. Definição do problema e conceitos basicos

Nesta seção apresentaremos a definição do Problema da Cadeia de Caracteres mais Próxima, porém antes introduziremos algumas notações que possibilitarão uma melhor compreensão do PCCP.

Seja $\Sigma = \{c_1,...,c_k\}$ um conjunto finito de elementos, denominado caracteres, a partir dos quais cadeias de caracteres podem ser construídas. Cada cadeia de caracteres corresponde a uma seqüência $(s_1,...,s_m)$, com $s_i \in \Sigma$, onde Σ é o alfabeto utilizado. O tamanho de uma cadeia de caracteres s, denotado por $|s|$, corresponde ao número de elementos (ou caracteres) na seqüência que compõe a cadeia s. Por exemplo, se $s = (s_1,...,s_m)$ então $|s| = m$.

Várias medidas têm sido propostas para encontrar as similaridades (ou diferenças) entre as seqüências. A mais utilizada é a distância de Hamming. Uma justificativa mais técnica para o uso freqüente da distância de Hamming na comparação de seqüências em biologia molecular, pode ser encontrada em [9]. A distância de Hamming entre duas cadeias de caracteres, denotada por $d_H(a,b)$, tal que $|a| = |b|$, corresponde ao número de posições nas quais as seqüências diferem. Assim, para o alfabeto $\Sigma = \{A,C,G,T\}$ e as cadeias de caracteres a = CATCC e b = CTTGC temos $d_H(a,b) = 2$.

Considere agora um conjunto $S = \{s^1,...,s^n\}$ de cadeia de caracteres, com $|s^i| = m$, deseja-se encontrar uma seqüência t, com $|t| = m$, que minimize o valor da distância k, tal que, $d_H(s^i,t) \leq k$ para cada cadeia de caracteres $s^i \in S$.

Como dito anteriormente, o Problema da Cadeia de Caracteres mais Próxima é NP-difícil. Em [7], os autores demonstraram que para um k fixo, o PCCP pode ser resolvido em tempo polinomial. Nesse trabalho resolvemos o caso geral do PCCP, ou seja, quando o valor de k, que corresponde à distância média entre uma cadeia caracteres e as cadeias de um conjunto é variável.

3. Algoritmo proposto

As metaheurísticas têm se mostrado uma das alternativas mais promissoras para a solução aproximada de problemas de elevado nível de complexidade computacional (NP-completo e NP-difícil). Dentre as metaheurísticas existentes, o *Greedy Randomized Adaptive Search Procedure* (GRASP), proposto por [3], tem se destacado como uma das mais competitivas em termos da qualidade das soluções alcançadas [4].

Neste trabalho apresentamos um estudo experimental da heurística *GRASP* na resolução do PCCP. Em seguida, apresentamos alguns detalhes do algoritmo proposto, como por exemplo, representação da solução e as heurísticas de construção e de busca local.

3.1 Representação de uma solução

O alfabeto utilizado na definição das cadeias de caracteres é representado pelas bases nitrogenadas do DNA, ou seja, $\Sigma = \{A,C,G,T\}$. Assim, uma solução para o Problema da Cadeia de Caracteres mais Próxima corresponde a uma cadeia de caracteres extraída desse alfabeto. Uma solução *s* foi codificada como um vetor de tamanho é igual a *m*.

3.2 Função de avaliação

O objetivo principal do Problema da Cadeia de Caracteres mais Próxima, como já foi dito em seções anteriores, é encontrar uma solução que seja o menos distante possível de um conjunto de cadeias pré-estabelecidas. Intuitivamente, a distância de Hamming entre as cadeias seria o foco principal da função objetivo, porém não é o mais viável, pelo fato de não permitir que se aceite um maior número de vizinhos distintos em busca de uma solução ótima.

Com isso percebeu-se que a melhor forma de evitar que isso ocorra é construir uma função objetivo que utilize o critério da soma das distâncias de Hamming para avaliar se uma solução é viável ou não. Trata-se da soma das distâncias de Hamming de cada cadeia com relação à solução corrente. O critério da soma como função objetivo não foge do propósito principal do PCCP, pois à medida que a soma é reduzida a cada iteração, diminuem-se as distâncias de Hamming com relação à solução corrente, e assim continuamos garantindo a viabilidade das soluções geradas.

