Uma abordagem baseada no *Simulated Annealing* para o Problema da Sequência Mais Próxima

Diego Franck José

JUIZ DE FORA

DEZEMBRO, 2014

Uma abordagem baseada no *Simulated Annealing* para o Problema da Sequência Mais Próxima

Diego Franck José

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Stênio Sã Rosário Furtado Soares

JUIZ DE FORA

DEZEMBRO, 2014

Uma abordagem baseada no *Simulated Annealing* para o Problema da Sequência Mais Próxima

Diego Franck José

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Stênio Sã Rosário Furtado Soares
Dr. em Computação - UFF - RJ

Luciana Brugiolo Gonçalves
Dra. em Computação - UFF - RJ

Luciana Conceição Dias Campos
Dra. em Engenharia Elétrica - PUC - RJ

JUIZ DE FORA

11 DE DEZEMBRO, 2014

Resumo

O Problema da Sequência Mais Próxima (PSMP) encontra aplicações na área de Biologia Computacional e tem despertado o interesse da comunidade científica nas últimas décadas. Este trabalho apresenta uma aplicação da meta-heurística *Simulated Annealing* ao problema. Resultados computacionais são apresentados de forma a comparar a abordagem proposta com um Algoritmo Genético da literatura.

Palavras-chave: Problema da Sequência Mais Próxima, DNA, Simulated Annealing, Busca Local, Heurística de Construção.

Abstract

The Closest String Problem (CSP) finds applications in Computational Biology area and has raised the interest of the scientific community in the last decades. This work presents an application of the Simulated Annealing meta-heuristic to the problem. Computational results are presented in order to compare the proposed approach with a Genetic Algorithm from literature.

Keywords: closest string problem, DNA, simulated annealing, local search, construction heuristic.

Agradecimentos

Agradeço primeiramente à Deus pelas oportunidades concedidas e pela força para chegar ao fim de mais uma caminhada.

Aos meus pais, Luiz Alberto e Márcia, pelo amor, confiança, suporte, incentivo e apoio irrestrito em todas as decisões tomadas. Ao meu irmão Daniel pela cumplicidade e amizade em todos os momentos.

À minha namorada Tharcilla, pelo amor, carinho e paciência, e que mesmo distante me lembrava constantemente das minhas obrigações, não me deixando esmorecer em momento algum.

A todos os meus parentes e amigos pelo encorajamento e apoio.

Ao professor Stênio pela orientação, sabedoria e conhecimentos compartilhados, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o meu enriquecimento pessoal e profissional.

"Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir".

Steve Jobs

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
DNA	Ácido Desoxirribonucleico
RNA	Ácido Ribonucleico
PSMP	Problema da Sequência Mais Próxima
CSP	*Closest String Problem*
GRASP	*Greedy Randomized Adaptive Search Procedure*
VNS	*Variable Neighbourhood Search*
RC	Reconexão de Caminhos
SA	*Simulated Annealing*

1 Introdução

1.1 Apresentação do tema e contextualização do problema

A descoberta da estrutura do DNA em 1953 por Watson e Crick fez surgir um número significativo de pesquisas na área de Biologia Molecular, com atenção especial no estudo dos ácidos nucléicos. Conforme cita (Yamamoto, 2004), o problema mais conhecido nesta área é o sequenciamento completo do genoma dos organismos. O genoma é um código genético, que possui toda a informação hereditária de um ser, codificada no DNA.

Atualmente, um dos grandes desafios da pesquisa em Computação no Brasil é conseguir gerenciar o grande volume de dados que é gerado todos os dias no estudo do Genoma, de forma que o armazenamento, tratamento, filtragem e acesso a esses dados compartilhados seja facilitado. Em consequência dessa imensa massa de dados, os problemas práticos relevantes a esse tema se tornam altamente combinatórios, sendo, portanto, fundamental o desenvolvimento de técnicas eficientes para sua resolução.

O processo de sequenciamento do DNA consiste em extrair deste as sequências de pares de bases nitrogenadas. Este processo demanda um esforço computacional acentuado devido ao volume de dados envolvido. Conforme visto em (Setubal *et al.*, 1997), o genoma humano, por exemplo, tem aproximadamente 10^8 pares de bases, sendo que, a maior sequência obtida em laboratório apresenta aproximadamente 700 pares de base. A partir deste exemplo, pode-se perceber a limitação das técnicas em laboratório se considerarmos o genoma como um todo.

Segundo (Lyra, 2012), em muitos problemas de biologia molecular com natureza combinatória existe a necessidade de se buscar por regiões comuns ou subcadeias de caracteres que sejam similares a outras sequências de DNA, RNA ou proteínas. Um desses problemas recebe o nome de Problema da Sequência Mais Próxima (PSMP), também conhecido na literatura como *Closest String Problem* (CSP), onde deseja-se determinar,

segundo alguma métrica definida, a sequência que mais se aproxima de um conjunto de sequências dadas como entrada. De acordo com (Lanctot *et al.*, 2003), uma das aplicações do mesmo é encontrada no desenvolvimento de drogas que possuem estruturas genéticas semelhantes em sua composição.

Para (Yamamoto, 2004), dado o volume de informação tratado, a Biologia Molecular, por si só, não seria capaz de lidar com os desafios do sequenciamento genético. Abrindo, portanto, um amplo campo de pesquisa na área de Otimização Combinatória, o que envolve a necessidade de se desenvolver técnicas exatas e aproximadas que possibilitem a resolução de problemas de sequenciamento.

Resumidamente, (Yamamoto, 2004) descreve o problema da seguinte forma: Seja $S = \{S_1, S_2, ..., S_m\}$ um conjunto de sequências (todas de tamanho n) sobre um alfabeto Σ. O objetivo é encontrar uma sequência S_H de tamanho n que minimize a maior distância d entre S_H e $S_i, \forall i = 1...m$, ou seja, deseja-se minimizar d, onde $d(S_H, S_i) \leq d$, para todo $i = 1...m$.

Neste trabalho foram desenvolvidas três heurísticas de busca local e três heurísticas de construção para o PSMP. A partir destas heurísticas, foi proposta uma heurística *Simulated Annealing* para o mesmo. Os resultados obtidos foram comparados com um Algoritmo Genético da literatura.

O restante do trabalho está assim organizado: o Capítulo 2 apresenta os conceitos e termos relacionados ao problema no que tange a área de Biologia Molecular relacionada ao PSMP, enquanto o Capítulo 3 descreve formalmente o problema e analisa os trabalhos relacionados. O Capítulo 4 descreve as técnicas utilizadas e os algoritmos desenvolvidos para a resolução do problema. Finalmente, nos Capítulos 5 e 6 são apresentados os resultados comparativos dos testes realizados e a conclusão do trabalho, respectivamente.

2 Fundamentação Teórica

Na literatura correlata, pode-se encontrar diversos trabalhos que abordam o tema sob diferentes perspectivas. Alguns destes trabalhos traçam um panorama geral sobre o tema e outros são compostos de abordagens específicas para o mesmo.

Para uma melhor compreensão da linguagem própria da área, são apresentados neste capítulo os conceitos mais relevantes relacionados ao tema.

2.1 Ácidos Nucléicos (DNA e RNA)

2.1.1 DNA

Conforme define (Cantor & Smith, 2002), o DNA (*deoxyribonucleic acid*), ou, em português, ácido desoxirribonucleico, é composto por uma fita dupla de cadeias de nucleotídeos, dando forma a uma estrutura em formato de hélice. O DNA é uma das moléculas fundamentais de toda a vida conhecida. Cada um desses nucleotídeos é constituído por uma molécula de açúcar, com cinco átomos de Carbono em sua constituição (numerados de 1 a 5), além de um fosfato e uma base nitrogenada.

Cada átomo de carbono presente na molécula de açúcar está ligado a uma das bases nitrogenadas a seguir: Adenina (A), Citosina (C), Guanina (G) ou Timina (T). Estas bases são responsáveis pela ligação entre as duas fitas (formando a dupla fita do DNA) e possuem algumas regras para que esta ligação possa ocorrer: A base A está sempre ligada à uma base T e a base C está sempre ligada à uma base G.

2.1.2 RNA

Ainda segundo (Cantor & Smith, 2002), o RNA (sigla para *ribonucleic acid*, em português ácido ribonucleico), é, ao contrário do DNA, formado por uma cadeia simples de nucleotídeos e não apresenta uma estrutura helicoidal em sua fita simples. Seus nucleotídeos contém as bases Adenina (A), Guanina (G), Citosina (C) e Uracila (U), que

substitui a Timina presente no DNA.

2.2 Comparação de sequências

Dentro da Biologia Molecular, algumas operações são necessárias para realizar a comparação de sequências. Como exemplo, destaca-se:

Comparação → Similaridade é uma métrica que define o quanto duas sequências são semelhantes e pode ser calculada atribuindo valores na comparação de cada posição das sequências, sendo uma das formas mais utilizadas para comparar sequências de DNA ou RNA.

