Uma 3−aproximação e uma formulação de PLI para o Problema do Alinhamento *Spliced* Múltiplo

Regina Beretta Mazaro

DISSERTAÇÃO APRESENTADA
À
FACULDADE DE COMPUTAÇÃO
DA
UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
CIÊNCIA DA COMPUTAÇÃO

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Said Sadique Adi

– Durante o desenvolvimento deste trabalho, a autora recebeu apoio financeiro da CAPES –

Campo Grande, outubro de 2014

Regina Beretta Mazaro

Uma 3−aproximação e uma formulação de PLI para o Problema do Alinhamento*Spliced* Múltiplo

Dissertação apresentada como requisito para obtenção do grau de Mestre em Ciência da Computação no curso de Mestrado em Ciência da Computação da Faculdade de Computação - Fundação Universidade Federal de Mato Grosso do Sul.
Orientação: Prof. Dr. Said Sadique Adi

Campo Grande, outubro de 2014.

Dedico essa dissertação aos que mais de
perto me acompanharam nesta
empreitada, minha família: mãe Valéria,
pai Alexandre, irmã sem fronteiras
Giovana e irmã de coração Janaína.

Agradecimentos

Agradeço aos meus pais, Valéria e Alexandre, pelo carinho, cuidado e apoio incondicional que lhe são peculiares, fatores essenciais para que hoje eu possa fazê-los sorrir com esta nova conquista;

A minha irmã Giovana, pela constante motivação em buscar seus sonhos que me contagiaram a também seguir atrás dos meus;

À amiga-irmã Janaína, por partilhar cada desafio do Mestrado tornando-os mais leves com sua companhia e amizade indispensáveis, e principalmente por ter sido responsável por me manter firme nas muitas vezes que vacilei;

Aos amigos Henrique e Marisa, que por extensão se fizeram presentes em muitos momentos dessa longa trajetória;

Ao meu orientador Said, pela paciência e atenção de sempre, por ter mais uma vez me acompanhado durante todo esse processo acadêmico compartilhando seu conhecimento, e acima de tudo por ter respeitado minhas decisões profissionais;

Agradeço também à professora Edna, que sempre com muita disposição me auxiliou em relação à formulação de PLI e disponibilizou a sua máquina com a licença acadêmica da biblioteca necessária para uso nos testes experimentais;

Aos colegas Leandro, pela valiosa contribuição com o algoritmo de aproximação, e Rodrigo, por fornecer o material de seu mestrado para extensão do estudo do prolema;

Por fim, agradeço aos meus colegas de trabalho na TI da EAD/UFMS pela compreensão e liberdade concedida para que conciliasse o desenvolvimento da dissertação com as demandas do setor nos últimos meses.

"Always be kind, for everyone is fighting a hard battle."

Plato

Resumo

Com os avanços recentes em áreas específicas da Ciência da Computação como a Biologia Computacional, vários problemas novos envolvendo sequências vêm surgindo, enquanto que problemas tradicionais tornam-se mais difíceis dada a expressiva quantidade de dados gerada nos últimos anos. O interesse aqui é no estudo de um problema específico que envolve sequências denominado Problema do Alinhamento *Spliced* Múltiplo, estendendo um trabalho anteriormente realizado por Kishi e Adi em cima desse mesmo problema. Enquanto que nesse estudo prévio mostrou-se que o Problema do Alinhamento *Spliced* Múltiplo é NP-completo e foram propostas heurísticas para o problema, o presente trabalho visa sugerir um algoritmo de aproximação e uma formulação de programação linear inteira para ele, possibilitando confrontar essas novas abordagens com as heurísticas já desenvolvidas para o Problema do Alinhamento *Spliced* Múltiplo. Para isso, foram executados testes com instâncias artificiais e reais, sendo essas últimas instâncias de um problema tradicional da Bioinformática denominado Problema da Identificação de Genes.

Palavras-chave: Otimização Combinatória, Problema do Alinhamento Spliced Múltiplo, Programação Linear Inteira, Aproximação, Biologia Computacional.

Abstract

Recent advances in specific areas of Computer Science as Computational Biology brought to light several new problems involving sequences, while traditional problems become more difficult given the significant amount of data generated in the last years. The interest here is to study a specific problem which involves sequences, called Multiple Spliced Alignment Problem, extending a previously study conducted by Kishi and Adi about that same problem. While in this previous study it was shown that the Multiple Spliced Alignment Problem is NP-complete and heuristics have been proposed for the problem, this paper aims to suggest an approximation algorithm and an integer linear programming formulation for it, allowing to confront these new approaches with the already developed heuristics for the Multiple Spliced Alignment Problem. For this purpose, experimental tests with real and artificial instances were executed, with the latter being instances of a traditional problem in Bioinformatics called Gene Prediction Problem.

Keywords: Combinatorial Optimization, Multiple Spliced Alignment Problem, Integer Linear Programming, Approximation, Computational Biology.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

1.1 Considerações iniciais

A Teoria da Computação aborda os mais variados tipos de problemas, que podem ser divididos basicamente em duas grandes classes: os problemas de decisão e os de otimização combinatória. Enquanto os primeiros buscam obter uma resposta do tipo "sim" ou "não" como saída, os problemas de otimização combinatória buscam uma solução ótima, ou seja, uma possível combinação de valores das variáveis do problema que maximize (ou minimize) o valor da função objetivo empregada. A finalidade do estudo desses problemas é o fato de que eles, em geral, modelam de forma matemática situações reais, podendo assim servir como auxílio na solução de problemas práticos. O problema do Caixeiro Viajante [2], por exemplo, pode ser visto como uma modelagem matemática para o problema enfrentado pelos Correios na determinação de uma rota que otimize a entrega de correspondências.

Dentre os vários problemas estudados em Teoria da Computação estão aqueles que envolvem sequências, que nos últimos anos ganharam bastante ênfase [23]. Esses problemas encontram aplicações em áreas variadas como Biologia, Bancos de Dados e Processamento de Textos, tornando assim problemas de grande interesse prático. O Problema do Alinhamento de Sequências e o de Casamento de Padrões, por exemplo, correspondem a problemas que envolvem sequências com motivações práticas no contexto da Biologia Molecular.

O foco principal deste trabalho é o aprofundamento dos estudos em um problema de otimização combinatória que envolve sequências denominado Problema do Alinhamento *Spliced* Múltiplo. Mais especificamente, pretende-se aqui estender o estudo sobre esse problema iniciado por Kishi e Adi em [24, 25], que definiram sua complexidade, propondo e avaliando quatro heurísticas para ele.

1.2 Objetivos

A intenção deste estudo é abordar o Problema do Alinhamento *Spliced* Múltiplo através de algoritmos de aproximação e programação linear inteira. Por meio dessas abordagens, visa-se implementar soluções algorítmicas aproximadas e exatas para o

problema e comparar a qualidade dos resultados obtidos com os alcançados empregando-se as heurísticas já propostas para ele no trabalho original.

1.3 Contribuições

Este trabalho apresenta as seguintes contribuições:

- Revisão teórica acerca do Problema do Alinhamento *Spliced* Múltiplo, com o estudo de problemas semelhantes na literatura, no intuito de encontrar estratégias que pudessem ser aproveitadas no desenvolvimento da aproximação ou ainda do modelo de programação linear inteira;
- Desenvolvimento de um algoritmo de aproximação de razão 3 para o Problema do Alinhamento *Spliced* Múltiplo;
- Formulação de um modelo de programação linear inteira para o Problema do Alinhamento *Spliced* Múltiplo;
- Desenvolvimento de um algoritmo *branch-and-bound* para o problema, aplicando o modelo de programação linear inteira elaborado;
- Implementação dos algoritmos de aproximação e de *branch-and-bound* desenvolvidos para o Problema do Alinhamento *Spliced* Múltiplo (disponíveis em `http://www.facom.ufms.br/~said/msaapprox/codes/`);
- Criação de um conjunto de instâncias artificiais de testes para ser utilizado como *benchmark* das ferramentas desenvolvidas (disponível em `http://www.facom.ufms.br/~said/msaapprox/benchmark/`);
- Avaliação empírica das ferramentas implementadas, assim como das heurísticas propostas em [24, 25], empregando as instâncias de testes propostas;
- Comparação da qualidade dos resultados obtidos pelas diferentes abordagens;
- Avaliação da ferramenta baseada no algoritmo de aproximação utilizando as mesmas instâncias reais de [24, 25] para verificar sua aplicação na tarefa prática de identificação de genes.

1.4 Organização do texto

Após esta breve introdução, o Capítulo 2 apresenta alguns conceitos importantes de Ciência da Computação necessários para uma melhor compreensão do trabalho, e estabelece as notações a serem utilizadas no estudo e abordagem do problema em questão. A definição formal do Problema do Alinhamento Spliced Múltiplo, assim como de uma versão mais simples do problema e sua solução, são encontradas no Capítulo 3. No Capítulo 4, são citados trabalhos relacionados ao problema e pontuadas as suas principais contribuições para o desenvolvimento deste. O Capítulo 5 detalha a

3-aproximação proposta para o Problema do Alinhamento *Spliced* Múltiplo e a sua formulação de programação linear inteira é apresentada e discutida no Capítulo 6. O Capítulo 7 descreve os resultados alcançados com ambas as propostas, comparados com os anteriormente obtidos através das heurísticas, enquanto que o Capítulo 8 traz uma discussão final sobre tudo o que foi estudado, com as principais contribuições deste trabalho e sugestões de pesquisas futuras.

Capítulo 2

Conceitos fundamentais

Neste capítulo, serão introduzidos alguns conceitos e notações essenciais para o entendimento do problema a ser estudado, especialmente os relacionados com alinhamento de sequências e suas medidas de comparação. Além desses, são retomadas definições de problemas NP-completos e de otimização combinatória, algoritmos de aproximação, modelos de programação linear inteira e síntese proteica, que é importante para a aplicação prática do problema.

2.1 Definições básicas

Nesta seção, serão definidos os conceitos básicos relacionados a sequências empregados ao longo do trabalho. Eles foram reunidos a partir de [11], [17].

Um **alfabeto** Σ é um conjunto finito de caracteres. Tem-se uma **sequência** s construída sobre um alfabeto Σ quando um número finito de caracteres de Σ são dispostos sequencialmente, e a ordem em que eles aparecem é importante. Não é obrigatório que todos os caracteres do alfabeto façam parte da sequência, nem que apareçam apenas uma única vez. Por exemplo, com o alfabeto $\Sigma = \{A, B, E, L, M, O, P, R\}$, podemos formar as sequências $s_1 = PROBLEMA$, $s_2 = BOLA$ e $s_3 = ERRAR$, que representam palavras válidas da Língua Portuguesa. O conjunto de todas as sequências de comprimento finito construídas com os caracteres de Σ é denotado por Σ^*.

O **tamanho** de uma sequência s corresponde ao número de caracteres que a compõem, e é denotado por $|s|$. As sequências do exemplo anterior possuem tamanho $8, 4$ e 5 respectivamente. Quando o tamanho de uma sequência é zero, ou seja, quando ela não possui nenhum caractere do alfabeto, a sequência é dita **vazia** e denotada por ϵ. Quando uma sequência não é vazia, utiliza-se a notação $s[i]$, com $i \in \mathbb{Z}^+$ e $1 \leq i \leq |s|$, para representar o $i-$ésimo caractere de s.

Uma **subsequência** s' de uma sequência s é outra sequência obtida através da remoção de (um, vários ou até nenhum) caracteres de s. Já um **segmento** de uma sequência s é uma subsequência s' necessariamente contínua de s. Ou seja, para a obtenção de um segmento a partir de s, podem ser removidos apenas caracteres consecutivos do início e/ou final de s. Em ambas as definições, o tamanho da subsequência ou do segmento pode variar de 0 a $|s|$. *PROLE*, por exemplo, é uma subsequência

de s_1, enquanto que *LEMA* é uma subsequência e um segmento de s_1. É possível indicar claramente um segmento não vazio de s iniciando no caractere $s[i]$ e finalizando no caractere $s[j]$ de uma sequência s por $s[i..j]$, com $1 \leq i, j \leq |s|$, apesar de não haver uma notação para a representação de uma subsequência de s. Finalmente, denota-se por $primeiro(s[i..j]) = i$ a posição do primeiro caractere de $s[i..j]$ em s e por $último(s[i..j]) = j$ a posição do último caractere do segmento $s[i..j]$ em s.

Dando continuidade aos conceitos sobre segmentos, tem-se que dois segmentos $s_1 = s[i..j]$ e $s_2 = s[k..l]$ de uma sequência s são **sobrepostos** se $i \leq k \leq j$ ou $k \leq i \leq l$. Caso não sejam sobrepostos, é possível definir uma ordem entre eles, em que s_1 é dito **predecessor** de s_2 se $j < k$, ou s_1 é **sucessor** de s_2 se $l < i$, relações representadas por $s_1 \prec s_2$ e $s_1 \succ s_2$, respectivamente. Ainda nesse contexto, um segmento é dito **prefixo** de s quando se inicia no primeiro caractere de s, sendo denotado por $s[1..p]$, com $1 \leq p \leq |s|$. Já um **sufixo** de s é um segmento de s que se encerra no último caractere da sequência, sendo denotado por $s[f..|s|]$, com $1 \leq f \leq |s|$. Assim, considerando a sequência s_1 do exemplo inicial, tem-se que $s_1[1..3] = PRO$ é um prefixo de s_1 e $s_1[5..8] = LEMA$ um sufixo dessa sequência, além de que $s_1[1..3] \prec s_1[5..8]$, não havendo portanto sobreposição entre eles.

Uma operação comum entre duas sequências s e t é a **concatenação** de s e t, que consiste em se obter uma nova sequência u posicionando os caracteres de t logo após os de s. A nova sequência gerada é denotada por $u = \overline{st}$. Assim, o tamanho de u é $|s| + |t|$. Concatenando as sequências s_1 e s_2 tomadas como exemplo, obtemos $\overline{s_1 s_2}$ $=PROBLEMABOLA$, cujo tamanho é $|s_1| + |s_2| = 8 + 5 = 13$.

Um conjunto $\mathcal{B} = \{b_1, b_2, ..., b_n\}$ de n segmentos de uma sequência s é dito um **conjunto ordenado de segmentos** se, para todos os segmentos de $\mathcal{B}$, for válida uma das relações a seguir, com $i \in \mathbb{Z}^+$ e $1 \leq i < |\mathcal{B}|$,:

- $primeiro(b_i) \prec primeiro(b_{i+1})$ OU
- $primeiro(b_i) = primeiro(b_{i+1})$ E $último(b_i) \prec último(b_{i+1})$

Em outras palavras, os segmentos de $\mathcal{B}$ devem estar ordenados de maneira não-decrescente pelas posições iniciais de cada segmento e, em caso de empate, pelas suas posições finais para que $\mathcal{B}$ seja considerado um conjunto ordenado de segmentos.

Por fim, um subconjunto $\Gamma = \{b_1, b_2, ..., b_p\}$ de um conjunto ordenado de segmentos $\mathcal{B}$ tal que $b_1 \prec b_2 \prec ... \prec b_p$ é chamado de **cadeia de segmentos** de $\mathcal{B}$. Ou seja, Γ não possui segmentos sobrepostos. A concatenação dos segmentos de Γ, isto é, $b_1 \bar{b_2} ... b_p$ é representada por $\bar{\Gamma}$.

Esses serão os conceitos e notações primordiais para a compreensão do problema a ser detalhado no Capítulo 3. Porém, ainda é necessário introduzir as definições de alinhamento de sequências, assim como um algoritmo conhecido para realizar esta tarefa. Isso será feito na Seção 2.2 a seguir.

2.2 Alinhamento de sequências

Especialmente no contexto da Biologia Molecular, a manipulação de sequências implicam situações em que se faz necessário compará-las, como forma de identificar semelhanças entre elas. Para tanto, o método mais estudado e comumente empregado é chamado de alinhamento de sequências, que será detalhado nesta seção, escrita com base em [12], [17], [33] e [38].

De maneira informal, pode-se dizer que alinhar duas sequências s_1 e s_2, de tamanhos n e m respectivamente, consiste em determinar uma função bijetora que associe cada caractere de s_1 a um caractere de s_2, respeitando a ordem deles nas suas respectivas sequências. Como as duas sequências podem ter tamanhos diferentes, é usado um caractere especial para compensar essa diferença. Esse caractere é chamado de **espaço** e representado por '$-$'. Assim, um caractere de s_1 pode ser associado a outro de s_2 (sendo eles iguais ou diferentes) ou a um espaço '$-$', e vice-versa. A única restrição está no fato de que um '$-$' em s_1 nunca pode corresponder a outro '$-$' em s_2, e vice-versa. Com isso, conta-se agora com um novo alfabeto Σ', que nada mais é do que $\Sigma \cup \{\text{'}-\text{'}\}$. Em [12], Brito define formalmente o que é um alinhamento de sequências:

Alinhamento de duas sequências: um alinhamento A de duas sequências s_1 e s_2 é uma matriz de ordem $2 \times k$, com $\max\{|s_1|, |s_2|\} \leq k \leq |s_1| + |s_2|$, cujas entradas são elementos de Σ', tal que:

- para cada i, $1 \leq i \leq 2$, existe um conjunto $J_i = \{j_1, j_2, ..., j_{|s_i|}\} \subseteq \{1, 2, ..., k\}$, com $j_1 < j_2 < ... < j_{|s_i|}$ e tal que $A[i][j_1]A[i][j_2]...A[i][j_{|s_i|}] = s_i$;
- para todo $j \in \{1, 2, ..., k\} - J_i$, temos que $A[i][j] =$'$-$';
- não existe j tal que $A[1][j] =$ '$-$' $= A[2][j]$.

Um exemplo de alinhamento das sequências $s_1 = PROBLEMA$ e $s_2 = ROLAR$ pode ser visto na Tabela 2.1 a seguir:

Tabela 2.1: Exemplo de alinhamento de duas sequências s_1 e s_2.

$A_1 =$

P	R	O	B	L	E	M	A	-
-	R	O	-	L	-	-	A	R

É possível notar que o alinhamento de duas sequências não é único. Há inúmeras outras formas de alinhar-se as sequências s_1 e s_2 do exemplo anterior segundo a definição descrita. Uma alternativa é mostrada na Tabela 2.2.

Tabela 2.2: Outro exemplo de alinhamento das duas sequências s_1 e s_2.

$A_2 =$

P	R	O	B	-	L	E	M	A
R	-	O	-	L	-	A	R	-

Um conceito muito importante relacionado ao alinhamento de sequências é o de função de pontuação. Uma **função de pontuação** $\omega : \Sigma' \times \Sigma' \to \mathbb{R}$ é uma função que atribui um valor real para cada par de caracteres de um alinhamento, ou seja,

uma pontuação para cada coluna possível de um alinhamento. Dessa forma, pode-se calcular uma pontuação para um alinhamento, tornando viável a comparação de diferentes alinhamentos para as mesmas duas sequências. Isso permite a escolha do melhor alinhamento para cada aplicação em especial. A **pontuação** de um alinhamento A de duas sequências s_1 e s_2 considerando uma função de pontuação ω é denotada por $Pontuação_\omega(A)$ e dada por $\sum_{i=1}^{k} \omega(A[1][i], A[2][i])$.

Embora pela definição da função de pontuação seja permitido atribuir uma pontuação diferente a cada possível par de caracteres dentro do alfabeto Σ', não é isso que costuma ser observado na prática. Em geral, é comum classificar as $|\Sigma'| \times |\Sigma'|$ combinações em três classes, cada qual recebendo um determinado peso. Dessa maneira, simplifica-se bastante a função de pontuação, ao passo que a torna mais adequada para ser empregada na comparação de alinhamentos diversos de um mesmo par de sequências. Os três casos considerados são:

1. `match`: os caracteres em A[1][j] e A[2][j] são iguais; houve portanto um *acerto*;
2. `mismatch`: os caracteres em A[1][j] e A[2][j] são diferentes; houve portanto um *erro*;
3. `space`: A[1][j] ou A[2][j] representam um '$-$' inserido no alinhamento, ocorrendo portanto um *espaço* em uma das sequência e um caractere na outra.

Vale lembrar que a única possibilidade descartada é a de se alinhar um espaço em uma das sequências com um espaço na outra sequência, pois essa é uma restrição prevista na definição de alinhamento de sequências. Atribui-se então um valor real às possibilidades descritas acima através de uma função de pontuação ω, tornando possível calcular um valor total referente ao alinhamento, $\texttt{Pontuação}_\omega(A)$, e utilizá-lo como medida comparativa para escolher o mais adequado em cada situação.

A definição de medidas é essencial para a comparação de sequências. Nesse contexto, é usual estabelecer a relação entre duas sequências s_1 e s_2 medindo a sua similaridade, denotada por $\texttt{sim}(s_1, s_2)$. Em [17], Gusfield formaliza essa medida da seguinte forma:

Similaridade de Duas Sequências: dada uma função de pontuação ω sobre um alfabeto Σ', a *similaridade* de duas sequências s_1 e s_2 consiste no valor do alinhamento A de s_1 e s_2 cujo valor de $\texttt{Pontuação}_\omega(A)$ é máximo, ou seja, $\texttt{sim}_\omega(s_1, s_2) = \max\{\texttt{Pontuação}_\omega(A)\}$.

O valor de $\texttt{sim}_\omega(s_1, s_2)$ é então chamado de **valor do alinhamento ótimo**, e consequentemente A é um **alinhamento ótimo** de s_1 e s_2. Funções de pontuação para medidas de similaridade em geral atribuem valores positivos ou iguais a zero para `matches` e valores negativos para `mismatches` ou `spaces`, favorecendo coincidências entre as sequências e penalizando suas diferenças. Um exemplo de ω normalmente utilizada para cálculos de similaridade é o seguinte: `match` $= 1$, `mismatch`$= -1$ e `space` $= -2$.

Retomando os exemplos de alinhamento das tabelas 2.1 e 2.2 e aplicando a função de pontuação ω exemplificada acima, sobre eles, obtemos a pontuação de A_1 e A_2, como é mostrado nas tabelas 2.3 e 2.4

Tabela 2.3: Aplicação da função de pontuação ω no primeiro exemplo de alinhamento. $\texttt{Pontuação}_\omega(A_1) = -6$.

$A_1 =$

P	R	O	B	L	E	M	A	-
-	R	O	-	L	-	-	A	R
-2	1	1	-2	1	-2	-2	1	-2

Tabela 2.4: Aplicação da função de pontuação ω no segundo exemplo de alinhamento. $\texttt{Pontuação}_\omega(A_2) = -12$.

$A_2 =$

P	R	O	B	-	L	E	M	A
R	-	O	-	L	-	A	R	-
-1	-2	1	-2	-2	-2	-1	-1	-2

Observando as tabelas 2.3 e 2.4, tem-se que A_1 é um alinhamento de maior pontuação em relação a A_2, sendo inclusive um alinhamento ótimo de s_1 e s_2.

A busca pela pontuação de um alinhamento ótimo de duas sequências pode ser definida como um problema computacional, chamado Problema da Similaridade de Duas Sequências:

Problema da Similaridade de Duas Sequências (PSS): dado um alfabeto Σ, em que $\{\text{`}-\text{'}\} \notin \Sigma$, duas sequências s_1 e s_2 sobre Σ e uma função de pontuação ω, deseja-se encontrar o valor de $\texttt{sim}_\omega(s_1, s_2)$.

Um problema intimamente relacionado ao da similaridade é o Problema do Alinhamento de Duas Sequências. Isso porque no problema em estudo é importante obter-se não apenas o valor do alinhamento ótimo entre duas sequências, mas também o próprio alinhamento. Sendo assim, encontrar um alinhamento ótimo entre duas sequências corresponde a um problema de otimização combinatória, em que dada uma função ω e duas sequências s_1 e s_2, busca-se por um alinhamento de s_1 e s_2 de maior pontuação dentre todos os possíveis alinhamentos de s_1 e s_2. Formalizando o problema do alinhamento entre duas sequências, tem-se que:

Problema do Alinhamento de Duas Sequências: dado um alfabeto Σ, em que $\{\text{`}-\text{'}\} \notin \Sigma$, duas sequências s_1 e s_2 sobre Σ e uma função de pontuação ω, quer-se encontrar um alinhamento A de s_1 e s_2 tal que $\texttt{Pontuação}_\omega(A) = \texttt{sim}_\omega(s_1, s_2)$.

Na próxima seção, será descrito um algoritmo para solucionar o problema da similaridade seguido do problema do alinhamento de duas sequências: o algoritmo de Needleman-Wunsch.

2.2.1 O algoritmo de Needleman-Wunsch

Como a determinação de um alinhamento ótimo de duas sequências está diretamente relacionada à obtenção do valor da similaridade entre elas, é conveniente abordar esses dois problemas de forma conjunta. Assim, a partir do cálculo da similaridade entre

duas sequências é possível obter um alinhamento com este valor, e possa ser considerado ótimo.

Uma solução para os problemas do alinhamento e similaridade de duas sequências foi sugerida por Needleman e Wunsch em [33], ficando portanto conhecida como **algoritmo de Needleman-Wunsch**. Através da técnica de programação dinâmica, a abordagem por eles proposta encontra primeiramente a similaridade entre duas sequências dadas como entrada, permitindo no passo seguinte realizar a reconstrução do alinhamento cuja pontuação é igual a este valor.

