O Problema do Mapeamento de Sequências em Grafos de De Bruijn

Lucas Barbosa Rocha

O Problema do Mapeamento de Sequências em Grafos de De Bruijn[1]

Lucas Barbosa Rocha

Orientador: *Prof. Dr Said Sadique Adi*
Coorientador: *Prof. Dr Francisco Eloi Soares Araujo*

Tese entregue à Faculdade de Computação da Universidade Federal de Mato Grosso do Sul - FACOM-UFMS como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

UFMS - Campo Grande
Agosto/2024

[1]Trabalho Realizado com Auxílio da Capes. Número do processo: 88887.494138/2020-00.

À minha esposa, Jaqueline da Silva Rocha

Aos meus pais,
Zilma Gonçalves Barbosa Rocha e José Aluisio Carinhanha Rocha,

À minha irmã,
Luana Barbosa Rocha,

À minha cachorra Mel Gibson,

In memoriam à minha cachorra Lola.

Agradecimentos

Agradeço aos meus orientadores Francisco Eloi Araujo e Said Sadique Adi pela paciência, motivação, compreensão e por me ajudarem grandemente no desenvolvimento deste trabalho, além de me ajudarem a crescer academicamente, profissionalmente e pessoalmente.

Agradeço à minha esposa, Jaqueline da Silva Rocha, que me ajudou em todos os momentos com suas energias positivas e sábias palavras, e compreendeu e ajudou nos momentos de ansiedade e estresse.

Agradeço à minha família, aos meus pais José Aluisio Carinhanha Rocha e Zilma Gonçalves Barbosa Rocha, e à minha irmã Luana Barbosa Rocha por me apoiarem em todos os momentos e compreenderem todas as vezes que precisei ficar ausente e focar nos estudos.

Agradeço à família da minha esposa, Joel Severino da Silva, Raquel Teixeira da Silva, Matheus da Silva e Nadiely da Silva, que se tornaram parte da minha família no ano de 2023 e me ajudaram com orações.

Agradeço aos amigos que ganhei nos últimos anos: Carlos Córdoba, Denis Novaes, Edison Borghezan, Franklin Messias, Gabriel Escobar, Hernanes Almeida e Jean Carlo.

Agradeço às professoras Edna Ayako, Graziela Araújo e Liana Duenha, e aos professores Edison Takashi, Carlos Higa, Fábio Viduani, Marcelo Carvalho, Ricardo Ribeiro, Marco Aurélio e Renato Ishii pela ajuda e motivação durante as aulas.

Agradeço a Deus pela oportunidade de estar vivo e ter essas pessoas maravilhosas ao meu lado.

Agradeço à CAPES por me disponibilizar uma bolsa de estudos durante dois anos do doutorado, agradeço à Universidade Federal de Mato Grosso do Sul - UFMS por me disponibilizar os ambientes necessários para realizar meus estudos. Agradeço aos funcionários e professores da FACOM que contribuíram de forma direta ou indireta durante o meu trabalho de doutorado.

Enfim, reservo este parágrafo para agradecer a existência da minha cachorra Mel Gibson, que foi o meu apoio emocional nas dificuldades, e dedico também para lembrar da minha cachorra Lola, que faleceu em 2022, e nunca me esquecerei de como ela também foi o meu apoio emocional desde 2013.

Abstract

A relevant problem in Computational Biology consists of the task of mapping one sequence onto another for comparison purposes. Typically, this process utilizes a high-quality reference sequence constructed from a specific set of sequences. However, the limitation of this approach is evident, as the reference sequence tends to be biased, representing only a restricted set of sequences and being incapable of encompassing all possibilities. To mitigate this bias, a good strategy is to represent multiple sequences through more robust structures, such as sequence graphs or De Bruijn graphs, and map sequences onto these graphs. The *sequence graph* is a graph where each vertex is labeled with one or more characters. In the *De Bruijn graph* of order k each vertex is labeled with a distinct sequence of length k, and there is an edge from one vertex to another if and only if there exists a length $k-1$ overlap of the suffix of the first vertex with the prefix of the second vertex. Given as input a sequence s and a sequence (or De Bruijn) graph G, mapping s onto G consists of finding a path p in G such that the induced sequence s' by p is as similar as possible to s. This definition gives rise to the problems addressed in this thesis, namely the Sequence Mapping onto Sequence Graphs problem – PMSG and the Sequence Mapping onto De Bruijn Graphs problem – PMSB. Both problems admit three variants: 1) changes only in the sequence, 2) changes in the graph, and 3) changes in both the sequence and the graph. In this work, we present an in-depth analysis of PMSB. For variant 1, we implement and evaluate exact algorithms that solve it. Furthermore, we propose heuristics for PMSB and conduct comparative tests between the exact algorithms, our heuristics, and those found in the literature. Additionally, we perform a study demonstrating that it is possible to convert a De Bruijn graph into a simple sequence graph, such that all sequences from the De Bruijn graph are also induced in the simple sequence graph. As for variant 2, we address the problem by considering the ability to induce new edges when a k-mer is modified in the De Bruijn graph. This approach makes the problem easier, allowing us to present a novel exact polynomial solution for this variant.

Keywords. Hamming distance, Edit distance, Changes in the sequence, Changes in the graph, De Bruijn graph.

Resumo

Um problema relevante na Biologia Computacional consiste na tarefa de mapear uma sequência em outra, visando a comparação entre elas. Normalmente, esse processo utiliza uma sequência de referência de alta qualidade construída a partir de um conjunto específico de sequências. No entanto, a limitação dessa abordagem é evidente, pois a sequência de referência tende a ser enviesada, representando apenas um conjunto restrito de sequências e sendo incapaz de abranger todas as possibilidades. Para contornar esse viés, uma boa estratégia é representar múltiplas sequências por meio de estruturas mais robustas, como o grafo de sequências ou o grafo de De Bruijn, e mapear sequências nesses grafos. O *grafo de sequência* é um grafo na qual cada vértice é rotulado com um ou mais caracteres. No *grafo de De Bruijn*, de ordem k, cada vértice é rotulado com uma sequência distinta de comprimento k e há uma arco de um vértice para outro vértice se e somente se existe uma sobreposição de comprimento $k-1$ do sufixo do primeiro vértice com o prefixo do segundo vértice. Dadas como entrada uma sequência s e um grafo de sequência (ou De Bruijn) G, mapear s em G consiste em encontrar um percurso p em G tal que a sequência induzia s' por p seja a mais semelhante possível a s. Essa definição dá origem aos problemas abordados nesta tese, a saber o Problema do Mapeamento de Sequências em Grafos de Sequência – PMSG e o Problema do Mapeamento de Sequências em Grafos de De Bruijn – PMSB. Ambos os problemas admitem três variantes: 1) mudanças apenas na sequência, 2) mudanças no grafo e 3) mudanças na sequência e no grafo. Apresentamos neste trabalho uma análise aprofundada do PMSB. Para a variante 1, temos a implementação e avaliação de algoritmos exatos que a resolvem. Propomos, ainda, heurísticas para o PMSB e conduzimos testes comparativos entre os algoritmos exatos, nossas heurísticas e aquelas encontradas na literatura. Além disso, realizamos um estudo demonstrando que é possível converter um grafo de De Bruijn em um grafo de sequência simples, de tal forma que todas as sequências do grafo de De Bruijn também são induzidas no grafo de sequência simples. No que diz respeito à variante 2, abordamos o problema considerando a capacidade de induzir novos arcos quando um k-mer é modificado no grafo de De Bruijn. Essa abordagem torna o problema mais fácil, permitindo-nos apresentar uma solução polinomial exata para essa variante.

Palavras-chave. Distância de Hamming, Distância de edição, Mudanças na sequência, Mudanças no grafo, Grafo de De Bruijn.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO

1

Introdução

Um problema é considerável tratável se ele possui um algoritmo eficiente[1] para resolvê-lo; caso contrário, é considerado como intratável. Na Teoria da Computação, a tratabilidade é frequentemente associada aos algoritmos polinomiais, pois, na prática, muitos algoritmos com tempo de processamento polinomial conseguem realizar suas tarefas dentro de um período razoável. No entanto, em várias aplicações biológicas atuais, quando a quantidade de dados de entrada é muito grande, até mesmo soluções com tempo polinomial podem não ser suficientes, pois demandam um tempo de processamento significativo para processar grandes volumes de dados. Nas últimas décadas, houve um notável crescimento no número de genomas sequenciados devido aos avanços na tecnologia de sequenciamento de DNA [Onimaru and Marcon (2019); Peixoto (2013)]. Quando não dispomos de algoritmos mais rápidos que possam fornecer respostas precisas, recorremos a métodos que oferecem soluções que devolvem respostas boas, como heurísticas. Embora essas abordagens não garantam uma relação direta entre boas respostas e respostas exatas, elas são bastante eficientes e, na prática, costumam fornecer resultados satisfatórios na maioria dos casos [Rocha et al. (2018, 2019)].

A Biologia Computacional é a área que aplica a computação para a solução de problemas da Biologia. Uma tarefa relevante nessa área é mapear uma sequência para outra com o propósito de compará-las com o objetivo de verificar o quão semelhantes elas são. Para isso, é utilizada uma sequência de referência de alta qualidade que representa um conjunto de sequências. A comparação entre duas sequências frequentemente envolve a análise de uma sequência em relação à sequência de referência [Li and Homer (2010); Holley and Melsted (2020)]. Por outro lado, uma sequência de referência é uma sequência enviesada, uma vez que representa um conjunto limitado de sequências, incapaz de abranger todas as possíveis variações. Para contornar esse viés, uma abordagem consiste em representar múltiplas sequências utilizando

[1] um algoritmo é considerado eficiente quando sua complexidade de tempo ou espaço atende a certos critérios estabelecidos.

estruturas mais robustas, como o grafo de sequências [Amir et al. (1997); Rautiainen and Marschall (2017); Jain et al. (2019)] ou o grafo de De Bruijn [De Bruijn (1946); Pevzner et al. (2001); Myers (2005)].

Dado um alfabeto Σ, o *grafo de sequência* é um grafo na qual cada vértice é rotulado com uma sequência de caracteres construídas sobre o alfabeto Σ e o *grafo de sequência simples* é um grafo onde cada vértice é rotulado com exatamente um caractere [Myers (2005)] de Σ . No *grafo de De Bruijn* [De Bruijn (1946); Pevzner et al. (2001)], de ordem k, cada vértice é rotulado com uma sequência distinta de comprimento k de Σ e há um arco de um vértice para outro vértice se e somente se existe uma sobreposição de comprimento $k-1$ do sufixo do primeiro vértice com o prefixo do segundo vértice.

Dado um grafo $G = (V, A)$, onde V é o conjunto de vértices e A é o conjunto de arcos, um passeio p em G é uma sequência de vértices $p = v_1, \ldots, v_n$, tal que para cada par de vértices v_i, v_{i+1} em p existe um arco $(v_i, v_{i+1}) \in A$. Dado um grafo de sequência G, um passeio p em G pode induzir uma sequência s' concatenando os caracteres associados com cada vértice de p. Dito isso, um mapeamento envolvendo uma sequência e um grafo de sequências dá origem ao Problema do Mapeamento de Sequências em Grafos de Sequências – PMSG. Nesse problema, dadas uma sequência s e um grafo de sequências G, o objetivo é encontrar um percurso p em G tal que a distância (edição ou Hamming) entre s a sequência induzida s' por p é mínima.

No contexto do PMSG, o mapeamento é dito exato se a sequência induzida s' é igual à sequência de entrada s, caso contrário, é classificado como aproximado. No mapeamento aproximado, são permitidas mudanças na sequência ou no grafo na determinação do mapeamento. O PMSG foi primeiramente abordado no contexto do mapeamento de um padrão em um hipertexto. O padrão pode ser considerado como uma sequência, enquanto o hipertexto pode ser visto como um grafo de sequências. Nos primeiros trabalhos, o termo "padrão" era utilizado para representar uma cadeia de caracteres. Posteriormente, os trabalhos passaram a usar "sequência" para representar uma cadeia de caracteres. Neste presente trabalho, é utilizado o termo sequência para referir a uma cadeia de caracteres. No trabalho intitulado *Approximate String Matching with Arbitrarty Costs for Text and Hypertext*, Manber e Wu propuseram um algoritmo para realizar o mapeamento aproximado, levando em conta mudanças na sequência, com o objetivo de encontrar essa sequência dentro do hipertexto [Manber and Wu (1992)]. Nesse trabalho, dados uma sequência s e um hipertexto H (representado por um grafo dirigido acíclico), os autores propuseram um algoritmo polinomial para o mapeamento aproximado de s em H, permitindo-se mudanças apenas em s. Para mapeamentos exatos, Akutsu propôs um algoritmo polinomial quando o hipertexto é representado por uma árvore [Akutsu (1993)], e Park e Kim propuseram um algoritmo polinomial quando o hipertexto é um grafo acíclico dirigido [Park and Kim (1995)].

Um dos primeiros artigos que aborda o PMSG com mais detalhes foi escrito por Amir *et al.* e intitula-se *Pattern Matching in Hypertext* [Amir et al. (1997)]. Para o mapeamento aproximado, Amir *et al.* foram os pioneiros na literatura ao identificar uma assimetria na localização das

mudanças. Os autores demostraram que realizar mudanças na sequência não é equivalente que realizar mudanças no hipertexto, ou seja, a quantidade de mudanças efetuadas na sequência pode não ser a mesma no hipertexto. Portanto, eles destacaram a importância de compreender se as mudanças acontecem na sequência ou no hipertexto.

Considerando a assimetria identificada por Amir *et al.*, o problema do mapeamento aproximado de uma sequência em um hipertexto admite três variantes:

- variante 1: permite mudanças apenas na sequência ao realizar o mapeamento da sequência no hipertexto;
- variante 2: permite mudanças apenas no hipertexto ao realizar o mapeamento da sequência no hipertexto;
- variante 3: permite mudanças na sequência e no hipertexto ao realizar o mapeamento da sequência no hipertexto.

Para a variante 1, Amir *et al.* propuseram um algoritmo polinomial de tempo $O(m(|V| \cdot \log|V| + |A|))$ que posteriormente foi melhorado por Navarro para $O(m(|V| + |A|))$. Esse aprimoramento foi detalhado no artigo intitulado *Improved Approximate Pattern Matching on Hypertext* [Navarro (1998)]. Neste contexto, $|V|$ é a quantidade de vértices, $|A|$ é a quantidade de arcos e m é o comprimento da sequência mapeado, sendo a distância de edição utilizada durante o mapeamento. Para as variantes 2 e 3, Amir *et al.* provaram que elas são NP-completas considerando a distância de Hamming e edição, quando o alfabeto Σ é limitado.

Posteriormente, o `PMSG` restrito para grafos de sequências simples foi abordado no artigo intitulado *Aligning Sequences to General Graphs in $O(|V| + m|A|)$ Time* [Rautiainen and Marschall (2017)]. Nesse trabalho, Rautiainen e Marschall propuseram um algoritmo polinomial considerando a distância de edição para a variante 1, que foi originalmente proposta por Amir *et al.*. Os autores mapeiam uma sequência em um grafo de sequências simples em tempo $O(|A| \cdot m + |V| \cdot m \cdot \log(|A| \cdot m))$ e nos referimos aqui como algoritmo do `PMSG`. Adicionalmente, eles também propuseram uma versão mais eficiente do algoritmo que devolve apenas a distância em tempo $O(|V| + m|A|)$, na qual nos referimos neste trabalho como o algoritmo do $\texttt{PMSG}_d$.

No trabalho intitulado *On the Complexity of Sequence-to-Graph Alignment* [Jain et al. (2019)], Jain *et al.* propuseram uma modificação para a variante 1 do problema, aprimorando o algoritmo de Rautiainen e Marschall ao considerar a possibilidade de deletar o prefixo da sequência mapeada. Eles conseguiram manter a complexidade de tempo em $O(|V| + m \cdot |A|)$. Além disso, os autores demonstraram que, para grafos de sequência com um ou mais caracteres em seus vértices e com um alfabeto Σ de tamanho $|\Sigma| \geq 2$, as variantes 2 e 3 são NP-completas, tanto para a distância de Hamming quanto para a distância de edição.

A primeira vez que o `PMSG` foi abordado usando um grafo de De Bruijn como entrada foi no artigo intitulado *Read mapping on de Bruijn graphs* [Limasset et al. (2016)]. Nesse trabalho, Limasset *et al.* propõem o seguinte problema, chamado aqui o Problema do Mapeamento de Sequências em Grafos de De Bruijn – `PMSB`: Dadas uma sequência s e um grafo de De Bruijn G_k, o objetivo é determinar um caminho (passeio que não repete vértices) p em G_k tal

que a sequência s' induzida por p tenha no máximo d diferenças entre s e s' com $d \in \mathbb{N}$. O PMSB foi provado ser NP-completo considerando a distância de Hamming. Os autores desenvolveram então uma heurística para o problema baseada no método *seed-and-extend* (semear-e-estender, em tradução livre). Observe que, nesta versão do PMSB, não é pertinente abordar as três variantes previamente mencionadas, uma vez que não há repetições de vértices em caminhos. Essas variantes tornam-se relevantes quando consideramos passeios, permitindo que um mesmo vértice seja incluído múltiplas vezes durante as alterações. Em outras palavras, um vértice modificado pode ser reaproveitado e submetido a modificações subsequentes. Contudo, nos caminhos em questão, cada vértice é utilizado apenas uma única vez.

Posteriormente, o PMSB foi abordado para passeios no artigo intitulado *On the Hardness of Sequence Alignment on De Bruijn Graphs* [Gibney et al. (2022)]. Aluru *et al.* provaram que o problema é NP-completo para a variante 2 (quando as mudanças ocorrem no grafo) e eles também provaram que a menor complexidade de tempo é $O(|A| \cdot m)$ para a variante 1 (quando as mudanças ocorrem apenas na sequência s) na qual $|A|$ é a quantidade de arcos no grafo e m é o comprimento de s.

Dada a importância prática do Problema do Mapeamento de Sequências em Grafos de Sequências – PMSG e a ampla utilização do grafo de De Bruijn, o objetivo principal deste trabalho é realizar um estudo aprofundado de abordagens para o Problema do Mapeamento de Sequências em Grafos de De Bruijn – PMSB, com foco em passeios, considerando apenas as duas primeiras variantes: *mudanças na sequência e mudanças no grafo*. Para a variante 1, desenvolvemos um algoritmo exato que resolve o PMSB com mudanças na sequência, além de propor e implementar heurísticas. Realizamos testes comparativos entre os algoritmos exatos, as heurísticas implementadas neste trabalho e heurísticas presentes na literatura. Faz parte também dos objetivos deste trabalho um estudo da variante 2, inspirada no estudo realizado pelos autores Gibney *et al.* [Gibney et al. (2022)], onde as mudanças ocorrem no grafo. Para a variante 2, apresentamos o problema considerando a característica dos arcos induzidos pelos vértices em grafos de De Bruijn. Dada essa característica, definimos o problema com mudanças no grafo considerando arcos induzidos, o que permitiu a aplicação de um algoritmo polinomial. Além disso, buscamos realizar uma análise teórica da conversão do grafo de De Bruijn em um grafo de sequência simples, com o objetivo de comprovar que essa conversão resulta em um grafo de sequência simples.

1.1 Contribuições

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

- a prova de que um grafo de De Bruijn pode ser convertido em um grafo de sequência simples e um algoritmo polinomial que faz essa conversão;
- (variante 1) a implementação de um algoritmo exato para o PMSB, sendo uma adaptação do algoritmo do PMSG e a implementação de uma heurística para essa versão;

- (variante 1) a implementação de duas heurísticas para o Problema do Mapeamento de Sequências em Grafos de De Bruijn utilizando o método *seed-and-extend*;
- (variante 1) a implementação de um algoritmo exato para o PMSB, sendo uma adaptação do algoritmo do $\texttt{PMSG}_d$ e a implementação de uma heurística para essa versão;
- (variante 2) a definição do problema, considerando a característica dos arcos induzidos pelos vértices em grafos de De Bruijn. Demonstramos que esta versão admite um algoritmo polinomial. Vale destacar que esta formulação do problema considerando arcos induzidos é tratada pela primeira vez na literatura neste trabalho.
- a disponibilização dos códigos no Github: github.com/LucasBarbosaRocha;
- a submissão, o aceite e a apresentação online de um artigo para a publicação na vigésima terceira edição do evento internacional *the International Conference on Computational Science and its Applications* – ICCSA, intitulado *Heuristics for the De Bruijn Graph Sequence Mapping Problem* [Rocha et al. (2023)];
- a submissão de um artigo (em *pre-print*) no biorxiv, intitulado *The De Bruijn Graph Sequence Mapping Problem with Changes in the Graph* [Rocha et al. (2024)].

1.2 Organização da tese

Esta tese está estruturada em sete capítulos subsequentes, além da introdução, organizados da seguinte maneira: O Capítulo 2 aborda conceitos computacionais e definições relevantes. No Capítulo 3, realizamos uma revisão bibliográfica do problema abordado neste estudo, bem como suas variantes. No Capítulo 4, apresentamos como converter um grafo de De Bruijn em um grafo de sequência simples e a equivalência entre esses dois grafos. O Capítulo 5 explora abordagens relacionadas à primeira variante (mudanças na sequência) do problema em análise. No Capítulo 6, oferecemos um estudo sobre a segunda variante (mudanças no grafo) do problema em questão e a apresentação de um algoritmo polinomial. O Capítulo 7 descreve os testes conduzidos, apresentando tabelas e gráficos detalhados que elucidam os resultados obtidos. Por fim, no Capítulo 8, apresentamos as conclusões finais e delineamos possíveis áreas a serem exploradas em futuras pesquisas.

CAPÍTULO

2

Conceitos Preliminares

Para um melhor entendimento dos problemas tratados neste trabalho, é necessário conhecer alguns conceitos preliminares descritos neste capítulo. Na Seção 2.1 abordamos conceitos sobre sequências, na Seção 2.2 apresentamos o conceito de distância de edição e de Hamming e na Seção 2.3 abordamos conceitos relacionados a grafos. Na Seção 2.4 abordamos a assimetria das operações de edição no mapeamento de sequência envolvendo duas sequências e envolvendo uma sequência e um grafo. E finalmente, na Seção 2.5 descrevemos um arcabouço para o desenvolvimento de heurísticas denominado *seed-and-extend*.

2.1 Sequências

Um **alfabeto** Σ é um conjunto finito de elementos chamados **símbolos** ou **caracteres**. Uma **cadeia**, **sequência** ou **padrão** s construído sobre Σ é uma lista finita de símbolos de Σ, denotada aqui por $s = s[1]s[2]\dots s[n]$, com $s[i]$ representando o i-ésimo símbolo de s, com $1 \leq i \leq n$. O **comprimento** de s, denotado por $|s|$, é n e corresponde à quantidade de símbolos que ela possui. A **sequência vazia** ou **cadeia vazia**, denotada por ϵ, é a sequência tal que $|\epsilon| = 0$. Para um $k \in \mathbb{N}$, o conjunto Σ^k é definido como o conjunto de todas as sequências de comprimento k construídas sobre Σ. O conjunto $\Sigma^* = \cup_{k \geq 0} \Sigma^k$ é então o conjunto de todas as sequências construídas sobre Σ.

Dada uma sequência $s = s[1]s[2]\dots s[n]$, a sequência $s[i]s[i+1]\dots s[j]$, com $1 \leq i, j \leq |s|$, é dita uma **subcadeia** de s, sendo aqui denotada por $s[i, j]$. Portanto, uma subcadeia de uma sequência s é um trecho de s com zero ou mais caracteres consecutivos dessa sequência. Se $i > j$, a subcadeia corresponde à cadeia vazia ϵ. Uma subcadeia de comprimento k de uma sequência s é denominada de k**-mer** de s. Dado um i tal que $1 \leq i \leq |s|$, a subcadeia $x = s[1, i]$ (respectivamente $s[i, |s|]$) é chamada de **prefixo** (respectivamente **sufixo**) de s e x é **prefixo próprio** (**sufixo próprio**) de s se $x \neq \epsilon$ e $x \neq s$. Dadas duas sequências distintas s e t, tais que

$s = uw$ (u é um prefixo e w é um sufixo de s) e $t = wv$ (w é um prefixo e v é um sufixo de t), a subcadeia w de s e t é dita uma **sobreposição** dessas sequências. Para duas sequências s e t, denotamos por st a **concatenação** de s e t, que consiste em unir essas duas sequências em uma única sequência contínua, colocando todos os caracteres de t, na sua ordem original, ao final de s.

Para exemplificar as definições acima, considere $s = \texttt{AACCCAACAC}$ e $t = \texttt{AACACAACAC}$ duas sequências construídas sobre o alfabeto $\Sigma = \{\texttt{A}, \texttt{C}\}$ e $k = 2$. Logo, $|s| = 10$, $s[1] = \texttt{A}$, $s[5] = \texttt{C}$, $\Sigma^k = \{\texttt{AA}, \texttt{AC}, \texttt{CA}, \texttt{CC}\}$, a subcadeia $s[2, 3] = \texttt{AC}$ é um k-mer de comprimento 2, $s[1, 5] = \texttt{AACCC}$ é um prefixo próprio de s, $s[7, 10] = \texttt{ACAC}$ é um sufixo próprio de s, $w = \texttt{AACAC}$ é uma sobreposição de s e t de comprimento 5 e st = AACCCAACACAACACAACAC é a concatenação de s e t.

2.2 Distâncias

A **distância** entre duas sequências s e t, $d(s, t)$, é uma medida que atribui um número não negativo ao par s e t de forma a mensurar quão similares são essas sequências. Distância precisa satisfazer as seguintes propriedades: $d(s, t) = 0$ se, e somente se, $s = t$; $d(s, t) = d(t, s)$; e dada uma terceira sequência u, a distância deve respeitar a desigualdade triangular $d(s, t) \leq d(s, u) + d(u, t)$. A distância pode representar, por exemplo, o número mínimo de operações necessárias para transformar uma sequência na outra. Dentre as medidas de distância existentes, destacam-se a de edição e a de Hamming que são utilizadas neste trabalho.

A **distância de edição** foi descrita pelo matemático soviético Vladimir Levenshtein [Levenshtein (1966)]. Dadas duas sequências s e t, a distância de edição representa o número mínimo de **operações de edição** (inserção, remoção e substituição de caracteres) para transformar s em t. Denotamos a distância de edição entre s e t como $d_E(s,\ t)$, onde $n = |s|$ e $m = |t|$. Essa medida pode ser calculada de forma recursiva usando a seguinte relação:

$$d_E(s,t) = \begin{cases} n + m, & \text{se } n = 0 \text{ ou } m = 0, \\ \min \left\{ \begin{array}{l} d_E(s[1,n], t[1,m-1]) + 1 \\ d_E(s[1,n-1], t[1,m]) + 1 \\ d_E(s[1,n-1], t[1,m-1]) + s[n] \stackrel{?}{=} t[m] \end{array} \right\} & \text{caso contrário,} \end{cases}$$

onde $s[n] \stackrel{?}{=} t[m]$ é 1 se $s[n] \neq t[m]$, e 0 caso contrário, que representa o custo de substituir o caractere $s[n]$ por $t[m]$, e vice-versa. O custo de inserir ou deletar um caractere é 1. Com base na recorrência mencionada e utilizando programação dinâmica com uma matriz de tamanho n por m, é possível calcular essa distância em tempo $O(nm)$. Apresentamos na Figura 2.1 um exemplo do cálculo da distância de edição para duas sequências.