2.2.1 Distância de *Hamming*

Define-se distância de *Hamming* como uma métrica utilizada para a distância ou diferença entre sequências de mesmo tamanho. Sejam duas sequências s_1 e s_2. A distância de *Hamming* é calculada comparando-se cada posição das sequências, contando-se o número de posições correspondentes em s_1 e s_2 em que as mesmas diferem.

A figura 2.1 representa a utilização da métrica para duas cadeias de DNA que tem o mesmo tamanho e diferem de dois nucleotídeos nas posições 2 e 8, tendo, portanto, uma distância de *Hamming* igual a dois.

	1	2	3	4	5	6	7	8	9	10	11
s1	A	A	G	T	A	C	A	T	T	G	C
s2	A	G	G	T	A	C	A	C	T	G	C

Figura 2.1: Exemplo do cálculo da Distância de *Hamming*

Esta métrica é eficiente para comparar strings que sofram apenas substituições de caracteres em relação a outras strings. Dessa forma, uma aplicação é comparar sequências de cadeias de DNA.

2.3 Heurística de Construção

Uma heurística de construção é um algoritmo que retorna uma solução após um determinado número de passos e que satisfaz um conjunto de regras de construção previamente

definido.

Conforme visto em em (Schneider & Kikrpatrick, 2006), as heurísticas de construção ou heurísticas de aumento são geralmente o caminho mais rápido para chegar a soluções viáveis para um problema proposto. Tais soluções normalmente não são tão boas quanto as soluções fornecidas por outras heurísticas e, por conta disso, são utilizadas principalmente se a solução inicial para tem de ser encontrada rapidamente e repassada para as heurísticas de melhoria, que serão as responsáveis por otimizar essa solução gerada.

Ainda em (Schneider & Kikrpatrick, 2006) são citadas as fases que um algoritmo baseado em uma heurística construtiva geralmente apresenta. Começa-se com uma fase de inicialização, seguida por um laço sobre as fases de seleção dos elementos até que as condições pré-determinadas sejam atendidas, concluindo com uma fase de filtragem e/ou limpeza da solução que será retornada.

2.4 Busca Local

Como visto em (Talbi, 2009), a Busca Local é a uma metaheurística para resolver problemas de otimização e o seu funcionamento é descrito no Algoritmo 1.

Algoritmo 1: Pseudocódigo de uma função de busca local

```
Entrada: Solução inicial S_0
Saída: Solução final encontrada S_best
início
    S_best ← S_0;
    enquanto não satisfeita a condição de parada faça
        S_aux ← GeraPerturbacaoVizinhanca(S_aux);
        se SatisfazCondicaoDeAceite(S_best, S_aux) então
            S_best ← S_aux;
        fim se
    fim enqto
    retorna S_best;
fim
```

Inicia recebendo uma dada solução e, a cada iteração, realiza perturbações na solução atual, gerando assim um novo vizinho (solução que é possível se alcançar através de mudanças na solução base) e que pode melhorar a função objetivo. A busca termina quando uma condição de parada predeterminada é satisfeita e a solução alvo é retornada, ou seja, quando um ótimo local é atingido, como observado na Figura 2.2.

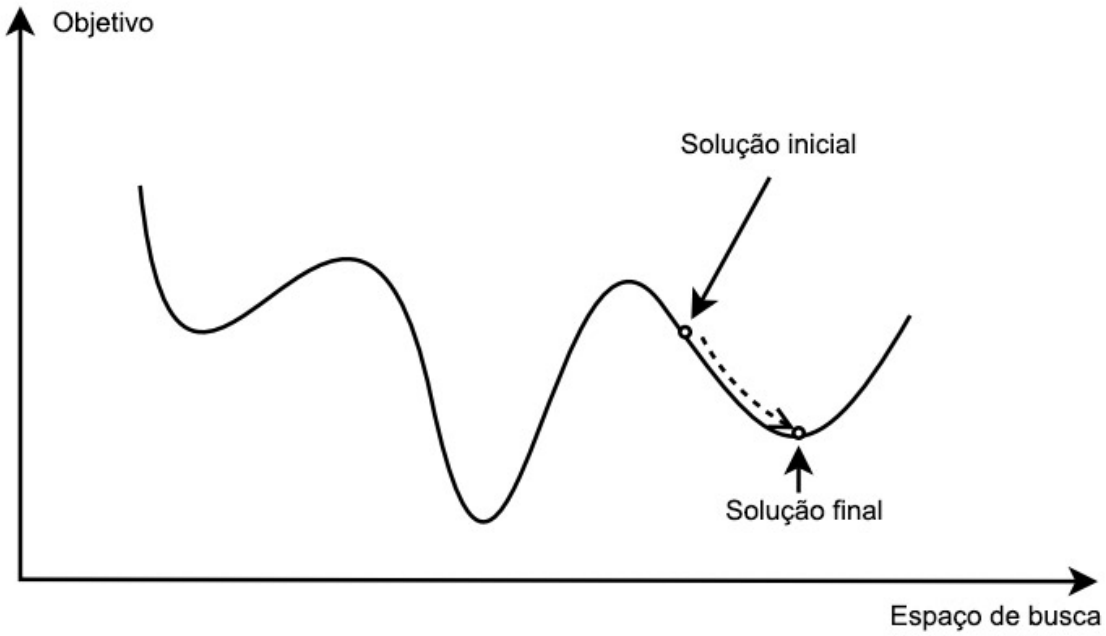

Figura 2.2: Comportamento da Busca Local para um problema de mimização (traduzido de (Talbi, 2009)).

3 O Problema da Sequência Mais Próxima (PSMP)

O Problema da Sequência Mais Próxima é um problema da área de teoria da codificação e consiste em encontrar uma sequência que está o mais próximo possível de cada uma das sequências de um dado conjunto.

Dadas duas sequêncis X e Y quaisquer, de mesmo comprimento, usa-se $d(X, Y)$ para indicar a distância de *Hamming* entre elas.

O Problema da Sequência Mais Próxima é definido do seguinte modo:

Considere um alfabeto finito Σ. Seja s uma sequência finita formada pelos caracteres presentes no alfabeto Σ , onde $|s|$ denota o tamanho de s e s[i] o i-ésimo caractere de s. Seja ainda um conjunto $S = \{s_1, s_2, ..., s_m\}$ de sequências (todas de tamanho n), sobre Σ. O probema consiste em encontrar uma sequência s_H de tamanho n que minimize d, onde, para cada $s_i \in S$, $d_H(s_H, s_i) \leq d$, para $i = 1..m$.

Formalmente, tem-se o modelo matemático do PSMP dado por:

Função objetivo:

$$min\ d \tag{3.1}$$

sujeito a:

$$dH(s_H, s_i) \leq d\ , \qquad \forall s_i \in S \tag{3.2}$$

$$s_H \in \Sigma^n \tag{3.3}$$

Onde Σ^n representa o conjunto de todas as sequências com n caracteres definidas sobre o alfabeto Σ.

Como exemplo, seja $S = \{AGTA, GCTG, ACTA, CGTG, TCGA\}$. A sequência que resolve o problema é a sequência $x = CCTA$, em que, no modelo matemático apresentado, d é minimizada quando a métrica utilizada é a distância de *Hamming*, que nesse caso possui o valor 2.

3.1 Trabalhos relacionados

Nas últimas duas décadas o PSMP tem sido objeto de diversos estudos e publicações. Na década de 90, o problema recebeu sua classficação em relação à complexidade quando Frances & Litman (1997) provaram que o mesmo é NP-difícil. Na mesma década Berman *et al.* (1997), apresentaram um algoritmo onde a distância d (que se deseja determinar) e o conjunto de sequências é constante.

Nos anos 2000, Pardalos *et al.* (2004) propuseram três formulações de programação linear inteira para o problema, enquanto Li *et al.* (2002) apresentaram um esquema de aproximação polinomial para o PSMP.

Conforme já citado, diversos trabalhos presentes na literatura apresentam abordagens distintas que procuram resolver o problema descrito acima, buscando melhorias tanto na qualidadde das soluções quanto no tempo necessário para gerá-las.

Diante disso, o presente capítulo visa mostrar os trabalhos relacionados ao estudo

proposto, apresentando as técnicas utilizadas por (Lyra, 2012) (GRASP) e (Soares *et al.*, 2013) (Algoritmo Genético), sendo este último utilizado para comparação com o trabalho proposto.

3.1.1 GRASP (*Greedy Randomized Adaptive Search Procedure*)

A utilização de metaheurísticas para a resolução de problemas de elevevado nível de complexidade computacional tem se apresentado como uma alternativa bem promissora. Dentre as existentes, o *Greedy Randomized Adaptive Search Procedure* (GRASP), proposto por (Feo & Resende, 1995), tem se destacado como uma das mais competitivas em termos da qualidade das soluções alcançadas. Conforme citado por (Mateus *et al.*, 2009), esta metaheurística tem se mostrado muito eficaz, produzindo as melhores soluções conhecidas para muitos problemas, apesar de sua simplicidade e facilidade de implementação.

