O Problema do Particionamento de Similaridade Máxima

Alex Zanella Zaccaron

Dissertação de Mestrado
à
Faculdade de Computação
da
Universidade Federal de Mato Grosso do Sul
para
obtenção do título
de
Mestre em Ciência da Computação

Orientador: Prof. Dr. Said Sadique Adi

Área de Concentração: Ciência da Computação

Este trabalho foi produzido com apoio financeiro da CAPES

Campo Grande, outubro de 2015

O Problema do Particionamento de Similaridade Máxima

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 29/09/2015.

Comissão Julgadora:

- Prof. Dr. Said Sadique Adi (orientador) - UFMS
- Prof. Dr. Carlos Henrique Aguena Higa - UFMS
- Prof. Dr. Cleber Valgas Gomes Mira - UEMS

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais, Oscar e Nilsa, pelos valores morais que me ensinaram e pelo apoio que me concederam durante toda a minha vida acadêmica.

Agradeço muito ao meu orientador, professor Said Sadique Adi, pela forma generosa e atenciosa com a qual me orientou. Sou grato por tudo que me ensinou e por estar sempre disposto a esclarecer minhas dúvidas, que não eram poucas. Suas ideias e dicas foram fundamentais para o desenvolvimento e conclusão deste trabalho.

Agradeço também ao Dr. Burt Bluhm, professor assistente da Universidade do Arkansas, por ter me concedido a incrível experiência de realizar um estágio em seu laboratório e pelas várias e divertidas conversas que tivemos sobre Ciência. Muito do que aprendi apliquei neste trabalho.

Por fim, agradeço à CAPES pelo apoio financeiro e a todos que contribuíram direta ou indiretamente para a realização deste trabalho.

Resumo

Neste trabalho, nós propomos um novo problema de otimização combinatória envolvendo sequências de caracteres chamado Problema do Particionamento de Similaridade Máxima. Apresentamos também uma prova de que esse problema é NP-difícil no sentido forte, o que significa que não existe um algoritmo polinomial e nem pseudo-polinomial que o resolve, a menos que P = NP. Além disso, desenvolvemos e testamos duas heurísticas baseadas na estratégia gulosa que encontram soluções para o Problema do Particionamento de Similaridade Máxima em tempo polinomial. Este trabalho também traz uma aplicação do problema em questão através da modelagem de um problema importante da Biologia Computacional chamado problema da reconstrução e quantificação de transcriptoma, modelagem essa pioneira na abordagem desse problema como um problema de otimização combinatória envolvendo sequências.
Palavras-Chave: Partição, Otimização Combinatória, Similaridade, Transcriptoma, Reconstrução e Quantificação.

Abstract

In this work, we propose a new combinatorial optimization problem involving strings called Maximum Similarity Partitioning Problem. We also present a proof that this problem is NP-hard in the strong sense, which means that there is no polynomial nor pseudo-polynomial time algorithm that can solve it, unless P = NP. Besides, we developed and tested two greedy heuristics that find solutions for the Maximum Similarity Partitioning Problem in polynomial time. This work brings as well an application for the corresponding problem in the modeling of an important problem in Computational Biology known as transcriptome reconstruction and quantification problem. We consider such modeling the first to deal with the transcriptome reconstruction and quantification problem as a combinatorial optimization problem involving strings.
Keywords: Partition, Combinatorial Optimization, Similarity, Transcriptome, Reconstruction and Quantification.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

A Ciência da Computação é uma grande área do conhecimento que aborda uma ampla variedade de problemas computacionais. Uma das subáreas da Ciência da Computação, a Teoria da Computação, lida com tais problemas através do desenvolvimento e análise de algoritmos que podem resolvê-los. Entretanto, alguns problemas são classificados como difíceis de se resolver, de modo que qualquer algoritmo que encontra uma solução ótima para algum deles consome muito tempo de execução e/ou muito espaço de armazenamento. Essa classificação de problemas computacionais é um dos objetivos da Teoria da Complexidade Computacional, um ramo da Teoria da Computação.

Alguns problemas computacionais recebem atenção especial pela possibilidade de serem aplicados em diferentes áreas do conhecimento, como a Biologia, a Física e a Química. Nesse sentido, destacam-se os problemas de otimização combinatória que envolvem sequências de caracteres, por serem aplicados em áreas como a Biologia Computacional, onde o objetivo é modelar e tentar solucionar problemas biológicos, geralmente relacionados com o genoma de um organismo.

Um dos principais objetos de estudo da Biologia Computacional é o conjunto de informações genéticas de um organismo presente no interior de suas células, conjunto esse codificado em moléculas de DNA. Nesse contexto, trechos específicos do DNA, chamados genes, são expressos produzindo uma grande variedade de moléculas de RNA essenciais para o funcionamento adequado do organismo. Esse conjunto de moléculas de RNA presentes na célula em um dado momento é conhecido como transcriptoma do organismo.

Conhecer o transcriptoma de um organismo é de fundamental importância no diagnóstico e tratamento de doenças relacionadas à hiper ou hipoexpressão de um ou mais genes específicos. Outra importância do estudo de um transcriptoma está na possibilidade de melhoramento genético de plantas e animais através da identificação de genes diferencialmente expressos sob determinadas condições naturais.

Uma das formas de estudar o transcriptoma de um organismo é extraindo os RNAs em suas células em um dado momento e analisando essas moléculas. Isso pode ser feito por meio de um protocolo chamado RNA-Seq. Infelizmente, esse protocolo não permite extrair o transcriptoma exatamente como ele se apresenta na célula. No processo que constitui o RNA-Seq, as moléculas de RNA que compõem o transcriptoma do organismo são quebradas em pedaços, produzindo uma espécie de quebra-cabeça genético. Daí surgem os problemas de montar o transcriptoma de forma que seja possível identificar os distintos RNAs que o constituem e determinar a quantidade de cada um deles. Esses problemas são comumente conhecidos como o problema da reconstrução e quantificação de transcriptoma, respectivamente.

Diversas ferramentas baseadas em diferentes abordagens já foram desenvolvidas para se reconstruir e quantificar o transcriptoma. No entanto, nem sempre essas ferramentas apresentam bons resultados. Esse fato expõe a grande dificuldade de se resolver eficientemente esse problema. Por isso, ainda é significativo a elaboração de novas abordagens para ele, como a modelagem através de um problema de otimização combinatória envolvendo sequências, que, até onde sabemos, nunca foi utilizada.

Dado o exposto acima, o objetivo principal deste trabalho consiste na proposta e estudo detalhado de um novo problema de otimização combinatória envolvendo sequências chamado Problema do Particionamento de Similaridade Máxima, e sua aplicação no problema da reconstrução e quantificação de transcriptoma.

1.1 Contribuições

Dentre as contribuições deste trabalho, listamos as principais como sendo:

- a proposta de um novo problema de otimização combinatória chamado Problema do Particionamento de Similaridade Máxima (PPSM) que modela o problema da reconstrução e quantificação de transcriptoma;
- uma prova de que o PPSM é NP-difícil no sentido forte;
- o desenvolvimento de duas heurísticas gulosas que encontram soluções para o PPSM em tempo polinomial, disponíveis em https://github.com/alexzaccaron/heuristicasppsm;
- a geração de um conjunto de testes artificiais para o PPSM, além de testes semi-artificiais e reais para o problema da reconstrução e quantificação de transcriptoma, disponível em https://github.com/alexzaccaron/testesppsm;
- a publicação de um artigo na *15th International Conference on Computational Science and Its Applications* (ICCSA 2015), intitulado *The Maximum Similarity Partitioning Problem and Its Application in the Transcriptome Reconstruction and Quantification Problem* (DOI: 10.1007/978-3-319-21404-7_19).

1.2 Organização do Texto

Além desta introdução, este trabalho está dividido em outros cinco capítulos. O Capítulo 2 apresenta conceitos de Ciência da Computação e Biologia Molecular necessários para a compreensão do problema abordado. No Capítulo 3 o Problema do Particionamento de Similaridade Máxima é formalmente definido e são apresentados detalhes de como ele modela o problema da reconstrução e quantificação de transcriptoma, seguidos pela prova de que ele é NP-difícil no sentido forte. As duas heurísticas para o Problema do Particionamento de Similaridade Máxima são apresentadas no Capítulo 4, juntamente com a análise da complexidade de tempo delas. O Capítulo 5 traz a metodologia dos testes aplicados e a análise dos resultados obtidos pelas heurísticas quanto ao Problema do Particionamento de Similaridade Máxima e ao problema da reconstrução e quantificação de transcriptoma. Por fim, o trabalho é concluído no Capítulo 6 junto com algumas sugestões que podem ser exploradas futuramente.

Capítulo 2

Fundamentação Teórica

Para uma compreensão adequada do problema abordado neste trabalho, é necessário o conhecimento de alguns conceitos de Computação e Biologia Molecular. Os conceitos detalhados neste capítulo baseiam-se em diversos trabalhos da literatura [1–13].

2.1 Conceitos de Computação

Um dos principais focos da Ciência da Computação é a formulação e análise de *problemas computacionais* que podem ter aplicação tanto na própria Ciência da Computação quanto em outras áreas de conhecimento como a Biologia, a Física e a Química. Dessa forma, a Ciência da Computação está presente em diversos campos de pesquisa, atuando no avanço científico deles. Por isso, a compreensão de conceitos de computação, sobretudo da Teoria da Computação, se faz necessária.

2.1.1 Problemas Computacionais

A Teoria da Computação tem como principal objetivo a análise e resolução de problemas computacionais. Esses problemas podem ser classificados basicamente em problemas de *busca*, de *contagem*, de *otimização* e de *decisão*. Neste trabalho, estamos interessados nos problemas de otimização e decisão, os quais serão definidos nesta seção.

Todo problema computacional possui uma ou mais *instâncias* que consistem na entrada necessária para se calcular uma solução para o problema. Nesse contexto, definimos um **algoritmo** como uma sequência de passos computacionais que toma um dado ou um conjunto de dados como *entrada* e o transforma em um novo dado ou um novo conjunto de dados de *saída*. Geralmente, os algoritmos são utilizados como uma ferramenta para resolver problemas computacionais, onde a entrada corresponde a uma instância do problema a ser resolvido e a saída corresponde a uma solução de tal instância [3]. Dizemos que um algoritmo é **correto** se, para qualquer instância de um problema computacional dada como entrada, ele gera uma saída correta. Dessa forma, dizemos que um algoritmo correto **resolve** um problema computacional [3].

Os **problemas de otimização** são problemas para os quais existe um conjunto de soluções *viáveis*, tal que cada solução possui um valor associado α calculado por uma *função objetivo*, e desejamos encontrar uma solução com o "melhor" α associado, chamada de **solução ótima**. Dada uma instância de um problema de otimização P que possui um conjunto de soluções viáveis S, P

é dito um **problema de minimização** se desejamos encontrar uma solução $s_i \in S$ tal que o valor α_i associado é o menor possível, ou seja, $\nexists s_k \in S$ tal que $\alpha_k < \alpha_i$. De maneira análoga, P é dito um **problema de maximização** se desejamos encontrar uma solução s_i cujo valor associado α_i é o maior possível, ou seja, $\nexists s_k \in S$ tal que $\alpha_k > \alpha_i$ [3]. Alguns problemas de otimização possuem conjunto infinito de soluções, enquanto outros não. Chamamos de **problema de otimização combinatória** um problema de otimização cujo conjunto de soluções é finito [14, 15].

Os **problemas de decisão** são problemas cuja resposta é simplesmente "sim" ou "não". Geralmente, problemas de decisão representam perguntas do tipo: "para um dado valor k, existe uma solução s_i cujo valor associado α_i é maior (ou menor) que k?". Perceba que a resposta para essa pergunta é "sim" ou "não". Denotamos por SIM_P e NAO_P os conjuntos de instâncias de um problema de decisão P cuja respostas são "sim" e "não", respectivamente. Podemos expressar um problema de otimização como um problema de decisão, estabelecendo um limite k [4]. Para exemplificar tal afirmação, considere o seguinte problema de otimização:

Problema do Caixeiro Viajante (PCV): *dadas n cidades $c_1, c_2, ..., c_n$ e a distância $d(c_i, c_j)$ entre cada par de cidades c_i e c_j, encontre um caminho que passe pelas n cidades cuja distância total seja mínima.*

Repare que o PCV é um problema de otimização combinatória de minimização, pois existe um número finito de caminhos possíveis entre as n cidades e, dentre todos esses caminhos, deseja-se o de menor distância. A partir da versão de otimização do PCV, é possível definir sua versão de decisão, apresentada a seguir.

Problema do Caixeiro Viajante - versão de decisão ($\mathbf{PCV_D}$): *dadas n cidades $c_1, c_2, ..., c_n$, a distância $d(c_i, c_j)$ entre cada par de cidades c_i e c_j, e um limite k, existe um caminho que passe pelas n cidades cuja distância total é menor ou igual a k?*

Observe que o PCV_D aparenta ser mais fácil que sua versão de otimização, pois uma solução para ele consiste em apenas verificar se existe tal caminho. Por isso, considerando um problema de otimização P e seu respectivo problema de decisão P_D, dizemos que P_D *não é mais difícil* que P, ou seja, P é *tão difícil quanto* P_D [4]. Tal argumento pode ser verificado considerando que, se resolvemos P, então podemos comparar o valor α associado à solução ótima encontrada com o limitante k de P_D. Se P é um problema de minimização e $\alpha \leq k$, então P_D terá como resposta "sim". Caso $\alpha > k$, P_D terá como resposta "não". De maneira semelhante, se P é um problema de maximização e $\alpha \geq k$, então P_D terá como resposta "sim". Caso $\alpha < k$, P_D terá como resposta "não".

Toda instância de um problema computacional pode ser mapeada em uma única cadeia de símbolos construída sobre um alfabeto finito. Esse mapeamento chama-se **codificação**. Geralmente, as instâncias são codificadas em cadeias binárias, formadas por 0s e 1s. Dessa forma, uma instância i de um problema computacional P é *codificada* em uma cadeia binária $e(i)$ por meio de uma codificação $e : i \to \{0,1\}^*$. Agora, considere um algoritmo A que resolve P. A cadeia $e(i)$ é dada como entrada para A, que, por sua vez, produz uma saída correta que corresponde a uma solução de P para a instância i. Ou seja, A não toma como entrada a instância i em si, mas a codificação $e(i)$ de i. Assim, definimos o **tamanho da entrada** correspondente à instância i dada para A como a quantidade de símbolos em $e(i)$ [3, 6]. Dado um problema de decisão Q e o conjunto de instâncias I_Q de Q, representamos o tamanho da entrada pela função **Length**: $I_Q \to \mathbb{Z}^+$ que determina o tamanho de uma instância $i \in I_Q$. Por exemplo, seja $i = \{a_1, a_2, ..., a_n\}$ um conjunto de n números

naturais codificados no sistema de numeração decimal, então Length$(i) \approx \sum_{k=1}^{n}(\lfloor \log_2 a_k \rfloor + 1)$, isto é, o valor de Length(i) é, aproximadamente, igual ao número de 0s e 1s necessários para codificar os n números no sistema de codificação binário. Já a função **Max**: $I_Q \rightarrow \mathbb{Z}^+$ determina a magnitude do maior número em uma instância $i \in I_Q$ [5]. Para o exemplo anterior, Max$(i) = max\{a_k : a_k \in i\}$.

Veremos na próxima seção que o tamanho da entrada dada para um algoritmo é utilizado para realizar a análise da sua complexidade. Para esse objetivo, geralmente é considerada uma definição de tamanho da entrada um pouco diferente da apresentada anteriormente. Informalmente, o tamanho da entrada de um algoritmo consiste no número de *itens* na entrada [3]. O termo *itens* é um tanto vago e, por isso, o consideramos como as unidades elementares que compõem uma instância de um problema, do ponto de vista do problema. Tais unidades elementares podem ser letras, palavras, vértices ou arestas. Por exemplo, o tamanho da entrada do PCV_D consiste no número de cidades, as distâncias entre cada par de cidades e o limite k.

2.1.2 Complexidade Computacional

Analisar a complexidade de um algoritmo significa prever os recursos (memória, largura de banda de comunicação, tempo de execução, etc) que ele necessita para resolver um certo problema. Frequentemente, nos preocupamos com o tempo de execução, ou *complexidade de tempo*, do algoritmo, que é dado em função do tamanho de sua entrada [3].

O tempo de execução de um algoritmo é medido pelo número de passos básicos que ele executa durante o processamento da entrada. Como entradas distintas quase sempre significam tempos de execução distintos, é muito comum a análise do tempo de execução de um algoritmo no *pior caso*. Nessa análise, dentre todas as entradas de tamanho n, considera-se aquela (ou aquelas) sobre a qual o algoritmo leva o maior tempo para processar e gerar uma saída [16]. Uma maneira de descrever a complexidade de tempo de um algoritmo é através da *notação assintótica*, sendo as principais a notação O, a notação Ω e a notação Θ, cujas definições apresentam-se a seguir.

Dada uma função $f(n)$, dizemos que $f(n) = O(g(n))$ se existem constantes positivas c e n_0 tais que $0 \leq f(n) \leq cg(n)$, para todo $n \geq n_0$. Ou seja, $g(n)$ é um *limite assintótico superior* de $f(n)$. Dizemos também que $f(n) = \Omega(g(n))$ se existem constantes positivas c e n_0 tais que $0 \leq cg(n) \leq f(n)$, para todo $n \geq n_0$. Ou seja, $g(n)$ é um *limite assintótico inferior* de $f(n)$. Por fim, dizemos que $f(n) = \Theta(g(n))$ se existem constantes positivas c_1, c_2 e n_0 tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, para todo $n \geq n_0$. Ou seja, $g(n)$ é um *limite assintótico restrito* de $f(n)$. Dado um algoritmo que executa $f(n)$ passos para processar uma entrada de tamanho n e gerar uma saída, dizemos que ele é $O(g(n))$ se $f(n) = O(g(n))$. Se $f(n) = \Omega(g(n))$ dizemos que o algoritmo é $\Omega(g(n))$. Caso $f(n) = \Theta(g(n))$, então dizemos que ele é $\Theta(g(n))$ [3].

Um algoritmo é dito **polinomial** se a sua complexidade de tempo de execução, no pior caso, for limitada superiormente por uma função polinomial sobre o tamanho da entrada. De maneira semelhante, dizemos que um algoritmo é **exponencial** se a sua complexidade de tempo de execução, no pior caso, for limitada superiormente por uma função exponencial sobre o tamanho da entrada. Considere, por exemplo, um algoritmo que tome uma entrada de tamanho n. Ele é um algoritmo polinomial se ele for $O(n^k)$, para alguma constante k. Caso ele seja $O(k^n)$, para alguma constante k, então ele é um algoritmo exponencial. Observe que, para uma entrada suficientemente grande, um algoritmo polinomial executará uma quantidade menor de passos em comparação a um algoritmo exponencial, sendo assim mais rápido. Além disso, todo algoritmo polinomial também é exponen-

cial, pois a mesma função que impõe um limite superior assintótico sobre o algoritmo exponencial também o faz para o algoritmo polinomial [3]. Além dos algoritmos polinomiais e exponenciais, existem também os algoritmos pseudo-polinomiais. Um algoritmo é chamado de **pseudo-polinomial** quando seu tempo de execução é limitado superiormente por um polinômio nas variáveis Length(i) e Max(i), relacionados à instância de entrada i [6].

Normalmente, dizemos que um problema computacional pode ser resolvido em tempo polinomial se existe um algoritmo polinomial que o resolve. De maneira análoga, dizemos que ele pode ser resolvido em tempo pseudo-polinomial ou exponencial se existe um algoritmo pseudo-polinomial ou exponencial que o resolve, respectivamente. Nem todo problema que é resolvido em tempo exponencial também é resolvido em tempo polinomial ou pseudo-polinomial. Tais problemas são considerados difíceis e, como veremos a seguir, pertencem a uma classe específica de problemas.

2.1.3 Classes de Problemas Computacionais

Os problemas computacionais podem ser classificados de acordo com os seus níveis de dificuldade. Esse é um dos principais objetivos de um dos ramos da Teoria da Computação, chamado de *Teoria da Complexidade Computacional*. Enquanto há problemas simples que podem ser resolvidos por algoritmos polinomiais, existem outros mais complexos para os quais não se conhecem algoritmos polinomiais que os resolvem. Nesta seção, apresentaremos as principais classes de problemas computacionais, que são a classe P, a classe NP e a classe NPC.

A Teoria da Complexidade Computacional se aplica diretamente aos problema de decisão. Como discutido na Seção 2.1.1, todo problema de otimização Q possui uma versão de decisão Q_{D} não mais difícil que a sua versão de otimização [4]. Dessa forma, se não existe um algoritmo polinomial que resolve Q_{D}, então Q também não pode ser resolvido por um algoritmo polinomial. Em outras palavras, os resultados obtidos para os problemas de decisão se aplicam aos problemas de otimização [6].

A classe **P** consiste dos problemas de decisão que podem ser resolvidos por um algoritmo polinomial, ou seja, ela inclui os problemas para os quais existem algoritmos que os resolvem em tempo $O(n^k)$ para alguma constante k, onde n é o tamanho da entrada [3]. Essa classe inclui problemas considerados simples de se resolver. Dado um grafo G, dois vértices u e v de G e um valor k, verificar se existe um caminho de u até v de tamanho menor ou igual a k é um exemplo de problema da classe P.

A classe **NP** inclui os problemas de decisão cujas soluções podem ser verificadas por um algoritmo polinomial. Mais especificamente, um problema de decisão Q está em NP se, dada uma solução s_q de uma instância $i \in SIM_Q$ de Q, podemos verificar se s_q é uma solução válida para i por meio de um algoritmo polinomial [3, 6]. O $\mathrm{PCV_D}$ é um exemplo de um problema que está em NP, pois, dado um caminho p que passa pelas n cidades, pode-se calcular a distância total de p somando-se as distâncias entre cada par de cidades dadas como solução, e então verificar se a distância total não é maior que k. Observe que ambos esses passos podem ser executados por um algoritmo polinomial.

Todo problema que está em P também está em NP ou seja, P $\subseteq$ NP. A demonstração de que P está em NP vai além do escopo deste trabalho, mas pode ser vista detalhadamente em [6]. Ainda sobre a relação entre as classes P e NP, nos deparamos com uma das maiores questões ainda em

aberto na Teoria da Computação, que consiste em determinar se P = NP ou não[1] [6].

Para o correto entendimento da classe NPC, é necessária a compreensão de um procedimento denominado *redução polinomial*. Considere dois problemas de decisão Q e Q'. Uma **redução polinomial** de Q em Q' é um mapeamento, feito por uma função $f : i_Q \rightarrow i_{Q'}$, de uma instância qualquer i_Q de Q em uma instância $i_{Q'}$ de Q' tal que: 1) $i_{Q'} \in SIM_{Q'}$ se, e somente se, $i_Q \in SIM_Q$ e 2) f é calculada em tempo polinomial. Se existe uma redução polinomial de Q em Q', dizemos que Q é **polinomialmente redutível** em Q' e denotamos esse fato por $Q \propto Q'$. Além disso, dada essa redução, pode-se afirmar que, se Q' é resolvido em tempo polinomial, então Q também é resolvido em tempo polinomial. De maneira semelhante, se não existe um algoritmo polinomial que resolve Q, então também não existe um algoritmo polinomial que resolve Q' [4].

A classe **NPC** ou **NP-completo** consiste dos problemas que 1) estão em NP e 2) são *tão difíceis* quanto qualquer outro problema em NP [3]. Mais especificamente, um problema de decisão Q é um problema NP-completo se 1) $Q \in$ NP e 2) dado qualquer outro problema de decisão Q' em NP, $Q' \propto Q$ [6]. Isso significa que os problemas NP-completos são os problemas mais difíceis em NP. Intuitivamente, se existir um algoritmo polinomial que resolve um problema NP-completo, então qualquer outro problema em NP pode ser resolvido em tempo polinomial, provando assim que P = NP[2]. Geralmente, para provar que um dado problema de decisão é NP-completo, é preciso estabelecer uma redução polinomial de algum outro problema NP-completo nele [6].

A análise de problemas considerados difíceis não se limita somente aos problemas em NP. Existe também a classe dos problemas **NP-difíceis** que correspondem aos problemas *tão difíceis* quanto qualquer outro problema em NP. Mais precisamente, um problema de decisão Q é NP-difícil se, para qualquer outro problema Q' em NP, $Q' \propto Q$, independente se Q está em NP ou não [3, 6]. Isso significa que todo problema NP-completo é NP-difícil, ou seja, a classe NP-completo está contida na classe NP-difícil. No entanto, a classe NP-difícil não inclui somente problemas de decisão, ela inclui também problemas de otimização. Por isso, é comum dizermos que, se um problema de decisão Q_{D} é NP-completo, então seu respectivo problema de otimização Q é NP-difícil, pois sabe-se que Q_{D} não é mais difícil que Q e, de maneira geral, não é uma tarefa simples decidir se Q está ou não em NP. A versão de otimização do Problema do Caixeiro Viajante, por exemplo, é NP-difícil [6].