A **distância de Hamming** foi estabelecida pelo matemático estadunidense Richard Hamming [Hamming (1950)] e é a distância que considera a operação de substituição como única operação de edição. Isso implica que é possível **calcular** essa medida somente quando as

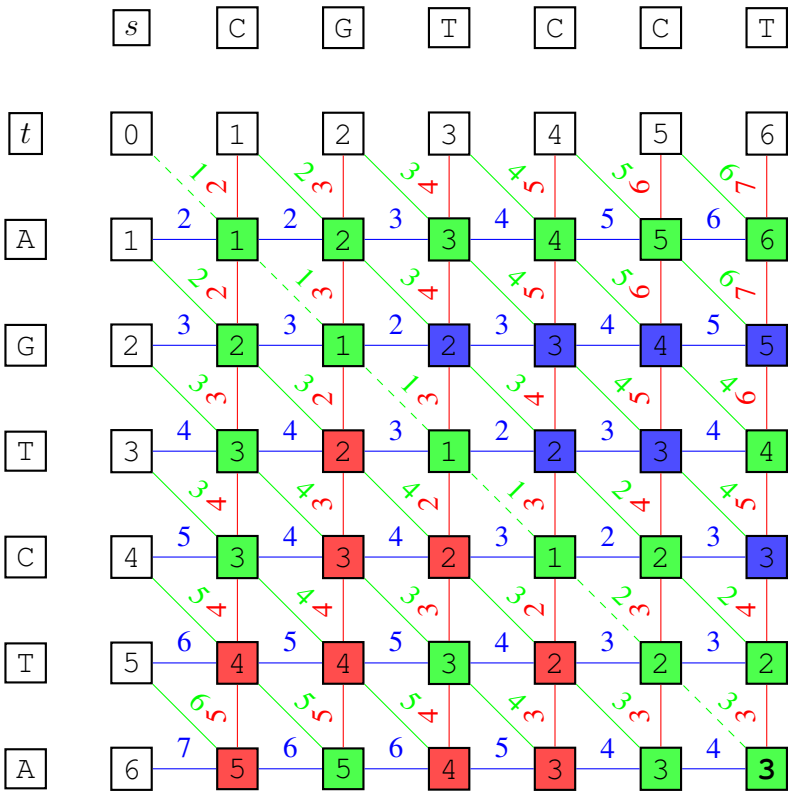


Figura 2.1: Exemplo do cálculo da distância de edição entre as sequências $s = \texttt{CGTCCT}$ e $t = \texttt{AGTCTA}$, sendo $d_E(s,\ t) = 3$. As linhas vermelhas, verdes e azuis representam os custos de inserção, substituição e deleção, respectivamente. Neste exemplo, durante o cálculo da distância, a linha vermelha denota o custo de inserção do caractere $s[i]$, a linha verde denota o custo de substituição do caractere $t[j]$ pelo caractere $s[i]$, e a linha azul denota o custo de deleção do caractere $t[j]$. As cores nos quadrados da matriz representa de qual linha o valor veio.

duas sequências consideradas possuem o mesmo comprimento. Formalmente, tomando duas sequências s e t de comprimento n, a distância de Hamming $d_H(s,t)$ entre s e t é definida como

$$d_H(s,t) = \sum_{i=1}^{n} s[i] \stackrel{?}{=} t[i],$$

onde $s[i] \stackrel{?}{=} t[i]$ é 1 se $s[i] \neq t[i]$, e 0 caso contrário. Isso representa o custo de substituir o caractere $s[i]$ por $t[i]$, e vice-versa. Essa medida pode ser calculada facilmente em tempo $O(n)$. Como exemplo, considere as sequências $s = \texttt{TGGTCT}$ e $t = \texttt{ACGTCT}$. Apresentamos na Figura 2.2 o cálculo da distância de Hamming entre s e t.

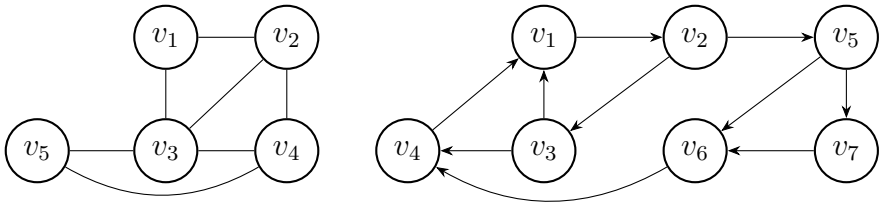
T	G	G	T	C	T
A	C	G	T	C	T
1	1	0	0	0	0

Figura 2.2: Exemplo de cálculo da distância de Hamming entre as sequências $s = \texttt{TGGTCT}$ e $t = \texttt{ACGTCT}$, ambas de mesmo comprimento, onde $d_H(s,t) = 2$. Essa distância é igual a 2 devido às diferenças nos locais $s[1] \neq t[1]$ e $s[2] \neq t[2]$, enquanto os caracteres restantes são idênticos entre s e t.

2.3 Grafos

Um grafo G é definido como um par ordenado (V, A), composto por dois conjuntos, em que V representa um conjunto de elementos chamados **vértices** (do grafo), enquanto que A representa um conjunto de elementos denominados **arestas** ou **arcos** (do grafo). Cada aresta de um grafo corresponde a um par não ordenado de vértices, enquanto que cada arco corresponde a um par ordenado de vértices desse mesmo grafo. Quando os grafos possuem apenas arcos, eles são denominados grafos **dirigidos** ou **digrafos**. Por outro lado, quando contêm apenas arestas, são ditos grafos **não dirigidos**. Por exemplo, $G = (\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}, \{(v_1, v_2), (v_2, v_3), (v_2, v_5), (v_3, v_1), (v_3, v_4), (v_4, v_1), (v_5, v_6), (v_5, v_7), (v_6, v_4), (v_7, v_6)\})$ é um grafo dirigido, enquanto que $G' = (\{v_1, v_2, v_3, v_4, v_5\}, \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_1\}, \{v_4, v_2\}, \{v_4, v_3\}, \{v_5, v_3\}, \{v_5, v_4\}\})$ é um grafo não dirigido.

Os grafos recebem esse nome pois podem ser representados graficamente por meio de círculos representando os seus vértices e semirretas representando suas arestas. No caso dos digrafos, as semirretas representativas de seus arcos possuem uma seta indicando a sua direção. Dados dois vértices u e v de um grafo não dirigido G, eles são ditos **adjacentes** se há uma aresta entre eles. Por outro lado, em um grafo dirigido G', u e v são ditos **adjacentes** se houver um arco direcionado de u para v ($u \to v$). Nesse caso, v é adjacente a u, enquanto que u não é necessariamente adjacente a v. Dado um vértice v em um grafo não dirigido G, o **grau** de v é o número de arestas que incidem em v. Dado um vértice v de um grafo dirigido G', o **grau de entrada** (=*in-degree*) $ge(v)$ de v é o número de arcos que incidem em v, ou seja, é o número de arcos direcionados para v. Uma representação gráfica de um grafo não dirigido G e de um grafo dirigido G' pode ser visualizada na Figura 2.3, juntamente com exemplos de adjacência e grau de entrada.



Grafo não dirigido G Grafo dirigido G'

Figura 2.3: Exemplos de grafos. Em G, os vértices v_2 e v_1 são adjacentes um ao outro, enquanto em G', v_1 é adjacente a v_2, mas não o contrário. No grafo G', o grau de entrada de v_3 é $ge(v_3) = 1$. No grafo G o grau de v_4 é 3.

A cada aresta $\{u, v\}$ de um grafo não dirigido G (ou arco (u, v) de um grafo dirigido G') pode estar associado um número real por meio de uma função $c(\{u, v\}) \to \mathbb{R}$ (ou $c((u, v)) \to \mathbb{R}$). Para custo de arcos pode ser indicado de forma mais simples como $c(u, v)$, desde que não haja ambiguidade. Esse número associado a $\{u, v\}$ (ou (u, v)) é denominado **custo** da aresta $\{u, v\}$ (ou arco (u, v)). Na representação gráfica de um grafo com custos nas arestas (ou arcos), esse custo é exibido junto com a aresta (ou arco) correspondente. Na Figura 2.4, é mostrado um grafo com custos nos arcos.

2.3.1 Percursos em grafos

Dado um grafo $G = (V, A)$, um **passeio** em G é uma sequência de vértices $p = v_1, \ldots, v_k$, tal que para cada par de vértices v_i, v_{i+1} em p existe um arco $(v_i, v_{i+1}) \in A$. Um passeio é **fechado** se o seu primeiro vértice coincide com o último. Um **ciclo** é um passeio fechado sem arcos e vértices (exceto o primeiro e o último) repetidos. Grafos dirigidos que não possuem ciclos são denominados grafos **acíclicos** – DAG (*Directed Acyclic Graph*). Um **caminho** em G é um passeio que não repete vértices. Um **caminho Hamiltoniano** em G é um caminho que visita todos os vértices de G. Dado um passeio p com n vértices em um grafo G, o **comprimento** de p é $n - 1$. Dados dois passeios $p = u_1, \ldots, u_n$ e $p' = v_1, \ldots, v_m$ em um grafo G, tal que (u_n, v_1) é um arco do grafo, $pp' = u_1, \ldots, u_n, v_1, \ldots, v_m$ é a **concatenação dos passeios** de p e p'. Veja exemplos desses conceitos na Figura 2.4.

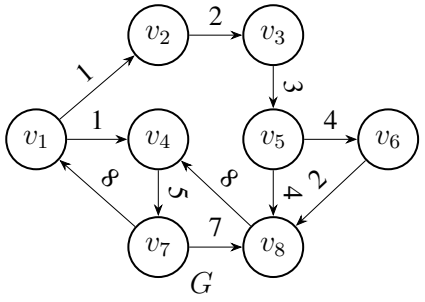


Figura 2.4: Exemplo de um grafo G com custos nos arcos. Aqui, $v_1, v_4, v_7, v_8, v_4, v_7, v_1$ forma um passeio de comprimento 6 no grafo, enquanto que v_1, v_4, v_7, v_8 é um caminho de comprimento 3. A sequência de vértices $v_1, v_2, v_3, v_5, v_6, v_8, v_4, v_7$ é um caminho Hamiltoniano de comprimento 8 e v_4, v_7, v_8, v_4 é um ciclo de comprimento 3 no grafo. Os passeios $p = v_1, v_4, v_7, v_1$ e $p' = v_2, v_3, v_5$ podem ser concatenados resultando no passeio $pp' = v_1, v_4, v_7, v_1, v_2, v_3, v_5$ no grafo.

Dado um grafo G com custos nos arcos, o **custo** de um caminho $p = v_1, \ldots, v_n$ em G é determinado pela função $c(p) = \sum_{j=1}^{n-1} c(v_j, v_{j+1})$, ou seja, o custo do caminho p é a soma dos custos dos arcos de p. Para grafos com custos associados aos arcos, um caminho de custo mínimo ou simplesmente **caminho mínimo**, de s até t é aquele cujo custo é mínimo. Se não existir um caminho de u até v o custo do caminho é infinito. Por exemplo, para o grafo G da Figura 2.4, o caminho mínimo de v_1 até v_8 é $p = v_1, v_2, v_3, v_5, v_8$ de comprimento 4 e $c(p) = 10$, embora exista o caminho v_1, v_4, v_7, v_8, que é mais curto em termos de número de vértices percorridos, mas seu custo total é 13. O problema do Caminho Mínimo pode ser definido da seguinte forma:

Problema 1. *Dados um grafo G com custo nos arcos e dois vértices u e v de G, encontrar um caminho mínimo de u até v.*

Esse problema encontra-se bem estudado na literatura e pode ser resolvido em tempo $O(|A| \cdot \log |V|)$ utilizando o algoritmo de Dijkstra [Dijkstra (1959)] no qual A é o conjunto de arcos e V o de vértices. O algoritmo de Dijkstra é utilizado para calcular o caminho mínimo entre vértices de um grafo. Ele se inicia escolhendo um vértice como origem, marcando-o com distância zero e os demais com distância infinita, além de marcá-los como não visitados. A cada iteração,

seleciona-se o vértice não visitado com a menor distância em relação ao vértice de origem, recalculando as distâncias para seus vizinhos. Se a nova distância para cada vizinho for menor, ela é atualizada. Esse processo se repete até a visita de todos os vértices. É importante notar que o algoritmo não suporta arcos com custos negativos.

Ainda sobre o algoritmo de Dijkstra, há uma variante dele, chamada *Optimized Dijkstra for small range weights*, que se refere a uma otimização aplicada ao algoritmo de Dijkstra em cenários nos quais os pesos dos arcos estão restritos a um intervalo pequeno (limitados por um parâmetro W). O algoritmo de Dial [Dial (1969)] incorpora essa otimização, e essa versão tem uma complexidade de tempo de $O(|V| \cdot W)$, em que V representa o conjunto de vértices.

2.3.2 Grafos bipartidos e emparelhamento

Um grafo não dirigido $H = (V, A)$ é dito um **grafo bipartido** $H = (V_1 \cup V_2, A)$ se existe uma bipartição dos seus vértices V em dois conjuntos disjuntos V_1 e V_2 e $V = V_1 \cup V_2$, de modo que para cada aresta $\{u, v\} \in A$, $u \in V_1$ e $v \in V_2$ ou vice-versa. Um grafo bipartido é dito **grafo bipartido completo** se para quaisquer dois vértices, $v_1 \in V_1$ e $v_2 \in V_2$, $\{v_1, v_2\}$ é uma aresta em G.

Um **emparelhamento** E em H consiste em um conjunto de arestas de H que não compartilham vértices. O **custo de um emparelhamento** E é $C(E) = \sum_{e \in E} c(e)$ tal que $c(e)$ é o custo de uma aresta $e = \{x, y\} \in A$. Um emparelhamento é dito de **custo mínimo** se não existe um emparelhamento com custo menor. Um emparelhamento E é **máximo** se não existe outro maior, ou seja, se não existe um emparelhamento E' tal que $|E'| > |E|$. Dado um conjunto E com todos os emparelhamentos máximos, um emparelhamento $E \in$ E é dito **máximo e de custo mínimo** se não existe um emparelhamento $E' \in$ E tal que $C(E') < C(E)$.

Um exemplo de grafo bipartido completo com custos nas arestas $H = (V_1 \cup V_2, A)$ é mostrado na Figura 2.5, considerando $V_1 = \{v_1, v_2, v_3\}$ e $V_2 = \{v_4, v_5, v_6\}$. Para o grafo bipartido da Figura 2.5, um exemplo de emparelhamento E máximo e de custo mínimo é $E = \{\{v_1, v_5\}, \{v_2, v_6\}, \{v_3, v_4\}\}$ (arestas pontilhadas), de custo 4.

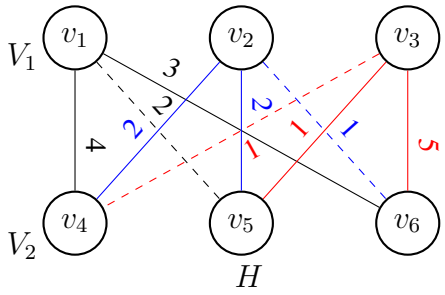


Figura 2.5: Exemplo de grafo bipartido completo H com custo nas arestas.

O problema do emparelhamento de custo mínimo em grafos bipartido é definido da seguinte forma:

Problema 2. *Dado um grafo bipartido $H = (V_1 \cup V_2, A)$, encontrar um emparelhamento E de custo mínimo em H.*

Esse problema encontra-se amplamente estudado na literatura e pode ser resolvido utilizando, por exemplo, o algoritmo de Edmonds (com complexidade de tempo $O((|V_1|+|V_2|)^3)$), Hopcroft-Karp (com complexidade de tempo $O(\sqrt{(|V_1|+|V_2|)\times|A|})$ e o Húngaro (com complexidade de tempo $O((|V_1|+|V_2|)^3)$) [Edmonds (1965); Hopcroft and Karp (1973); Kuhn (1955)].

Para uma melhor compreensão do algoritmo Húngaro, que corresponde a um dos algoritmos utilizado nesse trabalho, considere as seguintes definições para um grafo bipartido $H = (V_1 \cup V_2, A)$. Dado um vértice v de H e um emparelhamento E em H, v é dito E**-saturado** se alguma aresta de E incide em v. Dado um conjunto qualquer X de vértices, considere $\Gamma(X)$ como o conjunto de todos os vértices adjacentes a algum vértice em X. Um caminho no grafo bipartido H com um emparelhamento E é dito E**-alternante** se suas arestas estão alternadamente em E e em $A\backslash E$ (ou alternadamente em $A\backslash E$ e em E). Finalmente, um caminho E**-aumentador** é um caminho E-alternante, de comprimento maior do que zero, cujos extremos não estão saturados por E. Considere, sem perda de generalidade, que $|V_1| \leq |V_2|$. Note que todo emparelhamento E em H possui, no máximo, $|V_1|$ arestas. Assim, o problema de encontrar um emparelhamento máximo em H pode ser visto como o problema de encontrar um emparelhamento que sature todos os vértices de V_1, se existir. Portanto, H contém um emparelhamento que satura todos os vértices de V_1 se e somente se $|\Gamma(S)| \geq |S|$ para todo $S \subseteq V_1$. Na Figura 2.6, são mostrados exemplos destas definições em um grafo não dirigido H.

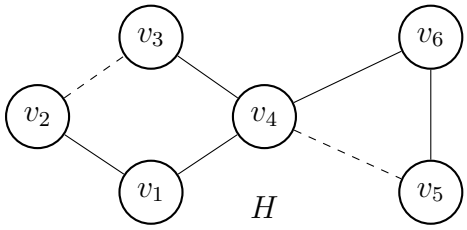


Figura 2.6: Exemplo de vértice E-saturado e caminhos E-alternante e E-aumentador em um grafo não dirigido H. Dado o emparelhamento $E\{\{v_2, v_3\}, \{v_4, v_5\}\}$ em H, v_2 é E-saturado. O caminho v_1, v_2, v_3, v_4, v_5 é um caminho E-alternante em H, enquanto que $v_1, v_2, v_3, v_4, v_5, v_6$ é um caminho E-aumentador em H.

Com base nessas definições e considerando um grafo bipartido $H = (V_1 \cup V_2, A)$, o Algoritmo Húngaro encontra um emparelhamento que satura todos os vértices de V_1, caso tal emparelhamento exista. Os passos do algoritmo são descritos a seguir.

1. Inicialize um conjunto E com uma aresta qualquer de H;
2. Se E satura todos os vértices de V_1, o algoritmo para. Caso contrário, escolha um vértice $u \in V_1$ não E-saturado e defina os conjuntos $S = \{u\}$ e $T = \emptyset$;
3. Se $\Gamma(S) = T$, o algoritmo para; caso contrário, escolha um vértice $v \in \Gamma(S)\backslash T$;
4. Se v é E-saturado, selecione a aresta $\{v, z\} \in E$, adicionando z em S e v em T, e retorne ao passo 3; caso contrário, atualize E com a diferença simétrica[1] entre E e C, onde C é um caminho E-aumentador entre u e v, e retorne ao passo 2.

[1]Conjunto de elementos que estão em um dos conjuntos, mas não em sua interseção.

Caso o algoritmo pare no passo 3, para encontrar um emparelhamento máximo, basta executar o algoritmo Húngaro para todos os vértices em V_1 não saturados e que ainda não foram examinados. Ainda sobre o algoritmo Húngaro, há uma variante que considera os custos nas arestas e devolve o emparelhamento máximo de custo mínimo [Schwartz et al. (2005); Karleigh et al. (2022)], mantendo a complexidade de tempo em $O((|V_1| + |V_2|)^3)$.

2.3.3 Grafos de sequência

Um **grafo de sequência** G é um grafo $G = (V, A)$ com uma sequência de caracteres, construída sobre um alfabeto Σ, associada a cada um dos seus vértices [Amir et al. (1997); Braga et al. (2000)]. Uma sequência de símbolos associada a um vértice de um grafo de sequência é denominada **rótulo** do vértice. Um **grafo de sequência simples** é um grafo na qual cada nó é rotulado com apenas um caractere [Amir et al. (1997); Rautiainen and Marschall (2017); Jain et al. (2019)]. Um exemplo de grafo de sequência (simples) é apresentado na Figura 2.7a.

Dado um passeio $p = v_1, v_2, \ldots, v_n$ em um grafo de sequência G, a **sequência induzida** por p é determinada pela concatenação de cada sequência associada a cada vértice de $p = v_1, v_2, \ldots, v_n$ na ordem que os vértices aparecem no caminho. Por exemplo, no grafo G da Figura 2.7a, a sequência induzida por $p = v_1, v_2, v_3, v_5, v_6, v_7, v_4$, de comprimento 6, é `ACGTCGT`.

Dados um conjunto de sequências $S = \{r_1, \ldots, r_m\}$ construídas sobre um alfabeto Σ e um inteiro $k \geq 2$, um **grafo de De Bruijn** [De Bruijn (1946)] de ordem k, é um grafo $G_k = (V, A)$ tal que

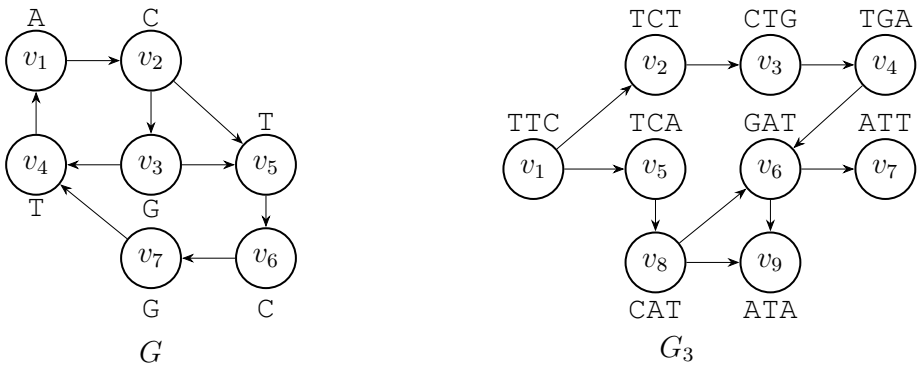
- $V = \{d \in \Sigma^k | d$ é uma subcadeia de comprimento k (k-mer) de $r \in S\}$;
- $A = \{(d, d') |$ o sufixo de comprimento $k-1$ de d é um prefixo de $d'\}$.

Observe que, em um grafo de De Bruijn, os arcos são **induzidos** pelo conjunto de vértices e pelo parâmetro k, portanto, podemos definir um grafo de De Bruijn G_k apenas pelo seu conjunto de vértices. Como exemplo, considere o conjunto $S = \{$`TTCTG`, `TGATA`, `TCATA`$\}$ e $k = 3$. Um exemplo de um grafo de De Bruijn G_3 juntamente com seus arcos induzidos para o conjunto S é apresentado na Figura 2.7b.

Dado um passeio $p = v_1, v_2, \ldots, v_n$ em um grafo de De Bruijn, a sequência induzida por p é $v_1[1, k]v_2[k] \ldots v_n[k]$ dada pela concatenação do k-mer v_1 com o último caractere de cada k-mer $v_2, \ldots, v_n$. Por exemplo, no grafo G_3 da Figura 2.7b, a sequência induzida por $p = v_1, v_2, v_3, v_4$ é `TTCTGA`.

2.4 Mapeamento e assimetria das operações de edição

Um **mapeamento entre duas sequências**, s e t, consiste em comparar essas sequências com o objetivo de verificar o quão semelhantes elas são. Essa comparação é realizada por meio de



(a) Exemplo de um grafo de sequência simples. (b) Exemplo de um grafo de De Bruijn com $k = 3$.

Figura 2.7: Exemplos de grafos de sequência.

uma métrica de distância, tal como a distância de edição ou a distância de Hamming, conforme mencionado anteriormente. Quando se realiza a comparação entre as sequências considerando operações de edição, surge uma propriedade de simetria, onde a mesma quantidade de alterações pode ser efetuada na primeira sequência ou na segunda sequência, de maneira simétrica [Amir et al. (1997)].

Dado um par de sequências s e t, as **operações simétricas** incluem: a substituição de um caractere em s por um caractere de t, que é equivalente a substituir o mesmo caractere em t pelo caractere de s; a remoção de um caractere de s, que é equivalente a inserir esse mesmo caractere em t e vice-versa. Portanto, considerando um conjunto de sequências s e t, é possível transformar s em t e também transformar t em s, com a aplicação da mesma quantidade de operações. Como exemplo, considere as sequências $s =$ AGAG e $t =$ ACAC. Uma representação visual da simetria entre s e t, ao considerar somente a operação de substituição, é mostrada na Figura 2.8. A título de exemplo adicional, considere as sequências $s =$ACGT e $t =$ACG. Uma ilustração da simetria entre s e t, desta vez envolvendo operações de inserção e remoção, é mostrada na Figura 2.9.

$$\begin{matrix} s = \texttt{AGAG} \\ t = \texttt{AGAG} \end{matrix} \text{ ou } \begin{matrix} \texttt{ACAC} \\ \texttt{ACAC} \end{matrix}$$

Figura 2.8: Exemplo de simetria das operações de edição entre as sequências $s =$AGAG e $t =$ACAC. Observe que são necessárias ou duas substituições em s ou duas substituições em t para transformar s em t (ou t em s).

$$\begin{matrix} s = \texttt{ACGT} \\ t = \texttt{ACGT} \end{matrix} \text{ ou } \begin{matrix} \texttt{ACGT} \\ \texttt{ACG} \end{matrix}$$

Figura 2.9: Exemplo de simetria das operações de edição entre as sequências $s =$ACGT e $t =$ACG. Observe que é necessária uma operação de deleção em s ou uma operação de inserção em t para transformar s em t (ou t em s).

Um **mapeamento** de uma sequência em um grafo de sequência (inclusive no grafo de De Bruijn) consiste em encontrar um passeio p no grafo tal que a distância (edição ou Hamming)

entre s e a sequência induzida por p é mínima. Diferente do que ocorre no mapeamento que envolve duas sequências, quando temos uma sequência e um grafo de sequência, há uma assimetria entre as operações de edição. Isso significa que realizar operações de edição na sequência não é equivalente a realizar as operações simétricas de edição no grafo [Amir et al. (1997)]. Essa assimetria surge devido ao fato de um mesmo vértice poder ser utilizado mais de uma vez para induzir uma sequência, o que implica que a quantidade de operações de edição necessárias para transformar s em s' pode não ser a mesma necessária para transformar s' em s. Um exemplo dessa assimetria é apresentado na Figura 2.10.

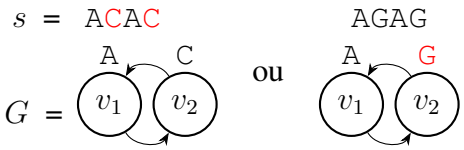


Figura 2.10: Exemplo de assimetria das operações de edição entre a sequência $s =$AGAG e uma sequência induzida por um passeio no grafo $G = (V, A)$, onde $\Sigma = \{\texttt{A}, \texttt{C}, \texttt{G}\}$, $V = \{v_1, v_2\}$, o rótulo de v_1 é A, o rótulo de $v_2 =$ é C, e $A = \{(v_1, v_2), (v_2, v_1)\}$. Observe que são necessárias ou duas substituições na sequência s ou uma substituição no rótulo do vértice v_2 em G (cada vértice é utilizado duas vezes) para que o passeio $p = v_1, v_2, v_1, v_2$ induza a sequência $s' =$AGAG.

2.5 Seed-and-extend

A abordagem ***seed-and-extend*** (semear-e-estender, em tradução livre) é um método utilizado no mapeamento de sequências que consiste em duas fases: semear e estender. Dadas duas sequência s e t, a fase de semear tem como objetivo identificar sementes (*seeds*) na sequência s que estão presentes na sequência t, ou seja, subcadeias de s que ocorrem em t. Uma vez que todas as sementes são identificadas, inicia-se a fase de extensão (*extend*). Nessa etapa, cada uma das sementes é estendida para a esquerda e para a direita, comparando os caracteres subsequentes com os da sequência t. Essa extensão prossegue até que um número predefinido de diferenças seja encontrado durante o processo de comparação. Um exemplo do uso da abordagem *seed-and-extend* é apresentado na Figura 2.11.

```
s   GGTACGTACACACTCGTAGCTAGCTGCATCTATCTATACTACTGCTAGCTACGATCGA
1         TACAC           GCTGC                   AGCTA
2   TTTACGTACACACTCGTAGCTAGCTGCATCTATCTATACTACTGCTAGCTACGATCG
t   TTTACGTACACACACGTAAATAGCTGCATCTATCTATACTACTGCTAGCTACGAAAAA
```

Figura 2.11: Exemplo de aplicação da abordagem *seed-and-extend* às sequências s e t, onde a extensão ocorre até um limite de três diferenças para este caso ilustrativo. A sequência s é apresentada na primeira linha, enquanto a sequência t é exibida na última linha. A linha numerada como 1 corresponde à fase de semear, onde são identificadas as sementes. Na linha numerada como 2, ocorre o processo de estender todas as sementes encontradas. Por fim os caracteres em vermelho na sequência t indicam as posições onde ocorrem diferenças em relação à sequência s.

CAPÍTULO

3

O Problema do Mapeamento de Sequências em Grafos de De Bruijn

Nesta seção, definimos e detalhamos o PROBLEMA DO MAPEAMENTO DE SEQUÊNCIAS EM GRAFOS DE SEQUÊNCIA – PMSG, traçando sua evolução histórica desde o início, quando surgiu como um desdobramento do problema de mapear padrões em hipertextos. A partir dessa origem em hipertextos, o PMSG evoluiu para seu formato atual, aplicado aos grafos de sequências conforme definido neste trabalho, culminando na sua forma mais recente como o PROBLEMA DO MAPEAMENTO DE SEQUÊNCIAS EM GRAFOS DE DE BRUIJN – PMSB, que se concentra em grafos de De Bruijn. Além disso, apresentamos variações do problema que levam em consideração caminhos e passeios. Focando especificamente nos passeios, exploramos variantes do problema que foram identificadas desde o início dos estudos em hipertextos e que continuam relevantes até os grafos de De Bruijn. Por fim, nesta seção, discutimos o estado da arte relacionado a esses problemas mencionados.

3.1 PMSG – Problema do Mapeamento de Sequências em Grafos de sequência

O PROBLEMA DO MAPEAMENTO DE SEQUÊNCIAS EM GRAFOS DE SEQUÊNCIA tem sido bem estudado na literatura, há muito tempo, por meio de um problema equivalente denominado MAPEAMENTO APROXIMADO DE PADRÕES NO HIPERTEXTO. O primeiro artigo que abordou esse problema foi escrito por Manber e Wu em 1992. No artigo intitulado *Approximate String Matching With Arbitrary Costs For Text and Hypertext* [Manber and Wu (1992)], Manber e Wu exploraram o problema de mapear um padrão em um hipertexto.

