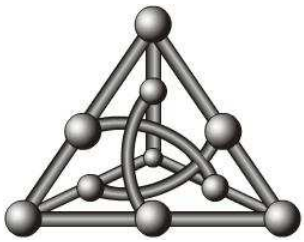
Identificação de Genes e o Problema do Alinhamento *Spliced* Múltiplo

Rodrigo Mitsuo Kishi

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, o autor recebeu apoio financeiro da FUNDECT –

— Campo Grande, dezembro de 2010 —

Identificação de Genes e o Problema do Alinhamento *Spliced* Múltiplo

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Rodrigo Mitsuo Kishi e aprovada pela Banca Examinadora.

Campo Grande, 14 de dezembro de 2010.

Banca Examinadora:

- Prof. Dr. Said Sadique Adi (Orientador) (FACOM/UFMS)
- Profª. Drª. Maria Emília Machado Telles Walter (CIC/UnB)
- Prof. Dr. Nalvo Franco de Almeida Jr. (FACOM/UFMS)
- Profª. Drª. Luciana Montera Cheung (FACOM/UFMS)

Dedicatória

Dedico essa dissertação à memória de minha avó, Julieta Parreira Sandoval.

Agradecimentos

Agradeço aos meus pais, Ester, Orlando e à minha avó, "Dona Zola", pelos ensinamentos fundamentais sobre caráter, respeito e educação. Além disso, agradeço a eles pela segurança que tenho desde que saí de casa para estudar. Agradeço também à minha namorada, Juliana, pelo apoio e compreensão.

Ao meu orientador, o Professor Said Sadique, com quem aprendi a maior parte do que sei sobre Bioinformática e que sempre esteve presente quando precisei, se desdobrando para conseguir tempo e me ajudar. Agradeço também ao Professor Fábio Viduani, com quem aprendi muito sobre pesquisa, durante a minha iniciação científica e que me assistenciou durante o estágio de docência. Agradeço aos professores Marcelo Henriques, Luciana Montera, Maria Emília, Nalvo Almeida e Irineu Sotoma pela ajuda com o meu trabalho. Agradeço ainda à Isabel e à Beth pelos vários documentos e instruções sobre burocracia que elas proveram.

Aos meus colegas de mestrado, Ronaldo e Lucas, que me acompanham desde a graduação e que compartilharam comigo os bons e maus momentos do mestrado.

Por fim, agradeço à Fundect pelo apoio financeiro.

Resumo

A identificação de genes em sequências de DNA de organismos eucariotos ainda pode ser considerado um problema em aberto na Bioinformática. Na busca por soluções deste problema, em muitos casos recorre-se à comparação de sequências. Várias combinações de sequências vêm sendo utilizadas pelas ferramentas de identificação de genes e neste trabalho propomos a comparação de diversas sequências de cDNA com uma sequência de DNA. Essa proposta foi abordada através da formulação e estudo de um problema de otimização combinatória denominado Problema do Alinhamento *Spliced* Múltiplo. Nessa dissertação descrevemos esse problema, demonstramos que ele é NP-completo para a distância de Levenshtein e propomos quatro heurísticas para resolvê-lo. Com base nessas heurísticas, desenvolvemos quatro ferramentas de identificação de genes por comparação de uma sequência de DNA com várias sequências de cDNA. Essas ferramentas foram avaliadas em instâncias de teste que construímos a partir de dados reais do genoma humano e os seus resultados mostraram-se melhores que os de outras ferramentas de identificação de genes disponíveis na literatura.

Palavras-chave: ferramentas de identificação de genes, comparação de sequências, problema do alinhamento *spliced* múltiplo.

Abstract

The gene prediction in DNA sequences of eukariotic organisms is still an open problem in Bioinformatics. The sequence comparison based approach is commonly used in the search of solutions for this problem. Several different combinations of sequences are being used by gene recognition tools and in this work we propose the comparison of many cDNA sequences with a DNA sequence. This proposal was addressed by the formulation and study of a combinatorial optimization problem, called Multiple Spliced Alignment Problem. In this work we describe this problem, show that it is NP-complete under the Levenshtein distance and propose four heuristics to solve it. Based on these heuristics, we developed four gene recognition tools based on the comparison of a DNA sequence with many cDNAs. These tools were evaluated with instances built from real human genome data and their results were better when compared to other gene recognition tools available in literature.

Keywords: gene recognition tools, sequence comparison, multiple spliced alignment problem.

Sumário

Lista de Figuras

Lista de Algoritmos

Lista de Tabelas

Capítulo 1

Introdução

As informações sobre as características de todos os seres vivos conhecidos são armazenadas nas suas moléculas de DNA. Mais especificamente, grande parte dessas informações estão codificadas em trechos da sequência de DNA que contêm a "receita" para a produção das proteínas que compõem qualquer organismo vivo. Esses trechos são chamados de genes. À tarefa de encontrar os limites dos genes dentro de uma sequência de DNA está associado um problema conhecido na literatura por Problema da Identificação de Genes. Esse problema corresponde ao tema principal deste trabalho.

Graças aos avanços obtidos nas técnicas de sequenciamento, atualmente há uma grande quantidade de sequências de DNA disponíveis para estudo. Por outro lado, as técnicas atuais para a identificação de genes não são suficientemente rápidas e precisas para tratar todos os dados disponíveis. As abordagens mais comuns para a identificação de genes são a busca estatística por sinais em uma sequência e a comparação entre sequências. Em ambos os casos, faz-se uso da teoria da evolução e do princípio de conservação das bases, que afirmam que genes de organismos evolutivamente relacionados tendem a apresentar uma certa semelhança em sua composição.

Em geral, as ferramentas baseadas na comparação entre sequências são mais precisas do que as que realizam uma busca estatística por sinais. Esse tipo de comparação pode ser realizada utilizando várias combinações de sequências: duas sequências de DNA, uma sequência de DNA e uma sequência de cDNA, uma sequência de DNA e uma sequência EST, várias sequências de DNA, etc. Um trabalho importante e bem conhecido nessa área é o de Gelfand *et al.* [19], onde é apresentada uma ferramenta de identificação de genes baseada na comparação de uma sequência de DNA com uma sequência de cDNA. Essa ferramenta denomina-se `PROCRUSTES` e foi desenvolvida com base em um algoritmo que soluciona um problema computacional, o Problema do Alinhamento *Spliced*, proposto pelos autores para modelar o Problema da Identificação de Genes.

As ferramentas de identificação de genes disponíveis na literatura baseiam-se na presença de evidências de um gene, sendo elas intrínsecas ou extrínsecas à sequência alvo do estudo. No caso da ferramenta `PROCRUSTES`, as evidências são encontradas pela comparação da sequência de DNA com a sequência de cDNA. A idéia que tivemos para melhorar os resultados de Gelfand *et al.* foi utilizar não apenas uma, mas várias

sequências de cDNA na comparação com uma sequência de DNA não anotada. Pretendíamos com isso, aumentar a quantidade de evidências sobre a presença de um gene na sequência alvo do estudo e consequentemente a qualidade das predições de genes.

1.1 Objetivos

Nosso principal objetivo neste trabalho é desenvolver novas ferramentas de identificação de genes baseadas na comparação entre sequências. Mais especificamente, objetivamos implementar novas ferramentas de identificação de genes através da comparação de uma sequência de DNA com várias sequências de cDNA. Na próxima seção descreveremos as contribuições resultantes deste trabalho.

1.2 Contribuições

As principais contribuições deste trabalho foram:

- Formulação de um novo problema matemático para modelar a tarefa de identificação de genes por comparação de uma sequência de DNA com várias sequências de cDNA. Esse problema é denominado Problema do Alinhamento *Spliced* Múltiplo;
- Demonstração da NP-completude do Problema do Alinhamento *Spliced* Múltiplo;
- Proposição de quatro heurísticas para o Problema do Alinhamento *Spliced* Múltiplo;
- Desenvolvimento de quatro ferramentas para o Problema da Identificação de Genes por comparação de uma sequência de DNA com várias sequências de cDNA, baseadas nas heurísticas para o Problema do Alinhamento *Spliced* Múltiplo;
- Criação de um conjunto de testes com instâncias baseadas em dados reais para o Problema da Identificação de Genes por comparação de uma sequência de DNA com várias sequências de cDNA;
- Avaliação experimental das ferramentas, utilizando o conjunto de testes proposto para o Problema da Identificação de Genes por comparação de uma sequência de DNA com várias sequências de cDNA.

1.3 Organização do texto

O texto dessa dissertação está dividido em seis partes principais, além desta introdução. No Capítulo 2 apresentamos os conceitos necessários à compreensão do restante do texto, subdivididos em conceitos computacionais, biológicos e de Bioinformática. No Capítulo 3 definimos o Problema da Identificação de Genes e realizamos

uma revisão de alguns dos principais métodos e ferramentas para identificação de genes. Na primeira parte do Capítulo 4 descrevemos um problema matemático que serve como modelagem para o Problema da Identificação de Genes por comparação de uma sequência de DNA com uma sequência de cDNA e uma solução para ele. Na segunda parte do Capítulo 4 propusemos um problema matemático, o Problema do Alinhamento *Spliced* Múltiplo, que que visa modelar o problema da identificação de genes por comparação de um DNA com vários cDNAs. Ainda na segunda parte do Capítulo 4, demonstramos que o Problema do Alinhamento *Spliced* Múltiplo é NP-completo. No Capítulo 5 apresentamos quatro heurísticas para o Problema do Alinhamento *Spliced* Múltiplo acompanhadas de uma avaliação comparativa entre elas. No Capítulo 6 descrevemos o trabalho realizado no desenvolvimento e avaliação das ferramentas de identificação de genes por comparação de um DNA com vários cDNAs baseadas nas quatro heurísticas propostas no capítulo anterior. Finalmente, no Capítulo 7, realizamos as considerações finais e apresentamos algumas idéias para trabalhos futuros.

Capítulo 2

Conceitos Básicos

Apresentaremos nesta seção os principais conceitos necessários à compreensão do problema abordado neste trabalho, assim como das estratégias utilizadas para sua solução. Essa contextualização está dividida em três seções: Biologia Molecular, Ciência da Computação e Bioinformática, cujas definições foram retiradas de [2, 14, 26, 41], [13, 15, 19, 27, 41] e [36, 41], respectivamente.

2.1 Conceitos de Biologia Molecular

Um dos objetivos deste trabalho é o estudo de um problema de otimização combinatória oriundo de um problema da Biologia, denominado de problema da identificação de genes. O problema da identificação de genes consiste, basicamente, em localizar os genes codificados em uma sequência de DNA. Um gene, de acordo com as leis de Mendel, é um "fator particular" que passa, sem ser modificado, do progenitor para a progênie [26]. Apesar de amplamente aceita, essa definição não é precisa. Para chegarmos a uma definição de gene mais adequada aos objetivos deste trabalho, precisamos antes dos conceitos de alguns elementos biológicos ligados aos genes. Falaremos sobre estes elementos a seguir.

2.1.1 Células

Os organismos vivos são compostos por unidades básicas chamadas de células. Há dois tipos de células, as **procariotas** e as **eucariotas**. As eucariotas possuem um núcleo bem definido e separado do citoplasma por uma membrana. Essa membrana denomina-se **membrana nuclear**. As procariotas não possuem membrana nuclear, de forma que o núcleo e o citoplasma não têm uma divisão clara. Essa classificação das células induz a uma classificação dos organismos formados por elas. Assim, os organismos também são divididos em **procariotos** e **eucariotos**, sendo as sequências genômicas desses últimos o alvo principal do problema abordado neste trabalho.

Todos os organismos originam-se a partir de um processo que divide uma célula em duas (mitose) ou quatro (meiose), chamado de divisão celular. As informações sobre as características das novas células produzidas e, consequentemente, do indivíduo

composto por estas células, ficam armazenadas na forma de **ácido nucleico**. Esses ácidos compõem estruturas intracelulares chamadas de **cromossomos**. Ao conjunto de cromossomos de um organismo é dado o nome de **genoma**. De maneira simplificada, podemos dizer que o genoma de um organismo contém a informação necessária para a construção daquele organismo. As células de um organismo qualquer são formadas principalmente por proteínas, explicadas a seguir.

2.1.2 Proteínas

Um **peptídeo** é uma molécula composta por **aminoácidos**. Os aminoácidos se unem na forma de uma cadeia linear, através das chamadas **ligações peptídicas**. Uma cadeia com muitos aminoácidos é chamada de **polipeptídeos**. Uma **proteína** é uma macromolécula formada por uma ou mais cadeias polipeptídicas. Existem vinte aminoácidos conhecidos que, combinados, formam diferentes proteínas. A informação sobre quais são os aminoácidos necessários na síntese das proteínas utilizadas na composição das células e outros produtos metabólicos fica armazenada nos ácidos nucleicos. Os detalhes do processo de síntese de proteínas serão dados a seguir.

2.1.3 Ácidos nucleicos (DNA e RNA)

Os ácidos nucleicos são cadeias de subunidades chamadas **nucleotídeos**. Cada nucleotídeo é formado por um grupo fosfato, uma pentose (açúcar) e uma **base nitrogenada**, que pode ser de cinco tipos diferentes. A Figura 2.1 ilustra a estrutura de um nucleotídeo.

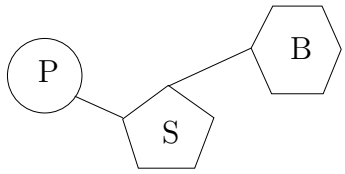


Figura 2.1: Exemplo da estrutura de um nucleotídeo onde S é a pentose, B é a base nitrogenada e P é o grupo fosfato.

Os ácidos nucleicos podem ser de dois tipos: o **ácido desoxirribonucleico** (do inglês, *deoxyribonucleic acid*), conhecido por **DNA**, e o **ácido ribonucleico** (do inglês, *ribonucleic acid*), conhecido por **RNA**. O DNA e o RNA possuem diferenças tanto composicionais quanto estruturais. Quanto às diferenças composicionais, no DNA as bases nitrogenadas podem ser: adenina (**A**), guanina (**G**), citosina (**C**) ou timina (**T**). No RNA, a timina é substituída pela uracila (**U**). Além disso, o açúcar componente do RNA é a **ribose**, enquanto a **2-desoxirribose** é o açúcar componente do DNA. A diferença entre eles está no fato de que a ribose possui um grupo OH na posição 2 do anel da pentose, enquanto a 2-desoxirribose possui apenas um H nessa posição. Na ribose e na desoxirribose, os átomos de carbono são numerados de um a cinco (com um apóstrofo após o número). Em uma cadeia linear de nucleotídeos, estes são unidos por uma ligação covalente do carbono $5'$ de um açúcar ao carbono $3'$ do açúcar seguinte, através de um grupo fosfato. Essas ligações recebem o nome de ligação fosfodiéster $5' - 3'$. A Figura 2.2 ilustra uma cadeia linear de nucleotídeos. Observe, na Figura 2.2,

que o nucleotídeo em uma das extremidades de uma fita[1] possui o carbono na posição $5'$ de sua pentose livre, enquanto que o nucleotídeo na outra extremidade possui o carbono na posição $3'$ livre. Por convenção, a leitura de uma seqüência de nucleotídeos é feita no sentido $5' \to 3'$.

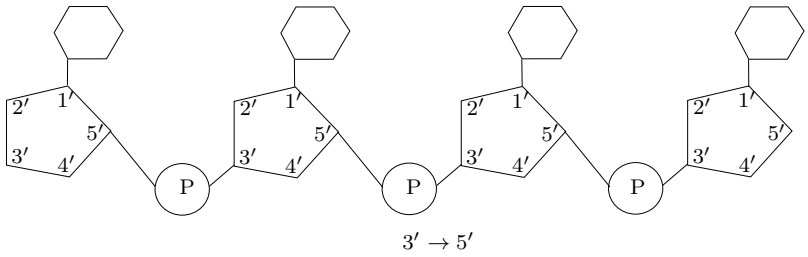


Figura 2.2: Cadeia linear de nucleotídeos, onde as ligações fosfodiésteres $5' - 3'$ estão representadas por arestas ligando os carbonos $5'$ e $3'$ das pentoses a um fosfato (representado por P).

Com base na Figura 2.2, podemos agora falar da diferença estrutural entre o DNA e o RNA. Enquanto esse último é constituído por uma única fita, o DNA é constituído por um par de fitas, em formato helicoidal e em sentidos opostos, ou seja, uma delas tem a orientação $5' \to 3'$ e a outra a orientação $3' \to 5'$. As bases nitrogenadas de uma das fitas se ligam às bases nitrogenadas da outra fita, em pares, através de pontes de hidrogênio. As pontes de hidrogênio se formam entre pares específicos de bases, que são adenina/timina (AT) e guanina/citosina (GC). Chamamos esses pares de **pares de bases complementares**. Na Figura 2.3 ilustramos as ligações fosfodiéster $5' - 3'$ e as pontes de hidrogênio de um par de fitas de DNA.

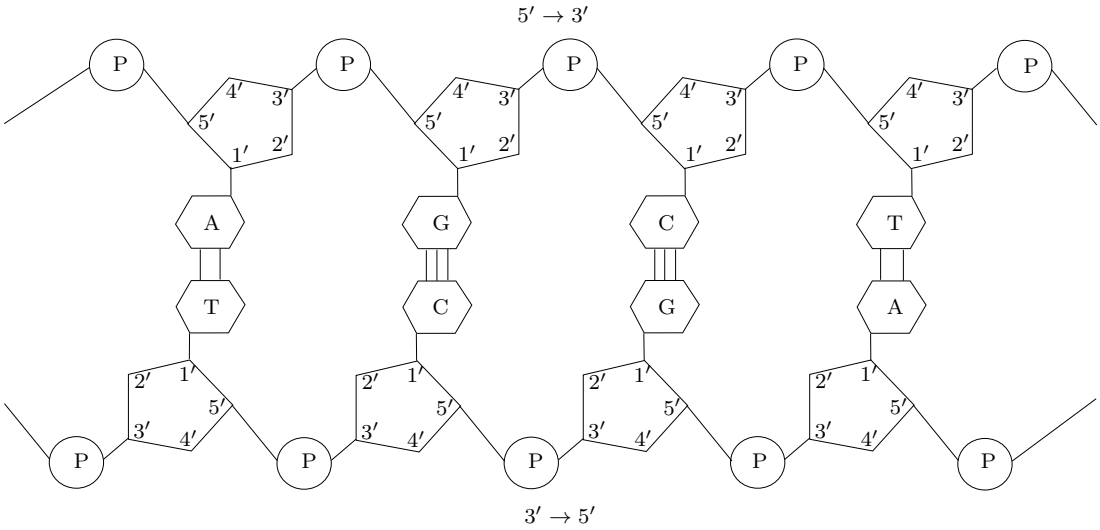


Figura 2.3: Par de fitas de DNA, com quatro pares de nucleotídeos, em uma visão bidimensional para facilitar a compreensão. As pontes de hidrogênio são representadas pelas arestas que ligam os hexágonos (bases) par a par e as ligações fosfodiéster são representadas pelos pares de arestas que ligam os pentágonos (açúcares) utilizando um fosfato (círculo com a letra P).

[1]Neste texto, os termos fita e cadeia linear referem-se a uma cadeia linear de nucleotídeos, e serão utilizados sem distinção entre si.

Dentre as várias funções dos ácidos nucleicos está a de codificar os aminoácidos que compõem as proteínas. Essa codificação é a base do processo de síntese de proteínas, explicado na próxima seção.

2.1.4 Síntese de proteínas

Um dos processos que ocorrem no interior das células é o de **síntese de proteínas**. Isso é feito traduzindo-se informações codificadas em trechos específicos do DNA em aminoácidos e conta com o auxílio de vários RNAs distintos e de algumas proteínas. Mais especificamente, o processo de síntese de proteínas, chamado também de **expressão gênica**, possui quatro fases: **transcrição**, ***splicing***, **transporte e tradução**.

A expressão gênica se inicia com a transcrição, que é a síntese de uma fita de RNA a partir de um trecho específico do DNA. Nos organismos eucariotos, esta fita de RNA é chamada de **pré RNA mensageiro** (pré-mRNA). O pré-mRNA é sintetizado através do pareamento de nucleotídeos livres existentes no núcleo com um trecho de uma das fitas do DNA. Dessa forma, o pré-mRNA corresponde a um trecho da sequência de uma das fitas do DNA (exceto pelas bases T do DNA, que no pré-mRNA são substituídas pelas bases U), e é complementar ao mesmo trecho da outra fita do DNA. A transcrição é realizada por enzimas denominadas **RNA polimerases**.

Nos organismos eucariotos existem regiões do trecho de DNA transcrito que não são traduzidas. Após a transcrição, ocorre o processo de ***splicing***, que remove os trechos não-codificantes do pré-mRNA, dando origem ao RNA mensageiro maduro ou simplesmente **RNA mensageiro** (mRNA). As fases de transcrição e de *splicing* ocorrem no núcleo da célula. A fase seguinte de tradução ocorre no citoplasma. Dessa forma, é necessário que o mRNA seja transportado para fora do núcleo através da membrana nuclear. Esse trabalho, que constitui a fase de transporte, é realizado por complexos de proteínas que formam os **poros nucleares**.

Na fase de tradução, os nucleotídeos do mRNA serão traduzidos em aminoácidos. Cada aminoácido é codificado por uma tripla de bases nitrogenadas, chamada de **códon**. Como existem quatro bases nitrogenadas diferentes, podemos pensar intuitivamente que existem 64 aminoácidos (4^3). No entanto, existem apenas 20 aminoácidos conhecidos, com alguns deles sendo codificados por mais de um códon. Há alguns códons que não representam um aminoácido e são chamados de **códons de parada**, pois sinalizam o final da fase de tradução. A relação entre os códons e seus respectivos aminoácidos denomina-se **Código Genético** e pode ser vista na Tabela 2.1.

A fase de tradução começa com o trabalho do **RNA de transferência** (tRNA). Existem vários tipos de moléculas de tRNA, e cada um deles corresponde a um determinado aminoácido. Em uma de suas extremidades, o tRNA possui uma tripla de nucleotídeos, chamada de **anticódon**, complementares aos nucleotídeos de um códon que representa um aminoácido. Do outro lado do tRNA pode estar presente o aminoácido correspondente ao códon complementar ao anticódon daquele tRNA. Se o aminoácido estiver presente no tRNA, dizemos que ele está **carregado** e o chamamos de **aminoacil-tRNA**. As moléculas de aminoacil-tRNA ligam-se a trechos do mRNA pelo pareamento de seu anticódon com o códon presente no mRNA. Um aminoacil-tRNA ligado a um

Primeira base	Segunda base T	C	A	G	Terceira base
T	Phe	Ser	Tyr	Cys	T
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	Stop	Stop	A
	Leu	Ser	Stop	Trp	G
C	Leu	Pro	His	Arg	T
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	T
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	T
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Tabela 2.1: Tabela contendo o Código Genético. A sigla Stop corresponde aos códons de parada.

mRNA tem seu aminoácido ligado à cadeia de aminoácidos que está sendo construída sobre aquele mRNA. Após isso, o tRNA pode ser liberado. O processo de ligação do aminoacil-tRNA no mRNA assim como a ligação do seu aminoácido à cadeia da proteína, e a liberação do tRNA são realizados pelo **ribossomo**, composto por dois RNAs e mais de 50 proteínas diferentes. Ele atua percorrendo a fita de mRNA e realizando a tradução daquela fita em uma proteína, até encontrar um códon de parada. Na Figura 2.4 ilustramos todo o funcionamento da expressão gênica.

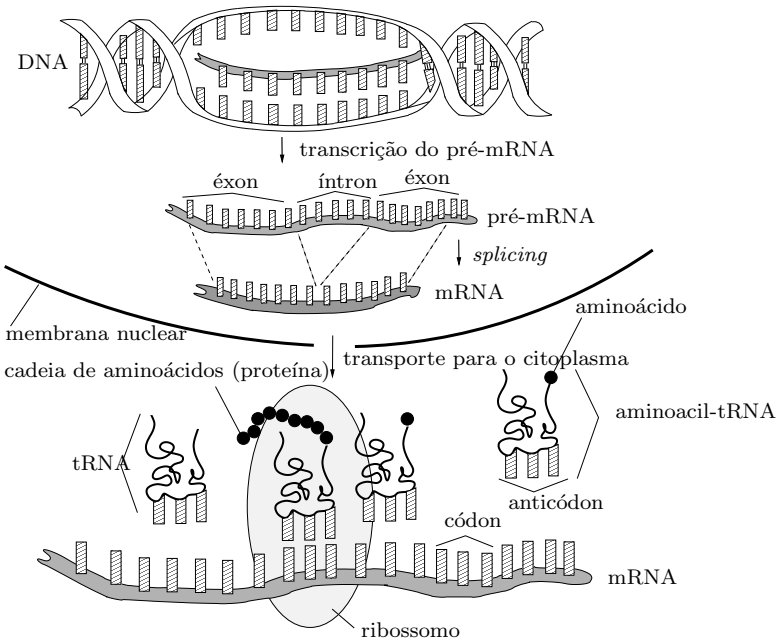


Figura 2.4: Processo de expressão gênica.

2.1.5 Genes e regiões funcionais

Na Seção 2.1 definimos gene como um "fator particular" que passa, sem ser modificado, do progenitor para a progênie. Esta definição é correta, mas vamos redefinir gene, dessa vez no contexto do processo de síntese de proteínas: um **gene** corresponde a um fragmento de DNA que pode ser transcrito em um pré-mRNA. As regiões do DNA situadas entre os genes são chamadas de **regiões intergênicas**.

Na Seção 2.1.4 dissemos que nem toda a informação de um gene é utilizada para a produção de proteínas e que parte do pré-mRNA é descartado no processo de *splicing*. A partir dessa informação, dividimos os genes em duas regiões: **éxons** e **íntrons**. Um éxon é um trecho contíguo de uma sequência de DNA que pode[2] ser utilizado na síntese do mRNA. Um íntron é um trecho do DNA que é descartado no processo de *splicing*. De acordo com a posição onde se encontram dentro do gene, os éxons podem ser classificados em quatro classes: **éxon inicial** (primeiro éxon do gene), **éxon final** (último éxon do gene), **éxon interno** (qualquer éxon situado entre os éxons inicial e final) e **éxon único** (éxon componente de um gene constituído por um único éxon). As regiões correspondentes aos éxons de uma sequência de DNA, também são chamadas de **regiões codificantes**.

É interessante notar que há genes que codificam mais de uma proteína. Isso é possível devido a um processo chamado de ***splicing* alternativo**, onde vários mRNAs diferentes podem ser sintetizados a partir de um mesmo gene, utilizando subconjuntos distintos do seu conjunto original de éxons. Os conjuntos alternativos de éxons que dão origem a proteínas costumam seguir a ordenação inicial.

Além de éxons e íntrons, existem outras porções da sequência de DNA com papéis variados na expressão gênica. Estas regiões são chamadas de **regiões funcionais**. Listamos algumas delas abaixo:

- **Promotor**: se localiza no início de um gene. A enzima RNA-polimerase liga-se a esta região para dar início à transcrição;
- **Terminador**: se localiza no final de um gene e sinaliza o final do processo de transcrição;
- **Códons de iniciação e de parada**: sinalizam, respectivamente, os locais onde começa e onde termina a tradução. Os códons de iniciação e de parada fazem parte dos éxons. O códon de iniciação corresponde a um `ATG`. O códon de parada pode ser um `TAA`, `TAG` ou `TGA`;
- **Sítios de aceitação e de doação**: em genes compostos por mais de um éxon, determinam os inícios (sítio de doação) e finais (sítio de aceitação) dos íntrons destes genes. Estes sítios fazem parte dos íntrons. Os sítios de doação e de aceitação são `GT` e `AG`, respectivamente. Apesar da grande maioria dos sítios de aceitação/doação e códons de iniciação/parada seguirem os padrões de nucleotídeos mencionados, existem exceções. Um sítio de aceitação pode ser, por exemplo, um `AC`.

[2] Nem todos os éxons são obrigatoriamente transcritos em um mRNA, graças ao *splicing* alternativo, que será melhor explicado mais adiante.

Um exemplo de fita de DNA com os éxons, íntrons e algumas regiões funcionais pode ser visto na Figura 2.5. Nesta figura são ilustradas, além das regiões funcionais, as regiões 5′UTR, 3′UTR e poly-(A).

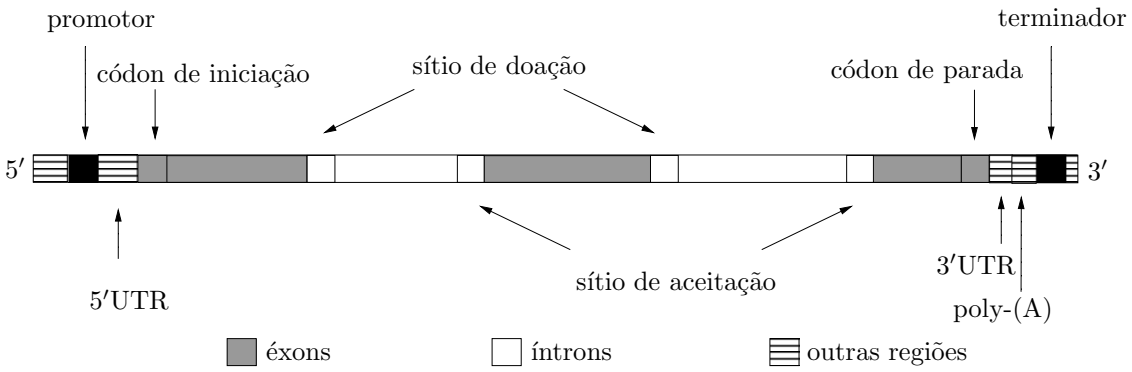


Figura 2.5: Exemplo de DNA com um gene composto por três éxons.

2.1.6 cDNAs e ESTs

No estudo da expressão gênica vimos que, na transcrição, o RNA (pré-mRNA) é sintetizado a partir de uma sequência de DNA. O caminho inverso, onde um RNA produz um trecho de uma sequência de DNA, também existe e é chamado **transcrição reversa**. Esse processo, descoberto inicialmente no processo de replicação de retrovírus[3], é realizado por uma enzima chamada de **transcriptase reversa**. Para estudar as características genéticas de um organismo, muitas vezes é conveniente estudar apenas os éxons, pois ainda não há evidências de que os íntrons atuam de alguma maneira na expressão gênica. Infelizmente, as moléculas de mRNA tornam-se instáveis quando estão fora da célula. No entanto, utilizando a transcriptase reversa, é possível construir em laboratório uma fita dupla de DNA a partir de uma fita de RNA. Como os mRNAs já são desprovidos de íntrons, o produto da transcrição reversa de um mRNA é um DNA composto apenas por éxons. Chamamos este tipo de sequência de **cDNA**.

Apesar dos grandes avanços no sequenciamento de cDNAs, esta tarefa ainda é cara e possui algumas restrições, principalmente quando realizada em genomas extensos. Para reduzir a complexidade desta tarefa, com pouca perda de precisão nos resultados, os pesquisadores costumam utilizar as sequências chamadas de **ESTs** (*Expressed Sequence Tags*). Essas sequências são trechos de cDNAs, com comprimento entre 200-800 bases, geradas a partir de uma de suas extremidades (5′ ou 3′). ESTs foram utilizadas com sucesso no sequenciamento do genoma humano [23, 51].

2.1.7 Variação genética

O genoma de qualquer organismo é suscetível a **mutações genéticas**. Elas acontecem devido a processos químicos que ocorrem dentro da própria célula ou pela influência de fatores ambientais, como por exemplo, da radiação. As mutações podem inserir, remover ou substituir bases em qualquer região do ácido nucleico. Existem também

[3]Os retrovírus são vírus compostos por uma fita única de RNA.

mutações em trechos do DNA que ocorrem durante a duplicação dessa molécula. Nessas mutações, um gene ou éxon pode ser omitido, duplicado ou combinado com outro, intregral ou parcialmente. Quando ocorrem nas regiões não-codificantes, as mutações geralmente não afetam o processo de expressão gênica. Mutações nas regiões codificantes, por outro lado, podem prejudicar a célula cujo DNA foi alterado. Graças à seleção natural, estas mutações, chamadas de **deletérias**, não costumam ser transmitidas a outras células e a outros indivíduos. Por fim, há mutações que produzem características desejáveis em um organismo. Estas mutações tendem a ser transmitidas a outros indivíduos. Como as mutações nas regiões não-codificantes geralmente não tem influência na sobrevivência do indivíduo, elas são muito mais comuns do que nas regiões codificantes. A menor ocorrência de mutações nas regiões codificantes é chamada de **princípio de conservação das bases**.