Assumindo que P $\neq$ NP, a prova de que um problema é NP-completo elimina qualquer possibilidade de resolvê-lo em tempo polinomial. No entanto, esse resultado não elimina, necessariamente, a possibilidade dele ser resolvido por um algoritmo pseudo-polinomial. A partir disso, classificamos como problemas *NP-completos no sentido forte* os problemas NP-completos que não podem ser resolvidos em tempo pseudo-polinomial. Explicando melhor, considere um problema NP-completo Q que possui um conjunto de instâncias I. Agora, considere um polinômio (sobre os inteiros) p e um subconjunto I' de I, tal que, para qualquer instância $i \in I'$,

$$\mathrm{Max}(i) \leq p(\mathrm{Length}(i)). \tag{2.1}$$

Observe que essa desigualdade impõe um limite superior em função do tamanho da entrada sobre a magnitude dos números nas instâncias em I'. Se existe um algoritmo pseudo-polinomial A que resolve Q, então Q restrito às instâncias em I', que denotaremos por Q_p, pode ser resolvido em tempo polinomial. Esse fato pode ser demonstrado utilizando o próprio algoritmo A para resolver

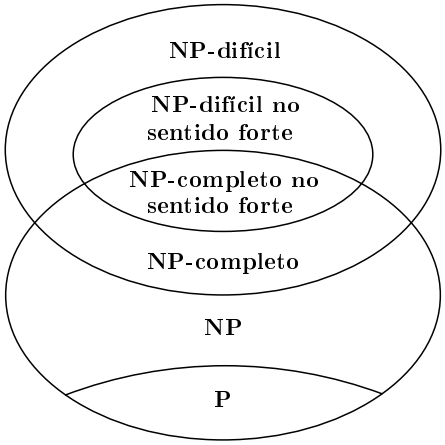
[1] Aqui vale salientar que existe uma forte crença entre os cientistas da computação de que P $\neq$ NP.

[2] No entanto, não se conhece nenhum problema NP-completo que possa ser resolvido por um algoritmo polinomial.

Q_p. Uma vez que todas as instâncias de Q_p satisfazem a desigualdade 2.1, A é executado em tempo polinomial tomando qualquer instância $i \in I'$ como entrada. Agora, suponhamos que Q_p é um problema NP-completo, ou seja, ele não pode ser resolvido por um algoritmo polinomial. Então Q_p também não pode ser resolvido por um algoritmo pseudo-polinomial, pois qualquer algoritmo pseudo-polinomial que o resolve executaria, de fato, em tempo polinomial, contradizendo nosso cenário hipotético. Intuitivamente, Q também não pode ser resolvido por um algoritmo pseudo-polinomial, caso contrário, tal algoritmo seria utilizado para resolver Q_p em tempo polinomial, contrariando novamente nosso cenário hipotético. Dessa forma, dizemos que um problema de decisão Q é **NP-completo no sentido forte** se $Q \in$ NP e Q_p é NP-completo. De maneira semelhante, dizemos que um problema Q é **NP-difícil no sentido forte**, se Q_p é NP-difícil, independente se Q está ou não em NP [4, 6].

Todas as classes de problemas computacionais apresentadas até agora relacionam-se entre si. Esse relacionamento encontra-se esquematizado na Figura 2.1, onde a classe P é representada como um subconjunto da classe NP e a classe NP-completo é representada como a interseção da classe NP e da classe NP-difícil.

Figura 2.1: *Diagrama que representa a relação entre as classes de problemas computacionais.*



Uma observação importante sobre os problemas NP-completos no sentido forte está relacionada com os ditos *problemas numéricos*. Os problemas de decisão cujos conjuntos de instâncias *não* satisfazem a desigualdade 2.1 são chamados de **problemas numéricos** (do inglês, *number problems*). Os *únicos* problemas NP-completos que podem ser resolvidos em tempo pseudo-polinomial são os problemas numéricos [6]. A partir disso, dado um problema de decisão Q, se Q é NP-completo e *não* é um problema numérico, então Q é um problema NP-completo no sentido forte. Dessa forma, pode-se provar que um problema não numérico é NP-completo no sentido forte simplesmente verificando se ele é NP-completo. Agora, quando o problema em questão é numérico, seguimos uma abordagem diferente, descrita a seguir.

Uma maneira de verificar se um dado problema numérico é NP-completo no sentido forte é através de uma *redução pseudo-polinomial* [4, 6]. A redução pseudo-polinomial é semelhante à redução polinomial, porém, com algumas condições adicionais. Considere Q e Q' dois problemas de decisão com conjuntos de instâncias I_Q e $I_{Q'}$, respectivamente. Uma redução **pseudo-polinomial** de Q em Q' é uma função $f : I_Q \to I_{Q'}$ tal que:

1. $\forall i \in I_Q, i \in SIM_Q$ se, e somente se, $f(i) \in SIM_{Q'}$.

2. A função f pode ser calculada em tempo pseudo-polinomial.

3. Existe uma função polinomial p tal que $\forall i \in I_Q$, $p(\text{Length}(f(i))) \geq \text{Length}(i)$.

4. Exite uma função polinomial q tal que $\forall i \in I_Q$, $\text{Max}(f(i))) \leq q(\text{Max}(i), \text{Length}(i))$.

Note que as condições 1 e 2 são quase idênticas às da redução polinomial, exceto que agora há um pouco mais de liberdade na complexidade de tempo para se calcular f, pois ela pode ser determinada em tempo pseudo-polinomial. A condição 3 requer que a redução não cause uma diminuição significativa no tamanho da entrada, e a condição 4 assegura que a magnitude do maior número da instância i não cresça exponencialmente em termos de $\text{Max}(i)$ e $\text{Length}(i)$. Se Q é NP-completo no sentido forte, $Q' \in \text{NP}$, e existe uma redução pseudo-polinomial de Q em Q', então Q' é NP-completo no sentido forte [6].

2.2 Conceitos Envolvendo Sequências

Dentre a imensa variedade de problemas computacionais existentes, podemos destacar aqueles que envolvem sequências de caracteres, que vão desde problemas simples, como encontrar uma palavra em um texto, até problemas mais complexos, como determinar o número mínimo de operações para transformar uma sequência em outra. Nesse contexto, serão apresentadas nesta seção as principais notações e conceitos relacionados a sequências.

2.2.1 Definições Básicas

Um **alfabeto** Σ é um conjunto finito de caracteres. Uma **sequência** s construída sobre Σ corresponde a uma concatenação ordenada e finita de caracteres em Σ. A partir do alfabeto $\Sigma = \{\text{A, B, C, D, R}\}$, por exemplo, podemos formar as sequências $s = \text{ABRACADABRA}$ e $t = \text{BRADA}$. Perceba que existe um grande número de sequências distintas que podem ser construídas sobre Σ. Representamos por Σ^* o conjunto de todas as sequências finitas construídas sobre Σ. Definimos ainda $\overline{\Sigma} = \Sigma \cup \{\text{`}-\text{'}\}$, onde '$-$' representa um espaço. O **tamanho** de uma sequência s é a quantidade de caracteres que s possui e é representado por $|s|$. Denotamos por $s[i]$, com $1 \leq i \leq |s|$, o i-ésimo caractere de uma sequência s. No exemplo anterior, onde $s = \text{ABRACADABRA}$ e $t = \text{BRADA}$, temos que $|s| = 11$, $|t| = 5$, $s[2] = \text{B}$ e $t[4] = \text{D}$. Chamamos uma sequência s de **sequência vazia** se $|s| = 0$ e a representamos por ϵ. A **concatenação** de duas sequências s e t, denotada por $u = s \bullet t$, é uma operação que gera uma terceira sequência u que consiste dos caracteres de s seguidos pelos caracteres de t na mesma ordem em que aparecem em suas respectivas sequências.

Uma **subsequência** s' de uma sequência s consiste em uma cópia de s com alguns, todos ou nenhum dos seus caracteres removidos. Já um **segmento** de uma sequência s é um trecho com zero ou mais caracteres consecutivos de s. Uma segmento de s, onde o i-ésimo e o j-ésimo caracteres de s correspondem, respectivamente, ao primeiro e último caracteres do segmento, é denotado por $s[i..j]$. Caso $i > j$, temos um segmento vazio. Considerando novamente a sequência s do nosso exemplo, podemos construir as subsequências $s_1 = \text{ABADABA}$, $s_2 = \text{ABRACADABRA}$ e $s_3 = \epsilon$, e definir os segmentos $s[5..8] = \text{CADA}$ e $s[2..2] = \text{B}$. Um **prefixo** de uma sequência s é um segmento $s[1..j]$ tal que $1 \leq j \leq |s|$. Se $j < 1$, chamamos o prefixo de **prefixo vazio**, que equivale à sequência vazia. De forma semelhante, um **sufixo** de uma sequência s é um segmento $s[i..|s|]$ tal que $1 \leq i \leq |s|$. Se $i > |s|$, chamamos o sufixo de **sufixo vazio**, que equivale à sequência vazia.

Seja $s_1 = s[i..j]$ um segmento de uma sequência s. Denotamos por $first(s_1)$ a posição do primeiro caractere de s_1 em s e $last(s_1)$ a posição do último caractere de s_1 em s, ou seja, $first(s_1) = i$ e $last(s_1) = j$. Agora, considere $B = \{s_1, s_2, ..., s_k\}$ um conjunto de k segmentos de uma sequência s. Dizemos que B é um **conjunto ordenado de segmentos** se, para qualquer segmento $s_i \in B$:

- $first(s_i) < first(s_{i+1})$ ou
- $first(s_i) = first(s_{i+1})$ e $last(s_i) < last(s_{i+1})$.

Dada uma sequência s e dois segmentos $s_1 = s[i..j]$ e $s_2 = s[k..l]$ de s, dizemos que s_1 e s_2 são **segmentos sobrepostos** se $i \leq k \leq j$ ou $i \leq l \leq j$ ou $k \leq i \leq l$ ou $k \leq j \leq l$. Diferentemente, se s_1 termina antes de s_2 começar, ou seja, se $j < k$, dizemos que s_1 **precede** s_2, e denotamos esta relação por $s_1 \prec s_2$. Dado um conjunto ordenado de segmentos B de uma sequência s, um subconjunto $\Gamma_B = \{s_1, s_2, ..., s_p\}$ de B é dito uma **cadeia de segmentos** de B se $s_1 \prec s_2 \prec ... \prec s_p$, e denotamos a concatenação dos segmentos de Γ_B por $\Gamma_B^\bullet$, ou seja, $\Gamma_B^\bullet = s_1 \bullet s_2 \bullet ... \bullet s_p$. Um exemplo de cadeia de segmentos pode ser visto na Figura 2.2.

Figura 2.2: *Exemplo de uma cadeia de segmentos Γ_B de um conjunto ordenado de segmentos B construído a partir de uma sequência s. Os números no canto superior esquerdo dos segmentos indicam a ordem deles em B.*

$$s = \text{ABRACARBBADARBRA}$$

$$B = \left\{ \begin{array}{ccc} {}^{1}\textbf{ABR} & {}^{4}\text{CAR} & {}^{6}\text{BAD} \quad {}^{9}\textbf{BRA} \\ {}^{3}\textbf{ACA} & {}^{5}\text{BBA} & {}^{8}\text{RBR} \\ {}^{2}\text{BRACA} & {}^{7}\textbf{DA} & \end{array} \right\}$$

$$\Gamma_B = \{{}^{1}\textbf{ABR}\ {}^{3}\textbf{ACA}\ {}^{7}\textbf{DA}\ {}^{9}\textbf{BRA}\}$$

2.2.2 Problema da Similaridade de Duas Sequências

Dentre os principais problemas envolvendo sequências destaca-se o Problema da Similaridade de Duas Sequências, que está diretamente relacionado com o problema abordado nesta dissertação. Antes de definirmos o Problema da Similaridade de Duas Sequências, apresentaremos algumas definições necessárias para o entendimento dele.

Seja Σ um alfabeto qualquer que não inclua o caractere espaço ('$-$'), e u e v duas sequências construídas sobre Σ tais que $|u| = n$ e $|v| = m$. Um **alinhamento** M das sequências u e v é uma matriz $2 \times l$, com $n, m \leq l \leq m + n$, tal que a primeira e segunda linha de M incluem, respectivamente, os caracteres de u e v, na mesma ordem em que aparecem em suas respectivas sequências, intercalados com $l - n$ e $l - m$ espaços. Além disso, não existe em M colunas contendo somente espaços, ou seja, $\nexists i$ tal que $M[1][i] = M[2][i] =$ '$-$'. As sequências u e v com espaços intercalados são representadas por $\overline{u}$ e $\overline{v}$, respectivamente, e pela definição de alinhamento, são tais que $\overline{u}$ e $\overline{v} \in \overline{\Sigma}$ e $|\overline{u}| = |\overline{v}| = l$ [7]. Nesse contexto, as colunas de M que contém um espaço na primeira ou na segunda linha são chamadas ***spaces***. Já as colunas que contém o mesmo caractere são chamadas ***matches*** e as colunas com caracteres diferentes são chamadas ***mismatches***. A matriz M mostrada na Figura 2.3 é um exemplo de alinhamento das sequências $u =$ INCIDENTAR e $v =$ CADERNETA, construídas sobre o alfabeto $\Sigma = \{$A, C, D, E, I, N, R, T$\}$. As colunas 1, 2, 7, 9 e 12 de M são *spaces*, as colunas 3, 5, 6, 8, 10 e 11 são *matches* e a coluna 4 é um *mismatch*.

Figura 2.3: *Exemplo de um alinhamento M das sequências u = INCIDENTAR e v = CADERNETA.*

	1	2	3	4	5	6	7	8	9	10	11	12	
$M =$	I	N	C	I	D	E	–	N	–	T	A	R	$= \overline{u}$
	–	–	C	A	D	E	R	N	E	T	A	–	$= \overline{v}$

Dada uma **função de pontuação** $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$, que atribui um valor real para cada par de caracteres (c_1, c_2) pertencentes a $\overline{\Sigma}$, a **pontuação de um alinhamento** M, denotada por $Score_\omega(M)$, é definida como $\sum_{i=1}^{l} \omega(M[1][i], M[2][i])$. A Tabela 2.1 apresenta um exemplo de função de pontuação para o alfabeto $\overline{\Sigma} = \{\text{A, C, D, E, I, N, R, T}\} \cup \{\text{'–'}\}$, onde o valor $\omega(c_1, c_2)$ de cada par de caracteres (c_1, c_2) encontra-se na linha c_1 e coluna c_2 da tabela.

Tabela 2.1: *Exemplo de uma função de pontuação ω para o alfabeto $\overline{\Sigma}$ = {A, C, D, E, I, N, R, T} $\cup$ {'–'}.*

c_1 \ c_2	A	C	D	E	I	N	R	T	–
A	3	1	1	1	1	1	1	1	−1
C		2	1	1	1	1	1	1	−2
D			3	1	1	1	1	1	−1
E				2	1	1	1	1	−2
I					3	1	1	1	−1
N						2	1	1	−1
R							3	1	−2
T								2	−3
–									0

Considerando o alinhamento M apresentado na Figura 2.3 e a função de pontuação representada pela Tabela 2.1, podemos determinar a pontuação de cada coluna de M, como apresentado na Figura 2.4, e calcular o valor de $Score_\omega(M) = \sum_{i=1}^{12} \omega(M[1][i], M[2][i]) = 7$.

Figura 2.4: *Alinhamento M com pontuação.*

	1	2	3	4	5	6	7	8	9	10	11	12	
$M =$	I	N	C	I	D	E	–	N	–	T	A	R	$= \overline{u}$
	–	–	C	A	D	E	R	N	E	T	A	–	$= \overline{v}$
$\omega =$	−1	−1	2	1	3	2	−2	2	−2	2	3	−2	

É fácil ver que podem existir vários alinhamentos distintos entre duas sequências, cada um com sua pontuação correspondente. Chamamos então de **similaridade** de duas sequências u e v com respeito a uma função de pontuação ω, denotada por $sim_\omega(u, v)$, a maior dentre as pontuações de todos os possíveis alinhamentos de u e v. Mais precisamente, dado o conjunto Π dos possíveis alinhamentos das sequências u e v, $sim_\omega(u, v) = max\{Score_\omega(M) : M \in \Pi\}$. A partir disso, denominamos de **alinhamento ótimo** das sequências u e v um alinhamento cuja pontuação é igual a $sim_\omega(u, v)$. Repare que podem existir alinhamentos ótimos distintos entre as mesmas sequências u e v, como demonstrado na Figura 2.5.

Com base nas definições anteriores, temos o seguinte problema conhecido como **Problema da Similaridade de Duas Sequências** [8, 17]:

Problema da Similaridade de Duas Sequências (PSDS): *dadas duas sequências u e v construídas sobre um alfabeto Σ tal que $\{-\} \notin \Sigma$ e uma função de pontuação ω, encontrar o valor da*

Figura 2.5: *Três alinhamentos ótimos distintos das sequências TREINAR e NECTAR. A similaridade dessas sequências, calculada pela função de pontuação da Tabela 2.1, é igual a 9.*

T	R	E	I	N	A	R
N	-	E	C	T	A	R
1	-2	2	1	1	3	3

T	R	E	I	N	A	R
N	E	C	T	-	A	R
1	-2	2	1	1	3	3

T	R	E	I	N	A	R
N	E	C	-	T	A	R
1	-2	2	1	1	3	3

similaridade $sim_\omega(u, v)$ dessas sequências.

Uma maneira de determinar a similaridade de duas sequências u e v é gerar todos os possíveis alinhamentos delas e devolver a pontuação de um dos alinhamentos ótimos gerados. Entretanto, essa estratégia é inviável porque o número de alinhamentos distintos entre duas sequências é exponencial em relação ao tamanho delas [18]. Felizmente, existe um abordagem muito mais rápida baseada na técnica de *programação dinâmica* que resolve o PSDS em tempo polinomial.

Dadas duas sequências u e v e uma função de pontuação ω, a ideia do algoritmo baseado em programação dinâmica para se determinar o valor de $sim_\omega(u, v)$ é calcular a similaridade de u e v através do cálculo da similaridade de todos os prefixos de ambas essas sequências. Observe que, considerando a cadeia vazia, existem $|u| + 1$ prefixos de u e $|v| + 1$ prefixos de v. Então, podemos organizar as soluções dos subproblemas do PSDS em uma matriz D de dimensões $(|u| + 1) \times (|v| + 1)$, onde $D[i][j]$ armazena o valor de $sim_\omega(u[1..i], v[1..j])$. Essa matriz é conhecida como **matriz de alinhamento**. Baseando-se nessa organização, existem três ações que são consideras para se determinar a similaridade de $u[1..i]$ e $v[1..j]$:

- determinar a similaridade de $u[1..i]$ e $v[1..j-1]$ e somar a ela o valor $\omega(\text{`}-\text{'}, v[j])$, referente a alinhar um espaço com $v[j]$;
- determinar a similaridade de $u[1..i-1]$ e $v[1..j-1]$ e somar a ela o valor $\omega(u[i], v[j])$, referente a alinhar $u[i]$ com $v[j]$;
- determinar a similaridade de $u[1..i-1]$ e $v[1..j]$ e somar a ela o valor $\omega(u[i],\text{`}-\text{'})$, referente a alinhar $u[i]$ com um espaço.

A partir dessas possíveis ações e observando que estamos em busca da maior pontuação, podemos determinar o valor que será armazenado em $D[i][j]$ examinando as posições $D[i][j-1]$, $D[i-1][j-1]$ e $D[i-1][j]$ e escolhendo a combinação de maior valor. Formalmente falando, preenchemos a posição $D[i][j]$ com a Recorrência 2.2, apresentada a seguir:

$$D[i][j] = max \begin{cases} D[i][j-1] + \omega(\text{`}-\text{'}, v[j]) \\ D[i-1][j-1] + \omega(u[i], v[j]) \\ D[i-1][j] + \omega(u[i],\text{`}-\text{'}). \end{cases} \tag{2.2}$$

A respeito da primeira linha e da primeira coluna de D, elas são preenchidas da seguinte forma: $D[i][0] = D[i-1][0] + \omega(u[i],\text{`}-\text{'})$, com $1 \leq i \leq |u|$, e $D[0][j] = D[0][j-1] + \omega(\text{`}-\text{'}, v[j])$, com $1 \leq i \leq |v|$, e $D[0][0] = 0$. Após o preenchimento da matriz D, o valor da similaridade entre as sequências u e v estará armazenado na última posição da matriz, ou seja, $D[|u|][|v|]$ [8].

É importante observar que, após o cálculo de $sim_\omega(u, v)$, pode-se construir um alinhamento ótimo das sequências u e v partindo-se da posição $[|u|][|v|]$ da matriz D e percorrendo-a no sentido

contrário àquele utilizado para o seu preenchimento, ou seja, voltando nas células correspondentes aos valores que deram origem à similaridade determinada.

As ideias apresentadas acima constituem um algoritmo bastante conhecido para determinação da similaridade de duas sequências denominado algoritmo de Needleman-Wunsch [9]. Sobre a sua complexidade de tempo, considerando que o valor de cada posição $D[i][j]$ pode ser determinado em tempo constante, o preenchimento da primeira linha e primeira coluna consomem tempo $O(|v|)$ e $O(|u|)$, respectivamente, e as demais posições $O(|u||v|)$. Portanto, a complexidade de tempo do algoritmo para determinar o valor de $sim_\omega(u, v)$ é $O(|u||v|)$. Recentemente, provou-se que essa complexidade de tempo é um limite inferior para o problema, isto é, não existe algoritmo capaz de resolver o PSDS cuja complexidade de tempo é menor que $O(|u||v|)$ [19].

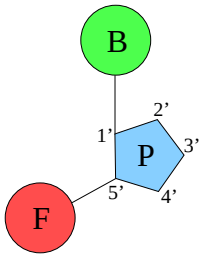
2.3 Conceitos de Biologia Molecular

A Bioinformática e a Biologia Computacional lidam com problemas biológicos e os abordam através da Ciência da Computação. Por isso, além de conhecimentos em Computação, a compreensão de conceitos básicos de Biologia e, mais especificamente, Biologia Molecular, são essenciais para lidar com tais problemas. Nesta seção serão apresentados alguns desses conceitos, com ênfase naqueles que estão relacionados ao tema desta dissertação.

2.3.1 O DNA, o RNA e as Proteínas

Todos os seres vivos conhecidos são formados por **células**, pequenas unidades fundamentais protegidas por uma membrana, preenchidas com soluções químicas e dotadas da habilidade de criar cópias delas mesmas [20]. No interior de todas as células de um organismo está armazenada a informação genética dele, responsável pelas suas funcionalidades e características. A essa informação genética dá-se o nome de **genoma** [20]. O genoma é armazenado em moléculas de **ácido desoxirribonucleico** (DNA, do inglês *deoxyribonucleic acid*), que são formadas por unidades básicas denominadas **nucleotídeos**. Os nucleotídeos se constituem de um grupo fosfato, uma pentose (açúcar) e uma dentre quatro bases nitrogenadas: *adenina* (A), *citosina* (C), *guanina* (G) ou *timina* (T) (Figura 2.6). Por se diferenciarem apenas pela base nitrogenada que possuem, é comum representar os nucleotídeos que contém adenina, citosina, guanina e timina pelas letras A, C, G e T, respectivamente. Para facilitar a referência dos cinco átomos de carbono da pentose que compõe o nucleotídeo, eles são numerados de 1' a 5'. Dessa forma, o fosfato liga-se com a pentose através do carbono 5' e com a base nitrogenada através do carbono 1' [20, 21].

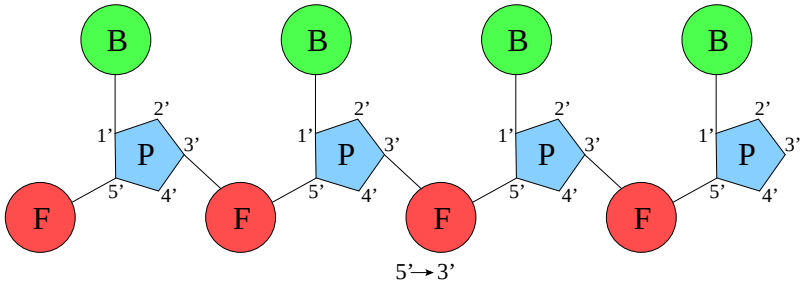
Figura 2.6: *Estrutura de um nucleotídeo, onde F corresponde a um grupo fosfato, P a uma pentose e B a uma das bases nitrogenadas.*



Os nucleotídeos que constituem o DNA são unidos por uma ligação covalente do carbono 3' de

uma pentose com o carbono 5' da pentose seguinte, ligação essa que se faz através de um fosfato. Dessa forma, os nucleotídeos se agrupam em uma cadeia, ou fita, que pode ser visualizada como uma sequência de bases nitrogenadas ligadas a uma espinha dorsal constituída de fosfatos e pentoses (Figura 2.7) [20]. A cadeia de nucleotídeos é organizada de tal forma que se permite estabelecer um sentido, ou polaridade, para ela. Observe que todo nucleotídeo liga-se a outros dois por meio dos carbonos 3' e 5' de sua pentose, exceto os nucleotídeos das extremidades da cadeia. A extremidade da cadeia cujo nucleotídeo não se liga a outro pelo carbono 3' chama-se **extremidade 3'**. De maneira análoga, chama-se de **extremidade 5'** a extremidade da cadeia cujo nucleotídeo não se liga a outro pelo carbono 5'. Sendo assim, se uma cadeia de nucleotídeos começa na extremidade 5' e termina na extremidade 3', dizemos que ela possui sentido **5' → 3'**. De maneira análoga, se a cadeia de nucleotídeos começa na extremidade 3' e termina na extremidade 5', dizemos que ela possui sentido **3' → 5'**.