Nesse trabalho, os autores propõem uma solução para o cenário em que um hipertexto T pode

ser representado como um grafo acíclico dirigido (DAG). A modelagem desse hipertexto envolve a construção de um grafo $G = (V, A)$, no qual cada vértice $v \in V$ está associado a um trecho de texto em T. De forma mais formal, temos que $G = (V, A)$ é um DAG, e $T = \{T_v \in \Sigma^+ | v \in V\}$. Dado um padrão s e um hipertexto T representado por um grafo G, o problema consiste em encontrar um caminho c em G de tal maneira que a distância entre o texto associado a esse caminho (semelhante à sequência induzida discutida na Seção 2.3.3) e s seja mínima. Com base nessas informações, a definição do problema é a seguinte:

Problema 3. MAPEAMENTO APROXIMADO DE PADRÕES EM HIPERTEXTO – PMAPH*: Dado um padrão s e um grafo G que representa um hipertexto T, o objetivo é encontrar um caminho c em G tal que distância entre o texto associado a c e s é mínima.*

No contexto do algoritmo PMAPH, quando dados uma sequência s, um hipertexto representado por um grafo acíclico dirigido G e um alfabeto Σ, a abordagem de Manber e Wu pode ser resumida da seguinte maneira: o algoritmo percorre o grafo em ordem topológica, armazenando e mesclando listas de índices ativos quando visita um nó pela primeira vez e novamente quando retorna a esse nó [Manber and Wu (1992)]. Cada lista de índices é uma lista ordenada que representa a posição de cada caractere $x \in \Sigma$ no padrão. A ordem topológica garante que todos os predecessores de um nó já foram explorados, garantindo que a lista mesclada contenha todas as possíveis correspondências ativas que terminam antes ou no nó atual. O resultado final desse algoritmo é um caminho c no grafo G, representado pela lista de índices, no qual a distância entre o texto associado a c e o padrão s é mínima. A complexidade de tempo desse algoritmo, conforme proposto pelos autores, é de $O(m \cdot |A| + R \cdot \log\log m)$, em que m é o comprimento do padrão mapeado, $|A|$ é a quantidade de arcos no grafo e R é a quantidade de vezes que o padrão é encontrado no grafo.

O trabalho pioneiro realizado por Manber e Wu se concentra em um *mapeamento aproximado* de um padrão no hipertexto, o que significa permitir mudanças no padrão. Além disso, esse problema também foi investigado no contexto de um *mapeamento exato*, onde o objetivo é encontrar um padrão no hipertexto sem permitir qualquer mudança no padrão correspondente. No artigo intitulado *A Linear Time Pattern Matching Algorithm Between a String and a Tree* [Akutsu (1993)], Akutsu aborda o seguinte problema para o mapeamento exato:

Problema 4. MAPEAMENTO EXATO DE PADRÕES NO HIPERTEXTO – PMEPH*: Dado um padrão s e um grafo G que representa um hipertexto T, o objetivo é encontrar um caminho c em G tal que distância entre o texto associado a c e s é zero.*

No contexto do PMEPH, Akutsu propôs um algoritmo para casos em que o hipertexto pode ser modelado como uma árvore (ou seja, um grafo acíclico e conexo). A complexidade de tempo desse algoritmo é $O(|V|)$, onde $|V|$ representa a quantidade de vértices da árvore [Akutsu (1993)]. Posteriormente, Park e Kim abordaram o PMEPH em situações em que o hipertexto pode ser representado como um grafo acíclico dirigido (DAG), como discutido no artigo intitulado *String Matching in Hypertext* [Park and Kim (1995)]. Os autores propuseram uma solução com

complexidade polinomial de tempo, que é $O(|V|+m\cdot|A|)$, onde m representa o comprimento do padrão mapeado, e $|V|$ e $|A|$ denotam a quantidade de vértices e arestas no DAG, respectivamente.

No artigo intitulado *Approximate Matching in Hypertext* [Amir et al. (1997)], Amir *et al.* estudaram o PMAPH em maior detalhe. Diferentemente dos autores anteriores que consideraram caminhos no hipertexto, neste artigo, os autores optaram por considerar passeios e modelaram o hipertexto como um grafo dirigido. Para distinguir essa abordagem do PMAPH convencional, utilizaremos a sigla $\texttt{PMAPH}_p$ para denotar que o problema envolve passeios.

No caso do mapeamento exato, os autores propuseram uma solução com complexidade polinomial de tempo, especificamente $O(|V| + m \cdot |A|)$ para o problema. Além disso, eles estenderam essa solução para o mapeamento aproximado, alcançando também uma solução polinomial de tempo, que é $O(m(|V| \cdot \log |V| + |A|))$. Vale ressaltar que esses autores foram os primeiros a identificar a assimetria de um mapeamento aproximado em um grafo quando se consideram passeios (ver Seção 2.4). Considerando a assimetria identificada por Amir *et al.*, percebe-se que o $\texttt{PMAPH}_p$ admite três variantes, sendo elas:

- **variante 1** – $\texttt{PMAPH}_{p_1}$: permite mudanças no padrão ao realizar o mapeamento do padrão no grafo;
- **variante 2** – $\texttt{PMAPH}_{p_2}$: permite mudanças no grafo ao realizar o mapeamento do padrão no grafo; e
- **variante 3** – $\texttt{PMAPH}_{p_3}$: permite mudanças no padrão e no grafo ao realizar o mapeamento do padrão no grafo.

Falando exclusivamente da variante 1, $\texttt{PMAPH}_{p_1}$, os autores resolveram esse problema utilizando a solução polinomial mencionada no parágrafo anterior para o mapeamento aproximado com complexidade de tempo $O(m(|V| \cdot \log |V| + |A|))$ para mapear um padrão s com $|s| = m$ em um hipertexto representado por um grafo dirigido $G = (V, A)$. Em resumo, a abordagem dos autores envolve a identificação de correspondências do padrão s em G, seguida pela criação de um grafo G' com base nessas correspondências. Posteriormente, é realizada uma busca em profundidade em G' para marcar os vértices alcançáveis e, por fim, verifica-se cada vértice em G para identificar correspondências adicionais com base nas conexões no grafo.

Para as outras duas variantes, $\texttt{PMAPH}_{p_2}$ e $\texttt{PMAPH}_{p_3}$, Amir *et al.* provaram que ambos os problemas são NP-completos quando se consideram as distâncias de Hamming e edição para um alfabeto de tamanho constante [Amir et al. (1997)].

Mais tarde, em seu artigo intitulado *Improved Approximate Pattern Matching on Hypertext* [Navarro (1998)], Navarro abordou a variante $\texttt{PMAPH}_{p_1}$ previamente tratada por Amir *et al.* e introduziu melhorias que resultaram em um algoritmo polinomial com complexidade de tempo $O(m(|V| + |A|))$.

Os artigos citados anteriormente abordam a questão de mapear um padrão em um hipertexto, no qual o hipertexto é representado por um grafo (que pode ser uma árvore, DAG, ou um grafo dirigido). Esses trabalhos são fundamentais para o Problema do Mapeamento de Sequências

em Grafos de Sequências – PMSG. Embora os estudos mencionados abordem o mapeamento de um padrão em um hipertexto, neste contexto, *podemos interpretar o padrão mapeado como uma sequência e o grafo usado para representar o hipertexto como um grafo de sequências*. Daqui em diante, apenas o termo sequência é utilizado para se referir a uma cadeia de caracteres. Com base nessas considerações, o PMSG é definido aqui da seguinte forma (para a definição de mapeamento, consulte a Seção 2.4):

Problema 5. Mapeamento de Sequências em Grafos de Sequência – PMSG*: Dados como entrada uma sequência s e um grafo de sequência G, determinar um mapeamento de s em G.*

Como o PMSG envolve a utilização de passeios no mapeamento, as três variantes identificadas por Amir *et al.* também são aplicáveis a este contexto. O PMSG é modelado da seguinte maneira:

- **variante 1** – $\texttt{PMSG}_{p_1}$: permite mudanças na sequência ao realizar o mapeamento da sequência no grafo de sequência;
- **variante 2** – $\texttt{PMSG}_{p_2}$: permite mudanças no grafo de sequência ao realizar o mapeamento da sequência no grafo de sequência; e
- **variante 3** – $\texttt{PMSG}_{p_3}$: permite mudanças na sequência e no grafo de sequência ao realizar o mapeamento da sequência no grafo de sequência.

Um dos primeiros artigos a abordar a variante $\texttt{PMSG}_{p_1}$, embora não publicado, foi escrito por Rautiainen e Marschall e intitula-se *Aligning Sequences to General Graphs in* $O(|V| + m \cdot |A|)$ *Time* [Rautiainen and Marschall (2017)]. Neste artigo, os autores propuseram um algoritmo polinomial para mapear uma sequência s com $|s| = m$ em um grafo de sequência simples $G = (V, A)$. O algoritmo possui uma complexidade de tempo $O(|A| \cdot m + |V| \cdot m \cdot \log(|A| \cdot m))$ e fornece como resultado o mapeamento realizado em G. Resumidamente, a solução proposta pelos autores envolve a construção de um novo grafo G' a partir da sequência s e do grafo G. No processo de construção, o grafo G é duplicado para cada caractere da sequência s, e novos arcos são inseridos entre os grafos duplicados em G'. A solução, então, consiste principalmente em executar um algoritmo para determinar um caminho mínimo p em G', e esse caminho p determina o passeio a ser percorrido no grafo de sequência G. Como esse algoritmo está diretamente relacionado a uma das soluções apresentadas neste trabalho para grafos de De Bruijn, será discutido com maior profundidade na Seção 3.1.2. Além disso, os autores também apresentaram uma versão mais eficiente do algoritmo, com complexidade de tempo $O(|V| + m \cdot |A|)$, que fornece apenas a distância do mapeamento.

No artigo intitulado *On the Complexity of Sequence to Graph Alignment* [Jain et al. (2019)], os autores Jain *et al.* conduziram um estudo sobre o PMSG, abordando as três variantes identificadas por Amir *et al.* Para a variante 1, $\texttt{PMSG}_{p_1}$, os autores propuseram uma modificação do algoritmo previamente proposto por Rautiainen e Marschall. Essa modificação permitiu que a solução excluísse o prefixo da sequência mapeada, o que não estava abrangido anteriormente por esse cenário, mantendo a complexidade de tempo em $O(|V| + m \cdot |A|)$.

Baseando-se na prova dos autores Amir *et al.*, Jain *et al.* estabeleceram que, para grafos de sequência que possuem um ou mais caracteres em seus vértices, bem como um alfabeto Σ de tamanho $|\Sigma| \geq 2$, tanto a variante 2 quanto a variante 3 ($\texttt{PMSG}_{p_2}$ e $\texttt{PMSG}_{p_3}$) são problemas NP-completos, independentemente se considerarmos a distância de Hamming ou a distância de edição. Adicionalmente, é relevante mencionar que os autores ressaltam que a prova deles não aborda grafos de De Bruijn.

Um resumo de todos os trabalhos mencionados para mapeamento exato e mapeamento aproximado com mudanças na sequência, abordados por cada um, juntamente com a complexidade de tempo dos respectivos algoritmos, pode ser visualizado na Tabela 3.1. A tabela está organizada da esquerda para a direita, da seguinte maneira: ano de publicação, autores, tipo de grafo, tipo de mapeamento, se é caminho ou passeio, tipo de distância e a complexidade de tempo dos algoritmos. Em relação às mudanças no grafo, de forma resumida, o problema é NP-completo, conforme demonstrado por Amir *et al.* e Jain *et al.* [Amir et al. (1997); Jain et al. (2019)].

Ano	**Autor(es)**	**Grafo**	**Exato/ Aproximado**	**(C)aminho/ (P)asseio**	**(H)amming/ (E)dição**	**Tempo**
1992	Manber e Wu	DAG	Aproximado	C	E	O(m ·|A|+ R ·log log m)
1993	Akutsu	Árvore	Exato	C	-	O(|V |)
1995	Park e Kim	DAG	Exato	C	-	O(|V |+ m ·|A|)
1997	Amir *et al.*	Dirigido	Exato	P	-	O(|V |+ m ·|A|)
1997	Amir *et al.*	Dirigido	Aproximado	P	H/E	O(m(|V |·log |V |+ |A|))
1998	Navarro	Dirigido	Aproximado	P	H/E	O(m(|V |+ |A|))
2017	Rautiainen e Marschall	Dirigido	Aproximado	P	H/E	O(|V |+ m ·|A|)
2019	Jain *et al.*	Dirigido	Aproximado	P	H/E	O(|V |+ m ·|A|)

Tabela 3.1: Tabela com os artigos fundamentais para o PMSG com mudanças na sequência. A tabela é composta por ano, autores, tipo de grafo, tipo de mapeamento, se o problema busca um caminho e/ou passeio, a distância utilizada e a complexidade de tempo de cada trabalho. O comprimento da sequência é m, $|V|$ é a quantidade de vértices e $|A|$ é a quantidade de arcos em um grafo.

3.1.1 Exemplificando o Problema do Mapeamento de Sequências em Grafos de Sequência

Restringindo para grafo de sequência simples, o Problema do Mapeamento de Sequências em Grafos de Sequência – PMSG consiste em, dadas como entrada uma sequência s, de comprimento m, e um grafo de sequência simples G, encontrar um passeio p em G cuja sequência induzida s' seja a mais semelhante possível à sequência s (dada uma medida de distância). Por exemplo, considere a sequência $s =$ATGCTAGA, o grafo G apresentado na Figura 3.1, e os seguintes passeios no grafo G: $p_1 = v_1, v_4, v_2, v_3, v_5, v_8, v_7, v_6$ e $p_2 = v_1, v_2, v_3, v_5, v_8, v_7, v_4, v_2$ que induzem as sequências $s_1 =$ ATCGTAGC e $s_2 =$ ACGTAGTC, respectivamente. Utilizando a distância de Hamming, tem-se que $d_H(s, s_1) = 3$ e $d_H(s, s_2) = 6$. Nesse caso, s_1 é mais semelhante à s em relação a sequência s_2. O passeio p_1 portanto é uma possível resposta para o mapeamento. Observe que existem outros passeios no grafo, mas nenhum passeio induz uma

sequência tal que as distâncias de Hamming entre s e as sequências induzidas sejam menor que 3.

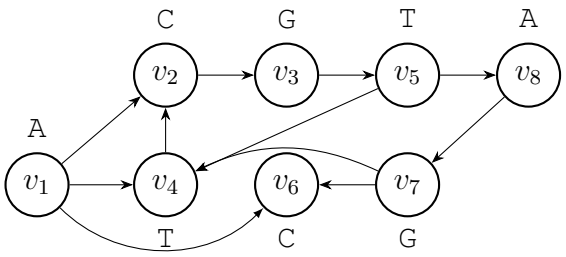


Figura 3.1: Exemplo de um grafo de sequência simples G.

3.1.2 Algoritmo RaMa

Como mencionado anteriormente, o PMSG admite três variantes. A primeira variante, PMSG_{p_1}, permite mudanças apenas na sequência. Para essa variante, a solução polinomial estudada por Rautiainen e Marschall [Rautiainen and Marschall (2017)] e Jain *et al.* [Jain et al. (2019)] utiliza um grafo auxiliar, chamado de grafo de multicamadas G', e tem complexidade de tempo $O(|A|\cdot m+|V|\cdot m\cdot\log(|V|\cdot m))$ para mapear uma sequência s em um grafo de sequência simples G, onde A e V representam os conjuntos de arcos e vértices, respectivamente, e m é o comprimento da sequência. Neste trabalho, nos referimos a esse algoritmo como o **algoritmo RaMa**.

Dada uma sequência s de comprimento m e um grafo de sequência simples $G = (V = \{v_1, \ldots, v_n\}, A)$, o **grafo de multicamadas** $G' = (V', A')$ é um grafo com custo nas arestas obtido a partir do grafo de sequência simples G e da sequência s com custo $c(e) \in \{0, 1\}$ para todo arco $e \in A'$. Informalmente, o grafo G' é composto por m camadas, onde cada camada contém uma cópia do grafo G, e há arcos conectando as camadas i e $i + 1$ para $1 \leq i < m$. Além disso, existem dois vértices, denotados como u e w, com arcos direcionados de u para todos os vértices da primeira camada e arcos direcionados de todos os vértices da camada m para w. Um exemplo de grafo de multicamadas é apresentado na Figura 3.2. Mais precisamente, o conjunto de vértices V' é $\{(V \cup \{v_0\}) \times \{1, 2, \ldots, m\}\} \cup \{\text{u}, \text{w}\}$ e o conjunto de arcos é $A' = S \cup T \cup L \cup D \cup I$ no qual

1. $S = \{(\text{u}, (v_j, 1)) : 0 \leq j \leq n\}$, com

$$c(s, (v_j, 1)) = \begin{cases} 1 & \text{se } j = 0; \\ 0 & \text{caso contrário;} \end{cases}$$

2. $T = \{((v_j, m), \text{w}) : 0 \leq j \leq n\}$, com

$$c((v_j, m), \text{w}) = 0;$$

3. $L = \{((v_j, i), (v_h, i)) : 1 \leq i \leq m \ \mathbf{E} \ (v_j, v_h) \in A\}$, com

$$c((v_j, i), (v_h, i)) = 1;$$

4. $D = \{((v_j, i), (v_h, i')) : 1 \leq i < i' \leq m \ \mathbf{E} \ v_j = v_h \ \mathbf{E} \ i + 1 = i'\}$, com

$$c((v_j, i), (v_h, i')) = 1;$$

5. $I = \{((v_j, i-1), (v_h, i))((v_0, i-1), (v_0, i)) : h \neq 0, 1 < i \leq m, (v_j, v_h) \in A \text{ ou } j = 0\}$, com

$$c((v_j, i-1), (v_h, i)) = v_h \stackrel{?}{=} s[i] \text{ e } c((v_0, i-1), (v_0, i)) = 1.$$

O $v_h \stackrel{?}{=} s[i]$ é igual a 1 se o caractere de $v_h \neq s[i]$, e 0 caso contrário. Os vértices $\mathrm{u}, \mathrm{w}, (v_0, j)$ são chamados de vértices *source*, *target* e *dummy*, respectivamente.

Uma vez construído o grafo de multicamadas, a solução proposta por Rautiainen e Marschall [Rautiainen and Marschall (2017)] para resolver o `PMSG` consiste em encontrar um menor caminho de u até w em G' usando o algoritmo de Dijkstra [Dijkstra (1959)], por exemplo. O grafo de multicamadas contém uma cópia do grafo de sequência simples em cada camada e possui arcos com custos, onde cada arco representa uma operação de edição: as arestas horizontais representam a operação de inserção, as verticais representam a operação de deleção, e as diagonais representam a operação de substituição. Com base nessas informações, encontrar um caminho mínimo de u até w consiste em percorrer todas as camadas e selecionar os arcos que minimizam a distância de edição.

Uma vez que tenha sido determinado um caminho $c = s, (v_j, i), \ldots, (v_n, m), w$ de u até w para $1 \leq j \leq n$ e $1 \leq i \leq m$, podemos induzir um passeio p em G que gera uma sequência s' cuja distância em relação a s é a menor possível a partir do caminho c. Para formar o passeio p em G, é necessário percorrer o caminho c na ordem indicada e construir o passeio p considerando os vértices $v_j, \ldots, v_m$ de c. A Figura 3.2 mostra um exemplo de mapeamento e do grafo de multicamadas obtido de uma sequência e um grafo de sequência simples. Para esse mesmo exemplo, uma possível solução do `PMSG` determinado pelo algoritmo `RaMa` é o passeio $w = v_1, v_2, v_3, v_4$ no grafo de sequência simples G, induzido pelo caminho $p = u, (v_1, 1), (v_2, 2), (v_3, 2), (v_4, 3), v$ em G'. A sequência induzida por w é `ACGT`, com uma distância de edição 1 em relação à sequência $s =$ `ACT`.

Rautiainen e Marschall propuseram uma versão mais eficiente do algoritmo, denominado aqui de **algoritmo `RaMa`$_d$**, que devolve apenas a distância do mapeamento. Esse algoritmo possui complexidade de tempo $O(|V| + m \cdot |A|)$, com A e V representando os conjuntos de arcos e vértices do grafo, respectivamente, e m o comprimento da sequência. Dada uma sequência s e um grafo de sequência simples G, a ideia principal desse algoritmo é calcular a distância do mapeamento utilizando apenas duas camadas por vez. Em outras palavras, o algoritmo constrói e processa as camadas i e $i + 1$ para $i = 1, \ldots, m - 1$, apaga a camada i, avança para $i + 1$ e repete o processo até finalizar.

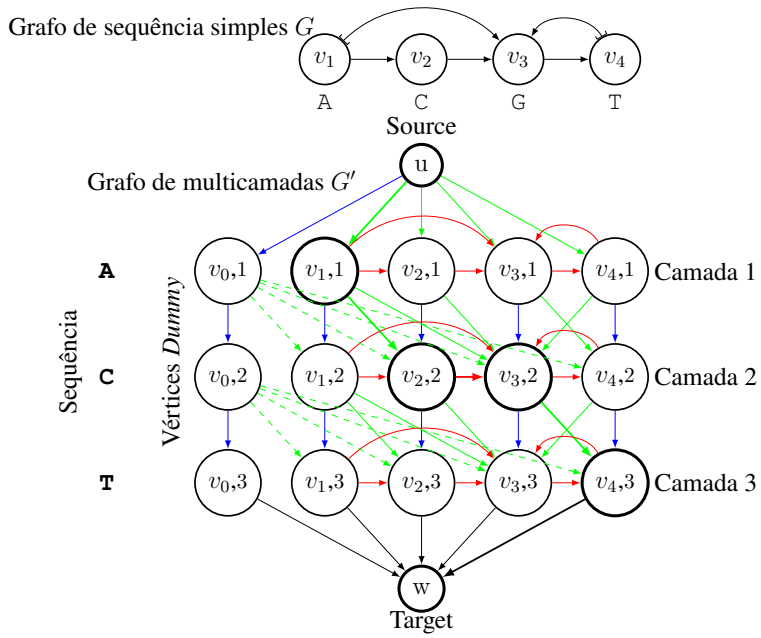


Figura 3.2: Exemplo do mapeamento da sequência $s =$ACT no grafo de sequência simples G. O grafo de multicamadas G' é construído com base na sequência s e no grafo de sequência simples G. As arestas vermelhas, azuis e verdes são os custos de inserção, deleção e substituição, respectivamente [Rautiainen and Marschall (2017); Jain et al. (2019)].

3.2 PMSB – Problema do Mapeamento de Sequências em Grafos de De Bruijn

Falando agora exclusivamente da tarefa de mapear sequências em grafos de De Bruijn, um dos primeiros artigos que trata desse problema, considerando caminhos, intitula-se *Read Mapping on De Bruijn Graphs* [Limasset et al. (2016)]. Nesse artigo, o problema foi descrito da seguinte forma:

Problema 6. Mapeamento de Sequências em Grafos de De Bruijn – PMSB*: Dados como entrada um conjunto S de sequências sobre um alfabeto Σ, um inteiro $k \geq 2$, uma sequência $s \in \Sigma^*$ onde $|s| \geq k$, uma função custo $F : \Sigma \times \Sigma \to \mathbb{N}$ e um limitante natural t, decidir se existe um caminho no grafo de De Bruijn construído sobre S, composto por $|s| - k + 1$ vértices, que induz a sequência $s' = s'_1 \dots s'_{|s|} \in \Sigma^{|s|}$, tal que $\Sigma_{i=1}^{|s|} = F(s'_i, s_i) \leq t$.*

Esse problema foi provado ser NP-completo pelos autores do artigo que também propuseram uma heurística para ele chamada BGREAT (*de Bruijn Graph Read mapping Tool*).

Os autores Liu *et al.* também estudaram o PMSB no artigo intitulado *deBGA: Read Alignment With De Bruijn Graph-based Seed And Extension* [Liu et al. (2016)] e desenvolveram uma heurística denominada deBGA (*de Bruijn Graph-based Aligner*). Heydari *et al.* também estudaram o PMSB no artigo intitulado *BrownieAligner: Accurate Alignment of Illumina Sequencing Data to De Bruijn Graphs* [Liu et al. (2016)] e desenvolveram uma heurística denominada BrownieAligner [Heydari et al. (2018)] para ele. As heurísticas mencionadas estão baseadas na técnica *seed-and-extend* (semear-e-estender, em tradução livre), que consiste em, para cada k-mer da

sequência a ser mapeada, identificar a localização desse k-mer (*seed*) no grafo e para cada k-mer semeado, estendê-lo (*extend)* em busca de um melhor caminho. Para uma sequência s e um grafo G, informalmente, entendemos como melhor caminho aquele que induz uma sequência s' tal que a distância entre s e s' é a menor possível.

Em resumo, considerando um grafo de De Bruijn G_k e uma sequência s, essas heurísticas diferem na maneira como utilizam as sementes e abordam a etapa de extensão. Por exemplo, o BGREAT, para ser eficiente, estende apenas um número limitado de sementes, procurando, em cada extensão, o melhor caminho que torna a sequência induzida por esse caminho mais semelhante possível a s. Por outro lado, o DeBGA utiliza de forma eficiente os chamados "caminhos únicos" em G_k. Caminhos únicos são aqueles que possuem uma única aresta de entrada e uma única aresta de saída em cada vértice, continuando até encontrar um vértice com múltiplas arestas de saída. Quando um caminho único é encontrado durante a extensão, automaticamente todo o caminho é incorporado à extensão, e consequentemente, toda a sequência induzida por esse caminho único é incluída no mapeamento. Por fim, o BrownieAligner se destaca por trabalhar tanto com sementes exatas quanto com sementes aproximadas, com uma distinção entre elas. As sementes aproximadas são identificadas com o auxílio da ferramenta chamada essaMEM [Vyverman et al. (2013)].

Observe que o PMSB consiste em mapear uma sequência em um grafo especial, porém a tarefa essencial ainda é mapear uma sequência em um grafo de sequência. Portanto, quando se trata de passeios, as três variantes identificadas por Amir *et al.* [Amir et al. (1997)] e mencionadas na Seção 3.1 também se aplicam aos grafos de De Bruijn, e a formulação do problema é a seguinte:

Problema 7. Mapeamento de Sequências em Grafos de De Bruijn para passeio – $\texttt{PMSB}_p$*: Dados como entrada uma sequência s e um grafo de De Bruijn G_k, determinar um mapeamento de s em G_k.*

Neste contexto, utilizamos $\texttt{PMSB}_p$ com o objetivo de distinguir os passeios dos caminhos. Além disso, empregamos o prefixo $\texttt{PMSB}_p$ para denotar as três variantes do problema, que são:

- **variante 1** – $\texttt{PMSB}_{p_1}$: permite mudanças na sequência ao realizar o mapeamento da sequência no grafo de De Bruijn;
- **variante 2** – $\texttt{PMSB}_{p_2}$: permite mudanças no grafo ao realizar o mapeamento da sequência no grafo de De Bruijn; e
- **variante 3** – $\texttt{PMSB}_{p_3}$: permite mudanças na sequência e no grafo ao realizar o mapeamento da sequência no grafo de De Bruijn.

Recentemente, o $\texttt{PMSB}_p$ foi abordado no artigo intitulado *On the Hardness of Sequence Alignment on De Bruijns Graphs* [Gibney et al. (2022)]. Os autores Gibney *et al.* provaram que o problema é NP-completo (para um alfabeto de tamanho constante) quando as mudanças ocorrem apenas no grafo de De Bruijn com $k \geq 4$, considerando exclusivamente a operação de substituição. É relevante relembrar que um grafo de De Bruijn pode induzir arcos com base nos rótulos (k-mers) de seus vértices. No entanto, essa característica não é utilizada no

trabalho dos autores, que utilizam um grafo de sequência simples para representar o grafo de De Bruijn. Na visualização do grafo de De Bruijn como um grafo de sequência simples, os k-mers são representados por passeios de comprimento k. Em outras palavras, cada passeio de comprimento k induz um k-mer distinto. No trabalho dos autores, uma modificação realizada no grafo não altera a estrutura (arcos) do grafo, o que torna o problema mais difícil. Nesse artigo, os autores abordaram a seguinte versão de decisão do problema:

Problema 8. *Dados como entrada um grafo de De Bruijn G_k, uma sequência s e um inteiro $\delta \geq 0$, determinar se existe um passeio p em G_k tal que a distância entre s a sequência induzida s' por p é zero com no máximo δ substituições no grafo.*

Além disso, eles também provaram que não existe um algoritmo mais eficiente do que $O(|A| \cdot m)$ para grafos de De Bruijn quando as mudanças acontecem apenas na sequência s, sendo que m é o comprimento da sequência e $|A|$ é a quantidade de arcos do grafo.