Como base de estudo foi utilizado o trabalho (Lyra, 2012), onde é apresentado um algoritmo GRASP para o PSMP em que, inicialmente, como heurística de construção, opta por utilizar a estratégia de atribuir para a solução os caracteres que possuem maior incidência em cada posição do conjunto de entradas.

Posteriormente, a autora apresenta um novo algoritmo construtivo que, diferentemente do anterior, a escolha dos caracteres passou a ser feita de maneira aleatória. Essa escolha randômica foi guiada pelo Método da Roleta, proposto por (Michalewicsj, 1996), onde cada caractere recebe uma probabildade de ser escolhido de acordo com a sua relevância (ou número de ocorrências) em um conjunto de dados passado como entrada, na coluna correspondente preenchida naquele momento.

Visando a melhoria das soluções geradas pela heurística de construção, Lyra (2012) desenvolveu uma Busca Local baseada na metaheurística VNS (*Variable Neighborhood Search*), proposta por (Hansen, 1997), que consiste na exploração do espaço de soluções através das vizinhanças adjacentes à solução observada atualmente. A cada iteração, o algoritmo verifica se houve melhora na solução gerada e, em caso afirmativo, continua explorando a mesma vizinhança. Caso contrário, ele passa a explorar a próxima vizinhança, até que o critério de parada definido seja satisfeito.

Glover em (Glover, 1996) define um método chamado de Reconexão de Caminhos (RC), cuja premissa básica é de que entre duas soluções de boa qualidade pode haver uma ainda melhor. O método consiste em explorar os possíveis caminhos que conectam uma solução de alta qualidade à uma solução base, gerando um caminho na vizinhança dessa solução na direção de uma outra, que recebe o nome de solução alvo. A Figura 3.1 ilustra o método.

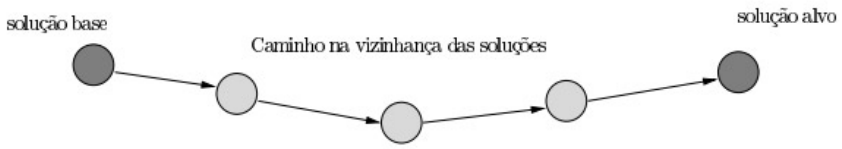

Figura 3.1: Reconexão de Caminhos (Lyra, 2012)

Em (Lyra, 2012) a reconexão de caminhos é feita sempre da melhor solução (base) para a de pior qualidade (alvo). Os caminhos são gerados através da introdução de características da solução base na solução alvo, analisando a cada iteração todos os movimentos que foram realizados e escolhendo aqueles que geraram os melhores resultados.

3.1.2 Algoritmo Genético

Os Algoritmos Genéticos (uma abordagem pertencente à classe dos algoritmos evolucionários) são descritos por (Goldberg *et al.*, 1989) como um processo que simula a seleção natural e evolução de uma população. Como base de estudo dessa abordagem foram utilizados os algoritmos desenvolvidos em (Soares *et al.*, 2013).

Através de operadores de seleção, cruzamento e mutação, o algoritmo simula o processo de seleção natural, onde indivíduos mais aptos tendem a transferir suas características para os novos indivíduos gerados. Cada solução do problema é considerada um indivíduo, que é codificado por uma estrutura composta por elementos associados ao problema.

Para gerar os indivíduos da população inicial (Soares *et al.*, 2013) se baseiam na frequência em que os caracteres ocorrem em cada posição do conjunto de cadeias da entrada, associando à cada caractere uma probabilidade de acordo com sua relevância na coluna observada atualmente. Para cada posição da sequência alvo, um sorteio é realizado e um elemento é selecionado através da estratégia da roleta, utilizada também por (Lyra, 2012). Ao final do processo a solução gerada é incluída na população.

Com o objetivo de inserir algumas características observadas nas melhores soluções nas demais soluções da população, (Soares *et al.*, 2013) propõem a criação de uma cadeia consenso de sequências. Para determinar uma cadeia consenso w associada a um conjunto de sequências S, deve-se associar, para cada posição de w, aquele caractere do alfabeto que possui maior ocorrência entre as sequências de S. Se necessário, existe também a possibilidade de computar uma subcadeia consenso w', onde são determinados os componentes de w apenas para um intervalo da sequência.

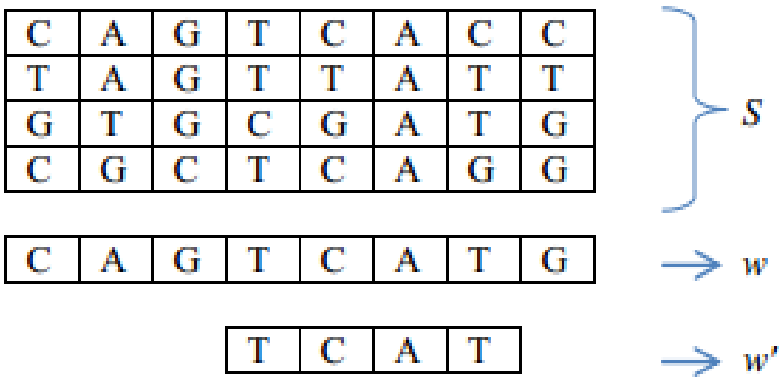

Figura 3.2: Exemplo da cadeia e subcadeia consenso do Algoritmo Genético. (Soares *et al.*, 2013)

Visando a obtenção de soluções que sejam compostas pela combinação de características provenientes de diferentes soluções de boa qualidade, (Soares *et al.*, 2013) utilizam, assim como (Lyra, 2012), o método da Reconexão de Caminhos (RC), que inicia com duas soluções e explora o caminho no espaço de vizinhança entre estas em busca de soluções ainda melhores.

Para intensificar a exploração dessas vizinhanças, (Soares *et al.*, 2013) desenvolveram uma estratégia de busca local que faz uso de janelas deslizantes ao longo de uma cadeia de solução s. A estratégia visa substituir, de forma iterativa, os caracteres da cadeia s pelos caracteres encontrados no mesmo intervalo da cadeia consenso w. O tamanho

t deste intervalo é obtido através de um sorteio, com os possíveis valores inseridos no conjunto $\{1, 2, ..., m\}$. A partir do valor de t, o algoritmo realiza as trocas dos t caracteres, na sequência da solução atual, a partir da posição inicial até a posição $m - t$. Ao final do processo a melhor solução é utilizada, a solução incumbente é atualizada e o processo é reiniciado. Se nenhuma troca resultar em melhora, a busca local termina.

Para gerar a população das iterações subsequentes (Soares *et al.*, 2013) preservam as melhores soluções anteriores. Para garantir diversidade na população, as piores soluções da iteração atual são substituídas por novas soluções geradas da mesma forma que as soluções da população inicial. O restante da população é obtido a partir das operações realizadas na execução do algortimo.

Figura 3.3: Atualização do conjunto Elite, utilizado para manter as melhores soluções geradas. (Soares *et al.*, 2013)

4 Abordagens desenvolvidas

Visando solucionar o Problema da Sequência Mais Próxima descrito anteriormente, algumas heurísticas de construção e de busca local foram implementadas e posteriormente incorporadas ao algoritmo *Simulated Annealing* proposto.

Para uma melhor compreensão das abordagens aqui apresentadas, considere uma instância representada por uma matriz de sequências, denominada M e composta por n sequências (linhas) de tamanho m (colunas). Cada sequência alvo (ou solução) possui o mesmo tamanho m das sequências de matriz M. Considere também uma alfabeto formado por todos os caracteres distintos que estão presentes nas cadeias da matriz M.

4.1 Heurísticas de Construção

Abordagem onde uma ou mais soluções são construídas elemento a elemento, seguindo algum critério, até que se tenha uma solução viável.

4.1.1 *Random Method* (RM)

A primeira abordagem desenvolvida é também a mais simples. A função RM funciona percorrendo cada uma das m colunas da solução que está sendo gerada, selecionando para cada posição o caractere que corresponder ao elemento da posição M$[i, k]$ da matriz base. Onde, k é um número aleatório no intervalo [0, n] e corresponde a linha (ou cadeia do arquivo de entrada) e i ao índice da coluna que está sendo preenchida no momento.

Algoritmo 2: Método Randômico

```
Entrada: objeto de solução
Saída: objeto de solução com cadeia preenchida
início
    para i ← 0 até m faça
        k ← Sorteio(0, n);
        sol[i] ← matriz_base[k][i];
    fim para
fim
```

Para realizar esta escolha aleatória, a probabilidade de um caractere ser selecionado é proporcional ao número de ocorrências deste na coluna correspondente.