3.3 Fase de construção

O propósito desta fase é fornecer, para a busca local, uma solução de boa qualidade. Para alcançarmos este objetivo, nós propomos um algoritmo de construção que obtenha soluções cuja distância de Hamming, com relação às cadeias de caracteres testadas, seja a menor possível.

Na fase de construção, uma solução é iterativamente construída, elemento por elemento. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordenação pré-determinado. Esse processo de seleção é baseado em uma função adaptativa gulosa g: $C \rightarrow \Re$, que estima o benefício da seleção de cada um dos elementos. A componente probabilística do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõe a lista de candidatos. Este subconjunto recebe o nome de Lista de Candidatos Restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração *GRASP*.

A Figura 1 ilustra o pseudocódigo da fase de construção, denominada C1, para o *GRASP*. Foram utilizados os seguintes parâmetros neste algoritmo:

n: número de cadeias de caracteres a serem testados;
m: tamanho de cada cadeia de caracteres;
S: conjunto de cadeia de caracteres, ou seja, $\{s^1, ..., s^n\}$;
α: aleatoriedade da fase de construção;
d: vetor de tamanho *n* que contém as distâncias de Hamming;
D: matriz de 0*s* e 1*s*.

procedimento FaseConstrução (n, m, S, α)
1. $s = \varnothing$; {solução inicial}
2. **para** $k = 1$ **até** n **faça**
3. $C_k = 0$; $d_k = 0$; $LCR_k = 0$;
4. **fim-para**
5. **para** $i = 1$ **até** m **faça**
6. $l_{max} \leftarrow 0$; $l_{min} \leftarrow \infty$;
7. **para** $j = 1$ **até** n **faça**
8. **para** $k = 1$ **até** n **faça**
9. **se** ($S_j^i \neq S_k^i$) **então** $D_j^k = 1$;
10. **senão** $D_k^k = 0$;
11. **fim-se**
12. **se** ($D_j^k + d_k > C_j$) **então** $C_j = D_j^k + d_k$;
13. **fim-para** // k
14. **se** ($C_j > l_{max}$) **então** $l_{max} = C_j$; {filtra os maiores valores de C_j }
15. **se** ($C_j < l_{min}$) **então** $l_{min} = C_j$; {filtra os menores valores de C_j }
16. **fim-para** // j
17. **para** $j = 1$ **até** n **faça**
18. **se** ($l_{min} + ((l_{max} - l_{min}) * \alpha) \geq C_j$) **então** $LCR_j = j$;
19. **fim-para** // j
20. Selecionar aleatoriamente um caractere pertencente a LCR;
21. Atribuir este caractere a cadeia de caracteres s na posição i;
22. $iMax = -1$;
23. **para** $j = 1$ **até** n **faça**
24. $d_j = d_H(s, s^j)$;
25. **se** ($d_j \geq iMax$) **então** $iMax = d_j$;
26. $C_j = 0$;
27. $LCR_j = 0$;
28. **fim-para** // j
29. **fim-para** // i
Retorne ($iMax$, s);

Figura 1. Pseudocódigo da fase de construção do *GRASP*.

3.3 Fase de busca local

Um algoritmo de busca local é utilizado, com o intuito de refinar a solução obtida na fase de construção. A busca local implementada tem um funcionamento simples, e pode ser descrita de acordo com uma estrutura de vizinhança.

A vizinhança define qual o conjunto de soluções para as quais o algoritmo pode se mover a partir da solução atual (ou corrente). Neste trabalho, a vizinhança proposta consiste em alterar uma das posições da cadeia de caracteres, que representa a solução corrente, por um caractere distinto do alfabeto. Com a modificação, o vetor das distâncias de Hamming é modificado e com isso uma nova soma é calculada com base na solução atual.