Antes de apresentar a solução proposta por Needleman e Wunsch em [33], vale apontar algumas características intrísecas ao problema da similaridade de duas sequências. A primeira observação diz respeito à sua natureza recursiva. A similaridade de $s_1[1..i]$ e $s_2[1..j]$ depende diretamente do valor da similaridade de seus prefixos, $sim(s_1[1..i-1], s_2[1..j-1])$, acrescida da respectiva pontuação definida pela função ω para o par $(s_1[i], s_2[j])$, sendo possível defini-la pela seguinte recorrência, adaptada de [17]:

$$\texttt{sim}(s_1[1..i+1], s_2[1..j+1]) = \max \begin{cases} \texttt{sim}(s_1[1..i], s_2[1..j+1]) + \omega(s_1[i+1], \text{`-'}) \\ \texttt{sim}(s_1[1..i+1], s_2[1..j]) + \omega(\text{`-'}, s_2[j+1]) \\ \texttt{sim}(s_1[1..i], s_2[1..j]) + \omega(s_1[i+1], s_2[j+1]) \end{cases} \tag{2.1}$$

Assim, pode-se estabelecer intuitivamente um algoritmo recursivo que quebra o problema original em subproblemas menores, resolve-os através de chamadas recursivas e utiliza suas soluções parciais para a obtenção da solução final. A cada nova chamada a esse algoritmo recursivo, três comparações devem ser efetuadas de acordo com as possibilidades de se estender uma solução parcial já obtida: com um `match`, um `mismatch` ou com um `space`. Porém, explorando essas possibilidades conforme a Recursão 2.1, essa abordagem gera $\mathcal{O}(3^{n+m})$ comparações de caracteres, com $|s_1| = n$ e $|s_2| = m$. Ou seja, apesar de sua simplicidade, este algoritmo apresenta complexidade exponencial, o que o torna inviável para valores de n e m grandes.

Com uma análise mais detalhada do algoritmo, pode-se observar que apesar do número de comparações de caracteres da solução proposta acima ser exponencial, muitas de suas chamadas recursivas calculam valores já determinados em momentos anteriores, realizando portanto tarefas redundantes. Uma forma de contornar esse problema é empregando a técnica de programação dinâmica, cujo cerne gira em torno de armazenar os valores já calculados para uso posterior no decorrer da construção de uma solução para o problema. Uma vez verificado que o número de comparações de caracteres distintas realizadas é de apenas $(n+1) \times (m+1)$, referentes às possíveis combinações de caracteres de s_1 e s_2, conclui-se que é possível armazenar os seus resultados em uma matriz M de ordem $(n+1) \times (m+1)$, denominada **matriz de alinhamento**, em que as linhas correspondem aos caracteres da sequência s_1 e as colunas aos de s_2. Dessa forma, cada cálculo da similaridade entre os prefixos das sequências é realizado apenas uma vez e armazenado em uma posição da matriz, tornando o seu acesso futuro direto. Mais especificamente, interpreta-se o valor contido em $M[i][j]$ como sendo a similaridade de $s_1[1..i]$ e $s_2[1..j]$. Após preenchida por inteiro, a matriz M armazenará então em $M[n][m]$ a similaridade procurada, e o tempo de execução passa a ser polinomial, mais especificamente $\mathcal{O}(nm)$.í

Ainda sobre a matriz de alinhamento M, a ordem em que ela é preenchida deve ser cuidadosamente escolhida, para permitir que os valores necessários para completar uma nova posição já tenham sido gerados. É essencial também a correta inicialização da matriz. Conforme a definição, a posição $M[0][0]$ representa a similaridade de duas sequências vazias, recebendo portanto o valor 0. A primeira coluna da matriz M indica a similaridade de prefixos de s_1 e uma sequência vazia, devendo portanto a posição $M[i][0]$ ser preenchida recursivamente por $M[i-1][0] + \texttt{space}$, para todo $1 \leq i \leq n$. Por sua vez, o preenchimento da primeira linha de M é similar ao da primeira coluna: $M[0][j] = M[0][j-1] + \texttt{space}$, para todo $1 \leq j \leq m$, uma vez que essa linha representa a similaridade dos prefixos de s_2 e uma sequência vazia.

Feitas as inicializações necessárias, a matriz M deve ser preenchida de cima para baixo, da esquerda para a direita, um elemento por vez. Para encontrar o valor da similaridade correspondente à posição atual, as três possibilidades devem ser analisadas e a mais vantajosa escolhida. De forma mais clara, as verificações para preencher $M[i][j]$, com $1 \leq i \leq n$ e $1 \leq j \leq m$, devem ser as seguintes, cada uma representando um possível casamento de caracteres de s_1, s_2 ou um espaço:

1. $s_1[i]$ e $s_2[j]$: $M[i-1][j-1] + \omega(s_1[i], s_2[j])$
2. $s_1[i]$ e '$-$': $M[i-1][j] + \omega(s_1[i],\text{'}-\text{'})$
3. '$-$' e $s_2[j]$: $M[i][j-1] + \omega(\text{'}-\text{'}, s_2[j])$

Esses passos encontram-se formalizados no Algoritmo 2.1, que transcreve o algoritmo de Needleman-Wunsch [33].

ALGORITMO 2.1 Needleman-Wunsch(s_1, s_2, ω)

Entrada: Sequências s_1 e s_2 a serem alinhadas e uma função de pontuação ω.
Saída: O valor da similaridade $sim(s_1, s_2)$ de s_1 e s_2.

```
1: //Inicialização da matriz
2: n ← |s1|;
3: m ← |s2|;
4: M[0][0] ← 0;
5: para i ← 1 até n faça
6:    M[i][0] ← M[i − 1][0] + ω(s1[i],'−');
7: para j ← 1 até m faça
8:    M[0][j] ← M[0][j − 1] + ω('−', s2[j]);

9: //Preenchimento da matriz
10: para i ← 1 até n faça
11:    para j ← 1 até m faça
12:       M[i][j] ← max(M[i − 1][j − 1] + ω(s1[i], s2[j]),
                        M[i][j − 1] + ω('−', s2[j]),
                        M[i − 1][j] + ω(s1[i],'−'));

13: devolva M[n][m];
```

			T	A	G	C
		0	1	2	3	4
	0	**0**	**-2**	**-4**	**-6**	**-8**
G	1	**-2**	**-1**	**-3**	**-3**	**-5**
G	2	**-4**	**-3**	**-2**	**-2**	**-4**
C	3	**-6**	**-5**	**-4**	**-3**	**-1**

Figura 2.1: Matriz de alinhamento para as sequências $s_1 = GGC$ e $s_2 = TAGC$ em questão produzida pelo Algoritmo 2.1.

Como é possível observar, é o preenchimento da matriz que dita a complexidade do Algoritmo 2.1. A linha 6 é executada n vezes, a linha 9 se repete m vezes e a linha 14, primordial para o preenchimento da matriz como um todo, nm vezes. Sendo assim, a complexidade dele é $\mathcal{O}(nm)$, como mencionado anteriormente.

A título de ilustração, sejam as sequências $s_1 = GGC$ e $s_2 = TAGC$, ambas construídas sobre o alfabeto $\Sigma = \{A, C, G, T\}$, e a função de pontuação $\omega(x, y)$ como definida anteriormente: $\omega(x, y) = 1$ para `match`, $\omega(x, y) = -1$ para `mismatch` e $\omega(x, y) = -2$ para inserções de espaços em alguma das sequências (`space`). A execução do Algoritmo 2.1 para essas entradas irá gerar a matriz de alinhamento apresentada na Figura 2.1. A similaridade entre s_1 e s_2, armazenada em $M[n][m] = M[3][4]$, é portanto igual a 1.

Encontrando o alinhamento ótimo

Em várias aplicações, além de se calcular a similaridade entre duas sequências, é importante também obter um alinhamento ótimo com esse valor. Uma maneira de se obter um alinhamento ótimo das sequências através da matriz de alinhamento é associar ponteiros a cada célula indicando a origem do valor que gerou seu preenchimento. Assim, uma vez preenchida a matriz de alinhamento como descrito na Subseção 2.2.1, é possível encontrar o alinhamento em questão de maneira praticamente direta. Inicia-se em $M[n][m]$, posição que encerra o valor da similaridade, e segue-se um dos caminhos indicados pelos ponteiros até atingir-se a posição $M[0][0]$. Considerando uma posição $M[i][j]$ da matriz, passos *horizontais* que levam à célula $M[i][j-1]$ indicam a inserção de um espaço em s_1 para casar com $s_2[j]$ (`space`); passos *verticais* direcionando a $M[i-1][j]$ representam a inserção de um espaço em s_2 para casar com $s_1[i]$ (`space`), e passos *diagonais* equivalem a um casamento entre $s_1[i]$ e $s_2[j]$, podendo ser tanto um `match` como um `mismatch`. A matriz da Figura 2.1 com os ponteiros pode ser observada na Figura 2.2.

O procedimento descrito pode ser visto de forma mais detalhada no Algoritmo 2.2. Vale salientar que, como o alinhamento ótimo pode não ser único, uma mesma célula da matriz M pode incluir mais de um ponteiro indicando uma posição de origem viável.

			T	A	G	C
		0	1	2	3	4
	0	**0**	← **-2**	← **-4**	← **-6**	← **-8**
G	1	↑ **-2**	↖ **-1**	↖← **-3**	↖ **-3**	↖← **-5**
G	2	↑ **-4**	↖↑**-3**	↖ **-2**	↖ **-2**	↖← **-4**
C	3	↑ **-6**	↖↑**-5**	↖↑ **-4**	↖ **-3**	↖ **-1**

Figura 2.2: Matriz de alinhamento completa para as sequências $s_1 = GGC$ e $s_2 = TAGC$ produzida pelo Algoritmo 2.1, com a representação dos ponteiros. Considerando $M[i][j]$ como a célula corrente, a seta ↖ aponta para $M[i-1][j-1]$, a seta ← para $M[i][j-1]$ e a seta ↑ para $M[i-1][j]$.

ALGORITMO 2.2 Constrói-Alinhamento-Ótimo(s_1, s_2, M)

Entrada: Sequências s_1 e s_2 e matriz de alinhamento M referente às sequências s_1 e s_2.
Saída: Um alinhamento ótimo A entre s_1 e s_2.

```
1: //Inicializações
2: i ← |s1|;
3: j ← |s2|;
4: A ← ∅;

5: //Percorrendo a matriz M no sentido inverso
6: enquanto i > 0 E j > 0 faça
7:    se ↖ ∈ M[i][j] então
8:       A ← A ∪ (s1[i], s2[j]);
9:       i ← i − 1;
10:      j ← j − 1;
11:   senão
12:      se ↑ ∈ M[i][j] então
13:         A ← A ∪ (s1[i],'−');
14:         i ← i − 1;
15:      senão
16:         A ← A ∪ ('−', s2[j]);
17:         j ← j − 1;

18: //Alinhando possível prefixo restante de s1 com espaço
19: enquanto i > 0 faça
20:    A ← A ∪ (s1[i],'−');
21:    i ← i − 1;

22: //Alinhando possível prefixo restante de s2 com espaço
23: enquanto j > 0 faça
24:    A ← A ∪ ('−', s2[j]);
25:    j ← j − 1;

26: devolva A;
```

2.2.2 Medidas de distância

Outra forma de mensurar quão parecidas são duas sequências é através das chamadas medidas de distância, baseadas em funções de pontuação que priorizam evidenciar as diferenças entre duas sequências s_1 e s_2 ao invés das suas semelhanças, como ocorre na similaridade. Destaca-se dentre essas medidas a distância de edição, que será introduzida a seguir com base nos conceitos encontrados em [34].

A **distância de edição** indica o número mínimo de *operações de edição* (inserção, remoção ou substituição de caracteres) que devem ser realizadas em uma sequência s_1 para transformá-la em outra s_2, observando que s_1 e s_2 não precisam ter necessariamente o mesmo tamanho. A fim de formalizar sua definição, é necessário apresentar o conceito de métrica.

Considere uma função d: $E \times E \mapsto \mathbb{R}$, com E sendo um conjunto não-vazio qualquer. A função d caracteriza uma **métrica** sobre E se, e somente se, para todo $x, y, z \in E$, d apresenta as seguintes características:

1. **Positividade:** $d(x, y) \geq 0$;
2. **Separação:** $d(x, y) = 0$ se, e somente se, $x = y$;
3. **Simetria:** $d(x, y) = d(y, x)$;
4. **Desigualdade triangular:** $d(x, y) \leq d(x, z) + d(z, y)$.

No contexto de comparação de sequências, a positividade estabelece que a distância entre uma sequência x e outra y deve sempre ser um valor não-negativo. Afinal, como ela representa o número de operações necessárias para transformar uma sequência x na sequência y, não faz sentido pensar em valores negativos. Sobre a separação, ela indica que a distância entre duas sequências apenas será nula se as duas sequências forem iguais, pois é a única situação em que não são necessárias operações para transformar uma sequência na outra. A simetria assegura que o número mínimo de operações necessárias para transformar a sequência x na y é o mesmo caso deseje-se realizar o inverso, obter-se x a partir de y. Ou seja, a distância independe da ordem em que as sequências são comparadas. E a quarta propriedade das métricas, a desigualdade triangular, garante que transformar-se a sequência x na z, e depois z na y exige mais ou a mesma quantidade de operações necessárias para se obter y diretamente de x.

Seja ainda $\delta : \Sigma' \times \Sigma'$ uma métrica representando uma função de pontuação para as operações de edição. Para dois caracteres $a, b \in \Sigma$, o custo de substituir-se a por b é dado por $\delta(a, b)$, a remoção de a custa $\delta(a, '-')$ e $\delta('-', b)$ indica o custo da inserção de b. Com isso, o **custo total** da sequência de operações é dado pelo somatório de seus termos individuais.

Dadas duas sequências s_1 e s_2 construídas sobre um alfabeto Σ e uma métrica $\delta : \Sigma' \times \Sigma' \mapsto \mathbb{R}$, define-se $\delta-$**distância de edição** de duas sequências s_1 e s_2 como sendo a sequência de operações de edição necessárias para transformar s_1 em s_2 de custo mínimo, ou seja, cujo somatório do valor das operações seja o menor possível, e denota-se essa distância por $d_\delta(s_1, s_2)$. Se for adotado um custo unitário para cada operação de edição, sendo δ uma métrica tal que para todo $a, b \in \Sigma'$ $\delta(a, b) = 1$ se $a \neq b$

e $\delta(a,b) = 0$ caso contrário, então a $\delta-$distância de edição é dita apenas **distância de edição** ou ainda **Distância de Levenshtein**, e é denotada por d_L.

Calculando a distância de edição

O Algoritmo Needleman-Wunsch, da forma como foi apresentado anteriormente, encontra o valor da similaridade de duas sequências. Porém, é possível adaptá-lo para que determine a distância de edição entre duas sequências s_1 e s_2. O primeiro ajuste diz respeito aos parâmetros. A função de pontuação exigida deve ser aquela definida para a Distância de Levenshtein (0 para `match`, 1 para `mismatch` e 1 para `space`). Estabelecido isso, fica evidente que agora o problema não deve mais ser tratado como um problema de maximização, mas sim como de minimização. Afinal, deseja-se obter uma sequência de operações de edição de custo mínimo. Assim, na Linha 14 do Algoritmo 2.1, deve-se substituir a função `max` por `min`, para que seja armazenado na matriz o menor valor dentre os três possíveis.

Uma vez encontrada a distância de edição entre duas sequências, a construção de uma menor série de operações que transforma uma sequência na outra é realizada de maneira semelhante à descrita para a determinação de um alinhamento ótimo de duas sequências. Considerando dois caracteres $a, b \in \Sigma$, agora a seta '$\nwarrow$' representa uma substituição do caractere a por b, a seta '$\leftarrow$' indica uma inserção de b e por fim, a seta '$\uparrow$' equivale a uma remoção de a.

2.3 Complexidade computacional

Cormen *et al.* definem em [11] um **algoritmo** como um procedimento computacional bem definido que recebe como **entrada** um valor (ou conjunto de valores) e após uma sequência de passos computacionais, transforma os dados de entrada em um valor (ou conjunto de valores) denominado **saída**. um problema p para o qual seja possível apresentar um algoritmo A que o resolva de forma correta, tomando uma entrada e devolvendo a saída esperada correspondente, é dito um **problema computacional**. O algoritmo A pode ainda indicar que a entrada não possui solução, se este for o caso. Uma entrada recebida por um algoritmo em geral é referenciada como uma **instância** I_p de um problema p.

Em várias situações, é importante quantificar o tempo gasto na execução de um algoritmo A, de modo a determinar sua eficiência e viabilidade. Para isso, considere uma função $T_A(I_p)$ que relaciona o tamanho da instância I_p de um problema p e o tempo necessário para o algoritmo A solucioná-la. O cálculo dessa função é normalmente realizado por uma *análise assintótica* dos algoritmos.

Dentre as três notações utilizadas para indicar a eficiência assintótica de um algoritmo, $\mathcal{O}$, Ω e Θ, a mais empregada para se analisar um algoritmo é a $\mathcal{O}$. Isso ocorre pelo fato desta notação determinar um limite assintótico superior do tempo de execução de um algoritmo, considerando o pior caso de instância que este possa receber como entrada. Conforme a definição em [11], dadas duas funções assintoticamente não negativas $f(n) \geq 0$ e $g(n) \geq 0$, se existirem uma constante $c_1 > 0$ e $n_1 \geq 0$ tal que para todo $n \geq n_1$,

$$f(n) \leq c_1 * g(n)$$

tem-se então que $f(n) = \mathcal{O}(g(n))$, ou seja, $g(n)$ é um limite assintótico superior da função $f(n)$ para um n suficientemente grande.

Dependendo da ordem da função $g(n)$, é possível classificar um algoritmo em eficiente ou ineficiente. Um algoritmo que apresente complexidade de tempo $\mathcal{O}(n^k)$, em que n representa o tamanho da instância e k é uma constante, é dito **algoritmo eficiente**, ou **polinomial**. Já um algoritmo cujo tempo de execução não possui um limitante superior polinomial é classificado como um algoritmo **ineficiente**. Esse é o caso, por exemplo, de algoritmos cuja complexidade de tempo é $\mathcal{O}(a^n)$, em que $a \in \mathcal{R}$ e $a > 1$ e n novamente indica o tamanho da instância, os chamados algoritmos **exponenciais**.

2.4 Problemas NP-completos

Dentre os problemas que aceitam solução computacional, é interessante distinguir os que aceitam soluções polinomiais, ou seja, apresentam um algoritmo eficiente que os resolva, dos que não podem ser solucionados de maneira eficiente. Sendo assim, é possível classificar os problemas conforme sua complexidade em três grandes classes, segundo a teoria da NP-Completude: P, NP e NPC [11, 26, 39].

A primeira classe de problemas é chamada de **P** e engloba os problemas solucionáveis em tempo polinomial, ou seja, os problemas eficientes. Isto é, problemas para os quais é conhecido um algoritmo A que, dada uma instância arbitrária I de tamanho n, encontra a solução ótima em tempo n^k para uma constante k. Problemas desse tipo são considerados mais fáceis, pois é possível resolvê-los em tempo razoável em função do tamanho da entrada. Esses problemas são conhecidos também como problemas **tratáveis**.

A classe **NP** inclui os problemas ineficientes, que em tempo polinomial podem ter uma solução verificada, mas não necessariamente encontrada. Em outras palavras, uma vez fornecida uma provável solução S para um problema NP, há um algoritmo polinomial capaz de confirmar se S é uma solução válida para o problema ou não. Contudo, não existe necessariamente um algoritmo A que consiga construir S gastando tempo polinomial. Estes problemas são vistos como problemas **difíceis**, ou **intratáveis**. Observa-se ainda que todos os problemas eficientes solucionados por algoritmos de tempo polinomial (pertencentes à classe P), também são coerentes com essa classificação, e logo diz-se que $P \subset NP$.

A terceira classe NPC inclui os problemas que estão presentes em NP e são considerados tão difíceis como qualquer outro desta classe, de forma que dentro dessa classe é possível reduzir qualquer problema a outro. Problemas que atendem a essa condição constituem a classe dos problemas **NPC**, ou **NP-completos**. Relacionados a essa classe, existem ainda os problemas **NP-difíceis**, que são problemas que podem ser reduzidos a um problema NP por serem pelo menos tão difíceis quanto eles, embora não necessariamente pertençam à classe NP.

Se for apresentado um algoritmo de tempo polinomial que resolva qualquer problema da classe, isso significa que todos os demais problemas também podem ser solucionados em tempo polinomial, pois estão todos relacionados através de reduções. Apesar de,

desde 1971, nenhum algoritmo do gênero ter sido sugerido com sucesso, até hoje também não foi apresentada nenhuma prova da sua inexistência absoluta. Sendo assim, essa é uma das questões em aberto mais importantes da Computação, e resume-se em provar se $P = NP$ ou $P \neq NP$.

Uma vez determinado que um problema é NP-completo, é pouco provável que se consiga solucioná-lo de forma exata em tempo viável. Portanto, é mais interessante buscar por soluções aproximadas ou heurísticas para o problema do que insistir no desenvolvimento de um algoritmo polinomial para resolvê-lo.

2.5 Problemas de otimização combinatória

Três conceitos principais definem um **problema de otimização combinatória**, segundo Johnson [20] e Carvalho *et al.*[8]:

- Um conjunto de **instâncias** $\mathcal{I} = \{I_1, ..., I_n\}$;
- Um conjunto de **soluções** $\texttt{Sol}(I_i) = \{S_1, ..., S_m\}$ para cada $I_i \in \mathcal{I}$; e
- Uma função f que associa um valor $\texttt{val}(S_j)$ a cada solução $S_j \in \texttt{Sol}(I_i)$.

Caso o conjunto $\texttt{Sol}(I_i)$ seja vazio, ou seja, não haja nenhuma solução para a I_i, a instância é dita **inviável**. Se houver ao menos uma solução viável para I_i, então a instância é dita **viável**.

Se o problema de otimização combinatória busca encontrar a solução S^* para uma dada instância I_i de valor máximo $\texttt{val}(S^*)$ associado quando submetida à função f, diz-se que o problema é de **maximização**. Caso o interesse seja o oposto, isto é, na solução S^* de valor mínimo associado, tem-se um problema de **minimização**. Em ambas as conjunturas, a solução S^* corresponde à **solução ótima** de I_i e seu valor é denotado por

$$\texttt{opt}(I_i)$$

Dado um conjunto finito $N = \{1, ..., n\}$, com pesos c_j para cada $j \in N$, e um conjunto F de subconjuntos viáveis de N, encontrar o subconjunto S de F de peso mínimo é um exemplo de problema de otimização combinatória de minimização. Esse problema pode ser escrito formalmente da seguinte maneira:

$$\min_{S \subseteq N} \{ \sum_{j \in S} c_j : S \in F \},$$

Por outro lado, encontrar o subconjunto S de F de peso máximo seria um problema de maximização.

No contexto de problemas de Otimização Combinatória que são NP-completos, a ideia intuitiva de se testar todas as possibilidades em busca da solução de peso mínimo ou máximo é até aplicável para instâncias de tamanho pequeno. Porém, para instâncias de tamanho considerável, passa a ser impraticável utilizar essa abordagem do tipo força bruta para solucionar esses problemas de forma eficiente. Sendo assim, é necessário

buscar-se estratégias mais elaboradas para resolver o problema, como Algoritmos de Aproximação, Algoritmos Probabilístico, Heurísticas e Programação Linear Inteira.

2.6 Algoritmos de aproximação

Embora permitam formalizar em termos matemáticos muitos problemas práticos, grande parte dos problemas de otimização combinatória são NP-difíceis, e portanto encontrar uma solução ótima para eles é uma tarefa complexa, até mesmo com auxílio computacional. Entretanto, nem sempre é essencial obter-se a solução exata para um problema, sendo uma solução aproximada mais do que suficiente para várias finalidades. É com base nessa observação que surgem os algoritmos de aproximação, que conseguem alcançar eficientemente uma solução viável para determinado problema de otimização, ao custo de uma redução na qualidade da solução por um fator pré-estabelecido. Maiores detalhes sobre algoritmos de aproximação como abordagem de problemas de otimização combinatória podem ser encontrados nesta seção, desenvolvida com base em [8], [22], [43] e [20].

Sejam dados um problema de otimização combinatória P e uma instância viável I de P, tal que $\texttt{val}(S) \geq 0$ para todo $S \in \texttt{Sol}(I)$. Um algoritmo A que devolve uma solução viável $S_A \in \texttt{Sol}(I)$ é dito uma $\alpha-$**aproximação** para o problema P se

$$\texttt{val}(S_A) \leq \alpha * \ \texttt{opt}(I),$$

para todas as instâncias viáveis I de P. O termo α é chamado de **razão de aproximação** do algoritmo, e pode ser dependente do tamanho de I. No caso do problema ser de minimização, tem-se que $\alpha \geq 1$, enquanto que para um problema de maximização a inequação anterior se torna

$$\texttt{val}(S_A) \geq \alpha * \ \texttt{opt}(I),$$

agora com $0 < \alpha \leq 1$. Sendo assim, uma $1-$aproximação para um problema de otimização é na verdade um algoritmo exato para ele, e encontra uma solução ótima para todas as instâncias viáveis I de P.

O fator α pode ser entendido como uma medida de qualidade da solução $\texttt{val}(S_A)$ encontrada pelo algoritmo em relação ao ótimo. Quanto mais "apertado" (*justo*) - ou seja, mais próximo de 1 - este limitante, melhor a qualidade do algoritmo de aproximação proposto, pois garante assim uma solução mais próxima à ótima.

Além de garantir que a relação entre a sua solução e a ótima esteja limitada por uma razão α, é também importante que um algoritmo seja polinomial no tamanho da instância I de entrada para que possa ser considerado um algoritmo de aproximação. Assim, configura-se a vantagem em utilizá-lo para buscar uma solução aproximada obtida em tempo eficiente, ao invés de insistir em um algoritmo exato que sempre forneça uma solução ótima, mas em tempo muitas vezes impraticável.