Figura 2.7: *Exemplo de uma cadeia de nucleotídeos, onde a extremidade 5' está à esquerda e a extremidade 3' está à direita.*



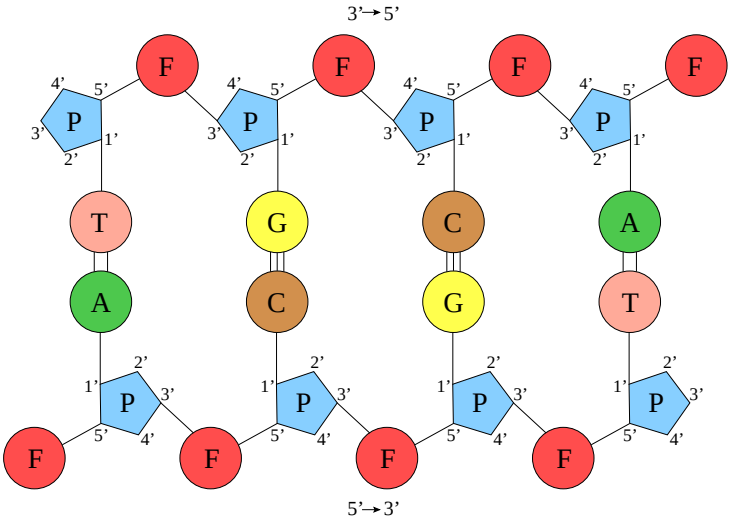
Estruturalmente, o DNA constitui-se de duas fitas de nucleotídeos interligadas por pontes de hidrogênio que ligam pares de nucleotídeos, com cada nucleotídeo desse par pertencendo a uma fita distinta. Nesse contexto, A liga-se com T através de duas pontes de hidrogênio, e C liga-se com G através de três pontes de hidrogênio. Assim, dizemos que existe uma **complementaridade** entre as bases, onde A é complementar a T, e C é complementar G. Da mesma forma, T é complementar a A e G é complementar a C. As fitas do DNA possuem sentidos opostos, e por isso são ditas **antiparalelas** (Figura 2.8a). Com isso, dizemos que uma fita do DNA é o **complemento reverso** da outra, ou seja, ela possui sentido oposto e as bases complementares da outra fita. As duas cadeias de nucleotídeos do DNA são torcidas de modo a formar a conhecida estrutura de *dupla hélice*, onde as bases nitrogenadas encontram-se na parte interior, enquanto que a espinha dorsal formada pelas pentoses e fosfatos encontram-se na parte exterior da estrutura (Figura 2.8b) [20, 21].

Além do DNA, na célula existe um outro tipo de molécula que se assemelha muito a ele, o **ácido ribonucleico** (RNA, do inglês *ribonucleic acid*). O RNA possui uma estrutura semelhante ao DNA, sendo que ele também corresponde a uma cadeia de nucleotídeos ligados por uma espinha dorsal formada pela alternância entre pentose e fosfato. No entanto, o RNA difere em três aspectos do DNA. Diferentemente do DNA, onde a pentose chama-se **desoxirribose** (por isso o nome ácido desoxirribonucleico), no RNA a pentose chama-se **ribose** (daí o nome ácido ribonucleico) [20, 21]. Além disso, o RNA corresponde a uma fita única de nucleotídeos, e não a duas fitas ligadas por pontes de hidrogênio, como o DNA. A outra diferença é que o RNA possui a base uracila (U) ao invés da timina (T), ou seja, ele corresponde a uma sequência formada pelas letras A, C, G e U.

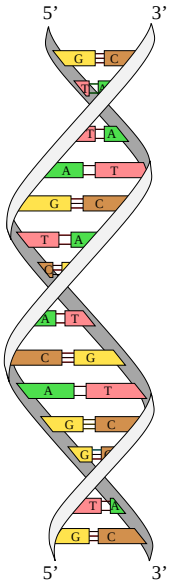
Ainda no interior das células encontram-se as **proteínas**. Essas moléculas executam uma grande variedade de funções como controlar a entrada e saída de moléculas na célula, transmitir mensagens

Figura 2.8: *Exemplificação da organização básica do DNA.*

(a) *Duas cadeias de nucleotídeos ligam-se para formar a estrutura do DNA.*



(b) *Estrutura tridimensional do DNA na forma de dupla hélice.*



entre as células e atuar como anticorpos e fontes de luminescência [20]. Diferentemente do DNA e do RNA que são formados por nucleotídeos, as proteínas são formadas por aminoácidos que se dividem em 20 tipos diferentes (Tabela 2.2). Cada aminoácido une-se a outro através de uma ligação peptídica, formando uma sequência linear. Tal sequência de aminoácidos dobra-se sobre si mesma, gerando uma estrutura tridimensional própria que depende da ordem em que os aminoácidos estão distribuídos [20, 21].

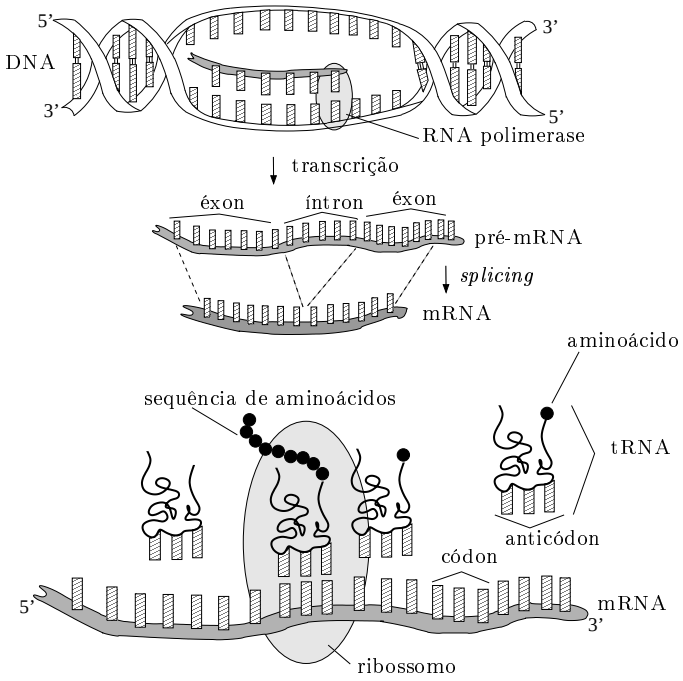
O DNA, o RNA e as proteínas estão diretamente relacionados em um processo que ocorre em todas as células. Em tal processo, que será detalhado na próxima seção, a informação genética contida no DNA é expressa com o auxílio do RNA, produzindo as proteínas.

2.3.2 Expressão Gênica

O DNA e o RNA se inter-relacionam através de um processo chamado **expressão gênica**. Nesse processo, esquematizado na Figura 2.9, proteínas ou RNAs são sintetizados a partir da expressão de trechos específicos do DNA, chamados **genes**. Frequentemente em eucariotos, as sequências dos genes que codificam proteínas são interrompidas por um ou mais trechos de DNA chamados **íntrons**, que, aparentemente, não possuem função. Já os trechos da sequência do gene que codificam de fato as proteínas são chamados de **éxons** [21].

Um tipo específico de RNA, chamado **RNA mensageiro** (mRNA), atua como intermediário no processo de transformar a informação contida nos genes em proteínas. A expressão gênica se inicia com a fase de **transcrição**, na qual uma molécula de RNA, chamada **pré-mRNA**, é produzida a partir de um gene. Nessa fase, a dupla hélice do trecho de DNA correspondente ao gene sendo expresso é rompida, e as duas fitas permanecem temporariamente separadas. Uma das fitas atua como *molde* para a produção da molécula de pré-mRNA. Um tipo de enzima, chamada **RNA polimerase**, liga-se à fita molde e a percorre no sentido 3' → 5'. À medida que a RNA polimerase percorre cada nucleotídeo do gene, ela adiciona o nucleotídeo complementar correspondente na

Figura 2.9: *Esquematização do processo de expressão gênica. Nesta figura, o produto final é uma sequência de aminoácidos (proteína). Figura adaptada de Kishi [1].*



molécula de pré-mRNA. Assim, a molécula de pré-mRNA é criada gradualmente no sentido 5' → 3', com os nucleotídeos complementares aos da fita molde, ou seja, tal mRNA será o complemento reverso da fita molde, mas com a diferença de que ele possui a base U ao invés da base T. Quando a RNA polimerase chega ao fim do gene, a molécula de pré-mRNA recém criada é liberada e a ligação entre as duas fitas do DNA é restabelecida. É importante notar que tanto os éxons quanto os íntrons de um gene são transcritos para o pré-mRNA produzido a partir dele. Por isso, considera-se que as moléculas de pré-mRNA também possuem íntrons e éxons, embora eles sejam trechos do gene.

Normalmente, em procariotos o pré-mRNA já está pronto para ser transformado em proteína assim que ele é produzido. No entanto, em organismos eucariotos é comum a ocorrência de uma fase adicional chamada **recomposição** (ou *splicing*, do inglês). Durante a recomposição, trechos que não codificam aminoácidos, os íntrons, são removidos do pré-mRNA, enquanto que os trechos restantes, os éxons, são unidos de forma que o pré-mRNA volte a ser uma sequência linear. Com isso, o pré-mRNA que contém somente os éxons passa a ser chamado de **mRNA maduro**, ou somente mRNA.

A última fase da síntese de proteínas consiste na tradução da molécula de mRNA em uma cadeia de aminoácidos. Por isso, essa fase recebe o nome de **tradução**. Cada aminoácido é traduzido a partir de uma trinca de nucleotídeos, chamada de **códon**. Esse sistema de codificação que associa um códon a um aminoácido é conhecido como **código genético**, e é apresentado na Tabela 2.2. Durante a fase de tradução, o mRNA liga-se ao ribossomo, uma estrutura formada por um outro tipo de RNA denominado RNA ribossômico (rRNA), e diversas proteínas. Então, com a ajuda de um terceiro tipo de RNA, chamado de RNA transportador (tRNA), cada códon do mRNA é traduzido em um aminoácido. Explicando melhor essa fase, cada molécula de tRNA carrega um aminoácido e uma trinca de nucleotídeos, chamada de **anticódon**, que corresponde ao complemento reverso do códon associado ao aminoácido carregado por ela. Quando o ribossomo se posiciona sobre um códon do mRNA, ocorre a ligação desse códon com o anticódon complementar carregado por uma

molécula de tRNA. O aminoácido trazido pelo tRNA é então retido no ribossomo, que avança para o próximo códon do mRNA, no sentido 5' → 3', de forma a permitir que o próximo tRNA ligue-se a ele enquanto que o tRNA anterior é liberado. Dessa forma, a cadeia de aminoácidos que constitui a proteína é criada gradualmente. Quando o ribossomo alcança um códon correspondente a um sinal de parada (**STOP**), ele libera a cadeia de aminoácidos e se desprende do mRNA. A partir desse ponto, a cadeia de aminoácidos sofre alguns eventos pós-traducionais, dentre os quais está o dobramento em sua estrutura tridimensional própria, para então a proteína começar a exercer suas funções na célula [20, 21].

Tabela 2.2: *Os 20 tipos de aminoácidos e os códons correspondentes a eles com os nucleotídeos na primeira, segunda e terceira posição na orientação 5' → 3'. O termo* ***STOP*** *corresponde ao sinal de parada.*

1ª base	2ª base				3ª base
	U	C	A	G	
U	Fenilalanina (F)	Serina (S)	Tirosina (Y)	Cisteína (C)	U
	Fenilalanina (F)	Serina (S)	Tirosina (Y)	Cisteína (C)	C
	Leucina (L)	Serina (S)	**STOP**	**STOP**	A
	Leucina (L)	Serina (S)	**STOP**	Triptofano (W)	G
C	Leucina (L)	Prolina (P)	Histidina (H)	Arginina (R)	U
	Leucina (L)	Prolina (P)	Histidina (H)	Arginina (R)	C
	Leucina (L)	Prolina (P)	Glutamina (Q)	Arginina (R)	A
	Leucina (L)	Prolina (P)	Glutamina (Q)	Arginina (R)	G
A	Isoleucina (I)	Treonina (T)	Asparagina (N)	Serina (S)	U
	Isoleucina (I)	Treonina (T)	Asparagina (N)	Serina (S)	C
	Isoleucina (I)	Treonina (T)	Lisina (K)	Arginina (R)	A
	Metionina (M)	Treonina (T)	Lisina (K)	Arginina (R)	G
G	Vanila (V)	Alanina (A)	Aspartato (D)	Glicina (G)	U
	Vanila (V)	Alanina (A)	Aspartato (D)	Glicina (G)	C
	Vanila (V)	Alanina (A)	Glutamato (E)	Glicina (G)	A
	Vanila (V)	Alanina (A)	Glutamato (E)	Glicina (G)	G

É interessante observar que o mRNA não é o único tipo de RNA que é produzido a partir dos genes. Existem genes cujo produto final não são as proteínas, mas tipos específicos de RNA, como o rRNA, o tRNA, já mencionados anteriormente, e o pequeno RNA nuclear (snRNA, do inglês *small nuclear* RNA) que controla a recomposição do pré-mRNA [20].

O processo de expressão gênica, cujo produto final são as proteínas, é um procedimento fundamental que ocorre em todas as células de todos os seres vivos. Por isso, ele recebe o nome de **Dogma Central da Biologia Molecular** [20].

2.3.3 O Transcriptoma

O transcriptoma está diretamente relacionado ao processo de expressão gênica. No contexto de tal processo, denominamos de **transcrito** uma molécula de RNA que foi transcrita a partir do DNA [12, 20]. No interior da célula existem vários transcritos distintos, os quais, por sua vez, possuem várias cópias. Ao conjunto de cópias de um transcrito dá-se o nome de **abundância do transcrito** ou simplesmente **abundância** [12]. O **transcriptoma** de uma célula é definido como o conjunto de transcritos existentes na célula em um dado momento juntamente com suas respectivas abundâncias [12, 22].

Durante o processo de recomposição do pré-mRNA, onde os íntrons são eliminados, pode ocorrer um fenômeno chamado **recomposição alternativa**. Esse fenômeno, muito comum em eucariotos, consiste na produção de duas ou mais moléculas distintas de mRNA a partir de um mesmo gene [10, 11]. Para se ter uma perspectiva, mais de 90% dos genes do ser humano sofrem recomposição alternativa [23]. Existem diferentes tipos de recomposição alternativa, dentre eles estão a eliminação de trechos de éxons ou éxons inteiros juntamente com os íntrons, e a retenção de um ou mais íntrons, que permanecem no mRNA após a recomposição [11]. Considerando um gene i que dá origem a um conjunto de transcritos distintos T_i, chamamos cada transcrito $t \in T_i$ de uma **isoforma** do gene i.

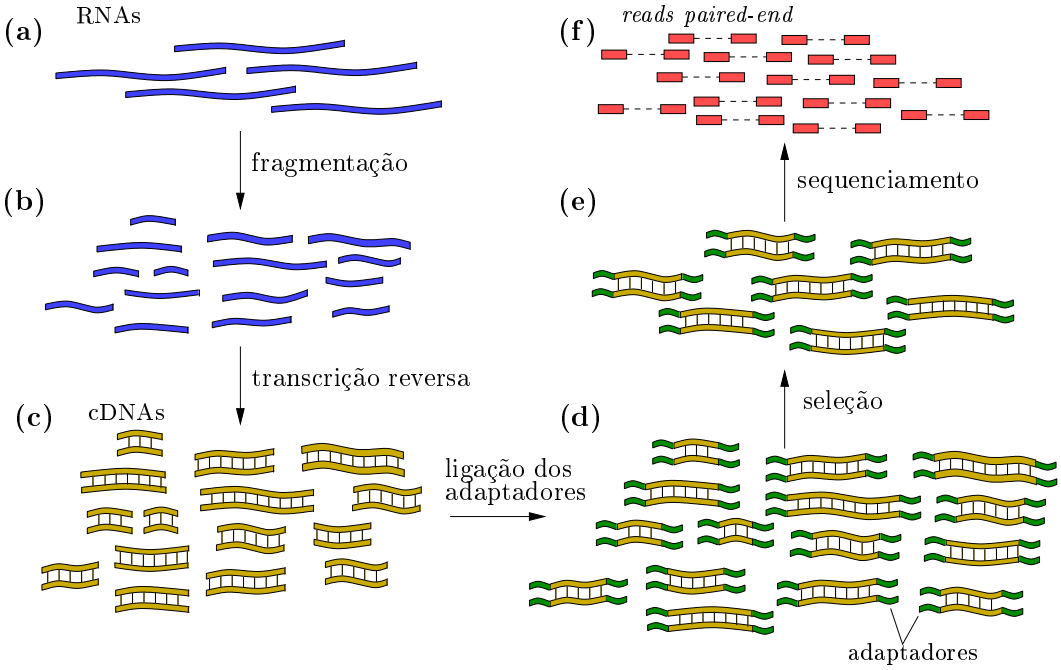
Da mesma forma que uma molécula de RNA é transcrita a partir do DNA, é possível realizar o processo inverso, onde uma molécula unifilamentar de DNA é sintetizada a partir de uma molécula de RNA. Esse processo, conhecido como **transcrição reversa**, é feito por meio de uma enzima chamada **transcriptase reversa** e é utilizado na produção do **DNA complementar** (cDNA). O cDNA consiste no DNA bifilamentar produzido a partir do RNA e é utilizado em um processo, descrito a seguir, cujo objetivo é proporcionar a análise do transcriptoma.

Uma forma de estudar o transcriptoma de um organismo é através de um experimento denominado RNA-Seq. O **RNA-Seq** (RNA *Sequencing*) é um protocolo de sequenciamento de moléculas de RNA que se utiliza da tecnologia de Sequenciamento da Nova Geração (NGS) [2, 22]. Um experimento típico de RNA-Seq possui as fases de geração e análise de dados. Para gerar os dados, primeiramente ocorre a extração dos transcritos presentes na célula em um dado momento, seguida pela quebra dessas moléculas em fragmentos, quebra essa realizada em posições aleatórias das moléculas de RNA (Figura 2.10a e 2.10b). Após a fragmentação, ocorre o processo de transcrição reversa, onde cada fragmento dá origem a um fragmento de cDNA (Figura 2.10c). Sequências específicas da tecnologia de sequenciamento a ser utilizada, chamadas **adaptadores** (*adaptors*), são então adicionadas às extremidades de cada fragmento de cDNA (Figura 2.10d), que são então classificados de acordo com seus tamanhos (Figura 2.10e). Finalmente, uma ou ambas as extremidades dos cDNAs são sequenciadas por meio de tecnologias de sequenciamento da nova geração, gerando pedaços sequenciados dos RNAs denominados ***reads***[3]. Os *reads* são chamados de *single-end* quando apenas uma extremidade do fragmento de cDNA foi sequenciada ou *paired-end* quando ambas extremidades do fragmento do cDNA foram sequenciadas (Figura 2.10f). Durante a fase de análise de dados, os *reads* passam por uma espécie de filtragem que elimina aqueles de baixa qualidade e outros objetos indesejados, como os adaptadores anteriormente adicionados aos cDNAs, para que então possam ser processados [2].

Ainda na fase de análise de dados de um experimento de RNA-Seq, definimos o **problema da reconstrução de transcriptoma** como o problema de determinar o conjunto T de moléculas de RNA transcritas em uma célula em um dado momento dado o conjunto de *reads* sequenciados. Já o **problema da quantificação de transcriptoma** consiste em estimar a abundância de cada transcrito $t \in T$, dado o conjunto de *reads* sequenciados e o conjunto de transcritos T [12]. Esses são os dois problemas principais abordados nesta dissertação.

[3] Na verdade, os *reads* são gerados a partir das moléculas de cDNA, que foram criadas a partir dos RNAs. Por isso, é comum dizermos que os RNAs são sequenciados e os *reads* são gerados a partir deles.

Figura 2.10: *Geração dos dados em um experimento de RNA-Seq. Figura adaptada de Martin e Wang [2].*



Capítulo 3

O Problema do Particionamento de Similaridade Máxima

Neste capítulo nós definimos o Problema do Particionamento de Similaridade Máxima e como ele pode ser aplicado ao problema da reconstrução e quantificação de transcriptoma. Além disso, serão apresentadas uma prova de que o Problema do Particionamento de Similaridade Máxima é um problema NP-difícil no sentido forte e uma análise sobre o número de soluções viáveis para ele.

3.1 Definindo o Problema do Particionamento de Similaridade Máxima

O **Problema do Particionamento de Similaridade Máxima** é um novo problema de otimização combinatória envolvendo sequências de caracteres e é o principal tema abordado nesta dissertação. Nós o definimos da seguinte forma.

Problema do Particionamento de Similaridade Máxima (PPSM): *dado um conjunto S de segmentos ordenados de uma sequência $\mathcal{S}$ construída sobre um alfabeto Σ, um conjunto T de segmentos ordenados e não-sobrepostos de uma sequência $\mathcal{T}$ também construída sobre Σ, e uma função de pontuação $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$, encontrar uma partição $P = \{s_1, s_2, ..., s_k\}$ de S em k partes, onde cada parte corresponde a uma cadeia de segmentos de S, e k subconjuntos $t_1, t_2, ..., t_k$ de T tal que $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ seja máximo.*

Informalmente, dados os conjuntos de segmentos ordenados S e T das sequências $\mathcal{S}$ e $\mathcal{T}$, respectivamente, o PPSM consiste em encontrar uma partição P de S em k cadeias de segmentos, e k subconjuntos de T tais que a sequência $s_i^\bullet$ correspondente à parte s_i de P seja bastante parecida com a sequência $t_i^\bullet$ correspondente ao subconjunto t_i de T. Mais precisamente, dada uma função de pontuação ω, cada par (s_i, t_i) deve ser tal que o valor de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ seja o maior possível dentre todas as partições e subconjuntos que podem ser construídos a partir de S e T, respectivamente. Um exemplo de instância do PPSM e sua respectiva solução podem ser vistos na Figura 3.1.

O PPSM pode ser utilizado para modelar problemas práticos, como o problema da reconstrução e quantificação de transcriptoma. Veremos na próxima seção como pode ser estabelecida um relação entre eles, de forma que seja possível encontrar uma solução para o problema da reconstrução e quantificação de transcriptoma a partir de soluções de instâncias do PPSM.

Figura 3.1: *Exemplo de instância do PPSM. Dadas as sequências $\mathcal{S}$ e $\mathcal{T}$, e os conjuntos S e T (ordenados), são encontradas duas soluções, sendo uma delas uma solução ótima. Neste exemplo, consideramos a função de pontuação $\omega(\alpha,\alpha)=1$, $\omega(\alpha,\beta)=-1$, $\omega(\alpha,-)=\omega(-,\alpha)=-2$.*

$\mathcal{S}=$ TCACGGTTTCCACGCTATC $\quad\mathcal{T}=$ ACGGTCTTCCAACGCTA

$$S=\left\{\begin{array}{ccc} {}^{1}\text{AC}, & {}^{5}\text{GTTT}, & {}^{10}\text{CGC}, \\ {}^{2}\text{ACG}, & {}^{6}\text{TTTC}, & {}^{11}\text{CTA}, \\ {}^{3}\text{ACGG}, & {}^{7}\text{CCA}, & {}^{12}\text{TA}, \\ {}^{4}\text{GGT}, & {}^{8}\text{CACG}, & {}^{13}\text{TA} \\ & {}^{9}\text{CGC}, & \end{array}\right\} \qquad T=\{\text{ACGGT},\ \text{TTCCA},\ \text{CGCTA}\}$$

Solução viável	**Solução ótima**
$s_1=\{{}^{1}\text{AC}, {}^{5}\text{GTTT}, {}^{10}\text{CGC}\}$	$s_1=\{{}^{2}\text{ACG}, {}^{5}\text{GTTT}, {}^{7}\text{CCA}, {}^{9}\text{CGC}, {}^{12}\text{TA}\}$
$t_1=\{\text{ACGGT}, \text{TTCCA}\}$	$t_1=\{\text{ACGGT}, \text{TTCCA}, \text{CGCTA}\}$
$s_2=\{{}^{2}\text{ACG}, {}^{6}\text{TTTC}, {}^{11}\text{CTA}\}$	$s_2=\{{}^{3}\text{ACGG}, {}^{6}\text{TTTC}, {}^{8}\text{CACG}, {}^{11}\text{CTA}\}$
$t_2=\{\text{ACGGT}, \text{TTCCA}\}$	$t_2=\{\text{ACGGT}, \text{TTCCA}, \text{CGCTA}\}$
$s_3=\{{}^{3}\text{ACGG}, {}^{7}\text{CCA}, {}^{9}\text{CGC}, {}^{12}\text{TA}\}$	$s_3=\{{}^{1}\text{AC}, {}^{4}\text{GGT}, {}^{10}\text{CGC}, {}^{13}\text{TA}\}$
$t_3=\{\text{ACGGT}, \text{TTCCA}, \text{CGCTA}\}$	$t_3=\{\text{ACGGT}, \text{CGCTA}\}$
$s_4=\{{}^{4}\text{GGT}, {}^{8}\text{CACG}, {}^{13}\text{TA}\}$	
$t_4=\{\text{TTCCA}, \text{CGCTA}\}$	
$sim_\omega(s_1^\bullet, t_1^\bullet)=3$	$sim_\omega(s_1^\bullet, t_1^\bullet)=15$
$sim_\omega(s_2^\bullet, t_2^\bullet)=5$	$sim_\omega(s_2^\bullet, t_2^\bullet)=15$
$sim_\omega(s_3^\bullet, t_3^\bullet)=6$	$sim_\omega(s_3^\bullet, t_3^\bullet)=10$
$sim_\omega(s_4^\bullet, t_4^\bullet)=1$	
$\sum_{i=1}^{4} sim_\omega(s_i^\bullet, t_i^\bullet)=15$	$\sum_{i=1}^{3} sim_\omega(s_i^\bullet, t_i^\bullet)=40$

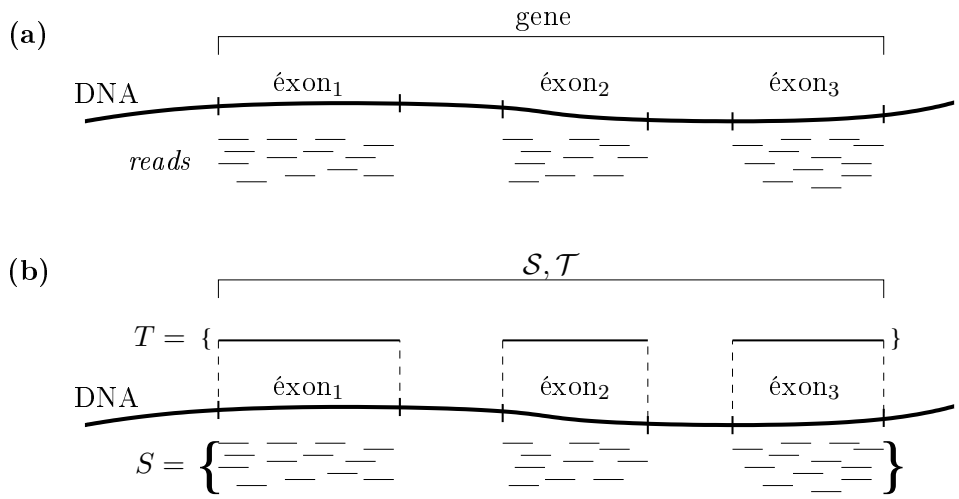
3.2 Relação entre o PPSM e o Problema da Reconstrução e Quantificação de Transcriptoma

O problema da reconstrução e quantificação de transcriptoma é um problema importante da Biologia Computacional que ainda vem sendo estudado por diversos pesquisadores. O resultado desse estudo são vários métodos distintos desenvolvidos na tentativa de se encontrar uma solução adequada para esse problema [12, 24–27]. No entanto, esses métodos nem sempre apresentam boas soluções [28–30], e os métodos mais recentes ainda não foram exaustivamente testados. Sendo assim, propomos como uma aplicação do PPSM o problema da reconstrução e quantificação de transcriptoma como uma maneira alternativa de se encontrar uma solução para ele. Até onde sabemos, nenhum método desenvolvido até agora modela o problema da reconstrução e quantificação de transcriptoma como um problema de otimização combinatória envolvendo sequências, como proposto neste trabalho.