As versões dos algoritmos implementados para as heurísticas de busca local se diferenciam nos seguintes aspectos:

- Busca local BL1 (determinística): definida através da substituição de um caractere na solução corrente por um caractere, escolhido de forma aleatória, do alfabeto. Além disso, também é escolhida a posição onde será colocado o caractere selecionado. Esta escolha é feita de forma determinística, ou seja, a cada iteração, a posição da solução corrente é incrementada em uma unidade.

- Busca local BL2 (aleatória): diferencia-se da busca local BL1 na escolha da posição da solução corrente, onde haverá a troca de caracteres gerando uma nova solução vizinha. Como foi dito anteriormente, essa escolha se dava de maneira determinística. Na segunda versão, esta escolha se dá de forma aleatória, ou seja, da mesma maneira que é utilizado na escolha do caractere do alfabeto.

Um exemplo de funcionamento da busca local BL1 é explicado em seguida. Considere o alfabeto Σ = {A,C,G,T} e a solução corrente s = (A,T,C). Aplicando a busca BL1 em s, podemos ter os seguintes vizinhos com a escolha aleatória do caractere G em Σ: (G,T,C), (A,G,C) e (A,T,G). O mesmo exemplo pode ser aplicado com a busca local BL2, porém a posição não será mais determinística e sim aleatória.

Além destes procedimentos, foram também propostos duas estratégias de busca intensiva usando o conceito de *first improvement* (FI) e *best improvement* (BI). A busca local que faz uso da estratégia *first improvement*, a solução vizinha escolhida é o primeiro vizinho encontrado, ou seja, o primeiro vizinho, que satisfaça as condições para que este seja aceito, substitui a solução corrente. Já na busca local que faz uso da estratégia *best improvement*, a solução vizinha escolhida é a melhor dentre todos os vizinhos possíveis de uma solução corrente, ou seja, após ter percorrido toda vizinhança, a melhor solução vizinha encontrada passa a ser a nova solução corrente.

Neste trabalho foram implementadas quatro versões da metaheurística *GRASP*. As quatro versões com a mesma fase de construção descritas na seção 3.3. A busca local difere no tipo de vizinhança empregada e na estratégia utilizada. A Tabela 1 descreve as diferenças entre os algoritmos.

Versão do *GRASP*	Construção	Busca Local	Estratégia de Busca
GRASP-1	C1	BL 2	BI
GRASP-2	C1	BL 1	BI
GRASP-3	C1	BL 2	FI
GRASP-4	C1	BL 1	FI

Tabela 1: Configuração das heurísticas *GRASP* propostas.

4. Resultados computacionais

Nesta seção, apresentamos os resultados computacionais obtidos com as metaheurísticas *GRASP*. Todos os algoritmos foram implementados utilizando a linguagem C. Os experimentos computacionais foram realizados em um computador com sistema operacional Windows XP e com processador Pentium IV 2.4 Ghz com 512 MB de memória RAM. Em [11], os autores propuseram instâncias para o PCCP. De acordo com o alfabeto empregado era possível identificar um grupo de instâncias. Os testes deste nosso trabalho utilizaram o alfabeto Σ = {A,C,G,T}, ou seja, as bases nitrogenadas do DNA. Ainda em [11], os autores obtiveram os valores ótimos de cada instância testada através de um algoritmo *branch-and-bound*.

Antes de apresentarmos os resultados obtidos, mencionaremos os valores adotados para os parâmetros da metaheurística *GRASP*. Estabeleceu-se que o número máximo de iterações, denotado *GRASP_MAX*, fosse igual a {n, 5n, 10n}. Os valores escolhidos para α pertencem ao conjunto {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. Foram testadas quatro instâncias para cada par (n,m), com n = {300, 400, 500, 700} e m = {10, 15, 20, 25, 30}, dando um total de 80 instâncias. Cada instância foi executada cinco vezes. Dessas cinco execuções foi calculada a média e o mínimo das soluções x_i encontradas. A partir daí foi calculado os *gaps* desses valores com relação à solução ótima s^*, utilizando-se das seguintes fórmulas:

$$Gap\ médio\ = \frac{\frac{1}{5}\left(\sum_{i=1}^{5} f(x_i)\right)}{f(s^*)} - f(s^*) \qquad (1)$$