Observe que essa forma de montar a solução passa a falsa impressão de que estão sendo geradas somente boas soluções, pois o algoritmo (quase sempre) faz a seleção dos caracteres mais comuns em cada coluna da matriz. Porém, esse método retornará a pior solução possível quando existir uma linha da matriz com uma sequência de caracteres que difere de todas as outras, fazendo com que a solução gerada possua o custo máximo, ou seja, $d_H = m$, conforme a Figura 4.1.

4.1.2 *Random Adaptative Method* (RAM)

Buscando evitar a armadilha do pior caso que poderia ocorrer na primeira abordagem puramente randômica (RM), uma segunda abordagem foi desenvolvida com o objetivo de

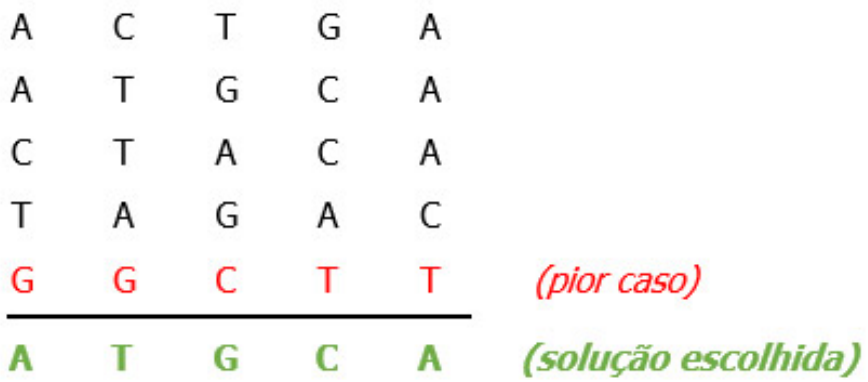

Figura 4.1: Exemplo do pior caso para o método randômico

adicionar um maior controle ao direcionar o simples sorteio que era realizado na abordagem anterior. Esse novo método foi nomeado Método Randômico Adaptativo (RAM).

Semelhante às abordagens descritas por (Lyra, 2012) e (Soares *et al.*, 2013), após escolher a posição que será preenchida da cadeia, de forma sequencial, da primeira até a última posição, o método RAM seleciona o caractere que irá ocupá-la através do Método da Roleta, onde cada caractere recebe uma probabilidade de ser escolhido de acordo com a Equação (4.1), ao invés de simplesmente sortear um número que corresponde à um caractere no vetor do alfabeto, onde nK corresponde ao número de ocorrências do caractere n na coluna K e nB é o número de vezes que esse mesmo caractere n aparece no conjunto de sequências passado como parâmetro para o problema.

$$p = \frac{nK}{nB} \tag{4.1}$$

Após cada iteração, as probabilidades do caractere selecionado anteriormente diminuem, fazendo com que a cada execução as chances de cada caractere ser selecionado fiquem mais próximas, evitando que um caractere muito comum na base, e inicialmente com probabilidade alta, seja sempre selecionado em detrimento do restante do alfabeto.

Essa forma de criar a cadeia de solução tende a ser melhor que o método puramente aleatório, já que o caractere selecionado, na maior parte dos casos, é aquele que é ao mesmo tempo o mais comum em cada coluna e em toda base, possibilitando que o custo da cadeia gerada seja menor que o custo encontrado no algoritmo anterior.

4.1.3 *Random Adaptative Restrictive Method* (RARM)

Apesar da abordagem anterior (RAM) ter diminuído o custo das soluções que eram geradas pelo método randômico, os custos ainda estavam elevados para as necessidades de uma boa heurística de construção. Somado à isso, a maneira de escolher os caracteres que seriam preenchidos (de forma sequencial, do primeiro ao último) não se mostrou a melhor para esse caso.

O problema dessa forma de preenchimento se deve ao fato de que colunas que possuem uma diferença grande entre seus caracteres mais e menos comuns e que possivelmente são relevantes para o custo atual da solução, devem ter seus caracteres definidos antes das colunas onde essa diferença não se mostra tão relevante. De forma a resolver o impasse das colunas levantado no paragráfo anterior, o algoritmo do *RAM* foi modificado.

A primeira alteração realizada foi a ordenação das colunas através da verificação entre as diferenças de seus caracteres mais e menos comuns, de maneira decrescente. Em

seguida uma lista restrita (representada por um vetor) de colunas candidatas foi criada, com tamanho definido por um parâmetro *alfa* que poderia receber valores entre 0.1 e 0.5 (equivalente a 10% e 50%, respectivamente do total da lista de colunas). Posteriormente essa lista restrita era preenchida com a lista de colunas ordenadas e a cada iteração do programa, uma coluna era escolhida pelo *Método da Roleta*, tendo seu caractere selecionado para aquela posição. Ao fim de cada iteração, o algoritmo gera uma nova lista restrita de candidatos e escolhe um novo caractere para as colunas ainda vazias da solução inicial. Esse processo se repete até que toda a cadeia retornada pela heurística de construção *RARM* esteja preenchida.

Essa abordagem foi a escolhida para gerar as soluções iniciais passadas posteriormente ao algoritmo do *Simulated Annealing* por conta dos resultados obtidos nos testes. Apesar de em alguns testes realizados ele possuir valores piores que o algortimo randômico em termos de custo de solução (devido à imprevisibilidade do aleatório), o mesmo possui valores mais consistentes e direcionados para a resolução do problema, garantindo sempre uma solução que possa ser aproveitada.

Algoritmo 3: Método Randômico Adaptativo Restritivo

Entrada: objeto de solução
Saída: objeto de solução com cadeia preenchida
início
  repita
    $OrdenaListaColunas()$;
    $k \leftarrow$ Sorteio(0, $|ListaColunas|$);
    $caractere \leftarrow MetodoRoleta()$;
    sol[k] $\leftarrow$ $caractere$;
    $AtualizaListaColunas(k)$;
  até *todas colunas preenchidas*;
fim

4.2 Heurísticas de Busca Local

Heurísticas de busca local são úteis e geralmente utilizadas para resolver problemas de otimização puros onde o objetivo é encontrar o melhor estado de acordo com uma função objetivo.

No caso específico do Problema da Sequência Mais Próxima, as funções de Busca Local foram criadas com o intuito de servir novas soluções que permitam ao método *Simulated Annealing* continuar a execução do algoritmo e fazer com que o processo saia de possíveis pontos onde a função encontre um mínimo local, impossibilitando que o custo da mesma continue decaindo ao longo do processo, conforme esperado.

4.2.1 *Neighbor Solution Explorer* (NSE)

A primeira função desenvolvida gera as soluções substituindo um número pré-definido de caracteres (1, 2 ou 3) em cada iteração, verificando se houve diminuição do custo com as modificações realizadas.

Em cada iteração que é realizada o algoritmo percorre todas as colunas fazendo todas as possíveis combinações de caracteres (1 a 1, 2 a 2 ou 3 a 3, de acordo com o parâmetro definido anteriormente), aceitando apenas as mudanças que geram as soluções com o menor custo possível.

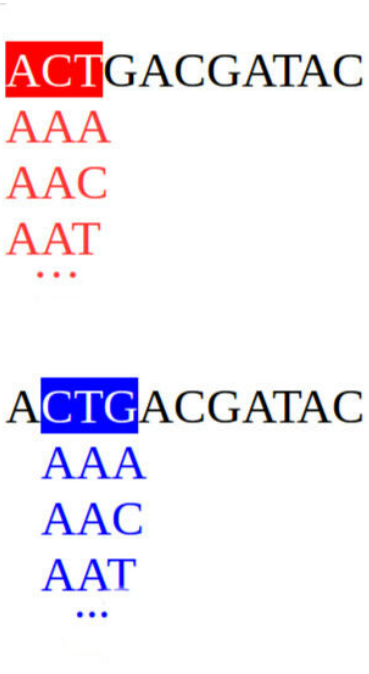

Figura 4.2: Exemplo da substituição de caracteres no método *Neighbor Solution Explorer* (NSE) onde o parâmetro é igual a 3 e todas as combinações (3 a 3) possíveis são realizadas.

4.2.2 *Random Neighbor Solution* (RNS)

A segunda função de Busca Local desenvolvida busca direcionar as suas análises para as sequências do conjunto passado como parâmetro que estão contribuindo para elevar o custo da solução atual.

Inicialmente são separadas as cadeias da matriz de entrada onde o custo seja o mesmo da solução atual, já que estas são as responsáveis pela *Distância de Hamming* atual da cadeia objetivo.

Posteriormente, uma cadeia é selecionada através do mesmo processo desenvolvido na Heurística de Construção *RARM (Random Adaptative Restrictive Method)*, onde uma lista restrita de candidatos é criada, com tamanho definido por um parâmetro *alfa*, e recebe as cadeias separadas na etapa inicial do *RNS*. Em seguida, um sorteio é realizado e a sequência correspondente ao índice sorteado é selecionada para ter seu contéudo alterado.