3.2.1 Exemplificando o Problema do Mapeamento de Sequências em Grafos de De Bruijn

O Problema do Mapeamento de Sequências em Grafos de De Bruijn – PMSB consiste em, dados como entrada uma sequência s, de comprimento m, e um grafo de De Bruijn de ordem k, G_k, encontrar um passeio p em G_k cuja sequência induzida (por p) s' seja a mais semelhante possível à sequência s (dada uma medida de distância). Por exemplo, considere a sequência $s =$ GTTCTACG e o grafo G_3 apresentado na Figura 3.3 e os seguintes passeios no grafo G_3: $p_1 = v_1, v_2, v_3, v_4, v_6, v_7$ e $p_2 = v_1, v_5, v_8, v_9, v_{10}, v_7$ que induzem as sequências $s_1 =$ GTTCGGAC e $s_2 =$ GTTGCGAC, respectivamente. Utilizando a distância de Hamming, tem-se que $d_H(s, s_1) = 4$ e $d_H(s, s_2) = 5$, portanto, s_1 é mais semelhante à s em relação a sequência s_2. O passeio p_1 é de fato a resposta para o mapeamento. É válido destacar que este trabalho aborda o mapeamento de sequências em um grafo de sequência simples. Uma das abordagens estudadas aqui consiste em converter o grafo de De Bruijn em um grafo de sequência simples, seguido pela execução do algoritmo RaMa.

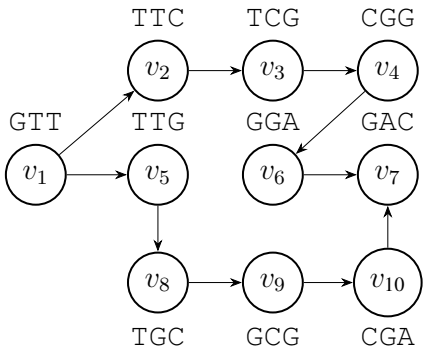


Figura 3.3: Exemplo de um grafo de De Bruijn G_3.

CAPÍTULO

4

Equivalência entre Grafos de De Bruijn e Grafos de Sequência Simples

Nesta seção, discutimos a equivalência entre grafos de De Bruijn e grafos de sequência simples, no sentido de que, para todo passeio p no grafo de De Bruijn, existe um passeio correspondente no grafo de sequência simples que induz a mesma sequência que p. A representação do grafo de De Bruijn, especialmente como um grafo de sequência simples, é abordada de forma abstrata em artigos teóricos, conforme discutido por Gibney *et al.* [Gibney et al. (2022)]. Nesse trabalho, os autores utilizam o grafo de De Bruijn atribuindo a cada vértice um caractere, e os k-mers são gerados por passeios de comprimento k. Para aprofundar essa análise, especialmente devido à sua relevância para nosso trabalho, apresentamos um algoritmo ingênuo para converter um grafo de De Bruijn G_k em um grafo de sequência simples G. Demonstramos a equivalência entre esses dois grafos, ou seja, mostramos que ambos induzem o mesmo conjunto de sequências. Também apresentamos um segundo algoritmo que realiza a conversão de um grafo de De Bruijn G_k em um grafo de sequência simples G, com a particularidade de que G tem menos vértices em relação a conversão do algoritmo ingênuo, mas G não é mínimo.

4.1 Conversão de um grafo de De Bruijn em um grafo de sequência simples

Considere que um grafo de De Bruijn, de ordem k, possui n vértices. Na Seção 4.1.1, apresentamos um **algoritmo ingênuo** para converter um grafo de De Bruijn em um grafo de sequência simples com $n \cdot k$ vértices, denominado aqui de *De Bruijn to Simple Sequence Graph Tool* (Algoritmo 1). Em seguida, demonstramos que as sequências induzidas no grafo de De Bruijn também são induzidas no grafo de sequência simples. Na Seção 4.1.2, introduzimos

um **algoritmo redutor de vértices** para essa conversão, com o objetivo de obter um grafo de sequência simples com x vértices, onde $n \leq x \leq n \cdot k$, que pode ter menos vértices do que a conversão ingênua. Essa solução é descrita pelo algoritmo denominado aqui de *De Bruijn to smaller Simple Sequence Graph Tool* (Algoritmo 4). Para essa conversão, também demonstramos que as sequências induzidas no grafo de De Bruijn são induzidas no grafo de sequência simples.

4.1.1 Algoritmo ingênuo

A partir de um grafo de De Bruijn G_k, podemos transformá-lo em um grafo de sequência simples G com dois passos:

Passo 1. para cada vértice (k-mer) u em G_k, o primeiro passo consiste em dividi-lo em outros k vértices $v_{u_1}, \ldots, v_{u_k}$, onde o rótulo de v_{u_i} corresponde ao i-ésimo símbolo do k-mer de u. Esses novos vértices constituem o conjunto de vértices de G.

Passo 2. o segundo passo envolve a inserção dos arcos em G para conectar os vértices desse grafo. De modo geral, para cada vértice u de G_k que foi dividido, inserimos os arcos $(v_{u_1}, v_{u_2}), (v_{u_2}, v_{u_3}), \ldots, (v_{u_{k-1}}, v_{u_k})$. Adicionalmente, para cada par de vértices adjacentes u e u' em G_k, adicionamos um arco conectando o vértice v_{u_k} da divisão de u ao vértice $v_{u'_k}$ da divisão de u' em G.

Um exemplo de conversão é mostrado na Figura 4.1, e o código correspondente está apresentado no Algoritmo 1. No algoritmo, para facilitar a sua escrita, assumimos na linha 3 e 4, onde os vértices do grafo de sequência simples são criados, que um vértice e seu símbolo são equivalentes. Nas linhas 6 e 8, também fazemos a mesma suposição para determinar os arcos. É importante destacar que essa consideração é estendida à prova do Teorema 1. Para determinar a complexidade de tempo do Algoritmo 1 considere que $|V| = n$ e $|A| = l$, assim a complexidade de tempo é $O(n \cdot (k + l))$.

Teorema 1. *Todas as sequências induzidas no grafo de De Bruijn G_k também são induzidas no grafo de sequência simples G.*

Demonstração. Dado um grafo de De Bruijn G_k e o grafo de sequência simples G resultante da conversão, observe que G contém todos os k-mers de G_k divididos e corretamente conectados com arcos. Dado um passeio $p = u_1, \ldots, u_n$ em G_k, existe um passeio $qp' = v_{u_{11}}, \ldots, v_{u_{1k-1}}, v_{u_{1k}}, v_{u_{2k}} \ldots, v_{u_{nk}}$ em que $q = v_{u_{11}}, \ldots, v_{u_{1k}}$ e $p' = v_{u_{2k}}, \ldots, v_{u_{nk}}$ em G, assim

$$\begin{aligned} p &= u_1, u_2, \ldots, u_n \\ &= v_{u_{11}}, \ldots, v_{u_{1k}}, v_{u_{2k}}, \ldots, v_{u_{nk}} \\ &= qp' \end{aligned}$$

podemos observar que $q = v_{u_{11}}, \ldots, v_{u_{1k}}$ induz a mesma sequência que o vértice u_1, e cada caractere é induzido de maneira equivalente em $v_{u_{2k}} = u_2[k], \ldots, v_{u_{nk}} = u_n[k]$. Portanto, p e qp' induzem as mesmas sequências. □

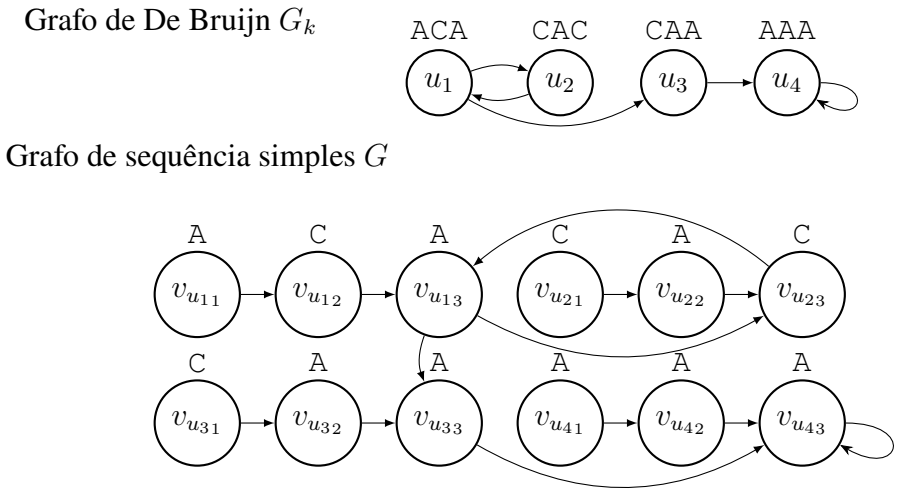


Figura 4.1: Exemplo da conversão de um grafo de De Bruijn G_k, onde $k = 3$, para um grafo de sequência simples G. Cada vértice u em G_k é subdividido em k vértices, com arcos conectando o primeiro vértice ao segundo, o segundo ao terceiro, e assim por diante, até que o vértice $k-1$ esteja conectado ao vértice k. Por exemplo, os arcos $(v_{u_{11}}, v_{u_{12}})$ e $(v_{u_{12}}, v_{u_{13}})$ em G representam a subdivisão para o k-mer ACA. Se um vértice u em G_k possui vértices adjacentes u', então há um arco entre o vértice k da subdivisão resultante de u e o vértice que representa o k-ésimo caractere de cada adjacência u'. Um exemplo é o arco do vértice $v_{u_{13}}$ para $v_{u_{23}}$ em G, representando a adjacência do k-mer ACA para CAC em G_k. Além disso, essa regra se aplica ao caso de laços em G_k, no qual existe um arco do vértice k da subdivisão resultante de u para o vértice que representa o k-ésimo caractere da adjacência u', e neste caso, é o vértice k da subdivisão resultante de u, como ilustrado pelo vértice $v_{u_{43}}$ em G para o k-mer AAA em G_k.

Excesso de vértices

A conversão ingênua adotada pelo Algoritmo 1 resulta em um número excessivo de vértices. Essa conversão cria k vértices para cada vértice original do grafo de De Bruijn. No entanto, muitos desses vértices são redundantes e podem ser descartados. Para ilustrar esse problema, considere a Figura 4.1, onde podemos observar que os vértices $v_{u_{21}}$ e $v_{u_{22}}$ poderiam ser descartados. Isso ocorre porque os vértices $v_{u_{12}}$ e $v_{u_{13}}$ já são capazes de induzir as mesmas sequências que os vértices $v_{u_{21}}$ e $v_{u_{22}}$ induziriam. Portanto, a presença desses vértices extras é desnecessária. Para exemplificar mais claramente o excesso de vértices, na Figura 4.2 apresentamos a mesma conversão da Figura 4.1, destacando os vértices excedentes na cor roxa.

Dadas essa consideração, torna-se importante o desenvolvimento de um algoritmo que converte um grafo de De Bruijn em um grafo de sequência simples com menos vértices. Os passos para reduzir a quantidade de vértices são apresentados no algoritmo denominado *De Bruijn to smaller Simple Sequence Graph Tool* (Algoritmo 4). A seguir, apresentamos os *De Bruijn to De Bruijn Equivalent Tool* (Algoritmo 2) e *De Bruijn to smaller simple sequence graph tool* (Algoritmo 3) que compõem o Algoritmo 4.

Algoritmo 1: De Bruijn to Simple Sequence Graph Tool

Entrada: um grafo de De Bruijn G_k e um inteiro $k \geq 2$.
Saída: um grafo de sequência simples $G = (V, A)$.

```
1 V, A ← ∅;
2 para cada vértice u de G_k faça
      /* v_{u_1} representa o primeiro caractere do k-mer do vértice u      */
3     V ← V ∪ {v_{u_1}};
4     para i = 1 até k − 1 faça
          /* v_{u_{i+1}} representa o caractere k[i+1] do k-mer do vértice u */
5         V ← V ∪ {v_{u_{i+1}}};
6         A ← A ∪ {(v_{u_i}, v_{u_{i+1}})};
7     para cada vértice adjacente u' de u em G_k faça
8         A ← A ∪ (v_{u_k}, v_{u'_k});
9 devolve G = (V, A)
```

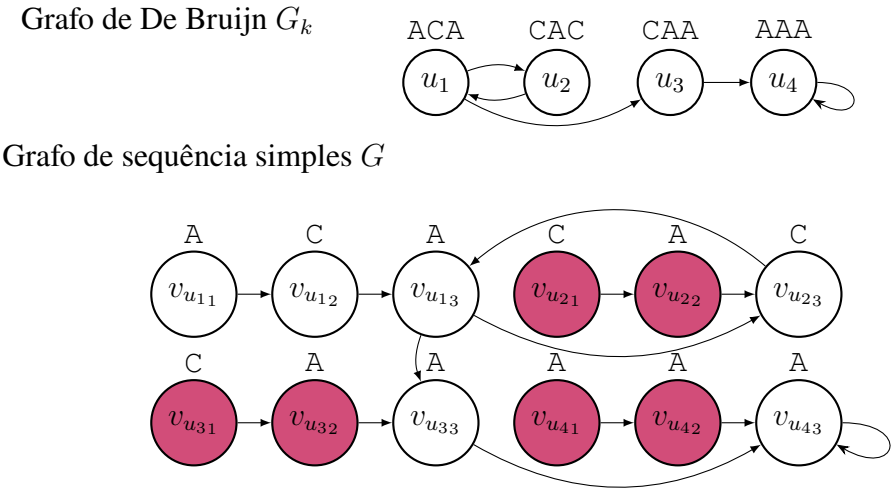


Figura 4.2: Exemplo da conversão de um grafo de De Bruijn G_k, onde $k = 3$, para um grafo de sequência simples G. Neste exemplo, os vértices roxos $v_{u_{21}}, v_{u_{22}}, v_{u_{31}}, v_{u_{32}}, v_{u_{41}}$ e $v_{u_{42}}$ podem ser removidos para reduzir a quantidade de vértices de G.

4.1.2 Algoritmo redutor de vértices

Como vimos anteriormente, a partir de um grafo de De Bruijn G_k, podemos transformá-lo em um grafo de sequência simples G. Agora, vamos ver uma transformação para obter um grafo G com menos vértices. Essa conversão foi desenvolvida neste trabalho e referida como De Bruijn To smaller simple sequence Graph Tool – `BTGT` e os passos são detalhados no Algoritmo 4. Resumidamente, o `BTGT` consiste na transformação de um grafo de De Bruijn G_k em um grafo de De Bruijn equivalente G'_k com o Algoritmo 2 e, em seguida, a conversão de G'_k em um grafo de sequência simples G com o Algoritmo 3. Essa sequência de passos garante a preservação das informações e resulta em um grafo de sequência simples G que induz as mesmas sequências que o grafo de De Bruijn G_k. Para tornar a escrita mais simples nos algoritmos e na prova assumimos que um vértice e seus símbolos são equivalentes, como empregado na Seção 4.1.1.

Dados dois passeios $p = v_1, v_2, \ldots, v_n$ e $p' = u_1, u_2, \ldots, u_m$ em um grafo de De Bruijn G_k, definimos $seq_1(p) = v_1 v_2[k] \ldots v_n[k]$ e $seq_2(p') = u_1[k] u_2[k] \ldots u_m[k]$. Dizemos que G_k e G'_k são **equivalentes**, se para cada $s \in \Sigma^*$, existe um passeio p em G_k tal que $s = seq_1(p)$ se e somente se existe um passeio p' em G'_k tal que $s = seq_2(p')$. Então, definimos o seguinte problema:

Problema 9. *Dado G_k, encontrar um G'_k, tal que G_k e G'_k são equivalentes.*

Dado grafo de De Bruijn G'_k pode ser obtido com os seguintes passos:

- Seja G'_k uma cópia de G_k com todos os seus vértices e considere uma função bijetora *b* que relaciona os vértices correspondentes u de G_k e v de G'_k (ou seja *b*$(u) = v$ implica que *b*$(v) = u$ e o rótulo de v é igual ao rótulo de u);
- para cada vértice v em G'_k com grau de entrada *ge*$(v) = 0$ insira em G'_k $k - 1$ vértices:

  v_{v_1}, com o rótulo $\star\, v[1, k-1]$;

  v_{v_2}, com o rótulo $\star\star v[1, k-2]; \ldots;$

  $v_{v_{k-1}}$, com o rótulo $\star^{k-1}\, v[1]$.

- os seguintes arcos serão induzidos por esses novos vértices:

  $$(v_{v_{k-1}}, v_{v_{k-2}}), \ldots, (v_{v_2}, v_{v_1}), (v_{v_1}, v),$$

  onde $\star \notin \Sigma$ e $i \geq 1$. Cabe observar que, a cada nova inserção dos $k - 1$ vértices seus rótulos podem já estar presente em G'_k, portanto, não é necessário criar um novo vértice para esses rótulos e realizar apenas a criação dos arcos necessários.

Um exemplo de um grafo de De Bruijn G_3 equivalente a um grafo de De Bruijn G'_3, com seus arcos induzidos, é mostrado na Figura 4.3 e o código correspondente é apresentado no Algoritmo 2. A demonstração da prova de que os dois grafos de De Bruijn são equivalentes é apresentada no Teorema 2.

Teorema 2. *G_k e G'_k são equivalentes.*

Demonstração. Seja G_k um grafo de De Bruijn cujos k-mers foram construídos sobre um alfabeto Σ. Seja $G'_k = (V', A')$ um grafo construído a partir de G_k conforme o Algoritmo 2 e com seus k-mers construídos sobre o alfabeto $\Sigma' = \Sigma \cup \{\star\}$. Seja um passeio $p = u_1, u_2, \ldots, u_n$ em G_k que induz a sequência s. Por construção, todos os vértices em p também estão presentes em G'_k. Assim, seja $c = v_1, v_2, \ldots, v_n$ um passeio em G'_k. Dado que todos os vértices originais de G_k estão presentes em G'_k e um caminho de comprimento $k - 1$ foi adicionado em G'_k para todos os vértices de grau de entrada zero, podemos afirmar que antes do vértice v_1, existe um passeio com pelo menos $k - 1$ vértices. Seja $c' = v'_1, \ldots, v'_{k-1}$ em G'_k esse passeio tal que

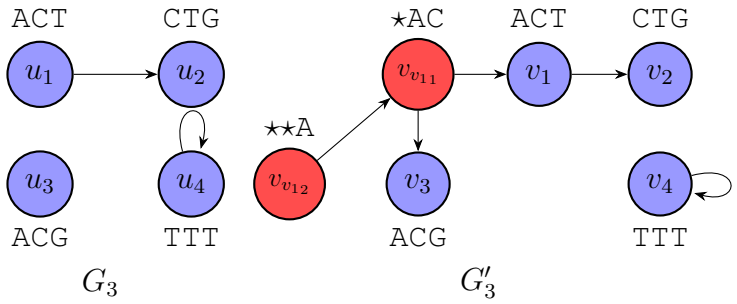


Figura 4.3: Exemplo de conversão de um grafo de De Bruijn G_3 para um grafo de De Bruijn equivalente G'_3. É importante notar que $ge(v_1) = 0$ e $ge(v_2) = 0$, o que resultou na criação dos vértices $v_{v_{11}}$ e $v_{v_{12}}$. Além disso, ao inserir os vértices relacionados a v_3, notamos que os vértices com os mesmos rótulos já existiam, levando-nos a reutilizar esses vértices. Observa-se que, para toda sequência induzida s por um passeio p em G_3, existe um passeio p' em G'_3 tal que $seq_1(p) = s = seq_2(p')$.

Algoritmo 2: De Bruijn to De Bruijn Equivalent Tool

Entrada: um grafo de De Bruijn $G_k = (V, A)$ e um inteiro $k \geq 2$.
Saída: um grafo de De Bruijn equivalente $G'_k = (V', A')$ com os arcos induzidos definidos em A'.

```
1  V' ← V;
2  A' ← A;
3  para v ∈ G_k faça
4      se ge(v) = 0 // grau de entrada igual a zero
5      então
6          para j de k − 1 até 1 faça
7              kmer ← ⋆^{k−j} v[1, j]; // k-mer com k − j estrelas no prefixo
8              kmer_adj ← ⋆^{k−j−1} v[1, j + 1];
9              V' ← V' ∪ {kmer};
10             A' ← A' ∪ (kmer, kmer_adj);
11 devolve G'_k = (V', A')
```

existe o arco $(v'_{k-1}, v_1) \in A'$. Portanto temos o passeio $v'_1, \ldots, v'_{k-1}, v_1, \ldots, v_n$ que induz uma sequência s', logo

$$\begin{aligned} s = seq_1(p) &= u_1 u_2[k] \ldots u_{n-1}[k] u_n[k] \\ &= v'_1[k] \ldots v'_{k-1}[k] v_1[k] \ldots v_n[k] = seq_2(p') = s' \end{aligned}$$

Considerando que sempre utilizamos o último caractere de cada k-mer de G'_k na função seq_2, o símbolo $\star$ não está presente na indução de nenhuma sequência, e portanto, afirmamos que é um símbolo neutro. Dito isso, concluímos que todas as sequências induzidas no grafo G_k também são induzidas no grafo G'_k.

Considere agora um $k > 0$ e $n > k$ e um passeio $p' = v'_1, \ldots, v'_{k-1} v_1, \ldots, v_n$ em G'_k e a sequência s' de p'. Logo temos o passeio $p = u_1 \ldots u_n$ em G_k e sua sequência s. Logo

$$\begin{aligned} s' = seq_2(p') &= v'_1[k] \ldots v'_{k-1}[k] v_1[k] \ldots v_n[k] \\ &= u_1 u_2[k] \ldots u_{n-1}[k] u_n[k] = seq_1(p) = s \end{aligned}$$

portanto G_k e G'_k são equivalentes. □

O grafo de De Bruijn G'_k resultante da conversão anterior pode ser facilmente convertido em um grafo de sequência simples G. Para essa conversão podemos simplesmente considerar o k-ésimo caractere de cada vértice de G'_k e obter o grafo de sequência simples. Um exemplo dessa conversão é mostrado na Figura 4.4. Observe que o conjunto de vértices e arestas são os mesmos em G'_k e G. Dito isso, para um vértice v de G, considere uma **função** $\boldsymbol{d}(v)$ que devolve o k-ésimo caractere do rótulo do vértice correspondente v em G'_k. O código correspondente é apresentado no Algoritmo 3. A demonstração da prova de que G'_k e G induzem as mesmas sequências é apresentada no Teorema 3.

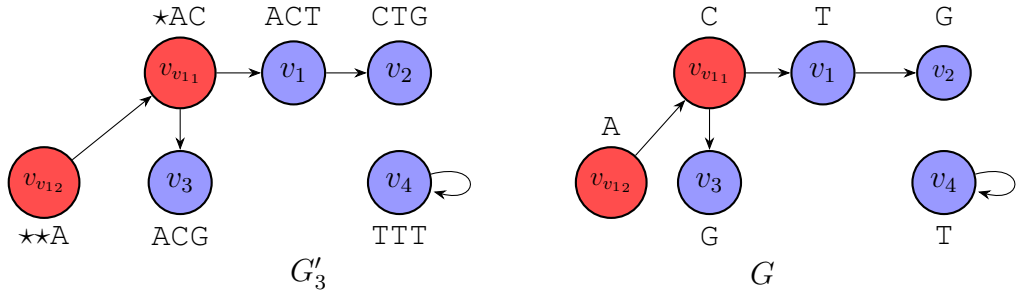


Figura 4.4: Exemplo de conversão de um grafo de De Bruijn G'_3 para um grafo de sequência simples G. Observe que essa conversão mantém a estrutura do grafo e apaga, de cada rótulo dos vértices, o prefixo de comprimento $k-1$.

Algoritmo 3: De Bruijn Equivalent to Smaller Simple Sequence Graph Tool

Entrada: um grafo de De Bruijn $G_k = (V, A)$ resultante do Algoritmo 2 e um inteiro $k \geq 2$.
Saída: um grafo de sequência simples $G = (V, A)$.
1 **para** ***cada*** $v \in V$ **faça**
2 Remover o prefixo de comprimento $k-1$ de v;
3 **devolve** $G = (V, A)$

Teorema 3. *Todas as sequências de comprimento* $\geq k$ *induzidas no grafo de De Bruijn* G'_k *também são induzidas no grafo de sequência simples* G.

Demonstração. Seja $G'_k = (V', A')$ o grafo de De Bruijn construído a partir do Algoritmo 2. Seja $G = (V, A)$ o grafo de sequência simples construído a partir de G'_k conforme o Algoritmo 3. Seja um passeio $p = v_1, v_2, \ldots, v_n$ em G'_k, com $n \geq k$, que induz a sequência $s = seq_2(p)$. Como G'_k e G possuem o mesmo conjunto de vértice e arcos, existe um passeio correspondente p' em G. Seja $p' = v'_1, v'_2, \ldots, v'_n$ o passeio correspondente em G que induz a sequência $s' = v'_1[1] \ldots v'_n[1]$, logo:

$$\begin{aligned} s &= v_1[k] \ldots v_n[k] \\ &= d(v_1) \ldots d(v_n) \\ &= v'_1[1] \ldots v'_n[1] = s' \end{aligned}$$

portanto, as sequências possíveis de serem induzidas no grafo G'_k também podem ser induzidas no grafo G. Esse resultado garante que é possível converter um grafo de De Bruijn G_k para um grafo de sequência simples G, com todas as sequências de comprimento $\geq k$ induzidas no grafo G_k sendo induzidas no grafo de sequência simples G. □

As demonstrações anteriores confirmam a equivalência entre um grafo de De Bruijn e um grafo de sequência simples obtido através do Algoritmo 4. Nesse processo, todas as sequências induzidas no grafo de De Bruijn também são induzidas no grafo de sequência simples resultante. Além disso, o grafo de sequência simples gerado possui um número menor de vértices em comparação ao grafo de sequência simples gerado pelo Algoritmo 1, conforme descrito na Seção 4.1.1. Considerando um grafo de De Bruijn G_k com n vértices, o Algoritmo 1 divide cada vértice em k outros vértices, resultando em um grafo de sequência simples com $n \cdot k$ vértices. Como observado no algoritmo redutor de vértices apresentado na Seção 4.1.2, temos que, para cada vértice com grau de entrada zero, o Algoritmo 4 adiciona $k-1$ novos vértices. Isso resulta em um grafo de sequência simples com um total de $O(n \cdot (k-1)+n)$ vértices. Por outro lado, um grafo de De Bruijn construído a partir de todos os k-mers para um determinado valor de k não possui vértices com grau de entrada zero. Por essa razão, a conversão pode resultar em um grafo com $\Omega(n)$ vértices. A quantidade x de vértices resultantes obtido pelo Algoritmo 4 está no intervalo $n \leq x \leq n \cdot k$.

Algoritmo 4: DE BRUIJN TO SMALLER SIMPLE SEQUENCE GRAPH TOOL – BTGT

Entrada: um grafo de De Bruijn $G_k = (V, A)$ e um inteiro $k \geq 2$.
Saída: um grafo de sequência simples G.

```
1 G'_k ←ALGORITMO 2(G_k, k); // Construir grafo De Bruijn equivalente
2 G ← ALGORITMO 3(G'_k, k); // converter G'_k para grafo de sequência simples
3 devolve G
```

Para determinar a complexidade de tempo do Algoritmo 4 considere que $|V| = n$, assim a complexidade de tempo é $O(n \cdot k)$ para o Algoritmo 2 e $O(n \cdot k)$ para o Algoritmo 3, portanto a complexidade do Algoritmo 4 é $O(n \cdot k) + O(n \cdot k) = O(2 \cdot n \cdot k) = O(n \cdot k)$.

Grafo de De Bruijn sem vértices com grau de entrada zero

Um grafo de De Bruijn G_k pode não possuir vértices com grau de entrada zero, ou seja, todos os vértices têm pelo menos um arco de entrada. Nesse caso, a conversão redutora cria uma cópia do grafo de De Bruijn G_k, resultando no grafo G'_k, sem adicionar novos vértices. Pelo Teorema 2, os grafos são equivalentes (as sequências induzidas no grafo G_k também são induzidas no grafo G'_k). O segundo passo converte G'_k em um grafo de sequência simples G, considerando apenas o k-ésimo caractere de G'_k. De acordo com o Teorema 3, as sequências induzidas no grafo G'_k também são induzidas no grafo G. Para exemplificar como o grafo de sequência simples G se configura quando não há vértices com grau de entrada zero no grafo de De Bruijn, a Figura 4.5 apresenta a conversão do grafo de De Bruijn mostrado na Imagem 4.2.

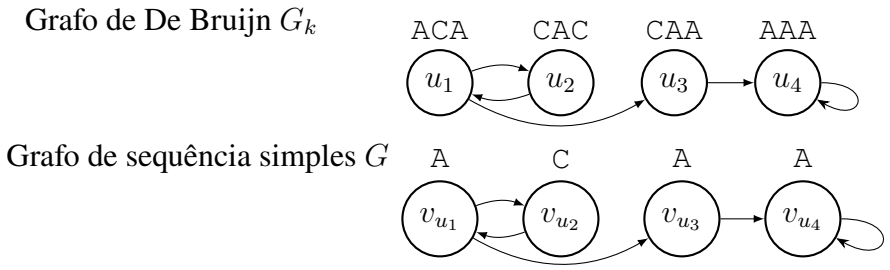


Figura 4.5: Exemplo da conversão de um grafo de De Bruijn G_k (sem vértices com grau de entrada zero), onde $k = 3$, para um grafo de sequência simples G. Neste exemplo, nenhum novo vértice é inserido, e os vértices têm seus rótulos alterados considerando o k-ésimo caractere de cada k-mer.