$$Gap\ mínimo = \frac{\min\{f(x_i); i = 1,...,5\} - f(s^*)}{f(s^*)} \quad (2)$$

Os resultados obtidos com as versões implementadas do *GRASP* são apresentados nas Tabelas 2-5. O cabeçalho, destas tabelas, possui o seguinte significado: a primeira coluna indica a instância testada, com a indicação dos parâmetros *n*, *m* e um seqüencial sobre o tipo (DNA), as outras colunas indicam os valores obtidos pelas quatro versões do *GRASP*. Cada coluna que corresponde aos resultados dos algoritmos é subdividida em duas colunas que significam, respectivamente, o desvio da média com relação à solução ótima (*gap_med*) e o desvio do valor mínimo obtido com relação à solução ótima (*gap_min*), ambos os valores expressos em termos percentuais. A última linha corresponde às médias dos valores de cada coluna de resultados.

	***GRASP*-1**		***GRASP*-3**		***GRASP*-2**		***GRASP*-4**	
Nome	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***
n10m300tai1.ins	2,97	2,86	2,74	2,29	3,43	2,86	3,09	2,29
n10m300tai2.ins	2,03	1,69	1,58	1,13	1,81	1,69	2,03	1,13
n10m300tai3.ins	1,82	1,70	1,36	1,14	1,59	1,14	1,82	1,70
n10m300tai4.ins	2,30	2,30	1,84	1,72	2,07	1,72	2,18	1,72
n15m300tai1.ins	1,86	1,64	1,97	1,64	2,08	1,64	2,62	2,19
n15m300tai2.ins	2,73	2,73	2,62	2,19	2,30	1,64	2,62	2,19
n15m300tai3.ins	1,93	1,60	1,93	1,60	1,71	1,6	1,93	1,60
n15m300tai4.ins	2,07	1,63	2,17	2,17	1,74	1,63	1,96	1,63
n20m300tai1.ins	2,42	2,11	2,63	2,63	2,11	2,11	2,63	2,63
n20m300tai2.ins	1,88	1,57	2,20	2,09	1,78	1,57	2,30	2,09
n20m300tai3.ins	1,79	1,58	1,68	1,58	1,68	1,58	2,00	1,58
n20m300tai4.ins	1,79	1,05	1,89	1,58	2,00	1,58	2,21	2,11
n25m300tai1.ins	1,94	1,53	2,14	2,04	1,63	1,53	2,55	2,55
n25m300tai2.ins	1,84	1,53	1,84	1,53	1,63	1,53	1,94	1,53
n25m300tai3.ins	2,45	2,04	3,06	2,55	2,55	2,55	3,16	3,06
n25m300tai4.ins	1,95	1,54	2,05	2,05	1,74	1,54	2,36	2,05
n30m300tai1.ins	1,31	1,01	1,62	1,52	1,82	1,52	2,02	2,02
n30m300tai2.ins	2,02	2,02	2,53	2,53	2,22	2,02	2,53	2,53
n30m300tai3.ins	1,51	1,01	1,91	1,51	1,51	1,51	1,71	1,51
n30m300tai4.ins	1,52	1,52	2,02	2,02	1,72	1,52	2,42	2,02
Média	2,01	1,73	2,09	1,88	1,96	1,72	2,30	2,01

Tabela 2: Desempenho dos algoritmos *GRASP* para *m* = 300.

Pode-se observar na Tabela 2 que o algoritmo de melhores *gaps* mínimo e médio foi o *GRASP*-2. Outro aspecto importante que pode ser observado é que os algoritmos com estratégia *best improvement* tiveram melhores resultados quando comparados com os de estratégia *first improvement*.

Na Tabela 3 observa-se que o algoritmo *GRASP*-3, obteve os melhores *gaps* mínimos e médios. Entretanto, vale ressaltar que o algoritmo *GRASP*-2 continua obtendo boas soluções, comprovando seu comportamento bastante robusto quando utiliza cadeias de tamanhos 300 e 400. Na Tabela 4 os algoritmos que utilizam estratégia de vizinhança aleatória obtiveram melhores resultados com relação aos demais, possibilitando concluir que estes aumentam o desempenho à medida que se aumenta o tamanho das cadeias de caracteres.