Apesar de se tornarem diferentes ao longo do tempo devido às mutações, alguns genes ainda podem estar relacionados de certa forma. Dizemos que um gene é **homólogo** a outro gene quando eles descendem de um mesmo gene ancestral. Um par de genes homólogos cuja origem foi uma duplicação são chamados de **parálogos**. Genes em espécies diferentes que derivam de um mesmo ancestral são chamados de **ortólogos**. Um trecho do DNA que no passado foi um gene mas que perdeu a capacidade de codificar uma proteína por ter sofrido uma ou mais mutações é chamado de **pseudogene**.

2.2 Conceitos de Ciência da Computação

Para podermos utilizar os conhecimentos da Ciência da Computação na solução de certos problemas da Biologia, é necessário encontrar uma maneira de formular esses problemas biológicos como problemas computacionais e então desenvolver algoritmos, heurísticas ou aproximações para eles. Nesta seção apresentamos alguns dos conceitos, definições e técnicas da Ciência da Computação que podem ser utilizados para esse fim.

2.2.1 Definições básicas

Um **alfabeto** Σ é um conjunto finito de caracteres. Uma **sequência** s construída sobre um alfabeto Σ, ou simplesmente uma sequência sobre Σ, é uma concatenação ordenada de caracteres pertencentes a Σ. Em uma sequência s, o **tamanho** ou **comprimento** dessa sequência corresponde à quantidade de caracteres de s, e é representado por $|s|$. Uma sequência cujo comprimento é zero é chamada de **sequência vazia** e denotada por ϵ. O caractere situado na posição i de uma sequência s é representado por $s[i]$. A **concatenação** de duas sequências s e t é uma sequência denotada por st, onde $|st| = |s| + |t|$, $st[i] = s[i]$, para $i = 1, \ldots, |s|$ e $st[|s| + i] = t[i]$, para $i = 1, \ldots, |t|$. Dado um alfabeto Σ, Σ^* é o conjunto contendo todas as sequências finitas possíveis de serem construídas sobre Σ.

Dizemos que s' é uma **subsequência** de uma sequência s se podemos obter s' a partir de s com a remoção de zero ou mais caracteres de s. Um **segmento** $s[i..j]$ de uma sequência s, com $1 \leq i \leq j \leq |s|$, é uma sequência formada pela concatenação dos caracteres $s[i]$, $s[i+1]$,..., $s[j]$, nesta ordem. Chamamos de **prefixo** de uma sequência s todo segmento de s da forma $s[1..j]$ com $1 \leq j \leq |s|$. Analogamente, chamamos de

sufixo de uma sequência s todo segmento de s da forma $s[i..|s|]$ com $1 \leq i \leq |s|$.

2.2.2 Complexidade computacional

Existem problemas onde, dado um conjunto S de possíveis soluções e uma função objetivo f que associa um valor a cada uma das soluções em S, é necessário encontrar uma solução que minimize ou maximize o valor de f. Esses problemas são conhecidos como **problemas de otimização combinatória**. Os problemas de otimização combinatória geralmente podem ser formulados como **problemas de decisão** onde, ao invés de buscar pelo valor máximo ou mínimo de uma função, buscaremos por uma resposta do tipo 'sim' ou 'não' ao problema. Para uma melhor compreensão desse fato, suponha que existe um problema onde a pergunta é: qual é o valor máximo de $f(s)$, com $f(s) \geq 0$, para qualquer $s \in S$? Em uma versão de decisão desse problema a pergunta seria: existe uma solução $s \in S$ tal que $f(s) \geq k$? Note que se existir uma solução para a versão de decisão do problema, podemos utilizá-la para resolver também a versão de otimização. Para isso perguntamos ao algoritmo que resolve a versão de decisão se existe uma solução para $k = 1$. Se houver, incremente k e pergunte novamente ao algoritmo, se existe uma solução para o novo k. No momento que o algoritmo da versão de decisão do problema responder 'não', saberemos que o valor anterior de k era o máximo. No estudo da complexidade computacional é comum utilizar a versão de decisão dos problemas.

Dizemos que um algoritmo é **limitado polinomialmente** (**algoritmo polinomial**) se a sua complexidade de tempo, no pior caso, é limitada superiormente por uma função polinomial sobre o tamanho da entrada. De maneira análoga, dizemos que um algoritmo é **limitado exponencialmente** (**algoritmo exponencial**) se a sua complexidade de tempo, no pior caso, é limitada superiormente por uma função exponencial sobre o tamanho da entrada. Na Ciência da Computação é comum dizer que um algoritmo limitado polinomialmente é **eficiente**.

Um problema é dito limitado polinomialmente se existe um algoritmo eficiente que o resolve. Com essa informação, definimos a classe de problemas P:

> P é a classe de complexidade dos problemas de decisão que são limitados polinomialmente.

Existem problemas de decisão para os quais não se conhece algoritmos eficientes que os resolvam. Entretanto, para alguns desses problemas é possível escrever algoritmos eficientes que verifiquem a validade de uma solução para o problema. Estes problemas são reunidos na classe NP:

> NP é a classe de complexidade dos problemas de decisão cujas soluções podem ser verificadas por um algoritmo eficiente.

Para todos os problemas em P existem algoritmos eficientes que podem verificar suas soluções. Por isso, dizemos que $P \subseteq$ NP. Entretanto, ainda não sabemos se NP $\subseteq P$.

Dados dois problemas A e B, dizemos que B é **redutível polinomialmente** a A se existe um algoritmo eficiente R, que para toda instância x de B, devolve uma instância $R(x)$ de A, de forma que uma solução 'sim' ou 'não' encontrada para A com a entrada $R(x)$ é a solução correta para B com a instância x [34]. Denotamos o fato de um problema B ser redutível polinomialmente a um problema A por $B \leq_P A$.

Dizemos que um problema C é **NP-difícil** se qualquer outro problema em NP pode ser reduzido polinomialmente a ele. Se, além disso, $C \in$ NP, dizemos que C é **NP-completo**. Suponha que C é NP-difícil. Como todos os problemas de NP são redutíveis a C, se encontrarmos um algoritmo eficiente que resolva C, poderemos resolver todos os problemas em NP reduzindo cada um deles a C.

As definições das classes P e NP tratam exclusivamente de problemas de decisão, enquanto que os problemas que nos interessam neste trabalho são problemas de otimização. Observe porém que, em geral, a versão de otimização de um problema é, no mínimo, tão difícil quanto a sua versão de decisão.

Aproximações e heurísticas

Dissemos que para cada problema da classe de complexidade P existe um algoritmo eficiente que o resolve. O termo eficiente significa simplesmente que estes algoritmos são limitados polinomialmente. Esse termo foi escolhido, pois os algoritmos eficientes geralmente podem ser executados para entradas de tamanho considerável em uma quantidade de tempo aceitável. Algoritmos limitados exponencialmente, ao contrário, são inviáveis até mesmo para entradas relativamente pequenas.

No contexto dos problemas da classe P, os estudos se resumem a buscar algoritmos polinomiais com o menor limite superior possível, e em alguns casos encontrar um algoritmo que tenha uma complexidade de tempo que não possa ser melhorada. Os problemas NP-difíceis podem ser resolvidos por algoritmos de complexidade de tempo exponencial. Apesar de corresponderem a soluções válidas para o problema, o tempo despendido por esses algoritmos é proibitivo, mesmo com a considerável capacidade de processamento das máquinas atuais. Com isso, outras abordagens foram desenvolvidas para o tratamento dos problemas em NP: os **algoritmos de aproximação** e as **heurísticas**.

Algoritmos de aproximação se aplicam a problemas de otimização. Para uma melhor compreensão desses algoritmos, considere um problema de minimização (maximização) A, tal que para uma instância x qualquer temos um conjunto $F(x)$ de possíveis soluções e uma função de custo $c : F(x) \mapsto \mathbb{N}^*$ que atribui um custo $c(s)$ para todo $s \in F(x)$. O custo de uma solução ótima é definido como $\mathrm{OPT}(x) = \min_{s \in F(x)} c(s)$ ($\mathrm{OPT}(x) = \max_{s \in F(x)} c(s)$). Seja M um algoritmo que, para uma instância x de A, retorna uma solução $M(x) \in F(x)$. Dizemos que M é um algoritmo de aproximação para A se, para uma constante real $\alpha \geq 1$, chamada de razão de aproximação, temos que:

$$\alpha \geq \max\left(\frac{c(M(x))}{\mathrm{OPT}(x)}, \frac{\mathrm{OPT}(x)}{c(M(x))}\right). \tag{2.1}$$

Como os valores das soluções são sempre positivos, a razão de aproximação de um algoritmo é sempre maior que 1. Um algoritmo que garante uma razão de aproximação

α é chamado de uma α**-aproximação**.

Heurísticas são critérios, métodos e princípios utilizados na escolha de um caminho, dentre vários, que supõe-se ser o mais adequado na busca por algum objetivo [35]. Heurísticas não dão garantias sobre o tempo de execução, a qualidade da solução ou ambos, mas sempre devolvem soluções válidas para um problema. Os métodos heurísticos geralmente são baseados em informações obtidas pelo estudo de peculiaridades do problema. Existe uma classe especial de heurísticas chamada de **metaheurísticas**.

Apesar de ainda não haver um consenso sobre a definição formal de metaheurística [7], elas são combinações de heurísticas básicas em arcabouços mais sofisticados, na exploração de um espaço de busca.

Algoritmo genético

Uma das metaheurísticas mais conhecidas é o **algoritmo genético**. Ela se baseia na teoria da evolução e utiliza os conceitos de reprodução, mutação e recombinação na busca de uma solução para um problema de otimização. O espaço de busca é mapeado em uma população, onde um processo de seleção que itera sobre gerações seleciona gradualmente os indivíduos mais aptos, através de uma função objetivo cujo resultado é chamado de **aptidão**. A estrutura genérica de um algoritmo genético é dada abaixo:

Passo 1. Inicialize e avalie a população inicial;

Passo 2. Realize uma seleção competitiva;

Passo 3. Aplique operadores genéticos para gerar novas soluções;

Passo 4. Avalie as soluções representadas pela população;

Passo 5. Volte ao Passo 2 e repita os passos seguintes até que algum critério de convergência seja satisfeito.

Apesar dos algoritmos de aproximação e heurísticas não nos garantirem uma solução ótima, eles são úteis pois geralmente não demandam uma quantidade proibitiva de recursos computacionais, pelo menos em comparação aos algoritmos limitados exponencialmente.

2.2.3 Programação Dinâmica

A Programação Dinâmica é uma estratégia para a solução de problemas baseada na (re)utilização dos valores de soluções de subproblemas. A técnica de divisão-e-conquista também tem esta proposta, mas há uma diferença entre elas. A divisão-e-conquista assume que os subproblemas são independentes e os resolve separadamente. Já a programação dinâmica utiliza o fato dos subproblemas compartilharem subsubproblemas. Com isso, ela reutiliza o valor de uma solução de um subsubproblema para calcular o valor de um ou mais um subproblemas.

Como na programação dinâmica a solução de um subsubproblema é utilizada na solução de vários subproblemas diferentes, ela é armazenada na memória. Dessa forma, poupa-se trabalho na requisição a uma subsubsolução ótima já calculada.

Dado um problema, o desenvolvimento de um algoritmo de programação dinâmica para resolvê-lo é dividido em quatro passos:

Passo 1. Caracterização da estrutura de uma solução ótima para o problema;

Passo 2. Definição recursiva do valor de uma solução ótima, ou seja, definição do valor de uma solução ótima em termos de subsoluções ótimas;

Passo 3. Cálculo do valor de uma solução ótima em uma orientação *bottom-up*;

Passo 4. Construção de uma solução ótima referente ao valor ótimo (esse passo pode ser omitido caso haja interesse somente no valor da solução).

Um problema deve possuir duas características para que possa ser resolvido através da programação dinâmica:

- Subestrutura ótima: uma solução ótima para o problema contém soluções ótimas para subproblemas;
- Subproblemas sobrepostos: um subproblema sobreposto é um problema cuja solução pode ser utilizada na construção da solução de dois ou mais problemas maiores.

2.2.4 Alinhamento de sequências

O alinhamento de sequências é uma técnica utilizada para comparar duas ou mais sequências, buscando pela ocorrência de uma série de caracteres ou por padrões de caracteres na mesma ordem em todas elas. Sejam um número inteiro $n > 0$, um alfabeto Σ que não contém o símbolo '$-$', um alfabeto $\overline{\Sigma} = \Sigma \cup \{' - '\}$ e um conjunto de sequências $S = \{s_1, s_2, ..., s_k\}$ sobre Σ, com $|s_i| = n_i$ para $1 \leq i \leq k$. Um **alinhamento** das sequências de S é uma matriz A de dimensões $k \times n$ com entradas $A[i][j] \in \overline{\Sigma}$ onde para cada i existe um conjunto $J_i = \{j_1, j_2, ..., j_{n_i}\} \subseteq \{1, 2, ..., n\}$, com $j_1 < j_2 < j_{n_i}$, tal que $A[i][j_1]A[i][j_2]...A[i][j_{n_i}] = s_i$ e $A[i][j] = ' - '$ para todo j em $\{1, 2, ..., n\} - J_i$. Além disso não deve existir um j tal que $A[i][j] = ' - '$ para todo $1 \leq i \leq k$, ou seja, não existe na matriz uma coluna formada exclusivamente de espaços. Para simplificar a notação, dado um conjunto de sequências $S = \{s_1, \ldots, s_k\}$ e um alinhamento A das sequências de S, chamaremos $A[i][]$ de $\overline{s_i}$ para $1 \leq i \leq k$.

Dado um alinhamento A de k sequências, uma **função de pontuação** $\omega : (\overline{\Sigma})^k \mapsto \mathbb{R}$ é uma função que mapeia k caracteres de $\overline{\Sigma}$ em um valor real. Podemos visualizar cada coluna de A como um vetor de entrada para ω, que especifica um valor em $\mathbb{R}$ para cada combinação possível de k caracteres. Dada uma função ω que atribui um valor real a uma coluna qualquer de um alinhamento A de dimensões $k \times n$, definimos o valor da pontuação de A com respeito a ω por $\sum_{j=1}^{n} \omega(A[][j])$, onde $A[][j]$ é a j-ésima coluna de A, e o denotamos por $\text{Score}_\omega(A)$. Dizemos que um alinhamento A de k sequências $\{s_1, s_2, ..., s_k\}$ é um **alinhamento ótimo** se não existe um alinhamento A' de $\{s_1, s_2, ..., s_k\}$ tal que $\text{Score}_\omega(A') > \text{Score}_\omega(A)$.

2.2.5 Alinhamento de duas sequências

Sejam s e t duas sequências quaisquer e A um alinhamento de s e t. Nesse contexto, chamamos de ***match*** uma coluna j de A tal que $A[1][j] = A[2][j]$ e de ***mismatch*** uma coluna j de A tal que $A[1][j] \neq A[2][j]$. Por fim, denominamos de ***space*** uma coluna j de A tal que $A[1][j] = \text{'}-\text{'}$ ou $A[2][j] = \text{'}-\text{'}$.

Podemos representar um alinhamento A das sequências s e t sobrepondo o conteúdo das duas linhas que formam a matriz A, ou seja, escrevendo-se $\overline{s}$ sobre $\overline{t}$. Um exemplo dessa representação com alguns *matches*, *mismatches* e *spaces* é mostrado na Figura 2.6, onde $s =$`ACAGTTCGTA` e $t =$`ACCGTGA`.

```
ACAGTTCGTA
ACCGT--GA-
```

Figura 2.6: Representação de um alinhamento entre duas sequências s e t, com 5 *matches*, 2 *mismatches* e 3 *spaces*.

A função de pontuação utilizada para determinar a pontuação de um alinhamento de duas sequências é uma função $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$ que mapeia dois caracteres de $\overline{\Sigma}$ em um valor real. O Score_ω de um alinhamento ótimo de um par de sequências s e t, dada uma função de pontuação ω, é também chamado de **similaridade** de s e t, e é denotado por $\text{sim}_\omega(s, t)$.

Com base nessas definições temos o seguinte problema:

> **Problema do alinhamento global de duas sequências**: dadas duas sequências s e t construídas sobre um alfabeto Σ, tal que $\text{'}-\text{'} \notin \Sigma$ e uma função de pontuação ω, encontrar o valor de $\text{sim}_\omega(s, t)$.

Para solução do problema do alinhamento global de duas sequências, bem como na solução de algumas de suas variantes, uma das técnicas utilizadas é a **programação dinâmica**. O algoritmo de programação dinâmica que apresentamos para o problema do alinhamento global de duas sequências foi proposto por Needleman e Wunsch em [31]. De forma simplificada, dadas duas sequências s e t, o algoritmo calcula a similaridade de todas as possíveis combinações de prefixos de s e de t, começando com a determinação da similaridade de prefixos curtos (incluindo prefixos vazios) e utilizando valores de similaridade já calculados de prefixos menores para calcular a similaridade de prefixos maiores.

Dadas duas sequências s e t de tamanhos m e n respectivamente, o algoritmo de Needleman-Wunsch utiliza uma matriz de dimensões $m + 1 \times n + 1$ para armazenar a similaridade dos possíveis prefixos das duas sequências. Essa matriz é chamada de **matriz de alinhamento**[4]. Em uma matriz de alinhamento M de duas sequências s e t, cada elemento $M[i][j]$ de M contém o valor de $\text{sim}_\omega(s[1..i], t[1..j])$. O preenchimento de cada elemento $M[i][j]$ é feito a partir do valor de um dos seguintes elementos (vizinhos de $M[i][j]$): $M[i-1][j]$, $M[i][j-1]$ ou $M[i-1][j-1]$. Mais detalhadamente, isso é feito com base na Recorrência 2.2.

[4]Neste texto consideraremos que seus índices iniciam em zero

$$M[i][j] = \max \begin{cases} M[i-1][j] + \omega(s[i], \text{'}-\text{'}) \\ M[i][j-1] + \omega(t[j], \text{'}-\text{'}) \\ M[i-1][j-1] + \omega(s[i], t[j]) \end{cases} \tag{2.2}$$

Explicando a Recorrência 2.2, ao utilizar o valor de $M[i-1][j]$, estamos calculando $\text{sim}_\omega(s[1..i], t[1..j])$ a partir de $\text{sim}_\omega(s[1..(i-1)], t[1..j])$ e devemos adicionar à $\text{sim}_\omega(s[1..(i-1)], t[1..j])$ o valor de $\omega(s[i], \text{'}-\text{'})$(*space* em t). Analogamente, ao utilizar o valor de $M[i][j-1]$, estamos calculando $\text{sim}_\omega(s[1..i], t[1..j])$ a partir de $\text{sim}_\omega(s[1..i]), t[1..(j-1)])$ e devemos adicionar à $\text{sim}_\omega(s[1..i], t[1..(j-1)])$ o valor de $\omega(t[j], \text{'}-\text{'})$ (*space* em s). Se utilizarmos o valor do elemento $M[i-1][j-1]$ estaremos calculando $\text{sim}_\omega(s[1..i], t[1..j])$ a partir de $\text{sim}_\omega(s[1..(i-1)], t[1..(j-1)])$ e devemos adicionar a pontuação do alinhamento de $s[i]$ com $t[j]$ (*match* ou *mismatch*).

O elemento $M[0][0]$ de toda matriz de alinhamento é inicializado com zero, pois armazena a similaridade de dois prefixos vazios. Os elementos da forma $M[0][j]$ e da forma $M[i][0]$ tem seus valores dados por $M[0][j-1] + \omega(t[j], \text{'}-\text{'})$ e $M[i-1][0] + \omega(s[i], \text{'}-\text{'})$ respectivamente e armazenam a similaridade entre prefixos de uma das sequências e uma sequência vazia. Em todas as outras posições de M, a escolha sobre qual das similaridades anteriormente calculadas deve ser utilizada no cálculo da atual é feita de tal forma que a pontuação da soma desse valor mais o valor de ω do novo par de caracteres inserido na solução seja máxima dentre as possíveis escolhas. Caso haja empate, escolhemos uma das pontuações máximas arbitrariamente. O algoritmo de Needleman-Wunsch é apresentado no Algoritmo 2.1.

ALGORITMO 2.1 Algoritmo de programação dinâmica que calcula a similaridade entre duas sequências.

NEEDLEMAN-WUNSCH(s, t, ω): recebe duas sequências s e t construídas sobre um alfabeto Σ, uma função de pontuação $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$, e devolve o valor da similaridade $\text{sim}_\omega(s, t)$

```
1:  M[0][0] ← 0; m ← |s|; n ← |t|;
2:  para j ← 1 até m faça
3:      M[0][j] ← M[0][j − 1] + ω(s[j], ' − ');
4:  para i ← 1 até n faça
5:      M[i][0] ← M[i − 1][0] + ω(t[i], ' − ');
6:  para i ← 1 até m faça
7:      para j ← 1 até n faça
8:          M[i][j] ← M[i − 1][j] + ω(s[i], ' − ');
9:          se M[i][j] ≤ M[i − 1][j − 1] + ω(s[i], t[j]) então
10:             M[i][j] ← M[i − 1][j − 1] + ω(s[i], t[j]);
11:         se M[i][j] ≤ M[i][j − 1] + ω(t[j], ' − ') então
12:             M[i][j] ← M[i][j − 1] + ω(t[j], ' − ');
13: devolva M[m][n];
```

Analisando o Algoritmo 2.1, observa-se que o trecho que domina a complexidade deste algoritmo está nos laços que se iniciam nas linhas 6 e 7. O número total de operações realizadas nesses laços é da ordem de mn. Dessa forma, podemos dizer que a complexidade de tempo do algoritmo é $O(mn)$. Sua complexidade de espaço também é $O(mn)$ pois é utilizada uma matriz de dimensões $m+1 \times n+1$ (matriz M). Observe que é possível reduzir essa complexidade para $O(\min\{m, n\})$, já que podemos utilizar

apenas dois vetores de comprimento $\min\{m, n\}$, pois a recorrência para o cálculo de uma posição na matriz utiliza apenas os valores da linha anterior ou da mesma linha da posição sendo calculada.

Na Figura 2.7 é ilustrada a matriz de alinhamento de duas sequências $s =$`PHEAE` e $t =$`AGHEE`, preenchida através do algoritmo de Needleman-Wunsch. Neste exemplo, a função de pontuação ω é tal que $\omega(\alpha, '-') = -2$, $\omega(\alpha, \alpha) = 1$ e $\omega(\alpha, \beta) = -1$, onde α e β são dois caracteres tais que $\alpha \neq \beta$. As setas indicam o caminho para a construção de um alinhamento A entre s e t, tal que $\text{Score}_\omega(A) = \text{sim}_\omega(s, t)$. Mais especificamente, as setas diagonais correspondem a uma coluna de A com um caractere de s sobre um caractere de t. As setas verticais correspondem a uma coluna de A com um caractere de s sobre um espaço (em t). As setas horizontais correspondem a uma coluna de A com um espaço (em s) sobre um caractere de t. Observe que pode haver mais de um caminho de setas tal que um alinhamento construído com base nesse caminho é ótimo. As sequências $\overline{s}$ e $\overline{t}$ de A são `P-HEAE` e `AGHE-E`, respectivamente.

		A	G	H	E	E
	0	−2	−4	−6	−8	−10
P	−2	−1	−3	−5	−7	−9
H	−4	−3	−2	−2	−4	−6
E	−6	−5	−4	−3	−1	−3
A	−8	−5	−6	−5	−3	−2
E	−10	−7	−6	−7	−4	−2

Figura 2.7: Matriz de alinhamento das sequências $s =$`PHEAE` e $t =$`AGHEE`.

Mostramos uma solução para o problema do alinhamento global de duas sequências no Algoritmo 2.1. Entretanto, existem aplicações onde o interesse não é somente pelo valor da similaridade, mas também pelo alinhamento ótimo. A construção deste alinhamento pode ser feita utilizando-se a matriz de alinhamento do algoritmo de Needleman-Wunsch, já preenchida. Seja uma matriz de alinhamento M de duas sequências s e t. No preenchimento de M, fizemos escolhas para calcular o valor de cada elemento $M[i][j]$ baseado nos valores de $M[i-1][j]$, $M[i][j-1]$ e $M[i-1][j-1]$. Como dissemos anteriormente, cada uma dessas escolhas corresponde a um alinhamento entre caracteres de s e t ou entre caracteres de s ou t e um '−'. Dessa forma, se partirmos de um elemento $M[i][j]$ de M e, a cada passo, identificarmos qual foi o alinhamento de caracteres que contribuiu para gerar o valor de $M[i][j]$, podemos construir o alinhamento que gerou a similaridade em $M[i][j]$. Para identificar o elemento cujo valor foi utilizado em $M[i][j]$ compara-se o valor de $M[i][j]$ com os valores de $M[i-1][j-1] + \omega(s[i], t[j])$, $M[i-1][j] + \omega(s[i], '-')$ e $M[i][j-1] + \omega(t[j], '-')$ buscando por um valor igual. Fazendo isso a partir do elemento $M[m][n]$, e parando ao atingirmos a posição $M[0][0]$, obtemos um alinhamento ótimo de s e de t. Observe que ao comparar o valor de $M[i][j]$ com os valores de $M[i-1][j-1] + \omega(s[i], t[j])$, $M[i-1][j] + \omega(s[i], '-')$ e $M[i][j-1] + \omega(t[j], '-')$ para identificar qual desses valores foi anteriormente utilizado para o cálculo da posição $M[i][j]$, pode haver ambiguidade. Nesse caso, qualquer um dos valores pode ser utilizado e cada um deles vai resultar em um alinhamento ótimo diferente.

No método descrito para a construção do alinhamento, uma coluna, uma linha ou

uma coluna e uma linha são "consumidas" obrigatoriamente a cada iteração, partindo de $M[m][n]$ até chegar em $M[0][0]$. Como essa é a operação mais custosa do algoritmo, concluímos que sua complexidade é $O(m+n)$. Em ambos os métodos a complexidade de espaço é $O(mn)$ pois os dois dependem de uma quantidade constante de matrizes de dimensão da ordem de $m \times n$. Vale salientar, porém, que existem alguns algoritmos que permitem a construção do alinhamento em espaço linear, como o descrito em [30]. Ele corresponde a uma adaptação da solução do problema da maior subsequência comum, em espaço linear, proposta em [22]. O algoritmo que constrói um alinhamento ótimo entre duas sequências s e t é apresentado no Algoritmo 2.2.

ALGORITMO 2.2 Algoritmo que constrói um alinhamento ótimo entre duas sequências.

ALINHA(s,t,ω,M): recebe duas sequências s e t construídas sobre um alfabeto Σ, uma função de pontuação $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$, uma matriz de alinhamento M de s e t preenchida utilizando ω, e devolve um alinhamento ótimo de s e t para ω.

```
1: m ← |s|; n ← |t|; blen ← 0;
2: enquanto m > 0 e n > 0 faça
3:    se M[m][n] = M[m][n − 1] + ω(t[n], ' − ') então
4:       B[0][blen] ← ' − '; B[1][blen] ← t[n];
5:       n ← n − 1;
6:    senão
7:       se M[m][n] = M[m − 1][n − 1] + ω(s[m], t[n]) então
8:          B[0][blen] ← s[m]; B[1][blen] ← t[n];
9:          m ← m − 1; n ← n − 1;
10:      senão
11:         se M[m][n] = M[m − 1][n] + ω(s[m], ' − ') então
12:            B[0][blen] ← s[m]; B[1][blen] ← ' − ';
13:            m ← m − 1;
14:   blen ← blen + 1;
15: devolva B com as colunas em ordem inversa;
```

Além do problema do alinhamento global de duas sequências, há duas outras variantes do problema do alinhamento global de duas sequências bastante citados na literatura. São eles o problema do **alinhamento local** de duas sequências e o problema do alinhamento **semi-global** de duas sequências.

No problema do alinhamento local de duas sequências, dadas duas sequências s e t construídas sobre um alfabeto Σ, tal que '$-$'$\notin \Sigma$ e uma função de pontuação ω, devemos encontrar o maior valor de $\text{sim}_\omega(s', t')$, onde s' é um segmento qualquer de s e t' é um segmento qualquer de t. Um algoritmo que resolve o problema do alinhamento local de duas sequências foi proposto por Smith e Waterman em [44] e corresponde a uma variante do algoritmo de Needleman-Wunsch [31], com uma pequena diferença na recorrência e na posição onde se encontra o valor de um alinhamento ótimo. A diferença na recorrência é que sempre há a opção de preencher uma posição da matriz de alinhamento com o valor 0 ao invés de utilizar um valor negativo e o valor de um alinhamento ótimo deve ser procurado em todas as posições dessa matriz.

O problema do alinhamento semi-global de duas sequências é semelhante ao problema do alinhamento global de duas sequências. Um alinhamento semi-global é um alinhamento global onde os espaços inseridos nas extremidades de s ou de t não são

considerados no cálculo da similaridade entre s e t. O algoritmo que resolve o problema do alinhamento semi-global de duas sequências é o mesmo do problema do alinhamento global de duas sequências, com algumas diferenças na inicialização da matriz e na posição onde se encontra o valor de um alinhamento ótimo. A linha 0 e a coluna 0 da matriz de alinhamento são inicializadas com o valor 0 e o valor de um alinhamento ótimo deve ser procurado na última linha ou na última coluna dessa matriz.

Sobre funções de pontuação e distância de edição

Os alinhamentos úteis na solução de problemas são geralmente aqueles que "posicionam" trechos semelhantes das sequências em uma mesma "região" do alinhamento. Se utilizarmos uma função de pontuação adequada, um alinhamento com essa característica terá uma pontuação alta. Em geral, as funções de pontuação que atendem a esse objetivo são as que favorecem (atribuem valores positivos) *matches* e penalizam (atribuem valores negativos) *mismatches* e *spaces*.

Um exemplo de função de pontuação com as características descritas no parágrafo anterior é a função $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$ definida como $\omega(\alpha, \beta) = 1$ quando $\alpha = \beta$, $\omega(\alpha, \beta) = -1$ quando $\alpha \neq \beta$ e $\omega(\alpha, '-') = -2$, onde Σ é um alfabeto e α e $\beta \in \overline{\Sigma}$.

Vale observar também que existe uma outra medida utilizada para comparar duas sequências que se chama **distância de edição**. Informalmente, uma distância de edição é a quantidade de operações necessárias para transformar uma sequência em outra. Diferente da similaridade, que mede quão parecidas duas sequências são, a distância de edição mede quão diferentes são duas sequências.

A descrição formal de distância de edição depende de alguns conceitos, apresentados a seguir. Esses conceitos foram retirados do trabalho de Nicolas e Rivals detalhado em [32].

Seja uma função $d : E \times E \mapsto \mathbb{R}$, onde E é um conjunto não-vazio. Dizemos que d é uma **métrica** sobre E se, e somente se, para todo x, y, $z \in E$, d satisfaz as seguintes condições:

- $d(x, y) \geq 0$ (positividade)
- $d(x, y) = 0 \iff x = y$ (separação)
- $d(x, y) = d(y, x)$ (simetria)
- $d(x, z) \leq d(x, y) + d(y, z)$ (desigualdade triangular)

Há três tipos de operações, chamadas de **operações de edição**, utilizadas para transformar uma sequência em outra. Essas operações são a inserção, a substituição e a remoção de um caractere. Uma métrica $\delta : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$ serve como uma função de pontuação para as operações de edição. Dados dois caracteres $a, b \in \Sigma$, o valor da substituição de a por b é dado por $\delta(a, b)$, o valor da remoção de um a é dado por $\delta(a, '-')$ e o valor da inserção de um b é dado por $\delta('-', b)$.