2.7 Programação linear inteira

Programação linear é outra abordagem largamente difundida para atacar problemas de otimização, em especial os NP-completos, e portanto merece ser considerada neste trabalho. Nesta seção, são introduzidas as principais definições da estratégia, compiladas de [4], [27] e [44].

Dado um vetor de custo $c = (c_1, ..., c_n)$, um problema de **programação linear** (PL) consiste em minimizar uma função linear $cx = \sum_{i=1}^{n} c_i x_i$, chamada **função objetivo**, sob todos os possíveis vetores $n-$dimensionais $x = (x_1, ..., x_n)$, estando sujeita a um conjunto de m **restrições** lineares. Nessas restrições, os coeficientes das variáveis x_i são representados por uma matriz A de ordem $m \times n$ e os dos termos à direita das suas desigualdades por um vetor $m-$dimensional $b = (b_1, ..., b_m)$, de forma que o problema pode ser definido como segue:

$$\begin{aligned} &\text{minimizar } cx \\ &\text{sujeita a } Ax \geq b \\ &\qquad\qquad x \geq 0 \end{aligned}$$

As variáveis $x_1, ..., x_n$ recebem o nome de **variáveis de decisão** e tem-se uma **solução viável** quando um vetor x satisfaz todas as restrições impostas. Uma solução viável x^* que minimize a função objetivo é chamada de **solução ótima**, para a qual vale:

$$cx^* \leq cx, \forall x \text{ viável}$$

Uma observação precipitada que pode surgir da análise dessas definições de programação linear é a de que apenas restrições de desigualdade podem ser definidas, uma vez que $Ax \geq b$. Essa afirmação é facilmente descartada quando compreende-se que, caso a $i-$ésima restrição de um problema seja do tipo $A_i x = b_i$, esta pode ser reescrita por meio de duas outras restrições: $A_i x \leq b_i$ e $A_i x \geq b_i$. Como $A_i x \leq b_i$ é equivalente a $-A_i x \geq -b_i$, tem-se que é possível expressar as soluções viáveis de qualquer problema de programação linear estritamente em termos de restrições de desigualdade da forma $A_i x \geq b_i$. Nesse mesmo contexto, embora as definições tenham sido descritas considerando um problema de minimização, é trivial estendê-las para problemas de maximização. Basta observar que maximizar uma função cx é análogo a minimizar $-cx$.

As variáveis de um problema de programação linear podem ainda estar sujeitas a restrições acerca dos valores que podem assumir: $x \geq 0$ (não-negativos), $x \leq 0$ (não-positivos), $x \in \mathbb{Z}$ (inteiros), $x \in \{0, 1\}^n$ (binários), ou simplesmente não apresentarem nenhuma restrição, as consideradas **variáveis livres**. Nesse sentido, é de especial interesse os problemas de programação linear cujos valores das variáveis estão restritos ao conjunto dos inteiros $\mathbb{Z}$, os chamados problemas de **programação linear inteira** (PLI). Eles podem ser classificados em:

1. *Programação Linear Inteira Mista (MIP)*: a restrição de assumir valores inteiros aplica-se apenas a uma parcela das variáveis do problema. Na definição a seguir, G é uma matriz de ordem $m \times p$, enquanto que h e y são vetores $p-$dimensionais.

$$\begin{aligned} &\text{minimizar } cx + hy \\ &\text{sujeita a } Ax + Gy \geq b \\ &\qquad x \geq 0 \\ &\qquad y \geq 0 \\ &\qquad y \in \mathbb{Z} \end{aligned}$$

2. *Programação Linear Inteira (PLI)*: os valores de todas as variáveis do vetor x estão restritos a valores inteiros.

$$\begin{aligned} &\text{minimizar } cx \\ &\text{sujeita a } Ax \geq b \\ &\qquad x \geq 0 \\ &\qquad x \in \mathbb{Z} \end{aligned}$$

3. *Programação Inteira Binária (PIB)*: os valores de todas as variáveis do vetor x estão restritos aos valores 0 e 1.

$$\begin{aligned} &\text{minimizar } cx \\ &\text{sujeita a } Ax \geq b \\ &\qquad x \in \{0,1\}^n \end{aligned}$$

Como os problemas de programação linear inteira são, em geral, NP-difíceis, emprega-se o método de **relaxação linear** para transformá-lo num problema de programação linear. Esse método consiste em substituir as restrições $\{0,1\}$ que fazem dele um problema de programação linear inteira por restrições lineares mais fracas, em que cada variável pertence ao intervalo $[0,1]$, resultando assim em um problema solucionável em tempo polinomial. Assim, cada restrição da forma $x_i \in \{0,1\}$ no problema inteiro original, passa a ser representada pelo par de restrições $0 \leq x_i \leq 1$. Uma vez resolvida essa versão mais simples, pode-se utilizar a solução encontrada como limitante na busca pela solução do problema original.

A técnica mais comum para resolver problemas de programação linear inteira consiste em usar enumeração implícita, ou *branch-and-bound*, em que as relaxações lineares fornecem os limitantes [10]. Proposto por Land e Doig em [30], o *branch-and-bound* consiste em um algoritmo que considera uma enumeração sistemática de todas as possíveis soluções, através de uma árvore de enumeração. Utilizando os limitantes inferior e superior da solução ajustados a cada nova iteração, é possível desconsiderar subárvores inteiras que representem soluções inviáveis ou subótimas para o problema, de forma a convergir mais rapidamente para a ótima.

2.8 Síntese proteica

Uma das aplicações práticas do problema estudado neste trabalho está relacionada com a tarefa de identificação de genes, tarefa essa constantemente enfrentada pelos biólogos

durante o estudo de um certo organismo. Antes de entrar em detalhes sobre essa atividade, é necessário explanar termos e processos importantes da Genética. Para isso, foram consultadas e sumarizadas definições encontradas em [40], [21] e [28].

No contexto da Biologia Molecular, é de grande importância o chamado *Dogma Central da Biologia*, ou simplesmente *Síntese Proteica*. Nele, duas das diversas moléculas presentes nos organismos celulares são fundamentais: o DNA e o RNA, os chamados ácidos nucleicos.

O **DNA** (ácido desoxirribonucleico) encerra informações responsáveis por gerenciar as tarefas das células e determinar o desenvolvimento, funcionamento e comportamento dos organismos. Já o **RNA** (ácido ribonucleico) regula o processo de transcrição e síntese de proteínas.

Os **nucleotídeos** constituem os blocos estruturais elementares dos ácidos nucleicos. Cada nucleotídeo possui três componentes: uma molécula de açúcar, uma molécula de fosfato e uma molécula contendo nitrogênio, a *base nitrogenada*. No DNA, o açúcar constituinte é a desoxirribose, enquanto no RNA é a ribose. Porém, é através da base nitrogenada que os nucleotídeos se diferenciam. No DNA, os quatro tipos de bases são *adenina* (A), *guanina* (G), *citosina* (C) e *timina* (T); no RNA, são mantidas as bases A, C, G, com a *uracila* (U) substituindo a *timina*. Assim, em ambos existem quatro tipos de nucleotídeos, e três deles são compartilhados pelos dois.

As informações contidas no DNA são então codificadas em sequências de nucleotídeos, geralmente representadas pela letra correspondente à sua base nitrogenada. Um trecho especial de nucleotídeos dessas sequências, que em geral contém as instruções para a síntese de proteína, é chamado de **gene**. Uma **proteína** consiste em uma sequência característica de **aminoácidos**, especificada por sequências de unidades codificantes elementares dentro de um gene, os **códons**, que são trincas de nucleotídeos adjacentes. Um códon especifica a incorporação de um aminoácido à proteína, de forma que a informação codificada dentro de um gene é utilizada para guiar a síntese de uma proteína.

O processo de formação de uma proteína apresenta duas etapas principais: a *transcrição* e a *tradução* (Figura 2.3). Primeiramente, a informação contida no DNA é copiada para uma molécula de RNA. Nesta cópia, chamada de transcrição, as bases nitrogenadas A, C, T e G no DNA são transcritas nas bases U, G, A e C no RNA, respectivamente. O RNA transcrito pode ainda sofrer algumas alterações antes do processo terminar, resultando então no RNA mensageiro, ou apenas mRNA, que contém toda a informação necessária para a síntese das proteínas.

A segunda fase da expressão da informação de um gene é denominada tradução. Neste estágio, o mRNA derivado de um gene atua como um molde para a síntese de uma proteína. Cada um dos códons presentes na sequência de mRNA determina um aminoácido a ser adicionado à cadeia proteica, com exceção dos códons de fim da tradução, que se encontram no gene apenas para delimitar o segmento do gene que está sendo tratado. Este processo é realizado de forma gradual, incluindo um aminoácido por vez e, quando terminado, a proteína se dissocia do mRNA e passa a desempenhar seu papel na célula.

Em grande parte dos genes eucarióticos, são observadas sequências não-codificantes, chamadas **íntrons**, interrompendo as sequências codificantes, os **éxons**. Assim, uma

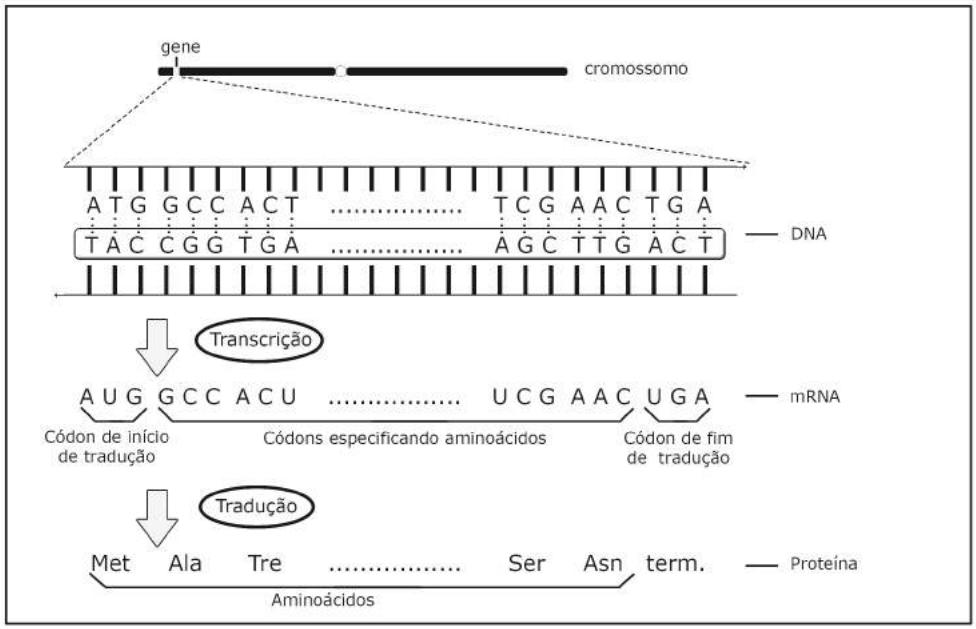

Figura 2.3: O processo de síntese de proteínas a partir do DNA

das alterações a que o RNA transcrito pode ser submetido antes de se transformar em mRNA é a remoção dos íntrons nele encontrados.

As sequências de DNA estão ainda sujeitas a modificações que afetam sua estrutura, resultando em sequências distintas das originais. Diversos fatores podem resultar na alteração do DNA, sejam eles externos, como exposição à radioatividade ou a certas substâncias químicas, ou internos, como replicações imperfeitas. Quando alguma modificação ocorre na sequência de DNA, é dito que houve uma *mutação*. De maneira geral, as mutações ocorridas são divididas em:

- *substituição*: um dado nucleotídeo é substituído por outro (`mismatch`);
- *inserção*: adição de um ou mais nucleotídeos (`space`); e
- *remoção*: deleção de um ou mais nucleotídeos (`space`).

Dependendo da região em que as modificações ocorrem, elas podem passar despercebidas ou até inutilizarem uma sequência genética. Mutações que ocorram em regiões não-codificantes não apresentam grande risco ao funcionamento da célula, enquanto que mutações em regiões codificantes podem comprometê-lo seriamente. Assim, o **princípio da conservação das bases** garante que a ocorrência de mutações em áreas não-codificantes sejam mais comuns, devido ao pequeno impacto na sobrevivência do organismo, do que em regiões codificantes, em que as mutações são mais prejudiciais.

Quando uma mutação ocorre dentro de um gene e, por algum motivo, beneficia o organismo ao qual pertence, ela tende a ser propagada aos seus descendentes. Assim, as mutações são capazes de "criar"genes novos a partir de um único gene. Genes que possuem um ancestral em comum são chamados de genes **homólogos**.

2.8.1 O Problema da Identificação de Genes

A compreensão dos conceitos biológicos discutidos anteriormente é essencial para se realizar nesta seção a definição do problema da identificação de genes, que será importante no decorrer deste trabalho. As informações sobre o problema foram retiradas de [3], onde maiores detalhes podem ser consultados.

Para que os dados de um novo DNA sequenciado possam ser interpretadas, é preciso identificar seus genes. O **problema da identificação de genes** consiste então em encontrar, em uma sequência de DNA, a posição de início e fim de seus éxons, as chamadas regiões codificantes, e consequentemente de seus íntrons, ou regiões não-codificantes. Essa tarefa não é trivial, e portanto vários métodos foram sugeridos para solucioná-la. Esses métodos dividem-se em três categorias principais de acordo com a perspectiva sob a qual abordam o problema:

1. **Baseados em conteúdo:** consideram as propriedades gerais da sequência, como frequência de códons, periodicidade de repetições e complexidade da composição da sequência, para realizar as determinações dos genes.

2. **Baseados em sinais**: buscam pela presença ou ausência de uma sequência, padrão ou consenso específicos, que ajudam, por exemplo, a identificar códons de início e fim de genes.

3. **Baseados em comparação:** consiste em determinar os genes de uma nova sequência através da comparação com sequências homólogas cujos genes já são conhecidos.

O presente estudo considera os métodos da última categoria, que segue uma abordagem combinatória, em que são realizadas buscas por correspondências entre segmentos da sequência alvo e as regiões codificantes de sequências relacionadas e previamente anotadas, na tentativa de encontrar semelhanças que permitam definir quais segmentos candidatos são de fato os éxons que compõem o gene sendo buscado.

Capítulo 3

Definição do Problema

Feitas as definições necessárias para compreensão do problema em estudo, é possível agora formalizá-lo. Porém, para um melhor entendimento do Problema do Alinhamento *Spliced* Múltiplo, que é objeto principal de estudo deste trabalho, é importante que o leitor seja antes apresentado ao problema do qual ele foi derivado, assim como à solução proposta para ele. Dessa forma, o presente capítulo inicia-se com a definição do Problema do Alinhamento *Spliced*, descrevendo em seguida uma solução para ele. Ao final, é exposto o Problema do Alinhamento Spliced Múltiplo, tanto na versão em que a similaridade é utilizada como medida de comparação entre sequências como naquela que emprega a Distância de Levenshtein.

3.1 O Problema do Alinhamento *Spliced*

Proposto por Gelfand *et al.* em 1996 [15], o Problema do Alinhamento *Spliced* modela o problema da identificação de genes [31] através da comparação da sequência alvo com uma sequência de cDNA.

De maneira informal, no Problema do Alinhamento *Spliced* deseja-se encontrar um alinhamento entre duas sequências que represente de forma adequada como uma delas é constituída de partes distintas e não sobrepostas da outra. Formalmente, o Problema do Alinhamento *Spliced* pode ser definido como segue [35]:

> **Problema do Alinhamento *Spliced* (PAS)**: Dadas duas sequências g e t, um conjunto ordenado $\mathcal{B}$ de segmentos de g e uma função de pontuação w, deseja-se encontrar uma cadeia de segmentos Γ de $\mathcal{B}$ tal que $sim_w(\overline{\Gamma}, t)$ é máxima.

As sequências g e t são conhecidas como sequências **base** e **modelo**, respectivamente, enquanto que os segmentos do conjunto $\mathcal{B}$ são chamados de **blocos**. Portanto, esse é um problema de otimização combinatória em que se quer encontrar uma combinação Γ de blocos de $\mathcal{B}$, respeitando-se a ordem e a restrição de não sobreposição, que ao serem concatenados geram uma sequência $\overline{\Gamma}$ mais similar possível a t. Uma representação da entrada e saída do PAS pode ser encontrada na Figura 3.1.

Ainda em [15], Gelfand *et al.* apresentam uma solução baseada em programação dinâmica para o Problema do Alinhamento *Spliced* por eles introduzido. Essa solução

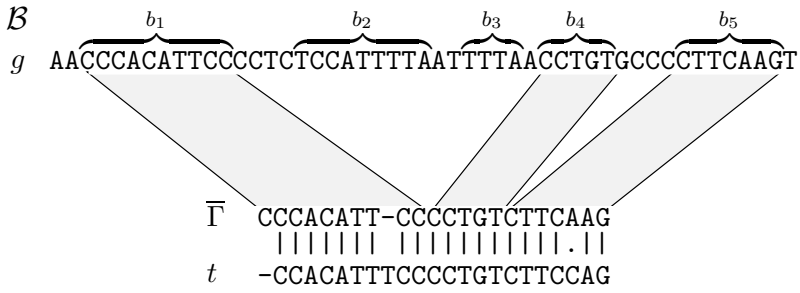

Figura 3.1: Um exemplo de instância do PAS com g, correspondendo à sequência base e t à sequência modelo, $\mathcal{B} = \{b_1, b_2, b_3, b_4, b_5\}$ ao conjunto de blocos e $\overline{\Gamma}$ à sequência gerada pela concatenação dos blocos pertencentes à solução encontrada neste caso (b_1, b_4 e b_5). Um alinhamento ótimo entre $\overline{\Gamma}$ e t também pode ser visto no exemplo (figura adaptada de Kishi e Adi [24, 25]).

utiliza-se dos conceitos a seguir. Seja b_k um segmento de g caracterizado por $g[p..q]$ que inclui uma posição i da sequência g, para $p \leq i \leq q$. Seja ainda $\Gamma = \{b_1, ..., b_k, ..., b_t\}$ uma cadeia de segmentos de g incluindo b_k, $\overline{\Gamma_i}$ a sequência formada pela concatenação de b_1, b_2, ..., $b_k[p..i]$, ou seja, $\overline{\Gamma_i} = \overline{b_1 b_2 ... b_k[p..i]}$ e Γ^k um conjunto que reúne todas as cadeias de segmentos de $\mathcal{B}$ contendo o bloco b_k. A ideia da solução é bastante semelhante ao cálculo usual da similaridade entre duas sequências, porém agora considerando uma terceira dimensão, indexada pelos blocos.

A solução proposta por Gelfand *et al.* para o PAS consiste em preencher uma matriz S de 3 dimensões em que cada posição $S[i][j][k]$ armazena o valor da solução do subproblema (do problema original) que envolve a sequência t até a sua posição j, os blocos anteriores a b_k e que inclui necessariamente o bloco b_k até a posição i de g. De modo mais formal:

$$S[i][j][k] = max_{\Gamma^k} sim_\omega(\overline{\Gamma_i}, t[1..j]),$$

A matriz S pode ser preenchida de acordo com a Recorrência 3.1 a seguir:

$$S[i][j][k] = max \begin{cases} S[i-1][j-1][k] + \omega(g[i], t[j]), & \text{se } i \neq primeiro(b_k) (a) \\ S[i-1][j][k] + \omega(g[i], \text{`}-\text{'}), & \text{se } i \neq primeiro(b_k) (b) \\ max_{l \in B_i} S[último(l)][j-1][l] + \omega(g[i], t[j]), & \text{se } i = primeiro(b_k) (c) \\ max_{l \in B_i} S[último(l)][j][l] + \omega(g[i], \text{`}-\text{'}), & \text{se } i = primeiro(b_k) (d) \\ S[i][j-1][k] + \omega(\text{`}-\text{'}, t[j]) & (e) \end{cases} \quad (3.1)$$

em que $1 \leq i \leq |g|$, $1 \leq j \leq |t|$, $1 \leq k \leq |B|$, ω é uma função de pontuação previamente definida e $B_i = \{l : último(b_l) < i\}$. Aqui, novamente considera-se uma função de pontuação ω que trata as três possíveis combinações de caracteres das sequências $\overline{\Gamma}$ e t: *match*, *mismatch* e *space*. Porém, deve-se atentar para o fato de que o alinhamento do primeiro caractere de um bloco b_i com a sequência t deve ser tratado de forma diferente do alinhamento dos caracteres restantes dela com t. Mais especificamente, ao alinhar o primeiro caractere do bloco b_k, considera-se que ele está sendo incluído na solução e portanto é preciso levar em conta a melhor solução possível calculada até o

momento com os blocos $b_1, b_2, b_3, ..., b_{k-1}$. Já para os demais caracteres de b_k, o processo se reduz ao alinhamento simples de duas sequências, deixando-se o conceito de blocos em segundo plano.

Para um melhor entendimento da Recorrência 3.1, considere inicialmente os três casos válidos quando $i \neq primeiro(b_k)$, que contempla o alinhamento do caractere que não é o primeiro de seu bloco. Como dito anteriormente, essa situação é idêntica à do alinhamento simples entre duas sequências. Nesse caso, $S[i][j][k]$ irá assumir o maior dentre os 3 valores possíveis especificados abaixo:

- alinhar o i-ésimo caractere de g com o j-ésimo caractere de t: aproveita-se o valor da solução do subproblema que envolve os blocos $b_1...b_{k-1}$, a sequência t até a posição $j-1$ e que inclui o bloco b_k até a posição $i-1$ de g, adicionando-se a pontuação adequada ($\omega(g[i], t[j])$);
- inserir um espaço em t e alinhá-lo com o i-ésimo caractere de g: aproveita-se o valor da solução do subproblema que envolve os blocos $b_1...b_{k-1}$, a sequência t até a posição j e que inclui o bloco b_k até a posição $i-1$ de g, somando-se a pontuação adequada ($\omega(g[i],\text{‘}-\text{’})$);
- alinhar um espaço com o j-ésimo caractere de t: aproveita-se o valor da solução do subproblema que envolve os blocos $b_1...b_{k-1}$, a sequência t até a posição $j-1$ e que inclui o bloco b_k até a posição i de g, acrescentando-se a pontuação adequada ($\omega(\text{‘}-\text{’}, t[j])$).

Essas possibilidades descritas referem-se aos itens $(a), (b)$ e (e) da Recorrência 3.1, respectivamente, e encontram-se representadas na Figura 3.2:

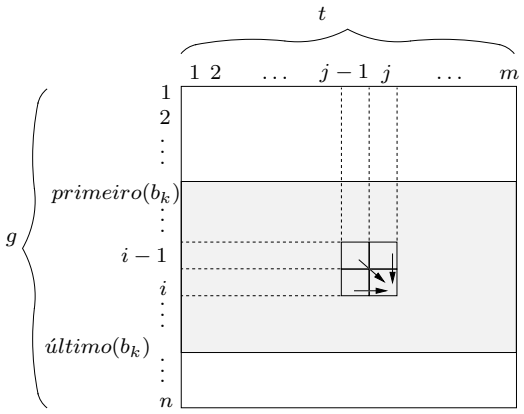

Figura 3.2: Matriz bidimensional representando o bloco b_k em S, necessária para se preencher a posição $S[i][j][k]$ quando $i \neq primeiro(b_k)$. As setas indicam as posições anteriormente preenchidas que serão consultadas para o cálculo do valor de $S[i][j][k]$ e a área sombreada esboça a região utilizada em S para representar o bloco b_k (figura adaptada de Kishi e Adi [24, 25]).

Agora, os casos em que $g[i]$ é o primeiro caractere de um bloco b_k precisam ser tratados de maneira distinta para garantir a correção da recorrência. Quando $i = primeiro(b_k)$ tem-se que b_k está sendo considerado pela primeira vez para constituir a solução sendo

calculada. Assim, é preciso verificar qual a melhor solução parcial encontrada até o momento, considerando apenas os blocos anteriores e não sobrepostos a b_k, e avaliar o custo da inclusão de b_k nela, através do alinhamento do primeiro caractere de b_k com a sequência t. As três possibilidades de alinhamento são ilustradas na Figura 3.3 e descritas a seguir, das quais dois casos exigem maior cuidado (considere $1 \leq l \leq k-1$):

- Alinhar o i-ésimo caractere de g com o j-ésimo de t: faz-se necessário verificar, para cada bloco anterior a b_k, qual o valor da melhor solução parcial alcançada que inclui esse bloco, e considerando-se t apenas até a sua posição $j-1$. Isso é feito varrendo-se as posições $S[último(l)][j-1][l]$ para todo $l \in B_i$, selecionando-se o maior valor encontrado e o acrescendo da pontuação adequada $\omega(g[i], t[j])$ (Caso (c) da Recorrência 3.1);

- inserir um espaço em t e alinhá-lo com $g[i]$: uma verificação semelhante à descrita no item anterior deve ser executada, porém agora considerando t até a posição j. Logo, os elementos a serem analisados à procura do maior valor são $S[último(l)][j][l]$ para todo $l \in B_i$. Uma vez encontrado, soma-se ao maior valor a pontuação adequada $\omega(g[i],\text{`}-\text{'})$ (Caso (d) da Recorrência 3.1);

- incluir um espaço em g: esse caso não precisa de tratamento especial, podendo ser calculado igualmente ao caso em que $g[i] \neq primeiro(b_k)$. Isso porque inserir um espaço em g logo no início do bloco b_k é o mesmo que (ainda) não incluí-lo na solução.