Teste para validar a quantidade de vértices nos grafos

Para validar as implementações, testes com a quantidade de vértices obtidos no grafo de sequência simples G tanto pela conversão ingênua quanto pela conversão redutora são mostrados na Tabela 4.1 para $k = 3, 4, 5$ e 10. Observa-se que a conversão redutora gerou um grafo de sequência simples com um número menor de vértices em comparação à conversão ingênua.

	Conversão Ingênua		Conversão Redutora		
k	G_k	G	G_k	G'_k	G
3	36	108	36	36	36
4	61	244	61	61	61
5	82	410	82	86	86
10	122	1220	122	139	139

Tabela 4.1: Comparação prática da quantidade de vértices: conversão ingênua vs. redutora.

O grafo de sequência simples obtido através do Algoritmo 4 não é mínimo

A conversão realizada pelo Algoritmo 4 pode gerar um grafo com um número inferior de vértices em comparação com a abordagem ingênua adotada pelo Algoritmo 1. No entanto, é importante notar que essa redução não garante a minimização do número total de vértices no grafo resultante.

Para ilustrar essa diferença, apresentamos um exemplo do grafo de sequência simples G obtido pelo Algoritmo 4, seguido por um contraexemplo que demonstra a possibilidade de diminuir a quantidade de vértices em G, evidenciando assim que G não é um grafo mínimo em termos de número de vértices.

As Figuras 4.6 e 4.7 apresentam os passos do Algoritmo 4 para converter o grafo de De Bruijn em um grafo de sequência simples. Na Figura 4.8, ilustramos um exemplo em que é possível obter um grafo com menos vértices. Nesse cenário específico, torna-se evidente que ambos os grafos de sequência simples G e T na Figura 4.8 induzem as mesmas sequências, confirmando que G não é mínimo. No contraexemplo fornecido, os vértices $v_{v_{32}}$, v_3, e v_4 podem ser removidos, uma vez que os vértices $v_{v_{12}}$, v_1, e v_2 já representam os rótulos de cada vértice, respectivamente. Dessa forma, o grafo T continua a induzir as mesmas sequências que o grafo

G. É fácil notar que todos os k-mers de G_3 (ACT, CTA, TAC, AGT e GTA) são induzidos por caminhos tanto por G quanto por T.

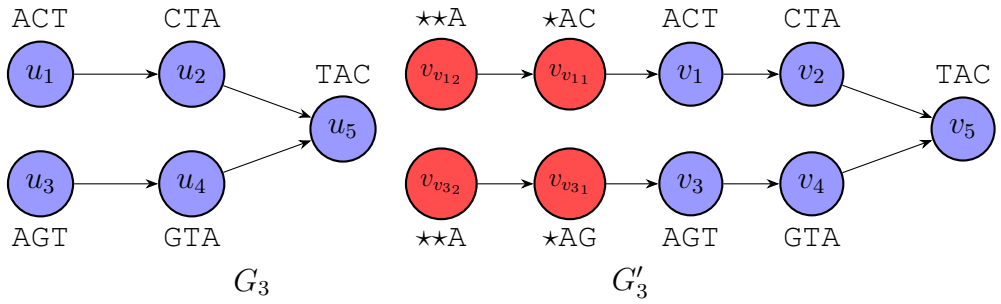


Figura 4.6: Exemplo de conversão de um grafo de De Bruijn G_3 para um grafo de De Bruijn equivalente G'_3.

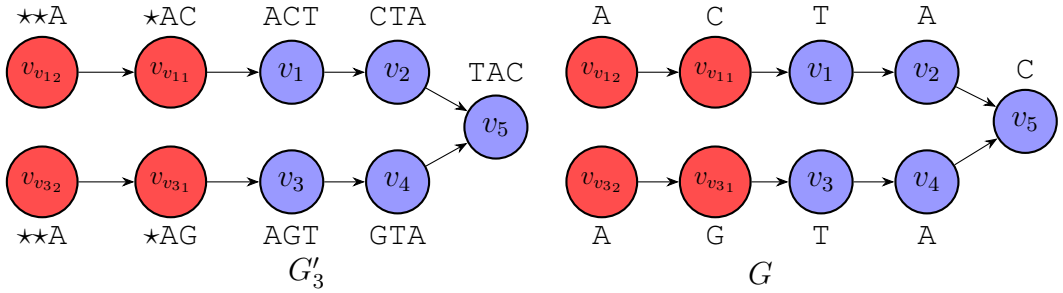


Figura 4.7: Exemplo de conversão de um grafo de De Bruijn G'_3 para um grafo de sequência simples G. Observe que essa conversão mantém a estrutura do grafo e apaga, de cada rótulo dos vértices, o prefixo de comprimento $k-1$.

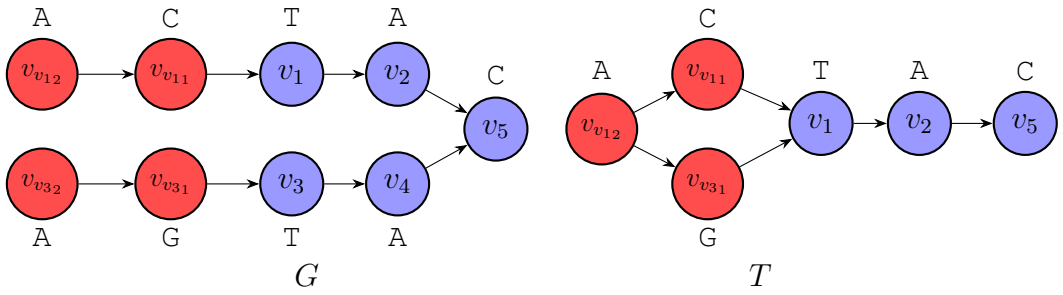


Figura 4.8: Exemplo de um grafo de sequência simples G obtido pelo Algoritmo 4. É importante observar que G não é mínimo. À direita, apresentamos um grafo de sequência simples T que contém menos vértices e, ao mesmo tempo, induz as mesmas sequências que G.

CAPÍTULO

5

Abordagens para o Problema do Mapeamento de Sequências em Grafos de De Bruijn com Mudanças na Sequência

Nesta seção, apresentamos uma adaptação baseada no algoritmo `RaMa` [Rautiainen and Marschall (2017)], que permite executar o mapeamento em grafos de De Bruijn com mudanças apenas na sequência. Adicionalmente, introduzimos uma heurística para o mesmo problema baseada no mesmo algoritmo `RaMa`, assim como duas outras heurísticas que utilizam a técnica *seed-and-extend*.

5.1 Algoritmo para mapear uma sequência em um grafo de De Bruijn

O algoritmo proposto para o $\texttt{PMSB}_{p_1}$, apresentado neste contexto, segue a mesma lógica do algoritmo `RaMa`. No entanto, incorpora um passo adicional de pré-processamento que converte o grafo de De Bruijn G_k em um grafo de sequência simples G. Dado o grafo de sequência simples G resultante da conversão detalhada na Seção 4.1, a solução para o $\texttt{PMSB}_{p_1}$ pode então ser obtida executando o algoritmo `RaMa` para encontrar o mapeamento ou o algoritmo $\texttt{RaMa}_d$ para calcular apenas a distância do mapeamento. Ambas as versões foram desenvolvidas e são referidas como DE BRUIJN SEQUENCE MAPPING TOOL – `BSMT` (ou $\texttt{BSMT}_d$).

5.1.1 Algoritmo para o mapeamento

Os passos para mapear uma sequência em um grafo de De Bruijn estão detalhados no Algoritmo 5 denominado neste trabalho De Bruijn Sequence Mapping Tool – BSMT. Dado um grafo de De Bruijn $G_k = (V, A)$ e uma sequência s, o BSMT consiste em converter o grafo de De Bruijn em um grafo de sequência simples $G = (V', A')$. Logo em seguida, constrói-se o grafo de multicamadas G' a partir de G e s, e então executa-se o Algoritmo RaMa com G' como entrada. O Algoritmo RaMa devolve um caminho $c = (v_i, 1), (v_i, 2), \ldots, (v_i, |s|)$ no grafo de multicamadas G', onde $v_i \in V'$. Através do caminho c, podemos obter o passeio p em G, e consequentemente um passeio p' em G_k, onde p e p' induzem a mesma sequência s' e a distância de edição entre s e s' é mínima.

Algoritmo 5: De Bruijn Sequence Mapping Tool – BSMT

Entrada: um grafo de De Bruijn $G_k = (V, A)$ e uma sequência s.
Saída: um passeio p' em G_k tal que a distância de edição entre a sequência induzida s' por p' e s é mínima.

1 $G \leftarrow \texttt{BTGT}(G_k)$; // Converter G_k em um grafo de sequência simples
2 $G' \leftarrow$ Construir o grafo de multicamadas a partir de G e s;
3 $c \leftarrow$ Executar o algoritmo RaMa em G';
4 $p' \leftarrow$ Obter Passeio(G_k, G, c);
5 **devolve** p'

Algoritmo 6: Obter Passeio

Entrada: um grafo de De Bruijn $G_k = (V, A)$, o grafo de sequência simples $G = (V', A')$ obtido a partir de G_k, e um caminho c onde cada vértice é no formato (v_i, j) com $v_i \in V'$.
Saída: um passeio p' em G_k construído a partir do caminho c.

1 $p \leftarrow \emptyset$;
2 **para** $(v_i, j) \in c$ **faça**
3 $p \leftarrow v_i, p$; // Esses vértices formam um passeio em G
4 $p' \leftarrow \emptyset$;
5 **para** $v_i \in p$ **faça**
  /* É necessário uma estrutura auxiliar que associe cada vértice de G ao seu vértice original em G_k, indicando qual k-mer o vértice v_i representa em G_k */
6 $u \leftarrow$ vértice em G_k referente a v_i;
7 **se** u *não está presente em* p' **então**
8 $p' \leftarrow u, p'$; // Esses vértices formam um passeio em G_k
9 **devolve** p'

Para determinar a complexidade de tempo do Algoritmo 5 considere uma sequência s de comprimento m, um grafo de De Bruijn $G_k = (V, A)$ com $|V| = n$, o grafo de sequência simples $G = (V', A')$ obtido de G_k com $|V| = n'$ e $|A'| = l$, e o grafo de multicamadas G' obtido a partir de s e G. Assim, a complexidade de tempo para construir o grafo de sequência simples é

$O(n \cdot k)$ na linha 1. Na linha 2, a construção do grafo de multicamadas tem uma complexidade de tempo $O(m \cdot (n' + l))$. A execução do Algoritmo RaMa na linha 3 para obter o caminho c em G' tem uma complexidade de tempo $O(m \cdot (l + n' \cdot \log(l + m)))$. Finalmente, para o Algoritmo 6, na linha 4, obter o passeio p' em G_k a partir do caminho c tem complexidade de tempo $O(2 \cdot n') = O(n')$. Portanto, a complexidade de tempo total é dada por

$$\begin{aligned} &= O(n \cdot k) + O(m \cdot (n' + l)) + O(m \cdot (l + n' \cdot \log(l + m))) + O(n') \\ &= O(n \cdot k + m \cdot (n' + l) + m \cdot (l + n' \cdot \log(l + m)) + n'); \end{aligned}$$

O principal problema do BSMT reside na criação do grafo multicamadas G'. Em situações práticas, um grafo de De Bruijn pode conter milhares de k-mers. Por exemplo, com $k = 32$, existiriam $|\Sigma|^{32}$ k-mers possíveis, onde $|\Sigma|$ representa o tamanho do alfabeto, e as sequências mapeadas costumam ter comprimentos superiores a 10.000 caracteres. Ao construir o grafo multicamadas, é necessário criar uma cópia do grafo de De Bruijn convertido em um grafo de sequência simples para cada caractere na sequência s. No entanto, o armazenamento de um número tão grande de grafos torna-se impraticável. Por essa razão, a utilização de heurísticas que utilizem um grafo multicamadas menor é crucial e pode viabilizar o mapeamento em cenários do mundo real.

5.1.2 Algoritmo para calcular apenas a distância do mapeamento

Os passos para calcular a distância de mapear uma sequência em um grafo de De Bruijn estão detalhados no Algoritmo 7 denominado neste trabalho como De Bruijn Sequence Mapping Tool$_{distance}$ – BSMT$_d$. Dado um grafo de De Bruijn $G_k = (V, A)$ e uma sequência S, o BSMT$_d$ consiste em converter o grafo de De Bruijn em um grafo de sequência simples $G = (V', A')$, e então executa-se o Algoritmo RaMa$_d$ para o obter a apenas distância de mapear s em G_k.

Algoritmo 7: De Bruijn sequence Mapping Tool$_{distance}$ – BSMT$_d$

Entrada: um grafo de De Bruijn $G_k = (V, A)$ e uma sequência s.
Saída: a distância d de um passeio p em G_k tal que a distância de edição entre a sequência induzida s' por p e s é mínima.

1 $G \leftarrow$ BTGT(G_k)
2 $d \leftarrow$ Executar o algoritmo RaMA$_d$ em G;
3 **devolve** d

Para determinar a complexidade de tempo do Algoritmo 7 considere uma sequência s de comprimento m, um grafo de De Bruijn $G_k = (V, A)$ com $|V| = n$ e o grafo de sequência simples $G = (V', A')$ obtido de G_k com $|V'| = n'$ e $|A'| = l$. Assim a complexidade de tempo para construir o grafo de sequência simples é $O(n \cdot k)$ na linha 1 e a complexidade de tempo para executar o algoritmo RaMa$_d$ na linha 3 é $O(|n'| + m \cdot l)$. Portanto, a complexidade de tempo total é dada por $O(n \cdot k) + O(|n'| + m \cdot l) = O(n \cdot k + |n'| + m \cdot l) = O(|n'| + m \cdot l)$.

O $\texttt{BSMT}_d$, ao contrário do BSMT, possui a capacidade de lidar com casos do mundo real, pois a ferramenta não exige o armazenamento completo do grafo multicamadas na memória. Entretanto, o principal desafio reside na necessidade de atualizar todos os arcos ao construir uma nova camada do grafo multicamadas, o que se torna um gargalo em termos de tempo. Portanto, heurísticas que visem reduzir o tempo de execução do $\texttt{BSMT}_d$ também são altamente desejáveis.

5.2 Heurísticas para o problema do mapeamento de sequências em grafos de De Bruijn

5.2.1 Heurística 1

A ideia abordada nesta seção tem como objetivo uma melhoria direta nas soluções do BSMT e $\texttt{BSMT}_d$.

Heurística para o $\texttt{PMSB}_{p_1}$ em busca do mapeamento

Dada uma sequência s e um grafo de De Bruijn G_k, a ideia da nossa primeira heurística para o $\texttt{PMSB}_{p_1}$, chamada aqui de De Bruijn sequence Mapping Tool$_{h1}$ – $\texttt{BSMT}_{h1}$, é ancorar alguns k-mers da sequência s no grafo e usar o BSMT para mapear apenas as subcadeias de s que não puderam ser ancoradas. Em maiores detalhes, o primeiro passo do $\texttt{BSMT}_{h1}$ é procurar por todos os k-mers em s que estão presentes em G_k, e a esses k-mers chamamos de **sementes** (*seed* em inglês). Note que, após essa etapa, podemos ter várias subcadeias de s formadas por k-mers da sequência que não estão em G_k, que chamamos de ***gaps***. Para cada uma desses *gaps* delimitados por duas sementes, a heurística usa o BSMT para encontrar um caminho que, partindo da semente imediatamente à esquerda do *gap* e terminando na semente imediatamente à direita do *gap*, preencha-a da melhor maneira possível.

As Figuras 5.1 e 5.2 ilustram um exemplo dos passos realizados pelo $\texttt{BSMT}_{h1}$. No primeiro passo (Figura 5.1), são identificadas todas as sementes, enquanto no segundo passo (Figura 5.2), o BSMT é executado para encontrar os melhores passeios entre duas sementes. As linhas onduladas representam os passeios (encontrados com o algoritmo BSMT) que preenchem melhor cada *gap* (partindo da primeira semente e terminando na segunda semente). Os passos do $\texttt{BSMT}_{h1}$ são detalhados na Heurística 1. O Encontra sementes, descrito no Pseudocódigo 1, devolve uma sequência de sementes $\mathcal{A}$, e para cada par de sementes consecutivas $a, a' \in \mathcal{A}$ com um *gap* entre elas, o $\texttt{BSMT}_{h1}$ utiliza o BSMT para encontrar o melhor passeio de a até a'.

A utilização do grafo de De Bruijn completo durante a execução do BSMT (linha 5 no $\texttt{BSMT}_{h1}$) é computacionalmente custosa. Com o objetivo de reduzir o consumo de memória por parte dessa heurística, o $\texttt{BSMT}_{h1}$ considera apenas um subconjunto de $V(G_k)$ na busca pelo melhor passeio entre duas sementes consecutivas: considerando que o gap possui tamanho N, apenas os vértices que estão a uma distância de $0.5 \cdot N$ em termos de seus arcos, a partir dos vértices de origem e destino, são considerados. Levando em conta essa restrição, a heurística é capaz de processar instâncias maiores, mas o passeio resultante pode não ser capaz de cobrir a

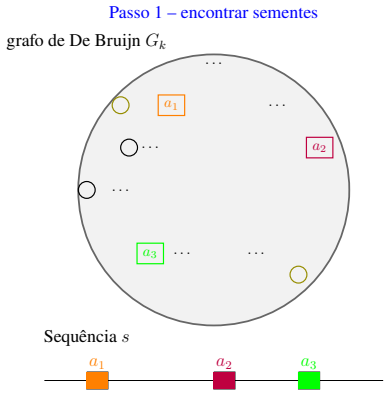


Figura 5.1: Primeiro passo – cada quadrado é uma semente encontrada em G_k e os círculos são outros k-mers que não são sementes.

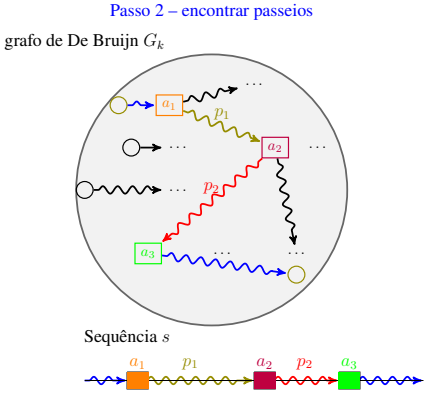


Figura 5.2: Segundo passo – As linhas onduladas entre duas sementes são os melhores passeios encontrado com o BSMT.

Heurística 1: De Bruijn Mapping Tool$_{h1}$ –BSMT$_{h1}$

Entrada: uma sequência s e um grafo de De Bruijn G_k.
Saída: um bom passeio p que cobre a sequência s.

```
A ← ENCONTRA SEMENTES(G_k, s, k);
p_temp, p_best ← ∅;
para cada par de sementes consecutivas a, a' em A com um gap entre elas faça
    q ← subcadeia de s de a até a'; // Contendo as duas sementes a e a'
    p ← BSMT(G_k, q); // Começando da semente a e terminando em a'
    se p é encontrado então p_temp ← p_temp p; // Concatenar os passeios
    se p não é encontrado então
        se len(p_temp) ≥ len(p_best) então
            p_best ← p_temp;
        p_temp ← ∅;
devolve p_best
```

Pseudocódigo 1: ENCONTRA SEMENTES

Entrada: uma sequência s, um grafo de De Bruijn G_k e um inteiro $k \geq 2$
Saída: uma sequência de sementes (k-mers que estão em s e G_k) $\mathcal{A}$.

```
A ← ∅;
para i = 1 até |s| − (k − 1) faça
    q ← s[i, i + k];
    se q ∈ G_k então A ← A, q;
devolve A
```

sequência inteira, já que o BSMT pode não encontrar um passeio a partir dos vértices de origem e destino. Nesse caso, consideramos como resposta final o mapeamento mais longo (linhas 6–10 no $\texttt{BSMT}_{h1}$).

A principal vantagem do $\texttt{BSMT}_{h1}$ em relação ao BSMT é a quantidade de memória utilizada e o tempo de processamento. Para o BSMT, precisamos construir um grafo de multicamadas considerando a sequência completa e o grafo de De Bruijn completo, enquanto para o $\texttt{BSMT}_{h1}$ construímos o grafo de multicamadas para a subcadeia dos *gaps* e consideramos apenas um subgrafo de G_k. Para o $\texttt{BSMT}_{h1}$, dado que $|s| = m$, podemos encontrar as sementes em tempo $O(k)$ e leva-se tempo $O(m \cdot k)$ (o que significa encontrar todas as sementes de s). Dado que foram encontradas N sementes de comprimento $m' \leq |s| = m$ e temos a complexidade de tempo $O(n'+l)+m' \cdot (l+n' \cdot \log(l+m'))+n'))$ para executar o BSMT (ver Seção 5.1.1) para cada *gap*, a complexidade de tempo da heurística é $O(N \cdot (m' \cdot (n'+l)+m' \cdot (l+n' \cdot \log(l+m'))+n'))$.

Heurística para o $PMSB_{p_1}$ em busca da distância do mapeamento

Também desenvolvemos o De Bruijn sequence Mapping Tool$_{h1_d}$ – $\texttt{BSMT}_{h1_d}$ usando a mesma ideia do $\texttt{BSMT}_{h1}$. O $\texttt{BSMT}_{h1_d}$ ancora alguns k-mers da sequência s no grafo e utiliza o $\texttt{BSMT}_d$ para encontrar a distância do melhor caminho. Em mais detalhes, o primeiro passo do $\texttt{BSMT}_{h1_d}$ é procurar todas as sementes (k-mers) em s que estão presentes em G_k. Note que, após essa etapa, também podemos ter vários *gaps*. Para cada um desses *gaps* delimitados por duas sementes, a heurística utiliza o $\texttt{BSMT}_d$ para encontrar a distância do melhor passeio, começando na semente imediatamente à esquerda do *gap* e terminando na semente imediatamente à direita do *gap*. Aqui, o $\texttt{BSMT}_{h1_d}$ também considera apenas um subconjunto de $V(G_k)$ na busca pela distância do melhor caminho entre duas sementes consecutivas e considera como resposta final a distância do mapeamento mais longo. O $\texttt{BSMT}_d$ é capaz de processar entradas grandes, mas leva muito tempo para ser executado, e a principal vantagem do $\texttt{BSMT}_{h1_d}$ em relação ao $\texttt{BSMT}_d$ está no tempo de execução, pois executamos o $\texttt{BSMT}_d$ apenas para os *gaps*, o que é vantajoso. Para o $\texttt{BSMT}_{h1_d}$, dado que $|s| = m$ podemos encontrar as sementes em tempo $O(k)$ e leva $O(m \cdot k)$ para encontrar todas as sementes. Dado que foram encontradas N sementes de comprimento $m' \leq |s| = m$ e temos a complexidade de tempo $O(n' + m' \cdot l)$ para executar o $\texttt{BSMT}_d$ (ver Seção 5.1.2) para cada *gap*, a complexidade de tempo da heurística é $O(N \cdot ((n' + m' \cdot l)))$.

Utilizando o Bifrost nas heurísticas $BSMT_{h1}$ e $BSMT_{h1_d}$

Para melhorar o desempenho do $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_d}$, desenvolvemos duas versões chamadas $\texttt{BSMT}_{h1_b}$ e $\texttt{BSMT}_{h1_{db}}$ para o $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_d}$, respectivamente. Nessas novas versões, construímos os grafos de De Bruijn com o Bifrost – uma ferramenta que permite construir grafos de De Bruijn de forma eficiente e executar algoritmos como a Busca em Largura e o algoritmo de Dijkstra nesses grafos [Holley and Melsted (2020)]. Um ponto relevante ao usar o Bifrost no $\texttt{BSMT}_{h1_b}$ e no $\texttt{BSMT}_{h1_{db}}$ é que o Bifrost possui seu próprio método eficiente de criação do grafo de De Bruijn e uma interpretação própria dos k-mers. Em relação à interpretação dos k-mers, o Bifrost considera como semente tanto o k-mer na ordem original quanto o seu

complemento reverso. Por exemplo, para as bases de DNA `A`, `C`, `G` e `T`, o complemento da base `A` é `T` (e vice-versa), e o complemento da base `C` é `G` (e vice-versa), então o complemento de `ACGTT` é `TGCAA`, e o complemento reverso é `AACGT`. Para o Bifrost, tanto a ordem original quanto o complemento reverso são considerados sementes. Em geral, a quantidade de sementes entre nossas ferramentas com e sem o Bifrost pode variar, o que impacta o resultado final

5.2.2 Heurística 2

A abordagem apresentada nesta seção visa explorar uma alternativa ao mapeamento proposto na Heurística 1. Para essa heurística chamamos de De Bruijn Sequence Mapping Tool$_{h2}$ – `BSMT`$_{h2}$. No entanto, o mapeamento de uma sequência s em um grafo de De Bruijn G_k é realizado utilizando a estratégia *seed-and-extend*.

Dados um grafo de De Bruijn G_k e uma sequência s, o `BSMT`$_{h2}$ consiste nos seguintes passos:

1. Encontrar todas as sementes de comprimento k;
2. Tentar estender todas as sementes para a direita, comparando os caracteres subsequentes com os caracteres percorridos por um passeio no grafo de De Bruijn a partir do vértice semente até que uma diferença seja encontrada;
3. Devolver o passeio mais longo.

Para permitir substituições na sequência s, para cada semente a ser estendida, alteramos seu último caractere considerando todos os caracteres do alfabeto Σ do grafo e, com isso, encontrar um passeio melhor no grafo começando em um novo vértice semente modificado. Se durante a extensão a partir de uma semente chegarmos a outra semente, continuamos a extensão a partir dele e essa nova semente passa a fazer parte da solução. Levando em conta a possibilidade de não atingir uma semente destino, o que permite a existência de vários passeios desconexos, isto é, cada semente pode ter um passeio começando a partir dela e não alcançando nenhuma semente destino durante o processo de extensão. Nesse caso, consideramos como resposta final o mapeamento mais longo.

A Figura 5.3 e 5.4 mostram um exemplo de como o `BSMT`$_{h2}$ prossegue. No primeiro passo (Figura 5.3) determinamos todas as sementes e no segundo e terceiro passo (Figura 5.4) determinamos, com a extensão, o melhor passeio entre duas sementes e o devolvemos. As linhas onduladas são os passeios, e o caminho verde é o melhor passeio encontrado (começando no novo vértice semente modificado e terminando na segunda semente). Os passos do `BSMT`$_{h2}$ são detalhados na Heurística 2. O Encontra sementes, descrito no Pseudocódigo 1, devolve uma sequência de sementes G e o Estende, descrito no Pseudocódigo 2, devolve um passeio p no grafo de De Bruijn referente a extensão feita a partir de uma determinada semente.

Para o `BSMT`$_{h2}$, dado que $|s| = m$ podemos encontrar as sementes em tempo $O(k)$ e gasta-se tempo $O(m \cdot k)$ para encontrar todas as sementes. Dadas N sementes encontradas, um alfabeto $|\Sigma| = M$ e $|A| = l$ a quantidade de arcos no grafo de De Bruijn, a complexidade de tempo desta heurística é $O(N \cdot (M \cdot l))$ para estender todas as sementes.

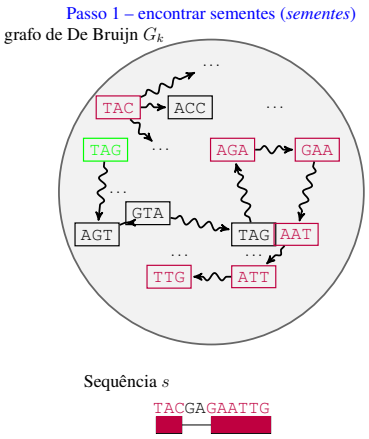


Figura 5.3: Passo 1 – quadrados roxos são sementes encontradas em G_k.

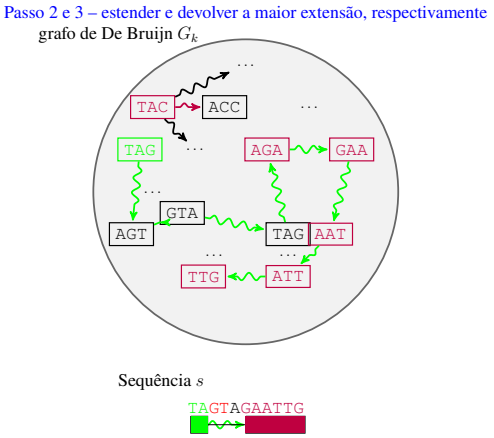


Figura 5.4: Passo 2 – estender alterando o último caractere. Neste exemplo, é preferível mudar o caractere `C` para `G` em s e estender a partir do novo k-mer `TAG`. Em seguida, alteramos `G` para `T` e estendemos a partir do novo k-mer `TAT`, não temos mais alterações porque os demais k-mers são sementes. Passo 3 — devolver a melhor extensão.

Heurística 2: De Bruijn Mapping Tool$_{h2}$ –BSMT$_{h2}$

Entrada: sequência s, grafo de De Bruijn G_k e um alfabeto Σ.
Saída: um bom passeio p que cobre a sequência s.