A relação entre o PPSM e o problema da reconstrução e quantificação de transcriptoma encontra-se esquematizado na Figura 3.2. Consideramos o conjunto S como o conjunto dos *reads* gerados

por um experimento de RNA-Seq que se mapeiam em um gene g, o conjunto T como o conjunto dos éxons de g, e ambas as sequências $\mathcal{S}$ e $\mathcal{T}$ como a sequência de nucleotídeos do gene g. Dado um *read* $r \in S$, dizemos que $first(r)$ e $last(r)$ são, respectivamente, as coordenadas de início e de térmico do mapeamento de r em g. A partir disso, estabelecemos a ordem para o conjunto S (veja (sub)Seção 2.2.1). Já a ordem do conjunto T é dada pela ordem com que os éxons aparecem no gene g, da esquerda para a direita.

Figura 3.2: *A partir dos reads mapeados em um gene anotado (a) é obtida uma instância do PPSM (b).*



Sob as circunstâncias estabelecidas, uma solução do PPSM, que consiste de k pares de cadeias (s_i, t_i), corresponde a k transcritos reconstruídos do gene g. Mais precisamente, dizemos que os *reads* em s_i representam o transcrito t_i. Observe que dentre os k pares (s_i, t_i), podem existir alguns que possuem o mesmo subconjunto t_i. Assim sendo, dizemos que cada t_i distinto é uma isoforma do gene g, e o número de pares que compartilham o mesmo t_i é o número de cópias da isoforma representada por t_i, isto é, a quantificação dela. Dessa maneira, podemos reconstruir as isoformas de um gene e quantificá-las

É importante deixar claro que a modelagem descrita exige que se saiba de antemão as coordenadas dos éxons do gene, ou seja, ele deve estar anotado. Apesar disso, o gene não necessariamente deve ser do organismo cujo transcriptoma estamos interessados em reconstruir e quantificar. Ele pode pertencer a um genoma de referência. Em vista disso, tendo todos os genes anotados do organismo de interesse ou do seu genoma de referência, podemos reconstruir e quantificar o transcriptoma inteiro desse organismo aplicando o PPSM para cada um dos seus genes separadamente, encontrando uma possível solução para o problema da reconstrução e quantificação de transcriptoma.

Para verificar se a modelagem apresentada gera bons resultados na prática, é necessário a implementação de um algoritmo que encontre uma solução para o PPSM. Porém, como será demonstrado na próxima seção, o PPSM é um problema difícil de se resolver. Mais precisamente, ele é um problema NP-difícil no sentido forte, ou seja, não existe um algoritmo pseudo-polinomial e, consequentemente, polinomial que o resolve, a menos que P = NP.

3.3 Provando que o PPSM é NP-difícil no Sentido Forte

Vimos na (sub)Seção 2.1.3 que existem, basicamente, duas maneiras de provar que um problema é NP-completo (ou NP-difícil) no sentido forte. Se o problema não é numérico, pode-se utilizar uma redução polinomial de um problema NP-completo. Agora, se o problema é numérico, utiliza-se uma redução pseudo-polinomial de um problema NP-completo no sentido forte a ele.

Com relação ao PPSM, o consideramos como um problema numérico, pois a função de pontuação ω, que faz parte da entrada, pode assumir qualquer valor, ou seja, não existe nenhum limite sobre a pontuação que ela pode atribuir para *matches*, *mismatches* e *spaces*. Portanto, provaremos que o PPSM é NP-difícil no sentido forte através de uma redução pseudo-polinomial.

Primeiramente, visto que o conceito de redução aplica-se somente a problemas de decisão e o PPSM foi apresentado como um problema de otimização, vamos definir uma versão de decisão para ele, que referenciaremos por PPSM_D, cuja definição apresenta-se a seguir.

Problema do Particionamento de Similaridade Máxima - versão de decisão (PPSM_D): *dado um conjunto S de segmentos ordenados de uma sequência $\mathcal{S}$ construída sobre um alfabeto Σ, um conjunto T de segmentos ordenados e não-sobrepostos de uma sequência $\mathcal{T}$ também construída sobre Σ, uma função de pontuação $\omega : \overline{\Sigma} \times \overline{\Sigma} \mapsto \mathbb{R}$ e um número real K, existe uma partição $P = \{s_1, s_2, ..., s_k\}$ de S em k partes, onde cada parte corresponde a uma cadeia de segmentos de S, e k subconjuntos $t_1, t_2, ..., t_k$ de T tal que $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet) \geq K$?*

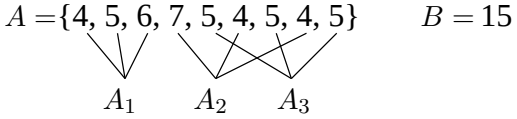
Observe que o PPSM_D corresponde ao PPSM com a inclusão de um limite K, e pergunta-se se existe uma partição de S em k partes $s_1, s_2, ..., s_k$ e k subconjuntos $t_1, t_2, ..., t_k$ de T tal que $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ é maior ou igual a K.

Agora, vamos apresentar o problema escolhido por nós que será reduzido ao PPSM_D. Esse problema é chamado 3-Partition e é definido da seguinte forma.

3-Partition: *dado um multiconjunto $A = \{a_1, a_2, ..., a_{3m}\}$ de $3m$ números inteiros positivos e um número inteiro positivo B tal que $\sum_{i=1}^{3m} a_i = mB$ e $\frac{B}{4} < a_i < \frac{B}{2}$, existe uma partição de A em m subconjuntos $A_1, A_2, ..., A_m$ tal que $\forall i, 1 \leq i \leq m, \sum_{a_j \in A_i} a_j = B$?*

O 3-Partition é um problema de decisão NP-completo no sentido forte [6] que já foi utilizado como problema base em diversas outras reduções [31–35]. Informalmente, ele consiste em decidir se existe uma partição de um multiconjunto numérico A em m partes $A_1, A_2, ..., A_m$, tal que cada parte possui três números e a soma desses três números é igual a um valor B, como demonstrado na Figura 3.3.

Figura 3.3: *Exemplo de instância do 3-Partition. O conjunto A é particionado em três partes A_1, A_2 e A_3 tal que a soma dos número em cada parte é igual a 15.*



A seguir, apresentamos a redução pseudo-polinomial do 3-Partition no PPSM_D como prova do Teorema 3.3.1.

Teorema 3.3.1: *O PPSM_D é NP-completo no sentido forte.*

Demonstração. Primeiramente, vamos mostrar que o PPSM_D está em NP. Considere uma solução $\sigma = (s_1, t_1, s_2, t_2, ..., s_k, t_k)$ do PPSM_D que consiste em uma partição P de S em k partes $s_1, s_2, ..., s_k$ e k subconjuntos $t_1, t_2, ..., t_k$ de T. Verificamos que σ é uma solução que está em $SIM_{\text{PPSM}_\text{D}}$ por meio do cálculo do valor $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ e de sua comparação com K. Para quaisquer s_i e t_i em σ, a concatenação dos seus elementos pode ser feita em tempo $O(|s_i|+|t_i|)$, e o valor de $sim_\omega(s_i^\bullet, t_i^\bullet)$ pode ser calculado em tempo $O(|s_i^\bullet| \cdot |t_i^\bullet|)$ através da aplicação do algoritmo de Needleman-Wunsch. Então, é possível determinar o valor $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ em tempo $O(k|s_i^\bullet| \cdot |t_i^\bullet|)$, uma vez que existem k pares (s_i, t_i). Como $k \leq |S|$, $|s_i^\bullet| \leq |\mathcal{S}|$, $|t_i^\bullet| \leq |\mathcal{T}|$ e a comparação com K é realizada em tempo constante, σ pode ser verificada em tempo $O(|S| \cdot |\mathcal{S}| \cdot |\mathcal{T}|)$, que é polinomial no tamanho da entrada do problema. Portanto, o PPSM_D está em NP.

Vamos agora mostrar como uma instância do PPSM_D pode ser definida a partir de uma instância do 3-Partition. Considerando uma instância $I = (A, B)$ do 3-Partition, onde $A = \{a_1, a_2, ..., a_{3m}\}$ é um conjunto de números inteiros positivos e B é um número inteiro positivo, definimos uma instância $f(I) = (\mathcal{S}, \mathcal{T}, S, T, \omega, K)$ do PPSM_D onde $S = \{a'_1, a'_2, ..., a'_{3m}\}$ e $T = \{B'\}$ são conjuntos de sequências construídas sobre o alfabeto $\Sigma = \{1\}$, tais que:

(i) para todo $a'_i \in S$, $|a'_i| = a_i$, onde $a_i \in A$;

(ii) $|a'_1| \leq |a'_2| \leq ... \leq |a'_{3m}|$;

(iii) para $B' \in T$, $|B'| = B$.

Definimos ainda os valores $K = 0$ e $\omega(\alpha, \beta) = 0$, se $\alpha = \beta$, ou -1, se $\alpha \neq \beta$. Definimos também $\mathcal{S} = a'_1 \bullet a'_2 \bullet ... \bullet a'_{3m}$ e $\mathcal{T} = B'$. Observe que pelo item (ii) acima estabelecemos uma ordem para os segmento em S. Além disso, assumiremos que os segmentos em S não se sobrepõem, uma vez que não existe uma sequência da qual eles se originam. Portanto, qualquer subconjunto de S é uma cadeia de segmentos. Com relação a T, ele é um conjunto com um único segmento, portanto já o consideramos ordenado e não possuindo segmentos sobrepostos. Assim, esses conjuntos estão de acordo com a definição do PPSM_D.

Vamos supor agora que $I = (A, B)$ é uma instância do 3-Partition que está em $SIM_{\text{3-Partition}}$, o que significa que existe uma partição de A em m subconjuntos $A_1, A_2, ..., A_m$ tal que $\sum_{a_j \in A_i} a_j = B$, para $1 \leq i \leq m$. Então, também existe uma partição de S em m partes $S_1, S_2, ..., S_m$ tal que $\sum_{a'_j \in S_i} |a'_j| = |B'|$, onde cada parte S_i de S está relacionada com o subconjunto A_i de A de maneira que $\sum_{a'_j \in S_i} |a'_j| = \sum_{a_j \in A_i} a_j$. Sendo assim, é verdade que $|S_i^\bullet| = |B'|$, para $1 \leq i \leq m$, e um alinhamento ótimo de $S_i^\bullet$ e B' é uma matriz M de dimensões $2 \times |B'|$, onde $M[1][j] = M[2][j] =$ '1', $1 \leq j \leq |B'|$. A pontuação de tal alinhamento será $sim_\omega(S_i^\bullet, B') = |B'| \cdot \omega(\text{'1'}, \text{'1'}) = 0$. Consequentemente, $\sum_{i=1}^{m} sim(S_i^\bullet, B') = 0$. Definimos anteriormente $K = 0$, então temos que $\sum_{i=1}^{m} sim(S_i^\bullet, B') \geq K$, ou seja, $f(I)$ é uma instância do PPSM_D que está em $SIM_{\text{PPSM}_\text{D}}$.

Agora, partiremos da suposição de que $f(I) = (\mathcal{S}, \mathcal{T}, S, T, \omega, K)$ é uma instância do PPSM_D que está em $SIM_{\text{PPSM}_\text{D}}$. Isso significa que existe uma partição de S em k partes $S_1, S_2, ..., S_k$ e k subconjuntos $T_1, T_2, ..., T_k$ de T tal que $\sum_{i=1}^{k} sim(S_i^\bullet, T_i^\bullet) \geq K$. É possível então obter o conjunto A e o valor B de uma instância I do 3-Partition contando os caracteres '1' em $S = \{S_1 \cup S_2 \cup ... \cup S_k\}$ e em T. Considerando que $\sum_{a'_j \in S_i} |a'_j| = |B'|$ e $\sum_{a_j \in A_i} a_j = B$, I é uma instância do 3-Partition que está em $SIM_{\text{3-Partition}}$. Ou seja, $f(I)$ é uma instância que está em $SIM_{\text{PPSM}_\text{D}}$ se, e somente se, I está em $SIM_{\text{3-Partition}}$. Dessa forma, a primeira condição da redução pseudo-polinomial está satisfeita.

O Algoritmo 1 apresenta os passos para se determinar $f(I) = (\mathcal{S}, \mathcal{T}, S, T, \omega, K)$ a partir de $I = (A, B)$. Os valores de K e ω podem ser definidos em tempo constante. A ordenação do conjunto A na linha 8 pode ser realizada em tempo $O(3m \log 3m) = O(m \log m)$. Já o tempo gasto para definir $\mathcal{S}$, $\mathcal{T}$, S e T depende da magnitude dos números em A e do valor B. Uma maneira simples de definir essas sequências e conjuntos é criar uma sequência de caracteres '1' para o valor B e para cada número em A, como apresentado nas linhas 5 e 10, respectivamente. Assumindo que cada caractere '1' possa ser criado em tempo constante, e que uma sequência a' de caracteres '1' possa ser criada em tempo $O(|a'|)$, a sequência $\mathcal{T}$ e o conjunto T podem ser definidos em tempo $O(B) = O(\text{Max}(I))$, e a sequência $\mathcal{S}$ e o conjunto S em tempo $O(\text{Max}(I) \cdot 3m) = O(\text{Max}(I) \cdot m)$. Assumindo também que é necessário pelo menos um símbolo para representar um número, podemos afirmar que $|A| \leq \text{Length}(I)$. Como $m \leq |A|$, é verdade que $m \leq \text{Length}(I)$. Agora já podemos determinar o tempo necessário para definir $f(I)$, que é:

$$O(m \log m + \text{Max}(I) + \text{Max}(I) \cdot m) = \\ O(\text{Length}(I) \log \text{Length}(I) + \text{Max}(I) \cdot \text{Length}(I)). \tag{3.1}$$

Ou seja, $f(I)$ é determinada em tempo pseudo-polinomial. Com isso, a segunda condição da redução pseudo-polinomial está satisfeita.

Algoritmo 1: Determina $f(I)$.

Entrada: Instância $I = (A, B)$ do 3-Partition.
Saída: Instância $f(I) = (\mathcal{S}, \mathcal{T}, S, T, \omega, K)$ do PPSM_D.

1 $K \leftarrow 0$;
2 defina $\omega(\alpha, \alpha) = 0; \omega(\alpha, \beta) = \omega(\alpha, -) = \omega(-, \alpha) = -1$;
3 $S \leftarrow \emptyset$;
4 $\mathcal{S} \leftarrow \epsilon$;
5 crie uma sequência B' de B caracteres '1';
6 $T \leftarrow B'$;
7 $\mathcal{T} \leftarrow T$;
8 ordene o conjunto A;
9 **para** $i \leftarrow 1$ **até** $3m$ **faça**
10 | crie uma sequência a'_i de $a_i \in A$ caracteres '1';
11 | $S \leftarrow S \cup a'_i$;
12 | $\mathcal{S} \leftarrow \mathcal{S} \bullet a'_i$;
13 **fim**
14 **devolver** ($\mathcal{S}$, $\mathcal{T}$, S, T, ω e K);

É fácil verificar que o valor de $\text{Length}(f(I))$ será maior que $\text{Length}(I)$, pois criamos sequências de caracteres '1' para todos os números em I de tamanho igual a magnitude do respectivo número. Além disso, também são criadas as sequências $\mathcal{S}$ e $\mathcal{T}$ cujos tamanho somam $mB + B$. Portanto, podemos afirmar que $\text{Length}(f(I)) \geq \text{Length}(I)$. Essa desigualdade garante a terceira condição da redução pseudo-polinomial.

A instância $f(I) = (\mathcal{S}, \mathcal{T}, S, T, \omega, K)$ do PPSM_D é formada pelas sequências $\mathcal{S}$ e $\mathcal{T}$, pelos conjuntos S e T, pela função de pontuação ω, que pode ser 0 ou -1, e pelo número $K = 0$. Como $\mathcal{S}$ e $\mathcal{T}$ são sequências de caracteres, e S e T são conjuntos de sequências de caracteres, eles não são considerados para determinar o valor de $\text{Max}(f(I))$. Dessa forma, $\text{Max}(f(I)) = |-1| = 1$. Ou seja,

a quarta e última condição da redução pseudo-polinomial é satisfeita.

Como o PPSM_D está em NP e existe uma redução pseudo-polinomial do 3-Partition nele, o PPSM_D é um problema NP-completo no sentido forte. □

A prova de que PPSM_D é NP-completo no sentido forte significa que, além de não existir um algoritmo pseudo-polinomial que o resolve, a menos que P = NP, ele é tão difícil quanto qualquer outro problema em NP. Esse resultado, em conjunto com o fato de que o PPSM_D não é mais difícil que a versão de otimização dele, constitui a prova do Teorema 3.3.2.

Teorema 3.3.2: *O PPSM é NP-difícil no sentido forte.*

Demonstração. A prova é uma implicação direta do Teorema 3.3.1 e do fato de que o PPSM_D não é mais difícil que o PPSM. □

Perceba que não podemos classificar o PPSM como um problema NP-completo no sentido forte, pois não sabemos se ele está em NP. Por isso, nos limitamos em dizer que ele é um problema NP-difícil no sentido forte.

3.4 Número de Soluções Viáveis do PPSM

Mesmo que um problema seja NP-difícil ou NP-difícil no sentido forte, um algoritmo que verifique todas as possíveis soluções e retorne a ótima ainda pode ser desenvolvido para resolver pequenas instâncias do problema. Nesta seção, vamos mostrar que, mesmo para instâncias consideradas pequenas, o número de soluções viáveis do PPSM é tão grande que torna a utilização de tal algoritmo praticamente impossível.

Primeiramente, dizemos que $\sigma = (s_1, t_1, s_2, t_2, ..., s_k, t_k)$ é uma solução válida, ou viável, do PPSM se:

1. $s_1, s_2, ..., s_k$ são subconjuntos de uma partição de S em k partes;
2. cada subconjunto s_i, para $1 \leq i \leq k$, é uma cadeia de segmentos de S;
3. $t_1, t_2, ..., t_k$ são subconjuntos de T.

Observe que o número de soluções válidas do PPSM está diretamente relacionado com o número de maneiras distintas de particionar S, que, por sua vez, depende da sobreposição dos segmentos em S. Por exemplo, considere os segmentos $u, v \in S$ e uma partição $P = \{s_1, s_2, ..., s_k\}$ de S tal que $u, v \in s_i$, para algum s_i em P. Se u se sobrepõe a v, então P é um particionamento de S que gera uma solução inválida do PPSM, pois s_i não é uma cadeia de segmentos de S. No entanto, se assumirmos que nenhum segmento em S se sobrepõe a outro, a condição 2 acima se torna irrelevante, uma vez que qualquer partição de S gera subconjuntos que são cadeias de segmentos. Sob essa condição, que corresponde ao *pior caso*, iremos encontrar o número de soluções distintas do PPSM em função do tamanho da entrada.

Considerando que o conjunto S não seja vazio, uma partição qualquer de S inclui pelo menos um subconjunto, com todos os elementos de S, e no máximo $n = |S|$ subconjuntos, cada um contendo um único elemento de S. Sendo assim, podemos determinar o número de partições distintas de S com o seguinte somatório:

$$\sum_{i=1}^{n} S(n,i), \tag{3.2}$$

onde $n = |S|$ e $S(n,i)$ é o número de maneiras distintas de particionar S em i subconjuntos não vazios. O valor de $S(n,i)$ corresponde aos *números de Stirling de segunda espécie* [36, 37], calculados da seguinte forma:

$$S(n,i) = \frac{1}{i!}\sum_{k=0}^{i}(-1)^{i-k}\binom{i}{k}k^n. \tag{3.3}$$

Utilizando a Equação 3.3, onde $\binom{i}{k}$ é um coeficiente binomial, podemos calcular o somatório em 3.2 e encontrar o número de maneiras distintas de particionar S. Entretanto, ainda temos que considerar o possível subconjunto t_i de T associado a cada parte s_i de uma partição de S. Sabe-se que, incluindo o subconjunto vazio, existem $2^{|T|}$ subconjuntos distintos de T. Então, se $S(n,i)$ representa o número de partições distintas de S em i subconjuntos, e cada subconjunto está associado a um dentre os $2^{|T|}$ subconjuntos de T, o valor de $S(n,i) \cdot i2^{|T|}$ representa o número de possíveis soluções do PPSM quando S é particionado em i subconjuntos. A partir disso, podemos determinar o número de possíveis soluções do PPSM com o seguinte somatório:

$$\sum_{i=1}^{n} S(n,i) \cdot i2^m, \tag{3.4}$$

onde $n = |S|$ e $m = |T|$.

Para demonstrar o quão rápido o valor do somatório 3.4 cresce em função do tamanho de n e m, ele foi utilizado para preencher a Tabela 3.1. Observe que o número de soluções distintas do PPSM cresce de forma exponencial à medida que $|S|$ e $|T|$ aumentam, ultrapassando 36 milhões quando $|S| = 10$ e $|T| = 6$.

Tabela 3.1: *Número de soluções distintas do PPSM, no pior caso, onde $n = |S|$ e $m = |T|$.*

n \ m	3	4	5	6
1	8	16	32	64
2	24	48	96	192
3	80	160	320	640
4	296	592	1184	2368
5	1208	2416	4832	9664
6	5392	10784	21568	43136
7	26104	52208	104416	208832
8	136056	272112	544224	1088448
9	758624	1517248	3034496	6068992
10	4500760	9001520	18003040	36006080

Nos testes apresentados mais adiante, veremos que o tamanho de S pode facilmente ultrapassar os milhares. Por isso, como será discutido no próximo capítulo, a utilização de heurísticas para encontrar uma solução que pode ser a ótima é uma boa alternativa, visto o número extremamente grande de soluções do PPSM.

Capítulo 4

Heurísticas para o PPSM

O fato do PPSM ser um problema NP-difícil no sentido forte praticamente impossibilita o desenvolvimento de um algoritmo eficiente que o resolva. Além disso, vimos que o número de possíveis soluções do PPSM cresce de forma exponencial à medida que a quantidade de segmentos em S e em T aumenta. Por isso, é completamente inviável a aplicação de um algoritmo força bruta que verifique todas as possíveis soluções e devolva a ótima. Em vista disso, uma possível abordagem é o desenvolvimento de heurísticas para encontrar boas soluções para o problema. Neste capítulo serão então apresentadas duas heurísticas desenvolvidas por nós que encontram soluções viáveis para o PPSM em tempo polinomial.

4.1 Heurística PASG

Uma das heurísticas para o PPSM foi desenvolvida com base na solução de um outro problema de otimização combinatória envolvendo cadeias de segmentos denominado **Problema do Alinhamento de Segmentos**. Esse problema foi proposto e estudado detalhadamente por de Lima [13], e encontra-se definido a seguir.