Dadas duas sequências s e t construídas sobre um alfabeto Σ e uma métrica $\delta : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$, o somatório dos valores da menor sucessão de operações de edição necessárias

para transformar s em t é chamado de **δ-distância de edição** e denotado por $d_\delta(s,t)$. Se δ é uma métrica sobre $\overline{\Sigma}$ tal que para todo $a, b \in \overline{\Sigma}$, $\delta(a,b) = 0$ se $a = b$ e $\delta(a,b) = 1$ caso contrário, então a δ-distância de edição é chamada simplesmente de **distância de edição** ou **distância de Levenshtein** e é denotada por d_E.

É possível utilizar o algoritmo do cálculo de similaridade entre duas sequências para calcular a d_E entre elas. Basta fazer com que a função de pontuação ω seja tal que para todo $a, b \in \overline{\Sigma}$, $\omega(a,b) = 0$ se $a = b$ e $\omega(a,b) = -1$ caso contrário. Ao final inverte-se o sinal da similaridade encontrada, que será equivalente à distância de edição entre as duas sequências [41].

2.2.6 Alinhamento de várias sequências ou alinhamento múltiplo

Até este ponto do texto, discutimos apenas o problema da comparação de duas sequências. Há problemas práticos, no entanto, onde a comparação de duas sequências não é suficiente ou adequada para resolvê-lo. Em algumas dessas situações, alinhar mais de duas sequências pode ser necessário. Um exemplo de alinhamento de várias sequências, também conhecido como alinhamento múltiplo, é ilustrado na Figura 2.8.

```
CA-GTTGAATGTGGAAGAA
CACGGTGAATGTGG-AGAA
GAAGGT-AATGTGGAAGAA
--AGGTGAATGTGGAAGAA
```

Figura 2.8: Exemplo de um alinhamento de quatro sequências.

A função de pontuação utilizada para determinar a pontuação de um alinhamento de k sequências é uma função $\omega : (\overline{\Sigma})^k \mapsto \mathbb{R}$ que mapeia k caracteres de $\overline{\Sigma}$ em um valor real. Nesse contexto temos o seguinte problema:

> **Problema do alinhamento de várias sequências (problema do alinhamento múltiplo)**: dadas $k > 2$ sequências $\{s_1, s_2, ..., s_k\}$ construídas sobre um alfabeto Σ, tal que '$-$'$\notin \Sigma$ e uma função de pontuação ω, encontrar a pontuação de um alinhamento múltiplo ótimo de $\{s_1, s_2, ..., s_k\}$.

O problema do alinhamento múltiplo pode ser resolvido utilizando-se o mesmo algoritmo de programação dinâmica proposto para o problema do alinhamento global, com a diferença de que ao invés de trabalhar com uma matriz bidimensional, o algoritmo utilizaria uma matriz M com k dimensões. Para preencher cada elemento de M, o algoritmo tem $2^k - 1$ possibilidades, contra $2^2 - 1 = 3$ da versão de duas sequências. Como cada posição da matriz deve ser visitada pelo menos uma vez, este algoritmo utiliza uma quantidade de tempo exponencial no tamanho da entrada, com um k variável. De fato o problema do alinhamento múltiplo é NP-completo, como demonstrado em [53].

Ao se pensar em uma função de pontuação ω no contexto de alinhamentos múltiplos, é desejável que ω não leve em consideração a ordem dos argumentos e que privilegie a ocorrência de vários caracteres semelhantes ou relacionados na mesma coluna e diminua

a ocorrência de espaços e resíduos. Uma função que atende a ambos estes requisitos é a função de **soma de pares** (**SP**). A função SP é dada pelo somatório das pontuações de pares de caracteres, onde cada par é uma combinação possível de dois caracteres de uma coluna. Formalmente, definimos $\mathrm{SP}_\omega : (\overline{\Sigma})^k \mapsto \mathbb{R}$, que mapeia uma coluna $\mathcal{C}$ em um valor real da seguinte maneira:

$$\mathrm{SP}_\omega(\mathcal{C}) = \sum_{1 \leq i < i' \leq k} \omega(\mathcal{C}[i], \mathcal{C}[i']), \tag{2.3}$$

onde ω é uma função de pontuação entre dois caracteres e $\mathcal{C}[i]$ corresponde ao caractere situado na linha i da coluna $\mathcal{C}$.

Com a definição de SP para uma coluna, podemos definir também uma pontuação SP para um alinhamento A, da seguinte maneira:

$$\mathrm{Score}_{\mathrm{SP}_\omega}(A) = \sum_{j=1}^{l} \mathrm{SP}_\omega(A[][j]) = \sum_{j=1}^{l} \sum_{1 \leq i < i' \leq k} \omega(A[i][j], A[i'][j]), \tag{2.4}$$

onde $A[][j]$ denota a coluna de índice j de A e l é a quantidade de colunas de A.

Dado um alinhamento múltiplo A de dimensões $k \times l$ de um conjunto de sequências construídas sobre um alfabeto Σ, podemos construir uma **sequência consenso** s_c deste alinhamento, que é uma espécie de resumo das sequências. A sequência s_c tem comprimento l, e cada caractere $s_c[i]$, com $1 \leq i \leq l$ é o caractere que mais ocorre na coluna i de A. Caso hajam dois ou mais caracteres com mais ocorrências que os demais, qualquer um deles pode ser escolhido [43].

Há algoritmos de aproximação para o problema do alinhamento múltiplo com a função SP, como o algoritmo **Estrela** de Gusfield, descrito em [21]. Antes de descrever o algoritmo Estrela definiremos os conceitos de **sequência central**, **alinhamento induzido** e de **alinhamento compatível**, utilizados por ele. Para isso, considere um conjunto de sequências $S = \{s_1, \ldots, s_k\}$ de tamanho k. Uma sequência s qualquer em S é dita uma sequência central de S se $\sum_{t \in S \setminus \{s\}} \mathrm{sim}(s, t)$ é o maior dentre todas as sequências de S. Dado um alinhamento múltiplo A das sequências de S, um alinhamento de duas sequências s_i e $s_j \in S$ induzido por A, denotado por $A_{|s_i s_j}$, corresponde a um alinhamento B, onde $B[1] = \overline{s_i}$ e $B[2] = \overline{s_j}$, com a remoção das colunas l de B onde $B[1][l] = B[2][l] = \text{'}-\text{'}$. Um alinhamento múltiplo A é dito compatível com um alinhamento A' das sequências s_i,$s_j \in S$ se $\mathrm{Score}(A_{|s_i s_j}) = \mathrm{Score}(A')$.

Dado um conjunto S de k sequências como entrada, o algoritmo Estrela tem como primeiro passo a determinação da sequência central de S. A determinação dessa sequência é realizada fixando-se uma sequência $s \in S$ e calculando o valor de $\sum_{t \in S \setminus \{s\}} \mathrm{sim}_\omega(s, t)$. Isso é feito para todas as sequências do conjunto, escolhendo-se ao final aquela mais próxima de todas as outras, que será chamada de s_c. Após a determinação de s_c, o algoritmo Estrela constrói um alinhamento múltiplo A compatível com os alinhamentos de s_c e cada uma das sequências em $S \setminus \{s_c\}$. Esse alinhamento corresponde à aproximação do alinhamento ótimo múltiplo e a sua construção é descrita a seguir.

Inicializamos o alinhamento múltiplo A com a sequência s_c e o aumentamos iterativamente, inserindo linhas compatíveis com os alinhamentos entre s_c e cada uma das

sequências em $S \setminus \{s_c\}$. A inserção de novas linhas em A é feita inserindo colunas de espaços em A ou na sequência que está sendo acrescentada de forma que, ao final, todas as linhas de A possuam o mesmo comprimento. A razão de aproximação do algoritmo Estrela é $2 - 2/k$. O algoritmo Estrela é apresentado no Algoritmo 2.3.

ALGORITMO 2.3 Algoritmo de aproximação que constrói um alinhamento múltiplo.

ESTRELA$(s_1, \ldots, s_k, \omega)$: recebe k sequências construídas sobre um alfabeto Σ, uma função de pontuação ω e devolve um alinhamento múltiplo A das k sequências tal que $\text{Score}_{\text{SP}_\omega}(A) \geq (2 - 2/k)\text{Score}_{\text{SP}_\omega}(A^*)$ onde A^* é um alinhamento múltiplo ótimo das k sequências com a função SP.

1: **para** $i \leftarrow 1$ até k **faça**
2: $\quad M[i] \leftarrow \sum_{j=1, j \neq i}^{k} \text{sim}_\omega(s_i, s_j)$;
3: $c \leftarrow i$ tal que $M[i]$ é máximo;
4: $A[1] \leftarrow s_c$;
5: **para** todo $s_i \neq s_c$ com $1 \leq i \leq k$ **faça**
6: $\quad a \leftarrow$ alinhamento ótimo de s_c e s_i com a função de pontuação ω;
7: $\quad$ Percorra as sequências $A[1]$ e $a[1]$ simultaneamente, utilizando o índice j. Para cada j tal que $A[1][j]$ contém um espaço e $a[1][j]$ não contém, insira uma nova coluna na posição j da matriz a e preencha essa coluna com espaços. Para cada j tal que $a[1][j]$ contém um espaço e $A[1][j]$ não contém um espaço, insira uma nova coluna na posição j da matriz A e preencha essa coluna com espaços;
8: $\quad$ Insira uma linha com a sequência $a[2]$ no final de A;
9: **devolva** A;

Para a determinação da complexidade de tempo do algoritmo Estrela, considere m como sendo o comprimento da maior sequência de S. O tempo consumido pelo laço que se inicia na linha 1 do Algoritmo 2.3 é igual a $O(k^2m^2)$. As operações das linhas 3 e 4 consomem tempo $O(k)$ e $O(m)$, respectivamente. O laço que se inicia na linha 5 é executado k vezes. O alinhamento da linha 6 consome tempo igual a $O(m^2)$ e a inserção da linha 8 gasta tempo $O(m)$. A linha 7 realiza, no máximo, m^2 operações, pois o índice j é menor ou igual a $4m$ e cada inserção de uma coluna em um alinhamento consome tempo igual a $O(m)$ se utilizadas estruturas de dados adequadas. Portanto, a linha 7 consome tempo igual a $O(m^2)$. Sendo assim, o laço que se inicia na linha 5 consome tempo igual a $O(km^2)$. Concluímos então que o tempo total do Algoritmo 2.3 é dominado pelo primeiro laço, ou seja, é igual a $O(k^2m^2)$.

Sobre a complexidade de espaço do algoritmo, observe que as linhas de 1 a 4 utilizam espaço $O(m)$ para serem executadas. É necessário espaço $O(m^2)$ para o alinhamento da linha 6 e uma matriz $O(k^2m)$ para as inserções da linha 8. A complexidade de espaço do algoritmo Estrela é, portanto, $O(k^2m + m^2)$.

2.2.7 Modelos Ocultos de Markov

O **Modelo Oculto de Markov**, ou **HMM** (do inglês, *Hidden Markov Model*), é um modelo estatístico utilizado, principalmente, no reconhecimento de padrões. Mais especificamente, um HMM é um conjunto finito de estados onde ocorrem transições de um estado para outro (ou para ele mesmo) ao longo de uma sucessão de instantes de tempo. As transições entre os estados são realizadas de acordo com um conjunto de

probabilidades de transição. A probabilidade de transição para um determinado estado depende apenas do estado anterior a ele. Cada um dos estados de um HMM pode gerar uma **observação**, que corresponde a um elemento escolhido de um **conjunto de observações**. A cada estado de um HMM está associado um conjunto de **distribuição de probabilidades** que determina a probabilidade de escolha de cada elemento do conjunto de observações naquele estado. O estado inicial de um HMM é escolhido de acordo com uma **distribuição de probabilidades dos estados iniciais**.

Definimos formalmente um HMM como uma quíntupla $\lambda = \{S, V, A, B, \Pi\}$, onde $S = \{S_1, \ldots, S_n\}$ é um conjunto finito de estados, $V = \{v_1, \ldots, v_m\}$ é um conjunto finito de observações, $A = \{a_{ij}\}$ com $1 \leq i \leq n$ e $1 \leq j \leq n$ é um conjunto de probabilidades de transição entre os estados, $B = b_{ij}$ com $1 \leq i \leq m$ e $1 \leq j \leq n$ é um conjunto de distribuição de probabilidades e $\Pi = \{\pi_i\}$ com $1 \leq i \leq n$ um conjunto com a distribuição de probabilidades dos estados iniciais.

Dado um modelo $\lambda = \{S, V, A, B, \Pi\}$, podemos gerar uma **sequência de observações** $O = \{o_1, \ldots, o_t\}$ onde $o_i \in V$. Essa sequência é gerada escolhendo-se um estado inicial através da distribuição de probabilidades dos estados iniciais, e realizando t transições entre os estados de S, onde antes de cada transição é gerada uma observação. Observe que uma determinada sequência de observações pode ser gerada por uma ou mais sequências de estados diferentes de um modelo. O modelo é chamado de oculto pelo fato de que dada uma sequência de observações, não sabemos qual foi a sequência de estados que deu origem a ela.

Existem três problemas básicos relacionados a HMMs que geralmente servem de modelo para problemas de reconhecimento de padrões. Para a definição desses problemas, considere um modelo $\lambda = \{S, V, A, B, \Pi\}$ e uma sequência de observações $O = \{o_1, \ldots, o_t\}$. Os problemas são:

1. Calcular a probabilidade de O ter sido gerada por λ, $P(O|\lambda)$;
2. Descobrir qual a sequência de estados de λ mais provável de gerar O;
3. Ajustar os parâmetros de λ para maximizar a probabilidade $P(O|\lambda)$.

Os dois primeiros problemas podem ser utilizados no reconhecimento de padrões. Uma solução para o terceiro problema serve para treinar um modelo para que ele possa ser utilizado nos dois primeiros problemas. Para calcular a probabilidade de O ter sido gerada por λ, é utilizado o algoritmo *forward-backward* [38]. Para construir a sequência de estados mais provável de gerar O, existe o algoritmo de Viterbi [52]. Esse algoritmo utiliza uma idéia semelhante a do algoritmo de Needleman-Wunsch. O ajuste de parâmetros de um HMM pode ser feito utilizando o algoritmo de Baum-Welch [5].

Existem algumas variantes do HMM comum (do qual falamos até este momento) que são mais adequadas para a aplicação em problemas específicos. Dissemos anteriormente que a probabilidade de transição para um determinado estado depende apenas do estado anterior a ele. Isso caracteriza um HMM de **primeira ordem**. Um HMM onde a transição para um determinado estado depende não só do anterior, mas dos k estados anteriores é chamado de HMM de $k^{\mathbf{a}}$**ordem**. Um outro tipo de HMM bastante utilizado é o **modelo oculto de Markov generalizado** ou (**GHMM**), onde diferentemente do HMM comum, não há transições de um estado para ele mesmo, mas sim um conjunto

explícito de durações dos estados, que contém a probabilidade do GHMM permanecer em um determinado estado por uma quantidade específica de tempo. Dessa forma, é possível ter um maior controle sobre os casos onde é necessário obter várias observações de um mesmo estado, pois no HMM comum a probabilidade de se permanecer em um estado tem sempre a forma de uma função exponencial (a probabilidade de se permanecer em um estado e durante t transições é dada por $P(e|e)^t$, onde $P(e|e)$ é a probabilidade de transição do estado e para si mesmo). Mais informações sobre HMMs e GHMMs podem ser encontradas em [38].

2.3 Conceitos de Bioinformática

A Bioinformática é uma área de estudos que busca soluções para problemas da Biologia Molecular utilizando conhecimentos provenientes da Ciência da Computação. Isso é possível dadas algumas relações entre determinadas estruturas biológicas e certos conceitos computacionais. Um exemplo de relação que possibilita o desenvolvimento de ferramentas computacionais para problemas biológicos é aquela que associa um DNA a uma sequência de caracteres construída sobre o alfabeto $\Sigma = \{A, C, G, T\}$ formado pelas iniciais A, C, G, T das bases Adenina, Citosina, Guanina e Timina, respectivamente.

Há uma grande quantidade de problemas da Bioinformática propostos na literatura. Alguns desses problemas são:

- Montagem de fragmentos: dados vários fragmentos sobrepostos provenientes de uma molécula de DNA, montá-los no intuito de obter a sequência dessa molécula;
- Construção de árvores filogenéticas: construir árvores filogenéticas a partir de certas características das espécies;
- Comparação de sequências: identificar os trechos semelhantes entre duas ou mais sequências de DNA, RNA ou aminoácidos;
- Rearranjo de genomas: identificar um conjunto de operações de rearranjo que transformem um genoma em outro;
- Predição da estrutura molecular: predizer a estrutura secundária ou terciária de uma molécula de DNA ou RNA a partir de sua estrutura primária;
- Problema da identificação de genes: encontrar as posições de início e fim dos genes de uma sequência de DNA. Além disso determinar quais porções destes genes são éxons e quais porções são íntrons.

Mais detalhes sobre os problemas citados, assim como outros problemas, podem ser encontrados em [36] e [41].

Capítulo 3

O Problema da Identificação de Genes

Nosso trabalho tem como objetivo estudar um problema da Biologia Molecular, conhecido como o problema da identificação de genes, no intuito de desenvolver soluções computacionais efetivas para ele. Esse problema será abordado neste capítulo, que se inicia com a definição, motivação e principais dificuldades encontradas na tentativa de solucioná-lo. Na segunda parte deste capítulo falaremos sobre os principais métodos utilizados para tratá-lo. Por fim, detalharemos o funcionamento de algumas das principais ferramentas de identificação de genes disponíveis na literatura.

3.1 Considerações sobre o problema da identificação de genes

No **Problema da Identificação de Genes** ou **Problema da Predição de Genes**, dada uma sequência de DNA, chamada de **sequência alvo**, devemos encontrar as posições de início e de término dos seus genes, assim como dos seus éxons e íntrons. Uma sequência cujos genes, éxons e íntrons tem suas posições identificadas e que tem suas funções biológicas determinadas é dita **anotada**.

Localizar os genes dentro de uma sequência de DNA para posteriormente relacioná-los às proteínas que sintetizam é uma tarefa de inquestionável importância prática para diversas áreas do conhecimento humano. Na medicina, por exemplo, conhecer os genes de um parasita como o *Trypanossoma cruzi* pode levar ao entendimento dos mecanismos que este organismo utiliza para se proteger do sistema imune do hospedeiro, auxiliando na busca pela cura da doença de Chagas [37]. Um outro exemplo no contexto da medicina é no diagnóstico do Câncer, onde a classificação de um tumor pode ser realizada com base na ocorrência de certos genes [4, 20, 39]. Um último exemplo da importância do problema está ligado à existência nos vegetais de genes resistentes a doenças. Localizar esses genes possibilita selecionar ou modificar espécies para aumentar a sua produtividade [6, 58].

Infelizmente, em organismos eucariotos, distinguir os genes do resto do material genético é uma tarefa difícil. Um dos motivos para essa dificuldade é que a maior parte

do DNA corresponde a regiões não codificantes. Além disso, não basta encontrar apenas as posições de início e fim de um gene, mas também as posições de início e fim de seus éxons. Além de se encontrarem separados por longos trechos de nucleotídeos que não são traduzidos em mRNA, os éxons não possuem uma característica determinante que os diferenciem do restante da sequência. Ainda assim, os genes apresentam algumas particularidades que, apesar de não serem exatas, possibilitam o desenvolvimento de modelos teóricos e métodos determinísticos para a sua identificação.

3.2 Modelos e métodos para o problema da identificação de genes

Os métodos de predição de genes baseiam-se no princípio de conservação das bases[1] e na existência de sinais relacionados à transcrição e à tradução (sítios de início, fim, doação, etc) para tentar identificar os limites dos genes dentro de uma sequência de DNA. Os principais métodos para identificação de genes dividem-se em duas categorias: **métodos intrínsecos** e **métodos extrínsecos**.

3.2.1 Métodos intrínsecos

Nos métodos intrínsecos, apenas informações contidas na sequência alvo são utilizadas na busca pelos seus genes. Os métodos intrínsecos mais comuns de identificação de genes baseiam-se em **dados estatísticos** (métodos estatísticos) e/ou na **ocorrência de sinais** (métodos de busca por sinais). Os métodos intrínsecos são conhecidos também como métodos *ab initio*.

Métodos estatísticos

A identificação de genes através de dados estatísticos baseia-se em características estatísticas presentes nos éxons e íntrons. Uma determinada característica encontrada somente em algumas regiões específicas do DNA e que não se repete em outras regiões pode ser transformada em uma métrica que determina quão parecidas são duas regiões, em relação àquela característica. Um exemplo disso é a maior frequência de bases Gs e Cs em éxons do que em íntrons, nos genes do homem [29]. As métricas mais comuns para a identificação de éxons são as frequências de:

- Códons: é sabido que os códons não encontram-se distribuídos de maneira aleatória no DNA e que os genes possuem frequências altas de alguns códons;
- Aminoácidos: essa abordagem é derivada das frequências diferenciadas de códons, já que cada códon representa um aminoácido;
- Hexamers: um hexamer é uma cadeia de seis nucleotídeos. De acordo com [54] é uma das métricas mais adequadas para a identificação de éxons.

[1] Esse princípio foi abordado na Seção 2.1.7

Dada uma métrica específica, podemos pontuar um trecho qualquer de uma sequência de DNA e usar essa pontuação na discriminação desse trecho em relação a outras partes da sequência. Suponha que queremos avaliar se um trecho de uma sequência de DNA é parte de um éxon, utilizando a frequência de códons. Fazemos isso somando as frequências dos seus códons. Se a soma das frequências dos códons comuns em éxons for alta para aquele trecho da sequência de DNA, significa que ele provavelmente é um éxon. Para utilizar a frequência de aminoácidos, bastaria traduzir os códons daquele trecho da sequência de DNA em aminoácidos e somar a frequência de cada um deles. Na prática, a identificação de possíveis éxons em uma sequência alvo é feita deslizando-se uma "janela"ao longo da sequência, calculando a pontuação de cada um dos trechos da sequência sobre as quais a janela passa e comparando seus valores com um certo valor limite.

As características estatísticas costumam ser identificadas analisando-se as sequências anotadas dos genes do organismo de interesse. Infelizmente, há espécies para as quais não se tem uma quantidade mínima de sequências anotadas capaz de evidenciar uma ou mais características estatísticas. Para resolver esse problema, existem métricas que não foram definidas utilizando informações provenientes de uma amostra particular, como as de Entropia, Assimetria de Posição, entre outras. Essas métricas, que independem do estudo de uma amostra específica de sequências anotadas, também são baseadas no princípio de conservação das bases, assim como os métodos que utilizam amostras de sequências.

Os primeiros trabalhos de identificação de genes utilizando dados estatísticos sobre o DNA foram publicados por Fickett em [16], Shepherd em [42] e Staden e McLachlan em [46]. Há uma compilação dos principais métodos estatísticos de identificação de genes em [54].

Métodos de busca por sinais

Os métodos baseados na ocorrência de sinais utilizam informações sobre a ocorrência de nucleotídeos, ou padrões de nucleotídeos, em regiões que contém possíveis sinais, como o promotor, o terminador e os sítios de doação e de aceitação. Assim como nos métodos estatísticos, é necessária uma amostra contendo sequências anotadas para a construção de uma métrica que determine a probabilidade de um trecho da sequência ser um sinal verdadeiro.

Os primeiros métodos de busca por sinais consistiam em construir um alinhamento múltiplo de trechos de sequências anotadas contendo um sinal, e obter a sequência consenso desse alinhamento múltiplo. As regiões de uma sequência alvo mais parecidas com essa sequência consenso eram considerados como sinais verdadeiros (do mesmo tipo do sinal cujas sequências foram alinhadas). Note que a sequência consenso é um modelo representativo limitado de todos os trechos de sequências utilizados como referência.

Um outro modelo representativo para as sequências utilizadas como referência, menos limitado que a sequência consenso, é a **matriz de peso**, proposta por Staden em [45]. Uma matriz de peso contém as frequências de cada nucleotídeo em cada uma das posições na região de um sinal. Para obter a frequência de cada nucleotídeo na região de um sinal, assim como na sequência consenso, também é necessário construir um alinhamento múltiplo de trechos de sequências anotadas contendo exemplos daquele

sinal. Após a construção do alinhamento múltiplo, a frequência de cada nucleotídeo em cada posição é determinada pela proporção com que cada nucleotídeo aparece em uma coluna (cada coluna do alinhamento múltiplo corresponde a uma posição próxima do sinal, ou a uma posição dentro do sinal). A matriz de peso é aplicada deslizando-se uma "janela"ao longo da sequência, assim como é feito nos métodos estatísticos. Os trechos da sequência de DNA onde a pontuação da janela for alta provavelmente são sinais verdadeiros.

Da forma como descrevemos a construção da matriz de peso, note que ela desconsidera possíveis dependências entre nucleotídeos vizinhos. O fato é que essas dependências existem e podem ser úteis na classificação de um sinal. O método apresentado por Zhang e Marr em [59] é chamado de *weight array matrix* e calcula a probabilidade de um nucleotídeo ocorrer em uma determinada posição, levando em consideração qual era o nucleotídeo presente na posição anterior. Existem ainda alguns métodos que relacionam nucleotídeos em posições não-adjacentes, como o de modelos de máxima entropia e redes Bayseanas apresentados respectivamente em [57] e [11]. Uma das limitações das matrizes de peso é a impossibilidade de modelar sequências de comprimento variável.

Há ainda uma outra abordagem para a busca por sinais baseada no uso do modelo oculto de Markov, descrito no Capítulo 2. Um HMM pode ser construído através de um alinhamento múltiplo de forma a representar o sinal contido nas sequências desse alinhamento. O uso de um HMM na busca por sinais é interessante pois vai além da busca em si. É possível construir um HMM que não somente representa os sinais, mas também modela a estrutura de éxons e íntrons dos genes. Além disso um HMM possibilita uma modelagem mais adequada de sequências de comprimento variável, ao contrário das matrizes de peso. A maioria dos métodos de identificação de genes baseados em HMM segue uma sequência de passos padrão, dada abaixo:

1. Desenvolve-se um conjunto de estados que representam os diferentes trechos de uma sequência de DNA, como éxons, íntrons, regiões intergênicas, regiões de promotores, etc;

2. Criam-se transições biologicamente consistentes entre os estados, como por exemplo uma transição que sai de um éxon para um sítio de doação;

3. Obtém-se as distribuições de probabilidade do modelo (transição entre os estados, emissão de observações e escolha de estado inicial) através do estudo de amostras (sequências de DNA) da espécie escolhida como alvo do estudo;

4. Executa-se o algoritmo de Viterbi sobre o HMM especificado nos passos anteriores e a sequência alvo. A sequência de estados obtida é sobreposta à sequência alvo e o tipo dos estados sobre cada nucleotídeo da sequência corresponde à possível classificação desse nucleotídeo (como pertencente a um éxon ou não).

Observe que os três primeiros passos só precisam ser executados uma vez e fazem parte da fase de desenvolvimento de uma ferramenta. Uma melhor descrição sobre a aplicação de HMMs na identificação de genes pode ser encontrada em [25].

Mais informações sobre os métodos de busca por sinais podem ser encontradas em [18].

3.2.2 Métodos extrínsecos

Os métodos extrínsecos utilizam informações de outras sequências na tentativa de identificar regiões codificantes na sequência de DNA alvo. Essa classe de métodos realiza comparações entre a sequência alvo e sequências anotadas. As abordagens mais comuns no contexto dos métodos extrínsecos utilizam variações dos algoritmos de alinhamento de sequências que apresentamos na Seção 2.2. Uma dessas variações corresponde a algoritmos que encontram vários alinhamentos locais entre duas sequências.

Graças ao princípio de conservação das bases, as regiões comuns entre as sequências provavelmente serão bons pontos de partida para a identificação de genes. Mais especificamente, um trecho da sequência alvo apontado pelo método como sendo semelhante a um gene de uma sequência anotada tem boas chances de também ser um gene.

Vários tipos de sequências anotadas podem ser utilizadas na tarefa de localizar os genes por meio de comparação. Dentre elas estão as sequências de DNAs, cDNAs e ESTs. Algumas das ferramentas baseadas em métodos extrínsecos de identificação de genes são o `PROCRUSTES` [19] e o `PROGEN` [33].

Apesar de serem chamados de métodos de identificação de genes, a maioria dos métodos intrínsecos e extrínsecos, isoladamente, não indicam qual região corresponde a um gene, mas quais regiões são boas candidatas a éxons de um gene. Após a identificação de um conjunto de possíveis éxons, é necessário realizar uma montagem desses elementos para se chegar a solução final do problema. Ou seja, aos limites dos genes procurados. Na próxima seção apresentamos uma breve descrição de algumas ferramentas de identificação de genes em sequências de DNA.

Neste trabalho, estamos interessados no problema da identificação de genes por comparação de sequências. Mais especificamente na identificação de genes por comparação de DNA com cDNA(s). Essa abordagem será detalhada no capítulo a seguir.

3.3 Algumas ferramentas de identificação de genes

As ferramentas de identificação de genes também são classificadas, basicamente, em duas categorias. Essa classificação é feita de acordo com o método que utilizam para a identificação de genes. As ferramentas que utilizam apenas informações contidas na sequência alvo, ou seja, aquelas baseadas em métodos intrínsecos, formam a primeira categoria e geralmente são chamadas de ferramentas ***ab initio***. A segunda categoria de ferramentas abrange aquelas que utilizam métodos extrínsecos e geralmente são referenciadas como ferramentas **comparativas** ou **baseadas em homologia**. Na prática, muitas ferramentas de identificação de genes não se limitam a apenas um dos métodos, baseando-se tanto nos métodos intrínsecos quanto nos extrínsecos para realização de suas tarefas. Essas ferramentas podem ser classificadas em uma categoria separada das duas anteriores.

Apresentamos a seguir uma descrição geral das ferramentas utilizadas no nosso trabalho. Os programas `AUGUSTUS+`, `EST2GENOME`, `SIM4`, `GENESEQER`, `PROCRUSTES`, `SPIDEY` e `TWINSCAN` foram comparados às ferramentas que propusemos, das quais falaremos mais adiante. O algoritmo do `PROCRUSTES` foi utilizado como parte de algumas heurísticas

que propusemos, por isso ele é descrito separadamente e com mais detalhes no Capítulo 4. O GENSCAN foi utilizado em nossas ferramentas e uma descrição dele está contida na Seção 3.3.6.

3.3.1 AUGUSTUS

O AUGUSTUS é uma ferramenta *ab initio* baseada em um GHMM e descrita por Stanke e Waack em [48]. Ele recebe uma sequência de DNA como entrada e executa o algoritmo de Viterbi sobre a sequência e um GHMM de 47 estados que modelam as diferentes regiões do DNA. O resultado da execução do algoritmo de Viterbi é a sequência mais provável de estados que gera a sequência de DNA. Os estados correspondem a regiões intergênicas, éxons, íntrons e sítios de aceitação e de doação, em ambas as fitas do DNA. As transições entre os estados respeitam determinadas regras biológicas, como por exemplo, a de que um íntron deve vir depois de um sítio de doação. Dessa forma, a sequência de estados do GHMM escolhida pelo algoritmo de Viterbi define as prováveis posições dos possíveis genes, éxons e íntrons na sequência alvo.

Uma característica interessante do AUGUSTUS, que o diferencia de outras ferramentas que utilizam HMM, é a abordagem que ele utiliza para modelar os íntrons. Nas ferramentas baseadas em HMM mais conhecidas, como o GENSCAN [9] e o TWINSCAN [24], os íntrons são modelados utilizando uma distribuição geométrica de probabilidades de duração, que se comporta de modo semelhante a um estado sem uma distribuição explícita de probabilidades de duração. O problema dessa abordagem é que a distribuição geométrica sempre atribui a maior probabilidade para a duração mais curta possível. Isso dificulta uma modelagem precisa de íntrons. O uso de uma distribuição explícita de probabilidades de comprimento da sequência emitida para este estado, ao invés de uma distribuição geométrica, resolveria esse problema. Isso infelizmente tornaria os tempos de execução dos algoritmos *forward* e de Viterbi impraticáveis para sequências longas.