$\mathcal{A} \leftarrow$ ENCONTRA SEMENTES(G_k, s, k);
$p_{best}, p_{temp} \leftarrow \emptyset$;
para *cada par de sementes consecutivas* a, a' *em* $\mathcal{A}$ *com um* ***gap*** *entre elas* **faça**
    $alcancei \leftarrow Falso$;
    para *cada caractere* $c \in \Sigma$ **faça**
        $q \leftarrow$ subcadeia de s de a até a'; // Contendo as duas sementes a e a'
        $q[k] \leftarrow c$; // Modificando o caractere k de q associada à semente a
        $p \leftarrow$ ESTENDE(G_k, q); // Estende um passeio para alcançar a'
        se a' *está em* p **então**
            $alcancei \leftarrow Verdadeiro$ // a' foi alcançado
            $p_{temp} \leftarrow p_{temp}p$; // p faz parte da solução
            break;
        senão se ***len***$(p) \geq$ ***len***(p_{temp}) **então**
            $p_{temp} \leftarrow p$;
    se $alcancei = Falso$ ***E*** ***len***$(p_{temp}) \geq$ ***len***(p_{best}) **então**
        $p_{best}, p_{temp} \leftarrow p_{temp}, \emptyset$;
se ***len***$(p_{temp}) \geq$ ***len***(p_{best}); **então devolve** p_{temp};
devolve p_{best}

Pseudocódigo 2: ESTENDE

Entrada: uma sequência s, um grafo de De Bruijn G_k e um inteiro $k \geq 2$
Saída: um passeio p em G_k tal que a distância entre a sequência induzida s' por p e s é no máximo 1.

```
1  p ← ∅;
2  existe ← Verdadeiro;
3  i ← 1;
4  enquanto existe = Verdadeiro E i < |s| − (k − 1) faça
5      q ← s[i, i + k];
6      se q ∈ G_k então p ← p, q; // Verificando se o k-mer está presente em G_k
7      senão
8          existe ← Falso; // Uma diferença encontrada
9      i ← i + 1;
10 devolve p // p contém todos os k-mers até uma diferença encontrada
```

5.2.3 Heurística 3

Para melhorar a cobertura do $\texttt{BSMT}_{h2}$, desenvolvemos uma nova heurística que permite substituição e exclusão na sequência mapeada, chamada de $\texttt{BSMT}_{h3}$, que consiste nos seguintes passos:

1. Encontrar todas as sementes de comprimento k;
2. Tentar estender todas as sementes para a direita, comparando os caracteres subsequentes com os caracteres percorridos por um passeio no grafo de De Bruijn a partir do vértice semente até que uma diferença seja encontrada;
3. Se a partir de uma semente a não alcançamos nenhuma outra semente, vamos remover na sequência s os $k-1$ caracteres referente ao sufixo da semente a e realizar uma nova extensão;
4. Devolver o passeio mais longo.

Resumidamente, se durante a etapa de extensão não conseguirmos estender para nenhum passeio, removemos $k-1$ caracteres do sufixo da semente e repetimos o processo de extensão. Para o $\texttt{BSMT}_{h3}$, dado que $|s| = m$, podemos encontrar as sementes em tempo $O(k)$ e leva $O(m \cdot k)$ para encontrar todas as sementes. Dadas N sementes encontradas, um alfabeto $|\Sigma| = M$ e $|A| = l$ a quantidade de arcos no grafo de De Bruijn, a complexidade de tempo é $O(N \cdot (M \cdot (m - (k-1)) \cdot l))$ para estender todas as sementes, onde $m - (k-1)$ é o número de exclusões em s.

No Capítulo 7, apresentamos uma série de testes para comparar os algoritmos propostos e as heurísticas desenvolvidas. Esses testes abrangem diversas situações, permitindo validar tanto o tempo de execução quanto a acurácia das heurísticas em relação aos algoritmos. Apresentamos uma análise detalhada dos resultados para fornecer uma compreensão mais profunda do desempenho das heurísticas, destacando sua eficiência e acurácia em comparação com os algoritmos e outras heurísticas da literatura.

Heurística 3: De Bruijn Mapping Tool$_{h3}$ –BSMT$_{h3}$

Entrada: sequência s, grafo de De Bruijn G_k e um alfabeto Σ.
Saída: bom passeio p cobrindo s.

$\mathcal{A} \leftarrow$ ENCONTRA SEMENTES(G_k, s, k);
$p_{best}, p_{temp} \leftarrow \emptyset$;
para *cada par de sementes consecutivas* a, a' *em* $\mathcal{A}$ *com um* ***gap*** *entre elas* **faça**
    $q \leftarrow$ subcadeia de s de a até a';
    $p_{local} \leftarrow$ BSMT$_{h2}(G_k, q, k)$; // Estende usando o BSMT$_{h2}$
    se a' *não está em* p_{local} **então**
        /* Não alcançamos a semente a' */
        Remover $a[2, k]$ caracteres de q a partir de a;
        $p \leftarrow$ ESTENDE(G_k, q); // Estende um passeio para alcançar a'
        se ***len***$(p) \geq$ ***len***(p_{local}) **então**
            $p_{local} \leftarrow p$; Remover $a[2, k]$ caracteres de s a partir de a;
    se a' *é alcançado* **então**
        $p_{temp} \leftarrow p_{temp}p_{local}$; // p_{local} faz parte da solução
    senão se a' *não é alcançado* **então**
        se ***len***$(p_{temp}) \geq$ ***len***(p_{best}) **então**
            $p_{best} \leftarrow p_{temp}$;
        senão se ***len***$(p_{local}) \geq$ ***len***(p_{best}) **então**
            $p_{best} \leftarrow p_{local}$;
        $p_{temp} \leftarrow \emptyset$; // O passeio em p_{temp} chegou no seu comprimento máximo
devolve p_{best};

CAPÍTULO

6

O Problema do Mapeamento de Sequências em Grafos de De Bruijn com Mudanças no Grafo

Neste capítulo, abordamos uma variante do PROBLEMA DO MAPEAMENTO DE SEQUÊNCIAS EM GRAFOS DE DE BRUIJN, que considera mudanças exclusivamente no grafo. Diferentemente dos autores Gibney *et al.* [Gibney et al. (2022)], que definiram um problema sem permitir que a estrutura do grafo mudasse e provaram sua NP-completude, neste trabalho permitimos que a estrutura do grafo seja alterada. Em resumo, a principal diferença entre este estudo e o trabalho conduzido pelos autores é se permitimos ou não a alteração dos arcos quando alteramos os rótulos dos vértices. Para a variante estudada neste capítulo, formulamos o problema permitindo a indução de novos arcos em um grafo de De Bruijn sempre que o rótulo de um vértice é modificado, e essa flexibilidade nos permite encontrar soluções em tempo polinomial. Dado que o grafo de De Bruijn possibilita a indução de arcos por meio de seus k-mers, apresentamos as definições necessárias para delinear o problema. Por fim, apresentamos um algoritmo polinomial desenvolvido para essa variante do problema, juntamente com sua análise de tempo de execução e sua correção.

6.1 A edição em um grafo de De Bruijn

Seja s uma sequência e G_k um conjunto de vértices rotulados por k-mers representando um grafo de Bruijn (as arestas estão implícitas). Definimos $k(s)$ como o conjunto de k-mers em s e assumimos que $m = |k(s)| \leq |G_k| = n$.

Uma **edição** E com j operações em G_k referente a s, ou simplesmente edição E em G_k,

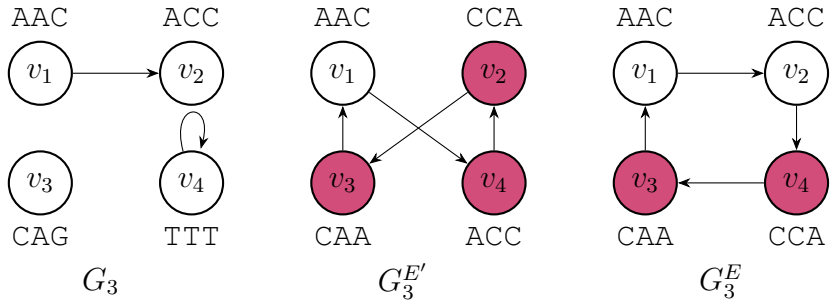
consiste em substituir os rótulos de uma sequência $v_1, v_2 \ldots, v_j$ de vértices distintos em G_k por uma sequência $u_1, \ldots, u_j$ de k-mers distintos em $k(s)$ (cada rótulo v_i é substituído por u_i), de modo que, após a edição, exista um passeio no grafo de Bruijn obtido que induz s. O grafo obtido após a edição E é denotado por G_k^E. Note que todo k-mer em $k(s)$ é rótulo de algum vértice de G_k^E. O **custo da edição** E é

$$c(E) = \sum_{i=1}^{j} d_H(u_i, v_i).$$

Como todo k-mer em $k(s)$ é rótulo de algum vértice de G_k^E, se $j < m$ significa que $m - j$ k-mers $u_{j+1}, u_{j+2}, \ldots, u_m$ em $k(s)$ são rótulos de $m - j$ vértices $v_{j+1}, v_{j+2}, \ldots, j_m$ em G_k com $d_H(u_i, v_i) = 0$ para cada $i = j + 1, j + 2, \ldots, m$, o que implica que a substituição dos rótulos de $v_1, v_2, \ldots, v_m$ por $u_1, u_2, \ldots, u_m$ é uma edição E' de mesmo custo da edição E, ou seja,

$$c(E') = c(E) + \sum_{i=j+1}^{m} d_H(u_i, v_i) = c(E) + 0 = c(E).$$

Assumimos então, sem perda de generalidade, que toda edição possui exatamente m operações. O conjunto $\mathcal{E}$ é definido como o conjunto de todas as edições em G_k. Uma **edição ótima** é então uma edição de menor custo e denotamos o custo de uma edição ótima em G_k referente a s por $D_H(s, G_k) = \min_{E \in \mathcal{E}} c(E)$. para exemplificar essas definições, considere, por exemplo, a sequência $s =$ AACCAACCA e o grafo de De Bruijn $G_3 = \{$AAC, ACC, CAG, TTT$\}$. Nesse caso, temos $k(s)$=$\{$AAC, ACC, CCA e CAA$\}$ e a Figura 6.1 mostra exemplos de edição e edição ótima em G_3 para esse caso.



Grafo original | Edição E' | Edição ótima E

Figura 6.1: Exemplo de uma edição E' em G_3 referente a $s =$AACCAACCA que substitui os rótulos de v_2, v_3 e v_4. O custo de E' é $c(E') = 6$, refletindo o custo para substituir o rótulo ACC por CCA, CAG por CAA e o rótulo TTT por ACC. Além da edição E', temos um exemplo de uma edição ótima E em G_3 referente ao mesmo $s =$ AACCAACCA que substitui os rótulos de v_3 e de v_4. O custo de E é $c(E) = 4$, refletindo o custo para substituir o rótulo CAG por CAA e o rótulo TTT por CCA. Como resultado da edição E, temos o grafo de De Bruijn G_3^E. Note que o passeio $p = v_1, v_2, v_4, v_3, v_1, v_2, v_4$ em G_3^E induz a sequência s.

Com base nas definições anteriores, definimos o problema de mapear uma sequência em grafos de De Bruijn com mudanças no grafo da seguinte maneira:

Problema 10. MAPEAMENTO DE SEQUÊNCIAS EM GRAFOS DE DE BRUIJN COM MUDANÇAS NO GRAFO*: Dadas uma sequência s e um grafo de Bruijn G_k encontrar uma edição ótima em G_k referente a s.*

Para abordar esse problema e propor um algoritmo para resolvê-lo, considere as definições e o resultado a seguir.

6.2 Bipartição e emparelhamento

A seguir vamos descrever a criação de um grafo bipartido a partir de um $k(s)$ e de um grafo de De Bruijn G_k (representado apenas pelo seus vértices) e a relação entre um emparelhamento máximo de custo mínimo e a edição ótima em G_k referente a s. Para dois conjuntos A e B, denotamos por $A \times B$ o conjunto formado por todos os vértices de A e B de tal forma que $u \in A$ e $v \in B$, ou seja, $A \times B = \{\{u, v\} : u \in A \text{ e } v \in B\}$. Considere $H = (k(s) \cup G_k, k(s) \times G_k)$ um grafo bipartido completo onde cada vértice em G_k é adjacente a cada um dos k-mers de $k(s)$. O custo de uma aresta $\{u, v\}$ de H é dado por $d_H(u, v)$.

Um emparelhamento máximo ou simplesmente emparelhamento M em H possui m arestas e possui um custo

$$C(M) = \sum_{\{u,v\} \in M} d_H(u, v).$$

Note que, como $|k(s)| \leq |G_k|$, esse emparelhamento cobre todos os vértices de $k(s)$. Definimos aqui que um **emparelhamento ótimo** M^* é um emparelhamento máximo de custo mínimo.

Seja $M = \{\{u_1, v_1\}, \ldots, \{u_m, v_m\}\}$ um emparelhamento em $H = (k(s) \cup G_k, k(s) \times G_k)$. A edição E_M com m operações correspondente ao emparelhamento M é a edição onde cada aresta $\{u, v\}$ do emparelhamento ($u \in k(s)$ em $v \in G_k$) corresponde a uma operação de substituir o rótulo de v por u. Dizemos que M transforma G_k em $G_k^{E_M}$. Observe que pela definição de edição e emparelhamento

$$C(M) = \sum_{i=1}^{m} d_H(u_i, v_i) = c(E_M).$$

Similarmente, a operação de substituição de cada rótulo de v por u em uma edição E corresponde a uma aresta em um emparelhamento, denotado por M_E, em $H = (k(s) \cup G_k, k(s) \times G_k)$ e note novamente que, pela definição de edição e emparelhamento

$$c(E) = \sum_{i=1}^{m} d_H(u_i, v_i) = C(M_E).$$

Um exemplo desse grafo bipartido completo, emparelhamento e custo de uma edição é mostrado na Figura 6.2, referente ao grafo de De Bruijn $G_3 = \{\texttt{ACT}, \texttt{CTG}, \texttt{ACG}, \texttt{TTT}\}$ e $k(s) = \{\texttt{ACT}, \texttt{CTG}, \texttt{TGC}, \texttt{GCG}\}$ obtido da sequência $s =$AACCAACCA. Nesse exemplo, o custo de cada aresta $e = \{u, v\}$, onde $u \in k(s)$ e $v \in G_k$, é determinado por $c(e) = d_H(u, v)$.

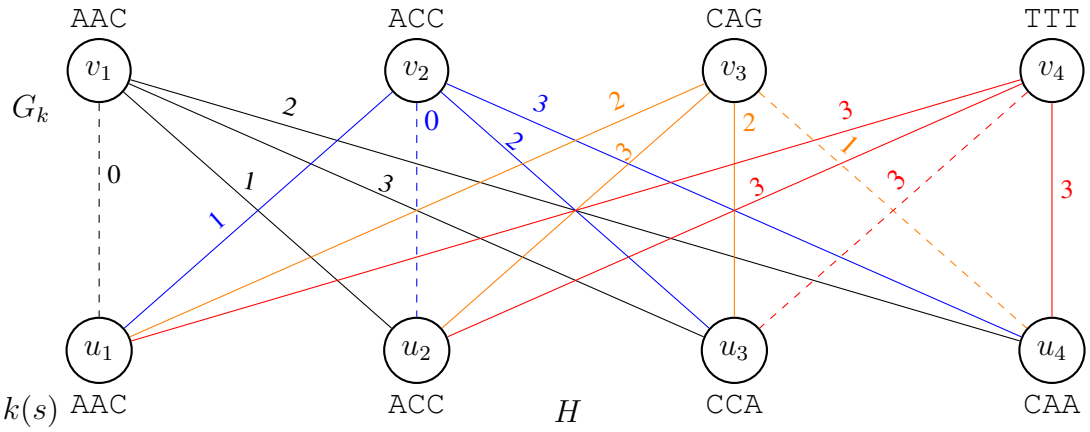


Figura 6.2: Exemplo de um grafo bipartido completo $H = (k(s) \cup G_k, k(s) \times G_k)$ para o grafo de De Bruijn $G_3 = \{\texttt{AAC}, \texttt{ACC}, \texttt{CAG}, \texttt{TTT}\}$ e $k(s) = \{\texttt{AAC}, \texttt{ACC}, \texttt{CCA}, \texttt{CAA}\}$, obtido da sequência s =AACCAACCA. Cada aresta $e = \{u, v\} \in A$ tem um custo $c(e) = d_H(u, v)$ associado. As arestas pontilhadas representam um emparelhamento ótimo $M^* = \{\{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_4\}, \{u_4, v_3\}\}$, com custo total 4. O custo da edição E_{M^*} é igual a 4, considerando a alteração do k-mer CAG do vértice v_3 para CAA e do k-mer TTT do vértice v_4 para CCA. As cores nas arestas servem para facilitar a visualização do grafo.

O teorema a seguir é usado para a descrição de um algoritmo que encontra uma edição ótima em G_k referente a s.

Teorema 4. *Seja s uma sequência, G_k um grafo de Bruijn. Se M^* é um emparelhamento ótimo em $H = (k(s) \cup G_k, k(s) \times G_k)$, então $C(M^*) = D_H(s, G_H)$ e E_{M^*} corresponde a uma edição ótima em G_k referente a s.*

Demonstração. A edição E_{M^*} (correspondente ao emparelhamento M^* em H) transforma G_k em $G_k^{E_{M^*}}$ e tem custo $c(E_{M^*}) = C(M^*)$ e, portanto, $D_H(s, G_k) \leq c(E_{M^*}) = C(M^*)$. Por outro lado, um emparelhamento M_{E^*} (correspondente a edição ótima E^* em G_k) em H tem custo $C(M_{E^*}) = D_H(s, G_k)$ o que implica que $C(M^*) \leq C(M_{E^*}) = D_H(s, G_k)$.

Portanto, $C(M^*) = c(E_{M^*}) = D_H(s, G_H)$ o que implica que $C(M^*) = D_H(s, G_k)$ e por consequência a edição E_{M^*} corresponde a uma edição ótima em G_k referente a s. □

O Teorema 4 garante que um algoritmo para encontrar um emparelhamento ótimo em H pode ser usado para encontrar uma edição ótima em G_k e é justamente isso que faz o algoritmo a seguir. Dado um grafo de De Bruijn G_k (representado exclusivamente pelo seu conjunto de vértices), e uma sequência s, o algoritmo proposto aqui é denominado De Bruijn sequence Mapping Tool with Graph Changes – BMTC e segue os seguintes passos. Inicialmente, é determinado o conjunto $k(s)$. Em seguida, o grafo bipartido completo $H = (k(s) \cup G_k, k(s) \times G_k)$ é criado, atribuindo um custo $d_H(u, v)$ para cada aresta $\{u, v\}$ com $u \in k(s)$ e $v \in G_k$, conforme especificado no Algoritmo 9 (Bipartido). O algoritmo Húngaro [Kuhn (1955)] é aplicado ao grafo H para buscar um emparelhamento máximo de custo mínimo M^*. Esses passos descritos para resolver o problema do mapeamento de sequências em grafos de De Bruijn com mudanças no grafo são detalhados no pseudocódigo mostrado no Algoritmo 8, e o algoritmo devolve o M^* e o custo da edição ótima.

Algoritmo 8: De Bruijn sequence Mapping Tool with graph Changes – BMTC

Entrada: Uma sequência s e os vértices de um grafo de De Bruijn G_k.
Saída: O emparelhamento ótimo M^* e o custo da edição ótima.

1 $k(s) \leftarrow k$-mers de s;
2 **se** $|G_k| \geq |k(s)|$ **então**
/* H é um grafo bipartido considerando G_k e k(s) */
3 $H \leftarrow$Bipartido$(k(s), G_k)$;
/* M^* é um emparelhamento ótimo obtido com o algoritmo Húngaro com a entrada H */
4 $M^* \leftarrow$Algoritmo Húngaro(H);
5 $custo \leftarrow 0$;
6 **para** *cada aresta* $\{u, v\} \in M^*$ **faça**
7 $custo \leftarrow custo + d(u, v)$;
8 **devolve** $M^*, custo$;
9 **devolve** *Não tem solução porque* $|G_k|$ *precisa ser maior ou igual a* $|k(s)|$;

Algoritmo 9: Bipartido

Entrada: dois conjuntos de vértices (k-mers) $k(s)$ e G_k.
Saída: um grafo bipartido $H = (k(s) \cup G_k, k(s) \times G_k)$ com custo nas arestas.

1 $A \leftarrow \{\}$;
2 **para** *cada* $u \in k(s)$ **faça**
3 **para** *cada* $v \in G_k$ **faça**
4 $c \leftarrow d_H(u, v)$; // custo da aresta
5 $A \leftarrow A \cup \{u, v\}$ onde $\{u, v\}$ tem custo c;
6 **devolve** $H = (k(s) \cup G_k, A)$;

Para determinar a complexidade de tempo do Algoritmo 8 considere $m = |k(s)| \leq |G_k| = n$ e $|s| = l$. Assim, a complexidade de tempo para identificar todos os k-mers na linha 1 é $O(l \cdot k)$. Na linha 3, a criação do grafo bipartido e o cálculo do custo das arestas têm uma complexidade de tempo $O(n \cdot m \cdot k)$ pois para cada um dos n vértices v de G_k, calculamos para cada um dos m k-mers u de $k(s)$ o custo $d_H(u, v)$ consumindo tempo $O(k)$. A execução do algoritmo Húngaro na linha 4 tem uma complexidade de tempo $O(n + m)^3$. Finalmente, percorrer todas as arestas do emparelhamento e calcular o custo da edição nas linhas 6–7 é $O(m)$. Portanto, a complexidade de tempo total é dada por $O(l \cdot k) + O(n \cdot m \cdot k) + O((n + m)^3) + O(m) = O((n + m)^3 + (n \cdot m \cdot k) + m + (l \cdot k))$ = $O(n^3 + l)$ pois k é menor que o número de vértices no grafo e $m \leq n$.

No Capítulo 7, realizamos teste para validar a ideia proposta neste capítulo. Devido à inexistência de um algoritmo na literatura para compararmos com a nossa proposta, apresentamos uma estratégia de teste para validar o algoritmo aqui desenvolvido.

CAPÍTULO

7

Experimentos, Resultados e Análise

Neste capítulo, descreveremos os testes conduzidos para avaliar as implementações realizadas neste trabalho. Na Seção 7.1, apresentamos os testes relacionados às implementações que realizam o mapeamento no grafo quando apenas mudanças na sequência são permitidas. Para avaliar as implementações da Seção 7.1, detalhamos a acurácia e o desempenho na Seção 7.1.1. Na Seção 7.1.2, realizamos uma comparação entre a acurácia e o desempenho de execução das heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$. Estes testes abordam exclusivamente a heurística 1 baseada no algoritmo `BSMT`. Utilizamos o algoritmo $\texttt{BSMT}_d$ como parâmetro de comparação para obter a distância exata do mapeamento, visto que o `BSMT` não consegue processar grandes quantidades de k-mers.

Na Seção 7.1.2, comparamos a acurácia e o desempenho de execução da solução exata $\texttt{BSMT}_d$ com as heurísticas $\texttt{BSMT}_{h1_d}$ e $\texttt{BSMT}_{h1_{db}}$, focando esses testes na avaliação das duas heurísticas baseadas no $\texttt{BSMT}_d$. Na Seção 7.1.3, realizamos testes abrangentes para comparar todas as quatro heurísticas para realizar o mapeamento da sequência no grafo desenvolvidas neste trabalho em termos de acurácia e desempenho de execução. Mais uma vez, utilizamos o $\texttt{BSMT}_d$ para obter a distância exata. Na Seção 7.1.4, comparamos a acurácia e o desempenho de execução das heurísticas desenvolvidas neste trabalho com a ferramenta `BGREAT`, desenvolvida por Limasset *et al.* [Limasset et al. (2016)], e com a ferramenta `BrownieAligner`, desenvolvida por Heydary *et al.* [Heydari et al. (2018)]. O objetivo é avaliar o desempenho das nossas heurísticas em relação às existentes na literatura. Finalmente, concluímos o capítulo na Seção 7.2, apresentando os resultados relacionados à nossa aplicação `BMTC` proposta para realizar o mapeamento quando apenas mudanças no grafo são permitidas. Apesar de não existirem aplicações para esse propósito na literatura, conduzimos testes para avaliar o desempenho do algoritmo e apresentar, em testes práticos, os custos devolvidos pelo `BMTC` para modificar o grafo. Todos os experimentos foram conduzidos em um computador equipado com processador Intel(R) Xeon(R) CPU E5-2620 2.00GHz, contendo 24 CPUs e 62 GB de RAM.

7.1 Mudanças na sequência

Esta seção descreve uma série de testes relacionados às aplicações que realizam o mapeamento com alterações exclusivamente na sequência. Esses testes abrangem tanto os algoritmos para o mapeamento quanto as heurísticas implementadas neste trabalho, bem como as heurísticas propostas por outros autores.

7.1.1 Conjunto de testes, acurácia e desempenho

Para avaliar nossas abordagens, conduzimos uma análise da acurácia e do desempenho das heurísticas propostas por meio de uma série de experimentos em um conjunto de sequências. Nosso conjunto de testes abrange mais de 20 grafos de De Bruijn, construídos a partir de sequências curtas (com comprimento médio de 100) de Escherichia coli (conjunto A) e Escherichia fergusonii (conjunto B). A quantidade de k-mers no grafo atinge até 557522 k-mers para os testes que envolvem apenas a busca da distância do mapeamento e até 86214 k-mers no grafo para os testes que envolvem a busca do mapeamento no grafo. Adicionalmente, para realizar o mapeamento no grafo, incluímos mais de 100 sequências longas de comprimentos (comprimento maior do que 1000) diversos de Escherichia coli (conjunto C) e mais de 100 sequências curtas (comprimento médio de 100) de Escherichia coli (conjunto A). Todas essas sequências foram obtidas no banco de dados GenBank [Benson et al. (2012)] do NCBI.

No contexto exclusivo das heurísticas e do algoritmo que busca apenas a distância do mapeamento, calculamos a média das distâncias para cada abordagem. Para cada conjunto de teste, somamos todas as distâncias obtidas pelas heurísticas $\texttt{BSMT}_{h1_d}$ e $\texttt{BSMT}_{h1_{db}}$, assim como pelo algoritmo $\texttt{BSMT}_d$. Em seguida, dividimos essa soma pelo número total de testes realizados, resultando na média das distâncias relacionadas a um conjunto de testes. Para avaliar a acurácia das heurísticas, dividimos a média de cada heurística pelo valor da média obtida pelo $\texttt{BSMT}_d$.

Agora, no contexto das heurísticas que executam o mapeamento de uma sequência em um grafo (ou seja, nos testes que envolvem as heurísticas $\texttt{BSMT}_{h1}$, $\texttt{BSMT}_{h1_b}$, $\texttt{BSMT}_{h2}$, $\texttt{BSMT}_{h3}$, `BGREAT` e o `BrownieAligner`), utilizamos as sequências geradas por essas heurísticas. Calculamos a distância de edição entre cada uma dessas sequências e a sequência original mapeada, somando essas distâncias e dividindo pelo número total de testes. Para avaliar a proximidade desse valor da distância de edição entre a sequência obtida pelo mapeamento e a sequência original mapeada em relação ao custo real de mapeamento, utilizamos o $\texttt{BSMT}_d$ para obter a distância exata do mapeamento. Somamos essas distâncias e dividimos pelo número total de testes realizados. Com as médias obtidas, dividimos as médias das heurísticas pela média das distâncias obtidas pelo $\texttt{BSMT}_d$, obtendo assim a acurácia.

Para interpretar o significado da acurácia nos testes a seguir, é relevante observar que valores próximos de 1 indicam que as heurísticas $\texttt{BSMT}_{h1}$, $\texttt{BSMT}_{h1_b}$, $\texttt{BSMT}_{h2}$ ou $\texttt{BSMT}_{h3}$ obtiveram resultados satisfatórios. Isso implica que o mapeamento realizado no grafo é considerado eficaz para essas heurísticas. No caso das heurísticas $\texttt{BSMT}_{h1_d}$ ou $\texttt{BSMT}_{h1_{db}}$, que fornecem apenas a distância do mapeamento, uma acurácia próxima de 1 significa que a distância calculada por

essas heurísticas é uma boa estimativa e se aproxima da distância real do mapeamento.

Falando exclusivamente do desempenho das implementações, o cálculo empregado é a média das durações de cada execução nos conjuntos de testes. Em outras palavras, para um determinado conjunto de testes, somaremos todas as durações de execução e dividiremos pela quantidade total de testes realizados. O resultado é sempre expresso em segundos.

7.1.2 Testes com a heurística 1

Estes testes abrangem exclusivamente a heurística 1, e serão conduzidos em duas etapas distintas. O primeiro teste envolve a execução das heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$ para realizar o mapeamento de diversas sequências nos grafos de De Bruijn. Em seguida, os testes subsequentes incluem a execução das heurísticas $\texttt{BSMT}_{h1_d}$ e $\texttt{BSMT}_{h1_{db}}$, além do algoritmo $\texttt{BSMT}_d$, com o objetivo de obter apenas a distância de mapear sequências nos grafos de De Bruijn.