Para se obter a solução procurada, ou seja, os blocos que fazem parte do alinhamento *spliced* ótimo, o procedimento é similar ao usado no alinhamento ótimo entre duas sequências. A posição $max\{S[último(k)][|t|][k]\}$ apresenta o valor da similaridade do alinhamento *spliced* ótimo, de forma que retornar à posição inicial partindo desta, respeitando as decisões tomadas, fornece a solução desejada. Para isso, basta realizar os passos da recorrência de forma inversa, adicionando o bloco b_k à solução Γ sempre que o índice k se alterar. Refeitos todos os passos, o conjunto Γ conterá os blocos que fazem parte da solução ótima do problema. Os passos descritos acima constituem um algoritmo para o PAS que será referenciado nesse texto como *algoritmo de Gelfand*.

Para analisar a complexidade da solução descrita, assuma que $n = |g|$, $m = |t|$, $p = |\mathcal{B}|$ e observe que a posição $S[i][j][k]$ é preenchida apenas para $i \in b_k$, sendo que as demais posições nada representam, sendo portanto ignoradas pela recorrência. Dessa forma, um algoritmo baseado na Recorrência 3.1 precisará preencher $m\sum_{k=1}^{p}|b_k|$ posições. Denotando a cobertura da sequência g pelos blocos em $\mathcal{B}$ por $c = \frac{1}{n}\sum_{k=1}^{p}|b_k|$, a complexidade de tempo da implementação da Recorrência 3.1 é $\mathcal{O}(mnc + mp^2)$ e de espaço é $\mathcal{O}(mnc)$.

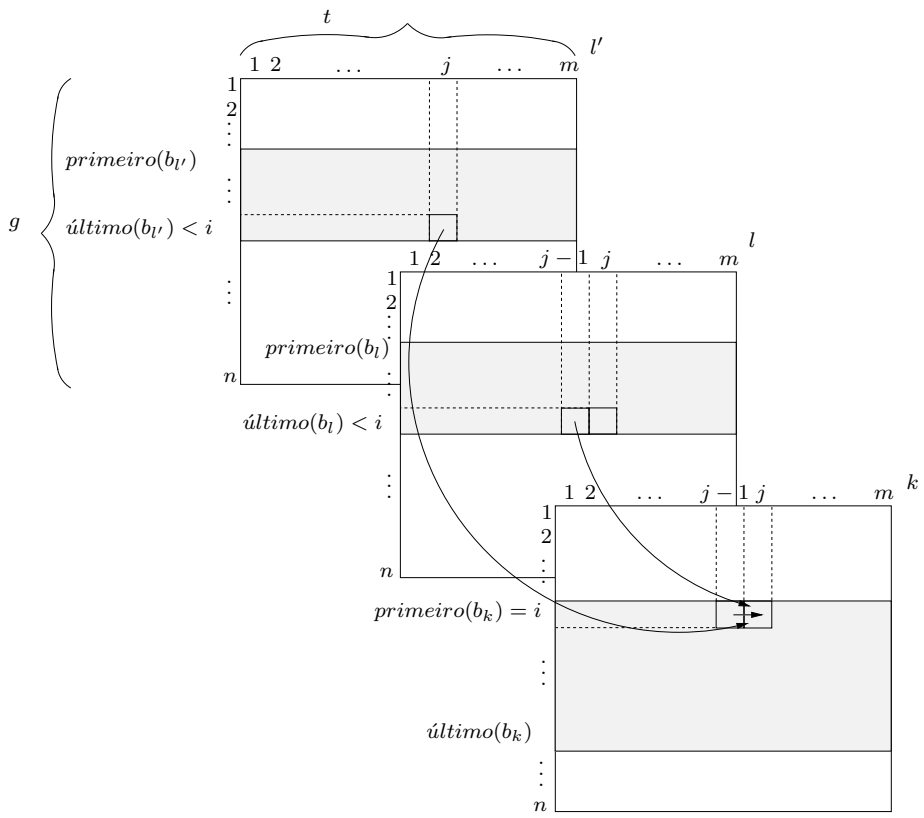

Figura 3.3: Matrizes bidimensionais representando os blocos $b_{l'}, b_l$ e b_k em S, necessárias para preenchimento da posição $S[i][j][k]$ quando $i = primeiro(b_k)$. As setas indicam as posições anteriormente preenchidas que serão consultadas para o cálculo do valor de $S[i][j][k]$ e as áreas sombreadas esboçam as regiões utilizadas em S para representar os blocos $b_{l'}, b_l$ e b_k (figura adaptada de Kishi e Adi [24, 25]).

3.2 O Problema do Alinhamento *Spliced* Múltiplo

Como visto na seção anterior, o Problema do Alinhamento *Spliced* possui relação direta com o problema da identificação de genes, e uma solução para ele é conhecida e gera bons resultados na prática. Nesse contexto, uma suposição razoável é a de que se pode melhorar os resultados das identificações de genes se mais sequências de cDNA forem empregadas na comparação. Isso porque mais informações e evidências acerca do gene procurado estariam disponíveis, permitindo uma identificação mais precisa dos seus éxons do que a alcançada pela versão simples do problema proposta por Gelfand *et al.* em [15]. Essa variante do Problema de Alinhamento *Spliced*, conhecida como Problema do Alinhamento *Spliced* Múltiplo, corresponde ao principal problema abordado neste trabalho de mestrado. Sua formalização matemática é a seguinte:

Problema do Alinhamento *Spliced* Múltiplo (PASM): Dados uma sequência g, um conjunto de sequências $\mathcal{T} = \{t_1, t_2, ..., t_u\}$, um conjunto ordenado $\mathcal{B}$ de segmentos de g e uma função de pontuação w, encontrar uma cadeia de segmentos Γ de $\mathcal{B}$ tal que $\sum_{i=1}^{u} sim_\omega(t_i, \overline{\Gamma})$ seja a maior possível.

Observe que essa variante do PAS considera que a escolha dos melhores blocos não será mais feita com base apenas em uma sequência t, mas sim com base em várias sequências. Assim, pretende-se encontrar uma sequência formada pela concatenação de blocos ordenados e não sobrepostos de g cuja similaridade total em relação às sequências de $\mathcal{T}$ seja a maior possível, o que torna essa variante mais complexa do que o problema original. Um exemplo de instância do PASM e uma solução para ela podem ser conferidos na Figura 3.4.

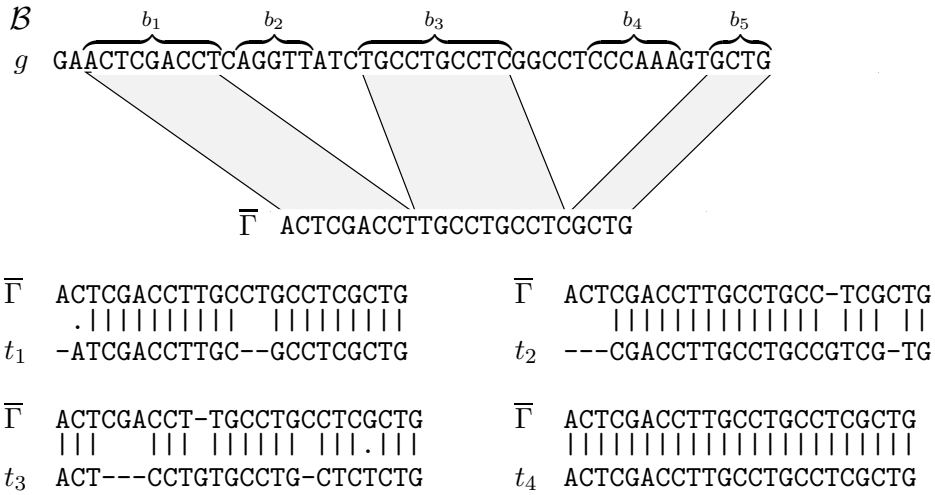

Figura 3.4: Um exemplo de instância do PASM com g correspondendo à sequência base, $\mathcal{T} = \{t_1, t_2, t_3, t_4\}$ ao conjunto de sequências modelos, $\mathcal{B} = \{b_1, b_2, b_3, b_4, b_5\}$ ao conjunto de blocos e $\overline{\Gamma}$ à sequência gerada pela concatenação dos blocos pertencentes à solução encontrada neste caso (b_1, b_3 e b_5). Um alinhamento ótimo entre $\overline{\Gamma}$ e cada uma das sequências em $\mathcal{T}$ também pode ser visto no exemplo (figura adaptada de Kishi e Adi [24, 25]).

Em [24, 25], Kishi e Adi provaram que o PASM é NP-completo usando uma função de pontuação específica e apresentaram quatro heurísticas para ele. Assim, dar continuidade ao trabalho iniciado em [24, 25] é o principal objetivo deste estudo, motivado pelo fato de que o PASM é um problema ainda pouco explorado na literatura, fornecendo inúmeras possibilidades de aprofundamento e novas propostas a serem avaliadas. Mais especificamente, objetiva-se aqui abordar o problema sob diferentes perspectivas. Em especial, serão apresentadas neste trabalho uma aproximação e uma formulação de programação linear inteira para o PASM, completando assim as estratégias mais comumente utilizadas na abordagem de problemas NP-completos: heurísticas, aproximações e PLI.

Inaproximabilidade do PASM empregando similaridade

Considerando que um dos objetivos deste trabalho é o desenvolvimento de um algoritmo de aproximação para o PASM, e considerando ainda que a definição do algoritmo de aproximação aplica-se somente a problemas de otimização tais que $\texttt{val}(S_A) \geq 0$, será feito uso daqui para frente de uma versão do PASM que utiliza distância ao invés de similaridade como medida de comparação de sequências. Mais especificamente, essa

versão do PASM que será estudada se utiliza da Distância de Levenshtein como medida de comparação. Essa versão do problema pode ser assim definida:

Problema do Alinhamento *Spliced* Múltiplo - Levenshtein (PASM): Dados uma sequência g, um conjunto de sequências $\mathcal{T} = \{t_1, t_2, ..., t_u\}$, um conjunto ordenado $\mathcal{B}$ de segmentos de g e a Distância de Levenshtein d_L, encontrar uma cadeia de segmentos Γ de $\mathcal{B}$ tal que $\sum_{i=1}^{u} d_L(t_i, \overline{\Gamma})$ seja a menor possível.

É importante observar que, mesmo nessa versão de minimização, o PASM continua sendo NP-completo. Isso é verdade porque a função de pontuação utilizada por Kishi e Adi em [24, 25] para a prova da NP-completude do PASM (original) é equivalente àquela utilizada pela Distância de Levenshtein. Essa função de pontuação atribui 0 para `matches` e -1 para `spaces` e `mismatches`, de forma que basta multiplicar o valor da similaridade encontrado através dela por -1 para se obter a Distância de Levenshtein entre duas sequências.

Capítulo 4

Problemas Relacionados

Sugerir um algoritmo de aproximação e/ou um modelo de PLI para um problema de otimização nem sempre é um processo trivial. É comum, antes que algo concreto possa ser proposto, dispender um pouco de tempo para analisar o problema que está sendo investigado. Essa análise envolve, dentre outras coisas, o estudo de problemas semelhantes ao problema em foco. Dessa forma, com a intenção de se estudar problemas relacionados ao Problema do Alinhamento Spliced Múltiplo e reunir, dentre as estratégias empregadas para tratar cada um deles, inspirações para o desenvolvimento de novas abordagens para o PASM, foi realizada uma revisão bibliográfica acerca de problemas envolvendo sequências que se assemelham ao PASM. Neste capítulo são detalhados os principais trabalhos analisados e indicadas as suas contribuições para a evolução do estudo aqui proposto.

4.1 Revisão de literatura

Para essa revisão de literatura, foram considerados como relevantes estudos envolvendo aproximações e formulações de programação linear inteira para diferentes problemas relacionados a sequências, visando reunir estratégias e evidenciar características interessantes dos problemas a serem aplicadas no tratamento do PASM. Os trabalhos que se destacaram são aprofundados nesta seção.

Dentre os trabalhos relevantes para o desenvolvimento desse estudo, destaca-se o trabalho de Chen *et al.* detalhado em [9]. Nesse trabalho são propostas aproximações para dois problemas envolvendo sequências: o Problema da Sequência Mediana e o Problema da Sequência Central. Higuera e Casacuberta mostraram em [13] que ambos os problemas são NP-difíceis. A seguir podem ser encontradas as definições formais de cada problema, assim como um esboço das aproximações sugeridas para eles.

Problema da Sequência Mediana (PSM): dado um conjunto de n sequências $W = \{w_1, w_2, ..., w_n\}$, construídas sobre um alfabeto Σ, deseja-se encontrar uma sequência w^* em Σ^*, denominada sequência mediana, que minimize o valor de $\sum_{i=1}^{n} d_L(w_i, w^*)$.

Em [9], é proposta uma $(2 - \frac{2}{n})$-aproximação para o PSM. A ideia do algoritmo consiste

em reduzir o espaço de busca da sequência w^*. Agora, ao invés de se considerar qualquer sequência de Σ^* como candidata a w^*, apenas as n sequências contidas no conjunto W serão candidatas válidas. Mais especificamente, o algoritmo consiste em, dado o conjunto $W = \{w_1, w_2, ..., w_n\}$, selecionar uma sequência $w' \in W$ tal que o valor de $\sum_{i=1}^{n} d_L(w', w_i)$ seja mínimo. Essa sequência pode ser encontrada em tempo $\mathcal{O}(n^2 * |u|^2)$, sendo $|u|$ o comprimento da maior sequência em W e portanto o algoritmo proposto é polinomial.

A prova de que o valor da solução devolvida pelo algoritmo é limitado superiormente por $2 - \frac{2}{n}$ consiste, basicamente, em utilizar a sequência mediana ótima w^* como sequência intermediária para transformar w_i em w_j, com $1 \leq i, j \leq n$. Isso permite que a propriedade da desigualdade triangular seja aplicada, para mostrar que $n * \sum_{i=1}^{n} d_L(w', w_i) \leq \sum_{j=1}^{n} \sum_{i=1}^{n} d_L(w_j, w_i) \leq 2 * n * \sum_{i=1}^{n} d_L(w^*, w_i)$. Para tornar a aproximação mais justa, deve-se observar que a distância entre uma sequência w_i e ela mesma é 0, $d_L(w_i, w_i) = 0, \forall i$, ao passo que se for utilizada uma sequência intermediária w^* este valor poderá ser maior do que 0, $d_L(w_i, w^*) + d_L(w^*, w_i) \geq 0$, gerando uma distância superior à real. Sendo assim, o valor de $d_L(w_i, w^*) + d_L(w^*, w_i), \forall i$, não deve ser contabilizado. Ao fazer este ajuste, chega-se então a $\sum_{i=1}^{n} d_L(w', w_i) \leq (2 - \frac{2}{n}) \sum_{i=1}^{n} d_L(w_i, w^*)$. Maiores detalhes dessa prova podem ser conferidos em [9].

O Problema da Sequência Central é o outro problema abordado em [9]. Formalmente, ele é descrito como segue:

> **Problema da Sequência Central (PSC)**: dado um conjunto de n sequências $W = \{w_1, w_2, ..., w_n\}$, construídas sobre um alfabeto Σ, encontrar uma sequência w' construída sobre Σ^* que minimize o valor de $max(d_L(w_i, w'))$, com $1 \leq i \leq n$.

O algoritmo de aproximação sugerido para o PSC novamente baseia-se na redução do espaço de busca da solução, sendo w' selecionada *aleatoriamente* apenas dentre as n sequências de W. Também pela desigualdade triangular, prova-se em [9] que esta é uma 2-aproximação para o problema, em que a razão 2 é justa.

Os agoritmos sugeridos por [9] são inspirações promissoras para uma aproximação do PASM, uma vez que apresentam características semelhantes ao problema. A ideia de gerar a sequência mediana a partir do alfabeto em questão pode ser aproveitada trocando o alfabeto pelos blocos $\mathcal{B}$ disponíveis. Além disso, a prova de que são 2-aproximações é relativamente trivial, o que motiva ainda mais o emprego dessa linha de raciocício.

Outro problema NP-difícil semelhante ao PASM é o Problema da Maior Subsequência Comum (do inglês *longest common subsequence*, LCS), proposto inicialmente por Hirschberg em [18]. Ele pode ser definido como segue:

> **Problema da Maior Subsequência Comum (LCS)**: dado um conjunto finito S de sequências construídas sobre Σ, encontrar uma sequência l de maior tamanho possível que seja uma subsequência de cada uma das sequências de S.

Esse problema é amplamente difundido e estudado na área de teoria da computação e vários dos trabalhos tomados como base durante o desenvolvimento deste estudo abordam o LCS. Dentre eles, dois se destacam e são detalhados a seguir.

Jiang e Li discorrem em [19] sobre a não-aproximabilidade do LCS com boas razões de aproximação, ou seja, provam através de uma redução do Problema da Maior Clique que, se existir uma constante $\delta > 0$ tal que o LCS apresente um algoritmo de aproximação de tempo polinomial com razão de aproximação $1/n^{\delta}$, com n sendo $|S|$, então $P = NP$.

Ainda em [19], os autores discorrem sobre um algortimo chamado *Long Run* para o problema da maior subsequência comum, que para qualquer conjunto S de sequências sobre Σ, retorna uma solução de valor pelo menos $opt(S)/|\Sigma|$. Esse algoritmo consiste em encontrar o maior m para o qual a^m seja uma subsequência comum a todas as sequências de S, considerando algum $a \in \Sigma$. Por ser composta por apenas um caractere de Σ, a solução fornecida aplicando-se o *Long Run* agrega pouco significado prático a esse trabalho, pois limita a aplicação do algoritmo.

Em [5], Bonizzoni *et al.* propõem um novo algoritmo de aproximação para o LCS, chamado de *Expansion*. Sua abordagem é semelhante à do *Long Run*, porém ao invés de selecionar apenas um caractere de Σ para constituir a solução, esse algoritmo tenta expandir a estratégia de forma a incluir outros símbolos do alfabeto. Assim como o *Long Run*, o algoritmo *Expansion* possui razão de aproximação $1/|\Sigma|$, embora este segundo tenha se mostrado mais eficiente na prática que o primeiro, uma vez que a razão indicada não parece ser justa para ele.

Um dos artigos mais completos e interessantes encontrado a respeito de problemas envolvendo sequências foi o de Lanctot *et al.*[29], em que é feita uma coletânea das principais variantes do Problema da Sequência mais Distante (do inglês *Farthest String Problem - FSP*) e do Problema do Segmento mais Próximo (do inglês *Closest Substring Problem - CSSP*). No contexto desse estudo, apenas o CSSP e suas variantes - Problema da Sequência mais Próxima (do inglês *Closest String Problem - CSP*) e Problema da Sequência mais Próxima da Maioria (do inglês *Close to Most String Problem - CMSP*)- são relevantes, e por isso são definidas e comentadas a seguir.

Considere o valor da Distância de Hamming entre duas sequências x e y de mesmo comprimento como sendo a quantidade de posições dessas sequências que possuem caracteres distintos, ou seja, quantas substituições devem ser feitas em x para transformá-la em y, ou vice-versa. A Distância de Hamming será denotada por $d_H(x, y)$.

Problema da Sequência mais Próxima (CSP): dado um conjunto S de sequências de comprimento n, construídas sobre um alfabeto Σ, encontrar uma sequência x de comprimento n sobre Σ que minimize $d_c = \max d_H(x, s)$ tal que, para toda sequência $s \in S$, $d_H(x, s) \leq d_c$.

Problema do Segmento mais Próximo (CSSP): dado um conjunto S de sequências de comprimento pelo menos n, construídas sobre um alfabeto Σ, encontrar uma sequência x de comprimento n sobre Σ que minimize $d_c = \max d_H(x, y)$ tal que, para toda sequência $s \in S$, $d_H(x, y) \leq d_c$ seja válido para alguma subsequência y de s, com y apresentando comprimento igual a n.

Problema da Sequência mais Próxima da Maioria (CMSP): dados um conjunto S de sequências de comprimento n, construídas sobre um alfabeto Σ e um limiar

$k > 0$, encontrar uma sequência x de comprimento n sobre Σ que maximize a quantidade de sequências $s \in S$ que satisfazem a restrição $d_H(x, s) \leq k$.

Primeiramente Lanctot *et al.* provam que o Problema da Sequência mais Distante é NP-difícil, e através de uma redução do FSP ao Problema da Sequência mais Próxima, mostram que este também é NP-difícil. Como o CSP é um caso especial do CSSP, o último apresenta a mesma complexidade.

Em [29], foi demonstrada a existência de uma 2-aproximação que se aplica tanto ao CSP quanto ao CSSP. Para isso, considera-se uma solução ótima x para a qual exista uma subsequência p_i de s_i tal que $d(x, p_i) \leq k$ para toda sequência $s_i \in S$, e deseja-se encontrar uma subsequência p_1 de s_1 de forma que a desigualdade triangular assegure que $d(p_1, p_j) \leq 2k$, com p_j sendo uma subsequência de $s_j \in S$. O algoritmo então consiste em um caso especial do alinhamento estrela [17], em que p_1 pode ser encontrada testando-se todas as subsequências de comprimento n em s_1.

Para o CSP, foi proposta ainda uma aproximação mais justa, com razão $\frac{4}{3}(1 + \epsilon)$, cujos detalhes não serão explorados aqui por não agregarem valor ao relacionamento do CSP com o PASM. Já o CMSP é apenas mencionado no trabalho, que não apresenta contribuições para o problema.

Reunidas informações diversificadas sobre aproximações para problemas envolvendo sequências, partiu-se para o campo da programação linear inteira. Nesse contexto, Tjandraatmadja e Ferreira abordam em [42] uma variante do LCS, com a restrição de não poder haver caracteres repetidos na subsequência comum de maior tamanho sendo buscada. Essa variante do LCS denomina-se Problema da Maior Subsequência Comum sem Repetições (do inglês *Repetition-Free Longest Common Subsequence - RFLCS*):

Problema da Maior Subsequência Comum sem Repetições (RFLCS): dadas duas sequências s e t construídas sobre um alfabeto Σ, encontrar uma maior sequência sem repetições l que seja uma subsequência de ambas as sequências s e t.

Através de uma L-redução do problema APX-completo MAX 2,3-SAT para o RFLCS, Adi *et al.* demonstraram em [1] que o RFLCS é APX-difícil. Dessa forma, Tjandraatmadja e Ferreira propõem em seu estudo duas formulações de programação linear inteira para o LCS, chamadas de *formulação por casamentos* e *formulação por estrelas*. Ambas são sucintas e possuem apenas uma variável e uma restrição, além da função objetivo. A primeira considera apenas casamentos que não se cruzam dois a dois, enquanto a segunda é baseada no conceito de cliques em grafos e admite conjuntos de casamentos sem cruzamentos, sendo que um *casamento* é definido como um par ordenado de índices (i, j) para o qual $s[i] = t[j]$, isto é, o $i-$ésimo símbolo da sequência s é igual ao $j-$ésimo símbolo da sequência t. Diz-se que dois casamentos (i, j) e (k, l) se cruzam se $i \geq k$ e $j \leq l$, ou $i \leq k$ e $j \geq l$.

Para o RFLCS, as duas formulações descritas são estendidas, com a adição de uma restrição referente à repetição de caracteres. Outras duas formulações são ainda propostas: a *formulação por conflitos*, que busca casamentos que não apresentem conflitos dois a dois, ou seja, que não se cruzem e nem sejam do mesmo símbolo; e a formulação

por estrelas estendidas, que retoma o conceito de clique e conflitos, mas considerando conjuntos de casamentos ao invés de pares.

Tjandraatmadja e Ferreira também propõem em [42] duas aproximações para o RFLCS. Uma delas consiste em computar uma LCS e em seguida remover as ocorrências de símbolos repetidos. O segundo algoritmo proposto faz exatamente o inverso, removendo primeiramente as repetições das sequências s e t e só após encontrando a LCS. Prova-se que ambos os algoritmos são uma $occ_{max}-$aproximação, em que occ_{max} indica o maior valor de $occ(\sigma)$ para todos os $\sigma \in \Sigma$, sendo $occ(\sigma)$ o número de ocorrências de σ nas sequências s ou t, o que for menor. Por fim, três heurísticas gulosas também são introduzidas para o RFLCS em [42].

O que mais chama a atenção no estudo de Tjandraatmadja e Ferreira é a abordagem adotada, semelhante à empregada aqui em relação ao PASM: são sugeridas formulações de programação linear, algoritmos de aproximação e heurísticas, além de algoritmos aplicando outras técnicas, como *branch-and-cut* e enumeração. No caso, o presente trabalho traz um algoritmo de aproximação e uma formulação de PLI para o PASM, complementando as heurísticas propostas em [24, 25] e descritas em maiores detalhes a seguir.

A primeira das quatro heurísticas introduzidas por Kishi e Adi para o PASM é a Heurística da Sequência Central (H1). Inicialmente, ela calcula a similaridade entre cada par de sequências em $\mathcal{T}$ para determinar a sequência central do conjunto, ou seja, a sequência mais parecida com todas as outras. Essa sequência, juntamente com $\mathcal{B}$ e g, forma uma instância do PAS que é então submetida ao algoritmo de alinhamento *spliced* de Gelfand *et al.*. A solução devolvida para essa instância do PAS é assumida como uma solução para o PASM pela heurística.

A segunda heurística é a Heurística da Sequência Consenso (H2), que utiliza o algoritmo Estrela de Gusfield [16] para gerar um alinhamento múltiplo das sequências de $\mathcal{T}$, do qual é extraída uma sequência consenso. Novamente, esta sequência consenso em conjunto com g e $\mathcal{B}$ representam uma instância do PAS, que é utilizada como entrada para o algoritmo de alinhamento *spliced* e tem sua saída adotada como solução para o PASM.