A solução que o AUGUSTUS implementa para lidar com os comprimentos dos íntrons é o uso combinado de estados com e sem distribuição explícita de probabilidades de comprimento da sequência emitida. Os íntrons curtos são emitidos por um estado com distribuição explícita de probabilidades de comprimento da sequência emitida. Os íntrons longos utilizam a outra abordagem, com a probabilidade de transição para si mesmo ajustada para melhor identificá-los.

Em [47], Stanke *et al.* apresentaram o AUGUSTUS+, que é uma versão do AUGUSTUS que pode utilizar informações externas à sequência para identificar os seus genes. Essas dicas são evidências sobre a existência e a estrutura de supostos genes na sequência alvo e provêm de diversas fontes, como alinhamentos de ESTs, DNAs e sequências de proteínas com a sequência alvo. As probabilidades de transição entre os estados são modificadas nos casos onde existe uma informação adicional, diminuindo a probabilidade de transições incompatíveis com ela e aumentando a probabilidade de transições compatíveis com a informação.

3.3.2 EST_GENOME

O EST_GENOME é uma ferramenta desenvolvida por Mott, descrita em [28]. Ele recebe como entrada uma sequência de DNA (sequência alvo) e uma EST. Esse programa utiliza uma variação do algoritmo de alinhamento local de Smith-Waterman [44] que trata de forma específica (menor penalização) regiões da sequência de DNA que supostamente correspondem a íntrons.

A menor penalização das regiões correspondentes a possíveis íntrons é feita armazenando-se a pontuação B do melhor alinhamento local que termina no início de um provável íntron. A recorrência do algoritmo de Smith-Waterman é modificada, incluindo a opção de utilizar o valor de B, acrescido de uma pontuação para um íntron. Com isso, é possível ignorar o valor de um alinhamento local ruim, decorrente do alinhamento de um íntron da sequência de DNA com a EST, obtido desde o momento em que B foi modificado pela última vez.

No EST_GENOME, existem duas pontuações distintas para íntrons, que são o `intron` e o `splice`. O valor de `splice` é utilizado quando sítios de doação e de aceitação estão presentes na sequência de DNA nas extremidades do possível íntron. Caso contrário é utilizado o valor de `intron`. Para um melhor entendimento de seu uso, suponha que a pontuação para a inserção de um espaço em qualquer uma das sequências seja dado por `space`. Se houver um íntron de comprimento l na sequência de DNA, ao invés de adicionar ao melhor alinhamento local o valor de $l * \texttt{space}$, é possível utilizar o valor de `intron` (ou `splice` se o íntron possuir os sítios de doação e aceitação), caso $l * \texttt{space} < \texttt{intron}$. O valor de `intron` é menor que o valor de `splice`, o que prioriza a identificação correta dos sítios de aceitação e de doação. A recorrência utilizada no EST_GENOME pode ser vista na Recorrência 3.1.

$$
\begin{aligned}
A[i][j] &= \max \begin{cases} A[i-1][j] + \omega(s[i], \text{'}-\text{'}) \\ A[i][j-1] + \omega(t[j], \text{'}-\text{'}) \\ A[i-1][j-1] + \omega(s[i], t[j]) \\ B \\ 0 \end{cases} \\
B &= \begin{cases} B[i] + \texttt{splice} & \text{se há um par de sítios de aceitação/doação em C[i] e j} \\ B[i] + \texttt{intron} & \text{caso contrário} \end{cases} \\
(B[i], C[i]) &= \begin{cases} (A[i][j], j) & \text{se } A[i][j] > B[i] \\ (B[i], C[i]) & \text{caso contrário} \end{cases}
\end{aligned}
\tag{3.1}
$$

3.3.3 SIM4

O SIM4 foi desenvolvido por Florea *et al.* e encontra-se descrito em [17]. O SIM4 recebe como entrada uma sequência de DNA (sequência alvo) e uma de cDNA ou EST (que os autores chamam de sequência expressa). Essa ferramenta atua em quatro etapas.

Na primeira etapa, o SIM4 encontra todos os alinhamentos exatos[2] de comprimento doze e os estende em ambas as direções com pontuações 1 para match e -5 para mismatch, parando essa extensão quando ela deixar de aumentar o valor da pontuação total do alinhamento. Essas regiões de alta pontuação e sem buracos entre as duas sequências são chamadas de *high-scoring segment pairs* ou HSPs. O algoritmo que encontra os HSPs foi apresentado por Schwartz *et al.* em [40].

A segunda etapa consiste em escolher um conjunto de HSPs que possa representar um gene. Isso é feito por meio de um algoritmo de programação dinâmica que escolhe a melhor sequência de HSPs tais que suas posições de início estão em ordem crescente na sequência expressa e HSPs consecutivos encontram-se próximos o bastante para fazerem parte de um mesmo éxon ou suficientemente distantes para admitir um íntron. A cada sequência de HSPs é atribuída uma pontuação dada pela soma da pontuação dos seus HSPs multiplicada por 100, menos a soma das distâncias entre HSPs consecutivos da sequência.

Após a seleção dos HSPs, é realizado um pós-processamento onde os HSPs são estendidos ou aparados até que sítios de aceitação ou doação sejam alcançados na sequência alvo. Os trechos de sequências definidos nesse passo correspondem aos possíveis éxons.

Na última etapa, são obtidos os alinhamentos individuais para cada possível éxon determinado no passo anterior, utilizando o método proposto por Chao *et al.* em [12].

3.3.4 GENESEQER

Em [50], Usuka *et al.* apresentaram a ferramenta GENESEQER, que utiliza uma subrotina chamada SAHMTD (*Spliced Alignment Hidden Markov Tool for cDNA*) para alinhar uma sequência de DNA com uma sequência de cDNA, dadas como entrada. O SAHMTD utiliza um algoritmo de programação dinâmica sobre um HMM para alinhar as duas sequências.

Sejam g e c as sequências de DNA e de cDNA dadas como entrada, respectivamente, com $|g| = n$ e $|c| = m$. Considere que Σ é o alfabeto contendo os caracteres das sequências g e c. O HMM do SAHMTD é gerado com um conjunto Q de $2n$ estados, divididos em um conjunto de estados que representam éxons $E = \{e_1, e_2, ..., e_n\}$ e um conjunto de estados que representam íntrons $I = \{i_1, i_2, ..., i_n\}$. Cada $e \in E$ emite uma coluna de alinhamento do tipo $\binom{x}{y}$ e cada $i_k \in I$ emite uma coluna de alinhamento do tipo $\binom{g_k}{\star}$, onde x e $y \in \Sigma \cup \{-\}$, $g_k \in \Sigma$ e $\star$ é um símbolo que representa os trechos de íntrons (espaços no cDNA).

As probabilidades de transição entre os estados são calculadas utilizando os valores de $P_{D(x)}$ e $P_{A(x)}$, que são as probabilidades do nucleotídeo x ser o primeiro nucleotídeo de um sítio de doação ou o último nucleotídeo de um sítio de aceitação, respectivamente. Estes valores são provenientes da ferramenta SPLICEPREDICTOR [8]. As probabilidades de emissão de cada coluna do alinhamento por um estado são substituídas por uma pontuação, similar à utilizada nos algoritmos de alinhamento de sequências.

Um alinhamento qualquer de g e c possui uma sequência de estados de Q associada a ele, que é denotada por $q = q_1, q_2, \ldots, q_l$, com $\max\{m, n\} \leq l \leq m+n$. No algoritmo do

[2] Alinhamentos exatos são alinhamentos onde não ocorre nenhum mismatch ou space.

SAHMTD, um alinhamento ótimo é uma sequência de estados $q = q_1, q_2, \ldots, q_l$, com um alinhamento S_M^N associado a ela, de tal forma que a probabilidade conjunta $P(q, S_M^N)$ é máxima dentre todos os possíveis pares q e S_M^N. Esse alinhamento é encontrado através de um algoritmo de programação dinâmica. Durante a execução deste algoritmo, são armazenadas as transições de maior pontuação entre cada par de estados, para a construção do caminho ótimo (e consequentemente do alinhamento entre g e c). As regiões desse alinhamento correspondentes aos estados do conjunto E são os possíveis éxons da sequência de DNA dada como entrada.

3.3.5 SPIDEY

A ferramenta SPIDEY foi desenvolvida e apresentada por Wheelan *et al.* em [55]. Ela recebe uma sequência de DNA (sequência alvo) e um conjunto de sequências de mRNA como entrada. Apesar de receber várias sequências de mRNA como entrada, ela as compara uma por uma com a sequência alvo, ou seja, é equivalente a executá-lo uma vez para cada elemento do conjunto de mRNAs com a mesma sequência alvo. A idéia geral do funcionamento do SPIDEY, descrita a seguir, refere-se à comparação de uma sequência de DNA com uma sequência de mRNA.

O SPIDEY inicialmente utiliza o BLAST [3] para encontrar bons alinhamentos locais entre a sequência de DNA e o mRNA. Estes alinhamentos locais são então ordenados por suas pontuações e associados a janelas na sequência alvo. Uma janela corresponde a um par de índices na sequência de DNA, que determinam um início e um fim. As posições de início e fim de uma janela são definidas de acordo com os alinhamentos locais nela contidos, sendo que o índice de início de uma janela é o mesmo do início do alinhamento local situado mais à esquerda na sequência alvo, e o índice do fim da janela é o mesmo do fim do alinhamento local situado mais à direita na sequência alvo. O primeiro melhor alinhamento é inserido em uma janela e, depois disso, percorre-se a lista ordenada procurando-se por todos os alinhamentos consistentes com ele (mesma orientação do mRNA, índices não sobrepostos e linearmente consistentes na sequência alvo e no mRNA). Esses alinhamentos são inseridos na mesma janela. Os alinhamentos restantes são processados da mesma maneira que o primeiro melhor e inseridos em suas respectivas janelas. As janelas não devem se sobrepor. A intenção é fazer com que cada janela contenha um modelo de gene.

No próximo passo, cada uma das janelas é alinhada com todo o mRNA, utilizando o BLAST com uma tolerância maior que na primeira execução. Um novo conjunto de alinhamentos é gerado (o conjunto anterior de alinhamentos é descartado) e então um algoritmo guloso é utilizado para escolher um subconjunto de melhor pontuação, de alinhamentos não sobrepostos, provenientes da segunda execução do BLAST. Realiza-se uma varredura em busca de lacunas entre os alinhamentos do subconjunto, em relação ao mRNA. Caso alguma seja encontrada, o BLAST é executado para alinhar esta lacuna no DNA. O SPIDEY realiza ajustes nos alinhamentos do subconjunto, unindo alinhamentos próximos (para formar um éxon por exemplo) e procurando por sítios de doação e de aceitação, de forma que no final cada um destes alinhamentos corresponda a um éxon.

Ao final, cada possível gene contido em uma janela é avaliado quanto às suas características e devolvido como um gene, caso sua pontuação esteja acima do mínimo definido pelo usuário.

3.3.6 TWINSCAN e GENSCAN

O TWINSCAN, apresentado por Korf *et al.* em [24], é uma ferramenta de identificação de genes híbrida, já que utiliza um GHMM (método intrínseco) aliado a informações extrínsecas à sequência. O GHMM utilizado pelo TWINSCAN é uma versão modificada do GHMM da ferramenta GENSCAN, proposta por Burge e Karlin em [9]. Por isso, vamos descrever o GENSCAN antes de explicar o funcionamento do TWINSCAN.

O GENSCAN recebe uma sequência de DNA como entrada e funciona de acordo com os passos citados ao final da Seção 3.2.1. Como o GENSCAN utiliza um GHMM ao invés de um HMM, o algoritmo de Viterbi é modificado para encontrar também uma sequência de durações associadas aos estados.

O GENSCAN associa cada nucleotídeo da sequência alvo a uma categoria geral: promotor, 5′UTR, éxon, íntron, 3′UTR, sinal poly-A e intergênico. Seu GHMM é ilustrado na Figura 3.1.

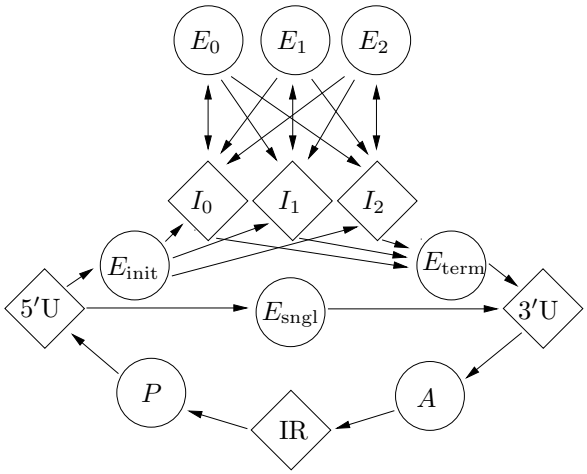


Figura 3.1: Alguns dos estados do GHMM do GENSCAN.

Os estados E_0, E_1 e E_2 representam éxons internos de acordo com a posição em que interrompem seus últimos códons. Os estados I_0, I_1 e I_2 representam íntrons de acordo com a posição onde os éxons que os precedem interromperam seus últimos códons. E_{sngl}, E_{init} e E_{term} representam éxons únicos, iniciais e finais, respectivamente. O estado P representa a região do promotor e o estado A representa o sinal de poliadenilação, que nesse modelo substitui o terminador. O estado 5′U representa o trecho do DNA que fica entre o promotor e o primeiro éxon. O estado 3′U representa o trecho do DNA que fica entre o último éxon e o sinal de poliadenilação. Por fim, o estado IR representa as regiões intergênicas. As setas representam transições, com probabilidade não-nula, em uma das fitas do DNA. Um segundo modelo análogo é utilizado para modelar as diferentes regiões e sinais na outra fita.

O TWINSCAN recebe como entrada um par de sequências de DNA, onde uma delas é a sequência alvo e a outra é uma sequência homóloga à sequência alvo, chamada de sequência informante. No TWINSCAN foi introduzido o conceito de sequência de conservação, que é uma sequência com caracteres que representam *matches*, *mismatches* e *spaces* de um alinhamento de duas sequências. Essa sequência é construída por meio

de um algoritmo que mescla HSPs provenientes de uma execução do `BLASTN` entre as sequências alvo e informante.

Conforme dissemos anteriormente, o GHMM do `TWINSCAN` é baseado no do `GENSCAN`. A diferença entre eles está no fato de que no `TWINSCAN`, o GHMM foi modificado para gerar uma sequência de conservação, além da sequência de DNA. As probabilidades de emissão para cada uma das sequências em cada estado são independentes entre si.

Dadas uma sequência de DNA e uma sequência de conservação, é possível calcular a probabilidade conjunta daquele par de sequências ter sido gerada por uma sequência de estados do GHMM. Com este modelo, que possui informações adicionais, utiliza-se o algoritmo de Viterbi para classificar os trechos das sequências alvo e de conservação em relação ao seu significado biológico.

Capítulo 4

Identificação de Genes por Comparação de DNA com cDNA

Como dissemos no Capítulo 2, as moléculas de cDNA obtidas a partir do mRNA contêm apenas as regiões codificantes do trecho da sequência de DNA que deu origem ao mRNA. Com essa informação, um cDNA constitui uma fonte de informação valiosa na busca pelos genes de uma dada sequência de interesse. Neste capítulo falaremos mais sobre a identificação de genes pela comparação de um DNA com um cDNA, e apresentaremos uma extensão dessa idéia que é a comparação de um DNA com vários cDNAs. Ao final, propomos um problema matemático para modelar o problema da identificação de genes por comparação de um DNA com vários cDNAs e demonstramos que ele é NP-completo para uma função de pontuação específica.

4.1 Identificação de genes por comparação de um DNA com um cDNA

A tarefa de identificação de genes em uma sequência de DNA através de sua comparação com um cDNA pode ser modelada através de um problema matemático denominado problema do alinhamento *spliced*, detalhado a seguir.

4.1.1 O problema do alinhamento *spliced*

O problema do alinhamento *spliced* foi proposto por Gelfand *et al.* em [19]. Para uma melhor compreensão desse problema, considere as seguintes definições. Seja s uma sequência qualquer. Dizemos que um segmento $s_1 = s[i..j]$ de s **antecede** um outro segmento $s_2 = s[k..l]$, também de s, se $j < k$, e denotamos essa relação por $s_1 \prec s_2$. Dessa forma, se tivermos um conjunto $\Gamma = \{s_1, s_2, ..., s_n\}$ de segmentos de s, onde $s_i \prec s_{i+1}$, para $1 \leq i < n$, dizemos que Γ está ordenado e o chamamos de **conjunto ordenado de segmentos**. A **concatenação** dos segmentos em Γ é dada por $s_1 s_2 ... s_n$ e denotada por Γ^*.

Feitas as definições, o problema do alinhamento *spliced* encontra-se assim definido por Gelfand *et al.* em [19]:

> **Problema do alinhamento *spliced***: Dadas duas sequências g e t construídas sobre um alfabeto Σ, tal que $\{-\} \notin \Sigma$, um conjunto $\mathscr{B}$ de segmentos de g e uma função de pontuação ω, encontrar um conjunto ordenado de segmentos Γ de $\mathscr{B}$ tal que $\text{sim}_\omega(\Gamma^*, t)$ é máxima.

As sequências g, t e os segmentos do conjunto $\mathscr{B}$ são chamados de sequência **base**, sequência **modelo** e **blocos**, respectivamente. Informalmente, podemos dizer que o problema do alinhamento *spliced* consiste em encontrar, dentre todos os possíveis subconjuntos ordenados de blocos (não-sobrepostos) de $\mathscr{B}$, um subconjunto tal que a sequência resultante da concatenação de seus blocos mais se assemelha à sequência modelo. Uma instância do problema com sua respectiva solução podem ser vistos na Figura 4.1.

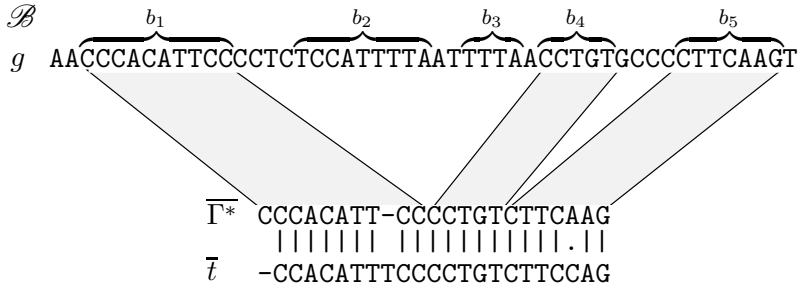


Figura 4.1: Exemplo de uma instância do problema do alinhamento *spliced* e sua respectiva solução. Os segmentos do conjunto $\mathscr{B} = \{b_1, b_2, b_3, b_4, b_5\}$ correspondem aos trechos localizados abaixo das chaves na sequência g e a solução do problema aos blocos b_1, b_4 e b_5.

Podemos mapear o problema da identificação de genes por comparação de um DNA com um cDNA no problema do alinhamento *spliced* utilizando a sequência de DNA como a sequência base, a sequência de cDNA como a sequência modelo e um conjunto com possíveis éxons do DNA como o conjunto $\mathscr{B}$. Caso a sequência de cDNA esteja relacionada a um gene codificado no DNA, e utilizando uma função de pontuação adequada, uma solução para o problema do alinhamento *spliced* corresponde a uma resposta plausível para o problema da identificação de genes por comparação de DNA com cDNA. Isso pode ser justificado com base no princípio de conservação das bases e no fato do cDNA não possuir íntrons, indicando que provavelmente os blocos de $\mathscr{B}$ que correspondem aos éxons reais serão utilizados na solução do problema.

No mesmo trabalho onde o problema do alinhamento *spliced* foi definido, Gelfand *et al.* apresentaram uma solução para ele baseada em programação dinâmica. Antes de mostrar a recorrência principal deste algoritmo, enunciamos algumas definições.

Seja $b_k = g[l..m]$ um segmento de g contendo uma posição i dessa sequência, com $l \leq i \leq m$. Um segmento da forma $g[l..i]$ é chamado de i-prefixo de b_k e o denotamos por $b_k[i]$. Para um bloco $b_k = g[l..m]$, definimos $\mathit{first}(k) = l$, $\mathit{last}(k) = m$ e $\mathit{size}(k) = m-l+1$. Seja $\Gamma = \{b_1, ..., b_k, ..., b_t\}$ um conjunto ordenado de segmentos de g que inclui o bloco b_k (lembrando que b_k contém a posição i). Definimos ainda Γ_i^* como sendo a sequência

$\Gamma_i^* = b_1 b_2 ... b_k[i]$ e Γ^k como o conjunto de todos os conjuntos ordenados de segmentos contendo o bloco b_k. Considere uma função de pontuação ω qualquer. Dadas essas definições, considere uma matriz tridimensional S onde

$$S[i][j][k] = \max_{\Gamma^k} \text{sim}_\omega(\Gamma_i^*, t[1..j]), \tag{4.1}$$

para $1 \leq i \leq |g|$, $1 \leq j \leq |t|$ e $1 \leq k \leq |\mathscr{B}|$, ou seja, $S[i][j][k]$ contém, dentre todas as possíveis similaridades entre a concatenação de um conjunto ordenado de segmentos contendo o bloco b_k (até a posição i desse bloco) e a sequência t até a posição j, aquela de valor máximo. Então, a pontuação de um alinhamento *spliced* ótimo estará situado no elemento:

$$\max_k S[last(k)][m][k]. \tag{4.2}$$

Dada uma matriz de alinhamento *spliced* previamente preenchida, podemos encontrar os blocos de um alinhamento *spliced* ótimo partindo do elemento $\max_k S[last(k)][m][k]$ e refazendo, em ordem inversa, as escolhas que resultaram em seu valor. Sempre que houver uma mudança no valor da variável k (que indexa os blocos), inclua o antigo k no conjunto Γ. Ao final, basta devolver Γ, que corresponde à solução procurada.

Seja $\mathscr{B}(i) = \{k : last(k) < i\}$ o conjunto de blocos que terminam estritamente antes da posição i em g. A Recorrência 4.3 computa $S[i][j][k]$ para todas as posições da matriz S.

$$S[i][j][k] = \max \begin{cases} S[i-1][j-1][k] + \omega(g[i], t[j]) & \text{se } i \neq first(k) \\ S[i-1][j][k] + \omega(g[i], \text{'}-\text{'}) & \text{se } i \neq first(k) \\ \max_{l \in \mathscr{B}(first(k))} S[last(l)][j-1][l] + \omega(g[i], t[j]) & \text{se } i = first(k) \\ \max_{l \in \mathscr{B}(first(k))} S[last(l)][j][l] + \omega(g[i], \text{'}-\text{'}) & \text{se } i = first(k) \\ S[i][j-1][k] + \omega(\text{'}-\text{'}, t[j]). \end{cases} \tag{4.3}$$

Detalhando a Recorrência 4.3, ela apresenta três possibilidades para preenchimento de $S[i][j][k]$ quando o índice i encontra-se no início do bloco k ($i = first(k)$) e três possibilidades de preenchimento de $S[i][j][k]$ quando i encontra-se depois do início do bloco k ($i \neq first(k)$). Se $i \neq first(k)$ as possibilidades de preenchimento de $S[i][j][k]$ são equivalentes às do alinhamento global de duas sequências visto no Capítulo 2. Uma ilustração dessas três possibilidades de preenchimento pode ser vista na Figura 4.2. Agora, caso $i = first(k)$, estamos iniciando a construção de uma solução que inclui o bloco k, alinhando-o com sequência t a partir da posição j. Aqui, uma das possibilidades para o preenchimento da posição $S[i][j][k]$ é o alinhamento do caractere $t[j]$ com um espaço em Γ^* utilizando o valor de $\text{sim}_\omega(\Gamma_i^*, t[1..(j-1)])$. Outra possibilidade é o alinhamento do caractere $\Gamma_i^*[i]$ com o caractere $t[j]$. Nesse caso, precisamos verificar qual é a melhor solução envolvendo blocos anteriores ao bloco k ($l \in \mathscr{B}(first(k))$) e estendê-la alinhando $\Gamma_i^*[i]$ com $t[j]$. Para isso, utilizamos o valor de $\text{sim}_\omega(\Gamma_{last(l)}^*, t[1..(j-1)])$ onde l é tal que $last(l) < i$ e $\text{sim}_\omega(\Gamma_{last(l)}^*, t[1..(j-1)])$ é máximo para todo $1 \leq l \leq |\mathscr{B}|$. A última possibilidade é o alinhamento do caractere $\Gamma_i^*[i]$ com um espaço em t utilizando o

valor de $\text{sim}_\omega(\Gamma^*_{last(l)}, t[1..j])$ onde l é tal que $last(l) < i$ e $\text{sim}_\omega(\Gamma^*_{last(l)}, t[1..j])$ é máximo para todo $1 \leq l \leq \mathscr{B}$. Uma ilustração das três possibilidades de preenchimento de $S[i][j][k]$ quando $i = first(k)$ pode ser vista na Figura 4.3.

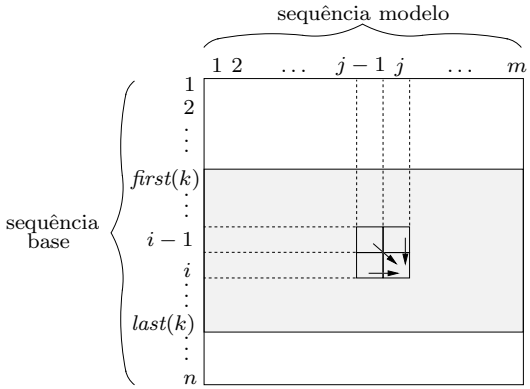


Figura 4.2: Preenchimento da posição $S[i][j][k]$ quando $i \neq first(k)$. Apenas a matriz bidimensional referente ao bloco k está representada. As setas indicam quais posições podem ter seus valores utilizados para calcular o valor de $S[i][j][k]$. A região sombreada representa a região da matriz S coberta pelo bloco k.

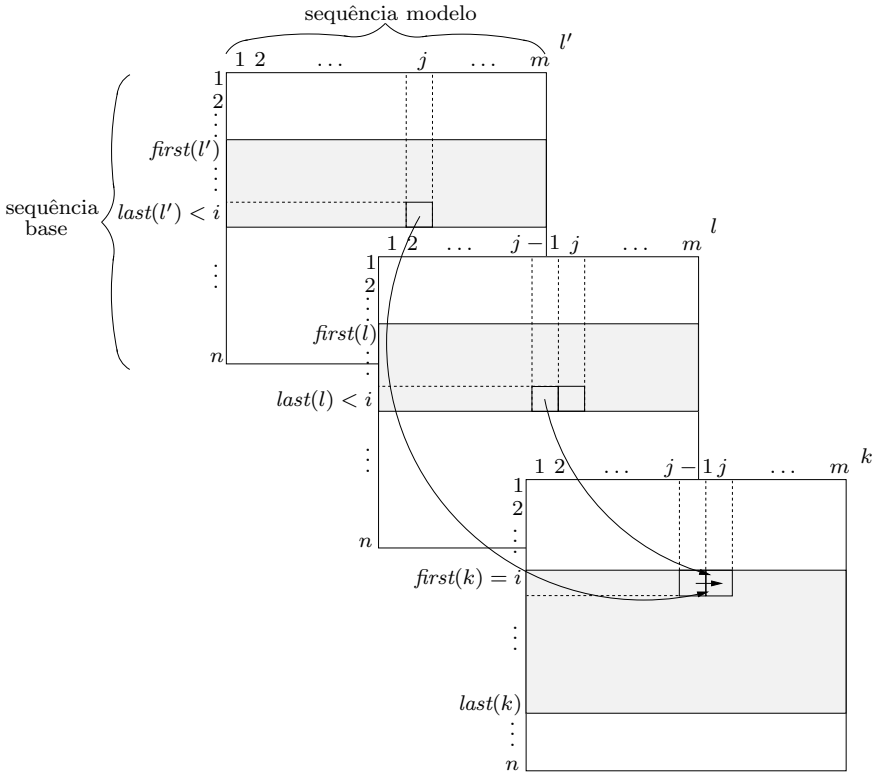


Figura 4.3: Preenchimento da posição $S[i][j][k]$ quando $i = first(k)$. Apenas as matrizes bidimensionais referentes aos blocos l', l e k estão representadas. As setas indicam quais posições podem ter seus valores utilizados para calcular o valor de $S[i][j][k]$. As regiões sombreadas representam regiões contidas em blocos.

Sobre a complexidade de uma solução para o problema do alinhamento *spliced* baseada na Recorrência 4.3, observe que $S[i][j][k]$ só encontra-se definido para $i \in b_k$, pois

a montagem desconsidera os trechos de g que não estão dentro dos blocos. Tomando-se $b = |\mathscr{B}|$, $n = |g|$ e $m = |t|$, a quantidade de entradas de S que devem ser preenchidas é dada por $m \sum_{k=1}^{b} size(k)$. Se considerarmos $c = \frac{1}{n} \sum_{k=1}^{b} size(k)$ como o valor correspondente à cobertura da sequência base pelos blocos em $\mathscr{B}$, uma implementação direta da Recorrência 4.3 terá complexidade de tempo igual a $O(mnc + mb^2)$ e complexidade de espaço igual a $O(mnc)$. O algoritmo de programação dinâmica que resolve o problema do alinhamento *spliced* baseado na Recorrência 4.3 pode ser visto no Algoritmo 4.1.

ALGORITMO 4.1 Algoritmo do alinhamento *spliced*.

ALINHAMENTO_SPLICED$(g, t, \mathscr{B}, \omega)$: recebe uma sequência g de comprimento n, uma sequência t de comprimento m, um conjunto ordenado $\mathscr{B} = \{b_1, b_2, \ldots, b_l\}$ de l segmentos de g, uma função de pontuação ω e devolve um subconjunto Γ de $\mathscr{B}$ tal que $\text{sim}_\omega(\Gamma^*, t)$ é máxima.

1: Preencha a matriz tridimensional S utilizando a Recorrência 4.3, armazenando em uma matriz P, com as mesmas dimensões de S, a opção da recorrência escolhida para preencher cada célula de S;
2: Encontre o k tal que $S[last(k)][m][k]$ é máximo para todo $1 \leq k \leq l$;
3: $\Gamma \leftarrow \{b_k\}$;
4: Percorra a matriz P, partindo do elemento $P[last(k)][m][k]$ e seguindo, de maneira inversa, as escolhas de índices da Recorrência 4.3. Sempre que o índice k mudar, inclua o bloco b_k em Γ;
5: **devolva** Γ;

No mesmo trabalho onde descrevem o problema do alinhamento *spliced* e a solução que utiliza um caminho em um grafo, Gelfand *et al.* em [19] apresentaram uma ferramenta que implementa essa idéia, chamada de `PROCRUSTES`. O `PROCRUSTES` recebe como entrada uma sequência de DNA e uma sequência de proteína. A ferramenta disponibiliza três modos de geração de blocos. O primeiro deles inclui todos os possíveis éxons situados entre potenciais sítios de aceitação e de doação. Os dois outros modos realizam uma triagem dos possíveis éxons sugeridos pelo primeiro modo. Ao final da execução, o `PROCRUSTES` fornece como saída os possíveis éxons identificados na sequência genômica. Esses possíveis éxons, quando concatenados e traduzidos, são os que mais se assemelham à proteína utilizada como sequência modelo.

4.2 Identificação de genes por comparação de um DNA com vários cDNAs

Apesar de apresentarem um certo nível de precisão, as ferramentas de identificação de genes baseadas na comparação de uma sequência de DNA com uma sequência de cDNA ainda realizam predições errôneas, como pode ser visto em [1]. Esta lacuna entre os resultados obtidos atualmente e os desejados nos motiva a desenvolver soluções mais adequadas para esta tarefa.