O próximo passo do algoritmo é definir qual coluna terá seu conteúdo modificado, selecionando-a através de um sorteio e determinando o caractere que irá preenchê-la após comparar todas as possíveis combinações para aquela posição e escolhendo a que gerar o menor custo.

4.2.3 *Neighbor Solution* (NS)

A função escolhida para ser utilizada foi a que obteve melhores resultados em termos de qualidade de solução em comparação com as implementadas anteriormente.

O algoritmo do NS gera cadeias a partir da substituição de n caracteres da solução que é passada como parâmetro para o método. Esse valor de n corresponde a um inteiro chamado de janela e que é definido através de um sorteio de números entre 0 e metade da temperatura atual do processo de iterações do *Simulated Annealing* (que será detalhado posteriormente).

Após definir quantas colunas serão alteradas (tamanho da janela), o algoritmo

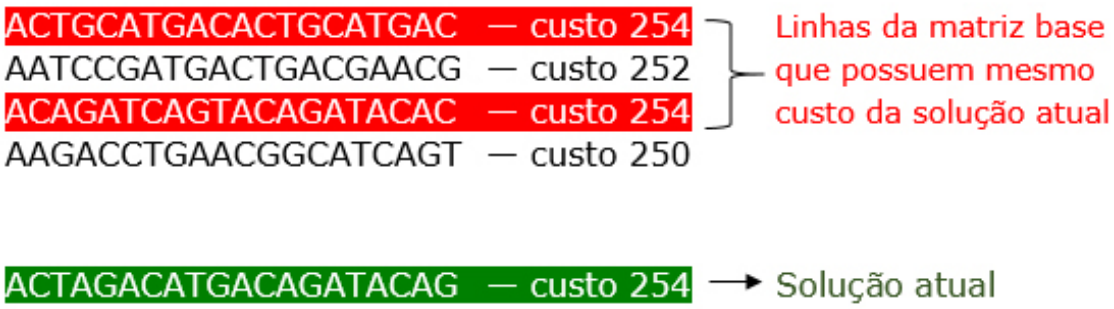

Figura 4.3: Exemplo da comparação e escolha das cadeias que possuem mesmo custo da solução atual realizada no método *Random Neighbor Solution* (RNS).

realiza as trocas da posição i (valor aleatório) até a posição $i+n$. Cada um dos caracteres dessas colunas é alterado por correspondentes que se encontram na Cadeia Consenso (sequência formada pelos caracteres mais comuns de cada coluna da matriz de entrada).

Ao fim do processo o custo dessa nova solução é calculado e a mesma é retornada para o algoritmo do *Simulated Annealing*.

4.3 *Simulated Annealing*

O *Simulated Annealing* (ou SA) é uma metaheurística probabilística de otimização global e que foi inicialmente aplicada por (Kirkpatrick *et al.*, 1983) na resolução de problemas de otimização e também no trabalho de (Černỳ, 1985), onde foi utilizado no particionamento de grafos.

Conforme descrito por (Talbi, 2009), ela é baseada nos princípios da mecânica estatística, onde o processo de recozimento exige que se aqueça e, em seguida, esfrie lentamente uma substância para se obter uma estrutura cristalina forte e a resistência dessa estrutura dependerá da taxa de resfriamento dos materiais. Se a temperatura inicial não for suficientemente alta ou o resfriamento for feito de maneira rápida, as imperfeições ocorrerão. Neste caso, o sólido não vai atingir o equilíbrio térmico desejado.

Sistema físico	**Problema de otimização**
Estado do sistema	Solução
Posição das moléculas	Variáveis de decisão
Energia	Função objetivo
Estado fundamental	Solução global ótima
Estado metaestável	Ótimo local
Supressão rápida	Busca local
Temperatura	Parâmetro de controle T
Recozimento cuidadoso	recozimento simulado

Tabela 4.1: Analogia entre o sistema físico e o problema de otimização (Talbi, 2009)

O algoritmo SA simula as mudanças de energia de um sistema sujeito a um processo de arrefecimento até que o mesmo convirja para um estado de equilíbrio e seu objetivo é escapar de ótimos locais, atrasando, assim, a convergência.

O algoritmo funciona a partir de uma solução inicial passada como parâmetro

e executa por várias iterações. A cada iteração, uma solução é gerada aleatoriamente na vizinhança. Soluções que melhoram (reduzem) a função de custo são sempre aceitas. Caso contrário, essa solução pode ser aceita com base em uma probabilidade que depende da temperatural atual do sistema e da quantidade de degradação ΔE da função objetivo. ΔE representa a diferença no valor objetivo (energia) entre a solução atual e a solução gerada da vizinhança.

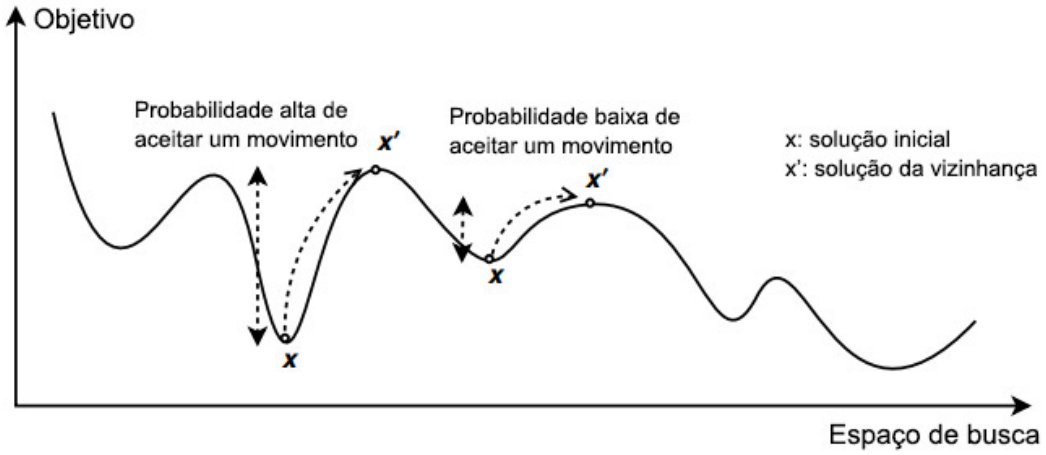

Figura 4.4: *Simulated Annealing* escapando de um ótimo local (Talbi, 2009)

A medida que o algoritmo progride, a probabilidade de que tais movimentos sejam aceitos diminui. A fórmula da probabilidade é apresentada abaixo:

$$P(\Delta E, T) = e^{-\frac{f(s') - f(s)}{T}} \tag{4.2}$$

4.3.1 Implementação do *Simulated Annealing*

Algoritmo 4: Algoritmo do *Simulated Annealing*

Entrada: objeto de solução
Saída: melhor solução encontrada
início
  $s \leftarrow s_0$; /* Gera a solução inicial */
  $T \leftarrow T_{max}$; /* Define a temperatura inicial */
  repita
    repita
      $s \leftarrow SolucaoAleatoriaVizinhanca()$;
      $\Delta E \leftarrow f(s') - f(s)$;
      se $\Delta E \leq 0$ **então**
        $s \leftarrow s'$;
      senão
        Aceita s' com probabilidade $e^{-\frac{\Delta E}{T}}$;
      fim se
    até *condição de equilíbrio*;
    $T \leftarrow g(T)$; /* Atualiza a temperatura */
  até *critério de parada satisfeito*;
fim

O algoritmo do SA apresenta ainda, além dos parâmetros citados acima, outras variáveis que controlam a maneira como a execução do mesmo será realizada. São eles: Temperatura inicial (ou temperatura máxima) que define o ponto de partida do algoritmo; temperatura final (ou temperatura mínima) que define até qual valor de temperatura o algoritmo irá executar; o número de vezes que o algoritmo irá executar para uma mesma temperatura, representado como condição de equilíbrio no Algoritmo 4; além de uma variável que define de que forma será feito o decaimento (ou a taxa de resfriamento) da temperatura durante a execução do mesmo. Esses parâmetros são de extrema importância e a correta calibragem dos mesmos pode definir o sucesso ou a falha do algoritmo implementado.

Nesse trabalho, o SA foi utilizado em sua forma original, conforme descrito no Algoritmo 4 acima, que sofreu apenas uma modificação na chamada da sua função geradora de soluções aleatórias, onde um novo laço de iteração foi inserido para que execute um certo número de vezes para um parâmetro que é sorteado a cada iteração.

Essa mudança deveu-se ao fato da função se basear no processo de intensificação, encontrado no Algoritmo Genético de (Soares *et al.*, 2013), onde um parâmetro k é sorteado e indica o número de posições da cadeia que serão substituídos pelos caracteres correspondentes da sequência consenso criada a partir dos caracteres mais comuns de cada coluna da matriz de entrada. A idéia dessa mudança é que o método de busca funcione de maneira parecida com o método de Janelas Deslizantes utilizado por (Soares *et al.*, 2013), mantendo o tamanho da janela de substituição por algumas iterações, e modificando apenas o parâmetro que define a posição inicial de onde a troca será efetuada.