A heurística seguinte, dita Heurística do Consenso de Blocos (H3), sugere que o algoritmo de alinhamento *spliced* de Gelfand *et al.* seja executado u vezes, cada uma delas considerando uma das diferentes sequências de $\mathcal{T}$. Concluído esse processamento, blocos que apareçam em mais do que metade das soluções devolvidas são incluídos na solução final para o PASM, de forma que blocos sobrepostos nunca farão parte de uma mesma solução.

As três primeiras heurísticas descritas acima são determinísticas, ou seja, para uma dada entrada a saída gerada é sempre a mesma. A última heurística é a única de caráter "aleatório", desenvolvida com base no conceito de algoritmos genéticos. Dessa forma, é chamada de Heurística do Algoritmo Genético (H4). Nela, cadeias de $\mathcal{B}$, representando possíveis soluções para o PASM, são tomadas como a população, sendo que os indivíduos da população inicial são selecionados de maneira aleatória. Em seguida, os indivíduos são ranqueados de acordo com o somatório das similaridades entre a concatenação de seus blocos e as sequências em $\mathcal{T}$. A população é então atualizada, mantendo os melhores indivíduos sem modificações, aprimorando os medianos e

descartando os piores, que são substituídos por novos indivíduos gerados. Parâmetros pré-estabelecidos indicam quantas vezes esse processo deve ser realizado para aperfeiçoar os indivíduos a cada iteração. Ao final de todas as iterações, o indivíduo que apresente a melhor pontuação é devolvido como resposta para o PASM.

As quatro heurísticas propostas em [24, 25] foram avaliadas com instâncias artificiais do PASM e com instâncias reais do problema da identificação de genes. No primeiro caso, a Heurística H1 obteve os melhores resultados, enquanto que na aplicação prática foi a Heurística H3 que se destacou. Esses resultados são utilizados para comparar o desempenho das heurística já introduzidas para o PASM com as novas abordagens para o problema aqui apresentadas.

Capítulo 5

Uma Aproximação para o PASM

A substituição da similaridade pela Distância de Levenshtein como medida de comparação de sequências na definição do PASM busca tornar viável o desenvolvimento de um algoritmo de aproximação para ele. Dessa forma, é introduzido neste capítulo um algoritmo de aproximação para o PASM que explora a propriedade da desigualdade triangular relacionada à Distância de Levenshtein como meio de garantir uma solução no máximo três vezes pior do que uma solução ótima para o problema. A prova de que o algoritmo proposto é uma 3-aproximação para o PASM também encontra-se a seguir.

5.1 O Algoritmo de aproximação

Embora as soluções propostas por Kishi e Adi em [24, 25] para o PASM tenham apresentado bons resultados, a natureza heurística delas não fornece nenhuma garantia em relação à qualidade dos resultados obtidos. É exatamente essa lacuna que se pretende cobrir com este trabalho através da elaboração de um algoritmo de aproximação para o problema.

A principal inspiração para o algoritmo de aproximação para o PASM aqui proposto originou-se do estudo de Chen *et al.*[9] sobre o Problema da Sequência Mediana. A ideia central do algoritmo de aproximação para o PSM, de escolher como mediana uma das sequências de $\mathcal{W}$, foi adaptada para o contexto do PASM. Além disso, aplicou-se também a propriedade da desigualdade triangular associada às métricas, como é o caso da distância de edição, na prova de que o algoritmo sugerido é uma 3-aproximação.

O algoritmo de aproximação sugerido para o PASM faz uso do algoritmo de Gelfand [15] para o PAS tendo como função de pontuação ω a Distância de Levenshtein. Essa aproximação corresponde portanto a uma extensão natural da solução para o PAS, a versão original do problema. Mais especificamente, esse algoritmo de aproximação consiste em encontrar u soluções para o PAS, sendo uma solução para cada instância $g, \mathcal{B}$ e t_i, com $1 \leq i \leq u$, e escolher como solução final para o PASM aquela mais similar a todas as sequências em $\mathcal{T}$, ou seja, aquela solução Γ para o PAS tal que o valor de $\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i)$ seja mínimo. O pseudo-código desse algoritmo é mostrado no Algoritmo 5.1 a seguir.

ALGORITMO 5.1 PASM-3-APROX

Entrada: Uma sequência g, um conjunto de sequências $\mathcal{T} = \{t_1, t_2, ..., t_u\}$ e um conjunto $\mathcal{B} = \{b_1, b_2, ..., b_k\}$ de segmentos ordenados de g.
Saída: Uma cadeia Γ de $\mathcal{B}$ tal que o valor de $\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq 3 * \texttt{opt}$.

1: $\Gamma \leftarrow \emptyset$;
2: $minimo \leftarrow +\infty$;
3: **para** $i \leftarrow 1$ **até** u **faça**
4: $\quad \Gamma_i \leftarrow Gelfand(g, t_i, \mathcal{B})$; //chamada ao algoritmo de Gelfand tendo g, t_i e $\mathcal{B}$ como entrada
5: $\quad soma \leftarrow 0$;
6: $\quad$ **para** $j \leftarrow 1$ **até** u **faça**
7: $\quad\quad soma \leftarrow soma + d_L(\overline{\Gamma_i}, t_j)$;
8: $\quad$ **se** $soma < minimo$ **então**
9: $\quad\quad minimo \leftarrow soma$;
10: $\quad\quad \Gamma \leftarrow \Gamma_i$;
11: **devolva** Γ;

5.1.1 Prova da razão de aproximação

Nesta seção, é demonstrado formalmente que o Algoritmo 5.1 é um algoritmo de aproximação para o PASM de razão 3, ou seja, o valor de uma solução devolvida por ele é no máximo três vezes maior do que o valor de uma solução ótima para o problema.

Para a demonstração da razão de aproximação do algoritmo, considere uma instância I qualquer do PASM. Seja então Γ^* uma solução ótima para I tal que $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i)$ seja a menor possível. Ou seja, $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) = \texttt{opt}(I)$. Seja ainda Γ_i um subconjunto ordenado de segmentos não sobrepostos de $\mathcal{B}$ tal que $d_L(\overline{\Gamma_i}, t_i)$ é mínima. O Algoritmo 5.1 obtém um Γ_i para $i = 1, 2, ..., u$, ao aplicar o algoritmo de Gelfand, baseado na Recorrência 3.1, sobre uma instância do PAS formada por g, t_i e $\mathcal{B}$. Como solução final Γ para a instância I do PASM, é devolvido um Γ_i que apresente o menor valor de $\sum_{j=1}^{u} d_L(\overline{\Gamma_i}, t_j)$.

Antes de demonstrar em detalhes com o Teorema 5.1.2 que a razão de aproximação do Algoritmo 5.1 é 3, faz-se necessário considerar o Lema 5.1.1, que será essencial para a sua prova. Ele mostra que $\sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i)$ é limitado superiormente por $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i)$.

Lema 5.1.1 $\sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i)$.

Prova: Tem-se, pela forma como Γ_i é determinada pelo Algoritmo 5.1, que $d_L(\overline{\Gamma_i}, t_i)$ é mínima, e portanto $d_L(\overline{\Gamma_i}, t_i) \leq d_L(\overline{\Gamma_i^*}, t_i)$ para todo i. Aplicando-se essa desigualdade para todos os t_i, tem-se que $\sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma_i^*}, t_i)$.

■

A relação entre a solução Γ devolvida pelo Algoritmo 5.1, cujo valor é $\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i)$ e Γ^*, que corresponde a uma solução ótima de valor $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) = \texttt{opt}(I)$ do PASM, encontra-se descrita a seguir.

Teorema 5.1.2 *O Algoritmo 5.1 é uma 3-aproximação para o PASM*

Prova: Devido aos Passos 6, 7, 8, 9 e 10 do Algoritmo 5.1, tem-se que:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma_j}, t_i), \forall j = 1, ..., u \tag{5.1}$$

Ao somar essas inequações, obtém-se:

$$u * \sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{j=1}^{u} \sum_{i=1}^{u} d_L(\overline{\Gamma_j}, t_i) \tag{5.2}$$

Como a Distância de Levenshtein satisfaz a desigualdade triangular, tem-se que $d_L(\overline{\Gamma_j}, t_i) \leq d_L(\overline{\Gamma_j}, \overline{\Gamma^*}) + d_L(\overline{\Gamma^*}, t_i)$. Substituindo-se o lado direito da Inequação 5.2 por essa desigualdade, obtém-se:

$$u * \sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq u * \sum_{j=1}^{u} d_L(\overline{\Gamma_j}, \overline{\Gamma^*}) + u * \sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) \tag{5.3}$$

Renomeando-se, por conveniência, a variável j por i, dividindo ambos os lados por u ($u > 0$), e utilizando-se a igualdade $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) = \texttt{opt}(I)$, tem-se:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma_i}, \overline{\Gamma^*}) + \texttt{opt}(I) \tag{5.4}$$

Novamente, utilizando-se a desigualdade triangular associada à Distância de Levenshtein, tem-se que $d_L(\overline{\Gamma_i}, \overline{\Gamma^*}) \leq d_L(\overline{\Gamma_i}, t_i) + d_L(t_i, \overline{\Gamma^*})$. Assim, é possível expandir o termo $\sum_{i=1}^{u} d_L(\overline{\Gamma_i}, \overline{\Gamma^*})$ na Inequação 5.4 como mostrado a seguir:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i) + \sum_{i=1}^{u} d_L(t_i, \overline{\Gamma^*}) + \texttt{opt}(I) \tag{5.5}$$

Aplicando-se novamente a igualdade $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) = \texttt{opt}(I)$ tem-se:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i) + \texttt{opt}(I) + \texttt{opt}(I) \tag{5.6}$$

De acordo com o Lema 5.1.1, é possível substituir $\sum_{i=1}^{u} d_L(\overline{\Gamma_i}, t_i)$ por $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i)$ na Inequação 5.6:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq \sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) + \texttt{opt}(I) + \texttt{opt}(I) \tag{5.7}$$

Por fim, aplicando-se a igualdade $\sum_{i=1}^{u} d_L(\overline{\Gamma^*}, t_i) = \texttt{opt}(I)$ mais uma vez, tem-se:

$$\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i) \leq 3 * \texttt{opt}(I) \tag{5.8}$$

Demonstrando, portanto, que o valor da solução Γ devolvida pelo Algoritmo 5.1 é, no máximo, três vezes maior do que o valor ótimo esperado para o PASM.

■

Sobre a complexidade de tempo do Algoritmo 5.1, observe que durante a sua execução, são obtidas u soluções para o PAS, uma para cada sequência i de $\mathcal{T}$. Como isso é feito utilizando-se o algoritmo de Gelfand, é sabido que o tempo gasto na solução de cada instância é polinomial. Além disso, cada solução deve ser comparada com as demais sequências de $\mathcal{T}$, para se calcular a soma da Distância de Levenshtein de Γ_i com os elementos de $\mathcal{T}$, e então decidir pela solução de soma mínima. A Distância de Levenshtein entre duas sequências também pode ser obtida em tempo polinomial através do Algoritmo 2.1, de Needleman-Wunsch. Assim, considerando-se novamente que a cobertura da sequência g pelos blocos em $\mathcal{B}$ é denotada por $c = \frac{1}{n}\sum_{k=1}^{p}|b_k|$, que o tamanho de g é dado por n, que a maior sequência de $\mathcal{T}$ tenha tamanho m, e que p seja o tamanho do conjunto $\mathcal{B}$, a complexidade de tempo do Algoritmo 5.1 é $\mathcal{O}(um(nc + p^2 + un))$ e, portanto, polinomial.

Visando tornar a prova mais clara e bem definida, a Distância de Levenshtein foi adotada como métrica de distância. Porém, ressalta-se aqui que a demonstração permaneceria válida se outra métrica qualquer fosse utilizada em seu lugar, uma vez que ela está baseada apenas na propriedade da desigualdade triangular obedecida por essas funções.

Capítulo 6

Uma Formulação de PLI para o PASM

Uma outra forma de abordar problemas de otimização NP-completos, como é o caso da PASM, é através da Programação Linear Inteira (PLI). Através dela, é possível obter a solução ótima de maneira mais elaborada do que por meio da força bruta, empregando técnicas como um algoritmo *branch-and-bound* para descartar de antemão possibilidades que não sejam viáveis ou que não melhorem a solução atual. Essa foi a motivação para o desenvolvimento de uma formulação de PLI para o PASM, que será apresentada neste capítulo.

6.1 Formulação de PLI

O processo de transformar a descrição de um problema em uma formulação de PLI deve ser realizado seguindo critérios sistemáticos, para garantir a qualidade do modelo construído. Wolsey em [44] esquematiza passos a serem seguidos na elaboração de um modelo de programação linear inteira para um problema:

1. Definir quais são as prováveis variáveis necessárias;
2. Empregar essas variáveis para definir um conjunto de restrições de forma a garantir que soluções viáveis do modelo correspondam também a soluções viáveis do problema; e
3. Empregar essas variáveis para definir a função objetivo.

Esses passos foram aplicados no desenvolvimento do modelo de PLI aqui proposto, aliados a iterações de melhorias e simplificações da formulação conforme restrições redundantes eram observadas. O modelo resultante é apresentado nesta seção, e corresponde a uma formulação do PASM para encontrar sua solução ótima através de programação linear inteira. Pelo fato do modelo proposto não ser trivial, serão inicialmente enumeradas e detalhadas as variáveis utilizadas na formulação do problema, seguidas pelas constantes e função objetivo. Por fim, são descritas as restrições a que se encontram sujeitas as variáveis do modelo. A notação de cada um desses elementos que compõem

o modelo está acompanhada de uma breve descrição da sua finalidade, no intuito de auxiliar no seu entendimento.

Relembrando as definições e notações empregadas no problema, uma instância do PASM consiste de uma sequência g, com n caracteres, um conjunto $\mathcal{T}$ de u sequências e um conjunto ordenado $\mathcal{B}$ de b segmentos de g. Adotando a Distância de Levenshtein como função de pontuação, objetiva-se encontrar uma cadeia Γ de $\mathcal{B}$ tal que $\sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i)$ seja a menor possível.

Na busca por uma solução Γ do PASM, serão selecionados um certo número de blocos ordenados e não sobrepostos de g a serem concatenados e alinhados às u sequências de $\mathcal{T} = \{t_1, t_2, ..., t_u\}$, gerando portanto u alinhamentos. Tais alinhamentos estão representados no modelo de PLI pelo par de sequências g'_q e t'_q, com $1 \leq q \leq u$, construídas sobre um alfabeto $\Sigma \cup \{\text{`}-\text{'}\}$. Será feito uso também dos índices i, com $1 \leq i \leq |g|$, j, com $1 \leq j \leq |t_q|$, k, com $1 \leq k \leq |g| + |t_q|$, l e r, com $1 \leq l, r \leq b$, além da constante $M = |g| + \max(|t_q|)$.

Detalhes do modelo com uma descrição de suas variáveis, constantes e restrições podem ser encontrados a seguir.

Variáveis:

1. X_{qik}: determina se a posição k de g'_q contém o caractere i de g
   $$X_{qik} = \begin{cases} 1, \text{ se } g[i] \text{ está em } g'_q[k] \\ 0, \text{ caso contrário} \end{cases}$$

2. Y_{qjk}: determina se a posição k de t'_q contém o caractere j de t_q
   $$Y_{qjk} = \begin{cases} 1, \text{ se } t_q[j] \text{ está em } t'_q[k] \\ 0, \text{ caso contrário} \end{cases}$$

3. Z_{qijk}: indica a existência do caractere i de g em $g'[k]$ e do caractere j de t_q em $t'_q[k]$
   $$Z_{qijk} = \begin{cases} 1, \text{ se } g[i] \text{ e } t_q[j] \text{ estão em } g'_q[k] \text{ e } t'_q[k], \text{ respectivamente} \\ 0, \text{ caso contrário} \end{cases}$$

4. Θ^g_{qk}: indica a existência de um espaço na posição k de g'_q
   $$\Theta^g_{qk} = \begin{cases} 1, \text{ se há um espaço em } g'_q[k] \\ 0, \text{ caso contrário} \end{cases}$$

5. Θ^t_{qk}: indica a existência de um espaço na posição k de t'_q
   $$\Theta^t_{qk} = \begin{cases} 1, \text{ se há um espaço em } t'_q[k] \\ 0, \text{ caso contrário} \end{cases}$$

6. α_l: indica os blocos de $\mathcal{B}$ selecionados para compor a solução Γ
   $$\alpha_l = \begin{cases} 1, \text{ se } b_l \text{ é escolhido para } \Gamma \\ 0, \text{ caso contrário} \end{cases}$$

7. β_{qk}: indica se a posição do alinhamento k é válida, ou seja, se existe um caractere diferente de $\{\text{`}-\text{'}\}$ em g'_q e/ou t'_q
   $$\beta_{qk} = \begin{cases} 1, \text{ se há um caractere em } g'_q[k] \text{ e/ou } t'_q[k] \\ 0, \text{ caso contrário} \end{cases}$$

8. γ_i: indica se o caractere $g[i]$ está em Γ^* ou não
   $\gamma_i = \begin{cases} 1, \text{ se } g[i] \text{ faz parte de } \Gamma^* \\ 0, \text{ caso contrário} \end{cases}$

Com relação às variáveis descritas acima, tem-se que:

$$X_{qik}, Y_{qjk}, Z_{qijk}, \Theta^g_{qk}, \Theta^t_{qk}, \alpha_l, \beta_{qk}, \gamma_i \in \{0, 1\}$$

Portanto, a formulação aqui proposta pode ser considerada de programação linear binária, uma vez que as suas oito variáveis são binárias.

Constantes:

1. I_{lr}: indica blocos incompatíveis, ou seja, que se sobrepõem
   $I_{lr} = \begin{cases} 1, \text{ se } \mathcal{B}_l \text{ e } \mathcal{B}_r \text{ têm intersecção} \\ 0, \text{ caso contrário} \end{cases} \quad l \neq r$
2. A_{il}: indica quais caracteres fazem parte de cada bloco
   $A_{il} = \begin{cases} 1, \text{ se } g[i] \in \mathcal{B}_l \\ 0, \text{ caso contrário} \end{cases}$
3. w_{qij}: indica se dois caracteres de g e t_q são diferentes
   $w_{qij} = \begin{cases} 1, \text{ se } g[i] \text{ é diferente de } t_q[j] \\ 0, \text{ caso contrário} \end{cases}$

Função Objetivo:

$$min \sum_{q=1}^{u} [\sum_{i=1}^{n} \sum_{j=1}^{|t_q|} \sum_{k=1}^{n+|t_q|} w_{qij} * Z_{qijk} + \sum_{k=1}^{n+|t_q|} (\Theta^g_{qk} + \Theta^t_{qk})]$$

Por empregar a Distância de Levenshtein como métrica na comparação das sequências, o PASM é formulado como um problema de minimização. O valor da função objetivo é calculado somando-se a Distância de Levenshtein entre a sequência gerada pela concatenação da cadeia Γ, devolvida como solução do problema, e cada uma das sequências $t_q \in \mathcal{T}$, o que é tratado pela expressão interna ao somatório que possui q como índice. O caso em que há um caractere tanto em $g'[k]$ como em $t'_q[k]$ após o alinhamento é considerado no primeiro termo desta expressão. Nessa situação, Z_{qijk} é igual a 1, e então a constante w_{qij} indica se tais caracteres são iguais ($w_{qij} = 0$) e o valor da distância não deve se alterar, ou se são caracteres diferentes ($w_{qij} = 1$) e o valor precisa ser incrementado em uma unidade. Já a presença de um espaço em g' ou em t'_q é representada no segundo termo, em que soma-se à distância o valor de Θ^g_{qk} e Θ^t_{qk}, que assumem o valor 1 apenas se há um espaço na posição k de suas respectivas sequências, havendo a restrição de que nunca ambos são iguais a 1.

Restrições:

1. Garante que o caractere $g[i]$ ou esteja presente uma única vez em Γ^* ou não faça parte da solução:

$$\sum_{k=1}^{n+|t_q|} X_{qik} = \gamma_i, \forall i, \forall q$$

2. Garante que o caractere $t_q[j]$ esteja presente exatamente uma vez em t'_q:

$$\sum_{k=1}^{n+|t_q|} Y_{qjk} = 1, \forall j, \forall q$$

3. Garante que blocos que se sobrepõem não sejam ambos escolhidos para compor uma solução:

$$\alpha_l + \alpha_r \leq 1(l \neq r), \forall l, r \text{ com } I_{lr} = 1$$

4. Garante que caso o i-ésimo caractere de g esteja em $g'[k]$, então nenhum dos caracteres que antecedem $g[i]$ em g podem assumir posições posteriores a $k-1$ em g':

$$\sum_{l=k}^{n+|t_q|} \sum_{i'=1}^{i-1} X_{qi'l} + M * X_{qik} \leq M, \forall i, \forall q, \forall k$$

5. Garante que caso o j-ésimo caractere de t_q esteja em $t'_q[k]$, então nenhum dos caracteres que antecedem $t_q[j]$ em t_q podem assumir posições posteriores a $k-1$ em t'_q:

$$\sum_{l=k}^{n+|t_q|} \sum_{j'=1}^{j-1} Y_{qj'l} + M * Y_{qjk} \leq M, \forall j, \forall q, \forall k$$

6. Garante que caso o i-ésimo caractere de g esteja em $g'[k]$, então nenhum dos caracteres que sucedem $g[i]$ em g podem assumir posições anteriores a $k+1$ em g':

$$\sum_{l=1}^{k} \sum_{i'=i+1}^{n} X_{qi'l} + M * X_{qik} \leq M, \forall i, \forall q, \forall k$$

7. Garante que caso o j-ésimo caractere de t_q esteja em $t'_q[k]$, então nenhum dos caracteres que sucedem $t_q[j]$ em t_q podem assumir posições anteriores a $k+1$ em t'_q:

$$\sum_{l=1}^{k} \sum_{j'=j+1}^{|t_q|} Y_{qj'l} + M * Y_{qjk} \leq M, \forall j, \forall q, \forall k$$

8. Garante que seja colocado um espaço em $g'_q[k]$ caso haja um caractere em $t'_q[k]$ mas não em $g'_q[k]$:

$$\sum_{i=1}^{n} X_{qik} + \Theta^g_{qk} = \beta_{qk}, \forall q, \forall k$$

9. Garante que seja colocado um espaço em $t'_q[k]$ caso haja um caractere em $g'_q[k]$ mas não em $t'_q[k]$:

$$\sum_{j=1}^{|t_q|} Y_{qjk} + \Theta^t_{qk} = \beta_{qk}, \forall q, \forall k$$

10. Garante que, se houver um espaço na posição k do alinhamento, ele esteja ou em $g'_q[k]$, ou em $t'_q[k]$, mas nunca em ambos:

$$\Theta^g_{qk} + \Theta^t_{qk} \leq 1, \forall q, \forall k$$

11. Garante que Z_{qijk} será 1 sempre que haja caracteres em $g'_q[k]$ e $t'_q[k]$:

$$X_{qik} + Y_{qjk} - 1 \leq Z_{qijk}, \forall i, \forall j, \forall q, \forall k$$

12. Garante que γ_i seja 1 caso o i-ésimo caractere de g esteja presente em Γ^*, ou seja, caso $g[i]$ faça parte de um dos blocos selecionados:

$$\sum_{l=1}^{b} A_{il} * \alpha_l = \gamma_i, \forall i$$

13. Garante que seja dada preferência a preencher a posição k de g'_q com um caractere ao invés de com um espaço:

$$\Theta^g_{qk} \leq \sum_{i=1}^{n} X_{qik}, \forall i, \forall q, \forall k$$

Um dos entraves consequentes da complexidade do modelo apresentado diz respeito à memória exigida para sua implementação. É possível observar que, com exceção da variável 6 referente aos blocos selecionados (b), nas demais variáveis são essenciais o tamanho da sequência g (n) e/ou t_q (m), assim como do conjunto $\mathcal{T}$ (u), de forma que o número de variáveis do modelo é dado pela expressão $u(n+m)(n+m+nm+3)+n+b$. O mesmo ocorre em relação às restrições, em que excluindo-se a Restrição 3 que se baseiam em b, todas as restantes são dependentes de n, e/ou m e/ou u, sendo que a quantidade de restrições de determinada instância é calculada por $u(n+m)(2n+2m+nm+4u)+un+um+(b^2-b)/2+n$. Como esses valores costumam ser muito grandes em instâncias reais do problema da identificação de genes, é esperado que o consumo de memória se torne um gargalo do modelo.

Capítulo 7

Avaliação das Abordagens Propostas

A fim de avaliar a eficácia das duas abordagens propostas para o PASM neste trabalho, o algoritmo de aproximação e a formulação de PLI foram implementados e submetidos a testes experimentais. Para isso, utilizou-se um conjunto de instâncias artificiais do problema para a comparação das duas estratégias, tendo ainda a aproximação sido empregada em testes reais do problema da identificação de genes. Em ambas as rodadas de testes, confrontou-se os resultados obtidos pela aproximação e pelo PLI com aqueles alcançados pelas heurísticas propostas para o PASM em [24, 25]. Neste capítulo serão detalhadas as baterias de teste realizadas, assim como as condições em que elas ocorreram e os resultados obtidos pelas implementações em cada ocasião.