Observe que a identificação de genes se baseia na busca por evidências da presença de um gene, sejam elas intrínsecas ou extrínsecas à sequência alvo. Partindo dessa observação, podemos supor que a quantidade e a qualidade de evidências sobre a presença de um gene influenciam na dificuldade de encontrá-lo. A partir dessa suposição, uma

hipótese razoável é a de que podemos melhorar a qualidade de uma predição através da comparação de uma sequência de DNA com várias sequências de cDNA (ao invés de com uma única sequência como no trabalho de Gelfand *et al.* em [19]).

4.2.1 O problema do alinhamento *spliced* múltiplo

Antes de começar a desenvolver soluções para a tarefa de comparação de um DNA com vários cDNAs, precisamos de uma formalização matemática adequada para ela. Uma possível formalização pode ser obtida com base no problema do alinhamento *spliced* proposto por Gelfand *et al.* em [19]. Esse novo problema, que chamaremos de **problema do alinhamento *spliced* múltiplo**, encontra-se definido abaixo:

> **Problema do alinhamento *spliced* múltiplo (PASM)**: dados uma sequência g, um conjunto de sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_u\}$, um conjunto $\mathscr{B}$ de segmentos de g e uma função de pontuação ω, encontrar um conjunto ordenado de segmentos Γ de $\mathscr{B}$ tal que $\sum_{i=1}^{u} \mathrm{sim}_w(t_i, \Gamma^*)$ seja a maior possível.

De maneira análoga ao problema do alinhamento *spliced*, chamaremos a sequência g de sequência base, as sequências $t_i \in \mathscr{T}$ de sequências modelo e os segmentos $b_i \in \mathscr{B}$ de blocos. Diferentemente do problema do alinhamento *spliced*, no problema do alinhamento *spliced* múltiplo estamos interessados em encontrar, dentre todos os possíveis subconjuntos ordenados de blocos (não-sobrepostos) de $\mathscr{B}$, um subconjunto tal que a sequência resultante da concatenação de seus blocos mais se assemelha a todas as sequências modelo.

A Figura 4.4 ilustra uma instância do problema do alinhamento *spliced* múltiplo e sua respectiva solução.

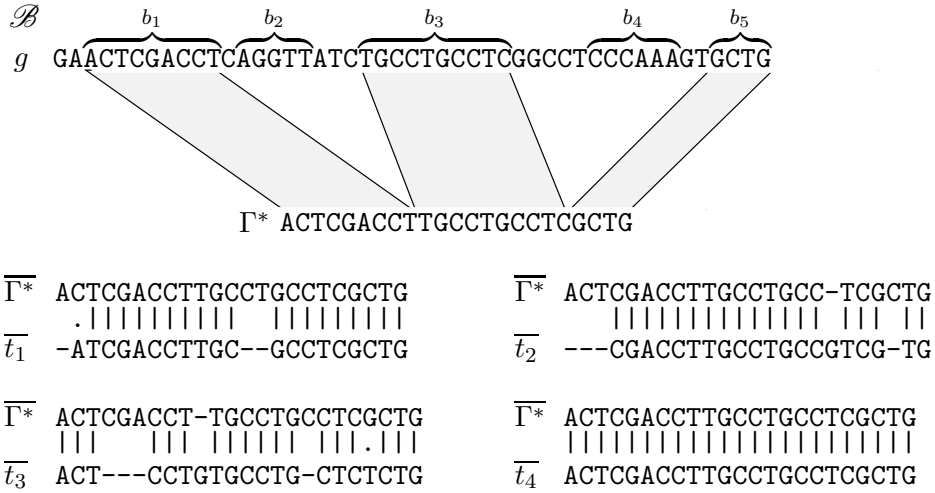


Figura 4.4: Exemplo de uma instância do problema do alinhamento *spliced* múltiplo e sua respectiva solução. Os segmentos do conjunto $\mathscr{B} = \{b_1, b_2, b_3, b_4, b_5\}$ correspondem aos trechos abaixo das chaves na sequência g. O conjunto $\mathscr{T}$ é composto pelas sequências t_1, t_2, t_3 e t_4 e a solução é composta dos blocos b_1, b_3 e b_5.

Assim como existe uma relação entre o problema do alinhamento *spliced* e o problema da identificação de genes por comparação de uma sequência de DNA com uma sequência de cDNA, também há uma relação entre o problema do alinhamento *spliced* múltiplo e o problema da identificação de genes por comparação de uma sequência de DNA com várias sequências de cDNA. Essa relação se dá substituindo a sequência g por uma sequência de DNA, cada uma das sequências t_i de $\mathscr{T}$ por uma sequência de cDNA e utilizando possíveis éxons da sequência de DNA como os elementos de $\mathscr{B}$. Mais uma vez, graças ao princípio de conservação das bases, uma saída gerada por uma solução para o problema do alinhamento *spliced* múltiplo com estas entradas representará uma solução plausível para o problema da identificação de genes através da comparação de uma sequência de DNA com várias sequências de cDNA.

4.2.2 Determinação da complexidade do problema do alinhamento *spliced* múltiplo

Antes de pensarmos em soluções para o problema do alinhamento *spliced* múltiplo que possam ser utilizadas na tarefa de identificação de genes pela comparação de um DNA com vários cDNAs, é necessário classificá-lo quanto à sua complexidade. Para esse fim formulamos uma versão de decisão do problema, que é:

> **Problema do alinhamento *spliced* múltiplo – versão de decisão (PASMD)**: dados uma sequência g, um conjunto de sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_u\}$, um conjunto $\mathscr{B}$ de segmentos de g, uma função de pontuação ω e um $k \in \mathbb{Z}$, determinar se existe um conjunto ordenado de segmentos Γ de $\mathscr{B}$ tal que $\sum_{i=1}^{u} \mathrm{sim}_w(t_i, \Gamma^*) \geq k$.

Observe que a versão de otimização do problema do alinhamento *spliced* múltiplo é no mínimo tão difícil quanto a versão de decisão, pois se existir um algoritmo que resolve a versão de otimização, podemos utilizá-lo para resolver a versão de decisão. Para isso, basta executar o algoritmo de otimização em uma instância da versão de decisão e então comparar o valor da solução devolvida pelo algoritmo com o valor de k.

Demonstraremos agora que o PASMD é NP-completo, para uma função de pontuação específica. Faremos isso provando que o PASMD pertence à classe NP e reduzindo um problema da classe NP a ele.

Para mostrar que o PASMD pertence à classe NP, precisamos mostrar que uma solução para ele pode ser verificada em tempo polinomial. Para verificar um conjunto de blocos Γ como solução do PASMD, basta calcular o valor de $\sum_{i=1}^{u} \mathrm{sim}_w(t_i, \Gamma^*)$ com o algoritmo de Needleman-Wunsch e compará-lo com k. Sejam n o comprimento da sequência g e m o comprimento da maior sequência em $\mathscr{T}$. O tempo gasto por essa verificação é $O(umn)$, portanto o PASMD $\in NP$.

O restante da demonstração de que o PASMD é NP-completo envolve um problema conhecido como problema da sequência mediana. Para a definição desse problema, considere uma métrica δ, um alfabeto Σ e um conjunto W de sequências construídas sobre Σ. Uma **sequência mediana** de W com respeito a δ é uma sequência μ construída sobre Σ tal que o valor de $\sum_{w \in W} d_\delta(\mu, w)$ é mínimo, onde d_δ é uma δ-distância de edição.

Dada a definição de sequência mediana, o problema da sequência mediana corresponde a:

> **Problema da sequência mediana**: dados um alfabeto Σ, um conjunto W de sequências construídas sobre Σ e uma δ-distância de edição d_δ, encontrar uma sequência μ construída sobre Σ tal que μ é uma sequência mediana de W, com respeito a d_δ.

De acordo com Nicolas e Rivals em [32], encontrar uma sequência mediana de um conjunto W equivale a resolver o problema do alinhamento múltiplo. Nesse mesmo trabalho, Nicolas e Rivals demonstram que o problema da sequência mediana é NP-completo mesmo para alfabetos binários ($|\Sigma| = 2$), sob a distância de Levenshtein (utilizando d_E ao invés de d_δ). Essa versão do problema encontra-se definida abaixo, e a demonstração da sua NP-completude é feita através de uma redução do problema LCS (problema da maior subsequência comum) a ele.

> **Problema da sequência mediana binário (PSMB)**: dados um alfabeto Σ de tamanho 2, um conjunto W de sequências construídas sobre Σ e um $i \in \mathbb{N}$, determinar se existe uma sequência μ construída sobre Σ tal que $\sum_{w \in W} d_E(\mu, w) \leq i$, onde d_E é a distância de Levenshtein.

A segunda parte da demonstração da NP-completude do PASMD se dá pela redução do PSMB a ele. Ou seja, desenvolvendo um algoritmo eficiente que transforma uma entrada do PSMB em uma entrada válida do PASMD de tal forma que uma solução obtida para o PASMD com esta entrada seja também uma solução para o PSMB. Essa redução é explicada nos próximos parágrafos.

Considere que temos uma instância $I = (W, i)$ do PSMB e precisamos construir uma entrada $I' = (g, \mathscr{T}, \mathscr{B}, \omega, k)$ do PASMD a partir de I. Para isso, primeiro atribuímos o conjunto W ao conjunto $\mathscr{T}$ e o valor de i multiplicado por -1, a k. Com isso, se existe um conjunto ordenado de segmentos $\Gamma \subseteq \mathscr{B}$ tal que $\sum_{t \in \mathscr{T}} \text{sim}_\omega(t, \Gamma^*) \geq k$ então Γ também é tal que

$$\sum_{w \in W} \text{sim}_\omega(w, \Gamma^*) \geq -i. \tag{4.4}$$

Definiremos a função de pontuação ω do PASMD como $\omega(\alpha, \beta) = 0$ se $\alpha = \beta$ e $\omega(\alpha, \beta) = -1$ caso contrário. Aqui é importante relembrar que, de acordo com Setubal e Meidanis [41], o valor da similaridade entre duas sequências com essa função de pontuação é igual ao valor da distância de Levenshtein entre essas duas sequências multiplicado por -1. Aplicando essa propriedade na Inequação 4.4 temos que Γ é tal que

$$-\sum_{w \in W} d_E(w, \Gamma^*) \geq -i, \tag{4.5}$$

e portanto

$$\sum_{w \in W} d_E(w, \Gamma^*) \leq i. \tag{4.6}$$

Ou seja, a existência de um conjunto $\Gamma \subseteq \mathscr{B}$ tal que $\sum_{t \in \mathscr{T}} \text{sim}_\omega(t, \Gamma^*) \geq k$ implica que $\sum_{w \in W} d_E(w, \Gamma^*) \leq i$.

A sequência g e o conjunto $\mathscr{B}$ que fazem parte da entrada do PASMD necessitam de algumas observações para um melhor entendimento das suas construções. Uma dessas observações é o fato de que uma sequência mediana de um conjunto $W = \{w_1, w_2, \ldots, w_k\}$ é sempre menor que $2n$, onde n é o comprimento da maior sequência em W. Para mostrarmos que isso é verdade, suponha que s é uma sequência mediana de W, sob a distância de Levenshtein e tem comprimento $2n$, onde n é o comprimento da maior sequência em W. O valor de $\sum_{i=1}^{k} d_E(s, w_i)$ é no mínimo kn pois a distância de Levenshtein entre duas sequências é maior ou igual à diferença dos seus comprimentos. Tome agora uma sequência s' do conjunto W tal que $|s'| = n$, ou seja, s' é uma das maiores sequências do conjunto. Observe que o valor de $\sum_{i=1}^{k} d_E(s', w_i)$ não pode ser maior que $(k-1)n$, já que o valor da distância de Levenshtein entre duas sequências é menor ou igual ao comprimento da maior sequência. Como $\sum_{i=1}^{k} d_E(s', w_i) < \sum_{i=1}^{k} d_E(s, w_i)$ temos que s não é uma sequência mediana, o que é uma contradição. Daí, o comprimento de uma sequência mediana de W deve ser menor que $2n$.

Dado o fato de que o tamanho de uma sequência mediana é sempre menor que $2n$, onde n é o comprimento da maior sequência do conjunto W, construiremos a sequência g e o conjunto $\mathscr{B}$ de tal forma que eles possibilitem a obtenção, através de concatenações de subconjuntos ordenados de $\mathscr{B}$, de todas as sequências possíveis sobre Σ de comprimento até $2n$. A sequência g é construída da seguinte maneira: concatene os caracteres de Σ em uma sequência s_Σ, e concatene $2n$ cópias de s_Σ. O resultado disso é uma sequência de comprimento $4n$ de 0's e 1's concatenados alternadamente. Criando um segmento em $\mathscr{B}$ para cada caractere em g, possibilitamos a escolha de qualquer caractere de Σ $2n$ vezes em um conjunto ordenado de segmentos $\Gamma \subseteq \mathscr{B}$ do PASMD.

Observe que se existe um conjunto ordenado de segmentos $\Gamma \subseteq \mathscr{B}$ tal que $\sum_{t \in \mathscr{T}} \mathrm{sim}_\omega(t, \Gamma^*) \geq k$ para a instância I' que acabamos de descrever para o PASMD, então existe uma sequência $\mu = \Gamma^*$ construída sobre Σ tal que $\sum_{w \in W} d_E(\mu, w) \leq i$. Por outro lado, se não há um conjunto ordenado de segmentos $\Gamma \subseteq \mathscr{B}$ tal que $\sum_{t \in \mathscr{T}} \mathrm{sim}_\omega(t, \Gamma^*) \geq k$, então todos os conjuntos ordenados de segmentos $\Gamma \subseteq \mathscr{B}$ são tais que $\sum_{t \in \mathscr{T}} \mathrm{sim}_\omega(t, \Gamma^*) < k$ e consequentemente $\sum_{w \in W} d_E(w, \Gamma^*) > i$. Como é possível obter qualquer sequência μ sobre Σ onde $|\mu| \leq 2n$ através da concatenação de um conjunto ordenado de segmentos $\Gamma \subseteq \mathscr{B}$, então não há uma sequência μ tal que $\sum_{w \in W} d_E(w, \Gamma^*) \leq i$. Por isso, se conseguirmos decidir o PASMD com a instância I', estamos decidindo também o PSMB para a instância I. Uma ilustração da redução do PSBM ao PASMD pode ser vista na Figura 4.5

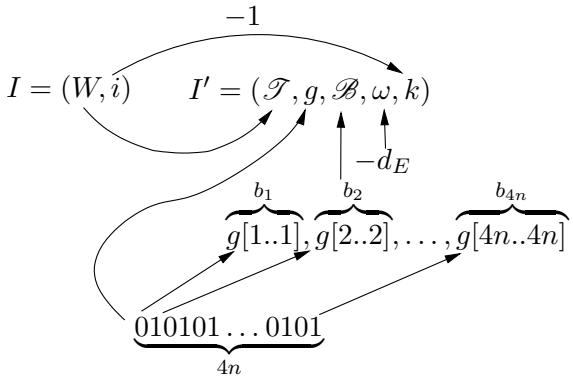


Figura 4.5: Redução do PSMB ao PASMD.

É importante ainda observar que, apesar da função ω ter sido fixada na redução do PSMB ao PASMD, podemos afirmar que o PASMD é NP-completo no caso geral. Isso porque, se resolvermos o PASMD para um ω genérico, estamos resolvendo também o problema para o caso em que ω é equivalente à distância de Levenshtein.

O algoritmo que transforma uma entrada do PSMB em uma entrada do PASMD é apresentado em pseudocódigo no Algoritmo 4.2.

ALGORITMO 4.2 Redução de uma entrada do PSMB a uma entrada do PASMD.

PSMB_PARA_PASMD(W, i): recebe um conjunto W de sequências construídas sobre um alfabeto binário $\Sigma = \{0, 1\}$, um número natural i e devolve uma quíntupla $(g, \mathscr{T}, \mathscr{B}, \omega, k)$ onde g é uma sequência, $\mathscr{T}$ é um conjunto de sequências, $\mathscr{B}$ é um conjunto ordenado de segmentos, ω é uma função de pontuação e k é um número inteiro.

1: $\omega \leftarrow -d_E$;
2: $k \leftarrow -i$;
3: $\mathscr{T} \leftarrow W$;
4: $n \leftarrow \max_{t \in \mathscr{T}} |t|$;
5: Atribua a g uma sequência de comprimento $4n$ formada por $0's$ e $1's$ alternadamente;
6: Crie um conjunto ordenado de segmentos $\mathscr{B}$ e inclua um segmento em $\mathscr{B}$ composto por cada caractere em g;
7: **devolva** $(g, \mathscr{T}, \mathscr{B}, \omega, k)$;

Sobre a complexidade do Algoritmo 4.2 observe que a definição da função de pontuação e do valor de k gastam tempo $O(1)$. A definição do conjunto $\mathscr{T}$ gasta tempo $O(m)$, onde m equivale à soma dos comprimentos de todas as sequências de W. A escolha da maior sequência de $\mathscr{T}$ também gasta tempo $O(m)$, pois basta percorrer todo o conjunto W. A construção da sequência g e a inclusão de um segmento para cada caractere de g em $\mathscr{B}$ gastam tempo $O(n)$, onde n é o comprimento da maior sequência de W. Com isso, nossa redução tem complexidade de tempo $O(m)$.

Capítulo 5

Heurísticas Para o Problema do Alinhamento *Spliced* Múltiplo

Como demonstramos no capítulo anterior, o problema do alinhamento *spliced* múltiplo é NP-completo sob a distância de Levenshtein. Com isso, nossa busca por soluções para o problema se restringe a aproximações e heurísticas. Descreveremos neste capítulo quatro heurísticas desenvolvidas por nós para o problema do alinhamento *spliced* múltiplo, bem como suas respectivas análises de complexidade de tempo e de espaço. Ainda neste capítulo, discutimos a implementação das quatro heurísticas e, ao final, detalhamos os resultados de uma avaliação experimental que realizamos com elas.

Na descrição de cada uma das heurísticas, suponha que a entrada para o PASM seja dada como a quádrupla $(g, \mathscr{T}, \mathscr{B}, \omega)$ onde g é uma sequência, $\mathscr{T}$ é um conjunto de sequências, $\mathscr{B}$ é um conjunto ordenado de segmentos de g e ω é uma função de pontuação. Considere que as sequências de $\mathscr{T}$ e também a sequência g são construídas sobre um alfabeto Σ qualquer.

5.1 Heurística da sequência central

Nessa heurística, uma sequência central de $\mathscr{T}$ é encontrada através de sucessivos alinhamentos globais entre cada sequência e todas as outras do conjunto. Após a escolha da sequência central de $\mathscr{T}$, o algoritmo de alinhamento *spliced* de Gelfand *et al.* [19] é utilizado para alinhar os blocos da sequência alvo com a sequência central de $\mathscr{T}$. Os segmentos de $\mathscr{B}$ escolhidos neste último passo constituem a solução devolvida pela heurística para o PASM. Essa heurística é apresentada através de uma ilustração na Figura 5.1 e em pseudocódigo no Algoritmo 5.1.

O laço `para` das linhas 1 e 2 do Algoritmo 5.1 realiza k^2 cálculos de similaridade entre duas sequências. Tomando-se m como o comprimento da sequência mais longa em $\mathscr{T}$, esse laço realiza uma quantidade de operações da ordem de k^2m^2. A linha 3 realiza uma quantidade de operações linear sobre k. A operação da linha 4 tem complexidade de tempo $O(mnc + mb^2)$, onde c é a cobertura da sequência g por segmentos[1], $n = |g|$

[1] O cálculo do valor da cobertura de uma sequência por segmentos é descrito na Seção 4.1.1

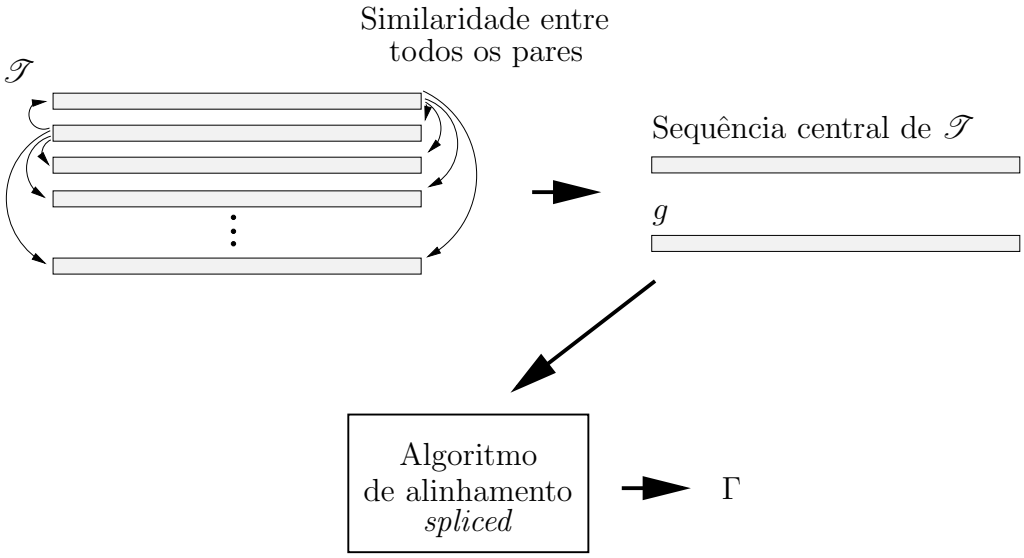


Figura 5.1: Heurística da sequência central.

ALGORITMO 5.1 Heurística da sequência central.

HEURÍSTICA_SEQUÊNCIA_CENTRAL$(g, \mathscr{T}, \mathscr{B}, \omega)$: recebe uma sequência g, um conjunto de k sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$, um conjunto $\mathscr{B}$ de segmentos de g, uma função de pontuação ω e devolve um subconjunto ordenado Γ de $\mathscr{B}$.

1: **para** $i \leftarrow 1$ até k **faça**
2: $\quad v[i] \leftarrow \sum_{j=1, j\neq i}^{k} \mathrm{sim}(t_i, t_j)$;
3: $c \leftarrow i$, tal que $v[i]$ é máximo em v;
4: $\Gamma \leftarrow$ALINHAMENTO_SPLICED$(g, t_c, \mathscr{B}, \omega)$;
5: **devolva** Γ;

e $b = |\mathscr{B}|$. Reunindo as complexidades e simplificando-as, temos que a complexidade de tempo da heurística da sequência central passa a ser $O(k^2m^2 + mnc + mb^2)$. As linhas de 1 a 3 necessitam de espaço linear sobre m (cálculo da similaridade) ou k ($|v|$). Por isso a complexidade de espaço do Algoritmo 5.1 é dominada pela complexidade de espaço do algoritmo ALINHAMENTO_SPLICED (apresentado no Algoritmo 4.1), que é $O(mnc)$.

5.2 Heurística da sequência consenso

Na heurística da sequência consenso, o algoritmo Estrela de Gusfield [21], descrito na Seção 2.2.6 no Algoritmo 2.3, é utilizado para obter um alinhamento múltiplo das sequências em $\mathscr{T}$. A sequência consenso extraída desse alinhamento é então utilizada como entrada do algoritmo ALINHAMENTO_SPLICED. Assim como na heurística da sequência central, os segmentos de $\mathscr{B}$ escolhidos constituem a solução devolvida pela heurística para o PASM. Essa heurística é apresentada através de uma ilustração na Figura 5.2 e em pseudocódigo no Algoritmo 5.2.

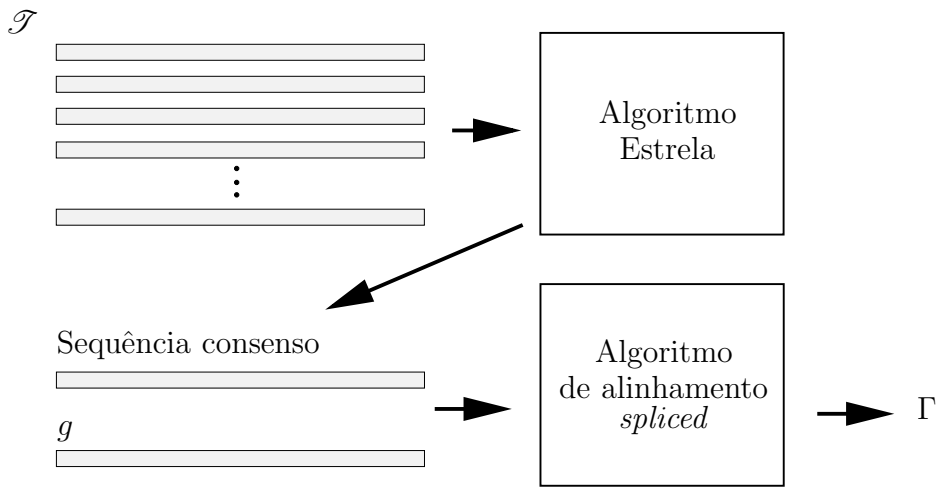


Figura 5.2: Heurística da sequência consenso.

ALGORITMO 5.2 Heurística da sequência consenso.

HEURÍSTICA_SEQUÊNCIA_CONSENSO($g, \mathscr{T}, \mathscr{B}, \omega$): recebe uma sequência g, um conjunto de k sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$, um conjunto $\mathscr{B}$ de segmentos de g, uma função de pontuação ω e devolve um subconjunto ordenado Γ de segmentos de $\mathscr{B}$.

1: $A \leftarrow$ ESTRELA($t_1, \ldots, t_k$);
2: Construa a sequência consenso de A e atribua a s_c;
3: $\Gamma \leftarrow$ALINHAMENTO_SPLICED($g, s_c, \mathscr{B}, \omega$);
4: **devolva** Γ;

Seja m o comprimento da maior sequência de $\mathscr{T}$. A complexidade de tempo do algoritmo ESTRELA, executado na linha 1 do Algoritmo 5.2, é $O(k^2m^2)$ como descrito no Capítulo 2. A construção da sequência consenso na linha 2 tem complexidade de tempo e de espaço $O(k^2m)$, pois consiste, basicamente, em percorrer a matriz A uma vez. O algoritmo ALINHAMENTO_SPLICED, executado na linha 3, tem complexidade de tempo $O(|s_c|nc + |s_c|b^2)$, onde $n = |g|$, $b = |\mathscr{B}|$ e c é a cobertura de g por segmentos. Observe que s_c pode ter comprimento máximo km e com isso o tempo gasto na linha 3 pode ser reescrito como $O(kmnc + kmb^2)$. A complexidade de tempo do Algoritmo 5.2 é dada então por $O(k^2m^2 + kmnc + kmb^2)$. Sua complexidade de espaço é dada por $O(k^2m + m^2)$ referente ao espaço gasto pelo algoritmo ESTRELA, acrescido de $O(kmnc)$ referente ao espaço gasto pelo ALINHAMENTO_SPLICED, ou seja é $O(kmnc + k^2m + m^2)$.

5.3 Heurística do consenso de blocos

A heurística do consenso de blocos originou-se da idéia da sequência consenso. Nessa heurística, o algoritmo ALINHAMENTO_SPLICED é executado várias vezes tomando como entrada a sequência g, com seu respectivo conjunto $\mathscr{B}$, e cada uma das sequências de $\mathscr{T}$. Após isso, realiza-se uma contagem para verificar quantas vezes cada bloco participou de cada uma das soluções do algoritmo ALINHAMENTO_SPLICED. Uma solução para o PASM é então construída reunindo no conjunto Γ todos os blocos que

participam de um número maior que metade das soluções. Observe que escolhendo somente blocos que participam de mais que metade das soluções não é possível escolher um par de blocos sobrepostos. Essa heurística é apresentada através de uma ilustração na Figura 5.3 e em pseudocódigo no Algoritmo 5.3.

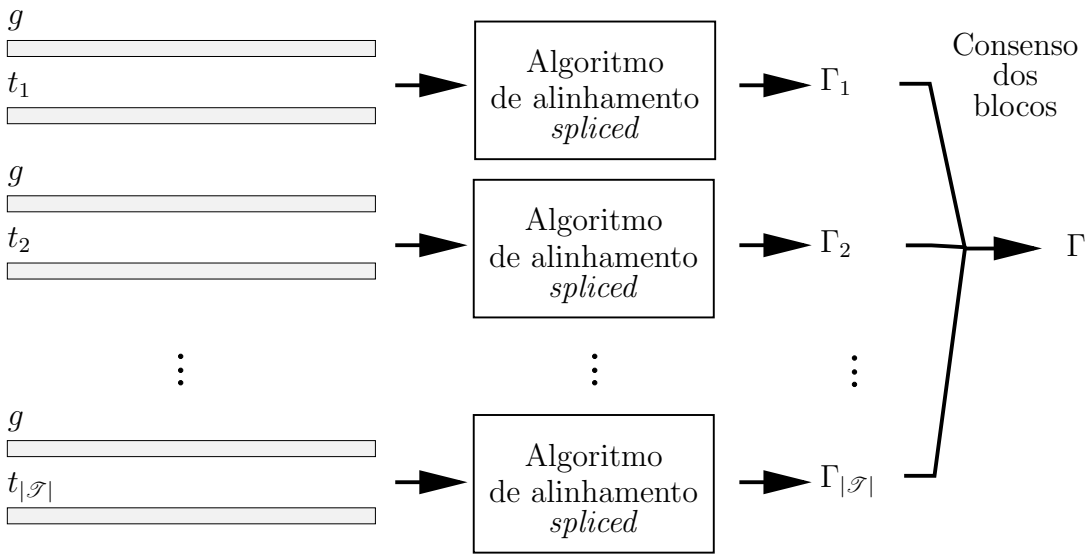


Figura 5.3: Heurística do consenso de blocos.

ALGORITMO 5.3 Heurística do consenso de blocos.

HEURÍSTICA_CONSENSO_BLOCOS$(g, \mathscr{T}, \mathscr{B}, \omega)$: recebe uma sequência g, um conjunto de k sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$, um conjunto de b segmentos $\mathscr{B} = \{b_1, b_2, \ldots, b_b\}$ de g, uma função de pontuação ω e devolve um subconjunto ordenado Γ de segmentos de $\mathscr{B}$.

```
1:  para i ← 1 até k faça
2:      γ_i ← ALINHAMENTO_SPLICED(g, t_i, ℬ, ω);
3:  para i ← 1 até b faça
4:      o[i] ← 0;
5:      para j ← 1 até k faça
6:          se b_i ∈ γ_j então
7:              o[i] ← o[i] + 1;
8:  Γ ← ∅;
9:  para i ← 1 até b faça
10:     se o[i] ≥ (k/2) + 1 então
11:         Γ ← Γ ∪ {b_i};
12: devolva Γ;
```

Seja m o comprimento da maior sequência de $\mathscr{T}$. A complexidade de tempo do laço `para` que se inicia na linha 1 é limitada por k vezes a complexidade do algoritmo ALINHAMENTO_SPLICED, que é $O(mnc + mb^2)$, onde $n = |g|$, $b = |\mathscr{B}|$ e c é a cobertura por segmentos da sequência g. A complexidade de tempo dos laços `para` aninhados e situados entre as linhas 3 e 7 é da ordem de kb^2, pois suas operações são executadas kb vezes e verificar se um bloco está em uma solução pode utilizar até b operações. A linha 8 consome tempo $O(1)$. O laço `para` que se inicia na linha 9 é executado k vezes e suas operações internas consomem tempo $O(1)$. Reunindo as complexidades de tempo de cada trecho do Algoritmo 5.3, concluímos que a sua complexidade de

tempo é $O(kmnc + kmb^2)$. O Algoritmo 5.3 utiliza espaço $O(mnc)$ para realizar os alinhamentos *spliced* das linhas 1 e 2, espaço da ordem de kb para armazenar as soluções dos alinhamentos *spliced* e da ordem de b para o vetor que contabiliza os segmentos nas soluções. Sua complexidade de espaço é, portanto, $O(mnc + kb)$.

5.4 Heurística do algoritmo genético

A última heurística corresponde a um algoritmo genético. Nele, a população é formada por subconjuntos ordenados de segmentos de $\mathscr{B}$, ou seja, possíveis soluções para o PASM. Os segmentos que compõem cada indivíduo da população inicial são escolhidos aleatóriamente.