4.3.2 Modificações no algoritmo do *Simulated Annealing*

Na tentativa de minimizar ainda mais o custo das soluções geradas pelo algoritmo do SA descrito anteriormente, algumas tentativas de modificações foram realizadas.

Na primeira, ao invés do algoritmo gerar em cada iteração uma solução a partir de uma função de busca local, ele realiza três chamadas ao método NSE, passando como parâmetro os valores 1, 2 e 3, respectivamente. Dessa forma, a solução atual é gerada na vizinhança e modificada, explorando ainda mais o espaço de soluções, antes mesmo de ter seu custo verficado e comparado com o ΔE da solução atual do método e definido se essa solução será aceita ou não.

Na segunda modificação do SA, a função de busca local utilizada era a mesma utilizada na versão final do código do problema (NS), cujo funcionamento é similar ao método das Janelas Deslizantes de (Soares *et al.*, 2013). A diferença é a inserção de um conjunto chamado Elite, também presente no trabalho de Soares *et al.* (2013), responsável por guardar as melhores soluções geradas nas iterações anteriores e responsável por atualizar a cadeia consenso (com os caracteres mais relevantes em cada coluna), ao invés de utilizar apenas a cadeia gerada a partir da matriz-base no passo inicial do algoritmo final.

Apesar das modificações realizadas, que visaram otimizar a minimização dos custos gerados pelo *Simulated Annealing*, nenhuma delas foi capaz de melhorar os valores apresentados pelo algoritmo do SA em sua forma mais original nos testes comparativos realizados, ou seja, as alterações não se mostraram interessantes e foram preteridas pelo método apresentado anteriormente e que está presente no código final do trabalho desenvolvido.

5 Experimentos e Resultados

Nesta seção, são apresentados alguns dos experimentos realizados com os métodos descritos na seção anterior. O algoritmo foi implementado e testado em um ambiente Windows 7 64 bits, utilizando a linguagem C++ (*Code::Blocks IDE*) em uma máquina com processador *Intel Core* i3 3.10GHz e 8GB de memória RAM.

Os testes foram realizados através da utilização de um script que executava o algoritmo para cada instância, salvando em um arquivo de texto o custo final (e consenquentemente o menor) da solução encontrada, além do tempo total de execução e do tempo necessário para encontrar a melhor cadeia solução. Este processo foi repetido 10 vezes para cada uma das instâncias, que continham a matriz-base. Estas instâncias foram as mesmas utilizadas em (Lyra, 2012) e em (Soares *et al.*, 2013).

A Tabela 5.1 mostra a comparação entre as Heurísticas de Construção que foram propostas quanto à qualidade da solução. A primeira coluna indica o nome da instância no formato $nXmYt*$, onde X representa o número de sequências da base e Y o tamanho de cada uma delas. As colunas restantes representam o custo da melhor solução gerada e as diferenças percentuais de cada um dos algoritmos (RM, RAM, RARM) em relação à melhor solução obtida.

É possível observar, conforme já mencionado na descrição dos algoritmos, que o método Randômico obteve alguns resultados superiores aos outros dois métodos. Entretanto, a Tabela 5.1 mostra que algoritmo Randômico Restritivo Adaptativo obteve melhores resultados em quase 75% das instâncias. Convém destacar o compostamento do algoritmo RAM, que se mostrou ineficiente para todas as instâncias testadas, o que destaca as caracterìsticas do RARM.

A partir dos resultados apresentados pela heurística de construção RARM, optou-se por utilizar esta na fase de construção do *Simulated Annealing* proposto. A Tabela 5.3 apresenta os resultados computacionais dos testes realizados com a combinação com a Heurística de Busca Local *Neighbor Solution* (*NS*), que apresentou melhores resultados em comparação às demais implementadas. A segunda coluna indica o valor da solução obtid e a média de custo das 10 execuções. As colunas seguintes apresentam as médias de melhor tempo (tempo necessário para o algoritmo encontrar a solução que foi retornada) e de tempo total de execução. Os parâmetros utilizados nos testes do SA foram definidos de forma empírica como sendo: A temperatura inicial recebeu o valor de m (tamanho da sequência), enquanto a temperatura final foi definida como 0,01; já a taxa de decaimento foi definida como 0,90, ou seja, a temperatura diminui 10% a cada atualização.

Os resultados mostrados na Tabela 5.3 mostram a escalabilidade da abordagem proposta. As instâncias encontram-se agupadas quanto ao tamanho em termos de número de sequêcias e de número de caracteres. Pode-se observar que o aumento no tempo de execução é linear.

É possível verificar que a diferença entre o melhor custo e a média de custos é pequena, indicando que o algoritmo possui a robustez da abordagem proposta, demonstrando que o melhor valor encontrado não se trata de um valor encontrado ao acaso por conta de um sorteio bem sucedido das partes aleatórias do código.

Já em relação ao tempo percebe-se uma diferença grande entre as médias de melhor tempo e de tempo total. Indicando que, na grande maioria dos casos, o algoritmo encontra a melhor solução rapidamente e em seguida não consegue melhorá-la, possivelmente após atingir algum ponto onde a função indica um mínimo local, e de onde o algoritmo da Busca Local escolhido deveria utilizar a capacidade de oscilação do custo

Instância	**Melhor solução**	**Randômico (RM)**	**Randômico Restritivo (RAM)**	**Randômico Restritivo Adaptativo (RARM)**
n10m300tai1	215	**0,00**%	10,70%	**0,00**%
n10m300tai2	214	**0,00**%	9,35%	1,40%
n10m300tai3	213	**0,00**%	12,68%	1,41%
n10m400tai1	281	1,07%	9,25%	**0,00**%
n10m400tai2	284	0,35%	9,51%	**0,00**%
n10m400tai3	284	0,00%	10,56%	**0,00**%
n10m500tai1	354	0,56%	10,45%	**0,00**%
n10m500tai2	351	0,85%	8,55%	**0,00**%
n10m500tai3	349	0,86%	13,18%	**0,00**%
n10m1000tai1	695	**0,00**%	11,37%	0,14%
n10m1000tai2	697	**0,00**%	9,90%	0,57%
n10m1000tai3	697	0,29%	10,62%	**0,00**%
n10m2000tai1	1380	0,72%	11,81%	**0,00**%
n10m2000tai2	1381	0,36%	10,86%	**0,00**%
n10m2000tai3	1378	0,15%	10,30%	**0,00**%
n10m3000tai1	2056	**0,00**%	10,99%	0,05%
n10m3000tai2	2058	0,29%	11,13%	**0,00**%
n10m3000tai3	2057	0,10%	10,94%	**0,00**%
n10m4000tai1	2736	0,44%	10,93%	**0,00**%
n10m4000tai2	2742	0,15%	12,33%	**0,00**%
n10m4000tai3	2744	0,18%	10,90%	**0,00**%
n10m5000tai1	3422	**0,00**%	10,67%	0,56%
n10m5000tai2	3414	0,09%	11,45%	**0,00**%
n10m5000tai3	3415	0,26%	11,07%	**0,00**%
n20m1000tai1	736	**0,00**%	6,39%	0,27%
n20m1000tai2	738	0,14%	4,88%	**0,00**%
n20m1000tai3	735	0,41%	5,85%	**0,00**%
n20m2000tai1	1457	0,27%	5,97%	**0,00**%
n20m2000tai2	1457	**0,00**%	5,15%	0,27%
n20m2000tai3	1460	0,14%	5,41%	**0,00**%
n20m3000tai1	2175	0,23%	5,47%	**0,00**%
n20m3000tai2	2178	0,05%	5,33%	**0,00**%
n20m3000tai3	2175	**0,00**%	5,10%	0,37%
n20m4000tai1	2897	0,03%	4,87%	**0,00**%
n20m4000tai2	2892	0,24%	6,50%	**0,00**%
n20m4000tai3	2901	0,17%	5,21%	**0,00**%
n20m5000tai1	3610	0,53%	4,90%	**0,00**%
n20m5000tai2	3617	0,08%	5,20%	**0,00**%
n20m5000tai3	3613	0,11%	5,12%	**0,00**%
Média		0,23%	8,74%	**0,13%**

Tabela 5.1: Comparação entre as Heurística de Construção propostas quanto à qualidade da solução.

das soluções do *Simulated Annealing* para fazer variar um pouco o custo do algoritmo, elevando-o temporiamente, para em seguida sair desse chamado buraco, em busca do mínimo absoluto dessa função.

A tabela 5.3 ilustra a comparação entre o método original do *Simulated Annealing* e as tentativas de modificação que foram realizadas no mesmo. Percebe-se que o algoritmo em sua forma original apresenta resultados bem superiores, em termos de qualidade de solução, justificando a razão da escolha desse método para compor o código final do algoritmo de resolução do problema.