7.1 Considerações gerais

A implementação do algoritmo de aproximação foi realizada em ANSI C++, e o algoritmo que implementa a formulação de PLI foi desenvolvido em ANSI C com o uso da biblioteca de PLI da FICO, o Xpress [14], versão 23.01.06. Optou-se pela utilização do resolvedor de PLI Xpress pois este figurou como o segundo melhor resolvedor para problemas de programação linear segundo a análise conduzida por Meindl e Templ em [32], em que foi avaliado o desempenho de resolvedores tanto comerciais como de código aberto. Além disso, o resolvedor da FICO estava disponível para uso no laboratório da FACOM/UFMS por meio de licença acadêmica.

Em relação às heurísticas propostas em [24, 25], escolheu-se descartar a heurística do algoritmo genético nos testes com instâncias artificiais devido aos resultados inferiores por ela alcançados em [24, 25] e principalmente pelo fato dela não ser determinística. Assim, nessa primeira etapa dos testes considerou-se as heurísticas da sequência central (H1), da sequência consenso (H2) e do consenso de blocos (H3). As implementações originais das três heurísticas, desenvolvidas em ANSI C por Kishi, foram adaptadas para empregar a Distância de Levenshtein ao invés da similaridade em suas soluções para o PASM. Já para os testes envolvendo sequências reais, o algoritmo genético voltou a ser incluído no comparativo dos resultados.

Devido à necessidade da licença para utilizar o Xpress nos testes da formulação PLI, foi utilizada uma máquina com processador Intel® Core™ 2 Duo CPU E7400 de 2.80GHz e 8 GB de memória que tinha tal resolvedor de PLI instalado e disponível para uso. Para manter a consistência da avaliação, a mesma máquina foi utilizada nos demais testes com a aproximação e as heurísticas.

7.2 Aplicação da 3-aproximação e da formulação de PLI do PASM sobre instâncias artificiais

A primeira avaliação das abordagens utilizou um conjunto de instâncias para o PASM geradas artificialmente. Com a análise dos resultados obtidos nessa bateria de testes, foi possível avaliar a eficácia e eficiência das duas novas estratégias propostas para o PASM, traçando um paralelo com os resultados alcançados pelas heurísticas já introduzidas para o problema.

Sobre o conjunto de testes artificiais utilizado, ele incluiu um total de 225 instâncias, cada qual contendo uma sequência g, um conjunto $\mathcal{T}$ de u sequências e um conjunto de b blocos $\mathcal{B}$. Nessas instâncias, o tamanho das sequências g variam de 8 a 50 caracteres, o conjunto $\mathcal{T}$ possui de 3 a 5 sequências, cujos tamanhos variam de 2 a 21 caracteres, e o conjunto $\mathcal{B}$ possui no mínimo 3 e no máximo 10 blocos, com tamanho limitado pelo comprimento das sequências de $\mathcal{T}$. Todas as sequências foram construídas sobre o alfabeto da Língua Portuguesa, constituído de 26 letras, de A a Z.

A limitação nos tamanhos das sequências que formam as instâncias artificiais deu-se devido à formulação de PLI proposta. Como ela é bastante complexa, o número de variáveis e restrições necessárias para modelar uma instância do problema cresce muito rapidamente em relação a g, u, $|t_i|$ e b. Dessa forma, foi preciso restringir-se ao uso de instâncias pequenas para garantir a viabilidade do experimento.

Uma observação necessária a ser feita antes da apresentação dos resultados diz respeito ao algoritmo exato desenvolvido com base na formulação de PLI para o PASM. Enquanto a aproximação e as heurísticas obtiveram uma solução final para todas as instâncias do conjunto de testes, a abordagem baseada no modelo de PLI apenas concluiu sua execução em 29 das instâncias. Isso ocorreu pois se fez necessário estabelecer um limitante de tempo máximo aceitável dentro do qual o algoritmo pudesse se manter executando. Após testes preliminares, observou-se que a solução alcançada durante os primeiros minutos de execução era mantida na grande maioria dos casos, alterando-se apenas o limitante dual. Portanto, o limitante adotado foi de 3600 segundos, o equivalente a uma hora.

Ainda devido ao fato do algoritmo que implementa a formulação de PLI não ter apresentado a solução ótima em todos os casos, para avaliar a qualidade das soluções obtidas com essa estratégia foi calculado o GAP (em %) entre a solução encontrada e o limitante dual correspondente, obtido da seguinte forma:

$$gap = \frac{\text{melhor solução - limitante dual}}{\text{limitante dual}}$$

Um GAP próximo de 0 indica que a solução está mais próxima de convergir para a

solução ótima buscada. Como essa é uma medida associada apenas à primeira estratégia, ela consta apenas na tabela de resultados detalhados presente no Apêndice A, juntamente com as características de cada instância considerada, e os resultados $\texttt{val}(S_I) = \sum_{i=1}^{u} d_L(\overline{\Gamma}, t_i)$ obtidos pelas cinco abordagens.

A seguir, a Tabela 7.1 explicita a soma dos resultados $\texttt{val}(S_I)$ obtidos pelas abordagens para cada instância I da bateria de testes artificiais. É possível notar que a Heurística H3 apresentou o pior resultado dentre os cinco métodos comparados, seguida da formulação PLI. Esse resultado do modelo de PLI se deve, basicamente, ao que já foi exposto sobre o limite de tempo estabelecido para sua execução. Já o melhor resultado foi obtido pela aproximação, que superou a pontuação das três heurísticas. A qualidade das heurísticas se mostrou a mesma observada em [24, 25], com a Heurística 1 apresentando os melhores resultados, seguida da Heurística 2 e da Heurística 3, respectivamente.

Tabela 7.1: Resultados obtidos aplicando-se as cinco abordagens sendo comparadas sobre um conjunto de 225 instâncias artificiais do PASM.

Abordagem	Pontuação
Formulação PLI	5487
PASM 3−aprox	5253
Heurística 1	5390
Heurística 2	5452
Heurística 3	5587

Analisando os resultados das 29 instâncias resolvidas de forma ótima pelo algoritmo que implementa a formulação de PLI ($GAP = 0\%$), mostradas na Tabela 7.2, tem-se que os resultados obtidos com a 3−aproximação foram os mesmos da Formulação PLI, que são sabidamente ótimos. Esses resultados novamente superam os alcançados pelas Heurística 1, Heurística 2 e Heurística 3, que obtiveram resultados ótimos apenas em 21, 19 e 17 instâncias desse grupo respectivamente.

Tabela 7.2: Resultados obtidos aplicando-se as cinco abordagens sendo comparadas sobre o conjunto de 29 instâncias artificiais do PASM resolvidas de forma ótima pela Formulação PLI.

Abordagem	Pontuação
Formulação PLI	234
PASM 3−aprox	234
Heurística 1	259
Heurística 2	263
Heurística 3	274

Em relação ao desempenho, a única abordagem que se diferenciou em relação às demais foi a Formulação PLI, cujo tempo de execução foi limitado em 3600 segundos para cada instância, tempo este que não foi suficiente para que o GAP fosse reduzido a 0 em

87, 9% dos casos de teste. Já as quatro estratégias restantes devolveram suas soluções para todas as instâncias em tempo insignificante, da ordem de milésimos de segundos.

7.3 Aplicação da 3-aproximação do PASM no Problema da Identificação de Genes

Uma das principais motivações para o estudo do Problema do Alinhamento *Spliced* Múltiplo é a sua relação direta com o problema da identificação de genes (PIG). É possível associar esses dois problemas olhando-se para a sequência base do PASM como o DNA do PIG, as sequências modelo do PASM como os cDNAs do PIG e, finalmente, para B como um conjunto de possíveis éxons do DNA. Considerando que os cDNAs estão evolutivamente relacionados com o gene sendo buscado no DNA, é esperado que uma solução do PASM corresponda a uma resposta aceitável para o problema da identificação de genes por comparação de DNA com cDNA. Essa afirmação pode ser feita devido ao princípio da conservação das bases e da inexistência de íntrons no cDNA, permitindo a correspondência direta entre instâncias de ambos os problemas.

Nesse contexto, é importante observar que uma solução ótima para o PASM nem sempre coincide com a solução real do problema da identificação de genes, ou seja, os blocos pertencentes a uma solução exata do PASM não necessariamente correspondem aos éxons reais do gene sendo buscado, apesar da grande possibilidade de o serem. Assim, é interessante avaliar o desempenho da aproximação para o PASM quando aplicada sobre o problema da identificação de genes para instâncias reais. Essa tarefa foi realizada em [24, 25] para as heurísticas propostas na ocasião, e agora o mesmo método foi utilizado para analisar a performance da 3-aproximação aqui introduzida.

7.3.1 Medidas de precisão de predição de genes

No intuito de avaliar a acurácia das abordagens aqui comparadas, adotou-se um conjunto de medidas de precisão de predições de genes, medidas essas calculadas em dois níveis: de nucleotídeos e de éxons. Essas medidas correspondem àquelas propostas em [7] e retomadas a seguir.

No nível de nucleotídeos, a predição recebida por cada nucleotídeo, ou seja, se a ferramenta o classificou como codificante ou não-codificante, é confrontada com sua classificação real, isto é, verifica-se se o nucleotídeo é de fato codificante ou não-codificante, segundo a anotação reconhecida como correta para a instância. Associando-se a essas informações uma variável binária, pode-se construir uma matriz de contingência 2×2 para representar a relação entre os nucleotídeos reais e preditos. A Figura 7.1 ilustra essa matriz, cujas linhas indicam a predição e as colunas a classificação real dos nucleotídeos. Cruzando-se as informações, a diagonal principal da matriz indica os acertos da predição - verdadeiros positivos (VP) e verdadeiros negativos (VN). Já a diagonal secundária traz os erros, referentes aos nucleotídeos cujas predições divergiram da realidade - falsos positivos (FP) e falsos negativos (FN).

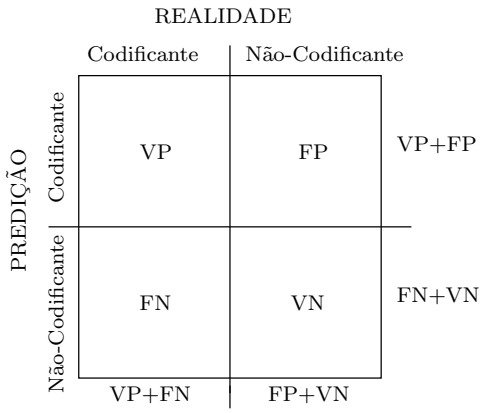

Figura 7.1: Matriz de contingência para a avaliação da precisão de predições de genes (figura adaptada de Burset e Guigó [7]).

Para extrair informações de precisão da matriz de contingência especificada, duas medidas escalares são comumente empregadas: a Sensibilidade (S_n) e a Especificidade (S_p). A **sensibilidade** mede a proporção de nucleotídeos preditos corretamente como codificantes, enquanto que a **especificidade** indica a proporção de nucleotídeos preditos corretamente como não-codificantes. Tradicionalmente, elas são computadas da seguinte maneira:

$$S_{n_n} = \frac{VP}{VP + FN} \quad e \quad S_{p_n} = \frac{VN}{VN + FP} \tag{7.1}$$

Observe que a quantidade de nucleotídeos não-codificantes é muito maior do que a de codificantes, tanto na realidade como nas previsões, de forma que se calculada como em 7.1, a especificidade irá produzir um valor pouco informativo. Sendo assim, é comum na literatura essa medida ser calculada de forma a indicar a proporção de nucleotídeos preditos como codificantes que são de fato codificantes:

$$S_{p_n} = \frac{VP}{VP + FP} \tag{7.2}$$

Será utilizada ainda uma outra medida que sintetiza a matriz de contingência 2×2, conhecida como Correlação Aproximada (CA) e definida no intervalo de -1 a 1. A CA pode ser vista como uma medida global de precisão de predição de genes, representando a associação entre o que foi predito e a realidade conhecida. Sua fórmula encontra-se detalhada na Equação 7.3:

$$CA = \frac{1}{2}\left(\frac{VP}{VP + FN} + \frac{VP}{VP + FP} + \frac{VN}{VN + FP} + \frac{VN}{VN + FN}\right) - 1 \tag{7.3}$$

Os resultados da predição foram também avaliados no nível de éxons, que estabelece outras medidas muito empregadas na avaliação de estratégias para a predição de genes. Nesse caso, é importante definir o critério adotado para indicar se um éxon foi corretamente predito, uma vez que existem diferentes abordagens para isso. Em nossos testes, a predição de um éxon é considerada correta apenas se as coordenadas desse éxon predito coincidem com as coordenadas de um éxon real presente na sequência de

DNA alvo. Éxons tais que apenas as coordenadas de início ou fim coincidam, ou que possuam somente trechos sobrepostos a algum gene real não são considerados como éxons corretos. Com base nesse critério, tem-se que NEC indica a quantidade de éxons corretamente preditos, NEP o total de éxons preditos e NEA a quantidade total de éxons reais na sequência anotada. No nível de éxons, então, a sensibilidade (S_{n_e}) e a especificidade (S_{p_e}) são calculadas utilizando-se as equações em 7.4:

$$S_{n_e} = \frac{NEC}{NEA} \;\; e \;\; S_{p_e} = \frac{NEC}{NEP} \tag{7.4}$$

Por fim, ainda é calculada a média aritmética entre os valores da sensibilidade e da especificidade, sumarizando as duas medidas no nível de éxons, análogo ao que é feito pela CA no nível de nucleotídeos:

$$Av_e = \frac{S_{n_e} + S_{p_e}}{2} \tag{7.5}$$

7.3.2 *Benchmark* e execução dos testes

Para proporcionar uma melhor avaliação da acurácia da aproximação sugerida na tarefa da identificação de genes, possibilitando a comparação dos seus resultados com aqueles previamente obtidos pelas heurísticas desenvolvidas e testadas por Kishi e Adi, foi escolhido como *benchmark* o mesmo empregado em [24, 25]. Esse conjunto de testes consiste de 240 fragmentos de sequências de DNA humano, extraídos dos cromossomos analisados pelo projeto ENCODE [41]. Todos estes fragmentos possuem apenas um gene e as sequências alvo correspondentes foram obtidas a partir de uma busca na base de dados HomoloGene [37] por sequências de cDNA evolutivamente relacionadas aos genes em questão. Por fim, o conjunto ordenado de segmentos para cada instância foi construído com uso da ferramenta GenScan [6], por meio de um algoritmo baseado em um modelo estatístico denominado de Modelo Oculto de Markov (do inglês, *Hidden Markov Model* - HMM) [36]. Nessas instâncias, o tamanho do DNA varia de 2842 a 575746 nucleotídeos, $\mathcal{T}$ possui de 1 a 5 sequências, com tamanhos variando de 222 a 6915 nucleotídeos cada, e a quantidade de blocos fornecidos como entrada varia de 2 a 622 de acordo com a instância em questão. Mais detalhes sobre a construção do conjunto de testes podem ser conferidos no trabalho original.

Os testes com as quatro heurísticas anteriormente propostas foram repetidos na mesma máquina em que foi executada a aproximação, utilizando o código original gentilmente cedido pelos autores. A intenção com isso é atualizar o tempo de execução com o tempo gasto no equipamento com a configuração descrita a fim de possibilitar a comparação da eficiência de cada abordagem.

7.3.3 Resultados

Dadas as medidas de avaliação empregadas, o *benchmark* utilizado e as condições em que os testes ocorreram, o Apêndice B detalha os resultados alcançados pela aplicação da 3-aproximação aqui proposta para o PASM na tarefa de identificação de genes para as 240 instâncias do conjunto de testes. A Tabela 7.3 traz uma média desses valores,

confrontados também com os resultados obtidos pelas heurísticas descritas em [24, 25] para o mesmo problema.

Os resultados apresentados na tabela estão divididos conforme os dois níveis avaliados: de nucleotídeos e de éxons. Em cada nível, é exibida a média aritmética[1] da sensibilidade (S_{n_n} e S_{n_e}), da especificidade (S_{p_n} e S_{p_e}) e das medidas que as sumarizam (AC e Av_e). As linhas da primeira coluna especificam as cinco abordagens comparadas, sendo a primeira referente à aproximação aqui desenvolvida e as demais às quatro heurísticas já avaliadas.

Tabela 7.3: Resultados obtidos aplicando-se as cinco abordagens sendo comparadas sobre um conjunto de 240 instâncias reais do problema da identificação de genes.

Abordagem	Nucleotídeo			Éxon		
	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
PASM-3-APROX	0.95	**0.96**	**0.95**	0.85	0.81	0.83
HEURÍSTICA H1	**0.96**	**0.96**	**0.95**	**0.86**	0.81	0.83
HEURÍSTICA H2	**0.96**	0.91	0.93	0.83	0.73	0.78
HEURÍSTICA H3	0.93	**0.96**	0.94	**0.86**	**0.84**	**0.85**
HEURÍSTICA H4	0.77	0.80	0.77	0.54	0.51	0.53

Em relação aos nucleotídeos codificantes reais, a aproximação identificou corretamente 95% deles (S_{n_n}), e 96% de todos os nucleotídeos preditos por ela como codificantes são de fato codificantes (S_{p_n}). Já no nível de éxons, 85% dos éxons reais foram corretamente identificados (S_{n_e}), e 81% dos éxons preditos pela aproximação são de fato éxons reais. Em números absolutos, de um total de 338778 nucleotídeos realmente codificantes, 323883 foram identificados corretamente pela aproximação, enquanto que 10606 nucleotídeos foram preditos incorretamente como codificantes. Já no nível de éxons, foram preditos pela ferramenta que implementa o algoritmo de aproximação um total de 1803 éxons, dos quais 1483 são de fato reais e 320 constituem falsos positivos, sendo que as instâncias totalizavam 1677 éxons reais.

É trivial notar que a precisão da aproximação na tarefa de identificação dos éxons de um gene depende diretamente do conjunto $\mathcal{B}$ de entrada. Se neste conjunto estiverem inclusos todos os éxons do gene em questão, é bastante provável que eles sejam corretamente identificados pelo algoritmo de aproximação. Porém, se um éxon real de um gene g não pertencer ao $\mathcal{B}$ passado como entrada referente a esta instância, tal éxon será perdido pela abordagem aproximada. O mesmo ocorre com as heurísticas avaliadas. Nesse conjunto de testes, de um total de 1667 éxons reais anotados, 1550 encontram-se presentes (como éxons candidatos) nos respectivos conjuntos $\mathcal{B}$ das instâncias de entrada. Desses éxons, apenas 67 não foram identificados pelo algoritmo de aproximação. Os outros 127 éxons reais não preditos pela estratégia não constavam como elementos de $\mathcal{B}$, de forma que também não foi possível identificá-los corretamente.

[1] Para cada uma das 240 instâncias do conjunto de testes, calculou-se os seus valores para as seis medidas consideradas, que foram então somados e divididos pela quantidade total de instâncias, resultando na média aritmética apresentada na Tabela 7.3.

Como é possível verificar na Tabela 7.3, especialmente através das medidas CA e Av_e que sumarizam as demais, nossa aproximação alcançou de uma maneira geral índices de sensibilidade e especificidade similares aos obtidos pelas melhores heurísticas, em especial no nível de nucleotídeos. Além disso, no nível de éxons os resultados também foram promissores, havendo pouca diferença entre a acurácia da PASM-3-APROX e a da HEURÍSTICA H3, a melhor nesta perspectiva.

Detalhando-se um pouco mais os resultados da comparação, os resultados da PASM-3-APROX equiparam-se aos da HEURÍSTICA H1, com essa sendo ligeiramente superior à aproximação na sensibilidade em ambos os níveis. A PASM-3-APROX superou a HEURÍSTICA H4 em todos as medidas, e foi superada pela HEURÍSTICA H2 apenas com relação à S_{n_n} em 1%. No que diz respeito à comparação com a HEURÍSTICA H3, a precisão da aproximação foi superior no nível de nucleotídeos, porém um pouco abaixo na dimensão dos éxons, com variação de 1% na sensibilidade e 3% na especificidade.

A Tabela 7.4 apresenta o tempo médio de execução gasto nos testes com as 240 instâncias reais por cada método, assim como o maior tempo alcançado dadas todas as instâncias.

Tabela 7.4: Comparativo do tempo de execução (em segundos) das diferentes abordagens para o mesmo conjunto de testes.

Abordagem	Tempo Médio (s)	Pior Tempo (s)
PASM-3-APROX	6.57	188.97
HEURÍSTICA H1	2.45	58.53
HEURÍSTICA H2	2.80	65.86
HEURÍSTICA H3	7.79	289.05
HEURÍSTICA H4	104.37	1854.40

Em relação ao tempo de execução gasto por cada estratégia, fica evidente que o desempenho da 3-aproximação assemelha-se ao da HEURÍSTICA H3, aquela com o segundo pior tempo dentre as quatro heurísticas avaliadas. De qualquer forma, o tempo gasto pela aproximação na solução das instâncias pode ser considerado razoável, uma vez que retornou uma solução aproximada para todas as instâncias em menos do que 189 segundos (pior tempo observado para a instância *snd1*), enquanto que a média de resposta ficou próxima aos 7 segundos.

Capítulo 8

Conclusão

8.1 Considerações finais

Problemas envolvendo sequências são de grande interesse em Teoria da Computação por apresentarem aplicações nas mais diversas subáreas da Ciência da Computação, como Banco de Dados, Inteligência Artificial e Bioinformática. Beneficiada por sua interdisciplinaridade, esta área de pesquisa vem se expandido a ritmo acelerado, com o surgimento de novos problemas e aplicações a serem explorados.

No contexto de problemas envolvendo sequências, escolheu-se estender neste trabalho o estudo de Kishi e Adi acerca do Problema do Alinhamento *Spliced* Múltiplo. Baseado na comparação de segmentos de uma sequência com um conjunto de outras sequências conhecidas, visando encontrar um subconjunto desses blocos que ordenados e concatenados geram uma sequência que seja o mais próximo possível das demais, a principal aplicação prática deste problema teórico é no problema biológico da identificação de genes.

Por ser um problema comprovadamente NP-completo, foram sugeridas e avaliadas em [24, 25] quatro heurísticas para o PASM. Embora os resultados alcançados por elas tenham sido promissores tanto na teoria quanto na prática, heurísticas não garantem um nível de confiança acerca da qualidade das soluções geradas. Preencher essa brecha na investigação do PASM foi então o principal objetivo deste estudo, seguindo uma das sugestões de trabalhos futuros apresentadas no estudo original: *tentar encontrar aproximações para o Problema do Alinhamento Spliced Múltiplo.*

Para contemplar o objetivo proposto, o PASM, que originalmente é um problema de maximização e faz uso da similaridade como medida de comparação entre sequências, foi adaptado para uma versão de minimização, empregando agora a Distância de Levenshtein como função de pontuação. Essa alteração possibilitou o desenvolvimento de um algoritmo de aproximação para o problema que fornece uma solução no máximo 3 vezes maior do que a solução ótima.

Nos testes experimentais realizados, a $3-$aproximação alcançou bons resultados. Para as instâncias artificiais, as soluções geradas pela aproximação foram melhores do que as produzidas pelas três heurísticas determinísticas de [24, 25]. Quando aplicada ao problema da identificação de genes, apresentou resultados comparáveis aos das heurísticas,

especialmente com os da HEURÍSTICA H1, sendo a melhor no nível de nucleotídeos e a segunda melhor no nível de éxons. Como a HEURÍSTICA H1 é baseada na ideia de selecionar uma sequência central de $\mathcal{T}$ e utilizá-la como entrada para o algoritmo de Gelfand a fim de obter uma solução para o PAS que possa ser estendida para o PASM, ficam evidentes as semelhanças com a estratégia adotada na aproximação e os resultados próximos não surpreendem. Em termos de desempenho, a aproximação tem complexidade polinomial, assim como as heurísticas.

Foi ainda empregada outra abordagem que surge como opção para o tratamento de problemas de otimização combinatória: a programação linear inteira. Um modelo de PLI foi proposto para formular o PASM e implementado através de um algoritmo de *branch-and-bound*. Os testes executados com instâncias artificiais evidenciaram que a formulação desenvolvida corresponde a um modelo complexo, cujo consumo de memória é altamente dependente do tamanho das sequências g e $\mathcal{T}$ de entrada. Sendo assim, suas aplicações práticas limitam-se a instâncias em que estes valores são pequenos, o que impossibilitou a aplicação desta abordagem no problema prático da identificação de genes. Contudo, o modelo se mostrou correto, e apesar de nem sempre concluir que suas soluções eram ótimas, devido a convergência entre os limitantes superior e inferior não ocorrer dentro do tempo de execução estipulado, as soluções devolvidas eram próximas aos valores ótimos esperados.

Acredita-se que tenham sido alcançados os objetivos pretendidos com este trabalho, contribuindo com o aprofundamento do estudo do PASM através principalmente do desenvolvimento de uma $3-$aproximação e de um modelo de PLI para o problema.

8.2 Publicações

Um artigo completo sobre o algoritmo de aproximação proposto para o PASM, sua prova e resultados alcançados sobre o problema da identificação de genes em comparação com os obtidos pelas heurísticas foi apresentado na 21ª edição do International Symposium on String Processing and Information Retrieval (SPIRE 2014), e publicado nos anais do evento no Lecture Notes in Computer Science (LNCS).

8.3 Trabalhos futuros

Com o desenvolvimento deste estudo, novas sugestões de trabalhos futuros envolvendo o problema do alinhamento *spliced* múltiplo podem ser adicionadas às encontradas em [24, 25]:

- Verificar se a razão 3 provada para o algoritmo de aproximação aqui sugerido é justa, determinando-se uma instância do problema cujo valor da solução da aproximação seja $3*$ `opt`; ou demonstrar formalmente que a razão pode ser melhorada;
- Aplicar ao modelo proposto outros paradigmas de programação linear inteira (como *branch-and-cut*, *branch-and-price* e geração de colunas) buscando aprimorar o desempenho alcançado;

- Aperfeiçoar o modelo de PLI para o PASM e avaliar sua aplicabilidade na tarefa de identificação de genes com instâncias reais;
- Buscar outras aplicações práticas para o PASM.