Os algoritmos com estratégia de vizinhança aleatória, na Tabela 5, continuam obtendo os menores *gaps*. Em todas as tabelas apresentadas, observa-se que os maiores valores são encontrados no algoritmo *GRASP*-4, comprovando sua inferioridade com relação aos demais. É possível verificar também que as melhores versões do *GRASP* correspondem às versões que fazem uso da estratégia *first improvement* e vizinhança aleatória, ou seja, a versão *GRASP*-3. No gráfico da Figura 2 percebe-se que os resultados obtidos pela versão *GRASP*-3, para instâncias com cadeias de tamanho superior a

500, obtiveram, na maioria dos casos, *gaps* mínimos abaixo de 1.5%. Ressalte-se ainda que em todas as versões há um ganho de qualidade das soluções à medida que aumenta o tamanho das cadeias.

	***GRASP*-1**		***GRASP*-3**		***GRASP*-2**		***GRASP*-4**	
Nome	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***
n10m400tai1.ins	2,49	2,15	2,15	2,15	2,58	2,15	2,49	2,15
n10m400tai2.ins	1,80	1,29	1,55	1,29	2,40	2,15	1,80	1,29
n10m400tai3.ins	2,37	2,54	2,20	2,12	2,37	1,69	2,80	2,12
n10m400tai4.ins	1,97	1,71	1,20	0,85	1,79	1,71	1,79	1,71
n15m400tai1.ins	1,98	1,65	1,65	1,23	1,89	1,65	2,22	2,06
n15m400tai2.ins	2,11	2,43	2,02	1,62	2,35	2,02	2,43	2,43
n15m400tai3.ins	1,71	1,63	1,22	1,22	1,30	1,22	1,14	0,81
n15m400tai4.ins	1,87	1,63	1,87	1,63	1,87	1,63	2,11	2,03
n20m400tai1.ins	1,65	1,57	1,42	1,18	1,26	1,18	1,50	1,18
n20m400tai2.ins	1,90	1,98	2,30	1,98	2,22	1,98	2,38	2,38
n20m400tai3.ins	1,50	1,58	1,90	1,58	1,74	1,58	1,98	1,98
n20m400tai4.ins	1,26	1,58	1,19	0,79	1,11	0,79	1,42	1,19
n25m400tai1.ins	1,62	1,54	1,69	1,54	1,54	1,54	1,92	1,92
n25m400tai2.ins	1,70	1,54	1,85	1,54	1,70	1,54	2,55	2,32
n25m400tai3.ins	1,85	1,93	1,78	1,54	1,62	1,54	1,85	1,54
n25m400tai4.ins	1,23	1,15	1,38	1,15	1,30	1,15	1,76	1,53
n30m400tai1.ins	2,37	2,29	2,67	2,67	2,14	1,91	2,75	2,67
n30m400tai2.ins	1,83	1,52	1,90	1,52	1,67	1,52	2,21	1,90
n30m400tai3.ins	1,59	1,52	1,67	1,52	1,52	1,52	1,82	1,52
n30m400tai4.ins	1,76	1,53	2,14	1,91	1,98	1,91	2,52	2,29
Média	1,83	1,74	1,79	1,55	1,82	1,62	2,07	1,85

Tabela 3: Desempenho dos algoritmos *GRASP* para $m = 400$.