Testes para obter mapeamento

Para executar os testes desta seção, utilizamos sequências cujos comprimentos variam entre 1000 e 10000 do conjunto C e grafos de De Bruijn construídos a partir de sequências do conjunto A. As heurísticas devolvem a sequência induzida por um bom passeio encontrado, e calculamos a distância de edição entre a sequência original e a sequência induzida, comparando-a com a distância exata devolvida pelo $\texttt{BSMT}_d$. Nestes testes, o número de sementes varia entre 2 e 9000. As distâncias médias calculadas são mostradas na Tabela 7.1, e os tempos médios de execução de cada heurística são apresentados na Tabela 7.2, com os respectivos gráficos para cada tabela sendo exibidos nas Figuras 7.1 e 7.2. Para cada linha dessas tabelas foram realizados 100 testes. Nesses testes, utilizamos diversas sequências com comprimento entre 1000 e 10000, e o valor de k sendo 5, 10 e 20. Conseguimos usar o BSMT em apenas 3 dos 100 testes, o que comprova a aplicabilidade de nossas abordagens. Podemos observar nas tabelas destes testes que as heurísticas são capazes de realizar o mapeamento em relação ao BSMT, e que a distância obtida pelas heurísticas corresponde a no máximo três vezes a distância obtida pelo $\texttt{BSMT}_d$. Esse resultado é consideravelmente bom com espaço para melhoria, como por exemplo considerar mais k-mers do grafo de De Bruijn para preencher os *gaps* ou ter um grafo de De Bruijn com mais k-mers para resultar em mais sementes na sequência mapeada.

Realizamos mais testes com grafos de De Bruijn construídos a partir das sequências do conjunto B, e selecionamos sequências com comprimento variando entre 4221 e 6991 do conjunto C. Nestes testes, o número de sementes varia entre 0 e 2015 e o valor de k é 10 e 15. As distâncias médias e os tempos de execução obtidos nesta rodada de testes são mostrados na Tabela 7.3 e na Tabela 7.4, respectivamente. Para cada linha dessas tabelas foram realizados 100 testes. Os gráficos das distâncias médias calculadas e do tempo de execução estão representados nas Figuras 7.3 e 7.4, respectivamente. Diferentemente das nossas heurísticas, o BSMT não foi capaz de lidar com a grande maioria dos casos de teste. Em nossas análises, o BSMT conseguiu processar casos de teste pequenos para sequências com comprimento próximo de 1000 e um grafo de De Bruijn com aproximadamente 2000 k-mers.

		Média das distâncias			Acurácia	
k	k-mers $\in G_k$	$\texttt{BSMT}_d$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$
5	1017	8,25	9,99	12,99	1,21	1,57
10	11375	2363,48	5149,13	5167,46	2,18	2,19
20	10243	5278,52	5381,95	5388,52	1,02	1,02

Tabela 7.1: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 1000 e 10000 e a média de comprimento de 5393.

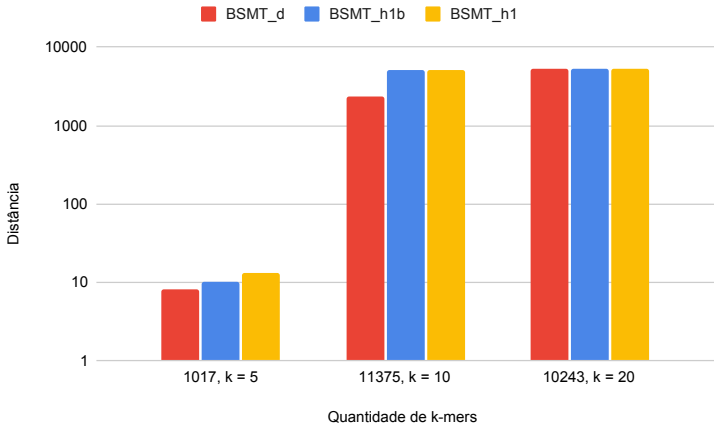


Figura 7.1: Média das distâncias: gráfico para a Tabela 7.1.

k	k-mers $\in G_k$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$
5	1017	4,48	5,60
10	11375	198,05	242,43
20	10243	181,45	226,61

Tabela 7.2: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 1000 e 10000 e a média de comprimento de 5393.

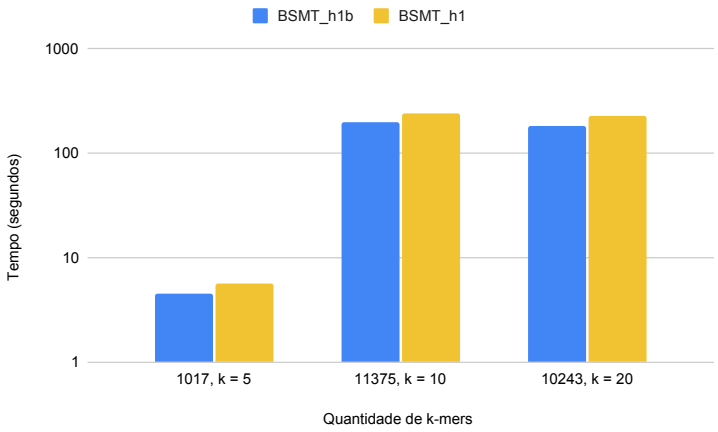


Figura 7.2: Média de tempo (segundos): gráfico para a Tabela 7.2.

		Média das distâncias			Acurácia	
k	k-mers em G_k	$\texttt{BSMT}_{d}$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$
10	78661	1746,38	3046,88	4716,38	1,7	2,7
15	86214	2416,25	5197,50	5307,00	2,1	2,2

Tabela 7.3: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 4221 e 6991 e a média do comprimento de 5528.

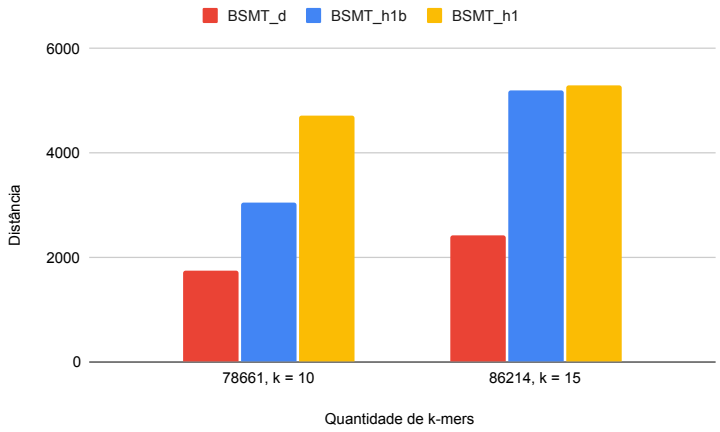


Figura 7.3: Média das distâncias: gráfico para a Tabela 7.3.

k	k-mers em G_k	$\texttt{BSMT}_{h1}$	$\texttt{BSMT}_{h1_b}$
10	78661	2250,00	1125,00
15	86214	3160,25	1365,00

Tabela 7.4: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 4221 e 6991 e a média do comprimento de 5528.

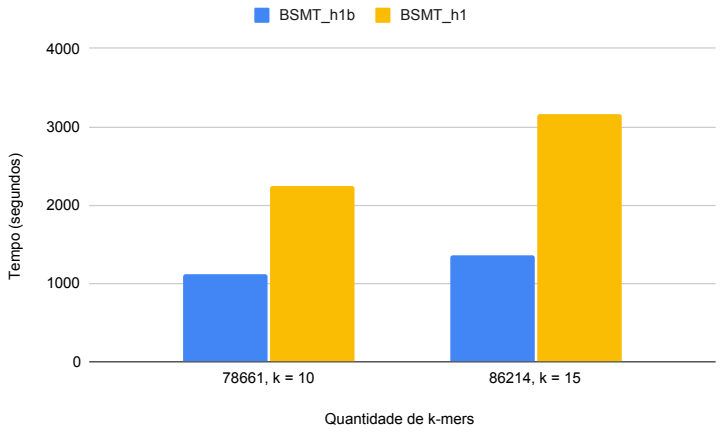


Figura 7.4: Média de tempo (segundos): gráfico para a Tabela 7.4.

Testes para obter somente a distância

Geralmente, os testes envolvem a busca pelo mapeamento conforme o teste anterior. Entretanto, existe o cenário em que precisamos conhecer apenas a distância do mapeamento (e não o mapeamento em si). Testar o algoritmo $\texttt{BSMT}_d$ e as heurísticas propostas para ele é de grande relevância para o nosso trabalho. Este teste é crucial, uma vez que a aplicação $\texttt{BSMT}_d$ é capaz de processar grafos de De Bruijn com grandes quantidades de k-mers, mas consome muito tempo de processamento. Os testes desta seção visam avaliar o quanto de redução no tempo com nossas heurísticas é possível de alcançar.

Para executar estes testes, utilizamos grafos de De Bruijn com uma quantidade variável de k-mers construídos a partir do conjunto A, e selecionamos sequências com comprimento variando entre 1271 e 9239 do conjunto C. Para $k = 5$, as médias das distâncias calculadas são apresentadas na Tabela 7.5, e os tempos médios de execução de cada heurística estão na Tabela 7.6, com os respectivos gráficos para cada tabela sendo exibidos nas Figuras 7.5 e 7.6.

Outros testes foram realizados aumentando o valor de k para 10, e as médias das distâncias calculadas são mostradas na Tabela 7.7, enquanto os tempos médios de execução de cada heurística estão na Tabela 7.8. Os respectivos gráficos para cada tabela são exibidos nas Figuras 7.7 e 7.8. Para cada linha dessas tabelas, foram realizados 100 testes a fim de observar o comportamento das heurísticas.

Observamos que, para a versão que busca apenas a distância, é possível obter um grande ganho de tempo com as heurísticas em comparação com o $\texttt{BSMT}_d$, pois aproveitamos as sementes para não processar a sequência inteira. Por outro lado, há uma perda de precisão com as heurísticas. O número de sementes nestes testes aumenta à medida que o número de k-mers no grafo aumenta, variando entre 14 e 9206 sementes.

Para avaliar ainda mais nossas heurísticas, conduzimos testes adicionais aumentando a quantidade de k-mers presentes no grafo. A quantidade de k-mers variou entre 4577 e 557522, enquanto o comprimento das sequências mapeadas variou entre 1493 e 6225, com uma média de comprimento igual a 3189. Para $k = 10$, as médias das distâncias calculadas estão apresentadas na Tabela 7.9, e os tempos médios de execução de cada heurística são indicados na Tabela 7.10, com os respectivos gráficos para cada tabela sendo exibidos nas Figuras 7.9 e 7.10. Devido ao alto tempo de execução do $\texttt{BSMT}_d$, que pode levar horas em testes com muitos k-mers, realizamos apenas 4 testes para cada linha dessas últimas tabelas. Isso foi feito com o objetivo de observar o comportamento das heurísticas ao aumentarmos a quantidade de k-mers no grafo e determinar o ganho de tempo conforme a quantidade de sementes aumenta em nossas heurísticas.

Novamente, observamos que, para esta versão que busca apenas a distância, é possível obter uma significativa redução no tempo (em torno de 70% de redução) com as heurísticas em comparação com o $\texttt{BSMT}_d$. Em nossos testes, conseguimos reduzir o tempo de dias para horas. O número de sementes nestes testes aumenta à medida que o número de k-mers no grafo cresce, variando entre 11 e 6080 sementes.

	Média das distâncias			Acurácia	
k-mers em Gk	$\texttt{BSMT}_{d}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$
977	94,93	226,59	225,59	2,39	2,38
1014	15,22	39,39	35,47	2,59	2,33
1017	9,85	28,06	27,10	2,85	2,75
1021	8,25	24,89	22,93	3,02	2,78

Tabela 7.5: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 1271 e 9239 e com o comprimento médio de 5393 e $k = 5$.

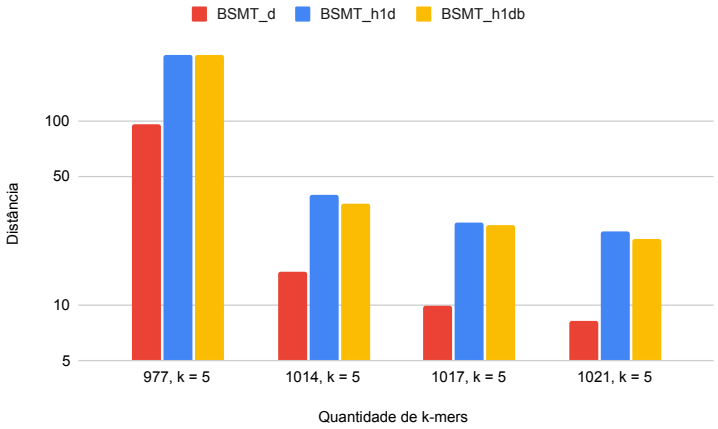


Figura 7.5: Média das distâncias: gráfico para a Tabela 7.5.

k-mers em G_k	$\texttt{BSMT}_{d}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$
977	14,25	9,03	6,53
1014	16,19	13,18	9,31
1017	16,28	11,36	7,19
1021	16,30	11,36	4,18

Tabela 7.6: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 1271 e 9239 e com o comprimento médio de 5393 e $k = 5$.

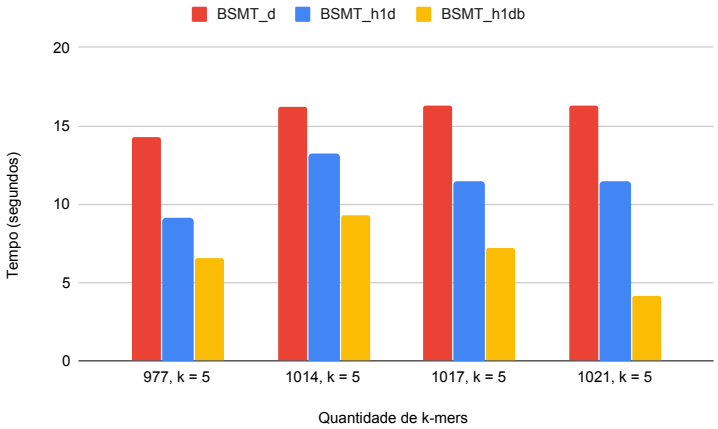


Figura 7.6: Média de tempo (segundos): gráfico para a Tabela 7.6.

	Média das distâncias			Acurácia	
k-mers em Gk	$\texttt{BSMT}_d$	$\texttt{BSMT}_{h1_d}$	$\texttt{BSMT}_{h1_{db}}$	$\texttt{BSMT}_{h1_d}$	$\texttt{BSMT}_{h1_{db}}$
4577	2596,40	5366,60	5319,54	2,07	2,05
9119	2438,16	5349,90	5246,14	2,19	2,15
11375	2410,55	5342,46	5203,89	2,22	2,16
22485	2363,48	5334,36	5173,17	2,26	2,19

Tabela 7.7: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 1271 e 9239 e com o comprimento médio de 5393 e $k = 10$.

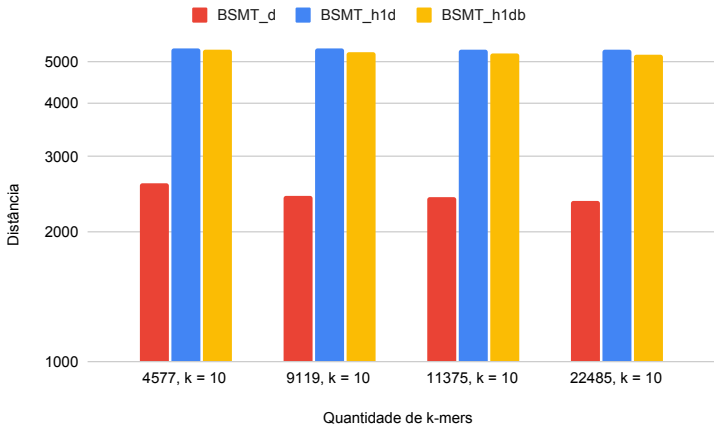


Figura 7.7: Média das distâncias: gráfico para a Tabela 7.7.

k-mers em G_k	$\texttt{BSMT}_d$	$\texttt{BSMT}_{h1_d}$	$\texttt{BSMT}_{h1_{db}}$
4577	5254,29	1679,14	1559,29
9119	5230,15	1644,95	1540,02
11375	5244,07	1650,23	1536,86
22485	5261,20	1679,95	1525,63

Tabela 7.8: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 1271 e 9239 e com o comprimento médio de 5393 e $k = 10$.

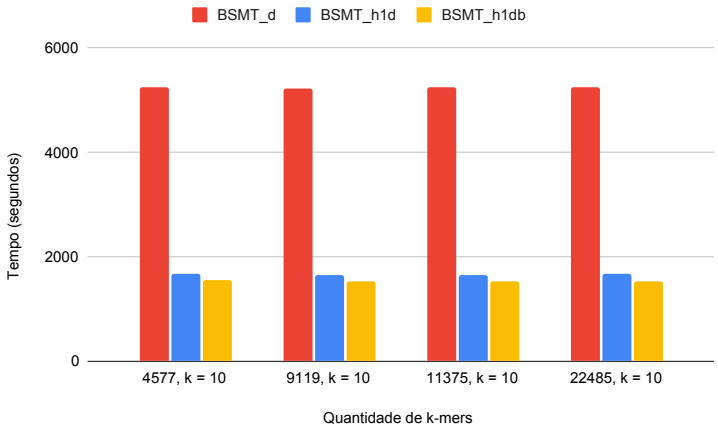


Figura 7.8: Média de tempo (segundos): gráfico para a Tabela 7.8.

	Média das distâncias			Acurácia	
k-mers em Gk	$\texttt{BSMT}_{d}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$
4577	1537,50	3181,00	3147,00	2,05	2,07
9119	1453,25	3184,25	3098,25	2,19	2,13
11375	1427,00	3142,25	3028,00	2,20	2,12
22485	1201,75	3111,25	2835,00	2,59	2,36
43842	995,25	2850,50	2558,00	2,86	2,57
84683	662,75	1663,25	1550,00	2,51	2,34
557522	314,50	552,25	477,25	1,76	1,52

Tabela 7.9: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 1493 e 6225 e com o comprimento médio de 3189 e $k = 10$.

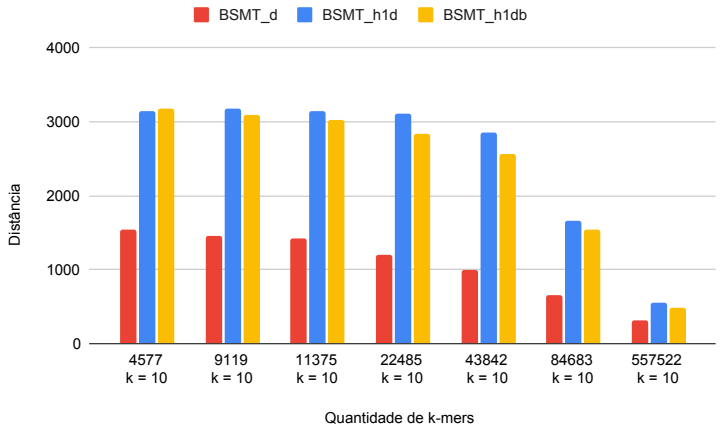


Figura 7.9: Média das distâncias: gráfico para a Tabela 7.9.

k-mers em G_k	$\texttt{BSMT}_{d}$	$\texttt{BSMT}_{h1_{d}}$	$\texttt{BSMT}_{h1_{db}}$
4577	5457,75	47,00	42,25
9119	5924,00	65,00	62,50
11375	6608,50	105,00	121,00
22485	7014,75	314,75	328,25
43842	15002,00	533,75	546,00
84683	89152,00	344,25	342,00
557522	151787,25	101,75	113,50

Tabela 7.10: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 1493 e 6225 e com o comprimento médio de 3189 e $k = 10$.

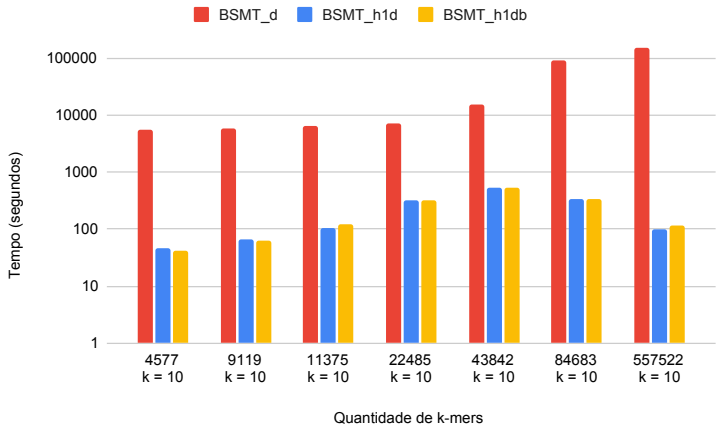


Figura 7.10: Média de tempo (segundos): gráfico para a Tabela 7.10.

7.1.3 Testes com todas as heurísticas

Para executar os testes desta seção, utilizamos sequências cujos comprimento variam entre 1000 e 10000 do conjunto C e grafos de De Bruijn construídos a partir de sequências do conjunto A. As distâncias médias calculadas são mostradas na Tabela 7.11, e os tempos médios de execução de cada heurística são apresentados na Tabela 7.12, com os respectivos gráficos para cada tabela sendo exibidos nas Figuras 7.11 e 7.12. Para cada linha dessas tabelas foram realizados 100 testes. Nesses testes, obtivemos diversas sequências com comprimento variando de 1000 a 10000, e o valor de k variou entre 5, 10 e 20. Podemos verificar uma competitividade interessante em termos de tempo e qualidade. As heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$ apresentaram valores que correspondem a no máximo três vezes a distância do mapeamento.

Aqui, o número de sementes varia de 0 a 9234. Para o valor de $k = 5$, observamos o maior número de sementes, indicando a presença de muitos k-mers da sequência no grafo de De Bruijn. Isso resulta em um mapeamento preciso da sequência. Quando $k = 10$, o número de sementes representa aproximadamente 50% do comprimento da sequência, resultando em um mapeamento parcial. Vale ressaltar que, embora a acurácia próxima de 1 seja indicativa de bons resultados, para $k = 20$, enfrentamos um cenário especial com poucas sementes, resultando em falsos positivos. Nesse contexto, uma acurácia próxima de 1 não necessariamente reflete um mapeamento eficaz, dado que a sequência pode ser induzida por um passeio consideravelmente pequeno. É apenas nesse cenário que surgem os falsos positivos, e pode ser necessário reavaliar o resultado considerando a adição de uma unidade à distância entre a sequência de entrada e a sequência induzida para cada caractere não mapeado da sequência de entrada. Esse cenário reforça a importância de ter uma sequência com muitas sementes, representando pelo menos 50% da sequência, para melhorar significativamente a acurácia das heurísticas e eliminar os falsos positivos.

Resumidamente, temos dois cenários nos testes com $k = 20$: em primeiro, a aplicação $\texttt{BSMT}_d$ consistentemente devolveu uma alta distância de mapeamento, indicando que, nesses testes, a tarefa de mapear uma sequência em um grafo com k-mers de comprimento 20 é desafiadora. Em segundo lugar, quando há poucas sementes no grafo, nossas heurísticas produzem um passeio p que gera uma sequência curta. A alta distância de edição entre p e a sequência mapeada, próxima à distância devolvida pelo $\texttt{BSMT}_d$, resulta em uma acurácia próxima de 1. No entanto, é crucial destacar que, nesse cenário, isso não implica necessariamente em um mapeamento eficaz das sequências no grafo.

Realizamos testes adicionais com sequências cujos comprimentos variam de 4221 a 6991 do conjunto C, utilizando grafos de De Bruijn construídos a partir das sequências do conjunto B. As distâncias médias calculadas são apresentadas nas Tabelas 7.13 e 7.14, enquanto os gráficos correspondentes podem ser visualizados nas Figuras 7.13 e 7.14. Observa-se que, para esses testes, o $\texttt{BSMT}_{h1_b}$ destacou-se como a melhor heurística em termos de tempo e qualidade. As heurísticas $\texttt{BSMT}_{h2}$ e $\texttt{BSMT}_{h3}$, embora sejam mais simples, conseguem alcançar valores próximos às demais heurísticas, porém, o tempo de execução é maior em comparação com as heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$.

k	k-mers em G_k	BSMT$_d$	Média das distâncias			
			BSMT$_{h1_b}$	BSMT$_{h1}$	BSMT$_{h2}$	BSMT$_{h3}$
5	1017	8,25	9,99	12,99	54,76	26,86
10	11375	2363,48	5149,13	5167,46	5188,08	5173,57
20	10243	5278,52	5381,95	5388,52	5382,62	5382,66
			Acurácia			
			1,21	1,57	6,64	3,26
			2,18	2,19	2,20	2,19
			1,02	1,02	1,02	1,02

Tabela 7.11: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 1000 e 10000 e a média do comprimento de 5393.

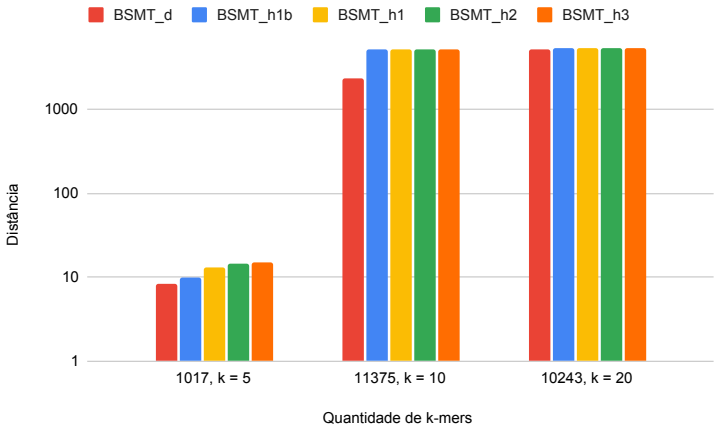


Figura 7.11: Média das distâncias: gráfico para a Tabela 7.11.

k	k-mers em G_k	BSMT$_{h1_b}$	BSMT$_{h1}$	BSMT$_{h2}$	BSMT$_{h3}$
5	1017	5,60	6,10	6,09	6,39
10	11375	198,05	242,43	263,51	276,69
20	10243	226,18	226,61	246,31	258,63

Tabela 7.12: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 1000 e 10000 e a média do comprimento de 5393.

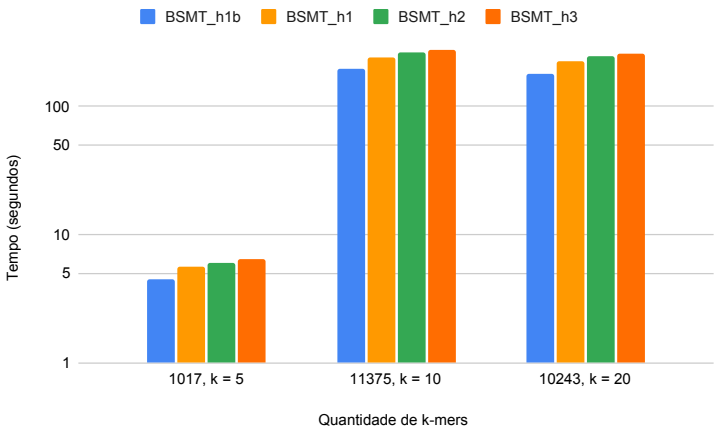


Figura 7.12: Média de tempo (segundos): gráfico para a Tabela 7.12.

			Média das distâncias			
k	k-mers em G_k	$\texttt{BSMT}_d$	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$	$\texttt{BSMT}_{h2}$	$\texttt{BSMT}_{h3}$
10	78661	1746,38	3046,88	4716,38	4721,13	4727,00
15	86214	2416,25	5197,50	5307,00	5299,00	5290,88
			Acurácia			
			1,74	2,70	2,70	2,71
			2,15	2,20	2,19	2,19

Tabela 7.13: Média das distâncias e acurácia: testes com o comprimento das sequências mapeadas entre 4221 e 6991 e a média do comprimento de 5528.

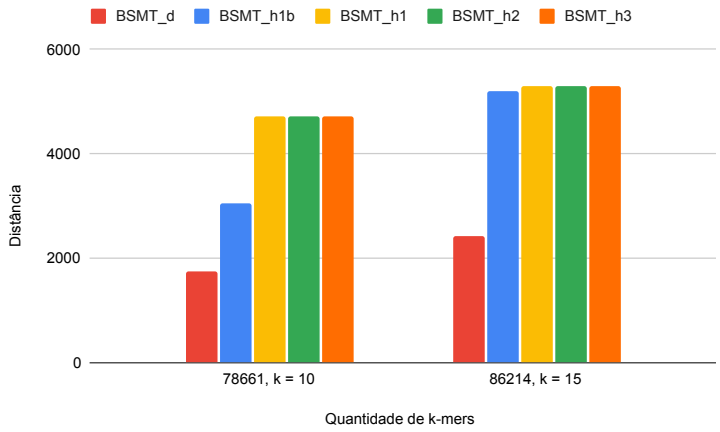


Figura 7.13: Média das distâncias: gráfico para a Tabela 7.13.

Um aspecto relevante que se destaca nos resultados desses testes é a influência da quantidade de sementes na execução eficaz das heurísticas. A quantidade de sementes é crucial, pois quanto maior esse número, mais k-mers da sequência estão presentes no grafo, resultando em uma menor quantidade de *gaps* a serem preenchidos pelas heurísticas. Em resumo, uma sequência com muitas sementes indica a presença de muitos k-mers da sequência no grafo, sugerindo que as heurísticas podem proporcionar resultados mais precisos nesse contexto.