Referências Bibliográficas

[1] Adi, S., Braga, M., Fernandes, C., Ferreira, C., Martinez, F., Sagot, M.-F., Stefanes, M., Tjandraatmadja, C., and Wakabayashi, Y. Repetition-free longest common subsequence. *Electronic Notes in Discrete Mathematics 30*, 0 (2008), 243 – 248. The IV Latin-American Algorithms, Graphs, and Optimization Symposium.

[2] Applegate, D. L., Bixby, R. E., Chvátal, V., and Cook, W. J. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[3] Baxevanis, A. D. *An Overview of Gene Identification: Approaches, Strategies, and Considerations*. John Wiley & Sons, Inc., 2002.

[4] Bertsimas, D., and Tsitsiklis, J. *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.

[5] Bonizzoni, P., Vedova, G. D., and Mauri, G. Experimenting an approximation algorithm for the LCS. *Discrete Applied Mathematics 110* (1998), 200–1.

[6] Burge, C., and Karlin, S. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol. 268* (1997), 78–79.

[7] Burset, M., and Guigó, R. Evaluation of gene structure prediction programs. *Genomics 34*, 3 (1996), 353 – 367.

[8] Carvalho, M., de Cerioli, M., Dahab, R., Feofiloff, P., Fernandes, C., Ferreira, C., Guimarães, K., Miyazawa, F., de Pina Jr., J., Soares, J., and Wakabayashi, Y. *Uma Introdução Sucinta a Algoritmos de Aproximação*. XXIII Colóquio Brasileiro de Matemática, Publicações Matemáticas do IMPA, 2001.

[9] Chen, Y. H., Tang, C. Y., and Wu, Q. S. 2-Approximation Algorithms for Median and Centre String Problems. *The 17th Workshop on Combinatorial Mathematics and Computation Theory* (2000).

[10] Clausen, J. Branch and bound algorithms-principles and examples. *Parallel Computing in Optimization* (1997), 239–267.

[11] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.

[12] de Brito, R. T. Alinhamento de Sequências Biológicas. Master's thesis, Universidade de São Paulo, 2003.

[13] de la Higuera, C., and Casacuberta, F. Topology of strings: Median string is np-complete. *Theoretical Computer Science 230*, 1–2 (2000), 39 – 48.

[14] Fico. Fico Xpress Optimization Suite. http://www.fico.com/en/products/fico-xpress-optimization-suite/. Accessado: 28-09-2014.

[15] Gelfand, M. S., Mironov, A. A., and Pevzner, P. A. Gene recognition via spliced sequence alignment. *Proceedings of the National Academy of Sciences of the United States of America 93* (1996), 9061–9066.

[16] Gusfield, D. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology 55*, 1 (1993), 141–154.

[17] Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, USA, 1997.

[18] Hirschberg, D. S. *The Longest Common Subsequence Problem.* PhD thesis, Princeton, NJ, USA, 1975.

[19] Jiang, T., and Li, M. On the approximation of shortest common supersequencesand longest common subsequences. *SIAM J. Comput. 24*, 5 (Oct. 1995), 1122–1139.

[20] Johnson, D. S. Approximation algorithms for combinatorial problems. In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1973), STOC '73, ACM, pp. 38–49.

[21] Junqueira, L., and Carneiro, J. *Biologia Celular e Molecular.* Guanabara Koogan, 2008.

[22] Kann, V. On the Approximability of NP-complete Optimization Problems. Master's thesis, Department of Numerical Analysis and Computing Science - Royal Institute of Technology, 1992.

[23] Kieffer, J. C., Szpankowski, W., and Yang, E.-H. Problems on sequences: Information theory and computer science interface. *IEEE Transactions on Information Theory 50*, 7 (2004), 1385–1392.

[24] Kishi, R. M. Identificação de Genes e o Problema do Alinhamento Spliced Múltiplo. Master's thesis, FACOM-UFMS, Campo Grande, Brasil, 2010.

[25] Kishi, R. M., dos Santos, R. F., and Adi, S. S. Gene prediction by multiple spliced alignment. In *Advances in Bioinformatics and Computational Biology*, O. Norberto de Souza, G. P. Telles, and M. Palakal, Eds., vol. 6832 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 26–33.

[26] Kleinberg, J., and Tardos, E. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[27] Korte, B., and Vygen, J. *Combinatorial Optimization: Theory and Algorithms*, 4th ed. Springer Publishing Company, Incorporated, 2007.

[28] Krane, D., and Raymer, M. *Fundamental Concepts of Bioinformatics*. The Genetics Place Series. Benjamin Cummings, 2003.

[29] Lanctot, J. K., Li, M., Ma, B., Wang, S., and Zhang, L. Distinguishing string selection problems. *Inf. Comput. 185*, 1 (Aug. 2003), 41–55.

[30] Land, A. H., and Doig, A. G. An automatic method of solving discrete programming problems. *Econometrica 28*, 3 (1960), 497–520.

[31] Majoros, W. H. *Methods for Computational Gene Prediction*, 1 ed. Cambridge University Press, Sept. 2007.

[32] Meindl, B., and Templ, M. Analysis of commercial and free and open source solvers for the cell suppression problem. *Trans. Data Privacy 6*, 2 (Aug. 2013), 147–159.

[33] Needleman, S. B., and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology 48* (1970), 443–453.

[34] Nicolas, F., and Rivals, E. Hardness results for the center and median string problems under the weighted and unweighted edit distances. *Journal of Discrete Algorithms 3*, 2–4 (2005), 390 – 415. Combinatorial Pattern Matching (CPM) Special Issue The 14th annual Symposium on combinatorial Pattern Matching.

[35] Pevzner, P. A. *Computational Molecular Biology - An Algorithmic Approach*. MIT Press, 2000.

[36] Rabiner, L. R. A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE* (1989), pp. 257–286.

[37] Sayers, E. W., Barrett, T., Benson, D. A., Bolton, E., et al. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research 40* (2012), D13–D25.

[38] Setubal, J. C., and Meidanis, J. *Introduction to Computational Molecular Biology*. Computer Science Series. PWS Pub., 1997.

[39] Sipser, M. *Introdução à Teoria da Computação*. Thomson Learning, 2007.

[40] Snustad, D., and Simmons, M. *Fundamentos de Genética*. Guanabara Koogan, 2008.

[41] The ENCODE Project Consortium. The ENCODE (encyclopedia of DNA elements) project. *Science 306*, 5696 (2004), 636–640.

[42] Tjandraatmadja, C., and Ferreira, C. E. O Problema da Subsequência Comum Máxima Sem Repetições. Master's thesis, IME-USP, São Paulo, Brasil, 2010.

[43] Vazirani, V. V. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[44] Wolsey, L. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998.

Apêndice A

Resultados do PLI sobre Instâncias Artificiais

A Tabela A.1 detalhas as instâncias utilizadas para traçar o comparativo entre a precisão das diferentes abordagens empregadas para tratar o PASM: a formulação de PLI, a 3−aproximação e as heurísticas. As duas primeiras abordagens são as principais contribuições do presente trabalho, enquanto as três heurísticas foram introduzidas anteriormente em [24]. Ao todo, foram geradas de maneira aleatória 225 instâncias, cujas informações a respeito de tamanho de g, quantidade de sequências em $\mathcal{T}$, tamanho das sequências em $\mathcal{T}$ e quantidade de blocos em $\mathcal{B}$ podem ser conferidos nas colunas $|g|$, Q_t, $|\mathcal{T}|$ e $Q_{\mathcal{B}}$ da Tabela A.1, respectivamente. As demais colunas da tabela apresentam os resultados obtidos por cada uma das cinco abordagens empregadas, sendo que para a formulação de PLI é ainda retratado o GAP da solução e o tempo gasto (em segundos), uma vez que não foi possível obter a solução ótima para todos os casos. Os tempos de execução da aproximação e das heurísticas não são retratados pois foram ínfimos.

Tabela A.1: Detalhes das instâncias artificiais do conjunto de testes empregado na avaliação da formulação de PLI, da 3−aproximação e das heurísticas para o PASM e resultados associados.

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g10d3t2	10	3	2	3	6	0	44	6	6	6	6
g10d3t5	10	3	5	3	14	0	3600	14	15	15	15
g10d5t2	10	3	2	5	6	0	254	6	6	6	6
g10d5t5	10	3	5	5	13	218	3600	13	13	13	13
g13d3t2	13	3	2	3	6	0	459	6	6	6	6
g13d3t6	13	3	6	3	16	23	3600	16	17	16	16
g13d5t2	13	3	2	5	6	0	1579	6	6	6	6
g13d5t6	13	3	6	5	14	326	3600	14	14	14	14
g16d3t3	16	3	3	3	9	50	3600	9	9	9	9
g16d3t8	16	3	8	3	22	100	3600	22	22	24	22
g16d5t3	16	3	3	5	9	64	3600	9	9	9	9
g16d5t8	16	3	8	5	21	133	3600	21	24	23	21
g19d3t3	19	3	3	3	8	0	3600	8	9	9	9
g19d3t9	19	3	9	3	25	314	3600	25	25	25	25
g19d5t3	19	3	3	5	9	125	3600	9	9	11	9

(continua na próxima página)

Tabela A.1 – Continuação

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g19d5t9	19	3	9	5	25	1028	3600	25	26	25	27
g22d3t11	22	3	11	3	30	1900	3600	31	31	31	31
g22d3t4	22	3	4	3	12	0	9	12	15	15	14
g22d5t11	22	3	11	5	29	996	3600	29	29	29	29
g22d5t4	22	3	4	5	11	388	3600	11	12	11	12
g25d3t12	25	3	12	3	33	1000	3600	33	33	33	33
g25d3t5	25	3	5	3	13	185	3600	13	13	13	13
g25d5t12	25	3	12	5	35	1067	3600	31	35	31	31
g25d5t5	25	3	5	5	13	377	3600	13	13	13	13
g28d3t14	28	3	14	3	37	106	3600	37	38	38	38
g28d3t5	28	3	5	3	14	75	3600	14	14	14	14
g28d5t14	28	3	14	5	38	15100	3600	36	37	37	37
g28d5t5	28	3	5	5	15	275	3600	14	14	16	14
g31d3t15	31	3	15	3	41	1226	3600	41	41	41	41
g31d3t6	31	3	6	3	15	184	3600	15	15	15	15
g31d5t15	31	3	15	5	40	344	3600	39	39	39	45
g31d5t6	31	3	6	5	17	656	3600	17	17	18	24
g34d3t17	34	3	17	3	47	683	3600	45	45	45	45
g34d3t6	34	3	6	3	15	482	3600	15	15	16	15
g34d5t17	34	3	17	5	47	inf	3600	44	44	44	44
g34d5t6	34	3	6	5	18	500	3600	18	20	20	18
g37d3t18	37	3	18	3	53	783	3600	46	49	47	54
g37d3t7	37	3	7	3	19	111	3600	19	19	20	19
g37d5t18	37	3	18	5	54	1700	3600	44	46	44	46
g37d5t7	37	3	7	5	21	200	3600	21	21	21	21
g40d3t20	40	3	20	3	55	267	3600	51	51	52	51
g40d3t8	40	3	8	3	24	300	3600	22	23	23	22
g40d5t20	40	3	20	5	59	inf	3600	50	52	53	52
g40d5t8	40	3	8	5	22	144	3600	20	22	22	20
g43d3t21	43	3	21	3	56	107	3600	55	55	55	55
g43d3t8	43	3	8	3	24	300	3600	21	21	22	24
g43d5t21	43	3	21	5	63	2000	3600	20	20	20	20
g10d10t2q3	10	3	2	10	5	0	79	5	5	6	18
g10d10t5q3	10	3	5	10	13	333	3600	13	13	13	13
g10d10t7q3	10	3	7	10	17	467	3600	18	18	17	18
g10d5t2q3	10	3	2	5	5	0	8	5	5	6	6
g10d5t5q3	10	3	5	5	13	160	3600	13	15	14	18
g10d5t7q3	10	3	7	5	17	240	3600	18	18	20	20
g10d7t2q3	10	3	2	7	6	0	161	6	6	8	6
g10d7t5q3	10	3	5	7	13	550	3600	13	13	13	13
g10d7t7q3	10	3	7	7	17	750	3600	17	20	19	21
g12d10t2q3	12	3	2	10	6	0	109	6	6	6	6
g12d10t6q3	12	3	6	10	14	1300	3600	14	17	16	21
g12d10t8q3	12	3	8	10	20	1900	3600	21	21	22	24
g12d5t2q3	12	3	2	5	6	0	128	6	6	6	6
g12d5t6q3	12	3	6	5	15	275	3600	15	16	16	21
g12d5t8q3	12	3	8	5	21	inf	3600	21	21	21	21
g12d7t2q3	12	3	2	7	5	0	88	5	6	5	6
g12d7t6q3	12	3	6	7	15	1400	3600	16	16	16	18
g12d7t8q3	12	3	8	7	21	2000	3600	22	22	22	22
g14d10t2q3	14	3	2	10	4	0	960	4	4	6	4
g14d10t7q3	14	3	7	10	18	1700	3600	19	20	20	20

(continua na próxima página)

Tabela A.1 – Continuação

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g14d10t9q3	14	3	9	10	24	inf	3600	25	25	25	25
g14d5t2q3	14	3	2	5	6	0	18	6	6	6	6
g14d5t7q3	14	3	7	5	17	467	3600	18	20	19	21
g14d5t9q3	14	3	9	5	24	inf	3600	24	24	26	24
g14d7t2q3	14	3	2	7	5	0	440	5	6	5	6
g14d7t7q3	14	3	7	7	19	1800	3600	19	21	21	21
g14d7t9q3	14	3	9	7	24	inf	3600	24	24	24	24
g16d10t11q3	16	3	11	10	28	inf	3600	28	30	28	30
g16d10t8q3	16	3	8	10	21	inf	3600	21	22	21	24
g16d5t3q3	16	3	3	5	9	50	3600	9	9	9	9
g16d5t8q3	16	3	8	5	20	inf	3600	20	20	20	20
g16d7t3q3	16	3	3	7	9	200	3600	9	9	9	9
g16d7t8q3	16	3	8	7	19	1800	3600	19	22	19	20
g18d10t3q3	18	3	3	10	9	350	3600	9	9	9	9
g18d10t9q3	18	3	9	10	24	inf	3600	24	25	25	27
g18d5t3q3	18	3	3	5	9	200	3600	8	8	9	9
g18d7t12q3	18	3	12	7	34	inf	3600	31	31	34	31
g18d7t3q3	18	3	3	7	9	200	3600	7	7	7	24
g18d7t9q3	18	3	9	7	25	inf	3600	22	22	22	22
g20d10t10q3	20	3	10	10	38	inf	3600	25	25	26	26
g20d10t4q3	20	3	4	10	11	1000	3600	11	11	11	11
g20d5t10q3	20	3	10	5	27	inf	3600	24	24	24	24
g20d5t4q3	20	3	4	5	12	500	3600	11	12	12	12
g20d7t10q3	20	3	10	7	28	inf	3600	26	26	28	30
g20d7t4q3	20	3	4	7	14	223	3600	11	12	11	24
g22d10t4q3	22	3	4	10	11	1000	3600	11	11	13	11
g22d5t11q3	22	3	11	5	28	inf	3600	27	27	33	27
g22d7t4q3	22	3	4	7	12	1100	3600	12	12	12	12
g24d10t4q3	24	3	4	10	11	1000	3600	11	14	12	24
g24d5t12q3	24	3	12	5	35	1067	3600	32	32	32	32
g24d5t4q3	24	3	4	5	12	500	3600	12	12	12	12
g24d7t12q3	24	3	12	7	32	inf	3600	32	33	32	36
g24d7t4q3	24	3	4	7	11	1000	3600	11	11	11	11
g26d10t5q3	26	3	5	10	11	inf	3600	11	12	15	12
g8d10t1q3	8	3	1	10	2	0	2	2	3	3	3
g8d10t4q3	8	3	4	10	10	150	3600	11	11	11	18
g8d10t5q3	8	3	5	10	12	200	3600	12	13	13	18
g8d5t1q3	8	3	1	5	3	0	1	3	3	3	3
g8d5t4q3	8	3	4	5	10	150	3600	10	10	10	18
g8d5t5q3	8	3	5	5	13	117	3600	13	14	14	13
g8d7t1q3	8	3	1	7	3	0	2	3	3	3	3
g8d7t4q3	8	3	4	7	10	150	3600	10	10	11	10
g8d7t5q3	8	3	5	7	13	117	3600	13	13	13	18
g10d2t2q5	10	5	2	2	10	0	5	10	25	25	24
g10d2t4q5	10	5	4	2	19	0	144	19	19	19	19
g10d5t2q5	10	5	2	5	8	0	476	8	8	8	9
g10d5t4q5	10	5	4	5	19	443	3600	19	19	19	19
g10d5t5q5	10	5	5	5	23	360	3600	23	25	24	25
g10d8t2q5	10	5	2	8	9	80	3600	9	9	9	9
g10d8t4q5	10	5	4	8	19	533	3600	19	19	19	19
g10d8t5q5	10	5	5	8	20	567	3600	20	20	20	20
g12d2t2q5	12	5	2	2	10	0	82	10	10	10	10

(continua na próxima página)

Tabela A.1 – Continuação

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g12d2t4q5	12	5	4	2	20	54	3600	20	25	25	25
g12d2t6q5	12	5	6	2	30	200	3600	30	30	30	30
g12d5t2q5	12	5	2	5	9	125	3600	9	9	9	9
g12d5t4q5	12	5	4	5	18	260	3600	18	20	20	20
g12d5t6q5	12	5	6	5	26	550	3600	26	28	28	28
g12d8t2q5	12	5	2	8	9	125	3600	9	9	10	10
g12d8t4q5	12	5	4	8	18	1700	3600	18	18	18	18
g12d8t6q5	12	5	6	8	28	inf	3600	28	28	29	30
g14d2t2q5	14	5	2	2	10	0	8	10	10	10	10
g14d2t5q5	14	5	5	2	22	340	3600	22	24	22	22
g14d2t7q5	14	5	7	2	34	580	3600	34	34	35	35
g14d5t2q5	14	5	2	5	10	150	3600	10	10	10	10
g14d5t5q5	14	5	5	5	22	214	3600	22	22	22	22
g14d5t7q5	14	5	7	5	30	2900	3600	30	32	30	35
g14d8t2q5	14	5	2	8	10	150	3600	10	10	10	10
g14d8t5q5	14	5	5	8	25	317	3600	22	22	22	22
g14d8t7q5	14	5	7	8	31	inf	3600	32	32	33	35
g16d2t3q5	16	5	3	2	14	56	3600	14	15	15	15
g16d5t3q5	16	5	3	5	15	88	3600	15	15	15	15
g16d5t6q5	16	5	6	5	27	170	3600	27	27	27	27
g16d5t8q5	16	5	8	5	35	inf	3600	35	37	42	40
g16d8t3q5	16	5	3	8	13	333	3600	13	14	14	14
g16d8t6q5	16	5	6	8	25	inf	3600	25	26	26	30
g16d8t8q5	16	5	8	8	36	inf	3600	37	37	39	40
g18d2t3q5	18	5	3	2	15	50	3600	15	15	15	15
g18d2t7q5	18	5	7	2	36	620	3600	32	32	36	36
g18d2t9q5	18	5	9	2	42	320	3600	42	42	42	42
g18d5t3q5	18	5	3	5	14	1300	3600	14	14	14	14
g18d5t7q5	18	5	7	5	33	230	3600	33	33	33	33
g18d5t9q5	18	5	9	5	42	740	3600	39	42	42	42
g18d8t3q5	18	5	3	8	14	600	3600	14	15	15	15
g18d8t7q5	18	5	7	8	31	inf	3600	31	31	31	32
g18d8t9q5	18	5	9	8	43	inf	3600	42	42	43	45
g20d2t4q5	20	5	4	2	19	111	3600	19	20	20	20
g20d2t8q5	20	5	8	2	36	inf	3600	36	36	36	36
g20d5t10q5	20	5	10	5	48	inf	3600	41	41	45	41
g20d5t4q5	20	5	4	5	20	1900	3600	20	20	20	20
g20d5t8q5	20	5	8	5	36	inf	3600	35	35	35	35
g20d8t10q5	20	5	10	8	44	inf	3600	43	43	47	43
g20d8t4q5	20	5	4	8	17	1600	3600	17	17	19	17
g20d8t8q5	20	5	8	8	34	inf	3600	34	34	38	36
g22d10t4q5	22	5	4	10	13	333	3600	11	11	13	11
g22d2t11q5	22	5	11	2	51	2450	3600	49	49	51	49
g22d2t4q5	22	5	4	2	20	186	3600	20	20	20	20
g22d2t8q5	22	5	8	2	39	160	3600	37	37	37	37
g22d5t4q5	22	5	4	5	20	900	3600	20	20	20	20
g22d5t8q5	22	5	8	5	37	inf	3600	37	37	37	37
g22d8t4q5	22	5	4	8	19	inf	3600	19	19	19	19
g22d8t8q5	22	5	8	8	37	inf	3600	36	36	38	40
g24d2t4q5	24	5	4	2	19	533	3600	19	19	19	19
g24d2t9q5	24	5	9	2	54	81	3600	42	42	42	42
g24d5t4q5	24	5	4	5	18	inf	3600	18	18	19	19

(continua na próxima página)

Tabela A.1 – Continuação

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g8d2t1q5	8	5	1	2	5	0	0	5	5	5	5
g8d2t3q5	8	5	3	2	13	0	1	13	15	15	15
g8d2t4q5	8	5	4	2	19	280	3600	19	19	19	19
g8d5t1q5	8	5	1	5	5	0	13	5	5	5	5
g8d5t3q5	8	5	3	5	14	180	3600	14	14	14	14
g8d5t4q5	8	5	4	5	18	350	3600	18	19	19	19
g8d8t1q5	8	5	1	8	3	0	3	3	3	3	3
g8d8t3q5	8	5	3	8	13	160	3600	13	13	14	14
g10d2t5q5	10	5	5	2	22	69	3600	22	22	22	22
g10d5t5q5	10	5	5	5	22	340	3600	22	24	24	22
g16d10t3q3	16	3	3	10	8	129	3600	8	9	8	9
g16d2t6q5	16	5	6	2	27	17	3600	28	29	29	29
g16d2t8q5	16	5	8	2	37	0	257	37	37	37	39
g16d5t11q3	16	3	11	5	29	867	3600	29	29	29	29
g16d7t11q3	16	3	11	7	28	460	3600	27	28	27	28
g18d10t12q3	18	3	12	10	33	inf	3600	32	32	35	32
g18d5t12q3	18	3	2	5	33	450	3600	33	34	34	34
g18d5t9q3	18	3	9	5	24	140	3600	24	24	25	25
g20d10t14q3	20	3	14	10	35	inf	3600	33	33	34	36
g20d2t10q5	20	5	10	2	44	inf	3600	44	44	44	44
g20d3t10q5	20	5	10	3	47	inf	3600	45	45	45	45
g20d3t4q5	20	5	4	3	19	90	3600	19	19	19	19
g20d5t10q5	20	5	10	5	44	inf	3600	44	45	45	50
g20d5t14q3	20	3	14	5	39	inf	3600	38	38	38	38
g20d5t4q5	20	5	4	5	17	183	3600	17	19	19	19
g20d7t14q3	20	3	14	7	38	inf	3600	36	39	36	36
g22d10t11q3	22	3	11	10	29	480	3600	28	28	28	28
g22d10t15q3	22	3	15	10	41	inf	3600	38	39	40	38
g22d5t11q5	22	5	11	5	28	inf	3600	27	27	33	27
g22d7t15q3	22	3	15	7	38	1167	3600	38	38	39	38
g22d8t11q5	22	5	11	8	59	inf	3600	49	51	49	55
g24d10t12q3	24	3	12	10	33	inf	3600	30	33	30	36
g24d10t16q3	24	3	16	10	45	inf	3600	41	41	41	41
g24d2t12q5	24	5	12	2	51	70	3600	51	51	51	51
g24d5t12q5	24	5	12	5	55	inf	3600	54	55	55	55
g24d5t16q3	24	3	16	5	52	inf	3600	42	42	42	42
g24d5t9q5	24	5	9	5	40	700	3600	40	40	42	42
g24d7t16q3	24	3	16	7	45	650	3600	42	42	43	48
g24d8t12q5	24	5	12	8	56	inf	3600	53	56	52	54
g24d8t4q5	24	5	4	8	18	inf	3600	18	18	18	18
g24d8t9q5	24	5	9	8	42	inf	3600	39	42	42	45
g26d10t13q3	26	3	13	10	34	inf	3600	32	32	33	39
g26d10t18q3	26	3	18	10	51	inf	3600	47	49	49	49
g26d5t13q3	26	3	13	5	40	inf	3600	36	36	37	39
g26d5t18q3	26	3	18	5	50	108	3600	46	50	49	47
g30d3t6q5	30	5	6	3	29	93	3600	29	30	30	30
g30d5t15q5	30	5	15	5	67	inf	3600	65	65	70	66
g30d5t6q5	30	5	6	5	27	170	3600	27	27	27	27
g40d3t8q5	40	5	8	3	38	660	3600	38	38	38	38
g40d5t20q5	40	5	20	5	96	284	3600	86	86	89	100
g40d5t8q5	40	5	8	5	38	inf	3600	37	37	37	38
g50d3t10q5	50	5	10	3	47	inf	3600	47	47	47	47

(continua na próxima página)

Tabela A.1 – Continuação

Instância	$\|g\|$	Q_t	$\|\mathcal{T}\|$	$Q_{\mathcal{B}}$	PLI			Aprox	H1	H2	H3
					solução	gap	tempo (s)				
g50d5t10q5	50	5	10	5	45	inf	3600	44	44	47	44
g8d8t4q5	8	5	4	8	18	500	3600	18	18	20	20
Total					5487			5253	5390	5452	5587

Apêndice B

Resultados da 3-Aproximação no Problema da Identificação de Genes

Este apêndice traz os dados completos acerca das 240 instâncias que constituem o conjunto de testes utilizados na determinação da acurácia da aproximação na tarefa de identificação de genes. A nomenclatura das colunas é a mesma utilizada em [24], para facilitar a comparação. Cada instância é distinguida pelo nome do gene presente na sua sequência de referência, indicado na coluna GENE. O tamanho da sequência de DNA aparece na coluna T_{DNA} e a coluna T_c exibe o tamanho médio das sequências de cDNA que constituem o conjunto T das instâncias. A quantidade dessas sequências, ou seja, $|T|$, pode ser encontrado na coluna Q_c. A quantidade de éxons corretos é mostrada na coluna Q_{ea}, e a de possíveis éxons passados como entrada na coluna Q_{eg}. As colunas APROX e T(S) encerram as informações a respeito de cada instância com o valor da solução aproximada retornada pela 3-aproximação para o PASM proposta e o tempo de execução em segundos, respectivamente. Por fim, as seis colunas finais indicam os valores alcançados pela solução em cada uma das medidas de precisão de genes discutidas na Seção 7.3.1: sensibilidade (Sn_n), especificidade (Sp_n) correlação aproximada (CA), no nível de nucleotídeos, e sensibilidade (Sn_e), especificidade (Sp_e) e média aritmética dos valores de Sn_e e Sp_e (Av_e), no nível de éxons.