Problema do Alinhamento de Segmentos (PASG): *dados um alfabeto Σ, tal que $'-' \notin \Sigma$, duas sequências s_1 e s_2 sobre Σ de tamanho n e m respectivamente, um conjunto ordenado $B = \{b_1, b_2, ..., b_u\}$ de segmentos de s_1, um conjunto ordenado $C = \{c_1, c_2, ..., c_v\}$ de segmentos de s_2, e uma função de pontuação ω sobre $\overline{\Sigma}$, determinar uma cadeia $\Gamma_B = \{b_p, b_q, ..., b_r\}$ de segmentos de B e uma cadeia $\Gamma_C = \{c_w, c_x, ..., c_y\}$ de segmentos de C, tais que $sim_\omega(\Gamma_B^\bullet, \Gamma_C^\bullet)$ é máxima entre todas as cadeias de segmentos de B e de C.*

Informalmente, o PASG consiste em encontrar um conjunto de segmentos ordenados não sobrepostos de B e um conjunto de segmentos ordenados também não sobrepostos de C tais que as sequências resultantes da concatenação dos elementos desses conjuntos sejam as mais parecidas possível. Mais especificamente, dada uma função de pontuação ω, deseja-se encontrar duas cadeias de segmentos Γ_B e Γ_C de B e de C, respectivamente, tal que o valor de $sim_\omega(\Gamma_B^\bullet, \Gamma_C^\bullet)$ é o maior possível. A Figura 4.1 apresenta um exemplo de instância do PASG.

O PASG pode ser resolvido de forma eficiente por um algoritmo polinomial [13]. Tal algoritmo, que chamaremos de SolPASG, é baseado na técnica de programação dinâmica e resolve o PASG em tempo:

$$O(u^2 \cdot v^2 + u \cdot v \cdot bMax \cdot cMax + u^2 \cdot v \cdot cMax + u \cdot v^2 \cdot bMax), \tag{4.1}$$

Figura 4.1: *Exemplo de instância do PASG cuja solução ótima corresponde às cadeias de segmentos Γ_B e Γ_C. Considerando a função de pontuação $\omega(\alpha,\alpha) = 1$, $\omega(\alpha,\beta) = -1$, $\omega(\alpha,-) = \omega(-,\alpha) = -2$, o valor de $sim_\omega(\Gamma_B^\bullet, \Gamma_C^\bullet)$ é igual a 10.*

$$s_1 = \text{AACCCATCTCCACACCAGAGAT} \qquad s_2 = \text{ATCCCGTCTCCACACTAGAGAT}$$

$$B = \left\{ \begin{matrix} {}^{1}\text{AAC}, {}^{4}\text{CAT}, {}^{5}\text{TCC}, {}^{8}\text{CAC}, {}^{9}\text{AGA}, \\ {}^{2}\text{AACC}, \quad {}^{6}\text{CCACA}, \quad {}^{10}\text{GAT} \\ {}^{3}\text{CCA}, \quad {}^{7}\text{ACAC}, \end{matrix} \right\} \quad C = \left\{ \begin{matrix} {}^{1}\text{ATCC}, \quad {}^{5}\text{TCC}, \quad {}^{7}\text{ACTA}, \\ {}^{2}\text{TC}, {}^{3}\text{CGTC}, {}^{6}\text{CCAC}, {}^{8}\text{TAGA}, \\ {}^{4}\text{GTC}, \quad {}^{9}\text{AGAT} \end{matrix} \right\}$$

$$\Gamma_B = \{{}^{2}\text{AACC}, {}^{6}\text{CCACA}, {}^{10}\text{GAT}\}$$
$$\Gamma_C = \{{}^{1}\text{ATCC}, {}^{6}\text{CCAC}, {}^{9}\text{AGAT}\}$$

onde u e v correspondem à quantidade de elementos nos conjuntos B e C, respectivamente, e $bMax = max\{|b| : b \in B\}$ e $cMax = max\{|c| : c \in C\}$.

A heurística que desenvolvemos baseada no PASG para o PPSM, que denominaremos simplesmente por heurística PASG, encontra-se no Algoritmo 2. Ela consiste em fornecer como entrada ao SolPASG os conjuntos S e T e as sequências $\mathcal{S}$ e $\mathcal{T}$ de forma que ele encontre uma cadeia de segmentos Γ_S de S e outra cadeia de segmentos Γ_T de T tais que $sim_\omega(\Gamma_S^\bullet, \Gamma_T^\bullet)$ é máxima. Definimos então Γ_S como uma parte s_i da partição P de S e Γ_T como um subconjunto t_i de T. Ou seja, Γ_S e Γ_T são partes de uma solução para o PPSM. Em seguida, retiramos de S os elementos em s_i e o processo é repetido até S tornar-se vazio. Mais precisamente, em cada iteração i, para $1 \leq i \leq k$, SolPASG toma como entrada os conjuntos $S' = S \setminus \{s_{i-1} \cup s_{i-2} \cup ... \cup s_1\}$ e T, as sequências $\mathcal{S}$ e $\mathcal{T}$, e devolve a parte s_i de P e o subconjunto t_i de T. Ao final da k-ésima iteração, quando $S' = \emptyset$, estarão definidos uma partição P de S em k partes e k subconjuntos de T, constituindo uma solução para o PPSM. Observe que em cada iteração são encontradas as cadeias de segmentos Γ_S e Γ_T tais que $sim_\omega(\Gamma_S^\bullet, \Gamma_T^\bullet)$ é máxima, sem se preocupar com as cadeias de segmentos que serão encontradas futuramente e que farão parte da solução. Esse é um aspecto característico dos algoritmos gulosos, e por isso essa heurística corresponde a uma heurística gulosa.

Algoritmo 2: Heurística PASG.

Entrada: as sequências $\mathcal{S}$ e $\mathcal{T}$, os conjuntos S e T e uma função de pontuação ω.
Saída: partição P de S em k partes $s_1, s_2, ..., s_k$ e k subconjuntos $t_1, t_2, ..., t_k$ de T.

```
1 S' ← S;
2 k ← 1;
3 enquanto S' ≠ ∅ faça
4 |   (s_k, t_k) ← SolPASG(S', T, 𝒮, 𝒯, ω);
5 |   S' ← S' \ s_k;
6 |   k ← k + 1;
7 fim
8 devolver (s_1, t_1), (s_2, t_2), ..., (s_k, t_k);
```

A respeito da complexidade de tempo da heurística PASG, ela será dominada pelas linhas 4, 5, e 6 dentro do laço **enquanto** do Algoritmo 2, executado no máximo $|S|$ vezes. No intuito de definir uma complexidade de tempo mais sucinta, assumiremos que $|S| \geq max\{|s| : s \in S\}$, isto é, a quantidade de segmentos em S é maior ou igual ao tamanho do maior segmento em S. Com isso, a linha 4 é executada em tempo $O(|S|^2 \cdot |T|^2 + |S|^2 \cdot |T| \cdot tMax)$, onde $tMax = max\{|t| : t \in T\}$,

já que essa é a complexidade do SolPASG sob a nossa condição. A operação da linha 5 consome tempo $O(|S|)$, enquanto que a linha 6 é executada em tempo constante. Portanto, o Algoritmo 2 possui complexidade:

$$\begin{gathered} O(|S| \cdot (|S|^2 \cdot |T|^2 + |S|^2 \cdot |T| \cdot tMax + |S| + 1)) = \\ O(|S|^3 \cdot |T|^2 + |S|^3 \cdot |T| \cdot tMax + |S|^2 + |S|) = \\ O(|S|^3 \cdot |T|^2 + |S|^3 \cdot |T| \cdot tMax). \end{gathered} \tag{4.2}$$

Além da heurística PASG, nós desenvolvemos uma segunda heurística, detalhada na próxima seção, que encontra uma solução para o PPSM com base na busca por caminhos em um grafo.

4.2 Heurística DAG

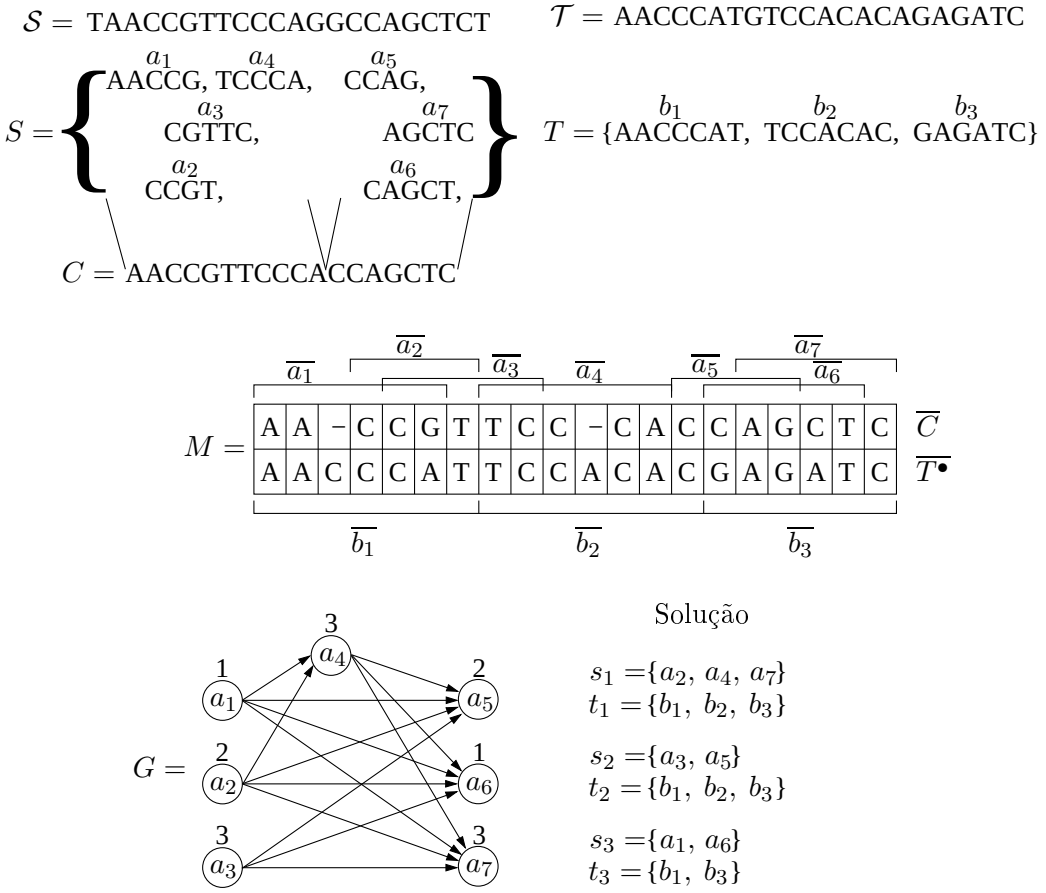
A segunda heurística do PPSM, que chamaremos de heurística DAG, o aborda de uma maneira diferente da heurística PASG. Mais precisamente, ela é baseada na busca por caminhos em um grafo dirigido acíclico, por isso o nome DAG (do inglês, *Directed Acyclic Graph*).

A heurística DAG consiste em representar os segmentos $a_1, a_2, ..., a_i$ em S como vértices $v_1, v_2, ..., v_i$ de um grafo dirigido acíclico $G = (V, A)$ com peso nos vértices. Nesse grafo, existe uma aresta dirigida $e = (v_k, v_l)$ entre dois vértices v_k e v_l se $a_k \prec a_l$. O peso w_k de cada vértice v_k de G é calculado da seguinte forma: primeiramente, uma sequência consenso C é construída a partir da sobreposição dos segmentos de S. Em seguida, é construído um alinhamento ótimo M de C e $T^\bullet$, gerando as sequências $\overline{C}$ e $\overline{T^\bullet}$. Observe que cada segmento a_k em S corresponde a um segmento tanto de C quanto de $\overline{C}$, sendo que nessa última ele pode aparecer intercalado com espaços. Vamos então denotar por $\overline{a_k}$ o segmento $a_k \in S$ de $\overline{C}$ e por $\overline{b_k}$ o segmento $b_k \in T$ de $\overline{T^\bullet}$. Agora, considere $u = first(\overline{a_k})$ e $v = last(\overline{a_k})$. O peso w_k de um vértice v_k de G corresponde ao valor $\sum_{i=u}^{v} \omega(M[1][i], M[2][i])$. Dessa forma, quanto maior o valor do peso w_k, mais parecido será o segmento $\overline{a_k}$ com um segmento de $\overline{T^\bullet}$.

Com o grafo G definido, a heurística DAG encontra um caminho de maior peso $R = \{v_i, v_j, ..., v_k\}$ em G. Repare que os vértices em R representam uma cadeia de segmentos $\Gamma_S = \{a_i, a_j, ..., a_k\}$ de S tal que a sequência $\Gamma_S^\bullet = \overline{a_i} \bullet \overline{a_j} \bullet ... \bullet \overline{a_k}$ é a mais parecida possível com $\overline{T^\bullet}$. Por isso, julgamos que $\Gamma_S^\bullet$ será parecida com $T^\bullet$, embora isso nem sempre é verdade. Assim, a partir de $R = \{v_i, v_j, ..., v_k\}$, definimos uma parte $s_i = \{a_i, a_j, ..., a_k\}$ da partição P de S sendo buscada. A respeito do subconjunto t_i associado a s_i, ele é construído da seguinte maneira: um segmento $b_k \in T$ estará em t_i se existe sobreposição entre $\overline{b_k}$ e algum $\overline{a_k} \in R$. O subconjunto t_i é assim escolhido porque se existe tal sobreposição, significa que o segmento b_k contribuiu positivamente para o valor do peso de pelo menos um segmento em R. Assim, julgamos que ele também contribuirá positivamente para o valor de $sim_\omega(s_i^\bullet, t_i^\bullet)$. Com isso, uma parte s_i de S e um subconjunto t_i de T estão definidos. Em seguida, os vértices que compõem o caminho R são removidos de G e um novo caminho R é buscado nesse grafo para então ser definida uma nova parte s_{i+1} de S e um novo subconjunto t_{i+1} de T. Ao final, quando não restar mais nenhum vértice em V, estarão definidos uma partição P de S em k partes e k subconjuntos de T, constituindo uma solução do PPSM. Os passos descritos que compreendem a heurística DAG podem ser visualizados na Figura 4.2.

De maneira semelhante à heurística PASG, a heurística DAG também é baseada na estratégia

Figura 4.2: *Esquema da heurística DAG. Um alinhamento ótimo M é produzido a partir da sequência consenso C e $T^\bullet$. Considerando a função de pontuação $\omega(\alpha,\alpha)=1$, $\omega(\alpha,\beta)=-1$, $\omega(\alpha,-)=\omega(-,\alpha)=-2$, o peso de cada vértice a_i do grafo G é igual a soma da pontuação das colunas no intervalo $first(\overline{a_i})..last(\overline{a_i})$ em M. O subconjunto s_1 na solução é determinado pelo caminho $R=\{a_2,a_3,a_4\}$ de peso 8; o subconjunto s_2 pelo caminho $R=\{a_3,a_5\}$ de peso 5 e o subconjunto s_3 pelo caminho $R=\{a_1,a_6\}$ de peso 2.*



gulosa. Isso pode ser observado considerando que, a cada iteração, é buscado um caminho de maior peso em G, ou seja, buscamos uma cadeia de segmentos de S que julgamos ser muito parecida com $T^\bullet$, sem se preocupar com as cadeias de segmentos que serão definidos futuramente e que farão parte da solução.

A heurística DAG encontra-se esquematizada no Algoritmo 3, onde a função Consensus() determina a sequência consenso C de S, Alignment() produz um alinhamento ótimo de duas sequências, LongestPath() encontra um caminho de maior peso R em G, Segments() devolve os segmentos representados pelos vértices em R e FindTi() encontra o subconjunto t_i de T como descrito anteriormente.

Com o propósito de se determinar a complexidade de tempo do Algoritmo 3, vamos apresentar algumas relações que envolvem a heurística DAG. Como cada vértice de G corresponde a um segmento de S, é verdade que $|S| = |V|$. Apesar da quantidade de arestas de G depender da sobreposição dos segmentos de S, ainda podemos determinar o valor máximo de $|A|$. De maneira ordenada, o primeiro segmento a_1 de S precede no máximo $|S|-1$ segmentos, o segundo segmento a_2 precede no máximo $|S|-2$ segmentos, e assim por diante até alcançarmos o penúltimo segmento, que precede no máximo um segmento. Ou seja, a quantidade máxima de arestas em G será $\sum_{i=1}^{|S|-1} i = \frac{(|S|-1)\cdot|S|}{2} = \frac{|S|^2-|S|}{2}$. Assim, temos que $|A| = O(|S|^2)$. A respeito da sequência consenso C, podemos estabelecer que $|C| \leq |S^\bullet|$. Agora, vamos determinar a complexidade da heurística DAG como

Algoritmo 3: Heurística DAG.

Entrada: sequências $\mathcal{S}$ e $\mathcal{T}$, os conjuntos S e T e uma função de pontuação ω.
Saída: partição P de S em k partes $s_1, s_2, ..., s_k$ e k subconjuntos de T $t_1, t_2, ..., t_k$.

```
1  k ← 1;
2  C ← Consensus(S);
3  A ← Alignment(C, T•, ω);
4  Construa G = (V, A);
5  enquanto V ≠ ∅ faça
6  |  R ← LongestPath(G);
7  |  s_k ← Segments(R);
8  |  t_k ← FindTi(s_k, A);
9  |  V ← V \ R;
10 |  k ← k + 1;
11 fim
12 devolver (s_1, t_1), (s_2, t_2), ..., (s_k, t_k);
```

segue.

A sequência consenso na linha 2 do Algoritmo 3 pode ser determinada em tempo $O(|S^\bullet|)$. O alinhamento na linha 3 é produzido em tempo $O(|C| \cdot |T^\bullet|)$ por meio do algoritmo de Needleman-Wunsch, e o grafo G é construído na linha 4 em tempo $O(|V| + |A|)$. Apesar do problema de encontrar um caminho de peso máximo R em um grafo qualquer G ser NP-completo [3], R pode ser encontrado em tempo $O(|V|+|A|)$ se G for um grafo dirigido acíclico [38]. Esse é o tempo necessário para a linha 6 ser executada. Na linha 7, a parte s_i que contém os segmentos representados pelo caminho R pode ser definida em tempo $O(|V|)$ e o subconjunto t_i na linha 8 pode ser construído em tempo $O(|T^\bullet|)$. Já a operação na linha 9 consome tempo $O(|V| + |A|)$ e as linhas 1 e 10 são executadas em tempo constante. Como as linhas 6, 7, 8, 9 e 10 são executadas no máximo $|S|$ vezes, a complexidade de tempo do Algoritmo 3 pode ser escrita como:

$$O(|S^\bullet| + |C| \cdot |T^\bullet| + |V| + |A| + |S| \cdot (|V| + |A| + |V| + |T^\bullet| + |V| + |A| + 1)). \tag{4.3}$$

Com base nas relações estabelecidas anteriormente, reescrevemos a Equação 4.3 como:

$$\begin{aligned} &O(|S^\bullet| + |S^\bullet| \cdot |T^\bullet| + |S| + |S|^2 + |S| \cdot (|S| + |S|^2 + |S| + |T^\bullet| + |S| + |S|^2 + 1)) = \\ &O(|S^\bullet| + |S^\bullet| \cdot |T^\bullet| + |S| + |S|^2 + |S|^2 + |S|^3 + |S|^2 + |S| \cdot |T^\bullet| + |S|^2 + |S|^3 + |S|)) = \\ &O(|S^\bullet| \cdot |T^\bullet| + |S|^3). \end{aligned} \tag{4.4}$$

4.3 Refinando a Complexidade das Heurísticas

Até agora, a análise da complexidade de tempo das heurísticas foi realizada de forma genérica. No entanto, tratando-se do problema da reconstrução e quantificação de transcriptoma, podemos estimar alguns valores da entrada do PPSM. Na Seção 4.1, assumimos que $|S| \geq max\{|s| : s \in S\}$. Tal suposição vem do fato de que podemos estimar o tamanho dos segmentos em S, pois eles representam *reads* gerados por RNA-Seq, cujo tamanho varia de 30 a 450 nucleotídeos [22]. Isso nos permite dizer que $|S^\bullet| = O(|S|)$. Além disso, podemos assumir também que o número de éxons de um gene é limitado superiormente pelo tamanho do maior éxon dele, ou seja, $|T| = O(tMax)$,

onde $tMax = max\{|t| : t \in T\}$[1]. Outra relação que também podemos estabelecer é que a soma do tamanho de todos os *reads* mapeados em um gene será maior que a soma do tamanho dos éxons desse mesmo gene, isto é, $|T^\bullet| = O(|S^\bullet|)$. A partir dessas suposições, podemos dizer que $|T^\bullet| = O(|S|)$. Em outras palavras, a soma do tamanho dos éxons de um gene é limitada superiormente pelo número de *reads* mapeados nesse mesmo gene. Agora, podemos refinar ainda mais a complexidade da heurística PASG:

$$\begin{gathered} O(|S|^3 \cdot |T|^2 + |S|^3 \cdot |T| \cdot tMax) = \\ O(|S|^3 \cdot tMax^2 + |S|^3 \cdot tMax \cdot tMax) = \\ O(|S|^3 \cdot tMax^2), \end{gathered} \tag{4.5}$$

e da heurística DAG:

$$\begin{gathered} O(|S^\bullet| \cdot |T^\bullet| + |S|^3) = \\ O(|S| \cdot |S| + |S|^3) = \\ O(|S|^3). \end{gathered} \tag{4.6}$$

Apesar da heurística PASG ter complexidade maior do que a heurística DAG, ambas são algoritmos cúbicos (grau do polinômio que representa a complexidade de tempo é 3). Porém, como será visto no próximo Capítulo, elas apresentam resultados distintos e possuem tempo de execução bem diferentes.

[1] Essa relação baseia-se no fato de que, geralmente, um gene não possui muitos éxons. No genoma humano (versão GRCh38.p2), por exemplo, o gene TTN é aquele com o maior número de éxons, 363.

Capítulo 5

Testes Práticos

No intuito de avaliar as heurísticas quanto à exatidão dos seus resultados e ao seu desempenho, ambas foram implementadas e testadas. Os testes compreendem instâncias artificiais do PPSM, e instâncias semi-artificiais e reais do problema da reconstrução e quantificação de transcriptoma. Antes de apresentarmos os resultados dos testes, vamos apontar alguns detalhes das implementações e as condições nas quais as heurísticas foram executadas.

5.1 Detalhes das Implementações

A implementação da heurística PASG faz uso da solução do PASG (SolPASG) implementada por de Lima [13] na linguagem de programação C. Um *shell script* simples foi escrito que invoca o SolPASG passando como parâmetro os conjuntos S e T e as sequências $\mathcal{S}$ e $\mathcal{T}$. O *script* define então as duas cadeias de segmentos Γ_S de S e Γ_T de T retornadas pelo SolPASG como uma parte da partição P de S e um subconjunto de T, respectivamente, e retira de S os elementos em Γ_S. Os passos descritos estão contidos em uma estrutura de repetição e são executados enquanto S não for vazio. A cada iteração, o valor de $sim_\omega(s_i^\bullet, t_i^\bullet)$ retornado pelo SolPASG é armazenado em um arquivo de saída, junto com o par (s_i, t_i) correspondente, sendo que a primeira informação é utilizada para o cálculo do valor de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$.

A heurística DAG foi implementada na linguagem de programação C e divide-se em quatro passos. O primeiro passo compreende a definição da sequência consenso C, a construção do alinhamento ótimo de C com $T^\bullet$ e a construção do grafo dirigido acíclico com pesos nos vértices $G = (V, A)$, grafo esse representado como uma matriz de adjacência. O segundo passo consiste na busca dos caminhos de maior peso em G até esgotar os seus vértices. A partir dos caminhos encontrados no passo dois, o terceiro passo define a partição P de S em k partes s_1, s_2, ..., s_k e define os k subconjuntos t_1, t_2, ..., t_k de T. O quarto e último passo constrói um alinhamento ótimo de $s_i^\bullet$ e $t_i^\bullet$, e determina o valor de $sim_\omega(s_i^\bullet, t_i^\bullet)$ para cada par (s_i, t_i). Os alinhamentos mencionados são construídos através do algoritmo Needleman-Wuncsh implementado no pacote EMBOSS [39]. O par (s_i, t_i), o alinhamento ótimo de $s_i^\bullet$ e $t_i^\bullet$, para $1 \leq i \leq k$, e o valor de $sim_\omega(s_i^\bullet, t_i^\bullet)$, utilizado no cálculo do valor de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$, ficam armazenados em um arquivo de saída.

Todos os testes detalhados neste capítulo foram realizados utilizando uma função de pontuação que atribui valor $+1$ para *matches*, -1 para *mismatches* e -2 para *spaces*. Com relação ao *hardware* utilizado, os testes foram executados em um computador com 16Gb de memória RAM e processador Intel® Core™ i7-4790 de 3.60GHz com 4 núcleos físicos (8 núcleos lógicos), utilizando o sistema

operacional Linux Ubuntu 14.04 LTS de 64 bits.

5.2 Testes Artificiais

Nos testes artificiais estamos interessados em verificar qual das heurísticas apresenta melhores resultados baseando-se no valor de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$, calculado por cada uma delas a partir de instâncias do PPSM.