Depois da obtenção de uma população inicial, cada indivíduo é classificado pelo somatório das similaridades entre a sequência resultante da concatenação de seus segmentos e cada uma das sequências de $\mathscr{T}$. A similaridade é calculada com um algoritmo de alinhamento global. As melhores sequências são mantidas intactas, outras são modificadas e as de menor pontuação descartadas. São então gerados novos indivíduos para substituir os que foram descartados. Os novos indivíduos são construídos a partir dos que não foram descartados. O processo de avaliação é então repetido um certo número de vezes com o objetivo de melhorar o somatório da similaridade entre a sequência resultante da concatenação dos segmentos do melhor indivíduo da população e cada uma das sequências de $\mathscr{T}$.

As mutações feitas em cada indivíduo podem ser a inserção ou remoção de um ou mais segmentos dele. Além disso, novos indivíduos são criados através do cruzamento dos melhores indivíduos. O cruzamento tenta unir dois subconjuntos ordenados escolhendo aleatóriamente um segmento de um dos conjuntos ordenados como ponto de cruzamento. Os segmentos que antecedem o ponto de cruzamento e o próprio são inclusos no novo indivíduo, acrescidos dos segmentos do outro conjunto ordenado que sucedem o ponto de cruzamento.

Ao final, a concatenação de melhor pontuação é devolvida como resposta. Essa heurística é apresentada através de uma ilustração na Figura 5.4 e em pseudocódigo no Algoritmo 5.4.

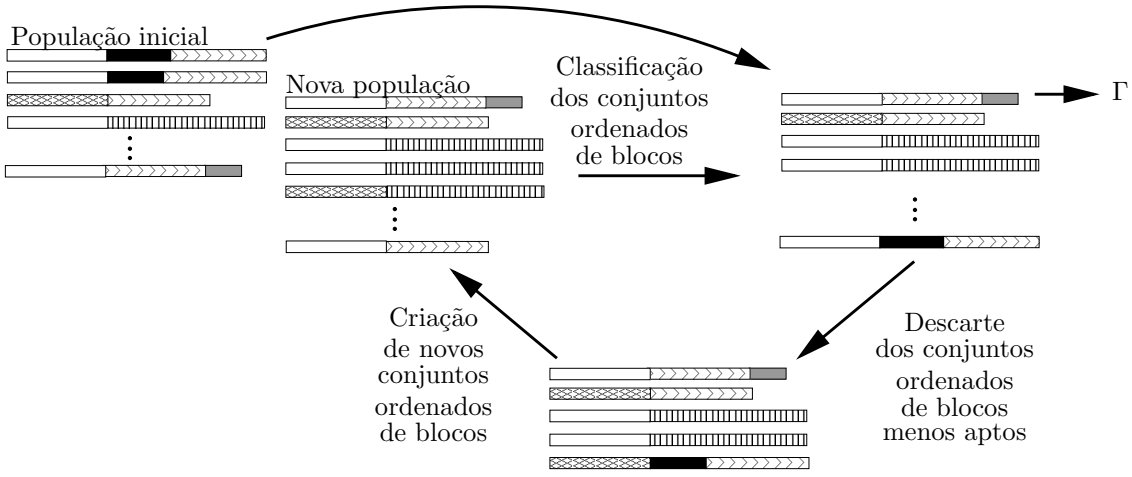


Figura 5.4: Heurística do algoritmo genético.

ALGORITMO 5.4 Algoritmo genético.

ALGORITMO_GENÉTICO($g, \mathscr{T}, \mathscr{B}, \omega$): recebe uma sequência g de comprimento n, um conjunto de k sequências $\mathscr{T} = \{t_1, t_2, \ldots, t_k\}$, um conjunto de l segmentos $\mathscr{B} = \{b_1, b_2, \ldots, b_l\}$ de g, uma função de pontuação ω, quatro números inteiros I, E, M e G e devolve um subconjunto ordenado Γ de segmentos de $\mathscr{B}$. Os números I, E, M e G correspondem ao tamanho da população, à quantidade de indivíduos mantidos inalterados a cada geração, à quantidade de indivíduos que vão sofrer mutações e à quantidade de gerações, respectivamente.

1: Construa um vetor P com I elementos, onde cada elemento de P é um subconjunto ordenado de segmentos de $\mathscr{B}$, escolhidos de maneira aleatória;
2: **para** $f \leftarrow 1$ até G **faça**
3: **para** $p \leftarrow 1$ até I **faça**
4: apt$[p] \leftarrow \sum_{i=1}^{k} \text{sim}(P[p]^*, t_i)$;
5: Ordene os elementos de P pelos valores contidos em apt, de maneira decrescente;
6: Realize mutações aleatórias nos indivíduos $P[E+1], \ldots, P[E+M]$;
7: Descarte os elementos $P[E+M+1], \ldots, P[I]$;
8: $i \leftarrow 1$;
9: **para** $j \leftarrow (E+M+1)$ até I **faça**
10: Realize o cruzamento dos elementos $P[i]$ e $P[i+1]$ e insira o resultado desse cruzamento em $P[j]$;
11: $i \leftarrow (i+1) \mod (E-1)$;
12: **para** $p \leftarrow 1$ até I **faça**
13: apt$[p] \leftarrow \sum_{i=1}^{k} \text{sim}(P[p]^*, t_i)$;
14: Ordene os elementos de P pelos valores contidos em apt, de maneira decrescente;
15: $\Gamma \leftarrow P[1]$;
16: **devolva** Γ;

A construção da população na linha 1 do Algoritmo 5.4 gasta tempo da ordem de Il, pois é necessário gerar I indivíduos de até l blocos. As operações entre as linhas 2 e 11 correspondem a uma geração e são executadas G vezes. Seja m o comprimento da maior sequência em $\mathscr{T}$, então o laço `para` que se inicia na linha 3 leva tempo da ordem de $Ikmn$. A ordenação da linha 5 gasta tempo da ordem de I^2. As mutações na linha 6 gastam tempo da ordem de Il, no pior caso. O descarte de indivíduos na linha 7 gasta tempo da ordem de Il no pior caso. A linha 8 gasta tempo 1. As linhas de 9 a 11 gastam tempo da ordem de Il, no pior caso. Podemos resumir a complexidade de tempo das linhas de 2 a 11 em $O(GIkmn + GI^2 + GIl)$ pois todas as operações entre as linhas 3 e 11 são executadas uma vez para cada geração (laço `para` da linha 2). As linhas de 12 a 14 são semelhantes às linhas de 3 a 5 e têm a mesma complexidade de tempo. As linhas de 15 e 16 tem complexidade de tempo $O(1)$. A complexidade de tempo do Algoritmo 5.4 é dada então por $O(GIkmn + GI^2 + GIl)$.

O Algoritmo 5.4 utiliza, basicamente, um vetor de listas ligadas representando os blocos, que ocupa espaço da ordem de Il e um par de vetores para os laços das linhas 3 e 12, que ocupam espaço da ordem de n. Sua complexidade de espaço é, portanto, $O(Il + n)$.

5.5 Implementação das heurísticas

As heurísticas apresentadas foram implementadas em quatro programas distintos utilizando a linguagem `ANSI C`. Os programas recebem como entrada um arquivo contendo a sequência base, um arquivo contendo índices de segmentos da sequência base correspondentes aos blocos e um arquivo contendo os nomes dos arquivos com as sequências modelo como entrada. Os arquivos de entrada devem estar em um formato específico, chamado `FASTA`[2].

Em todas as heurísticas, foi utilizada a distância de Levenshtein no lugar da função de pontuação ω. Para utilizar a distância de Levenshtein sem modificar os algoritmos de cálculo da similaridade e do alinhamento *spliced*, fizemos $\omega(\alpha, \beta) = -1$ quando $\alpha \neq \beta$ e $\omega(\alpha, \beta) = 0$ quando $\alpha = \beta$ caso contrário. Um valor de similaridade calculado dessa maneira corresponde ao valor da distância de Levenshtein multiplicado por -1.

Como pode ser observado pela descrição das heurísticas, o núcleo de todas elas corresponde ao algoritmo de alinhamento *spliced*, baseado na Recorrência 4.3 descrita por Gelfand *et al.*. Por não termos encontrado a implementação supostamente disponível em *http://www_hto.usc.edu/software/procrustes/* (última tentativa de acesso em 13 de dezembro de 2010) desse algoritmo, implementamos a Recorrência 4.3 em um programa denominado `PROCRUSTES`. Detalhes dessa implementação podem ser vistos a seguir.

5.5.1 Detalhes da implementação do algoritmo de alinhamento *spliced*

Uma das principais dificuldades da implementação de programas que tratam sequências de DNA é o limite de memória dos computadores. Muitas vezes é necessário processar uma sequência com mais de 500kb e nesse caso mesmo um algoritmo com complexidade de espaço polinomial pode não ser útil na prática. Nas análises de complexidade de espaço de nossas heurísticas baseadas no algoritmo de alinhamento *spliced*, pode-se observar que ele consome a maior parte do espaço utilizado pelas heurísticas. Por isso, necessitamos implementá-lo de forma eficiente.

Voltando à solução do problema do alinhamento *spliced*, proposta por Gelfand *et al.*, observe que $S[i][j][k]$ só é definido para os índices i que estão dentro de um bloco na sequência base. Com isso, torna-se desnecessário alocar uma matriz de dimensões $b \times m \times n$, sendo suficiente alocar apenas b matrizes de dimensões $m \times |b_k|$, onde b é o tamanho do conjunto de blocos $\mathscr{B}$, m é o comprimento da sequência modelo, n é o comprimento da sequência base e $b_k \in \mathscr{B}$. Implementamos essa versão, invertendo a ordem dos índices i, j, k para k, j, i pois dessa forma podemos alocar um vetor de comprimento b (indexado por k) que contém ponteiros para b matrizes de dimensões $m \times |b_k|$ (indexadas por j e i, respectivamente).

Um outro item importante é a inicialização da matriz S, que não está descrita em [19]

[2] O formato `FASTA`, bastante difundido na Bioinformática, é um padrão definido para representar uma sequência de caracteres qualquer em um arquivo texto. Nesse formato, a primeira linha (que é iniciada com um `>`) contém informações sobre a sequência. As linhas seguintes, que devem ter comprimento máximo de 120 caracteres (em geral, utiliza-se 70 ou 80), contêm a sequência propriamente dita.

onde Gelfand *et al.* propuseram o algoritmo de alinhamento *spliced*. As células da forma $S[0][0][k]$, para $k \leq 1 \leq b$, devem ser inicializadas com 0, representando um alinhamento de uma sequência modelo vazia com a concatenação de um conjunto de blocos contendo somente o bloco k até a sua posição 0. Além das células $S[0][0][k]$, as células da forma $S[i][0][k]$ com $\mathit{first}(k) \leq i \leq \mathit{last}(k)$ e $1 \leq k \leq b$ devem ser inicializadas com o valor de $S[i-1][0][k] + \omega(\alpha, '-')$ representando o alinhamento de uma concatenação de blocos contendo apenas o bloco k até a posição i com i espaços antes da sequência modelo.

É necessário ainda criar um bloco $k = 0$ (chamado de bloco zero) onde $\mathit{first}(k) = 0$ e $\mathit{last}(k) = 0$, e inicializar as células da forma $S[0][j][0]$ onde $1 \leq j \leq m$, com o valor de $S[0][j-1][0]+\omega(\alpha, '-')$ representando o alinhamento da sequência modelo até a posição j com j espaços antes do bloco 0. O bloco zero permite ao algoritmo de alinhamento *spliced* começar um novo alinhamento inserindo espaços antes do primeiro bloco. O bloco zero não deve ser incluído no conjunto de blocos devolvido como solução. Em nossa implementação, simulamos o bloco zero preenchendo as células $S[\mathit{first}(k)-1][j][k]$ para $1 \leq j \leq m$, com o valor de $S[\mathit{first}(k) - 1][j-1][k] + \omega(\alpha, '-')$ e modificando a Recorrência 4.3 para que ela possa escolher esses valores (além dos que estão definidos nela) quando $i = \mathit{first}(k)$. A versão modificada da recorrência do algoritmo de alinhamento *spliced* é apresentada na Recorrência 5.1.

$$S[i][j][k] = \max \begin{cases} S[i-1][j-1][k] + \omega(g[i], t[j]) & \\ S[i-1][j][k] + \omega(g[i], '-') & \\ \max_{l \in \mathscr{B}(\mathit{first}(k))} S[\mathit{last}(l)][j-1][l] + \omega(g[i], t[j]) & \text{se } i = \mathit{first}(k) \\ \max_{l \in \mathscr{B}(\mathit{first}(k))} S[\mathit{last}(l)][j][l] + \omega(g[i], '-') & \text{se } i = \mathit{first}(k) \\ S[i][j-1][k] + \omega('-', t[j]). & \end{cases} \tag{5.1}$$

5.5.2 Algumas observações sobre a implementação das heurísticas

O algoritmo Estrela, utilizado na heurística da sequência consenso, realiza inserções de colunas em alinhamentos. A forma mais natural de representar estes alinhamentos seria através de matrizes. Entretanto, se os alinhamentos fossem representados por matrizes, seria trabalhoso realizar inserções de colunas, já que matrizes possuem dimensões fixas. Por isso os alinhamentos do algoritmo Estrela foram implementados na forma de listas ligadas, que facilitam esse tipo de operação.

Na heurística do consenso de blocos, quando nenhum bloco está presente em mais da metade das soluções devolvidas pelas execuções do algoritmo Alinhamento_Spliced, utilizamos a solução de maior pontuação das execuções do algoritmo Alinhamento_Spliced como sua solução, ao invés de devolver a solução vazia do consenso de blocos.

No algoritmo genético, os parâmetros correspondentes às quantidades de indivíduos que compõem a população, indivíduos mantidos inalterados por geração, indivíduos que sofrem mutação por geração e de gerações foram fixados em 20, 10, 4 e 30, respectivamente.

As mutações no algoritmo genético são realizadas aleatóriamente em todos os aspectos, ou seja, tanto na escolha de uma posição onde um bloco será inserido/deletado, quanto na escolha do novo bloco que será inserido. Na inserção de um novo bloco em um indivíduo, é necessário verificar se esse bloco "cabe" entre o bloco que o precede e o bloco que o segue. Para simplificar essa verificação, o algoritmo genético tenta inserir um bloco escolhido aleatóriamente entre dois outros dez vezes. Caso não seja possível inserir o novo bloco nas dez tentativas o algoritmo desconsidera aquela inserção e segue adiante.

5.6 Avaliação das heurísticas

Realizamos uma avaliação das nossas heurísticas, que consistiu em compará-las em termos da semelhança entre a concatenação de um conjunto ordenado de segmentos devolvidos como resposta por cada heurística e as sequências do conjunto $\mathscr{T}$ dado como entrada. Mais detalhadamente, dada uma instância $I = (g, \mathscr{T}, \mathscr{B}, \delta)$ do problema do alinhamento *spliced* múltiplo, onde δ é uma métrica sobre um alfabeto $\overline{\Sigma}$ tal que para todo $\alpha, \beta \in \overline{\Sigma}$, $\delta(\alpha, \beta) = 0$ se $\alpha = \beta$ e $\delta(\alpha, \beta) = 1$ caso contrário, calculamos o valor $\text{Score}(I) = \sum_{i=1}^{k} d_E(t_i, \Gamma^*)$ para cada um dos conjuntos ordenados de segmentos Γ devolvidos pelas heurísticas. Quanto menor esse valor, melhor a solução e consequentemente a heurística para aquela instância.

As heurísticas foram comparadas com base em um conjunto de testes com sequências criadas artificialmente. Esse conjunto contém 720 instâncias e cada uma delas é composta de uma sequência g construída sobre o alfabeto Σ, um conjunto $\mathscr{T} = \{t_1, t_2, t_3, t_4, t_5\}$ de 5 sequências de mesmo comprimento construídas sobre Σ e um conjunto $\mathscr{B}$ de blocos de g. O alfabeto Σ sobre o qual as sequências foram construídas corresponde ao alfabeto da língua portuguesa, contendo os caracteres de A a Z.

As instâncias foram construídas variando-se os tamanhos de g, $t \in \mathscr{T}$ e $\mathscr{B}$. O comprimento de g varia entre 1000 e 60000 caracteres, com incrementos de 1000. Para cada diferente tamanho de g o comprimento de todos os $t \in \mathscr{T}$ correspondem a 2%, 5% ou 10% de $|g|$. Para cada combinação de $|g|$ e $|t| \in \mathscr{T}$ o tamanho de $\mathscr{B}$ varia em 10, 25, 50 e 100. Os blocos do conjunto $\mathscr{B}$ foram escolhidos aleatóriamente, com o tamanho limitado pelo comprimento das sequências do conjunto $\mathscr{T}$. Os caracteres que compõem as sequências g e $t \in \mathscr{T}$ foram escolhidos aleatóriamente dentro de Σ.

Os testes com as heurísticas foram executados em um computador com um processador Intel Core2Duo T6400, de 2 núcleos de 2GHz e 4GB de memória RAM sobre o sistema operacional Linux.

Os resultados das execuções das quatro heurísticas sobre as instâncias do conjunto com sequências artificiais podem ser vistos na Tabela 5.1. Para simplificar a apresentação dos resultados nesta tabela, chamaremos as heurísticas da sequência central, sequência consenso, consenso de blocos e algoritmo genético de `HEURÍSTICA1`, `HEURÍSTICA2`, `HEURÍSTICA3` e `HEURÍSTICA4`, respectivamente.

Heurística	Score	Tempo Médio	Pior Tempo
HEURISTICA1	**7011.42**	**3.03**	**24.35**
HEURISTICA2	7099.98	3.69	28.46
HEURISTICA3	7418.39	10.28	94.27
HEURISTICA4	7223.56	159.92	1163.33

Tabela 5.1: Resultados dos testes realizados com as 720 instâncias artificais para as quatro heurísticas propostas. A coluna **Score** contém, para cada heurística, a média aritmética dos valores de Score(I) calculados sobre as instâncias I do conjunto de testes. A coluna Tempo Médio contém as médias aritméticas dos tempos de execução de cada heurística sobre todas as instâncias. Esse tempo é dado em segundos. A coluna Pior Tempo contém o pior tempo de execução de cada heurística sobre o conjunto de testes. Esse tempo também é dado em segundos.

Observe que a heurística `HEURISTICA1` obteve o melhor resultado e a `HEURISTICA2` obteve um resultado próximo ao da `HEURISTICA1`. A `HEURISTICA4` obteve um resultado um pouco pior do que os das heurísticas `HEURISTICA1` e `HEURISTICA2` e a heurística `HEURISTICA3` foi a que se saiu pior dentre todas.

Analisando os resultados obtidos, pudemos notar que o comportamento ruim da `HEURISTICA3` se deve, principalmente, ao fato das sequências modelo das instâncias de teste serem muito diferentes entre si. Isso faz com que as soluções individuais dos alinhamentos *spliced* possuam poucos ou nenhum bloco em comum.

As complexidades de tempo das heurísticas baseadas no algoritmo de alinhamento *spliced* são bastante influenciadas pela complexidade deste último. Esse fato se reflete nos tempos de execução obtidos, com a `HEURISTICA3` apresentando um tempo pior de execução quando comparada às heurísticas `HEURISTICA1` e `HEURISTICA2` por executar mais vezes o algoritmo de alinhamento *spliced*. Sobre a `HEURISTICA4`, apesar dela não realizar um alinhamento *spliced*, ela apresentou tempos de execução ruins devido à enorme quantidade de operações realizadas para a classificação dos indivíduos em cada geração.

Capítulo 6

Ferramentas Para o Problema da Identificação de Genes Por Comparação de um DNA com Vários cDNAs

No Capítulo 4 descrevemos a relação entre o problema do alinhamento *spliced* múltiplo e o problema da identificação de genes por comparação de um DNA com vários cDNAs, que consiste em usar um DNA como a sequência base, cDNAs como as sequências modelo e um conjunto com possíveis éxons do DNA como o conjunto de blocos. Essa relação possibilitou-nos desenvolver quatro ferramentas para o problema da identificação de genes por comparação de um DNA com vários cDNAs utilizando as heurísticas que propusemos para o problema do alinhamento *spliced* múltiplo. Uma vez implementadas, essas ferramentas foram comparadas entre si e com outras disponíveis na literatura, no intuito de determinarmos quão precisos são seus resultados.

Iniciaremos este capítulo falando sobre como criamos as ferramentas baseadas nas heurísticas desenvolvidas. Na seção seguinte é apresentada uma revisão de um conjunto de medidas de avaliação de predições de genes. Após isso descrevemos os passos executados na criação de um conjunto para teste de ferramentas de identificação de genes. Por fim apresentamos os resultados da avaliação experimental das nossas ferramentas.

6.1 Transformando as heurísticas em ferramentas

As heurísticas desenvolvidas foram utilizadas na implementação de quatro ferramentas de identificação de genes distintas. Os passos executados por essas ferramentas consistem, basicamente, dos passos que compõem sua respectiva heurística. A única diferença está no fato das ferramentas receberem como entrada apenas a sequência g, referente ao DNA e o conjunto $\mathscr{T}$, que equivale às sequências de cDNA. Isso leva a necessidade de um pré-processamento da sequência de DNA para gerar os blocos (éxons) componentes do conjunto $\mathscr{B}$.

O método mais natural para a geração dos blocos consiste em encontrar todos os

possíveis códons de iniciação/parada e sítios de aceitação/doação na sequência de DNA e criar um conjunto $\mathscr{B}$ com todos os segmentos da sequência delimitados por esses sinais. Essa abordagem, apesar de adequada do ponto de vista teórico, não é aplicável devido à grande quantidade de blocos que seriam gerados ao final.

Dada a impossibilidade de processar todos os possíveis éxons existentes no DNA dado como entrada, decidimos pelo uso da ferramenta `GENSCAN`, descrita no Capítulo 3, para a criação do conjunto $\mathscr{B}$. A criação do conjunto $\mathscr{B}$ se dá pela execução do `GENSCAN` utilizando a sequência de DNA como entrada, com a opção `-subopt` definida com o seu valor mínimo (0.01). A opção `-subopt` faz com que o `GENSCAN` devolva, além de uma predição de genes, todos os possíveis éxons cuja pontuação de acordo com o seu HMM é maior do que o valor de `-subopt`, mas que não participam de nenhum gene predito. Esses éxons, além dos éxons pertencentes aos genes preditos, correspondem aos elementos do conjunto $\mathscr{B}$.

Com a utilização do `GENSCAN` para criar os conjuntos $\mathscr{B}$ em instâncias reais do problema da identificação de genes, obtivemos conjuntos com uma quantidade de blocos aceitável para a execução das ferramentas propostas. Além disso, o conjunto de blocos devolvido pelo `GENSCAN` só inclui blocos com alguma relevância biológica.

Além do conjunto $\mathscr{B}$, foi necessário definir uma função de pontuação para podermos utilizar as heurísticas como ferramentas de identificação de genes. Utilizamos a distância de Levenshtein como função de pontuação.

As ferramentas, assim como as heurísticas, foram implementadas na linguagem `ANSI C`. Elas recebem como entrada um arquivo, no formato `FASTA`, contendo uma sequência de DNA e um arquivo contendo nomes de vários outros arquivos onde as sequências de cDNA, também no formato `FASTA`, podem ser encontradas. Os códigos fonte de todas as nossas implementações estão disponíveis no endereço *http://projetomestrado.googlecode.com/* (último acesso em 13 de dezembro de 2010).

6.2 Avaliação das ferramentas

Por estarem baseadas em heurísticas, é de extrema importância medir a exatidão das ferramentas desenvolvidas[1]. Por isso, executamos nossas ferramentas com dados reais e avaliamos os resultados, comparando-os com os de algumas ferramentas de identificação de genes disponíveis na literatura.

Os testes com as ferramentas foram realizados no mesmo computador onde foram executados os testes das heurísticas e foram feitos em duas fases. Na primeira fase comparamos as nossas ferramentas com ferramentas de identificação de genes por comparação de um DNA com um cDNA, utilizando dados reais. Na segunda fase comparamos as nossas ferramentas com a ferramenta `TWINSCAN`, mais uma vez utilizando dados reais. As ferramentas de identificação de genes são referenciadas com os mesmos nomes da heurísticas em que são baseadas.

Antes de falar sobre os testes, vamos falar sobre um conjunto de medidas de predições de genes, que utilizamos na avaliação de nossas ferramentas.

[1] Observe que mesmo que existisse uma solução ótima para o PASM, ela não seria necessariamente uma solução ótima para o problema da identificação de genes.

6.2.1 Medidas de avaliação de predições de genes

Nossas avaliações foram feitas através de algumas medidas tradicionais denominadas especificidade e sensibilidade, além de outras medidas derivadas dessas duas. Essas medidas de avaliação encontram-se detalhadas em [10] e são normalmente calculadas em dois níveis distintos: o de nucleotídeos e o de éxons. No nível de nucleotídeos, os cálculos são feitos com base na contagem de nucleotídeos corretamente preditos como codificantes/não-codificantes (verdadeiros positivos ou **VP**/ verdadeiros negativos ou **VN**) e de nucleotídeos erroneamente preditos como codificantes/não-codificantes (falsos positivos ou **FP**/ falsos negativos ou **FN**).

Dados os valores de VP, VN, FP e FN a sensibilidade e a especificidade no nível de nucleotídeos é calculada através das seguintes fórmulas:

$$S_n = \frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FN}} \qquad S_p = \frac{\mathrm{VN}}{\mathrm{VN} + \mathrm{FP}}. \tag{6.1}$$

A sensibilidade (S_n) é a proporção de nucleotídeos preditos como codificantes de maneira correta. A especificidade (S_p) é a proporção de nucleotídeos preditos como não-codificantes de maneira correta. Entretanto, como a quantidade de nucleotídeos não-codificantes nas sequências de DNA é muito maior do que a frequência de nucleotídeos codificantes, o valor VN tende a ser muito maior do que FP fazendo com que a especificidade calculada dessa maneira produza valores altos e pouco informativos. Por esse motivo, a especificidade geralmente é calculada utilizando a seguinte fórmula, que é a que adotamos em nosso trabalho:

$$S_p = \frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FP}}. \tag{6.2}$$

Há uma terceira medida que relaciona VP, VN, FP e FN, chamada de correlação aproximada (**CA**) definida como

$$\mathrm{CA} = \frac{1}{2}\left(\frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FN}} + \frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FP}} + \frac{\mathrm{VN}}{\mathrm{VN} + \mathrm{FP}} + \frac{VN}{VN + FN}\right) - 1. \tag{6.3}$$

A medida CA sumariza S_n e S_p em uma medida global de precisão de uma predição de genes.

No nível de éxons, a especificidade e a sensibilidade são calculadas com base na contagem de éxons corretamente preditos **NEC**, a quantidade total de éxons preditos **NEP** e a quantidade total de éxons na sequência anotada **NEA**. Dados os valores de NEC, NEP e NEA, calculamos a sensibilidade (S_{n_e}) e a especificidade (S_{p_e}) no nível de éxons por meio das seguintes fórmulas

$$S_{n_e} = \frac{\mathrm{NEC}}{\mathrm{NEA}} \qquad S_{p_e} = \frac{\mathrm{NEC}}{\mathrm{NEP}}. \tag{6.4}$$

Assim como na medida CA, sumarizamos S_{n_e} e S_{p_e} na medida

$$\mathrm{Av}_e = \frac{S_{n_e} + S_{p_e}}{2}. \tag{6.5}$$

Além da avaliação nos níveis de nucleotídeos e de éxons, realizamos ainda uma avaliação no nível de bordas. Chamamos de bordas as posições de início e de término de um éxon. No nível de bordas, um éxon com pelo menos uma das bordas consistente com a de um éxon de uma sequência anotada é considerado corretamente predito. As fórmulas utilizadas para a avaliação no nível de bordas são baseadas nas de avaliação no nível de éxons. Sejam **NBC** a quantidade de bordas preditas corretamente, **NEP** a quantidade de éxons preditos e **NEA** a quantidade de éxons na sequência anotada. A sensibilidade (S_{n_b}) e a especificidade (S_{p_b}) no nível de bordas são dadas por

$$S_{n_b} = \frac{\text{NBC}}{2 \times \text{NEP}} \qquad S_{p_b} = \frac{\text{NBC}}{2 \times \text{NEA}}. \tag{6.6}$$

A medida que sumariza a S_{n_b} e a S_{p_b} é

$$\text{Av}_b = \frac{S_{n_b} + S_{p_b}}{2}. \tag{6.7}$$

6.2.2 Comparativo com ferramentas que utilizam uma sequência transcrita

Os resultados da avaliação das heurísticas indicam que a `HEURISTICA1` (heurística da sequência central) é a mais apropriada para o problema do alinhamento *spliced* múltiplo. Esse fato, entretanto, não significa que a ferramenta baseada nessa heurística é a melhor alternativa para o problema da identificação de genes por comparação de um DNA com vários cDNAs. Por esse motivo, e também pelo fato de que uma boa resposta para o problema do alinhamento *spliced* múltiplo não é obrigatóriamente uma boa resposta para o problema da identificação de genes por comparação de um DNA com vários cDNAs, testamos as quatro ferramentas que propusemos em instâncias que representavam entradas reais para o problema da identificação de genes.

Além de comparar as nossas ferramentas entre si, comparamos os seus resultados com os de outras ferramentas de identificação de genes por comparação de uma sequência de DNA com uma ou mais sequências de cDNA (ou ESTs) disponíveis na literatura. São elas o `AUGUSTUS+`, `EST2GENOME` (versão atualizada do `EST_GENOME`), `SIM4`, `GENESEQER`, `SPIDEY` e `PROCRUSTES`. Dentre elas o `AUGUSTUS+` e o `SPIDEY` podem receber mais de uma sequência de cDNA (ou EST) como entrada, apesar de processá-las individualmente. A comparação dos nossos resultados com os de outras ferramentas nos permite verificar quão próximas ou distantes estão as nossas ferramentas de outras já desenvolvidas e utilizadas na tarefa da anotação de sequências de DNA.

Conjunto de testes

O conjunto de testes dessa avaliação foi criado utilizando dados do projeto ENCODE, descrito em [49]. O objetivo do projeto ENCODE é identificar todos os elementos funcionais do genoma humano (*Homo sapiens*). Em sua fase inicial foram analisadas 44 regiões que correspondem a cerca de 1% desse genoma (aproximadamente 30 milhões de bases).

Para a criação do conjunto de testes, realizamos uma varredura nas 44 regiões já analisadas pelo projeto ENCODE buscando por todos os seus genes. Analisamos cada um dos genes encontrados e escolhemos aqueles que não apresentavam características incomuns para compor nosso conjunto de testes, ou seja, escolhemos aqueles que codificassem uma e somente uma proteína. Então, pseudogenes e genes com *splicing* alternativo foram descartados. Além disso, genes cujos tamanhos não eram múltiplos de 3 ou que não obedeciam à estrutura de éxons e íntrons com seus respectivos sítios de doação/aceitação e códons de iniciação e de parada descrita no Capítulo 2 também foram descartados.

Feita a escolha dos genes, eles foram extraídos das regiões onde se encontram juntamente com as 1000 bases que os antecedem e os sucedem nessas regiões. Essas sequências correspondem às sequências alvo das nossas instâncias do problema da identificação de genes por comparação de um DNA com vários cDNAs. Completamos cada instância incluindo até 5 sequências de cDNA de genes homólogos ao gene da sequência alvo. Essas sequências foram obtidas da base de dados Homologene [56], disponível no *site* do NCBI (*National Center for Biotechnology Information*). As sequências de cDNA foram escolhidas por ordem de semelhança com a sequência alvo. Nessa fase, algumas instâncias foram descartadas porque os genes das sequências alvo não possuíam nenhuma sequência homóloga de um organismo de espécie diferente do *Homo sapiens*.