	GRASP-1		***GRASP-3***		***GRASP-2***		***GRASP-4***	
Nome	***gap_med***	***gap_min***	***gap_med***	***Gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***
n10m500tai1.ins	2,26	1,71	1,37	1,03	2,53	2,05	1,99	1,71
n10m500tai2.ins	2,01	1,38	1,18	1,04	1,94	1,73	1,73	1,38
n10m500tai3.ins	2,15	1,74	1,53	1,39	2,50	2,08	2,43	2,08
n10m500tai4.ins	2,63	2,42	1,94	1,73	2,28	1,73	2,49	2,08
n15m500tai1.ins	1,90	1,63	1,63	1,63	1,90	1,63	2,16	1,96
n15m500tai2.ins	2,03	1,96	2,03	1,96	2,35	2,29	2,35	1,96
n15m500tai3.ins	1,43	1,30	1,11	0,65	1,37	1,30	1,56	1,30
n15m500tai4.ins	1,95	1,63	1,63	1,30	1,43	1,30	1,89	1,63
n20m500tai1.ins	1,51	1,26	1,58	1,58	1,39	1,26	1,77	1,58
n20m500tai2.ins	1,85	1,59	1,97	1,91	1,85	1,59	2,48	2,23
n20m500tai3.ins	1,65	1,59	1,78	1,59	1,65	1,59	2,16	1,90
n20m500tai4.ins	1,57	1,25	1,69	1,57	1,69	1,57	2,19	2,19
n25m500tai1.ins	1,42	1,24	1,55	1,55	1,49	1,24	1,67	1,55
n25m500tai2.ins	1,25	0,93	1,31	1,25	1,31	0,93	1,81	1,56
n25m500tai3.ins	1,91	1,54	1,91	1,85	1,98	1,85	2,41	2,16
n25m500tai4.ins	1,66	1,54	1,78	1,54	1,72	1,54	2,03	1,85
n30m500tai1.ins	1,52	1,52	1,82	1,82	1,76	1,52	2,06	1,82
n30m500tai2.ins	1,57	1,51	1,81	1,81	1,51	1,51	1,99	1,81
n30m500tai3.ins	1,52	1,21	1,76	1,52	1,39	1,21	1,58	1,52
n30m500tai4.ins	1,60	1,53	1,84	1,84	1,53	1,53	1,84	1,53
Média	1,77	1,53	1,66	1,53	1,78	1,57	2,03	1,79

Tabela 4: Desempenho dos algoritmos *GRASP* para $m = 500$.

	GRASP-1		*GRASP*-3		*GRASP*-2		*GRASP*-4	
Nome	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***	***gap_med***	***gap_min***
n10m700tai1.ins	2,78	2,20	1,80	1,46	2,68	2,44	2,20	1,71
n10m700tai2.ins	2,47	2,22	1,58	1,23	2,86	2,22	2,57	2,22
n10m700tai3.ins	2,70	2,46	1,87	1,47	3,00	2,70	2,41	1,97
n10m700tai4.ins	2,60	1,47	1,18	0,98	2,35	2,21	2,11	1,96
n15m700tai1.ins	1,76	1,39	1,53	1,16	1,76	1,62	1,85	1,62
n15m700tai2.ins	1,97	1,41	1,69	1,41	1,88	1,64	2,07	1,88
n15m700tai3.ins	1,86	1,86	1,44	1,16	1,67	1,63	1,86	1,63
n15m700tai4.ins	1,97	1,41	1,69	1,41	1,88	1,64	2,07	1,88
n20m700tai1.ins	2,18	2,05	2,41	2,27	2,18	2,05	2,50	2,27
n20m700tai2.ins	1,53	1,35	1,31	1,13	1,40	1,35	1,58	1,58
n20m700tai3.ins	1,40	1,36	1,31	1,13	1,45	1,36	1,81	1,58
n20m700tai4.ins	1,54	1,36	1,40	1,36	1,72	1,58	1,72	1,58
n25m700tai1.ins	1,24	1,10	1,50	1,32	1,32	1,32	1,72	1,55
n25m700tai2.ins	1,28	1,10	1,41	1,32	1,28	0,88	1,77	1,55
n25m700tai3.ins	1,28	1,10	1,10	1,10	1,37	1,32	1,55	1,55
n25m700tai4.ins	1,42	1,33	1,47	1,33	1,29	1,11	1,73	1,56
n30m700tai1.ins	1,17	1,09	1,35	1,30	1,30	1,09	1,39	1,30
n30m700tai2.ins	1,26	1,09	1,26	1,09	1,26	1,09	1,57	1,53
n30m700tai3.ins	1,17	1,09	1,35	1,30	1,30	1,09	1,39	1,30
n30m700tai4.ins	1,57	1,52	1,57	1,52	1,70	1,52	1,87	1,74
Média	1,76	1,50	1,51	1,32	1,78	1,59	1,89	1,70

Tabela 5: Desempenho dos algoritmos *GRASP* para $m = 700$.