Antes de apresentarmos os resultados, vamos descrever como os testes foram elaborados. Ao total, criamos 90 testes artificiais, sendo cada um construído a partir de duas sequências de nucleotídeos $\mathcal{S}$ e $\mathcal{T} = \mathcal{S}$. Para cada instância, a partir da sequência $\mathcal{T}$ correspondente foi gerado um conjunto ordenado de segmentos não sobrepostos T, contendo de 2 a 14 elementos, com tamanhos que variam de 62 a 850 nucleotídeos[1]. A respeito do conjunto S, nós o criamos com base no conjunto T da seguinte forma: considerando um conjunto $T = \{b_1, b_2, ..., b_n\}$, atribuímos um valor aleatório c_i entre 5 e 40 para cada $b_i \in T$. Então, criamos pequenos segmentos de c_i cópias do segmento $b_i \in T$, e os incluímos em S, como mostrado na Figura 5.1. Perceba que, embora criado a partir de T, S não deixa de ser um conjunto de segmentos de $\mathcal{S}$, afinal, $\mathcal{S} = \mathcal{T}$. Com isso, criamos um conjunto S para cada teste, contendo de 248 a 3000 segmentos, cujos tamanho variam de 5 a 92 nucleotídeos.

Figura 5.1: *Exemplo de criação dos testes artificiais. Dada a sequência $\mathcal{S} = \mathcal{T}$, o conjunto $T = \{b_1, b_2, b_3\}$ de segmentos de $\mathcal{T}$ e os valores $c_1 = 3$, $c_2 = 4$ e $c_3 = 5$, é criado o conjunto S contendo segmentos de 3, 4 e 5 cópias de b_1, b_2 e b_3, respectivamente.*

$\mathcal{S} = \mathcal{T} =$ ATTCATGGCGATCCAGCGGTATACCAAACCTGATCGTCAGATGCAAGCTGACGTACGTGC

$$T = \{\overset{b_1}{\text{ATTCATGGCGATCCA}} \quad \overset{b_2}{\text{GTATACCAAACCTGATCG}} \quad \overset{b_3}{\text{GATGCAAGCTGACGTACGTGC}}\}$$

$$S = \left\{ \begin{array}{lll} \text{ATTCA TGGCG ATCCA} & \text{GTATAC CAAACC TGATCG} & \text{GATGCAA GCTGACG TACGTGC} \\ \text{ATTC ATGGC GATCCA} & \text{GTATA CCAAAC CTGATCG} & \text{GATGCA AGCTGAC GTACGTGC} \\ \text{ATT CATGG CGATCCA} & \text{GTAT ACCAAA CCTGATCG} & \text{GATGC AAGCTGA CGTACGTGC} \\ & \text{GTA TACCAA ACCTGATCG} & \text{GATG CAAGCTG ACGTACGTGC} \\ & & \text{GAT GCAAGCT GACGTACGTGC} \end{array} \right\}$$

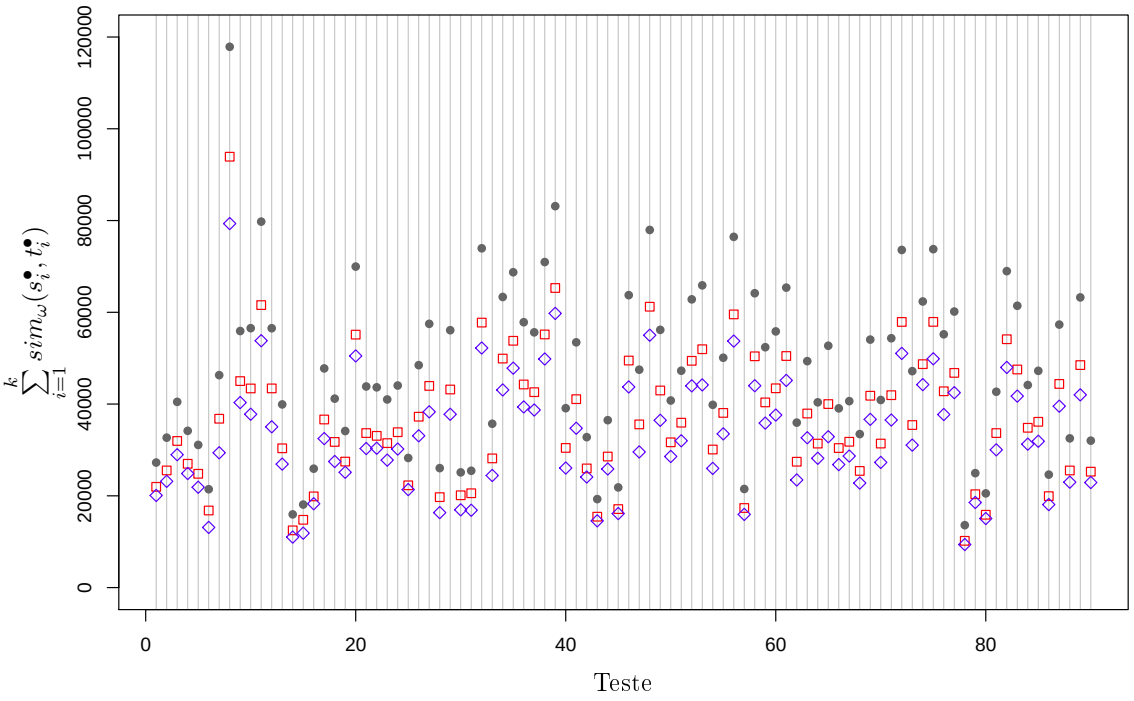
O procedimento descrito foi assim realizado para garantir a existência de uma partição de S em k cadeias de segmentos $s_1, s_2, ..., s_k$, tal que a concatenação de cada cadeia de segmentos resulte em uma sequência exatamente igual à sequência resultante da concatenação de um subconjunto de T. Dessa forma, podemos determinar o valor ótimo para cada instância, $max \sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$, que é igual a $\sum_{u \in S} |u| \cdot \omega(\alpha, \alpha)$, isto é, a soma do tamanho de todos os segmentos em S multiplicado pela pontuação de um *match*, e comparar com o valor encontrado pelas heurísticas.

Após a execução das heurísticas para cada teste separadamente, observamos que ambas encontraram o valor ótimo de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ para *todos* os 90 testes. Em outras palavras, elas encontraram uma solução ótima em todos os casos. No entanto, existe uma grande diferença entre o tempo médio de execução delas: 29 minutos para a heurística PASG e 2 segundos para a heurística DAG. Ou seja, a heurística DAG encontrou as solução ótimas em um tempo muito menor do que a heurística PASG.

[1] A quantidade de segmentos e o tamanho deles foram definidos baseando-se nos genes humanos, os quais possuem, em média, 8 éxons com tamanho médio de 145 nucleotídeos [40].

Como ambas as heurísticas encontraram soluções ótimas, não podemos indicar ainda qual delas possui maior acurácia. Por isso, modificamos os 90 testes de forma que $\mathcal{S}$ e $\mathcal{T}$ deixassem de ser idênticas. Mais precisamente, nós substituímos aproximadamente 10% dos nucleotídeos de $\mathcal{S}$, preservando o tamanho dela e as coordenadas dos seus segmentos. Sob essas condições, executamos novamente as heurísticas para os 90 testes. Os resultados obtidos encontram-se no gráfico da Figura 5.2 (para informações e resultados individuais dos 90 testes veja a Tabela A.1 do Apêndice A).

Figura 5.2: *Valores de* $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ *calculados pela heurística PASG (quadrados) e pela heurística DAG (losangos) para os 90 testes artificiais quando* $\mathcal{S} \neq \mathcal{T}$. *Os círculos demarcam o valor de* $\sum_{u \in S} |u|$ *que é maior ou igual a max* $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$.



Nesta nova bateria de testes, quando $\mathcal{S} \neq \mathcal{T}$, não sabemos o valor ótimo, $max \sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$, de cada instância. No entanto, sabemos que ele não será maior que $\sum_{u \in S} |u| \cdot \omega(\alpha, \alpha) = \sum_{u \in S} |u|$ (círculos no gráfico), pois como agora $\mathcal{S}$ é diferente de $\mathcal{T}$, haverá alguns *mismatches* ou *spaces* onde antes havia somente *matches*, reduzindo o valor ótimo de cada teste. Tendo isso em mente, podemos observar que os resultados das heurísticas chegam próximo do maior valor possível, e parecem representar porcentagens semelhantes desse valor em todos os testes. De fato, a média dos valores determinados pela heurística PASG é de 77.8% do maior valor possível, tendo um desvio padrão de somente 1.5. Já a média dos valores calculados pela heurística DAG é de 68.8% do maior valor possível, com desvio padrão de 3. O baixo desvio padrão obtido por elas sugere que as heurísticas tendem a encontrar uma solução cujo valor se aproxima das porcentagens mencionadas. Mas isso não pode ser generalizado, pois, aparentemente, a acurácia delas depende do quão parecidas $\mathcal{S}$ e $\mathcal{T}$ são.

Analisando o gráfico da Figura 5.2 podemos observar que a heurística PASG superou a heurística DAG em todos os testes. Esse fato nos permite dizer que a heurística PASG apresenta melhores resultados do que a heurística DAG. No entanto, o tempo médio de execução ainda é um fator muito distinto entre elas: 30 minutos para a heurística PASG e 2 segundos para a heurística DAG.

Os testes artificiais proporcionam uma boa comparação entre as heurística para o PPSM. No

entanto, eles estão longe das condições reais sob as quais o problema da reconstrução e quantificação de transcriptoma se encontra. Por isso, formulamos testes semi-artificiais com características mais próximas a tais condições.

5.3 Testes Semi-artificiais

Diferentemente dos testes artificiais, nos testes semi-artificiais nós iremos verificar a acurácia das heurísticas com relação à reconstrução e quantificação dos transcritos sobre um conjunto de dados gerados por um simulador de RNA-Seq.

Para criar os testes semi-artificiais, utilizamos o simulador FLUX [41] para simular a execução de experimentos de RNA-Seq sobre 103 genes do *Homo sapiens*. Tais genes fazem parte de um conjunto de testes criado e utilizado por de Lima [13]. Escolhemos o FLUX por ele ser o único simulador de RNA-Seq encontrado por nós que permite definir a quantidade de moléculas de cada transcrito distinto. Essa informação é essencial para verificar qual heurística se sai melhor na quantificação das isoformas. Outro fator positivo que o FLUX possui é o grande número de parâmetros que podem ser configurados. Dentre eles destaca-se ativação/desativação da simulação de uma amplificação por PCR[2]. No nosso caso decidimos desativar essa opção, pois a amplificação aumentaria o número de *reads* sequenciados, modificando a quantificação das isoformas. Além disso, desativamos também a opção de seleção dos fragmentos por tamanho, permitindo o sequenciamento de todos eles (vide (sub)Seção 2.3.3). Como o FLUX gera um *read* para cada fragmento de cDNA, se o tamanho deste último for maior que o do *read* a ser sequenciado, iremos perder informação, uma vez que o fragmento não será sequenciado por inteiro. Sendo assim, definimos o tamanho dos *reads* grande o suficiente (1300 nucleotídeos) para evitar que isso ocorra.

Considerando que uma isoforma i de um gene g corresponde a um conjunto não vazio com todos, alguns ou somente um dos seus éxons, criamos um total de 397 isoformas hipotéticas a partir dos 103 genes mencionados, e definimos um número aleatório de cópias entre 10 e 350 para cada uma, que corresponde à quantificação delas. Para cada teste, consideramos $\mathcal{S}$ e $\mathcal{T}$ como a sequência de nucleotídeos do respectivo gene, o conjunto S como os *reads* gerados pelo FLUX e o conjunto T como o conjunto de todos os éxons do gene. A ordem do conjunto S e do conjunto T foi estabelecida a partir das coordenadas dos *reads* devolvidas pelo FLUX, e a ordem que os éxons aparecem no gene, respectivamente.

Nós dizemos que um transcrito foi corretamente reconstruído se ele possui os mesmos éxons de alguma das isoformas hipotéticas mencionadas anteriormente. A partir disso, utilizamos as seguintes métricas, previamente utilizadas em [42–44], para avaliar os resultados das heurísticas:

$$\text{sensibilidade} = \frac{\text{VP}}{\text{VP} + \text{FN}}, \tag{5.1}$$

$$\text{precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}}, \tag{5.2}$$

onde VP (verdadeiros positivos) é o número de isoformas corretamente reconstruídas, FP (falsos positivos) é o número de isoformas incorretamente reconstruídas e FN (falsos negativos) é o número de

[2] A amplificação por PCR (*Polymerase Chain Reaction*) é comumente utilizada no sequenciamento de DNA/RNA, criando cópias das moléculas de DNA/cDNA com o objetivo de aumentar a sensibilidade das plataformas de sequenciamento de nova geração.

isoformas que não foram reconstruídas. Informalmente, a sensibilidade corresponde à porcentagem de isoformas que foram corretamente reconstruídas em relação ao número de isoformas presentes na anotação (hipotéticas) e a precisão corresponde à porcentagem de isoformas que foram corretamente reconstruídas em relação ao número total de isoformas reconstruídas pela heurística.

Primeiramente, vamos analisar a reconstrução dos transcritos pelas heurísticas, cujos resultados apresentam-se na Tabela 5.1.

Tabela 5.1: *Resultado da reconstrução dos transcritos pelas heurísticas, considerando as 397 isoformas dos 103 genes.*

	VP	FN	FP	sensibilidade	precisão
PASG	166	231	1123	41.8%	12.9%
DAG	143	254	939	36.0%	13.1%

A heurística PASG reconstruiu 1289 isoformas, das quais 166 são corretas e 1123 incorretas. Mesmo assim, ela obteve melhor sensibilidade do que a heurística DAG, que reconstruiu 1082 isoformas, sendo 143 corretas e 939 incorretas (para observar os resultados individuais e informações de cada gene veja a Tabela A.2 do Apêndice A). Com relação à precisão, ambas as heurísticas apresentaram praticamente o mesmo resultado, aproximadamente 13%. No entanto, o tempo médio de execução novamente se mostrou muito distinto: aproximadamente 3 horas para a heurística PASG e 7 segundos para a heurística DAG. Baseando-se nos resultados, não há dúvidas de que a reconstrução da heurística PASG foi melhor, porém, gastando muito mais tempo.

O resultado da quantificação é apresentado nos gráficos da Figura 5.3, nos quais o eixo horizontal representa a quantificação simulada e o eixo vertical a quantificação determinada pela respectiva heurística. Pode-se observar que ambas as heurísticas obtiveram resultados semelhantes. Porém, a quantificação obtida pela heurística PASG se mostrou um pouco melhor do que a quantificação da heurística DAG. Para chegar a essa conclusão, calculamos a distância média vertical de cada ponto à reta que representa a quantificação ideal. Tal valor corresponde a 42.8 para a heurística PASG e 45.7 para a heurística DAG. Portanto, dizemos que, além de ser mais sensível, a heurística PASG quantifica melhor as isoformas.

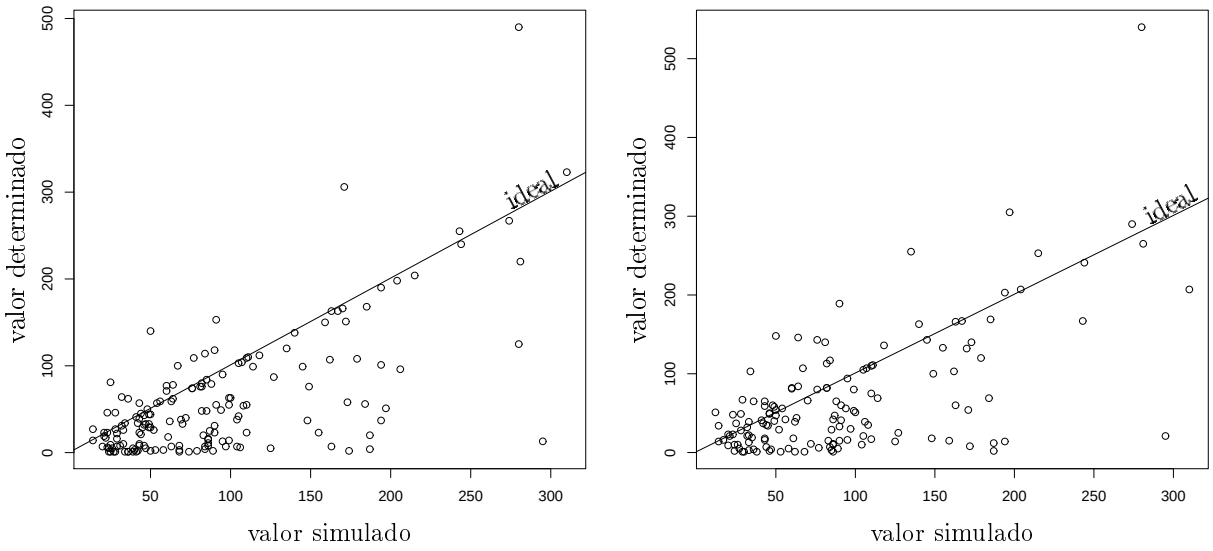
A respeito dos valores de $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$, a heurística PASG superou a heurística DAG em todos os 103 testes, obtendo um valor médio de 120164 para cada, em comparação com o valor médio de 78497 obtido pela heurística DAG.

Até aqui, realizamos análises sobre testes artificiais e semi-artificiais. Na próxima seção, iremos completar nossas análises com a verificação de resultados obtidos a partir de testes reais.

5.4 Testes Reais

Quando encaramos situações reais do problema de reconstruir e quantificar o transcriptoma, geralmente temos em mãos um conjunto desordenado de *reads* sequenciados do transcriptoma de interesse. Por isso, para modelar esses casos reais em instâncias do PPSM, é necessária a utilização de um mapeador e um genoma de referência anotado, do qual sabe-se as coordenadas de cada gene e de seus éxons. Nesta seção apresentaremos detalhes dessa modelagem, ou seja, como as sequências $\mathcal{S}$ e $\mathcal{T}$ e os conjuntos S e T podem ser obtidos a partir de dados reais do problema da reconstrução e quantificação de transcriptoma, e os resultados da aplicação das heurísticas sobre eles.

Figura 5.3: *Resultado da quantificação das heurísticas. Cada ponto no gráfico representa a quantificação de uma isoforma corretamente reconstruída. As retas que cortam os gráficos servem para identificar a quantificação ideal, isto é, onde os pontos deveriam estar.*



(a) *Heurística PASG* **(b)** *Heurística DAG*

Com o intuito de verificar a acurácia das heurísticas, adquirimos um conjunto de *reads*, gerados por RNA-Seq, de células B sanguíneas (linfócitos B) do ser humano[3], previamente utilizados em [27]. Os *reads* foram mapeados no genoma humano (versão GRCh38.p2[4]) utilizando o mapeador TopHat [45]. A partir desse mapeamento, escolhemos 99 genes que possuem de 300 a 1000 *reads* mapeados entre as suas coordenadas de início e término. A partir de cada gene escolhido g, definimos então uma instância do PPSM na qual ambas as sequências $\mathcal{S}$ e $\mathcal{T}$ correspondem à sequência de nucleotídeos de g. O conjunto S consiste dos *reads* mapeados entre as coordenadas de início e término de g. Com relação ao conjunto T, o consideramos como o conjunto dos éxons da principal isoforma de g, de acordo com a anotação APPRIS apresentada em [46]. Escolhemos a principal isoforma de cada gene por ela ser o seu representante, e, além disso, ser aquela com a qual todas as outras isoformas do mesmo gene devem ser comparadas [46].

Consideramos que um transcrito t de um gene g foi reconstruído corretamente se as coordenadas dos éxons internos (todos os éxons exceto o primeiro e o último) de t são exatamente iguais às coordenadas dos éxons internos de *alguma* isoforma t' presente na anotação de g, e se as coordenadas de início do primeiro e de término do último éxon de t possuem diferença de, no máximo, 100 nucleotídeos das respectivas coordenadas do primeiro e do último éxon de t'. Essa flexibilidade se deve ao fato de que a anotação pode não ser precisa com relação às coordenadas de início e de término das isoformas [27]. A partir disso, determinamos a sensibilidade e a precisão de ambas as heurísticas para o PPSM, cujos resultados encontram-se na Tabela 5.2.

Pode-se observar pela Tabela 5.2 que há um grande número de isoformas reconstruídas incorretamente. Para tentar reduzir esse número, os resultados foram filtrados de maneira que um transcrito reconstruído t_i é desconsiderado se o subconjunto s_i associado a t_i tiver somente um *read* ou se nenhum *read* em s_i sobrepor algum segmento em t_i. Essa filtragem vêm da suposição de que um

[3] Disponível em http://www.ncbi.nlm.nih.gov/sra?term=SRX174325. Último acesso em 25/08/2015.
[4] Obtido em http://www.gencodegenes.org/. Último acesso em 25/08/2015.

Tabela 5.2: *Resultado das heurísticas para os 99 genes escolhidos originalmente.*

	VP	FN	FP	sensibilidade	precisão
PASG	42	752	3774	5.3%	1.1%
DAG	13	781	2334	1.6%	0.5%

transcrito não pode ser constituído a partir de um único *read* ou de *reads* que não se sobrepõem a ele. Sendo assim, após a filtragem obtivemos os resultados apresentados na Tabela 5.3

Tabela 5.3: *Resultado das heurísticas após filtragem para os 99 genes escolhidos originalmente.*

	VP	FN	FP	sensibilidade	precisão
PASG	42	752	995	5.3%	4.0%
DAG	7	787	178	0.9%	3.8%

Comparando as Tabelas 5.2 e 5.3, a precisão de ambas as heurísticas aumentou consideravelmente. Porém, houve queda significativa da sensibilidade da heurística DAG. Tal redução no número de verdadeiros positivos pode ser justificada por um provável número elevado de transcritos distintos que a heurística DAG reconstruiu corretamente constituídos de um único *read* ou de *reads* que não sobrepõem a eles.

Existe um outro fator que também pode influenciar negativamente as heurísticas: os *reads* mapeados em cada gene podem não estar distribuídos uniformemente ao longo dele, podendo haver regiões do gene onde nenhum *read* foi mapeado. Isso pode resultar em éxons sem nenhum *read* mapeado em suas coordenadas, tornado a reconstrução das isoformas que compartilham tal éxon praticamente impossível. Sendo assim, numa segunda rodada de testes, selecionamos outros 50 genes que possuem pelo menos 6 *reads* mapeados em cada um de seus éxons, e com 290 a 3200 *reads* mapeados entre as coordenadas de início e término de cada gene. Os resultados de ambas as heurísticas sobre esse novo conjunto de genes são apresentados na Tabela 5.4.

Tabela 5.4: *Resultado das heurísticas para os 50 genes com um número mínimo de reads mapeado em cada um dos seus éxons.*

	VP	FN	FP	sensibilidade	precisão
PASG	57	518	2474	9.9%	2.2%
DAG	8	567	917	1.4%	0.9%

Analisando a Tabela 5.4, observa-se que, para este novo conjunto de genes, a sensibilidade da heurística PASG teve um aumento significativo em relação à Tabela 5.2. No entanto, a sensibilidade da heurística DAG teve uma pequena redução. Já a precisão de ambas as heurísticas foi ligeiramente melhor, ou seja, neste novo conjunto de genes a heurística DAG não obteve um ganho substancial.

Comparando os resultados filtrados dos 50 genes, apresentados na Tabela 5.5, com os resultados filtrados dos 99 genes (Tabela 5.3), podemos observar que, para os 50 genes, a filtragem resultou em um ganho menos significativo. Enquanto que a filtragem dos resultados dos 99 genes não reduziu a sensibilidade da heurística PASG e reduziu a da heurística DAG 0.7 ponto percentual, e aumentou a precisão em 2.9 e 3.3 pontos percentuais, respectivamente, a filtragem dos resultados dos 50 genes reduziu a sensibilidade da heurística PASG e da heurística DAG em 0.7 e 0.9 ponto percentual, e

aumentou a precisão em 1.6 e 1.8 pontos percentuais, respectivamente.

Tabela 5.5: *Resultado das heurísticas após filtragem para os 50 genes com um número mínimo de reads mapeado em cada um dos seus éxons.*

	VP	FN	FP	sensibilidade	precisão
PASG	53	522	1352	9.2%	3.8%
DAG	3	572	109	0.5%	2.7%

Considerando os resultados filtrados dos 149 genes testados, apresentados na Tabela 5.6, a heurística PASG reconstruiu 2442 isoformas, sendo 95 corretas e 2347 incorretas, obtendo sensibilidade cerca de dez vezes maior e com precisão ligeiramente melhor do que a heurística DAG, que reconstruiu 297 isoformas, das quais 10 são corretas e 287 são incorretas (veja a Tabela A.3 do Apêndice A para resultados mais detalhados e informações dos 149 genes testados). Entretanto, o tempo de execução continua sendo um ponto negativo da heurística PASG que, em média, foi de 124 minutos para os casos de testes aqui apresentados; enquanto que a heurística DAG foi de 16 segundos.

Tabela 5.6: *Resultado após filtragem dos 149 genes testados.*

	VP	FN	FP	sensibilidade	precisão
PASG	95	1274	2347	6.9%	3.9%
DAG	10	1359	287	0.7%	3.4%

Para determinar quão próximo os resultados obtidos pelas nossas heurísticas estão dos determinados por ferramentas frequentemente utilizadas para reconstruir o transcriptoma, executamos o Cufflinks [12] para os mesmos 149 genes. O Cufflinks é uma ferramenta muito utilizada pela comunidade científica para reconstruir e quantificar um transcriptoma. Basicamente, essa ferramenta constrói um conjunto de transcritos distintos através da busca por um emparelhamento máximo em um grafo bipartido, e utiliza um modelo estatístico para quantificar os transcritos. O Cufflinks pode reconstruir as isoformas com ou sem a ajuda de uma anotação. Por isso, nós o executamos duas vezes para cada gene, sendo uma sem a ajuda da anotação e a outra lhe informando as coordenadas do gene e as dos éxons da principal isoforma (como feito com as nossas heurísticas). Observando a Tabela 5.7, a reconstrução guiada pela anotação obteve melhores resultados, reconstruindo 531 isoformas, das quais 148 são corretas e 383 são incorretas.