A tabela 5.4 mostra a comparação entre os melhores resultados obtidos pela abordagem proposta e aqueles apresentados por (Soares *et al.*, 2013), onde é apresentada uma abordagem baseada no Algoritmo Genético para a resolução do Problema da Sequência Mais Próxima. As colunas representam a melhor solução encontrada e a média do melhor tempo, ou o tempo gasto pelo algoritmo para encontrar a melhor solução desde o início da execução, nos testes realizados, . Os valores em negrito indicam as instâncias em que cada uma das abordagens teve melhor desempenho.

Observa-se que em apenas uma das instâncias (n10m4000tai1) a abordagem desenvolvida obteve resultados melhores que a abordagem apresentada por (Soares *et al.*, 2013) e que apesar da média do tempo ser menor para diversas instâncias esse fato isolado não representa uma melhoria em termos de desempenho na resolução do problema devido ao fato apresentado anteriormente, na Tabela 5.2, de que esse baixo tempo para encontrar uma solução considerada ótima esteja relacionado à incapacidade do algoritmo de contornar uma situação de mínimo local, ignorando um possível ponto de mínimo global que esteja presente em alguma parte do espaço de soluções.

Instância	Melhor solução	Qualidade média	Tempo (seg)	
			Média melhor tempo	Média tempo total
n10m300tai1	185	187,6	0,350	2,738
n10m300tai2	183	185,6	0,265	2,630
n10m300tai3	184	185,6	0,234	2,622
n10m400tai1	243	245,7	0,662	3,471
n10m400tai2	238	239,6	0,240	2,996
n10m400tai3	244	247,4	0,900	3,643
n10m500tai1	300	303,1	0,548	3,809
n10m500tai2	295	295,6	0,312	3,626
n10m500tai3	298	299,7	0,651	3,929
n10m1000tai1	600	602,1	8,594	19,001
n10m1000tai2	590	593,9	4,804	15,015
n10m1000tai3	603	604,7	7,103	17,266
n10m2000tai1	1190	1194,7	39,619	60,283
n10m2000tai2	1189	1195	39,128	59,593
n10m2000tai3	1192	1202,9	37,923	58,495
n10m3000tai1	1766	1768,2	105,114	136,431
n10m3000tai2	1775	1779	114,583	145,726
n10m3000tai3	1769	1774	110,961	142,007
n10m4000tai1	2334	2336,6	249,730	291,458
n10m4000tai2	2374	2381	272,031	314,120
n10m4000tai3	2355	2359,2	214,069	251,767
n10m5000tai1	2963	2968	291,614	331,917
n10m5000tai2	2921	2924	289,119	329,71
n10m5000tai3	2935	2943,2	334,691	377,731
n20m1000tai1	648	651,1	5,721	20,316
n20m1000tai2	660	665,3	7,689	22,100
n20m1000tai3	655	661,1	4,343	18,843
n20m2000tai1	1293	1297,9	19,697	49,338
n20m2000tai2	1290	1294,6	20,283	50,102
n20m2000tai3	1292	1296,3	28,446	58,109
n20m3000tai1	1928	1933,7	87,864	133,998
n20m3000tai2	1934	1943,1	79,091	126,334
n20m3000tai3	1937	1948,3	84,479	128,617
n20m4000tai1	2559	2562	140,036	196,144
n20m4000tai2	2576	2599,3	177,183	238,236
n20m4000tai3	2574	2585,4	167,530	228,356
n20m5000tai1	3206	3213	304,453	380,496
n20m5000tai2	3207	3213,2	311,327	387,560
n20m5000tai3	3219	3233,6	320,891	397,244

Tabela 5.2: Comportamento dos algoritmos quanto à qualidade média e tempo de execução.

Instância	**Melhor solução**	**SA original**	**SA com substituições (1, 2 e 3)**	**SA com conjunto Elite**
n10m300tai1	185	**0,00%**	1,11%	1,11%
n10m300tai2	183	**0,00%**	1,11%	1,11%
n10m300tai3	184	**0,00%**	1,10%	1,11%
n10m400tai1	243	**0,00%**	1,12%	1,12%
n10m400tai2	238	**0,00%**	1,14%	1,15%
n10m400tai3	244	**0,00%**	1,11%	1,12%
n10m500tai1	300	**0,00%**	1,13%	1,14%
n10m500tai2	295	**0,00%**	1,14%	1,14%
n10m500tai3	298	**0,00%**	1,12%	1,12%
n10m1000tai1	600	**0,00%**	1,12%	1,12%
n10m1000tai2	590	**0,00%**	1,13%	1,13%
n10m1000tai3	603	**0,00%**	1,12%	1,12%
n10m2000tai1	1190	**0,00%**	1,13%	1,14%
n10m2000tai2	1189	**0,00%**	1,13%	1,13%
n10m2000tai3	1192	**0,00%**	1,13%	1,13%
n10m3000tai1	1766	**0,00%**	1,14%	1,14%
n10m3000tai2	1775	**0,00%**	1,13%	1,13%
n10m3000tai3	1769	**0,00%**	1,14%	1,14%
n10m4000tai1	2334	**0,00%**	1,15%	1,15%
n10m4000tai2	2374	**0,00%**	1,13%	1,13%
n10m4000tai3	2355	**0,00%**	1,14%	1,14%
n10m5000tai1	2963	**0,00%**	1,13%	1,13%
n10m5000tai2	2921	**0,00%**	1,15%	1,15%
n10m5000tai3	2935	**0,00%**	1,14%	1,15%
n20m1000tai1	648	**0,00%**	1,11%	1,10%
n20m1000tai2	660	**0,00%**	1,09%	1,09%
n20m1000tai3	655	**0,00%**	1,09%	1,09%
n20m2000tai1	1293	**0,00%**	1,11%	1,11%
n20m2000tai2	1290	**0,00%**	1,11%	1,11%
n20m2000tai3	1292	**0,00%**	1,11%	1,11%
n20m3000tai1	1928	**0,00%**	1,11%	1,11%
n20m3000tai2	1934	**0,00%**	1,11%	1,11%
n20m3000tai3	1937	**0,00%**	1,11%	1,11%
n20m4000tai1	2559	**0,00%**	1,12%	1,12%
n20m4000tai2	2576	**0,00%**	1,11%	1,11%
n20m4000tai3	2574	**0,00%**	1,11%	1,11%
n20m5000tai1	3206	**0,00%**	1,11%	1,11%
n20m5000tai2	3207	**0,00%**	1,11%	1,11%
n20m5000tai3	3219	**0,00%**	1,11%	1,11%
Médias		**0,00%**	1,12%	1,12%

Tabela 5.3: Comporação das diferentes implementações do *Simulated Annealing* em termos de qualidade de solução.

Instância	SA		AG	
	Melhor solução	Tempo (seg)	Melhor solução	Tempo (seg)
n10m1000tai1	1,01%	8,59	**0,00%**	23,22
n10m1000tai2	1,01%	4,80	**0,00%**	24,01
n10m1000tai3	1,02%	7,10	**0,00%**	18,15
n10m300tai1	1,03%	0,35	**0,00%**	15,01
n10m300tai2	1,02%	0,27	**0,00%**	15,81
n10m300tai3	1,05%	0,23	**0,00%**	9,26
n10m400tai1	1,02%	0,66	**0,00%**	15,80
n10m400tai2	1,01%	0,24	**0,00%**	14,79
n10m400tai3	1,02%	0,90	**0,00%**	18,59
n10m500tai1	1,01%	0,55	**0,00%**	15,24
n10m500tai2	1,01%	0,31	**0,00%**	6,64
n10m500tai3	1,02%	0,65	**0,00%**	15,79
n20m1000tai1	1,01%	5,72	**0,00%**	24,28
n20m1000tai2	1,02%	7,69	**0,00%**	34,08
n20m1000tai3	1,01%	4,34	**0,00%**	31,50
n10m2000tai1	1,01%	39,61	**0,00%**	44,31
n10m2000tai2	1,00%	39,12	**0,00%**	58,15
n10m2000tai3	1,02%	37,92	**0,00%**	48,62
n10m3000tai1	1,00%	105,11	**0,00%**	61,66
n10m3000tai2	1,00%	114,58	**0,00%**	56,60
n10m3000tai3	1,01%	110,96	**0,00%**	40,06
n10m4000tai1	**0,00%**	249,73	1,00%	102,94
n10m4000tai2	1,01%	272,03	**0,00%**	102,95
n10m4000tai3	1,00%	214,06	**0,00%**	109,91
n10m5000tai1	1,01%	291,61	**0,00%**	136,10
n10m5000tai2	1,00%	289,11	**0,00%**	173,46
n10m5000tai3	1,00%	334,69	**0,00%**	141,05
n20m2000tai1	1,01%	19,69	**0,00%**	43,29
n20m2000tai2	1,00%	20,28	**0,00%**	76,87
n20m2000tai3	1,00%	28,44	**0,00%**	60,77
n20m3000tai1	1,00%	87,86	**0,00%**	76,83
n20m3000tai2	1,01%	79,09	**0,00%**	119,08
n20m3000tai3	1,00%	84,47	**0,00%**	83,27
n20m4000tai1	1,00%	140,03	**0,00%**	117,23
n20m4000tai2	1,00%	177,18	**0,00%**	91,73
n20m4000tai3	1,00%	167,53	**0,00%**	112,91
n20m5000tai1	1,00%	304,45	**0,00%**	143,65
n20m5000tai2	1,00%	311,32	**0,00%**	144,56
n20m5000tai3	1,00%	320,89	**0,00%**	167,94
Médias	0,98%	99,54	**0,02%**	66,57

Tabela 5.4: Comparação com a literatura

6 Conclusão

O Problema da Sequência Mais Próxima tem se tornado cada vez mais importante para a área de Biologia Computacional. Avanços em desempenho e qualidade na resolução do mesmo são necessários, visto que encontrar uma solução melhor (com menor custo) ou reduzir o tempo necessário para obter as soluções já existentes, pode representar a descoberta de um novo gene ligado à uma doença ou até mesmo a cura de alguma já existente.