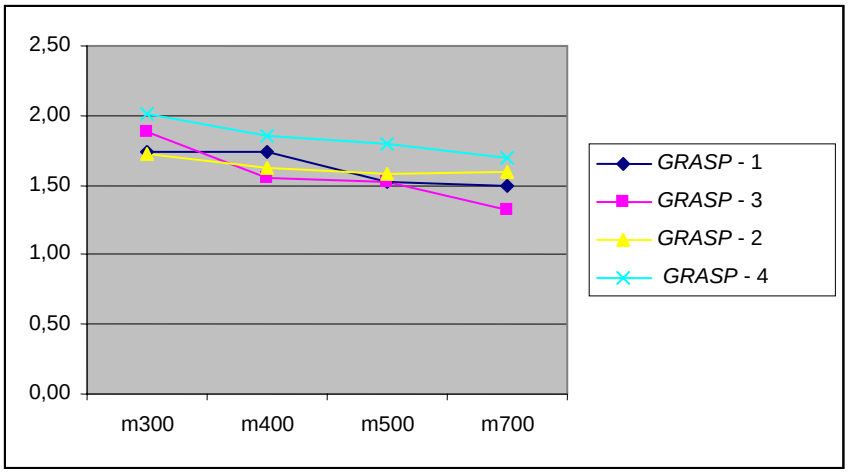

Figura 2: Comportamento dos algoritmos *GRASP* com relação ao tamanho das cadeias.

Para fins de comparação, como dito em seções anteriores, foram utilizados os resultados obtidos por uma heurística da literatura (ver [11]). Os resultados obtidos com o conjunto de instâncias teste são apresentados na Tabela 6. A primeira coluna indica a instância testada, com a indicação dos parâmetros n, m e um seqüencial da instância, a segunda coluna indica os valores dos *gaps* mínimos obtidos por esta heurística. A terceira coluna indica os valores obtidos com a metaheurística *GRASP*-3, que implementa a estratégia *best improvment* e a vizinhança aleatória na busca local. Observa-se que, nesta tabela, houve um ganho substancial obtido pelo algoritmo *GRASP*-3, onde os valores de *gap* mínimo foram bem inferiores aos *gaps* mínimos apresentados pela heurística da literatura em todas as instâncias testadas.

Nome	Heurística [11] (*gap_min*)	*GRASP*-3 (*gap_min*)
n10m300tai1.ins	3,85	**2,86**
n10m300tai2.ins	3,8	**1,69**
n10m300tai3.ins	3,83	**1,7**
n10m300tai4.ins	4,4	**2,3**
n10m400tai1.ins	4,51	**2,15**
n10m400tai2.ins	4,12	**1,29**
n10m400tai3.ins	4,07	**2,54**
n10m400tai4.ins	3,7	**1,71**
n15m300tai1.ins	5,18	**1,64**
n15m300tai2.ins	4,19	**2,73**
n15m300tai3.ins	4,59	**1,6**
n15m300tai4.ins	3,66	**1,63**
n15m400tai1.ins	5,08	**1,65**
n15m400tai2.ins	4,26	**2,43**
n15m400tai3.ins	4,65	**1,63**
n15m400tai4.ins	4,65	**1,63**
n20m300tai1.ins	4,52	**2,11**
n20m300tai2.ins	4,98	**1,57**
n20m300tai3.ins	4,04	**1,58**
n20m300tai4.ins	4,52	**1,05**
n20m400tai1.ins	4,51	**1,57**
n20m400tai2.ins	2,33	**1,98**
n20m400tai3.ins	4,89	**1,58**
n20m400tai4.ins	5,24	**1,58**
n25m300tai1.ins	4,39	**1,53**
n25m300tai2.ins	4,39	**1,53**
n25m300tai3.ins	5,31	**2,04**
n25m300tai4.ins	4,88	**1,54**
n30m300tai1.ins	4,35	**1,01**
n30m300tai2.ins	5,26	**2,02**
n30m300tai3.ins	4,78	**1,01**
n30m300tai4.ins	4,81	**1,52**

Tabela 6: Resultados obtidos com o algoritmo *GRASP*-3 e com a heurística da literatura.