Tabela 5.7: *Resultado do Cufflinks com e sem a ajuda da anotação para os 149 genes.*

	VP	FN	FP	sensibilidade	precisão
Cufflinks sem anotação	14	1355	490	1.0%	2.8%
Cufflinks com anotação	148	1221	383	10.9%	27.9%

Comparando as Tabelas 5.6 e 5.7(com anotação), percebe-se que a sensibilidade da heurística PASG não está tão distante da obtida pelo Cufflinks. Já o mesmo não pode ser dito da heurística DAG, cuja sensibilidade está muito abaixo do esperado. Entretanto, ambas as heurísticas possuem baixa precisão quando comparadas com o Cufflinks.

Os testes reais aqui apresentados limitam-se somente à reconstrução dos transcritos. Nós não avaliamos a quantificação deles em razão de que, diferentemente dos testes semi-artificiais, não

temos a informação do número de moléculas das isoformas correspondentes ao conjunto de *reads* utilizado.

5.5 Discussão

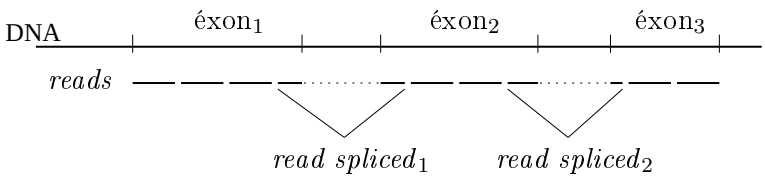
Dos resultados apresentados neste capítulo, observamos que a heurística PASG superou a heurística DAG nos testes artificiais, semi-artificiais e reais, obtendo valores maiores de $\sum_{i=1}^{k} sim_{\omega}(s_i^{\bullet}, t_i^{\bullet})$, maior sensibilidade e precisão na reconstrução dos transcritos e melhor quantificação. No entanto, pelo seu elevado tempo de execução, é provável que sua execução seja inviável para instâncias do PPSM maiores que as testadas neste trabalho. Essa inviabilidade ficaria mais evidente se o objetivo fosse reconstruir o transcriptoma inteiro de um organismo que possui milhares de genes. Para esses casos, propostas de melhorias para as heurísticas ou o desenvolvimento de novas heurísticas ou algoritmos de aproximação pode ser necessário.

Podemos observar também que a sensibilidade e a precisão das heurísticas são baixas. Um dos motivos para este fato pode estar relacionado à alta complexidade do transcriptoma humano. Isso pode ser verificado observando que o Cufflinks, mesmo sendo uma ferramenta amplamente utilizada pela comunidade científica, não apresentou resultados muito melhores (isso também pode ser verificado nos resultados apresentados por Pertea *et al.* [27]). Outro fator que limita a acurácia das heurísticas é o fato de que elas não reconstroem transcritos que não sejam uma combinação dos éxons em T. Ou seja, se existir uma isoforma que possui algum éxon com coordenadas distintas de todos os éxons em T, as heurísticas não poderão reconstruí-la. Mais do que isso, nenhum algoritmo que encontra uma solução para o PPSM poderia reconstruir essa isoforma, pois esse é um problema intrínseco à modelagem do problema da reconstrução e quantificação de transcriptoma pelo PPSM.

Uma outra questão importante é o fato de que nada impede de existir várias soluções ótimas, ou próximas da ótima, para o PPSM. Por outro lado, quando modelamos o problema da reconstrução e quantificação de transcriptoma no PPSM, queremos reconstruir um conjunto específico de transcritos, ou seja, aqueles que foram expressos a partir do gene. Isso significa que podem existir soluções ótimas do PPSM que não correspondem ao conjunto de transcritos buscado. Por isso, quando aplicado no problema da reconstrução e quantificação de transcriptoma, encontrar uma solução ótima do PPSM pode não ser suficiente. Uma ação que pode minimizar esse problema é a utilização de informações biológicas que possam direcionar a busca de uma solução do PPSM que chegue próximo ao conjunto de transcritos a ser reconstruído. Um exemplo de tais informações são os *reads spliced*, isto é, os *reads* que compreendem trechos de éxons distintos, como exemplificado na Figura 5.4. Esses *reads* são encontrados por mapeadores *spliced*, como o TopHat, e podem indicar quais éxons aparecem juntos em um mesmo transcrito. Por exemplo, observando a Figura 5.4, o *read spliced*$_1$ sugere a existência de um transcrito que possui os éxons 1 e 2, e o *read spliced*$_2$ sugere a existência de um transcrito que possui os éxons 2 e 3.

Os testes apresentados neste capítulo sustentam a afirmação de que é possível, apesar das limitações, encontrar uma solução para o problema da reconstrução e quantificação de transcriptoma modelando-o como instâncias do PPSM. Vale lembrar que essa é apenas umas das possíveis aplicações do PPSM, embora seja a única abordada neste trabalho.

Figura 5.4: *Exemplo de dois reads spliced. O read spliced$_1$ é mapeado no éxon$_1$ e éxon$_2$, e o read spliced$_2$ é mapeado no éxon$_2$ e éxon$_3$.*



Capítulo 6

Conclusão

Neste trabalho, apresentamos um novo problema de otimização combinatória envolvendo sequências chamado Problema do Particionamento de Similaridade Máxima (PPSM) e demonstramos como ele pode ser aplicado em um problema importante da Biologia Computacional, conhecido como problema da reconstrução e quantificação de transcriptoma. Além disso, provamos que o PPSM é um problema NP-difícil no sentido forte por meio de uma redução pseudo-polinomial e implementamos duas heurísticas, a heurística PASG e a heurística DAG, baseadas na estratégia gulosa para ele. Com o finalidade de avaliar as heurísticas, foi criado um conjunto de testes que divide-se em testes artificiais do PPSM, testes semi-artificiais e testes reais do problema da reconstrução e quantificação de transcriptoma.

No caso dos testes artificiais, observamos que as heurísticas encontram soluções do PPSM próximas à ótima. No entanto, em relação aos testes semi-artificiais e reais, a sensibilidade e a precisão da heurística PASG e da heurística DAG são baixas, ou seja, elas apresentaram resultados ruins para o problema da reconstrução e quantificação de transcriptoma. A respeito do desempenho individual das heurísticas, a heurística PASG é melhor que a heurística DAG em todos os aspectos verificados, exceto tempo de execução, sendo esse um fator que poderia impossibilitar a execução da heurística PASG para entradas muito grandes.

A modelagem do problema da reconstrução e quantificação de transcriptoma pelo PPSM possui duas características interessantes. Em primeiro lugar, a busca por uma solução para o problema da reconstrução e quantificação de transcriptoma consiste basicamente em aplicar o PPSM para cada gene separadamente. Isso permite a escolha de um ou alguns genes específicos de maior interesse para que o PPSM possa ser aplicado somente neles, evitando a reconstrução desnecessária e quantificação do transcriptoma inteiro. Em segundo lugar, a quantificação das isoformas é dada em número de moléculas. Isso proporciona fácil comparação do nível de expressão entre genes ou isoformas

Constatamos que, de maneira geral, reconstruir e quantificar um transcriptoma não é um problema fácil. Várias ferramentas já foram desenvolvidas para esse fim, mas nenhuma delas apresenta resultados condizentes com a realidade em todos os casos. Por isso, novas abordagens, como a apresentada neste trabalho, se fazem necessárias com o propósito de alcançar melhores resultados.

Este trabalho pode ser considerado pioneiro na abordagem do problema da reconstrução e quantificação de transcriptoma como um problema de otimização combinatória envolvendo sequências. Sendo assim, ainda há muitos aspectos que podem ser estudados mais a fundo, como:

- o aperfeiçoamento das implementações da heurística PASG e da heurística DAG, com o intuito

de reduzir o tempo de execução delas;

- a proposta de novas heurísticas ou algoritmos de aproximação para o PPSM que possam apresentar melhores resultados;
- o uso de informações biológicas nas heurísticas desenvolvidas a fim de alcançar resultados mais satisfatórios para o problema da reconstrução e quantificação de transcriptoma;
- o desenvolvimento de variações do PPSM com o objetivo de melhorar a modelagem do problema da reconstrução e quantificação de transcriptoma.

Vale lembrar também que o PPSM pode ser utilizado para modelar outros problemas, não necessariamente biológicos, que envolvem a busca por um particionamento de um conjunto.

Apesar das heurísticas apresentarem resultados ruins com relação ao problema de reconstruir e quantificar o transcriptoma, consideramos que o objetivo principal deste trabalho foi alcançado, com a proposta e estudo detalhado de um novo problema de otimização combinatória envolvendo sequências e sua aplicação em um problema prático e de bastante interesse para a comunidade científica.

Apêndice A

Conjuntos de Testes e Resultados Individuais

As Tabelas A.1, A.2 e A.3 apresentam informações e resultados obtidos pelas heurísticas sobre cada um dos testes que compõem o conjunto dos testes artificiais, semi-artificiais e reais, respectivamente. O tamanho da sequência $\mathcal{S}$, que é igual ao tamanho da sequência $\mathcal{T}$, é dado em número de nucleotídeos, e é denotada por $|\mathcal{S}|$. A quantidade de segmentos em S (conjunto dos *reads*) e em T (conjunto dos éxons) é denotada por $|S|$ e $|T|$, respectivamente.

Tabela A.1: *Conjunto dos testes artificiais. Para melhor visualização, chamamos de score o valor de* $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$ *calculado pela respectiva heurística quando* $\mathcal{S} \neq \mathcal{T}$*, e de OPT o valor de* $\sum_{u \in S} |u|$ *que é maior ou igual ao valor ótimo max* $\sum_{i=1}^{k} sim_\omega(s_i^\bullet, t_i^\bullet)$.

					Heurística PASG		Heurística DAG	
Teste	$\lvert\mathcal{S}\rvert = \lvert\mathcal{T}\rvert$	$\lvert S\rvert$	$\lvert T\rvert$	OPT	*score*	Tempo	*score*	Tempo
1	1023	528	5	27251	21960	1m25.372s	20081	0m0.905s
2	1510	860	6	32675	25533	5m6.812s	23157	0m0.983s
3	1360	968	4	40480	31958	8m39.195s	28964	0m1.224s
4	1345	816	7	34165	26981	3m51.068s	24859	0m1.007s
5	1940	752	8	31094	24802	4m10.619s	21868	0m0.987s
6	2180	432	6	21440	16800	0m53.533s	13112	0m0.733s
7	2320	1212	5	46304	36806	19m58.922s	29386	0m1.653s
8	5770	3000	10	117870	93914	329m18.068s	79334	0m8.758s
9	2470	1164	12	55920	45008	15m51.263s	40326	0m2.039s
10	2864	1312	4	56540	43402	31m1.134s	37781	0m2.245s
11	3367	2056	6	79762	61572	111m55.746s	53806	0m5.715s
12	2780	1632	5	56527	43413	45m58.755s	35045	0m2.424s
13	1864	864	3	39922	30360	8m34.579s	26914	0m1.225s
14	798	444	2	15956	12472	0m39.068s	11006	0m0.480s
15	937	464	2	18094	14774	0m56.783s	11856	0m0.544s
16	1173	672	3	25914	19876	2m42.230s	18276	0m0.763s
17	1763	1072	6	47767	36634	13m53.771s	32469	0m1.459s
18	1668	984	4	41179	31749	11m2.270s	27477	0m1.255s
19	1034	1056	3	34122	27460	9m35.251s	25124	0m1.153s
20	2865	1648	7	69963	55132	58m56.781s	50492	0m2.873s
21	1883	1008	5	43832	33676	13m14.730s	30306	0m1.481s
22	1929	1120	4	43645	33073	16m33.871s	30365	0m1.444s
23	1865	1040	6	41002	31489	11m29.506s	27782	0m1.301s
24	1756	996	6	44041	33867	10m58.698s	30193	0m1.334s
25	953	720	2	28250	22292	3m41.638s	21360	0m0.849s
26	1970	1268	5	48478	37246	22m3.046s	33082	0m1.642s
27	1926	1460	6	57486	43958	35m9.071s	38306	0m2.058s
28	1150	720	2	26036	19714	3m36.831s	16340	0m0.763s
29	2523	1240	7	56096	43172	24m45.175s	37770	0m2.089s
30	1103	662	5	25109	20117	2m4.553s	16955	0m0.739s
31	921	722	3	25457	20559	3m8.238s	16853	0m0.774s
32	2776	2240	7	73950	57756	116m51.391s	52218	0m3.409s
33	1351	792	5	35708	28158	5m9.579s	24432	0m1.083s
34	2963	1576	7	63343	49916	48m53.121s	43079	0m2.716s
35	2849	1804	7	68731	53789	68m40.884s	47825	0m2.939s

Continua na próxima página

Tabela A.1 – *Continuação da tabela anterior*

					Heurística PASG		Heurística DAG	
Teste	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	OPT	*score*	Tempo	*score*	Tempo
36	2368	1284	9	57837	44267	26m1.300s	39377	0m2.051s
37	1963	1430	9	55630	42594	30m13.245s	38708	0m1.945s
38	2745	1752	8	70950	55173	65m10.356s	49827	0m2.974s
39	2803	2066	9	83136	65299	107m37.300s	59760	0m3.633s
40	1716	1054	5	39097	30441	11m47.405s	26059	0m1.230s
41	2264	1304	5	53455	41069	26m12.843s	34726	0m1.921s
42	1172	892	5	32783	25972	5m35.469s	24105	0m1.024s
43	717	532	2	19287	15455	1m7.990s	14551	0m0.573s
44	1662	1076	5	36484	28558	11m19.397s	25854	0m1.186s
45	933	560	2	21835	17111	1m36.554s	16133	0m0.650s
46	2716	1458	8	63741	49487	39m42.288s	43732	0m2.525s
47	1877	1256	4	47488	35566	22m31.983s	29566	0m1.617s
48	2825	1988	10	77949	61218	94m20.765s	55003	0m3.427s
49	2345	1540	6	56170	42959	39m50.171s	36461	0m2.141s
50	1724	1066	5	40776	31664	12m14.679s	28576	0m1.267s
51	1922	1368	4	47263	35939	25m27.747s	31997	0m1.657s
52	2828	1506	14	62822	49428	41m6.573s	43970	0m2.576s
53	2974	1642	9	65873	51951	54m1.128s	44175	0m2.832s
54	1899	988	6	39829	30091	10m41.110s	25977	0m1.265s
55	1866	1072	4	50098	38072	16m24.281s	33502	0m1.603s
56	2783	1844	7	76445	59543	80m45.393s	53725	0m3.254s
57	848	454	3	21492	17344	0m56.596s	15950	0m0.710s
58	2808	1498	10	64155	50410	42m58.170s	44001	0m2.555s
59	2440	1242	7	52388	40363	23m31.344s	35846	0m1.892s
60	2152	1430	5	55835	43438	34m14.136s	37608	0m1.998s
61	2507	1692	7	65374	50474	58m31.521s	45156	0m2.569s
62	1802	996	5	35953	27432	10m7.305s	23459	0m1.146s
63	2066	1160	4	49353	37938	20m19.526s	32654	0m1.624s
64	1542	1080	3	40380	31384	14m13.467s	28178	0m1.270s
65	2360	1292	5	52712	39998	26m41.075s	32878	0m1.950s
66	2021	830	6	39063	30414	7m13.061s	26805	0m1.236s
67	1256	776	3	40635	31784	6m29.069s	28671	0m1.236s
68	1746	768	4	33456	25425	5m29.945s	22781	0m1.018s
69	2127	1276	5	54048	41814	24m53.834s	36664	0m1.840s
70	1869	1022	7	40889	31391	11m22.707s	27287	0m1.327s
71	2080	1402	6	54345	41937	30m35.224s	36515	0m1.957s
72	2878	1678	8	73587	57899	63m43.926s	51039	0m2.985s
73	1886	1240	4	47174	35432	22m13.533s	31054	0m1.569s
74	2591	1510	7	62357	48699	43m4.858s	44232	0m2.390s
75	2976	1852	7	73761	57915	77m2.220s	49865	0m3.305s
76	2339	1476	7	55202	42784	35m31.906s	37722	0m2.154s
77	2705	1448	7	60165	46796	39m9.067s	42483	0m2.369s
78	877	248	2	13604	10192	0m13.933s	9382	0m0.401s
79	983	592	2	24944	20348	2m15.722s	18546	0m0.731s
80	897	368	2	20529	15871	0m40.536s	15051	0m0.582s
81	1544	1216	4	42684	33678	17m29.092s	30024	0m1.457s
82	2993	1552	8	68968	54129	50m33.160s	47982	0m2.869s
83	2640	1720	7	61423	47535	54m52.098s	41731	0m2.489s
84	1591	1186	4	44144	34833	17m33.782s	31260	0m1.408s
85	2052	1236	5	47243	36135	21m37.724s	31895	0m1.807s
86	823	522	3	24619	19931	1m24.506s	18089	0m0.692s
87	2586	1472	7	57321	44386	36m8.958s	39537	0m2.214s
88	1543	1000	5	32526	25524	8m32.448s	22974	0m1.103s
89	2537	1590	9	63246	48496	48m27.430s	42003	0m2.440s
90	1383	738	4	32024	25266	4m38.602s	22902	0m0.911s

Tabela A.2: *Conjunto dos testes semi-artificiais. A quinta coluna apresenta o número de isoformas simuladas para cada gene. Os valores de VP e FP correspondem ao número de isoformas correta e incorretamente reconstruídas, respectivamente.*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	tempo	VP	FP	tempo
aff4	90284	1801	20	7	0	31	191m58.272s	0	25	0m8.170s
ankrd10	38530	1089	6	4	0	10	40m19.112s	0	7	0m3.790s
ascc2	51655	1553	19	8	0	33	91m58.920s	0	30	0m4.528s
asz1	66302	1426	13	7	0	19	63m9.520s	0	20	0m5.706s
bad	16877	797	3	3	3	2	16m34.040s	3	3	0m7.776s
c20orf152	64094	1386	12	5	0	21	68m12.310s	0	18	0m4.371s
c21orf59	12572	604	7	5	2	10	4m30.815s	1	9	0m1.712s
c21orf63	104946	1139	8	4	2	11	38m11.592s	2	4	0m3.935s
c7orf63	67164	1588	22	7	0	29	95m45.117s	0	25	0m4.907s

Continua na próxima página

Tabela A.2 – *Continuação da tabela anterior*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	tempo	VP	FP	tempo
calcr	151952	1234	12	7	0	23	39m23.856s	1	19	0m3.272s
cav1	38392	1069	3	2	2	4	31m13.909s	2	3	0m4.755s
ccdc157	22192	1086	10	5	1	14	43m44.823s	0	15	0m3.912s
ccdc88b	19312	1583	27	5	0	25	127m50.684s	0	22	0m7.826s
cdh8	384802	1431	11	8	1	16	113m31.912s	1	19	0m7.310s
cdx2	9040	1032	3	3	3	2	42m45.554s	3	1	0m4.924s
chmp2a	5554	4280	5	3	2	10	2005m50.197s	2	7	1m7.391s
ddx18	19699	1390	14	5	0	24	70m54.074s	0	21	0m5.092s
ddx43	25005	1342	16	6	0	25	50m47.446s	0	19	0m3.708s
dppa5	3167	506	3	2	2	2	3m55.551s	2	3	0m4.166s
eef1a1	7283	1450	7	7	0	10	107m53.751s	1	10	0m5.893s
esrra	13167	1178	6	6	1	9	51m38.455s	1	5	0m4.241s
flt3	99319	1193	24	5	0	22	38m3.481s	0	24	0m3.605s
frs3	11717	800	5	3	2	8	20m7.427s	1	6	0m3.041s
gabrq	17189	1294	9	4	3	12	83m19.536s	0	15	0m6.971s
gal3st1	12253	540	2	2	2	1	10m54.137s	2	1	0m2.626s
gng2	111469	617	2	2	2	1	3m58.598s	2	1	0m3.464s
gpr137	5633	647	7	3	1	9	7m0.001s	2	7	0m1.974s
gsx1	3310	1226	2	2	2	1	71m54.281s	2	0	0m7.219s
hba1	2842	647	3	2	2	4	5m56.590s	1	4	0m3.083s
hba2	2864	277	3	2	2	2	1m48.483s	1	3	0m2.551s
hbb	3606	527	3	3	3	3	3m12.441s	3	1	0m2.344s
hbg2	3591	392	3	2	2	4	3m8.570s	1	2	0m2.325s
hbq1	2844	738	3	3	3	3	8m6.372s	3	1	0m3.505s
hoxa4	4274	849	2	2	2	1	28m40.943s	2	0	0m4.014s
hoxa7	4962	618	2	2	1	1	12m1.775s	2	1	0m3.267s
hunk	132750	1350	11	4	0	16	83m21.191s	0	16	0m5.478s
il13	4937	581	4	4	3	6	3m31.652s	4	3	0m2.029s
il5	4079	579	4	4	3	4	4m9.628s	3	3	0m2.334s
kcnj6	293912	751	3	3	2	3	22m12.961s	2	3	0m3.541s
khdc1	23871	546	3	3	2	4	4m25.514s	2	1	0m2.468s
lace1	230161	1486	13	6	0	24	60m54.297s	1	20	0m4.090s
lif	8355	923	3	3	3	4	23m33.042s	3	1	0m3.235s
march11	114424	695	4	4	3	4	13m6.587s	2	6	0m2.910s
mdfi	17788	656	4	4	3	5	10m3.034s	2	4	0m2.255s
mettl2b	28196	606	9	3	2	9	4m31.694s	1	7	0m1.740s
mmp26	6236	887	6	5	4	9	16m15.626s	2	6	0m2.593s
mrpl23	11338	831	5	3	1	10	8m56.684s	0	8	0m2.540s
mzf1	13659	1314	5	4	3	10	129m7.713s	2	7	0m7.812s
nipal1	22294	1337	6	6	3	12	79m2.966s	3	7	0m5.945s
nr2e1	24799	529	9	4	0	12	3m24.646s	0	9	0m1.457s
ooep	3238	1579	3	3	3	0	71m22.562s	3	3	0m10.210s
ostm1	35329	676	6	3	2	8	7m9.468s	2	6	0m2.179s
pes1	17283	1596	15	3	1	26	98m28.472s	1	14	0m5.447s
pgc	12670	1113	9	7	0	20	27m22.832s	1	10	0m3.468s
pla2g3	7677	1281	7	6	1	9	67m0.012s	2	11	0m5.273s
polr3k	8626	510	3	2	2	3	3m38.296s	1	5	0m4.007s
pon3	38504	1425	9	8	1	22	56m14.699s	0	18	0m4.445s
ppp1r14b	4463	664	4	2	2	7	5m2.741s	1	5	0m1.744s
prickle4	8611	998	6	4	2	4	35m2.333s	2	5	0m3.279s
rasl10b	13862	665	3	2	2	2	10m30.437s	2	4	0m2.931s
rbm28	35527	1478	19	7	0	26	78m48.424s	0	23	0m3.984s
rps5	9536	3368	5	3	3	5	810m45.671s	3	8	0m42.648s
samd9l	20313	3882	1	1	1	0	3752m8.552s	1	0	0m55.512s
sec14l3	14799	646	12	5	0	19	5m2.006s	0	23	0m1.556s
selm	4789	1241	5	5	2	8	23m31.526s	2	5	0m4.364s
shroom1	5991	1298	7	4	1	10	94m29.355s	2	10	0m6.418s
skap2	199655	1248	12	7	0	28	28m13.387s	1	13	0m3.114s
slc22a11	17902	626	10	2	1	10	7m5.234s	1	3	0m1.660s
slc22a4	51755	5285	10	4	2	23	4922m41.115s	2	17	1m2.390s
slc25a13	203928	1449	18	2	1	27	82m25.286s	0	24	0m7.460s
slc35e4	14070	765	2	2	2	0	22m23.528s	2	1	0m3.563s
slfn12l	64955	638	4	2	2	5	34m41.089s	1	4	0m4.569s
snd1	442458	1087	24	3	0	18	28m52.243s	0	25	0m4.573s
spp2	28431	2281	7	4	3	13	182m31.489s	2	8	0m13.335s
steap1	12453	743	4	3	1	6	14m40.528s	1	5	0m2.851s
stip1	20434	1312	14	4	0	27	50m43.311s	0	20	0m4.139s
taf4b	167241	1424	15	5	0	20	94m10.365s	0	20	0m5.755s
tbc1d10a	36916	2125	9	4	1	17	298m51.340s	2	11	0m9.785s
tcn2	21887	831	9	3	1	13	16m40.654s	1	10	0m3.665s
tec	136015	1437	17	4	2	17	60m17.663s	1	16	0m4.206s
tfeb	53083	2273	8	4	3	12	354m36.048s	2	10	0m11.432s
tfpi2	6321	1054	5	5	2	8	25m38.861s	1	7	0m3.979s
tiam1	442555	1774	25	5	0	27	197m45.918s	0	24	0m10.168s
tmem8	13175	1217	13	4	1	16	64m50.657s	0	22	0m5.922s