As espécies que apresentaram maior número de genes homólogos aos do *Homo sapiens* foram, respectivamente o *Pan troglodytes*, o *Canis lupus familiaris*, o *Bos taurus*, o *Mus musculus* e o *Drosophila melanogaster*. No final obtivemos 240 instâncias de teste para avaliar nossas ferramentas de identificação de genes por comparação de um DNA com cDNA(s). Ao executar as ferramentas de identificação de genes por comparação de um DNA com um cDNA foi utilizada, para cada instância, a sequência de cDNA mais semelhante ao gene codificado na sequência de DNA. O conjunto de instâncias de teste que criamos está disponível no endereço *http://genesparateste.googlecode.com/* (último acesso em 13 de dezembro de 2010) e algumas informações sobre elas podem ser encontradas no Apêndice A.

Os genes das instâncias do nosso conjunto de testes possuem, em média, 7 éxons. O gene da instância `URB1`, formado por 39 éxons, é o gene com mais éxons do conjunto de teste. O `GENSCAN` gerou, em média, 55 possíveis éxons para cada sequência alvo do conjunto de testes. A instância `SND1`, com 622 possíveis éxons gerados pelo `GENSCAN`, é a instância com o maior número de éxons possíveis do conjunto de teste.

Resultados obtidos

Os resultados das qualidades das predições das nossas ferramentas e das ferramentas de identificação de genes por comparação de um DNA com um cDNA utilizando as instâncias do conjunto de testes criado a partir dos dados do projeto ENCODE podem ser vistos na Tabela 6.1.

Observe na Tabela 6.1 que a ferramenta `EST2GENOME` foi a que obteve os melhores resultados na avaliação no nível de nucleotídeos. A `HEURISTICA1`, apesar de ter se saído pior do que a `EST2GENOME` no nível de nucleotídeos, ficou muito próxima dela e foi a melhor dentre as nossas ferramentas nesse nível de avaliação. As ferramentas `HEURISTICA1` e `HEURISTICA3` obtiveram os melhores resultados nos níveis de éxons e de

Ferramenta	Nucleotídeos			Éxons			Bordas		
	$\mathbf{S_n}$	$\mathbf{S_p}$	**CA**	$\mathbf{S_{n_e}}$	$\mathbf{S_{p_e}}$	$\mathbf{Av_e}$	$\mathbf{S_{n_b}}$	$\mathbf{S_{p_b}}$	$\mathbf{Av_b}$
AUGUSTUS+	0.88	0.89	0.84	0.73	0.74	0.74	0.80	0.81	0.81
EST2GENOME	**0.96**	**0.97**	**0.96**	0.78	0.78	0.78	0.86	0.85	0.85
GENESEQER	0.84	0.91	0.85	0.67	0.64	0.65	0.74	0.71	0.72
PROCRUSTES	0.94	0.94	0.93	0.82	0.77	0.80	0.88	0.82	0.85
SIM4	0.92	0.96	0.93	0.73	0.73	0.73	0.82	0.80	0.81
SPIDEY	0.92	0.96	0.93	0.68	0.68	0.68	0.78	0.76	0.77
HEURISTICA1	0.95	0.96	0.95	**0.85**	0.80	0.82	**0.90**	0.84	0.87
HEURISTICA2	**0.96**	0.91	0.92	0.82	0.71	0.77	0.89	0.76	0.82
HEURISTICA3	0.93	0.96	0.94	**0.85**	**0.83**	**0.84**	0.89	**0.87**	**0.88**
HEURISTICA4	0.77	0.80	0.77	0.54	0.50	0.52	0.64	0.59	0.62

Tabela 6.1: Resultados dos testes realizados com 240 instâncias obtidas dos dados do projeto ENCODE. As colunas $\mathbf{S_n}$, $\mathbf{S_p}$, **CA**, $\mathbf{S_{n_e}}$, $\mathbf{S_{p_e}}$, $\mathbf{Av_e}$, $\mathbf{S_{n_b}}$, $\mathbf{S_{p_b}}$, $\mathbf{Av_b}$ contêm os valores das médias aritméticas das medidas de sensibilidade, especificidade e das medidas que as sumarizam nos níveis de nucleotídeos, éxons e bordas.

bordas, sendo que a HEURISTICA3 superou todas as outras ferramentas nas médias das medidas Av_e e Av_b.

As nossas ferramentas de identificação de genes, com exceção da ferramenta baseada no algoritmo genético, apresentaram bons resultados em todos os níveis de avaliação. As ferramentas que utilizam o algoritmo de alinhamento *spliced* obtiveram resultados melhores do que o PROCRUSTES na maioria dos casos, indicando que o uso de vários cDNAs melhora as predições de genes por comparação de DNA com cDNA. Podemos considerar a HEURISTICA3 como a ferramenta que se saiu melhor dentre todas as avaliadas, já que obteve os melhores resultados em todas as medidas de avaliação no nível de éxons e seus resultados são próximos aos dos melhores nas medidas de avaliação dos outros níveis.

Ao analisar isoladamente as instâncias para as quais nossas ferramentas devolveram soluções ruins, descobrimos alguns fatos interessantes. Como todas as nossas ferramentas utilizam o GENSCAN para gerar o conjunto de possíveis éxons, elas falham nos casos onde o GENSCAN não inclui os éxons reais no conjunto de possíveis éxons. Um outro problema causado pelo GENSCAN é a inclusão, no conjunto de possíveis éxons, de éxons falsos sobrepostos a éxons reais. Em alguns casos um éxon falso sobreposto a um éxon real é escolhido no lugar daquele éxon real. Esse fato afetou principalmente a HEURISTICA3. Na instância ZNF418, por exemplo, há um éxon de coordenadas 5819..5945 e dois cDNAs homólogos. Um dos alinhamentos *spliced* entre a sequência de DNA e um dos cDNAs sugere, corretamente, o possível éxon de coordenadas 5819..5945 para fazer parte da resposta final. O outro alinhamento *spliced*, no entanto, sugere o possível éxon de coordenadas 5816..5945. Apesar de ser óbvio que ambas as soluções se referiam ao mesmo éxon, ele não foi utilizado na predição.

A HEURISTICA2 tem o seu desempenho afetado quando um dos cDNAs possui um segmento não presente nos outros cDNAs, geralmente um prefixo ou um sufixo. Isso faz com que a sequência consenso seja alongada em uma de suas extremidades graças a esse trecho causando a inclusão de um ou mais blocos adicionais pelo algoritmo de alinha-

mento *spliced* para serem alinhados a ele. Essa característica da sequência consenso, entretanto, foi interessante em alguns casos, como na instância `OR52b6`, que contém um gene com um éxon único de coordenadas 1001..2008. Dois dos três cDNAs homólogos não possuíam o início desse éxon. Nesse caso a `HEURISTICA2` identificou corretamente o éxon, ao contrário da `HEURISTICA3`, que falhou pois duas das três soluções individuais indicavam que o éxon deveria possuir as coordenadas 1064..2008, e da `HEURISTICA1`, que também identificou o possível éxon de coordenadas 1064..2008 como sendo o verdadeiro, já que a sequência central é um dos cDNAs homólogos sem o trecho inicial do éxon.

A `HEURISTICA4` apresentou resultados bem piores do que os das outras ferramentas que propusemos. Analisando as soluções devolvidas por ela, é possível encontrar vários casos onde ajustes nos parâmetros talvez pudessem melhorar os resultados.

A média dos tempos de execução de cada uma das ferramentas pode ser vista na Tabela 6.2.

Ferramenta	Tempo Médio	Pior Tempo
AUGUSTUS+	2.43	35.48
EST2GENOME	9.44	180.60
GENESEQER	4.87	79.77
PROCRUSTES	2.34	64.69
SIM4	**0.03**	0.84
SPIDEY	0.04	**0.13**
HEURISTICA1	2.70	63.89
HEURISTICA2	2.99	70.70
HEURISTICA3	8.11	278.15
HEURISTICA4	104.37	1854.40

Tabela 6.2: Tempos de execução dos testes realizados com 240 instâncias obtidas dos dados do projeto ENCODE. A coluna Tempo Médio contém as médias aritméticas dos tempos de execução sobre todas as instâncias para cada ferramenta, em segundos. A coluna Pior Tempo contém o pior tempo de execução de cada ferramenta no conjunto de instâncias, em segundos.

Como pode ser observado na Tabela 6.2, as ferramentas `SPIDEY` e `SIM4` obtiveram tempos de execução muito melhores do que as outras ferramentas, mas em contrapartida seus resultados no nível de éxons ficaram bem abaixo dos de nossas ferramentas. A ferramenta que apresentou os piores resultados foi aquela baseada na `HEURISTICA4`. A justificativa para isso é, novamente, a enorme quantidade de operações realizadas para a classificação dos indivíduos em cada geração.

6.2.3 Comparativo com uma ferramenta baseada no GENSCAN

Na seção anterior, detalhamos os resultados da comparação de nossas ferramentas com algumas ferramentas de identificação de genes por comparação de um DNA com um cDNA. Como as nossas ferramentas dependem do `GENSCAN` para funcionar, realizamos uma comparação adicional, dessa vez com uma ferramenta também baseada no `GENSCAN`,

o TWINSCAN. Nossa intenção com essa rodada adicional de testes foi verificar quão boas são nossas ferramentas em relação a uma ferramenta desenvolvida para melhorar os resultados do GENSCAN.

Pretendíamos utilizar o mesmo conjunto de testes da seção anterior, mas como o TWINSCAN precisa de uma sequência de DNA[2] adicional, ao invés de sequências de cDNA adicionais como é o caso das nossas ferramentas, tivemos que modificá-lo. Realizamos uma busca na base de dados Homologene do NCBI por trechos de sequências de DNA contendo genes homólogos ao gene da sequência alvo de cada instância do nosso conjunto de testes baseado no projeto ENCODE. Os genes que escolhemos eram provenientes do *Mus musculus* ou de outros organismos entre *Rattus norvegicus*, *Bos taurus*, *Canis lupus familiaris*, *Danio rerio*, *Pan troglodytes* e *Drosophila melanogaster*. Genes de organismos diferentes do *Mus musculus* foram escolhidos quando não havia um gene do *Mus musculus* disponível. As sequências de DNA informantes foram criadas tomando-se o trecho contendo o gene homólogo em um organismo diferente do *Homo sapiens* acrescido das 1000 bases da sequência de DNA anteriores e posteriores ao gene. Assim como na escolha da sequência alvo, foram descartados os genes homólogos que não possuíam a estrutura de um gene descrita no Capítulo 2, e também os pseudogenes e genes com *splicing* alternativo.

O conjunto de testes que utilizamos para testar nossas ferramentas e o TWINSCAN é composto por 232 instâncias, derivadas do conjunto de testes da seção anterior. Das 240 instâncias do conjunto de testes da seção anterior, 6 foram descartadas por não termos conseguido encontrar uma sequência informante e 2 foram descartadas por problemas na execução do TWINSCAN. Os resultados das qualidades das predições das nossas ferramentas e do TWINSCAN, e a média dos tempos de execução de cada uma delas podem ser vistos nas Tabelas 6.3 e 6.4, respectivamente.

Ferramenta	Nucleotídeos			Éxons			Bordas		
	S_n	S_p	CA	S_{n_e}	S_{p_e}	Av_e	S_{n_b}	S_{p_b}	Av_b
TWINSCAN	0.76	0.90	0.78	0.50	0.63	0.56	0.60	0.75	0.67
HEURISTICA1	**0.96**	**0.96**	**0.95**	**0.86**	0.81	0.83	**0.91**	0.85	0.88
HEURISTICA2	**0.96**	0.91	0.93	0.83	0.73	0.78	0.89	0.77	0.83
HEURISTICA3	0.93	0.96	0.94	**0.86**	**0.84**	**0.85**	0.89	**0.88**	**0.89**
HEURISTICA4	0.77	0.80	0.77	0.54	0.51	0.53	0.64	0.60	0.62

Tabela 6.3: Resultados dos testes realizados com 232 instâncias obtidas dos dados do projeto ENCODE. As colunas desta tabela são semelhantes às da Tabela 6.1.

Ferramenta	Tempo Médio	Pior Tempo
TWINSCAN	266.02	3736.57
HEURISTICA1	**2.67**	**63.89**
HEURISTICA2	2.97	70.70
HEURISTICA3	7.96	278.15
HEURISTICA4	106.15	1854.40

Tabela 6.4: Tempos de execução dos testes realizados com 232 instâncias obtidas dos dados do projeto ENCODE. As colunas desta tabela são semelhantes às da Tabela 6.2.

[2]Essa sequência adicional é a sequência informante, que descrevemos na Seção 3.3.6.

Na Tabela 6.3 é possível observar que, com exceção da ferramenta baseada no algoritmo genético, todas as ferramentas que implementamos obtiveram resultados melhores que o `TWINSCAN` em todas as medidas de avaliação. Além disso, de acordo com a Tabela 6.4, todas as nossas ferramentas obtiveram médias de tempos de execução melhores que o `TWINSCAN`.

Os resultados obtidos sugerem que a comparação de uma sequência de DNA com várias sequências de cDNA fornece informações mais precisas para a localização de genes em uma sequência alvo do que simplesmente a comparação de uma sequência de DNA com uma outra sequência, seja ela de DNA ou de cDNA.

Capítulo 7

Conclusão

Alguns estudos mostram que ainda não existe uma solução computacional satisfatória para o Problema da Identificação de Genes, o que levou-nos a estudar esse problema no intuito de desenvolver novas ferramentas para ele. As soluções propostas nesse trabalho baseiam-se na comparação de uma sequência de DNA com várias sequências de cDNA, considerando-se a hipótese de que, ao aumentar a quantidade de sequências transcritas comparadas com o DNA, predições mais exatas podem ser obtidas.

Modelamos o problema da identificação de genes por comparação de uma sequência de DNA utilizando várias sequências de cDNA através de um problema matemático formulado por nós: o Problema do Alinhamento *Spliced* Múltiplo. Mostramos qu esse novo problema é NP-completo, por isso propusemos quatro heurísticas para ele.

As heurísticas que propusemos foram avaliadas experimentalmente, com sequências geradas aleatoriamente, e os resultados obtidos indicam que as heurísticas `HEURISTICA1` (heurística da sequência central) e `HEURISTICA2` (heurística da sequência consenso) devolvem melhores soluções para o problema do alinhamento *spliced* múltiplo do que as heurísticas `HEURISTICA3` (heurística do consenso de blocos) e `HEURISTICA4` (heurística do algoritmo genético), com uma pequena vantagem obtida pela `HEURISTICA1`. Acreditamos que a heurística `HEURISTICA3` não obteve bons resultados pelo fato de que as sequências modelo das instâncias artificiais eram muito diferentes entre si. Isso fez com que as soluções individuais dos alinhamentos *spliced* entre a sequência base e cada sequência modelo nem sempre apresentassem blocos comuns à maioria das sequências. Acreditamos ainda que a `HEURISTICA4` não obteve bons resultados por não termos realizados experimentos para ajustar seus parâmetros. Uma análise dos tempos de execução das heurísticas apontam que a `HEURISTICA1` é a mais indicada para aplicações onde o tempo de execução é importante, já que ela obteve o menor tempo médio e também o menor pior tempo de execução.

Ao final da avaliação das heurísticas, nosso trabalho prosseguiu com o desenvolvimento e avaliação experimental de quatro ferramentas para o problema da identificação de genes por comparação de uma sequência de DNA com várias sequências de cDNA. Essa avaliação foi dividida em duas fases. Na primeira fase, além de nossas ferramentas, executamos também outras seis ferramentas de identificação de genes baseadas em comparação de uma sequência de DNA com uma ou mais sequências transcritas.

Com exceção da HEURISTICA4, as nossas ferramentas apresentaram bons resultados. A HEURISTICA3 superou todas as outras ferramentas em todas as medidas no nível de éxons e em quase todas no nível de bordas. A ferramenta EST2GENOME obteve resultados um pouco melhores do que as nossas ferramentas no nível de nucleotídeos, mas seus resultados foram bem piores no nível de éxons. As nossas ferramentas baseadas em heurísticas que utilizam o algoritmo de alinhamento *spliced* obtiveram resultados melhores do que o PROCRUSTES, indicando que comparar uma sequência de DNA com várias sequências de cDNA aumenta a quantidade de evidências sobre a localização dos genes na sequência de DNA.

Na segunda fase da avaliação, comparamos os resultados obtidos pelas nossas ferramentas com a ferramenta TWINSCAN, para verificar o seu desempenho em relação a uma outra ferramenta baseada no GENSCAN. Nessa fase, as ferramentas HEURISTICA1, HEURISTICA2 e HEURISTICA3 obtiveram melhores resultados que o TWINSCAN em todos os níveis de avaliação de predições de genes. A HEURISTICA4 apresentou resultados piores, porém próximos aos do TWINSCAN. Com isso demonstramos que a precisão de nossas ferramentas não se deve somente ao fato delas serem baseadas no GENSCAN.

Durante o trabalho que desenvolvemos, várias idéias surgiram como sugestões para trabalhos futuros. São elas:

- Desenvolver um método eficiente para a geração de blocos;
- Determinar a complexidade do Problema do Alinhamento *Spliced* Múltiplo para outras funções de pontuação;
- Tentar encontrar aproximações para o Problema do Alinhamento *Spliced* Múltiplo;
- Realizar experimentos para verificar a influência, na solução, da quantidade de sequências de cDNA comparadas com a sequência de DNA;
- Estudar melhorias para a heurística do algoritmo genético, como técnicas para a obtenção de boas populações iniciais e ajuste dos seus parâmetros;
- Estudar versões do problema do Alinhamento *Spliced* Múltiplo onde estamos interessados em soluções subótimas para ele.

Os resultados obtidos ao final deste trabalho mostram que atingimos os objetivos inicialmente propostos com o estudo detalhado do Problema da Identificação de Genes e o desenvolvimento de quatro ferramentas de predição com resultados em geral melhores do que os de outras ferramentas disponíveis na literatura.

Apêndice A

Conjunto de instâncias e resultados individuais

Apresentamos na Tabela A.1 algumas informações sobre o conjunto de instâncias construído com base no genoma humano e os resultados obtidos pelas nossas ferramentas para cada uma dessas instâncias. A coluna **Gene** contém o nome do gene contido na sequência alvo de cada uma das instâncias. Esse nome é utilizado também para identificar a própria instância. A coluna $\mathbf{T_{DNA}}$ contém o tamanho da sequência de DNA de cada uma das instâncias. A coluna $\mathbf{T_c}$ contém a média dos tamanhos das sequências de cDNA de cada uma das instâncias. A coluna $\mathbf{Q_c}$ contém a quantidade de cDNAs de cada uma das instâncias. As colunas $\mathbf{Q_{ea}}$ e $\mathbf{Q_{eg}}$ contêm, respectivamente, as quantidades de éxons anotados e de possíveis éxons devolvidos pelo `GENSCAN` de cada uma das instâncias. As colunas **CA**, $\mathbf{Av_e}$ e $\mathbf{Av_b}$ contêm os valores das medidas de avaliação de predições de genes descritas no Capítulo 6, para as predições feitas pelas ferramentas **H1** (`HEURISTICA1`), **H2** (`HEURISTICA2`), **H3** (`HEURISTICA3`) e **H4** (`HEURISTICA4`).

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	CA				Av_e				Av_b			
						H1	H2	H3	H4	H1	H2	H3	H4	H1	H2	H3	H4
a1bg	8694	1546	5	8	40	0.80	0.77	0.80	0.54	0.75	0.68	0.75	0.15	0.81	0.73	0.81	0.29
aff4	90284	3488	5	20	99	0.98	0.98	0.98	0.78	0.87	0.87	0.87	0.44	0.89	0.89	0.89	0.49
alg10b	14972	1422	1	3	27	0.29	0.29	0.29	0.20	0.44	0.44	0.44	0.22	0.44	0.44	0.44	0.22
ankrd10	38530	1208	5	6	45	0.63	0.59	0.63	0.52	0.67	0.46	0.67	0.27	0.75	0.62	0.75	0.33
ankrd43	5457	1729	4	1	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arhgdig	4398	700	5	6	21	1.00	1.00	1.00	0.85	1.00	1.00	1.00	0.42	1.00	1.00	1.00	0.62
ascc2	51655	2253	5	19	80	0.97	0.97	0.98	0.50	0.74	0.74	0.79	0.26	0.87	0.87	0.89	0.39
ascl2	4455	733	5	1	8	1.00	0.77	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.33	1.00	1.00
asz1	66302	1428	5	13	84	0.90	0.90	0.90	-0.01	0.89	0.89	0.86	0.00	0.89	0.89	0.86	0.00
bad	16877	561	4	3	47	0.99	0.91	0.99	0.75	0.67	0.58	0.67	0.33	0.83	0.73	0.83	0.50
bet1	14691	355	5	4	26	0.58	0.58	0.57	0.55	0.62	0.62	0.59	0.59	0.62	0.62	0.59	0.59
bgn	16594	1253	5	7	80	0.99	0.81	0.99	0.77	0.86	0.68	0.86	0.31	0.93	0.74	0.93	0.46
c17orf50	6183	555	2	3	16	1.00	0.98	1.00	0.61	1.00	0.80	1.00	0.67	1.00	0.80	1.00	0.67
c20orf152	64094	1868	3	12	60	0.99	0.99	0.99	0.82	0.88	0.88	0.88	0.48	0.92	0.92	0.92	0.64
c21orf45	12847	682	5	5	44	0.88	0.88	0.88	0.66	0.80	0.80	0.80	0.22	0.80	0.80	0.80	0.34
c21orf59	12572	1019	4	7	46	1.00	0.73	0.90	0.69	1.00	0.67	0.93	0.57	1.00	0.80	0.93	0.64
c21orf63	104946	1321	5	8	172	0.96	0.95	0.95	0.29	0.83	0.76	0.79	0.11	0.83	0.76	0.79	0.16
c7orf63	67164	2826	1	22	69	0.77	0.77	0.77	0.51	0.64	0.64	0.64	0.24	0.68	0.68	0.68	0.34
c7orf68	4589	636	2	1	13	1.00	0.56	1.00	0.63	1.00	0.00	1.00	0.62	1.00	0.33	1.00	0.62
calcr	151952	1596	5	12	135	0.81	0.70	0.83	0.35	0.73	0.67	0.80	0.24	0.77	0.67	0.84	0.28
capza2	58751	847	5	10	41	1.00	1.00	1.00	0.89	1.00	1.00	1.00	0.70	1.00	1.00	1.00	0.75
cars2	66705	1881	4	15	118	0.89	0.82	0.81	0.53	0.83	0.65	0.79	0.29	0.86	0.74	0.83	0.32
cav1	38392	537	5	3	42	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.53	1.00	1.00	1.00	0.53
ccdc157	22192	1975	3	10	82	0.80	0.93	0.86	0.56	0.55	0.76	0.67	0.28	0.60	0.81	0.72	0.32
ccdc88b	19312	4738	4	27	68	0.99	0.94	0.96	0.86	0.96	0.82	0.96	0.59	0.96	0.86	0.96	0.71
ccdc93	100551	2003	5	24	176	0.95	0.93	0.97	0.33	0.87	0.88	0.90	0.19	0.87	0.92	0.92	0.28
ccl5	10883	259	5	3	9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
cdh8	384802	2349	5	11	380	1.00	1.00	1.00	0.34	1.00	1.00	1.00	0.15	1.00	1.00	1.00	0.15
cdx2	9040	913	5	3	37	1.00	1.00	1.00	0.87	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67
cftr	190703	4446	5	27	216	0.92	0.92	0.92	0.10	0.85	0.85	0.85	0.10	0.87	0.87	0.87	0.17
chmp2a	5554	667	5	5	12	1.00	1.00	1.00	0.81	1.00	1.00	1.00	0.45	1.00	1.00	1.00	0.56
cldn12	14473	735	5	1	29	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
cpne8	255419	1730	5	20	177	0.89	0.88	0.89	0.42	0.77	0.75	0.77	0.16	0.82	0.80	0.82	0.24

(continua na próxima página)

Tabela A.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	CA				Av_e				Av_b			
						H1	H2	H3	H4	H1	H2	H3	H4	H1	H2	H3	H4
csf2	3375	432	4	4	9	1.00	1.00	1.00	0.85	1.00	1.00	1.00	0.58	1.00	1.00	1.00	0.73
ctgf	5203	1045	5	5	11	0.94	0.94	0.94	0.93	0.80	0.80	0.80	0.60	0.80	0.80	0.80	0.70
ctsd	13238	1224	5	9	46	1.00	1.00	1.00	0.38	1.00	1.00	1.00	0.33	1.00	1.00	1.00	0.39
ddx18	19699	2009	5	14	55	1.00	1.00	1.00	0.89	1.00	1.00	1.00	0.59	1.00	1.00	1.00	0.72
ddx43	25005	1985	4	16	43	1.00	1.00	1.00	0.92	1.00	1.00	1.00	0.65	1.00	1.00	1.00	0.77
decr2	12630	887	5	8	44	1.00	1.00	1.00	0.71	1.00	1.00	1.00	0.38	1.00	1.00	1.00	0.50
dnajc4	6001	717	5	6	28	1.00	1.00	1.00	0.71	0.83	0.83	0.83	0.50	0.92	0.92	0.92	0.58
dppa5	3167	357	1	3	19	1.00	1.00	1.00	0.83	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.42
drg1	36634	1104	5	9	32	1.00	1.00	1.00	0.77	1.00	1.00	1.00	0.44	1.00	1.00	1.00	0.49
dusp18	7834	576	5	1	16	1.00	0.99	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
eef1a1	7283	1389	5	7	15	1.00	1.00	1.00	0.87	1.00	1.00	1.00	0.71	1.00	1.00	1.00	0.79
esrra	13167	1318	5	6	39	1.00	0.91	1.00	0.80	0.83	0.58	0.83	0.56	0.92	0.73	0.92	0.62
ext1	314457	2241	5	11	283	1.00	1.00	1.00	0.53	1.00	1.00	1.00	0.09	1.00	1.00	1.00	0.09
fam71f1	18355	1032	5	7	20	0.58	0.58	0.58	0.57	0.29	0.29	0.29	0.00	0.43	0.43	0.43	0.27
fkbp2	5195	466	5	5	18	1.00	0.81	1.00	1.00	1.00	0.73	1.00	1.00	1.00	0.82	1.00	1.00
flt3	99319	3112	5	24	114	0.98	0.90	0.98	0.68	0.88	0.77	0.88	0.36	0.94	0.83	0.94	0.47
frmd6	243591	1861	5	13	209	0.99	0.99	0.99	0.26	0.92	0.92	0.92	0.14	0.96	0.96	0.96	0.17
frs3	11717	1483	5	5	29	1.00	1.00	1.00	0.80	1.00	1.00	1.00	0.34	1.00	1.00	1.00	0.51
gabra3	286198	1479	5	9	320	0.95	0.95	0.95	0.27	0.78	0.75	0.75	0.11	0.78	0.75	0.75	0.17
gabrq	17189	1926	5	9	26	0.99	0.99	1.00	0.98	0.84	0.84	1.00	0.81	0.90	0.90	1.00	0.86
gal3st1	12253	1417	5	2	27	1.00	0.84	1.00	0.89	0.50	0.00	0.50	0.00	0.75	0.31	0.75	0.21
gas2l2	10361	2614	5	6	24	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.83	1.00	1.00	1.00	0.92
gdf9	5600	1348	5	2	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gng11	6811	222	5	2	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gng2	111469	229	5	2	91	0.47	0.39	0.47	0.00	0.35	0.00	0.35	0.00	0.35	0.00	0.35	0.00
gngt1	6666	225	5	2	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
gpr137	5633	1190	5	7	40	0.81	0.81	0.81	-0.12	0.80	0.76	0.76	0.00	0.80	0.76	0.76	0.00
gsx1	3310	828	5	2	10	1.00	0.84	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
hba1	2842	429	4	3	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hba2	2864	429	3	3	14	1.00	1.00	1.00	0.81	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67
hbb	3606	442	4	3	6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hbe1	3794	470	5	3	6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hbg1	3586	463	5	3	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hbg2	3591	444	2	3	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hbq1	2844	433	2	3	6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hbz	3651	436	5	3	11	1.00	0.96	1.00	0.98	1.00	0.67	1.00	0.67	1.00	0.83	1.00	0.83
hormad2	98610	812	3	10	85	0.78	0.78	0.78	0.47	0.60	0.60	0.60	0.27	0.60	0.60	0.60	0.27
hoxa2	4422	1122	5	2	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hoxa4	4274	933	5	2	6	1.00	1.00	1.00	0.73	1.00	1.00	1.00	0.42	1.00	1.00	1.00	0.62
hoxa5	4292	853	5	2	8	0.86	0.81	0.86	0.81	0.50	0.50	0.50	0.50	0.75	0.75	0.75	0.75
hoxa6	4253	699	4	2	6	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.50	1.00	1.00	1.00	0.75
hoxa7	4962	696	4	2	15	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
hunk	132750	2146	5	11	127	0.98	0.98	0.98	0.64	0.91	0.91	0.91	0.34	0.91	0.91	0.91	0.38
il13	4937	397	5	4	15	0.95	0.95	0.95	0.95	0.75	0.75	0.75	0.75	0.88	0.88	0.88	0.88
il3	4550	459	1	5	13	1.00	1.00	1.00	0.99	1.00	1.00	1.00	0.92	1.00	1.00	1.00	0.92
il5	4079	403	5	4	8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
inpp5j	13723	1074	1	13	41	0.68	0.68	0.68	0.49	0.71	0.71	0.71	0.38	0.71	0.71	0.71	0.47
ins	3431	330	5	2	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
insig2	23548	678	5	5	33	0.82	0.82	0.82	0.86	0.73	0.69	0.73	0.37	0.73	0.69	0.73	0.55
irf1	11165	1020	5	9	24	1.00	0.90	1.00	0.55	1.00	0.95	1.00	0.44	1.00	0.95	1.00	0.50
itfg3	33319	1757	5	11	94	1.00	0.91	1.00	0.52	1.00	0.87	1.00	0.45	1.00	0.87	1.00	0.45
kcnj15	47085	1162	5	1	29	1.00	0.95	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
kcnj6	293912	1399	4	3	332	0.98	0.86	0.99	0.73	0.50	0.39	0.83	0.27	0.50	0.39	0.83	0.27
kcnk4	10711	1426	5	6	36	1.00	0.75	1.00	0.70	1.00	0.64	1.00	0.33	1.00	0.71	1.00	0.42
kcnq5	575746	2530	5	14	606	0.96	0.96	0.97	0.42	0.65	0.67	0.76	0.00	0.68	0.70	0.79	0.07
khdc1	23871	712	2	3	34	0.77	0.77	0.63	0.73	0.43	0.43	0.29	0.23	0.54	0.54	0.29	0.46
kif3a	46943	1887	5	17	59	1.00	1.00	1.00	0.65	0.94	0.94	0.94	0.46	0.94	0.94	0.94	0.46
lace1	230161	1437	5	13	240	0.93	0.93	0.96	0.28	0.71	0.77	0.82	0.08	0.71	0.77	0.82	0.08
leap2	3225	224	5	3	8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
lep	18352	504	5	2	15	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.50	1.00	1.00	1.00	0.75
lif	8355	583	5	3	17	1.00	1.00	1.00	0.74	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67
lrrc4	5879	1928	5	1	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
lyzl6	6986	447	4	4	17	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.88
magea1	6595	1230	1	1	23	0.81	0.81	0.81	0.82	0.00	0.00	0.00	0.00	0.31	0.31	0.31	0.29
magea12	5880	945	1	1	14	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
map3k1	83080	4420	5	20	104	1.00	1.00	0.94	0.73	0.90	0.90	0.92	0.48	0.95	0.95	0.95	0.54
march11	114424	1320	4	4	149	0.77	0.87	0.78	0.87	0.62	0.59	0.88	0.21	0.62	0.69	0.88	0.42
mdfi	17788	819	5	4	53	0.99	0.85	0.99	0.78	0.75	0.62	0.75	0.21	0.88	0.73	0.88	0.42
mettl2b	28196	1205	5	9	23	1.00	0.98	1.00	0.90	1.00	0.84	1.00	0.61	1.00	0.90	1.00	0.66
mier3	34526	1710	4	13	36	1.00	0.92	1.00	0.79	0.88	0.85	0.88	0.53	0.92	0.92	0.92	0.66
mmp26	6236	822	2	6	48	0.94	0.94	0.98	0.89	0.77	0.77	0.92	0.31	0.85	0.85	0.92	0.62
morc2	43588	2955	5	23	57	0.99	0.96	0.99	0.88	0.94	0.87	0.96	0.68	0.94	0.89	0.96	0.75
moxd1	107471	1843	5	12	153	1.00	1.00	1.00	0.31	1.00	0.96	1.00	0.08	1.00	0.96	1.00	0.15
mrpl23	11338	498	5	5	39	1.00	0.85	1.00	0.37	1.00	0.69	1.00	0.00	1.00	0.69	1.00	0.28
mzf1	13659	2283	5	5	29	1.00	0.96	1.00	0.91	1.00	0.69	1.00	0.40	1.00	0.77	1.00	0.50
ndst3	226290	2622	5	13	237	0.89	0.89	0.89	0.54	0.57	0.58	0.57	0.19	0.60	0.62	0.60	0.26
nfxl1	69377	2749	4	22	82	0.86	0.86	0.86	0.70	0.68	0.68	0.70	0.39	0.77	0.77	0.79	0.52
nipal1	22294	1282	5	6	81	1.00	0.91	1.00	-0.03	1.00	0.64	0.93	0.00	1.00	0.71	0.93	0.00
nme4	5563	1003	4	5	16	1.00	0.77	1.00	0.98	1.00	0.45	1.00	0.80	1.00	0.60	1.00	0.90
nr2e1	24799	1158	5	9	41	0.97	0.97	0.97	0.71	0.89	0.89	0.89	0.35	0.89	0.89	0.89	0.47
nsdhl	40397	1075	5	7	41	0.93	0.99	0.95	0.66	0.80	0.86	0.86	0.27	0.80	0.93	0.86	0.40
of51f1	2939	892	3	1	8	0.98	0.98	0.98	1.00	0.00	0.00	0.00	1.00	0.50	0.50	0.50	1.00
ooep	3238	401	3	3	17	0.96	0.96	0.96	0.87	0.67	0.67	0.67	0.67	0.83	0.83	0.83	0.83
or51a2	2942	1099	5	1	9	1.00	0.98	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67	1.00	1.00
or51a4	2942	1099	5	1	10	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
or51a7	2939	951	4	1	8	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51b2	3055	1136	4	1	5	1.00	0.87	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67	1.00	1.00
or51b4	2933	935	3	1	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51b5	2939	939	1	1	4	0.87	0.87	0.87	0.87	0.00	0.00	0.00	0.00	0.38	0.38	0.38	0.38
or51b6	2939	945	3	1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51f2	3029	980	4	1	5	0.97	1.00	1.00	0.98	0.00	1.00	1.00	0.00	0.50	1.00	1.00	0.50
or51g1	2966	955	5	1	12	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51g2	2945	956	4	1	4	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
or51i1	2945	945	4	1	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51i2	2939	940	5	1	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