5. Conclusão

A partir dos resultados apresentados na seção 4, pode-se concluir que os algoritmos propostos para o Problema da Cadeia de Caracteres mais Próxima demonstraram serem eficientes na resolução do problema em questão.

As soluções obtidas pelos algoritmos implicaram em melhorias na qualidade das soluções, principalmente, se comparadas com as soluções obtidas pela heurística existente na literatura (vide [11]). Entretanto, é importante ressaltar que o tempo de processamento gasto, principalmente pelo algoritmo *GRASP* é um pouco alto se comparado com esta outra heurística, porém o compromisso entre a qualidade da solução obtida e o tempo computacional fica a favor dos quatro algoritmos *GRASP* propostos.

Vale ressaltar que os resultados obtidos apresentaram um *gap*, na grande maioria das vezes, inferior a 2%, logo, pode-se verificar a robustez do algoritmo. Conclui-se então, que o algoritmo se

mostrou bastante robusto, ou seja, sem grandes diferenças de desempenho entre as diferentes instâncias testadas.

Embora os algoritmos propostos tenham apresentado bons desempenhos, há ainda um *gap* a ser conquistado. Daí outras metaheurísticas estão sendo avaliadas tendo em mente alcançar uma melhor relação entre tempo de processamento e qualidade da solução obtida.

Agradecimentos. Os autores agradecem ao CNPq (No. do processo 47165/04-2) pelo financiamento deste trabalho.

6. Referências Bibliográficas

[1] Ben-Dor, A., Lancia, G., Perone, J. and Ravi, R. Banishing Bias from Consensus Sequences, *Combinatorial Pattern Matching, 8th Annual Symposium,* Springer-Verlag,Berlin, 1997.

[2] Berman, P., Gumucio, D., Hardison, R., Miler, W. and Stojanovic, N. A linear-time algorithm for the 1-mismatch problem, *Workshops on Algorithms and Data Structures,* pp. 126-135, 1997.

[3] Feo, T.A. e Resende, M.G.C. Greedy randomized adaptive search procedures, *Journal. of Global Optimization,* 6, 109-133, 1995

[4] Festa, P. e Resende M. G. C. An annotated bibliography of GRASP, AT&T Labs Research Technical Report TD-5WYSEW, February, 2004 (50 pages).

[5] Frances, M. e Litman, A. On covering problems of codes, *Theor. Comput. Systems*, 30:113-119, 1997.

[6] Gasieniec, L., Jansson, J., Lingas, A. Efficient approximations algorithms for the hamming center problem, *In Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, S905-S906, 1999.

[7] Gramm, J., Niedermeier, R. e Rossmanith, P. Exact solutions for closest string and related problems, *In Proc. of the 12th Annual International Symposium on Algorithms and Computation (ISAAC)*, Lectures Notes in Computer Science, 2223:441-452,Springer-Verlag, 2001;

[8] Lanctot, K., Li, M., Ma, B., Wang, S., Zhang, L. Distinguishing string selection problems, *In Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, 633-642, 1999.

[9] Li, M., Ma, B. e Wang, L. Finding regular regions in many strings, *In Proc. of the Thirty First Annual ACM Symposium on Theory of Computing*, 473-482, Atlanta, 1999.

[10] Li, M., Ma, B. and Wang, L. On the closest string and substring problems, *Journal of theACM*, 49 (2): pp. 157-171, 2002.

[11] Meneses, C N, Lu, Z, Oliveira, C A S, and Pardalos, P M, Optimal Solutions for the Closest-String Problem via Integer Programming, *INFORMS Journal on Computing* 16(4): pp. 419-429, 2004.

[12] Setúbal, J. e Meidanis, J, *Introduction to Computational Molecular Biology,* PWS Boston, 1997.