Este trabalho apresentou algumas funções de Heurísticas de Construção e de Busca Local que foram desenvolvidas e incorporadas ao código do *Simulated Annealing*. Após diversos testes, calibragem de parâmetros e novas tentativas de modificação do algoritmo implementado, conclui-se que, apesar dos resultados obtidos serem bem próximos (para algumas instâncias) aos encontrados por (Soares *et al.*, 2013) e ao contrário do que foi pensado inicialmente, os algoritmos desenvolvidos nesse trabalho e que foram incorporados ao algoritmo do *Simulated Annealing*, não se mostraram os mais indicados para a resolução do problema e objetivo dessa pesquisa.

Como trabalhos futuros, sugere-se um estudo dos parâmetros existentes no *Simulated Annealing*, bem como do desenvolvimento de novas funções para construção das sequências iniciais, que possibilitem ao método avançar no espaço de soluções, e também de outras heurísticas de Busca Local que sejam capazes de identificar e ultrapassar os pontos de mínimo local, possibilitando que o método avance até os pontos de mínimo global. Além da implantação do método conhecido como Reconexão de Caminhos, que pode fazer com que os resultados atuais sejam melhorados de imediato.

A Módulo PSMP

Com o objetivo de gerar soluções que minimizem a maior distância de Hamming para diferentes instâncias de entradas de dados foi desenvolvido um módulo que recebeu o nome de PSMP. Nesse módulo foram implementadas três classes (*Linkedlist*, PSMP e *Solution*) que em conjunto com a classe principal *Main* compõem o algoritmo desenvolvido no trabalho.

O presente capítulo descreve as funcionalidades do módulo citado acima, com foco nas funcionalidades desenvolvidas.

A.0.3 Módulo Principal (main.cpp)

Arquivo principal do projeto, responsável por inicializar as variáveis, fazer a leitura da instância de entrada que é passada como parâmetro, realizar a chamada das funções das outras classes para preenchimento inicial da cadeia de solução e otimização do custo através da função do *Simulated Annealing*. Ao final do processo, retorna para a saída de texto o resultado da última execução do algoritmo.

Armazena em uma matriz $m \times n$ (onde m equivale ao tamanho e n ao total de sequências presentes no arquivo de entrada) o conteúdo da instância passada como parâmetro. Essa matriz é nomeada como base.

A.0.4 Classe *Linkedlist*

A classe *Linkedlist* foi desenvolvida com o intuito de prover para os outros módulos as ferramentas para criar a estrutura de dados conhecida como Lista Encadeada.

A importância dessa estrutura para o projeto se deve ao fato de ser necessário armazenar qual o alfabeto (tamanho e caracteres) utilizado pelas instâncias de entrada e de não sabermos de antemão quais caracteres compõem o mesmo, pois apesar de todas as instâncias de entrada utilizadas pelo testes serem formadas pelos caracteres A, C, T e G, isso não poderia ser fixado ao código já que faria com que o mesmo não se tornasse genérico para outras sequências de caracteres de entrada.

A classe é formada pelos parâmetros *count_char* (inteiro que armazena o número de vezes que aquele caractere se repete em toda as sequências passadas como entrada), *character* (char que identifica qual o caractere ocupa aquela posição da lista) e *next* (ponteiro de *Linkedlist* que identifica qual o próximo elemento da lista).

A.0.5 Classe Solution

Classe responsável por instanciar objetos de solução. Seus atributos são:

- *sequence* → Ponteiro de *char* onde é armazenado a cadeia de caracteres que aquela solução possui atualmente.
- *distance* → Inteiro que guarda a maior distância de *Hamming* dessa solução em relação à todas as sequências presente na matriz base.
- *array_costs* → Array de inteiros com tamanho igual à n utilizado para armazenar as distâncias de *Hamming* do objeto atual para cada uma das cadeias da matriz base.

A.0.6 Classe PSMP

Classe que engloba as principais funções que foram desenvolvidas nesse trabalho. Possui os seguintes atributos:

- *array_sequence* → É a matriz base que foi citada anteriormente. Responsável por armazenar as cadeias existentes no arquivo de entrada.
- *alphabet* → Objeto de *LinkedList* que guarda quais os caracteres que compõem o alfabeto da base.
- *consensus_sol* & *elite_set* → Cadeia consenso e conjunto Elite de soluções; ambos definidos anteriormente e que guardam a cadeia consenso de solução e o conjunto Elite (que armazena soluções geradas anteriormente), respectivamente.

Possui também as funções baseadas em heurísticas de construção, os métodos de busca local e algoritmos para cálculo do custo (distância de *Hamming*) da solução, além da função baseada na metaheurística *Simulated Annealing* que foram detalhadas anteriormente.

Referências Bibliográficas

Stojanovic, N.; Berman, P.; Gumucio, D.; Hardison, R. ; Miller, W. **A linear-time algorithm for the 1-mismatch problem**. In: Algorithms and Data Structures, p. 126–135. Springer, 1997.

Cantor, C. R.; Smith, C. L. **Genomics: The science and technology behind the Human Genome Project**, volume 12. John Wiley & Sons, 2002.

Černỳ, V. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. **Journal of optimization theory and applications**, v.45, n.1, p. 41–51, 1985.

Feo, T. A.; Resende, M. G. Greedy randomized adaptive search procedures. **Journal of global optimization**, v.6, n.2, p. 109–133, 1995.

Frances, M.; Litman, A. On covering problems of codes. **Theory of Computing Systems**, v.30, n.2, p. 113–119, 1997.

Glover, F. **Tabu search and adaptive memory programmingâadvances, applications and challenges**. In: Interfaces in computer science and operations research, p. 1–75. Springer, 1996.

Goldberg, D. E.; Holland, J. H. Genetic algorithms and machine learning. **Machine learning**, v.3, n.2, p. 95–99, 1988.

Mladenović, N.; Hansen, P. Variable neighborhood search. **Computers & Operations Research**, v.24, n.11, p. 1097–1100, 1997.

Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P. ; others. Optimization by simmulated annealing. **science**, v.220, n.4598, p. 671–680, 1983.

Kevin Lanctot, J.; Li, M.; Ma, B.; Wang, S. ; Zhang, L. Distinguishing string selection problems. **Information and Computation**, v.185, n.1, p. 41–55, 2003.

Li, M.; Ma, B. ; Wang, L. On the closest string and substring problems. **Journal of the ACM (JACM)**, v.49, n.2, p. 157–171, 2002.

Lyra, A. **Métodos exatos e heurísticos para o problema da sequência mais próxima**. In: Simpósio Brasileiro de Pesquisa Operacional, 2012.

Mateus, G. R.; Resende, M. G. ; Silva, R. M. Grasp: Procedimentos de busca gulosos, aleatórios e adaptativos. **2a Escola Luso-Brasileira de Computação Evolutiva**, 2010.

Michalewicz, Z. **Genetic algorithms+ data structures= evolution programs**. springer, 1996.

Meneses, C. N.; Lu, Z.; Oliveira, C. A. ; Pardalos, P. M. Optimal solutions for the closest-string problem via integer programming. **INFORMS Journal on Computing**, v.16, n.4, p. 419–429, 2004.

Schneider, J. J.; Kirkpatrick, S. **Stochastic optimization**. Springer, 2006.

Setubal, J. C.; Meidanis, J. **Introduction to computational molecular biology**. PWS Pub., 1997.

Soares, S. S. R. F.; Gonçalves, L. B. ; de Lyra, A. R. Um algortimo genético para o problema da sequência mais próxima. 2013.

Talbi, E.-G. **Metaheuristics: from design to implementation**, volume 74. John Wiley & Sons, 2009.

Yamamoto, K. **Arredondamento randômico e o problema da sequência mais próxima**. Dissertação de Mestrado - .