Continua na próxima página

Tabela A.2 – *Continuação da tabela anterior*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	tempo	VP	FP	tempo
tnni2	4006	3682	7	5	1	32	903m15.616s	0	10	0m58.277s
trpm8	104124	1298	24	4	0	23	55m32.738s	0	26	0m4.688s
usp49	99717	961	4	3	2	6	50m5.701s	2	1	0m4.711s
wnt2	48659	1107	5	5	4	3	49m27.999s	3	6	0m6.014s
zbtb45	8025	1426	2	2	2	1	184m38.649s	2	0	0m10.363s
zfp92	5306	1123	4	3	2	8	65m47.483s	2	5	0m5.479s
zmat5	38025	307	5	3	3	3	0m43.923s	3	0	0m1.019s
znf132	9409	590	3	2	2	3	14m31.255s	1	2	0m2.946s
znf135	12165	1478	4	4	4	10	165m39.863s	2	6	0m8.719s
znf275	20772	1065	3	3	2	4	56m28.432s	2	3	0m5.888s
znf324	8303	1143	3	3	3	3	75m54.761s	2	2	0m5.661s
znf446	6803	690	6	4	2	7	13m6.445s	1	10	0m2.626s
znf551	9823	1248	3	3	3	3	107m32.812s	2	3	0m6.663s
znf584	11632	1867	4	2	2	8	280m1.370s	2	5	1m2.390s
znf622	16267	1430	6	2	2	5	98m18.304s	2	4	0m9.785s
znf8	18937	549	4	2	2	4	9m5.375s	2	4	0m2.372s
znf814	21696	958	3	3	2	4	55m13.243s	2	3	0m5.838s
zscan1	22566	1430	4	4	3	5	111m13.440s	3	6	0m6.605s
zscan22	17328	684	2	2	2	1	22m12.576s	2	1	0m3.628s

Tabela A.3: *Conjunto dos testes reais. Os campos são iguais aos da Tabela A.2, com a diferença de que o número de isoformas de cada gene corresponde com a anotação dele.*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	Tempo	VP	FP	Tempo
AC006116.20	37683	918	7	8	0	55	27m12.972s	0	10	0m9.445s
ACSM6	34730	823	11	4	0	48	56m33.898s	0	20	0m8.300s
ACSS3	324793	896	16	8	0	92	326m33.499s	0	30	0m15.135s
ADAMTS14	89640	924	22	2	0	87	157m31.171s	0	36	0m10.997s
ADAMTS16	179976	891	23	4	0	92	77m31.333s	0	44	0m11.914s
ADHFE1	41418	1935	14	18	2	122	188m12.849s	0	27	0m41.148s
ADORA2B	30831	550	2	2	1	2	7m44.277s	1	2	0m5.806s
ADORA3	80616	888	2	10	0	2	30m1.388s	1	2	0m10.288s
AIRE	12812	828	14	5	0	64	17m4.139s	0	30	0m8.422s
AKIP1	8947	1035	6	11	2	20	38m1.077s	0	13	0m9.718s
ALDH3B1	20706	738	10	10	1	51	25m17.933s	0	19	0m7.461s
APLP1	11894	655	17	11	0	67	17m39.043s	0	31	0m6.676s
ARL6	36590	1022	9	8	1	51	29m2.254s	0	16	0m11.267s
ASMT	28082	972	8	5	1	36	25m10.397s	1	15	0m10.094s
ATP2B2	384010	564	20	9	0	38	110m33.899s	0	34	0m8.243s
AZIN2	39428	567	12	19	1	40	6m49.849s	2	21	0m8.234s
BOC	76458	793	20	17	0	31	21m38.425s	0	41	0m4.359s
C18orf54	27303	940	8	6	1	23	53m57.513s	0	17	0m10.649s
C1orf198	32472	873	4	7	1	9	50m57.079s	0	6	0m5.813s
CACNA2D1	497356	990	39	12	0	158	251m55.757s	0	54	0m15.630s
CCDC129	144651	769	14	11	0	94	88m28.000s	0	24	0m9.133s
CCDC136	31377	1231	18	15	0	57	84m34.955s	0	35	0m15.400s
CCDC17	4015	1332	13	9	1	40	50m9.016s	1	32	0m8.101s
CDH24	10478	1070	13	6	0	41	50m24.211s	0	25	0m9.095s
CDRT1	54223	986	12	9	0	26	64m50.651s	0	22	0m11.534s
CES1	30488	878	14	9	0	37	50m22.816s	0	25	0m7.702s
CETN3	17527	636	5	6	1	15	25m54.013s	0	9	0m8.325s
CFB	6436	1116	18	14	0	58	64m14.750s	0	35	0m12.070s
CLDND2	1907	534	4	3	1	7	5m51.775s	0	8	0m3.895s
CLUAP1	38126	3154	12	12	1	136	1503m33.362s	0	21	2m33.376s
CMAS	19502	2754	8	5	2	67	535m25.235s	0	15	1m44.586s
CMTM1	12746	788	4	20	1	8	18m58.596s	0	7	0m6.990s
CNTN1	379978	907	23	11	0	92	161m32.845s	0	43	0m10.603s
CORO6	8153	405	11	12	0	31	2m51.009s	0	21	0m3.045s
CPAMD8	133870	968	42	20	0	94	77m14.515s	0	78	0m11.469s
CUX2	316532	773	22	3	0	62	227m18.601s	0	35	0m9.939s
CYP2E1	40816	988	9	10	1	26	36m39.705s	0	16	0m9.120s
DCAF12	41014	697	9	5	1	27	33m37.083s	0	17	0m6.659s
DHDH	11289	324	7	4	0	17	1m25.892s	0	13	0m3.243s
DNAH5	254214	578	79	4	0	78	107m58.215s	0	91	0m7.101s
DNAJC18	35905	2140	8	12	2	57	504m55.016s	0	14	0m27.807s
DNAJC5B	78968	1041	6	4	2	21	35m52.982s	0	10	0m13.129s
EBPL	30754	2208	4	7	3	10	333m1.074s	0	7	0m41.639s
ETV7	33711	990	8	8	1	32	20m12.896s	0	14	0m9.798s
FBXO15	74514	750	10	13	1	48	15m44.945s	1	17	0m12.803s
FHAD1	153013	858	31	22	0	64	87m18.223s	0	60	0m10.349s

Continua na próxima página

Tabela A.3 – *Continuação da tabela anterior*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	Tempo	VP	FP	Tempo
FHL2	80803	522	7	12	1	30	6m27.434s	0	13	0m5.420s
FMO4	27878	2512	10	5	1	66	635m6.915s	0	20	1m19.194s
GGCT	8225	1982	4	8	1	7	229m54.721s	0	7	0m43.086s
GH1	1640	794	5	6	1	15	9m17.321s	1	9	0m3.806s
GHRL	7274	1441	6	15	3	24	72m58.168s	0	11	0m17.264s
GIMAP1-GIMAP5	27094	772	6	2	0	27	28m5.827s	0	12	0m9.395s
GPC2	7768	1288	10	7	1	32	62m20.031s	1	19	0m7.883s
GRIA3	306764	653	16	9	0	55	56m58.223s	0	31	0m8.085s
GRIN2A	429352	773	13	11	0	34	146m21.994s	0	25	0m9.944s
HEPH	106320	962	21	9	0	91	169m40.922s	0	39	0m12.707s
HPGD	32979	291	7	14	0	14	6m16.455s	0	12	0m3.118s
HSD11B1L	7921	855	8	23	3	32	19m14.907s	0	15	0m6.323s
IFT22	8458	918	5	11	2	11	11m54.297s	0	9	0m6.663s
IL15RA	40620	2656	8	23	2	45	474m5.979s	0	15	1m33.097s
INSL3	5064	720	2	3	1	2	7m38.867s	1	2	0m8.269s
ITGA2	105455	866	30	6	0	104	115m9.024s	0	58	0m8.754s
KIAA1841	72165	2821	22	11	1	119	533m29.886s	0	41	1m41.315s
KLHL4	152303	961	11	2	0	42	266m56.857s	0	23	0m12.223s
KNDC1	65967	1070	30	5	0	61	65m28.699s	0	59	0m15.244s
LIN7B	15486	1326	6	7	2	24	31m35.559s	0	14	0m14.889s
LRRC34	19560	1910	10	12	1	90	215m26.464s	0	20	0m36.489s
LRRCC1	38931	1134	19	6	0	56	45m9.984s	0	38	0m36.448s
LYSMD1	6202	568	3	2	1	3	15m4.577s	1	5	0m4.148s
MADCAM1	16169	606	5	10	1	13	12m18.143s	1	8	0m6.220s
MAP6D1	9720	541	3	4	1	5	8m8.566s	0	3	0m3.186s
MS4A4E	41837	701	5	7	1	24	13m48.826s	0	7	0m7.242s
MYRF	35878	589	26	8	0	65	25m4.237s	0	50	0m5.251s
NACC2	88750	655	6	3	0	20	63m41.821s	0	11	0m12.269s
NAV3	382105	925	39	14	0	115	140m52.498s	0	70	0m11.211s
NCR1	9992	676	7	8	0	24	7m48.169s	0	13	0m6.631s
NEURL2	2664	486	2	2	1	2	3m43.977s	1	2	0m1.578s
NEXN	55384	827	13	8	0	50	96m42.459s	0	25	0m12.403s
NLRP3	32954	1720	9	8	1	34	149m7.406s	0	16	0m17.040s
NLRP9	29972	801	9	2	0	28	57m38.355s	0	17	0m7.757s
NOL4	373854	943	11	14	0	54	193m18.640s	0	21	0m10.937s
NOS1	244030	988	29	6	0	100	444m26.907s	0	36	0m14.952s
NPM3	2090	1425	6	4	2	12	60m21.608s	0	12	0m13.034s
NR1H3	20734	2202	10	35	1	61	237m27.667s	0	19	0m56.088s
NR1I3	8638	544	9	35	1	41	6m43.426s	0	18	0m4.840s
NUF2	89190	1820	14	12	1	135	238m13.160s	0	27	0m38.702s
PAK6	38730	370	11	17	0	30	8m47.389s	0	18	0m4.844s
PCDH7	426392	765	2	8	0	1	47m49.829s	1	2	0m11.320s
PCDHA6	184388	537	4	3	0	3	160m12.540s	0	5	0m7.863s
PHYHD1	21148	1237	13	13	0	73	60m9.497s	0	24	0m17.860s
PLA2G16	43690	951	4	5	0	6	90m21.040s	0	7	0m10.800s
PLXNB1	26199	659	38	19	0	38	11m23.738s	0	54	0m5.358s
PLXND1	51645	1017	36	17	0	67	47m44.096s	0	65	0m9.519s
PPIL1	20199	770	4	2	1	7	38m49.870s	0	7	0m8.499s
PPT2	10241	1935	9	12	1	47	216m38.019s	1	19	0m28.433s
PRRT2	4512	507	4	7	1	7	7m31.058s	1	8	0m2.970s
PTPRR	282772	933	14	11	0	55	149m49.142s	0	27	0m10.768s
PUS3	9738	2077	4	5	2	8	536m24.553s	0	7	0m49.341s
PXMP2	17387	1722	5	5	1	19	193m6.491s	0	9	0m30.372s
RAPSN	11423	512	8	5	2	29	4m59.439s	0	15	0m5.088s
RCAN1	102002	824	4	14	1	10	50m33.095s	0	6	0m8.529s
REEP1	124092	840	7	11	0	17	322m17.919s	0	13	0m12.071s
RPL39L	59962	885	3	3	1	5	23m32.257s	0	5	0m8.885s
RSPH3	27318	884	8	4	1	25	18m7.815s	0	15	0m9.216s
SAMD5	228622	877	2	2	0	1	195m36.046s	0	2	0m15.668s
SEMA3D	191304	887	17	4	1	79	119m43.221s	0	32	0m13.456s
SGCA	11719	1514	10	11	1	66	78m21.343s	1	19	0m23.907s
SGOL2	73776	962	9	6	0	37	67m10.024s	0	17	0m8.949s
SH3GL3	171517	941	9	6	0	32	129m41.836s	0	17	0m10.339s
SIRPG	28629	348	6	7	1	12	2m21.323s	0	12	0m3.235s
SIX5	4443	560	3	4	1	2	11m26.307s	1	5	0m2.446s
SLC14A1	28395	1891	10	21	1	74	370m16.535s	0	20	0m42.943s
SLC16A4	28236	3002	9	10	1	88	1497m37.324s	0	17	2m13.658s
SLC35D2	63006	2045	12	5	1	89	153m16.961s	0	22	1m3.660s
SLC35G1	62091	737	3	6	1	4	92m54.028s	0	5	0m13.538s
SLC46A2	11995	703	4	2	0	7	102m31.448s	0	7	0m6.133s
SLIT1	187884	789	37	7	0	65	39m38.380s	0	71	0m10.291s
SPATA17	240374	925	11	5	0	47	612m44.359s	0	20	0m14.221s
SPATA33	13472	856	3	10	1	4	67m5.466s	1	4	0m9.641s
SPC25	79241	552	7	4	0	21	10m11.026s	0	13	0m5.674s
SPTBN4	110225	844	36	14	0	87	148m19.517s	0	64	0m10.739s

Continua na próxima página

Tabela A.3 – *Continuação da tabela anterior*

					Heurística PASG			Heurística DAG		
Gene	$\|\mathcal{S}\| = \|\mathcal{T}\|$	$\|S\|$	$\|T\|$	isoformas	VP	FP	Tempo	VP	FP	Tempo
SSH3	9161	1855	14	11	1	92	198m34.945s	0	35	0m15.818s
SSPO	57939	705	107	11	0	24	20m41.508s	0	138	0m9.099s
SYN2	187000	549	13	6	0	51	49m45.095s	0	25	0m5.647s
SYNE4	5488	715	8	7	1	26	5m50.446s	0	17	0m4.379s
TEX30	7823	2131	6	7	2	29	228m59.561s	0	12	0m34.699s
THEMIS	210561	769	6	6	0	15	29m45.552s	0	10	0m8.820s
TLL2	149314	637	21	3	0	50	42m25.064s	0	42	0m8.736s
TMEM173	7403	808	8	15	1	22	9m14.073s	0	18	0m5.291s
TMEM178B	406150	856	4	3	0	5	399m53.975s	0	7	0m19.676s
TMEM237	23388	833	12	13	1	50	11m45.093s	0	23	0m8.774s
TMEM255A	52908	754	10	6	0	58	73m18.644s	0	19	0m9.487s
TRIM7	11371	795	7	7	1	30	34m49.053s	0	13	0m8.129s
TRO	10971	1346	13	25	1	37	102m50.266s	0	25	0m9.309s
TTC8	56927	1464	14	17	0	96	241m29.912s	0	26	0m33.853s
TTLL2	34419	717	3	3	0	4	40m2.070s	0	5	0m8.204s
TTLL9	74263	1008	17	11	0	107	63m55.133s	0	32	0m15.068s
TYROBP	3896	1381	5	9	2	16	34m44.513s	0	9	0m16.597s
VIPR1	48276	1290	13	17	1	88	64m14.636s	0	25	0m19.003s
WDR78	112004	694	17	12	0	61	36m29.211s	0	33	0m7.462s
YAE1D1	43946	989	3	6	1	6	19m43.374s	0	5	0m10.324s
ZBTB7C	384081	507	3	28	0	1	39m13.564s	0	5	0m6.341s
ZMYND10	5744	933	12	9	1	58	30m51.328s	0	28	0m6.458s
ZNF16	20532	1580	4	9	1	8	90m46.530s	0	7	0m29.424s
ZNF469	13288	817	2	2	1	1	47m47.026s	1	2	0m10.896s
ZNF610	31535	836	6	9	1	21	19m14.445s	0	10	0m9.596s
ZNF620	12746	976	5	4	1	12	89m24.306s	1	11	0m5.630s
ZNF879	11314	713	5	4	1	13	73m13.796s	0	9	0m4.502s
ZNF98	141468	550	4	5	1	5	37m19.716s	0	7	0m6.829s

Referências Bibliográficas

[1] KISHI, R. M. *Identificação de Genes e o Problema do Alinhamento Spliced Múltiplo.* 2010. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, Curso de Pós-graduação em Computação, 2010. vi, 3, 16

[2] MARTIN, J. A.; WANG, Z. Next-generation transcriptome assembly. *Nature Reviews Genetics*, v. 12, n. 10, p. 671–682, 2011. vi, 18, 19

[3] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Introduction to Algorithms.* 3. ed. MIT press, 2009. 3, 4, 5, 6, 7, 32

[4] LEUNG, J. Y. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, 2004. 4, 6, 7, 8

[5] GAREY, M. R.; JOHNSON, D. S. "Strong" NP-Completeness Results: Motivation, Examples, and Implications. *Journal of the ACM (JACM)*, v. 25, n. 3, p. 499–508, 1978. 5

[6] GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979. 4, 6, 7, 8, 9, 23

[7] PEVZNER, P. *Computational Molecular Biology: An Algorithmic Approach.* MIT Press, 2000. 10

[8] SETUBAL, J.; MEIDANIS, J. *Introduction to Computational Molecular Biology.* PWS-KENT Publishing Company, 1997. 11, 12

[9] NEEDLEMAN, S. B.; WUNSCH, C. D. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of molecular biology*, v. 48, n. 3, p. 443–453, 1970. 13

[10] BROWN, T. A. *Genomes.* 2. ed. Bios Scientific Publishers, 2002. 18

[11] KEREN, H.; LEV-MAOR, G.; AST, G. Alternative splicing and evolution: diversification, exon definition and function. *Nature Reviews Genetics*, v. 11, n. 5, p. 345–355, 2010. 18

[12] TRAPNELL, C.; WILLIAMS, B. A.; PERTEA, G.; MORTAZAVI, A.; KWAN, G.; VAN BAREN, M. J.; SALZBERG, S. L.; WOLD, B. J.; PACHTER, L. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, v. 28, n. 5, p. 511–515, 2010. 17, 18, 21, 41

[13] DE LIMA, L. I. S. *O Problema do Alinhamento de Segmentos.* 2013. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Mato Grosso do Sul, Faculdade de Computação, Curso de Pós-graduação em Computação, 2013. 3, 28, 34, 37

[14] PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial Optimization: Algorithms and Complexity.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982. 4

[15] FOULDS, L. R. *Combinatorial Optimization for Undergraduates.* Springer, 1984. 4

[16] LEVITIN, A.; MUKHERJEE, S.; BHATTACHARJEE, A. K. *Introduction to the Design and Analysis of Algorithms*. Pearson Addison-Wesley, 2007. v. 2. 5

[17] GUSFIELD, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge university press, 1997. 11

[18] DE BRITO, R, T. *Alinhamento de Seqüências Biológicas*. 2003. Dissertação (Mestrado em Ciência da Computação) - Universidade de São Paulo, Instituto de Matemática e Estatística, 2003. 12

[19] BACKURS, A.; INDYK, P. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, 2015. 13

[20] ALBERTS, B.; JOHNSON, A.; LEWIS, J.; MORGAN, D.; RAFF, M.; ROBERTS, K.; WALTER, P. *Molecular Biology of the Cell*. 6. ed. New York, NY: Garland Science, Dec. 2014. 13, 14, 15, 17

[21] GRIFFITHS, A. J. F.; ICLICKER; WESSLER, S. R.; LEWONTIN, R. C.; GELBART, W. M.; SUZUKI, D. T.; MILLER, J. H. *Introdução à Genética*. 8. ed. Guanabara Koogan, 2006. 13, 14, 15, 17

[22] WANG, Z.; GERSTEIN, M.; SNYDER, M. RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, v. 10, n. 1, p. 57–63, 2009. 17, 18, 32

[23] WANG, E. T.; SANDBERG, R.; LUO, S.; KHREBTUKOVA, I.; ZHANG, L.; MAYR, C.; KINGSMORE, S. F.; SCHROTH, G. P.; BURGE, C. B. Alternative isoform regulation in human tissue transcriptomes. *Nature*, London, v. 456, n. 7221, p. 470–476, 2008. 18

[24] LI, J. J.; JIANG, C.; BROWN, J. B.; HUANG, H.; BICKEL, P. J. Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation. *Proceedings of the National Academy of Sciences*, v. 108, n. 50, p. 19867–19872, 2011. 21

[25] MEZLINI, A. M.; SMITH, E. J.; FIUME, M.; BUSKE, O.; SAVICH, G. L.; SHAH, S.; APARICIO, S.; CHIANG, D. Y.; GOLDENBERG, A.; BRUDNO, M. iReckon: Simultaneous isoform discovery and abundance estimation from RNA-seq data. *Genome research*, v. 23, n. 3, p. 519–529, 2013.

[26] CHANG, Z.; LI, G.; LIU, J.; ZHANG, Y.; ASHBY, C.; LIU, D.; CRAMER, C. L.; HUANG, X. Bridger: a new framework for de novo transcriptome assembly using RNA-seq data. *Genome biology*, v. 16, n. 1, p. 30, 2015.

[27] PERTEA, M.; PERTEA, G. M.; ANTONESCU, C. M.; CHANG, T.-C.; MENDELL, J. T.; SALZBERG, S. L. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nature biotechnology*, v. 33, n. 3, p. 290–295, 2015. 21, 39, 42

[28] STEIJGER, T.; ABRIL, J. F.; ENGSTRÖM, P. G.; KOKOCINSKI, F.; HUBBARD, T. J.; GUIGÓ, R.; HARROW, J.; BERTONE, P.; CONSORTIUM, R. et al. Assessment of transcript reconstruction methods for RNA-seq. *Nature methods*, 2013. 21

[29] LU, B.; ZENG, Z.; SHI, T. Comparative study of *de novo* assembly and genome-guided assembly strategies for transcriptome reconstruction based on RNA-Seq. *Science China Life Sciences*, v. 56, n. 2, p. 143–155, 2013.

[30] ANGELINI, C.; CANDITIIS, D. D.; FEIS, I. D. Computational approaches for isoform detection and estimation: good and bad news. *BMC bioinformatics*, v. 15, n. 1, p. 135, 2014. 21

[31] LU, Y.; CHEN, D. Z.; CHA, J. Packing cubes into a cube is NP-complete in the strong sense. *Journal of Combinatorial Optimization*, v. 29, n. 1, p. 197–215, 2015. 23

[32] BRANDES, U.; DELLING, D.; GAERTLER, M.; GÖRKE, R.; HOEFER, M.; NIKOLOSKI, Z.; WAGNER, D. Maximizing Modularity is hard. *arXiv physics/0608255*, 2006.

[33] CABELLO, S. et al. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal of Graph Algorithms and Applications*, v. 10, n. 2, p. 353–363, 2006.

[34] STOCKMEYER, L. Optimal Orientations of Cells in Slicing Floorplan Designs. *Information and Control*, v. 57, n. 2, p. 91–101, 1983.

[35] DEMAINE, E. D.; HOHENBERGER, S.; LIBEN-NOWELL, D. Tetris is Hard, Even to Approximate. In: *Computing and Combinatorics*. Big Sky, MT, USA: Springer, 2003. p. 351–363. 23

[36] GRAHAM, R. L.; KNUTH, D. E.; PATASHNIK, O. *Concrete Mathematics: A Foundation for Computer Science*. Pearson Education India, 1994. 27

[37] ABRAMOWITZ, M.; STEGUN, I. A. et al. Handbook of Mathematical Functions. *Applied Mathematics Series*, v. 55, p. 62, 1966. 27

[38] SEDGEWICK, R.; WAYNE, K. *Algorithms*. 4. ed. Addison-Wesley, 2011. 32

[39] RICE, P.; LONGDEN, I.; BLEASBY, A. et al. EMBOSS: The European Molecular Biology Open Software Suite. *Trends in genetics*, v. 16, n. 6, p. 276–277, 2000. 34

[40] LANDER, E. S.; LINTON, L. M.; BIRREN, B.; NUSBAUM, C.; ZODY, M. C.; BALDWIN, J.; DEVON, K.; DEWAR, K.; DOYLE, M.; FITZHUGH, W. et al. Initial sequencing and analysis of the human genome. *Nature*, London, v. 409, n. 6822, p. 860–921, 2001. 35

[41] GRIEBEL, T.; ZACHER, B.; RIBECA, P.; RAINERI, E.; LACROIX, V.; GUIGÓ, R.; SAMMETH, M. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic acids research*, v. 40, n. 20, p. 10073–10083, 2012. 37

[42] MANGUL, S.; CACIULA, A.; AL SEESI, S.; BRINZA, D.; MANDOIU, I.; ZELIKOVSKY, A. Transcriptome assembly and quantification from Ion Torrent RNA-Seq data. *BMC genomics*, v. 15, n. Suppl 5, p. S7, 2014. 37

[43] LI, W.; FENG, J.; JIANG, T. IsoLasso: a LASSO Regression Approach to RNA-Seq Based Transcriptome Assembly. *Journal of Computational Biology*, v. 18, n. 11, p. 1693–1707, 2011.

[44] FENG, J.; LI, W.; JIANG, T. Inference of Isoforms from Short Sequence Reads. *Journal of computational biology*, v. 18, n. 3, p. 305–321, 2011. 37

[45] TRAPNELL, C.; PACHTER, L.; SALZBERG, S. L. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, v. 25, n. 9, p. 1105–1111, 2009. 39

[46] RODRIGUEZ, J. M.; MAIETTA, P.; EZKURDIA, I.; PIETRELLI, A.; WESSELINK, J.-J.; LOPEZ, G.; VALENCIA, A.; TRESS, M. L. APPRIS: annotation of principal and alternative splice isoforms. *Nucleic Acids Research*, v. 41, n. D1, p. D110–D117, nov 2012. 39