(continua na próxima página)

Tabela A.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	CA				Av_e				Av_b			
						H1	H2	H3	H4	H1	H2	H3	H4	H1	H2	H3	H4
or51l1	2948	993	4	1	2	1.00	1.00	1.00	1.00	0.75	0.75	1.00	1.00	0.75	0.75	1.00	1.00
or51m1	2981	972	4	1	7	1.00	1.00	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.75	1.00	1.00
or51q1	2954	954	5	1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51s1	2972	955	3	1	7	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or51t1	3065	1060	5	1	4	0.94	0.96	0.96	0.96	0.00	0.00	0.00	0.00	0.50	0.50	0.50	0.50
or51v1	2966	1003	3	1	10	0.98	1.00	0.99	0.99	0.00	0.75	0.00	0.00	0.38	0.75	0.50	0.50
or52a1	2939	971	5	1	13	1.00	0.99	0.99	0.98	0.75	0.67	0.67	0.00	0.75	0.67	0.67	0.33
or52a4	2915	936	1	1	11	0.89	0.89	0.89	0.89	0.00	0.00	0.00	0.00	0.67	0.67	0.67	0.67
or52a5	2951	952	5	1	4	0.85	0.85	0.85	0.85	0.00	0.00	0.00	0.00	0.50	0.50	0.50	0.50
or52b6	3008	972	3	1	6	0.95	1.00	0.95	0.98	0.00	1.00	0.00	0.00	0.50	1.00	0.50	0.50
or52d1	2957	956	4	1	4	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or52e2	2978	1001	5	1	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or52h1	2963	961	5	1	11	1.00	0.99	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.50	1.00	1.00
or52j3	2936	937	5	1	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
or52r1	3185	1030	4	1	3	0.85	0.87	0.85	0.85	0.00	0.00	0.00	0.00	0.50	0.38	0.50	0.50
osbp2	215019	2709	5	14	216	1.00	1.00	1.00	0.56	0.93	0.93	0.93	0.20	0.96	0.96	0.96	0.29
osm	6023	763	5	3	33	1.00	0.88	1.00	0.90	1.00	0.53	1.00	0.33	1.00	0.53	1.00	0.50
ostm1	35329	990	4	6	40	0.91	0.91	0.91	0.76	0.73	0.73	0.73	0.29	0.73	0.73	0.73	0.58
pan3	123056	2297	5	17	126	0.81	0.84	0.91	0.39	0.55	0.52	0.65	0.31	0.62	0.63	0.73	0.35
pdia2	6092	1546	3	11	37	0.99	0.99	0.99	0.69	0.91	0.91	0.91	0.45	0.95	0.95	0.95	0.55
pdk4	15117	1235	5	11	20	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.87	1.00	1.00	1.00	0.91
pdx1	8284	972	5	2	22	1.00	0.85	1.00	0.50	1.00	0.70	1.00	0.00	1.00	0.70	1.00	0.21
pes1	17283	1758	5	15	63	1.00	1.00	1.00	0.68	1.00	1.00	1.00	0.63	1.00	1.00	1.00	0.66
pex12	5843	1080	5	3	15	0.97	0.97	0.97	0.97	0.67	0.67	0.67	0.67	0.83	0.83	0.83	0.83
pgc	12670	1174	5	9	25	1.00	1.00	1.00	0.86	1.00	1.00	1.00	0.81	1.00	1.00	1.00	0.81
pla2g3	7677	1522	5	7	23	1.00	1.00	1.00	0.94	1.00	1.00	1.00	0.63	1.00	1.00	1.00	0.76
plcb3	17907	3682	5	31	67	1.00	0.99	1.00	0.87	0.97	0.94	0.97	0.73	0.98	0.97	0.98	0.81
pnma3	6062	1397	3	1	29	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
pnma5	5394	1584	2	1	24	0.81	0.82	1.00	1.00	0.62	0.62	1.00	1.00	0.62	0.62	1.00	1.00
polr3k	8626	341	5	3	13	1.00	0.88	1.00	0.88	1.00	0.67	1.00	0.67	1.00	0.83	1.00	0.83
pon1	28216	1068	5	9	50	1.00	1.00	1.00	0.58	1.00	1.00	1.00	0.22	1.00	1.00	1.00	0.39
pon3	38504	1065	5	9	42	0.96	0.96	0.96	0.40	0.84	0.84	0.84	0.24	0.90	0.90	0.90	0.41
ppp1r14b	4463	432	5	4	10	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ppp1r3a	44201	3331	5	4	39	1.00	1.00	1.00	0.94	1.00	1.00	1.00	0.21	1.00	1.00	1.00	0.31
ppp1r9a	390779	3706	5	15	323	0.91	0.91	0.91	0.52	0.89	0.89	0.89	0.06	0.89	0.89	0.89	0.13
prhoxnb	12532	525	3	2	37	0.99	0.99	1.00	0.68	0.83	0.83	1.00	0.42	0.83	0.83	1.00	0.42
prickle4	8611	1297	4	6	41	0.94	0.82	0.93	0.69	0.67	0.69	0.92	0.67	0.73	0.76	0.92	0.75
prss12	73506	2407	5	13	59	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.29	1.00	1.00	1.00	0.39
rasl10b	13862	612	5	3	11	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
rbm28	35527	2629	5	19	48	1.00	0.89	1.00	0.73	1.00	0.79	1.00	0.45	1.00	0.83	1.00	0.60
rnf152	80001	612	5	1	97	1.00	1.00	1.00	-0.01	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.00
rps5	9536	658	5	5	23	1.00	0.90	1.00	0.99	1.00	0.69	1.00	0.60	1.00	0.77	1.00	0.80
samd9	20492	4766	3	1	12	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
samd9l	20313	4834	5	1	25	1.00	0.97	1.00	1.00	1.00	0.62	1.00	1.00	1.00	0.62	1.00	1.00
sec14l3	14799	1203	5	12	29	1.00	1.00	1.00	0.75	1.00	1.00	1.00	0.44	1.00	1.00	1.00	0.57
sec14l4	18783	1197	4	12	31	1.00	1.00	1.00	0.81	1.00	1.00	1.00	0.50	1.00	1.00	1.00	0.67
selm	4789	441	3	5	21	1.00	1.00	1.00	0.98	1.00	1.00	1.00	0.80	1.00	1.00	1.00	0.90
shroom1	5991	2669	5	7	22	0.97	0.97	0.93	0.99	0.57	0.67	0.77	0.71	0.79	0.74	0.85	0.86
skap2	199655	1078	5	12	216	0.94	0.94	0.94	0.32	0.85	0.85	0.85	0.15	0.85	0.85	0.85	0.23
slc22a11	17902	1624	2	10	47	0.98	1.00	0.91	0.80	0.84	1.00	0.90	0.53	0.90	1.00	0.90	0.62
slc22a4	51755	1757	5	10	95	1.00	0.94	1.00	0.54	1.00	0.80	0.95	0.25	1.00	0.84	0.95	0.38
slc22a5	27906	1675	5	10	57	1.00	1.00	1.00	0.89	1.00	1.00	1.00	0.57	1.00	1.00	1.00	0.67
slc25a13	203928	2053	5	18	239	0.98	0.95	0.98	0.37	0.87	0.84	0.87	0.26	0.89	0.87	0.89	0.28
slc27a5	15733	2072	4	10	19	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.90	1.00	1.00	1.00	0.95
slc35e4	14070	1053	5	2	36	1.00	1.00	1.00	0.59	1.00	1.00	1.00	0.32	1.00	1.00	1.00	0.32
slc4a3	16411	3679	5	22	65	0.99	0.97	0.99	0.94	0.91	0.89	0.91	0.76	0.95	0.91	0.95	0.80
slfn12l	64955	1792	3	4	57	0.77	0.01	0.76	0.65	0.00	0.15	0.00	0.00	0.15	0.15	0.15	0.00
slfn13	15742	2718	3	4	39	1.00	0.99	1.00	0.92	0.83	0.79	1.00	0.42	0.83	0.79	1.00	0.62
slfn14	11967	2732	4	4	17	1.00	0.99	1.00	0.97	1.00	0.90	1.00	0.22	1.00	0.90	1.00	0.56
snd1	442458	2825	5	24	622	0.92	0.91	0.93	0.19	0.87	0.85	0.92	0.00	0.87	0.87	0.92	0.07
snrnp25	5841	512	5	5	15	1.00	0.74	1.00	1.00	1.00	0.69	1.00	1.00	1.00	0.77	1.00	1.00
snx19	42617	2929	5	11	80	0.96	0.96	0.96	0.66	0.86	0.86	0.86	0.11	0.91	0.91	0.91	0.27
spp2	28431	708	5	7	53	0.84	0.69	0.76	0.43	0.49	0.35	0.38	0.27	0.61	0.53	0.51	0.33
steap1	12453	1020	5	4	35	0.78	0.78	0.82	0.48	0.54	0.51	0.56	0.00	0.54	0.51	0.56	0.00
stip1	20434	1632	5	14	38	1.00	1.00	1.00	0.84	0.93	0.93	0.93	0.74	0.93	0.93	0.93	0.77
syt8	5077	1194	4	9	27	0.93	0.94	0.95	0.88	0.67	0.56	0.71	0.67	0.78	0.78	0.83	0.78
taf15	39751	1617	5	16	62	0.98	1.00	1.00	0.80	0.84	0.94	0.94	0.45	0.87	0.94	0.94	0.55
taf4b	167241	2010	5	15	172	0.95	0.92	0.97	0.52	0.84	0.76	0.88	0.08	0.84	0.79	0.88	0.16
tbc1d10a	36916	1547	5	9	83	1.00	1.00	1.00	0.78	1.00	1.00	1.00	0.47	1.00	1.00	1.00	0.56
tcn2	21887	1291	5	9	23	1.00	1.00	1.00	0.87	1.00	1.00	1.00	0.71	1.00	1.00	1.00	0.77
tec	136015	1844	5	17	113	0.98	0.91	0.98	0.39	0.86	0.74	0.86	0.21	0.92	0.77	0.92	0.28
tfeb	53083	1642	5	8	86	0.95	0.88	0.96	0.94	0.73	0.76	0.83	0.76	0.78	0.81	0.89	0.81
tfpi2	6321	700	5	5	13	1.00	1.00	1.00	0.99	1.00	1.00	1.00	0.80	1.00	1.00	1.00	0.90
tiam1	442555	4210	5	25	323	1.00	1.00	1.00	0.68	1.00	1.00	1.00	0.32	1.00	1.00	1.00	0.37
tmem8	13175	2326	5	13	56	0.89	0.85	0.95	0.66	0.75	0.79	0.86	0.57	0.75	0.79	0.86	0.61
tnni2	4006	549	5	7	18	0.99	0.97	0.99	0.69	0.93	0.77	0.93	0.51	0.93	0.77	0.93	0.60
tomm6	4454	225	3	2	31	0.55	0.55	0.55	0.46	0.42	0.42	0.42	0.00	0.42	0.42	0.42	0.25
trex2	3768	711	4	1	22	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.50
trim28	8247	2500	3	17	34	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.86	1.00	1.00	1.00	0.92
trpm8	104124	3316	5	24	130	0.99	0.98	0.98	0.65	0.94	0.85	0.92	0.21	0.94	0.87	0.92	0.40
txk	69864	1578	5	15	100	1.00	0.96	1.00	0.39	1.00	0.90	1.00	0.28	1.00	0.90	1.00	0.35
ube2m	5265	655	5	6	21	0.79	0.74	0.85	0.46	0.77	0.83	0.83	0.37	0.77	0.92	0.83	0.46
ubqln3	4624	1733	4	1	9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ubqlnl	4334	1836	2	1	5	1.00	1.00	1.00	1.00	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
uqcrq	4218	249	5	2	16	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
urb1	83983	6915	5	39	102	1.00	0.95	1.00	0.82	0.97	0.96	0.97	0.59	0.99	0.97	0.99	0.65
usp49	99717	2070	5	4	52	0.98	0.94	0.98	0.97	0.79	0.64	0.79	0.62	0.79	0.64	0.79	0.73
vegfb	5994	624	1	6	37	0.85	0.85	0.85	0.69	0.73	0.73	0.73	0.31	0.73	0.73	0.73	0.39
wnt2	48659	1083	5	5	83	0.94	0.94	0.94	0.73	0.80	0.80	0.80	0.55	0.80	0.80	0.80	0.55
zbtb45	8025	1440	4	2	26	1.00	0.93	1.00	0.92	1.00	0.75	1.00	0.42	1.00	0.75	1.00	0.83
zfp92	5306	1251	3	4	6	0.99	0.99	0.99	0.98	0.75	0.75	0.75	0.88	0.88	0.88	0.88	0.88
zmat5	38025	510	5	5	32	1.00	1.00	1.00	0.79	1.00	1.00	1.00	0.80	1.00	1.00	1.00	0.80
znf132	9409	2760	2	3	21	0.88	0.98	0.89	0.89	0.46	0.29	0.00	0.00	0.57	0.58	0.33	0.33
znf135	12165	1974	1	4	9	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
znf185	61026	2269	2	22	133	0.83	0.83	0.75	0.44	0.74	0.74	0.69	0.29	0.79	0.79	0.74	0.34

(continua na próxima página)

Tabela A.1 – Continuação

Gene	T_{DNA}	T_c	Q_c	Q_{ea}	Q_{eg}	CA				Av_e				Av_b			
						H1	H2	H3	H4	H1	H2	H3	H4	H1	H2	H3	H4
znf256	8877	1834	2	3	18	1.00	1.00	1.00	0.97	1.00	1.00	1.00	0.67	1.00	1.00	1.00	0.67
znf275	20772	1459	5	3	26	0.93	0.88	1.00	-0.04	0.53	0.46	0.67	0.00	0.67	0.57	0.83	0.11
znf324	8303	2066	5	3	25	0.93	0.84	0.99	0.89	0.46	0.44	0.58	0.25	0.57	0.56	0.73	0.38
znf324b	8229	2066	5	3	37	0.94	0.76	0.37	0.89	0.50	0.00	0.53	0.00	0.62	0.32	0.53	0.13
znf329	26454	1630	5	1	25	1.00	0.96	1.00	1.00	0.00	0.00	0.00	1.00	0.50	0.38	0.50	1.00
znf418	15489	2005	2	3	20	0.90	0.92	-0.07	0.92	0.24	0.27	0.00	0.00	0.60	0.53	0.00	0.38
znf446	6803	1572	1	6	38	0.81	0.81	0.81	0.69	0.62	0.62	0.62	0.15	0.77	0.77	0.77	0.46
znf497	10396	1854	1	1	28	0.89	0.89	0.89	0.89	0.00	0.00	0.00	0.62	0.31	0.31	0.31	0.62
znf551	9823	1954	2	3	20	0.95	0.96	0.45	0.95	0.33	0.29	0.42	0.33	0.67	0.58	0.62	0.67
znf584	11632	1860	2	4	22	0.96	0.77	0.93	0.84	0.68	0.42	0.75	0.62	0.68	0.52	0.75	0.73
znf606	28275	3650	5	6	59	0.99	0.74	0.99	0.89	0.77	0.20	0.77	0.18	0.77	0.40	0.77	0.46
znf622	16267	1357	5	6	21	1.00	1.00	1.00	0.91	1.00	1.00	1.00	0.62	1.00	1.00	1.00	0.70
znf8	18937	1723	2	4	36	1.00	1.00	1.00	0.92	1.00	1.00	1.00	0.25	1.00	1.00	1.00	0.38
znf814	21696	2791	2	3	30	0.94	0.89	-0.06	0.97	0.00	0.22	0.00	0.29	0.21	0.33	0.00	0.44
zscan1	22566	537	1	4	96	0.44	0.44	0.44	-0.04	0.00	0.00	0.00	0.00	0.12	0.12	0.12	0.00
zscan22	17328	1491	1	2	27	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
zscan4	12218	1360	3	3	6	1.00	1.00	1.00	1.00	0.88	0.88	1.00	1.00	0.88	0.88	1.00	1.00

Tabela A.1: Algumas informações sobre as sequências do conjunto de instâncias baseadas no genoma humano.

Referências Bibliográficas

[1] Adi, S. S., and Ferreira, C. E. An experimental evaluation of similarity-based gene prediction tools. In *2nd International Conference on Bioinformatics and Computational Biology* (Angra dos Reis, 2004), Proceedings of the 2nd ICO-BICOBI.

[2] Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. *Molecular Biology of The Cell*, 4th ed. Garland Science, 2002.

[3] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215* (1990), 403–410.

[4] Bagirov, A. M., Ferguson, B., Ivkovic, S., Saunders, G., and Yearwood, J. New algorithms for multi-class cancer diagnosis using tumor gene expression signatures. *Bioinformatics 19*, 14 (2003), 1800–1807.

[5] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics 41*, 1 (1970), 164–171.

[6] Bent, A. F. Plant disease resistance genes: Function meets structure. *The Plant Cell 8* (1996), 1757–1771.

[7] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35*, 3 (2003), 268–308.

[8] Brendel, V., and Kleffe, J. Prediction of locally optimal splice sites in plant pre-mRNA with applications to gene identification in *Arabidopsis thaliana* genomic DNA. *Nucleic Acids Research 26*, 20 (1998), 4748–4757.

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

[10] Burset, M., and Guigó, R. Evaluation of gene structure prediction programs. *Genomics 34*, 298 (1996), 353–367.

[11] Castelo, R., and Guigó, R. Splice site identification by idlBNs. *Bioinformatics 20* (2004), 169–176.

[12] Chao, K.-M., Zhang, J., Ostell, J., and Miller, W. A tool for aligning very similar DNA sequences. *CABIOS 13*, 1 (1997), 75–80.

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

[14] Creighton, T. E. *Encyclopedia of Molecular Biology*, 1st ed. Wiley-Interscience, 1999.

[15] de Brito, R. T. Alinhamento de seqüências biológicas. Master's thesis, Universidade de São Paulo, 2003.

[16] Fickett, J. W. Recognition of protein coding regions in DNA sequences. *Nucleic Acids Research 10*, 17 (1982), 5303–5318.

[17] Florea, L., Hartzell, G., Zhang, Z., Rubin, G. M., and Miller, W. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Research 8* (1998), 967–974.

[18] Gelfand, M. S. Prediction of function in DNA sequence analysis. *Journal of Computational Biology 2*, 1 (1996), 87–117.

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

[20] Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science 286*, 5439 (1999), 531–537.

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

[22] Hirschberg, D. S. A linear space algorithm for computing max subsequences. *Communications of the ACM 18* (1975), 341–343.

[23] International Human Genome Sequencing Consortium (Lander, E. S. *et al.*). Initial sequencing and analysis of the human genome. *Nature 409*, 6822 (2001), 860–921.

[24] Korf, I., Flicek, P., Duan, D., and Brent, M. R. Integrating genomic homology into gene structure prediction. *Bioinformatics 17*, 1 (2001), S140–S148.

[25] Krogh, A. *An Introduction to Hidden Markov Models for Biological Sequences.* Elsevier, Center for Biological Sequence Analysis Technical University of Denmark Building 206, 2800 Lyngby, Denmark, 1998, ch. 4.

[26] Lewin, B. *Genes VII*, 1st ed. Oxford, 2001.

[27] Lewis, H. R., and Papadimitriou, C. H. *Elements of the theory of computation*, 1st ed. Prentice-Hall, 1981.

[28] Mott, R. EST_genome: a program to align spliced DNA sequences to unspliced genomic DNA. *CABIOS 13*, 4 (1997), 477–478.

[29] Mouchiroud, D., D'Onofrio, G., Aïssani, B., Macaya, G., Gautier, C., and Bernardi, G. The distribution of genes in the human genome. *Gene*, 100 (1991), 181–187.

[30] Myers, E. W., and Miller, W. Optimal alignments in linear space. *Computer Applications in the Biosciences 4*, 1 (1988), 11–17.

[31] 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.

[32] 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* (2004), 390–415.

[33] Novichkov, P. S., Gelfand, M. S., and Mironov, A. A. Gene recognition in eukaryotic DNA by comparison of genomic sequences. *Bioinformatics 17*, 11 (2001), 1011–1018.

[34] Papadimitriou, C. M. *Computational Complexity*, 1st ed. Addison-Wesley, 1994.

[35] Pearl, J. *Heuristics: intelligent search strategies for computer problem solving*, 1st ed. Addison-Wesley, 1984.

[36] Pevzner, P. A. *Computational Molecular Biology: An Algorithmic Approach*, 1st ed. MIT Press, 2001.

[37] Porcel, B. M., Tran, A.-N., Tammi, M., Nyarady, Z., Rydåker, M., Urmenyi, T. P., Rondinelli, E., Pettersson, U., Andersson, B., and Åslund, L. Gene survey of the pathogenic protozoan trypanosoma cruzi. *Genome Research 10* (2000), 1103–1107.

[38] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. 267–296.

[39] Ramaswamy, S., Tamayo, P., Rifkin, R., Mukherjee, S., Yeang, C.-H., Angelo, M., Ladd, C., Reich, M., Latulippe, E., Mesirov, J. P., Poggio, T., Gerald, W., Loda, M., Lander, E. S., and Golub, T. R. Multiclass cancer diagnosis using tumor gene expression signatures. *In Proceedings of the National Academy of Sciences 98*, 26 (2001), 15149–15154.

[40] Schwartz, S., Miller, W., Yang, C.-M., and C.Hardison, R. Software tools for analyzing pairwise alignments of long sequences. *Nucleic Acids Research, 19*, 17 (1991), 4663–4667.

[41] Setubal, J., and Meidanis, J. *Introduction to Computational Molecular Biology*, 1st ed. PWS Publishing Company, 1997.

[42] Shepherd, J. C. W. Method to determine the reading frame of a protein from the purine/pyrimidine genome sequence and its possible evolutionary justification. *In Proceedings of the National Academy of Sciences 78*, 3 (1981), 1596–1600.

[43] Sim, J. S., and Park, K. The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms 1*, 1 (2003), 111–117.

[44] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology 147* (1981), 195–197.

[45] Staden, R. Computer methods to locate signals in nucleic acid sequences. *Nucleic Acids Research 12* (1984), 505–519.

[46] Staden, R., and McLachlan, A. Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Research 10*, 1 (1982), 141–156.

[47] Stanke, M., Schöffmann, O., Morgenstern, B., and Waack, S. Gene prediction in eukaryotes with a generalized Hidden Markov Model that uses hints from external sources. *BMC Bioinformatics 7* (2006), 62.

[48] Stanke, M., and Waack, S. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics 19*, 2 (2003), 215–225.

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

[50] Usuka, J., Zhu, W., and Brendel, V. Optimal spliced alignment of homologous cDNA to a genomic DNA template. *Bioinformatics 16*, 3 (2000), 203–211.

[51] Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., Gocayne, J. D., Amanatides, P., Ballew, R. M., Huson, D. H., Wortman, J. R., Zhang, Q., Kodira, C. D., Zheng, X. H., Chen, L., Skupski, M., Subramanian, G., Thomas, P. D., Zhang, J., Miklos, G. G. L., Nelson, C., Broder, S., Clark, A. G., Nadeau, J., Mckusick, V. A., Zinder, N., Levine, A. J., Roberts, R. J., Simon, M., Slayman, C., Hunkapiller, M., Bolanos, R., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., Florea, L., Halpern, A., Hannenhalli, S., Kravitz, S., Levy, S., Mobarry, C., Reinert, K., Remington, K., Abu-Threideh, J., Beasley, E., Biddick, K., Bonazzi, V., Brandon, R., Cargill, M., Chandramouliswaran, I., Charlab, R., Chaturvedi, K., Deng, Z., Francesco, V. D., Dunn, P., Eilbeck, K., Evangelista, C., Gabrielian, A. E., Gan, W., Ge, W., Gong, F., Gu, Z., Guan, P., Heiman, T. J., Higgins, M. E., Ji, R. R., Ke, Z., Ketchum, K. A., Lai, Z., Lei, Y., Li, Z., Li, J., Liang, Y., Lin, X., Lu, F., Merkulov, G. V., Milshina, N., Moore, H. M., Naik, A. K., Narayan, V. A., Neelam, B., Nusskern, D., Rusch, D. B., Salzberg, S., Shao, W., Shue, B., Sun, J., Wang, Z., Wang, A., Wang, X., Wang, J., Wei, M., Wides, R., Xiao, C., Yan, C., Yao, A., Ye, J., Zhan, M., Zhang, W., Zhang, H., Zhao, Q., Zheng, L., Zhong, F., Zhong, W., Zhu, S., Zhao, S., Gilbert, D., Baumhueter, S., Spier, G., Carter, C., Cravchik, A., Woodage, T., Ali, F., An, H., Awe, A., Baldwin, D., Baden, H., Barnstead, M., Barrow, I., Beeson, K., Busam, D., Carver, A., Center, A., Cheng, M. L., Curry, L., Danaher, S., Davenport, L.,

Desilets, R., Dietz, S., Dodson, K., Doup, L., Ferriera, S., Garg, N., Gluecksmann, A., Hart, B., Haynes, J., Haynes, C., Heiner, C., Hladun, S., Hostin, D., Houck, J., Howland, T., Ibegwam, C., Johnson, J., Kalush, F., Kline, L., Koduru, S., Love, A., Mann, F., May, D., Mccawley, S., Mcintosh, T., Mcmullen, I., Moy, M., Moy, L., Murphy, B., Nelson, K., Pfannkoch, C., Pratts, E., Puri, V., Qureshi, H., Reardon, M., Rodriguez, R., Rogers, Y. H., Romblad, D., Ruhfel, B., Scott, R., Sitter, C., Smallwood, M., Stewart, E., Strong, R., Suh, E., Thomas, R., Tint, N. N., Tse, S., Vech, C., Wang, G., Wetter, J., Williams, S., Williams, M., Windsor, S., Winn-Deen, E., Wolfe, K., Zaveri, J., Zaveri, K., Abril, J. F., Guigó, R., Campbell, M. J., Sjolander, K. V., Karlak, B., Kejariwal, A., Mi, H., Lazareva, B., Hatton, T., Narechania, A., Diemer, K., Muruganujan, A., Guo, N., Sato, S., Bafna, V., Istrail, S., Lippert, R., Schwartz, R., Walenz, B., Yooseph, S., Allen, D., Basu, A., Baxendale, J., Blick, L., Caminha, M., Carnes-Stine, J., Caulk, P., Chiang, Y. H., Coyne, M., Dahlke, C., Mays, A., Dombroski, M., Donnelly, M., Ely, D., Esparham, S., Fosler, C., Gire, H., Glanowski, S., Glasser, K., Glodek, A., Gorokhov, M., Graham, K., Gropman, B., Harris, M., Heil, J., Henderson, S., Hoover, J., Jennings, D., Jordan, C., Jordan, J., Kasha, J., Kagan, L., Kraft, C., Levitsky, A., Lewis, M., Liu, X., Lopez, J., Ma, D., Majoros, W., Mcdaniel, J., Murphy, S., Newman, M., Nguyen, T., Nguyen, N., Nodell, M., Pan, S., Peck, J., Peterson, M., Rowe, W., Sanders, R., Scott, J., Simpson, M., Smith, T., Sprague, A., Stockwell, T., Turner, R., Venter, E., Wang, M., Wen, M., Wu, D., Wu, M., Xia, A., Zandieh, A., and Zhu, X. The sequence of the human genome. *Science 291*, 5507 (2001), 1304–1351.

[52] Viterbi, A. G. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions Information Theory*, 13 (1967), 260–269.

[53] Wang, L., and Jiang, T. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1 (1994), 337–348.

[54] W.Fickett, J., and Tung, C.-S. Assessment of protein coding measures. *Nucleic Acids Research, 20*, 24 (1992), 6441–6450.

[55] Wheelan, S. J., Church, D. M., and Ostell, J. M. Spidey: a tool for mRNA-to-genomic alignments. *Genome Research 11* (2001), 1952–1957.

[56] W.Sayers, E., Barrett, T., Benson, D. A., Bolton, E., Bryant, S. H., Canese, K., Chetvernin, V., Church, D. M., Cuccio, M. D., Federhen, S., Feolo, M., Geer, L. Y., Helmberg, W., Kapustin, Y., Landsman, D., Lipman, D. J., Lu, Z., Madden, T. L., Madej, T., Maglott, D. R., Marchler-Bauer, A., Miller, V., Mizrachi, I., Ostell, J., Panchenko, A., Pruitt, K. D., Schuler, G. D., Sequeira, E., Sherry, S. T., Shumway, M., Sirotkin, K., Slotta, D., Souvorov, A., Starchenko, G., Tatusova, T. A., Wagner, L., Wang, Y., Wilbur, W. J., Yaschenko,

E., AND YE, J. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research 38* (2010), D5–D16.

[57] YEO, G., AND BURGE, C. B. Maximum entropy modeling of short sequence motifs with applications to RNA splicing signals. In *Proceedings of the seventh annual international conference on Computational molecular biology* (2003), pp. 322–331.

[58] YU, Y. G., BUSS, G. R., AND MAROOF, M. A. S. Isolation of a superfamily of candidate disease-resistance genes in soybean based on a conserved nucleotide-binding site. *In Proceedings of the National Academy of Sciences 93* (1996), 11751–11756.

[59] ZHANG, M. Q., AND MARR, T. G. A weight array model for splicing signal analysis. *Computer Applications in the Biosciences*, 9 (1993), 499–509.