Tabela B.1: Dados das instâncias do conjunto de testes empregado na avaliação da 3-aproximação na tarefa de identificação de genes.

GENE	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	APROX	T(S)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
a1bg	8694	1546	5	8	40	2510	2.89	0.8	0.87	0.8	0.75	0.75	0.75
aff4	90284	3488	5	23	99	1237	17.97	0.98	0.99	0.99	0.95	0.83	0.89
alg10b	14972	1422	1	9	27	617	0.16	0.27	0.43	0.29	0.67	0.22	0.44
ankrd10	38530	1208	5	6	45	2461	1.90	0.62	0.67	0.63	0.67	0.67	0.67
ankrd43	5457	1729	4	1	2	1881	1.31	1	1	1	1	1	1
arhgdig	4398	700	5	6	21	574	0.53	1	1	1	1	1	1
ascc2	51655	2253	5	19	80	1384	7.17	0.98	0.97	0.98	0.79	0.79	0.79
ascl2	4455	733	5	1	8	1101	0.47	1	1	1	1	1	1
asz1	66302	1428	5	15	84	979	3.64	0.89	0.9	0.9	0.92	0.8	0.86
bad	16877	561	4	3	47	541	0.57	1	0.98	0.99	0.67	0.67	0.67
bet1	14691	355	5	6	26	465	0.19	0.56	0.62	0.58	0.75	0.5	0.62
bgn	16594	1253	5	7	80	1230	3.96	1	0.99	0.99	0.86	0.86	0.86
c17orf50	6183	555	2	5	16	291	0.10	1	0.96	0.98	1	0.6	0.8

(continua na próxima página)

Tabela B.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	Aprox	T(s)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
c20orf152	64094	1868	3	13	60	1060	1.94	1	0.99	0.99	0.92	0.85	0.88
c21orf45	12847	682	5	5	44	764	1.21	0.88	0.89	0.88	0.8	0.8	0.8
c21orf59	12572	1019	4	7	46	1136	1.78	1	1	1	1	1	1
c21orf63	104946	1321	5	10	172	1032	9.15	0.94	0.96	0.95	0.88	0.7	0.79
c7orf63	67164	2826	1	25	69	560	0.97	0.78	0.78	0.77	0.68	0.6	0.64
c7orf68	4589	636	2	1	13	929	0.10	1	1	1	1	1	1
calcr	151952	1596	5	16	135	2155	7.71	0.84	0.8	0.81	0.83	0.62	0.73
capza2	58751	847	5	10	41	252	0.89	1	1	1	1	1	1
cars2	66705	1881	4	14	118	2571	6.32	0.88	0.9	0.89	0.8	0.86	0.83
cav1	38392	537	5	3	42	150	0.56	1	1	1	1	1	1
ccdc157	22192	1975	3	12	82	1696	4.14	0.72	0.91	0.8	0.6	0.5	0.55
ccdc88b	19312	4738	4	27	68	5513	16.62	0.99	0.99	0.99	0.96	0.96	0.96
ccdc93	100551	2003	5	25	176	1534	14.62	0.98	0.97	0.97	0.92	0.88	0.9
ccl5	10883	259	5	3	9	240	0.06	1	1	1	1	1	1
cdh8	384802	2349	5	11	380	1813	59.11	1	1	1	1	1	1
cdx2	9040	913	5	3	37	696	1.49	1	1	1	1	1	1
cftr	190703	4446	5	27	216	4116	83.98	0.91	0.93	0.92	0.85	0.85	0.85
chmp2a	5554	667	5	5	12	420	0.50	1	1	1	1	1	1
cldn12	14473	735	5	1	29	504	0.90	1	1	1	1	1	1
cpne8	255419	1730	5	19	177	1160	11.67	0.9	0.92	0.91	0.75	0.79	0.77
csf2	4375	432	4	4	9	384	0.13	1	0.75	0.86	1	0.75	0.88
ctgf	5203	1045	5	5	11	648	0.98	0.94	0.96	0.94	0.8	0.8	0.8
ctsd	13238	1224	5	9	46	1176	2.19	1	1	1	1	1	1
ddx18	19699	2009	5	14	55	1214	5.09	1	1	1	1	1	1
ddx43	25005	1985	4	16	43	1751	2.87	1	1	1	1	1	1
decr2	12630	887	5	8	44	1569	1.25	1	1	1	1	1	1
dnajc4	6001	717	5	6	28	925	0.67	1	1	1	0.83	0.83	0.83
dppa5	3167	357	1	3	19	85	0.03	1	1	1	1	1	1
drg1	36634	1104	5	9	32	312	1.07	1	1	1	1	1	1
dusp18	7834	576	5	1	16	365	0.52	1	1	1	1	1	1
eef1a1	7283	1389	5	7	15	513	1.66	1	1	1	1	1	1
esrra	13167	1318	5	6	39	644	2.58	1	1	1	0.83	0.83	0.83
ext1	314457	2241	5	11	283	338	33.90	1	1	1	1	1	1
fam71f1	18355	1032	5	7	20	1914	0.87	0.53	0.66	0.58	0.29	0.29	0.29
fkbp2	5195	466	5	5	18	366	0.30	1	1	1	1	1	1
flt3	99319	3112	5	24	114	2595	15.00	0.98	0.99	0.98	0.88	0.88	0.88
frmd6	243591	1861	5	13	209	1188	16.42	1	0.99	0.99	0.92	0.92	0.92
frs3	11717	1483	5	5	29	781	2.49	1	1	1	1	1	1
gabra3	286198	1479	5	13	320	971	32.77	0.94	0.95	0.95	0.89	0.62	0.75
gabrq	17189	1926	5	9	26	1613	4.15	1	1	1	1	1	1
gal3st1	12253	1417	5	2	27	1483	2.55	1	0.99	1	0.5	0.5	0.5
gas2l2	10361	2614	5	6	24	2518	8.24	1	1	1	1	1	1
gdf9	5600	1348	5	2	5	1315	1.42	1	1	1	1	1	1
gng11	6811	222	5	2	5	74	0.05	1	1	1	1	1	1
gng2	111469	229	5	6	91	476	0.42	0.4	0.52	0.46	0.5	0.17	0.33
gngt1	6666	225	5	2	5	83	0.05	1	1	1	1	1	1
gpr137	5633	1190	5	9	40	617	2.04	0.82	0.88	0.81	0.86	0.67	0.76
gsx1	3310	828	5	2	10	757	0.56	1	1	1	1	1	1
hba1	2842	429	4	3	10	329	0.15	1	1	1	1	1	1
hba2	2864	429	3	3	14	215	0.11	1	1	1	1	1	1
hbb	3606	442	4	3	6	278	0.10	1	1	1	1	1	1
hbe1	3794	470	5	3	6	483	0.21	1	1	1	1	1	1
hbg1	3586	463	5	3	4	461	0.14	1	1	1	1	1	1

(continua na próxima página)

Tabela B.1 – Continuação

GENE	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	APROX	T(S)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
hbg2	3591	444	2	3	4	93	0.03	1	1	1	1	1	1
hbq1	2844	433	2	3	6	211	0.04	1	1	1	1	1	1
hbz	3651	436	5	3	11	451	0.22	1	1	1	1	1	1
hormad2	98610	812	3	13	85	760	1.13	0.78	0.8	0.79	0.7	0.54	0.62
hoxa2	4422	1122	5	2	4	297	0.94	1	1	1	1	1	1
hoxa4	4274	933	5	2	6	660	0.78	1	1	1	1	1	1
hoxa5	4292	853	5	2	8	1366	0.92	1	0.79	0.86	0.5	0.5	0.5
hoxa6	4253	699	4	2	6	238	0.35	1	1	1	1	1	1
hoxa7	4962	696	4	2	15	369	0.37	1	1	1	1	1	1
hunk	132750	2146	5	11	127	1467	10.64	0.97	0.98	0.98	0.91	0.91	0.91
il13	4937	397	5	4	15	397	0.19	0.9	1	0.95	0.75	0.75	0.75
il3	4550	459	1	5	13	4	0.02	1	1	1	1	1	1
il5	4079	403	5	4	8	333	0.15	1	1	1	1	1	1
inpp5j	13723	1074	1	8	41	480	0.34	0.48	0.97	0.68	0.54	0.88	0.71
ins	3431	330	5	2	4	230	0.09	1	1	1	1	1	1
insig2	23548	678	5	7	33	512	0.67	0.82	0.84	0.82	0.8	0.57	0.69
irf1	11165	1020	5	9	24	963	1.06	1	1	1	1	1	1
itfg3	33319	1757	5	11	94	1994	6.70	1	1	1	1	1	1
kcnj15	47085	1162	5	1	29	712	2.41	1	1	1	1	1	1
kcnj6	293912	1399	4	6	332	945	23.14	0.98	0.98	0.98	0.67	0.33	0.5
kcnk4	10711	1426	5	6	36	2209	2.91	1	1	1	1	1	1
kcnq5	575746	2530	5	20	606	3334	155.60	0.96	0.96	0.96	0.79	0.55	0.67
khdc1	23871	712	2	7	34	629	0.24	0.84	0.86	0.85	0.33	0.14	0.24
kif3a	46943	1887	5	17	59	1795	4.04	1	1	1	0.94	0.94	0.94
lace1	230161	1437	5	19	240	998	18.73	0.93	0.94	0.93	0.85	0.58	0.71
leap2	3225	224	5	3	8	181	0.06	1	1	1	1	1	1
lep	18352	504	5	2	15	305	0.36	1	1	1	1	1	1
lif	8355	583	5	3	17	472	0.38	1	1	1	1	1	1
lrrc4	5879	1928	5	1	2	655	2.59	1	1	1	1	1	1
lyzl6	6986	447	4	4	17	248	0.16	1	1	1	1	1	1
magea1	6595	1230	1	4	23	76	0.22	0.99	0.71	0.81	0	0	0
magea12	5880	945	1	1	14	9	0.15	1	1	1	1	1	1
map3k1	83080	4420	5	20	104	3347	26.94	1	1	1	0.9	0.9	0.9
march11	114424	1320	4	4	149	1742	5.78	1	1	1	1	1	1
mdfi	17788	819	5	4	53	934	1.24	1	0.99	0.99	0.75	0.75	0.75
mettl2b	28196	1205	5	9	23	1032	1.31	1	1	1	1	1	1
mier3	34526	1710	4	12	36	999	2.21	0.99	1	1	0.85	0.92	0.88
mmp26	6236	822	2	6	48	233	0.38	1	1	1	1	1	1
morc2	43588	2955	5	24	57	1717	8.63	0.99	0.99	0.99	0.96	0.92	0.94
moxd1	107471	1843	5	12	153	1004	12.02	1	1	1	1	1	1
mrpl23	11338	498	5	5	39	574	0.51	1	1	1	1	1	1
mzf1	13659	2283	5	5	29	2109	5.91	1	1	1	1	1	1
ndst3	226290	2622	5	20	237	1822	36.64	0.88	0.9	0.89	0.69	0.45	0.57
nfxl1	69377	2749	4	21	82	1920	7.28	0.85	0.88	0.86	0.68	0.71	0.7
nipal1	22294	1282	5	7	81	854	3.61	1	1	1	1	0.86	0.93
nme4	5563	1003	4	5	16	2156	0.41	1	1	1	1	1	1
nr2e1	24799	1158	5	9	41	483	1.84	0.98	0.96	0.97	0.89	0.89	0.89
nsdhl	40397	1075	5	8	41	901	1.54	0.9	0.96	0.93	0.86	0.75	0.8
of51f1	2939	892	3	1	8	512	0.48	1	0.98	0.98	0	0	0
ooep	3238	401	3	3	17	430	0.13	0.93	1	0.96	0.67	0.67	0.67
or51a2	2942	1099	5	1	9	1836	1.59	1	1	1	1	1	1
or51a4	2942	1099	5	1	10	1818	1.67	1	1	1	1	1	1
or51a7	2939	951	4	1	8	426	0.97	1	1	1	1	1	1

(continua na próxima página)

Tabela B.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	Aprox	T(s)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
or51b2	3055	1136	4	1	5	1429	0.71	1	1	1	1	1	1
or51b4	2933	935	3	1	2	409	0.23	1	1	1	1	1	1
or51b5	2939	939	1	2	4	139	0.05	0.85	0.98	0.87	0	0	0
or51b6	2939	945	3	1	3	317	0.31	1	1	1	1	1	1
or51f2	3029	980	4	1	5	600	0.82	1	1	1	1	1	1
or51g1	2966	955	5	1	12	601	1.43	1	1	1	1	1	1
or51g2	2945	956	4	1	4	559	0.67	1	1	1	1	1	1
or51i1	2945	945	4	1	5	429	0.62	1	1	1	1	1	1
or51i2	2939	940	5	1	4	439	0.86	1	1	1	1	1	1
or51l1	2948	993	4	1	2	557	0.43	1	1	1	1	1	1
or51m1	2981	972	4	1	7	721	0.84	1	1	1	1	1	1
or51q1	2954	954	5	1	3	649	0.80	1	1	1	1	1	1
or51s1	2972	955	3	1	7	519	0.68	1	1	1	1	1	1
or51t1	3065	1060	5	1	4	1052	1.08	0.94	1	0.96	0	0	0
or51v1	2966	1003	3	1	10	535	0.73	0.98	1	0.99	0	0	0
or52a1	2939	971	5	3	13	895	1.75	1	0.99	0.99	1	0.33	0.67
or52a4	2915	936	1	3	11	214	0.10	0.91	0.94	0.89	0	0	0
or52a5	2951	952	5	1	4	1590	0.72	0.79	1	0.85	0	0	0
or52b6	3008	972	3	1	6	879	0.63	0.94	1	0.95	0	0	0
or52d1	2957	956	4	1	4	440	0.63	1	1	1	1	1	1
or52e2	2978	1001	5	1	5	914	1.10	1	1	1	1	1	1
or52h1	2963	961	5	1	11	573	1.65	1	1	1	1	1	1
or52j3	2936	937	5	1	3	637	0.75	1	1	1	1	1	1
or52r1	3185	1030	4	1	3	955	0.58	0.8	1	0.85	0	0	0
osbp2	215019	2709	5	14	216	1797	28.93	1	1	1	0.93	0.93	0.93
osm	6023	763	5	3	33	1110	0.97	1	1	1	1	1	1
ostm1	35329	990	4	8	40	968	1.15	0.9	0.92	0.91	0.83	0.62	0.73
pan3	123056	2297	5	21	126	2698	12.92	0.96	0.97	0.96	0.82	0.67	0.75
pdia2	6092	1546	3	11	37	1068	1.28	1	0.99	0.99	0.91	0.91	0.91
pdk4	15117	1235	5	11	20	582	1.26	1	1	1	1	1	1
pdx1	8284	972	5	2	22	1345	1.06	1	1	1	1	1	1
pes1	17283	1758	5	15	63	999	4.25	1	1	1	1	1	1
pex12	5843	1080	5	3	15	725	1.18	1	0.95	0.97	0.67	0.67	0.67
pgc	12670	1174	5	9	25	981	1.18	1	1	1	1	1	1
pla2g3	7677	1522	5	7	23	1355	2.39	1	1	1	1	1	1
plcb3	17907	3682	5	31	67	3337	13.92	1	1	1	0.97	0.97	0.97
pnma3	6062	1397	3	1	29	561	2.04	1	1	1	1	1	1
pnma5	5394	1584	2	1	24	1164	1.05	1	1	1	1	1	1
polr3k	8626	341	5	3	13	269	0.32	1	1	1	1	1	1
pon1	28216	1068	5	9	50	641	1.69	1	1	1	1	1	1
pon3	38504	1065	5	10	42	735	1.60	0.96	0.96	0.96	0.89	0.8	0.84
ppp1r14b	4463	432	5	4	10	291	0.18	1	1	1	1	1	1
ppp1r3a	44201	3331	5	4	39	2555	11.73	1	1	1	1	1	1
ppp1r9a	390779	3706	5	19	323	3508	79.39	1	0.83	0.91	1	0.79	0.89
prhoxnb	12532	525	3	2	37	415	0.28	1	1	1	1	1	1
prickle4	8611	1297	4	7	41	1507	1.80	1	0.99	0.99	0.83	0.71	0.77
prss12	73506	2407	5	11	59	2402	7.45	0.88	1	0.94	0.85	1	0.92
rasl10b	13862	612	5	3	11	124	0.32	1	1	1	1	1	1
rbm28	35527	2629	5	19	48	3394	6.24	1	1	1	1	1	1
rnf152	80001	612	5	1	97	299	1.90	1	1	1	1	1	1
rps5	9536	658	5	5	23	457	0.55	1	1	1	1	1	1
samd9	20492	4766	3	1	12	1550	8.46	1	1	1	1	1	1
samd9l	20313	4834	5	1	25	4688	22.09	1	1	1	1	1	1

(continua na próxima página)

Tabela B.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	Aprox	T(s)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
sec14l3	14799	1203	5	12	29	449	1.41	1	1	1	1	1	1
sec14l4	18783	1197	4	12	31	846	1.02	1	1	1	1	1	1
selm	4789	441	3	5	21	327	0.10	1	1	1	1	1	1
shroom1	5991	2669	5	7	22	4468	6.88	1	0.99	0.99	0.86	0.86	0.86
skap2	199655	1078	5	14	216	560	9.91	0.94	0.94	0.94	0.92	0.79	0.85
slc22a11	17902	1624	2	10	47	364	1.06	1	1	1	1	1	1
slc22a4	51755	1757	5	10	95	1664	5.92	1	1	1	1	1	1
slc22a5	27906	1675	5	10	57	1189	4.11	1	1	1	1	1	1
slc25a13	203928	2053	5	19	239	985	25.24	0.97	0.98	0.98	0.89	0.84	0.87
slc27a5	15733	2072	4	10	19	1409	2.49	1	1	1	1	1	1
slc35e4	14070	1053	5	2	36	651	1.64	1	1	1	1	1	1
slc4a3	16411	3679	5	22	65	2131	15.32	0.98	1	0.99	0.91	0.91	0.91
slfn12l	64955	1792	3	2	57	2754	3.00	0.58	0.95	0.76	0	0	0
slfn13	15742	2718	3	4	39	1542	4.75	1	1	1	1	1	1
slfn14	11967	2732	4	4	17	1548	4.73	1	1	1	1	1	1
snd1	442458	2825	5	27	622	1802	188.97	0.92	0.93	0.92	0.92	0.81	0.87
snrnp25	5841	512	5	5	15	825	0.28	1	1	1	1	1	1
snx19	42617	2929	5	10	80	2563	14.66	0.97	0.96	0.96	0.82	0.9	0.86
spp2	28431	708	5	10	53	1208	0.94	0.8	0.85	0.82	0.43	0.3	0.36
steap1	12453	1020	5	11	35	975	1.55	0.79	0.81	0.78	0.75	0.27	0.51
stip1	20434	1632	5	14	38	710	3.07	0.99	1	1	0.93	0.93	0.93
syt8	5077	1194	4	9	27	1058	1.05	0.97	0.93	0.93	0.67	0.67	0.67
taf15	39751	1617	5	15	62	1222	3.59	0.97	0.99	0.98	0.81	0.87	0.84
taf4b	167241	2010	5	19	172	4332	14.29	0.94	0.95	0.95	0.93	0.74	0.84
tbc1d10a	36916	1547	5	9	83	840	3.97	1	1	1	1	1	1
tcn2	21887	1291	5	9	23	1075	1.52	1	1	1	1	1	1
tec	136015	1844	5	18	113	1344	7.75	0.99	0.97	0.98	0.88	0.83	0.86
tfeb	53083	1642	5	13	86	1342	5.13	1	0.9	0.95	0.88	0.54	0.71
tfpi2	6321	700	5	5	13	726	0.49	1	1	1	1	1	1
tiam1	442555	4210	5	25	323	4393	93.96	1	1	1	1	1	1
tmem8	13175	2326	5	16	56	2361	6.74	0.96	0.96	0.95	0.92	0.75	0.84
tnni2	4006	549	5	6	18	237	0.29	0.97	0.97	0.97	0.71	0.83	0.77
tomm6	4454	225	3	3	31	201	0.14	0.57	0.57	0.55	0.5	0.33	0.42
trex2	3768	711	4	1	22	344	0.57	1	1	1	1	1	1
trim28	8247	2500	3	17	34	1012	2.34	1	1	1	1	1	1
trpm8	104124	3316	5	26	130	2291	19.68	0.98	0.99	0.98	0.96	0.88	0.92
txk	69864	1578	5	15	100	1318	5.57	1	1	1	1	1	1
ube2m	5265	655	5	7	21	878	0.64	0.8	0.83	0.79	0.83	0.71	0.77
ubqln3	4624	1733	4	1	9	2063	1.96	1	1	1	1	1	1
ubqlnl	4334	1836	2	2	5	1433	0.66	1	1	1	1	0.5	0.75
uqcrq	4218	249	5	2	16	178	0.10	1	1	1	1	1	1
urb1	83983	6915	5	39	102	6871	48.86	1	0.99	1	0.97	0.97	0.97
usp49	99717	2070	5	7	52	1458	4.78	1	0.96	0.98	1	0.57	0.79
vegfb	5994	624	1	8	37	147	0.09	0.83	0.9	0.85	0.83	0.62	0.73
wnt2	48659	1083	5	5	83	558	3.55	0.92	0.95	0.94	0.8	0.8	0.8
zbtb45	8025	1440	4	2	26	1784	1.59	1	1	1	1	1	1
zfp92	5306	1251	3	4	6	1212	0.46	1	0.99	0.99	0.75	0.75	0.75
zmat5	38025	510	5	5	32	410	0.50	1	1	1	1	1	1
znf132	9409	2760	2	1	21	2134	1.68	0.83	1	0.89	0	0	0
znf135	12165	1974	1	4	9	294	0.27	1	1	1	1	1	1
znf185	61026	2269	2	21	133	1216	3.79	0.83	0.85	0.83	0.73	0.76	0.74
znf256	8877	1834	2	3	18	615	0.94	1	1	1	1	1	1
znf275	20772	1459	5	3	26	2479	2.39	1	1	1	0.67	0.67	0.67

(continua na próxima página)

Tabela B.1 – Continuação

GENE	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	APROX	T(S)	Sn_n	Sp_n	CA	Sn_e	Sp_e	Av_e
znf324	8303	2066	5	4	25	3622	3.74	1	0.98	0.99	0.67	0.5	0.58
znf324b	8229	2066	5	6	37	3719	7.50	1	0.99	0.99	0.67	0.33	0.5
znf329	26454	1630	5	1	25	1353	3.04	0.99	1	1	0	0	0
znf418	15489	2005	2	7	20	1549	1.46	0.86	0.96	0.9	0.33	0.14	0.24
znf446	6803	1572	1	7	38	547	0.43	0.86	0.83	0.81	0.67	0.57	0.62
znf497	10396	1854	1	4	28	101	0.48	1	0.82	0.89	0	0	0
znf551	9823	1954	2	3	20	1125	2.31	0.92	1	0.95	0.33	0.33	0.33
znf584	11632	1860	2	5	22	1258	0.69	0.95	0.71	0.8	0.5	0.4	0.45
znf606	28275	3650	5	7	59	7292	15.13	0.99	0.99	0.99	0.83	0.71	0.77
znf622	16267	1357	5	6	21	1435	1.76	1	1	1	1	1	1
znf8	18937	1723	2	4	36	375	1.04	1	1	1	1	1	1
znf814	21696	2791	2	2	30	2427	3.49	0.89	1	0.94	0	0	0
zscan1	22566	537	1	4	96	95	0.42	0.27	0.65	0.44	0	0	0
zscan22	17328	1491	1	2	27	321	0.54	1	1	1	1	1	1
zscan4	12218	1360	3	3	6	1344	0.52	1	1	1	1	1	1