7.1.4 Testes para comparar com outras heurísticas

Nesta seção, o objetivo é utilizar a ferramenta *de Bruijn Graph REAd mapping Tool* – `BGREAT` [Limasset et al. (2016)] e a ferramenta `BrownieAligner` [Heydari et al. (2018)], para realizar comparações com as heurísticas desenvolvidas neste trabalho. Em ambos os artigos, tanto o `BGREAT` quanto o `BrownieAligner` executam o mapeamento de uma sequência em um grafo de De Bruijn. Nos testes propostos, isso essencialmente envolve a construção de um grafo de De Bruijn a partir de um conjunto de sequências e, a partir desse grafo, mapear sequências que compartilham a mesma origem. Para esses testes, utilizamos como base as sequências da Escherichia coli. Construímos o grafo de De Bruijn considerando 229122 k-mers e $k = 10$. As sequências mapeadas possuem um comprimento médio de 100. Os resultados estão apresentados na Tabela 7.15, com os gráficos correspondentes exibidos na Figura 7.15 para a primeira linha da tabela e na Figura 7.16 para a última linha.

k	k-mers em G_k	$\texttt{BSMT}_{h1_b}$	$\texttt{BSMT}_{h1}$	$\texttt{BSMT}_{h2}$	$\texttt{BSMT}_{h3}$
10	78661	1125,00	2250,00	1508,38	2160,63
15	86214	1365,00	3160,25	1882,55	4560,00

Tabela 7.14: Média de tempo (segundos): testes com o comprimento das sequências mapeadas entre 4221 e 6991 e a média do comprimento de 5528.

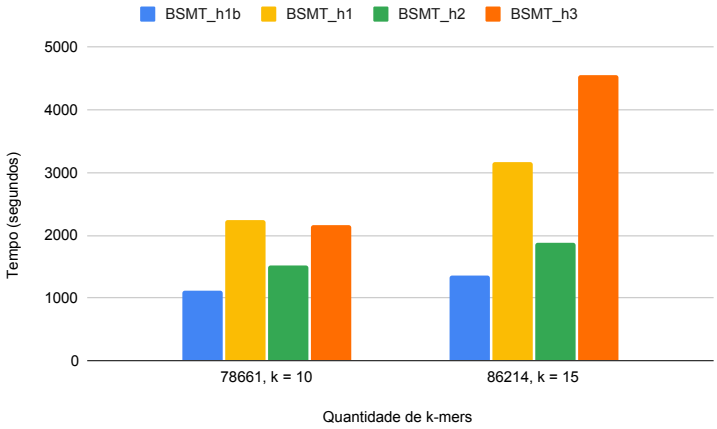


Figura 7.14: Média de tempo (segundos): gráfico para a Tabela 7.14.

Realizamos 100 testes para avaliar o desempenho de nossas heurísticas em comparação com o `BGREAT` e o `BrownieAligner`. Para validar as respostas tanto das ferramentas quanto das heurísticas, utilizamos a versão $\texttt{BSMT}_d$ para determinar a distância de mapeamento e avaliar o quão distantes as respostas estão da distância exata do mapeamento. O número de sementes nestes testes variou entre 20 e 60. Observamos que, nestes testes, nossas heurísticas apresentam uma qualidade de resposta levemente superior ao `BGREAT` e ao `BrownieAligner`, tanto nas heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$, que utilizam o algoritmo `RaMa`, quanto na heurística $\texttt{BSMT}_{h3}$, que emprega a técnica *seed-and-extend* para realizar o mapeamento. Este resultado sugere que nossas heurísticas estão bem direcionadas quanto à qualidade de resposta.

No entanto, tanto o `BGREAT` quanto o `BrownieAligner` executam em menos tempo do que todas as heurísticas que desenvolvemos. Isso aponta para uma oportunidade de aprimoramento no aspecto de tempo de execução de nossas heurísticas. Esses testes proporcionam uma janela de melhorias, permitindo a exploração de possibilidades de otimização nas implementações, visando alcançar resultados próximos aos obtidos pelo `BGREAT` e `BrownieAligner` em termos de tempo de execução. Além disso, esses testes validam nossas heurísticas em comparação com ferramentas existentes na literatura, confirmando a eficácia das ideias propostas para nossas heurísticas no contexto de mapeamento de sequências em um grafo de De Bruijn com alterações na sequência.

7.2 Mudanças no grafo

Esta seção descreve os testes associados à aplicação que realiza o mapeamento com alterações exclusivas no grafo. Este teste engloba o algoritmo que leva em consideração que o grafo de De

	Média das distâncias					
BSMT$_d$	BrownieAligner	BGREAT	BSMT$_{h1}$	BSMT$_{h1_b}$	BSMT$_{h2}$	BSMT$_{h3}$
31,26	65,06	55,61	53,68	51,14	58,39	54,79
	Acurácia					
	2,08	1,78	1,72	1,64	1,87	1,75

Média dos tempos					
BrownieAligner	BGREAT	BSMT$_{h1}$	BSMT$_{h1_b}$	BSMT$_{h2}$	BSMT$_{h3}$
6,02	5,16	74,63	71,13	95,36	88,74

Tabela 7.15: Média das distâncias, acurácia e tempo médio (em segundos): testes realizados com sequências mapeadas de comprimento médio de 100 e $k = 10$.

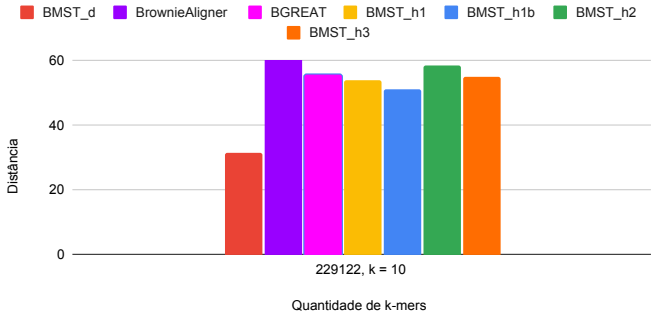


Figura 7.15: Média das distâncias: gráfico para a primeira linha da Tabela 7.15.

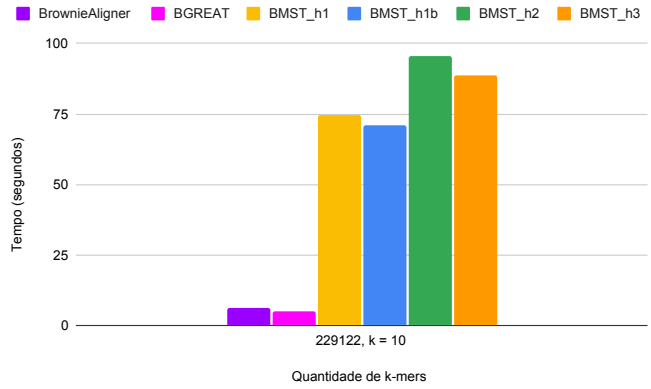


Figura 7.16: Média de tempo (segundos): gráfico para a última linha da Tabela 7.15.

Bruijn, após ter algum k-mer alterado, induz novos arcos.

7.2.1 Testes para validar as mudanças no grafo

Nesta seção, o objetivo é utilizar a ferramenta desenvolvida denominada BMTC e avaliar o custo das modificações feitas no grafo, assim como o tempo de execução da ferramenta. Conforme discutido no Capítulo 6, nossa proposta do BMTC considera que as modificações no grafo induzem novos arcos e é tratada pela primeira vez neste trabalho, portanto, não existe na literatura uma ferramenta similar para comparação. Para testar nosso algoritmo, criamos um cenário de teste capaz de validar se o custo determinado pelo emparelhamento máximo de custo mínimo é um custo que faz sentido ser considerado para modificar o grafo, e o cenário é descrito a seguir:

- Consideramos todos os k-mers da sequência s a ser mapeada no grafo;
- Construímos um grafo de De Bruijn com esses k-mers;
- Alteramos $\frac{\text{comprimento de } s}{k}$ k-mers no grafo (observe que este passo pode diminuir os k-mers do grafo, dado que uma modificação pode gerar um k-mer já existente);
- Executamos o `BMTC`.

Dado os passos anteriores, temos o controle sobre quantos k-mers foram alterados no grafo, considerando que uma única modificação foi feita em cada k-mer no grafo, resultando em $\frac{\text{comprimento de } s}{k}$ modificações realizadas no grafo. Ao executar o `BMTC`, obtemos o custo do emparelhamento, e conforme esperado, o custo é no máximo $\frac{\text{comprimento de } s}{k}$. Nestes testes, executamos o `BMTC` com valores de k iguais a 10, 20 e 30. Os resultados são exibidos na Tabela 7.16. Para cada valor de k, foram conduzidos 100 testes, e os valores de cada linha na tabela representam a média de cada teste. Para esses testes, utilizamos sequências de comprimento médio de 5566 para mapear no grafo e grafos de De Bruijn com k-mers entre 1464 e 11033.

Para $k = 10$, a quantidade de k-mers no grafo varia entre 1482 e 10933, com uma média de 5519 k-mers. Para $k = 20$, a quantidade de k-mers no grafo varia entre 1474 e 11043, com uma média de 5547 k-mers. Já para $k = 30$, a quantidade de k-mers no grafo varia entre 1464 e 11033, com uma média de 5537 k-mers. Observa-se, na tabela, que com base no comprimento médio da sequência de 5566, temos para $k = 10$ uma média de 556 k-mers modificados, com o valor devolvido pelo `BMTC` sendo 372 (na média). Para $k = 20$, temos 278 k-mers modificados, com um custo médio de 191 devolvido pelo `BMTC` para realizar as modificações no grafo. Já para $k = 30$, temos 185 k-mers modificados, com um custo médio de 126 para as modificações.

Nos testes, observamos que a aplicação executa em segundos, e também notamos que aumentar o valor de k resulta em um aumento no tempo de execução. Entretanto, essa aplicação demonstra um desempenho promissor, e a estratégia adotada revela-se eficaz na execução de modificações no grafo.

k	Média da quantidade de k-mers no grafo	k-mers modificados	Custo	Tempo
10	5519	556	372	6,83
20	5547	278	191	8,69
30	5537	185	126	13,43

Tabela 7.16: Média dos custos obtidos com a ferramenta `BMTC` e a média do tempo de execução (em segundos): testes realizados com sequências mapeadas de comprimento médio de 5566.

CAPÍTULO

8

Discussão, Perspectivas e Conclusões

Esta tese representa uma compilação abrangente do trabalho de pesquisa desenvolvido ao longo do programa de doutorado em Ciência da Computação na UFMS. O documento descreve o problema investigado, examina os principais resultados encontrados na literatura, e destaca os resultados obtidos por nós, que foram publicados e apresentados na *the International Conference on Computational Science and its Applications* – ICCSA [Rocha et al. (2023)], evento qualis A3. Além disso, foram conduzidas análises e testes práticos. O foco deste trabalho foi o *Problema do Mapeamento de Sequências em Grafos de De Bruijn* (PMSB). Após uma revisão abrangente da literatura e uma compreensão aprofundada do problema, identificamos duas variantes do PMSB para serem exploradas neste trabalho: mudanças na sequência e mudanças no grafo. Investigamos algoritmos e heurísticas específicos para a primeira variante, explorando também a equivalência entre o grafo de De Bruijn e o grafo de sequência simples. Finalmente, abordamos as mudanças no grafo, destacando a capacidade de induzir novas arestas quando alterações são realizadas. Este trabalho representa uma contribuição relevante para a compreensão e resolução do PMSB, apresentando uma análise abrangente das variantes do problema e explorando diversas abordagens algorítmicas. A combinação de resultados teóricos, revisão de literatura e testes práticos oferece uma visão abrangente do estado da arte do PMSB.

Mudanças na sequência

Falando exclusivamente da variante do PMSB que realiza mudanças na sequência ($\texttt{PMSB}_{p_1}$), foram desenvolvidos dois algoritmos exatos e quatro heurísticas para ela. Os algoritmos exatos estão diretamente relacionados ao algoritmo RaMa, proposto por Rautiainen e Marschall [Rautiainen and Marschall (2017)] para resolver o Problema do Mapeamento de Sequências em Grafos de Sequência (PMSG), e ao algoritmo $\texttt{RaMa}_d$, que fornece apenas a distância do mapeamento.

Para a aplicação do RaMa e $\texttt{RaMa}_d$ no contexto do PMSB, exploramos a equivalência entre

grafos de De Bruijn e grafos de sequência simples. O objetivo principal aqui foi demonstrar que é possível converter um grafo de De Bruijn em um grafo de sequência simples, com as sequências induzidas no grafo de De Bruijn sendo também induzidas no grafo de sequência simples. Inicialmente, propusemos um algoritmo ingênuo que realiza essa conversão para um grafo de De Bruijn de ordem k com n vértices, gerando sempre um grafo de sequência simples com $n \cdot k$ vértices. Em seguida, apresentamos um algoritmo redutor de vértices, que reduz o número de vértices no grafo de sequência simples, podendo gerar um grafo com x vértices, onde $n \leq x \leq n \cdot k$. Nos teoremas propostos demonstramos que as sequências induzidas no grafo de De Bruijn também são induzidas no grafo de sequência simples em ambas as conversões.

Comprovada a possibilidade de converter um grafo de De Bruijn em um grafo de sequência simples, implementamos o primeiro algoritmo denominado De Bruijn Sequence Mapping Tool (BSMT) (Algoritmo 5) que utiliza o grafo de sequência simples para a execução do RaMa. O segundo algoritmo, chamado $\texttt{BSMT}_d$ (Algoritmo 7), também realiza a conversão dos grafos e emprega o algoritmo $\texttt{BSMT}_d$. Testamos ambas as versões, e ambas apresentaram limitações. O BSMT não consegue processar grafos de De Bruijn com mais de 1000 k-mers, principalmente devido ao consumo excessivo de espaço. Já o $\texttt{BSMT}_d$, que fornece apenas a distância do mapeamento, consegue representar grafos maiores em comparação ao BSMT; no entanto, o tempo de execução ultrapassou várias horas. Essas limitações motivaram a exploração de heurísticas para abordar o PMSB.

Em seguida, introduzimos, como nossa primeira heurística, a ferramenta De Bruijn Mapping Tool$_{h1}$ – $\texttt{BSMT}_{h1}$, que visa aprimorar o BSMT ao considerar apenas subcadeias da sequência que não foram ancoradas no grafo. Essa abordagem identifica sementes (k-mers na sequência presentes no grafo) e utiliza o BSMT para mapear os intervalos entre essas sementes em um subgrafo do grafo original, reduzindo o consumo de memória em comparação ao BSMT. Como resultado direto dessa ideia, desenvolvemos a $\texttt{BSMT}_{h1_d}$, que emprega o $\texttt{BSMT}_d$ para calcular apenas a distância do mapeamento. Ambas as heurísticas são direcionadas a lidar com instâncias do mundo real, aliviando a carga computacional ao considerar subgrafos do grafo de De Bruijn e sementes na sequência. Para otimizar ainda mais o desempenho, apresentamos as versões $\texttt{BSMT}_{h1_b}$ e $\texttt{BSMT}_{h1_{db}}$, que incorporam o Bifrost na construção eficiente do grafo de De Bruijn. Essas abordagens e heurísticas visam fornecer soluções eficazes para o problema estudado, considerando desafios práticos e otimizando o uso de recursos computacionais.

Uma alternativa de mapeamento e também muito utilizada na literatura é a abordagem *seed-and-extend*, consideramos essa abordagem e introduzimos duas outras heurísticas, De Bruijn Mapping Tool$_{h2}$ – $\texttt{BSMT}_{h2}$ e De Bruijn Mapping Tool$_{h3}$ – $\texttt{BSMT}_{h3}$, oferecendo abordagens alternativas para o mapeamento de sequências em grafos de De Bruijn. A $\texttt{BSMT}_{h2}$ adota a abordagem *seed-and-extend*, identificando sementes de comprimento k na sequência e estendendo-as à direita. A extensão compara os caracteres subsequentes com os caracteres percorridos no grafo até encontrar uma diferença, devolvendo o passeio mais longo. A $\texttt{BSMT}_{h3}$ melhora a $\texttt{BSMT}_{h2}$ ao permitir tanto substituições quanto exclusões. Se a extensão a partir de uma semente não alcança outra semente, a heurística remove os caracteres do sufixo da semente na sequência e

repete o processo.

Conduzimos experimentos para avaliar as heurísticas propostas para o mapeamento de sequências em grafos de De Bruijn, utilizando conjuntos de dados do NCBI. Métricas de acurácias e desempenho foram empregadas para avaliar as heurísticas. Na Seção 7.1.2, comparamos as heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$, que buscam mapear sequências permitindo apenas mudanças na sequência original. Os resultados indicaram que ambas as heurísticas apresentaram uma acurácia que corresponde a no máximo três vezes à solução exata ($\texttt{BSMT}_d$) e desempenho aceitável para diferentes valores de k. Também na Seção 7.1.2, concentramos nossa análise no $\texttt{BSMT}_d$ e nas heurísticas $\texttt{BSMT}_{h1}$ e $\texttt{BSMT}_{h1_b}$, com o objetivo de avaliar a eficácia do $\texttt{BSMT}_d$, que devolve apenas a distância do mapeamento. Os resultados desses testes foram competitivos, uma vez que o $\texttt{BSMT}_d$ demanda um tempo significativo, ultrapassando vinte e quatro horas em alguns casos, enquanto que as heurísticas conseguem executar em poucos minutos. Esses resultados indicam a utilidade das heurísticas em cenários nos quais apenas a distância do mapeamento é necessária escalando para genomas maiores.

Na Seção 7.1.3, conduzimos uma análise abrangente das quatro heurísticas propostas ($\texttt{BSMT}_{h1_b}$, $\texttt{BSMT}_{h1}$, $\texttt{BSMT}_{h2}$, $\texttt{BSMT}_{h3}$), avaliando-as em termos de acurácia e desempenho. A heurística $\texttt{BSMT}_{h1_b}$ destacou-se ao apresentar a melhor acurácia, mantendo uma boa competitividade em relação ao tempo. As heurísticas $\texttt{BSMT}_{h2}$ e $\texttt{BSMT}_{h3}$, apesar de sua simplicidade, apresentaram resultados próximos às demais. No quesito tempo de execução, a $\texttt{BSMT}_{h2}$ superou a $\texttt{BSMT}_{h3}$. Por outro lado, $\texttt{BSMT}_{h3}$ foi a heurística que registrou o maior tempo de processamento entre as abordagens analisadas.

Adicionalmente, na Seção 7.1.4, comparamos as heurísticas propostas com as ferramentas `BGREAT` e `BrownieAligner`, ambas utilizando a abordagem *seed-and-extend* para realizar o mapeamento. Os resultados revelaram que as nossas heurísticas apresentam acurácia competitiva em relação com essas heurísticas. No entanto, é crucial observar que `BGREAT` e `BrownieAligner` demonstraram ser bem mais eficientes, indicando a necessidade de otimizações em nossas abordagens, apesar da competitividade em acurácia.

Em resumo, os experimentos destacaram a capacidade das heurísticas propostas, especialmente a heurística $\texttt{BSMT}_{h1}$, em mapear sequências no grafo de De Bruijn com uma boa acurácia, representando uma contribuição significativa para a resolução do problema de mapeamento. Em relação ao `BSMT`, $\texttt{BSMT}_{h1}$ possibilita um número maior de mapeamentos, uma vez que o `BSMT` não consegue processar a maioria dos casos de testes. Além dessa heurística, as outras abordagens, que se utilizam *seed-and-extend*, também se mostraram capazes de realizar o mapeamento.

Mudanças no grafo

Ao abordar exclusivamente as mudanças no grafo ($\texttt{PMSB}_{p_2}$), avaliamos o trabalho de Gibney *et al.* [Gibney et al. (2022)], que definiram o problema sem alterar a estrutura do grafo e para essa versão os autores provaram que o problema é NP-completo. No nosso trabalho, permite-se que uma alteração no grafo não apenas modifique os rótulos dos vértices, mas também a sua

topologia. A diferença fundamental reside na flexibilidade permitida, explorando a indução de novos arcos após uma modificação no grafo de De Bruijn. Enquanto Gibney *et al.* se concentram em modificações que preservam a estrutura fundamental do grafo, este trabalho considera que as alterações podem induzir novos arcos e remover outros, possibilitando soluções em tempo polinomial.

Considerando essa possibilidade de mudança no conjunto de arcos, definimos o problema, introduzindo conceitos como a *edição de um grafo de De Bruijn*, *bipartição* e *emparelhamento*. Desenvolvemos um algoritmo denominado `BMTC`, que utiliza o algoritmo húngaro para buscar um emparelhamento máximo de custo mínimo em um grafo bipartido, resultando em um conjunto modificado de vértices do grafo de De Bruijn. A prova de que o algoritmo funciona baseia-se em um teorema que estabelece a relação entre a edição do grafo e os emparelhamentos no grafo bipartido. O teorema demonstra que o custo do emparelhamento máximo encontrado no grafo bipartido é igual à distância para alterar os k-mers no grafo de tal forma que a sequência s é induzida por um passeio no grafo de De Bruijn. Este trabalho é o primeiro a lidar com o problema do mapeamento onde sequências em grafos de De Bruijn com arcos induzidos pelos vértices e por isso o `BMTC` é uma abordagem nova para lidar com esse problema.

Trabalhos futuros

No contexto das soluções desenvolvidas para lidar com alterações na sequência, uma estratégia promissora é focar na melhoria do tempo de execução, tornando-as mais competitivas. Uma possível abordagem é a implementação de paralelismo, utilizando programação multiprocesso de memória compartilhada com o OpenMP [OpenMP (2015)] ou explorando o paralelismo em placas gráficas (GPU) através do CUDA [Nvidia (2006)]. Em soluções paralelas, é viável considerar cada *gap* separadamente, permitindo a busca pelo mapeamento em diferentes núcleos. Além do paralelismo, é válido explorar melhorias no Algoritmo `RaMa`, buscando reduzir o espaço utilizado. Nesse sentido, uma possível abordagem está em utilizar a ideia do Algoritmo de Hirschberg [Hirschberg (1975)] que é utilizado no mapeamento entre duas sequências com o objetivo de reduzir o espaço utilizado [de Brito (2003)]. Isso permitiria ao algoritmo `BSMT` processar grafos maiores, tornando sua utilização mais viável em substituição às heurísticas. Outra alternativa é realizar o mapeamento utilizando outros recursos para determinar sementes, como o algoritmo *k-nearest neighbors* (KNN), conforme explorado em um artigo recentemente publicado por Joudaki *et al.* [Joudaki et al. (2023)].

No contexto da equivalência entre um grafo de De Bruijn e um grafo de sequência simples, é relevante desenvolver uma abordagem para converter o grafo de De Bruijn em um grafo de sequência simples, minimizando o número de vértices e comprovando essa minimalidade. No contexto de mudanças no grafo, nossa estratégia para alterações no grafo é um campo pouco explorado na literatura. Enquanto Gibney *et al.* [Gibney et al. (2022)] abordam o problema de outra forma, nosso trabalho introduz a indução de novos arcos em grafos de De Bruijn, oferecendo uma solução polinomial. No entanto, ainda é necessário investigar as aplicações práticas dessa solução.

Bibliografia

Akutsu, T. (1993). A linear time pattern matching algorithm between a string and a tree. In Annual Symposium on Combinatorial Pattern Matching, páginas 1–10. Springer. Citado nas páginas 2 e 17.

Amir, A., Lewenstein, M., e Lewenstein, N. (1997). Pattern matching in hypertext. Journal of Algorithms, 35(1):82–99. Citado nas páginas 2, 13, 14, 15, 18, 20, e 24.

Benson, D. A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., e Sayers, E. W. (2012). Genbank. Nucleic acids research, 41(D1):D36–D42. Citado na página 52.

Braga, M. D. V. et al. (2000). Grafos de sequências de dna. UNICAMP. Citado na página 13.

de Brito, R. T. (2003). Alinhamento de seqüências biológicas. PhD thesis, Master's thesis, Universidade de Sao Paulo. Citado na página 69.

De Bruijn, N. G. (1946). A combinatorial problem. In Proc. Koninklijke Nederlandse Academie van Wetenschappen, volume 49, páginas 758–764. Citado nas páginas 2 e 13.

Dial, R. B. (1969). Algorithm 360: Shortest-path forest with topological ordering [h]. Communications of the ACM, 12(11):632–633. Citado na página 11.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische mathematik, 1(1):269–271. Citado nas páginas 10 e 22.

Edmonds, J. (1965). Paths, trees, and flowers. Canadian Journal of mathematics, 17:449–467. Citado na página 12.

Gibney, D., Thankachan, S. V., e Aluru, S. (2022). On the hardness of sequence alignment on de bruijn graphs. Journal of Computational Biology, 29(12):1377–1396. Citado nas páginas 4, 24, 26, 46, 68, e 69.

Hamming, R. W. (1950). Error detecting and error correcting codes. Bell System technical journal, 29(2):147–160. Citado na página 7.

Heydari, M., Miclotte, G., Van de Peer, Y., e Fostier, J. (2018). Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. BMC bioinformatics, 19(1):1–10. Citado nas páginas 23, 51, e 62.

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. Communications of the ACM, 18(6):341–343. Citado na página 69.

Holley, G. e Melsted, P. (2020). Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. Genome biology, 21(1):1–20. Citado nas páginas 1 e 41.

Hopcroft, J. E. e Karp, R. M. (1973). An nˆ5/2 algorithm for maximum matchings in bipartite graphs. SIAM Journal on computing, 2(4):225–231. Citado na página 12.

Jain, C., Zhang, H., Gao, Y., e Aluru, S. (2019). On the complexity of sequence-to-graph alignment. Journal of Computational Biology, 27(4):640–654. Citado nas páginas xiii, 2, 3, 13, 19, 20, 21, e 23.

Joudaki, A., Meterez, A., Mustafa, H., Groot Koerkamp, R., Kahles, A., e Ratsch, G. (2023). Aligning distant sequences to graphs using long seed sketches. Genome Res, 33:1208–1217. Citado na página 69.

Karleigh, M., Nathan, L., e Jimin, K. (2022). Hungarian maximum matching algorithm. brilliant.org. retrieved 18:12, may 1, 2024 from https://brilliant.org/wiki/hungarian-matching/. Accessed on May 1, 2024. Citado na página 13.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. Naval research logistics quarterly, 2(1-2):83–97. Citado nas páginas 12 e 49.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In Soviet physics doklady, páginas 707–710. Citado na página 7.

Li, H. e Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. Briefings in bioinformatics, 11(5):473–483. Citado na página 1.

Limasset, A., Cazaux, B., Rivals, E., e Peterlongo, P. (2016). Read mapping on de bruijn graphs. BMC bioinformatics, 17(1):1–12. Citado nas páginas 3, 23, 51, e 62.

Liu, B., Guo, H., Brudno, M., e Wang, Y. (2016). debga: read alignment with de bruijn graph-based seed and extension. Bioinformatics, 32(21):3224–3232. Citado na página 23.

Manber, U. e Wu, S. (1992). Approximate string matching with arbitrary costs for text and hypertext. In Advances In Structural And Syntactic Pattern Recognition, páginas 22–33. World Scientific. Citado nas páginas 2, 16, e 17.

Myers, E. W. (2005). The fragment assembly string graph. Bioinformatics, 21 Suppl 2:ii79–85. Citado na página 2.

Navarro, G. (1998). Improved approximate pattern matching on hypertext. Theoretical Computer Science, 237(1-2):455–463. Citado nas páginas 3 e 18.

Nvidia (2006). NVIDIA Corporation: CUDA Parallel Computing Platform. https://developer.nvidia.com/cuda-zone. Citado na página 69.

Onimaru, K. e Marcon, L. (2019). Systems biology approach to the origin of the tetrapod limb. arXiv preprint arXiv:1907.02730. Citado na página 1.

OpenMP (2015). OpenMP Architecture Review Board: OpenMP Application Programming Interface Version 4.5. https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf. Citado na página 69.

Park, K. e Kim, D. K. (1995). String matching in hypertext. In Annual Symposium on Combinatorial Pattern Matching, páginas 318–329. Springer. Citado nas páginas 2 e 17.

Peixoto, B. M. (2013). Classificação de sequências e análise de diversidade em metagenômica. Universidade Estadual de Campinas. Master's thesis. Citado na página 1.

Pevzner, P. A., Tang, H., e Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. Proceedings of the national academy of sciences, 98(17):9748–9753. Citado na página 2.

Rautiainen, M. e Marschall, T. (2017). Aligning sequences to general graphs in o (v+ me) time. bioRxiv, pagina 216127. Citado nas páginas xiii, 2, 3, 13, 19, 21, 22, 23, 36, e 66.

Rocha, L. B., Adi, S. S., e Araujo, E. (2018). Specific substring problem: an application in bioinformatics. In BSB. Citado na página 1.

Rocha, L. B., Adi, S. S., e Araujo, E. (2023). Heuristics for the de bruijn graph sequence mapping problem. In Gervasi, O., Murgante, B., Taniar, D., Apduhan, B. O., Braga, A. C., Garau, C., e Stratigea, A., editors, Computational Science and Its Applications – ICCSA 2023, páginas 152–169, Cham. Springer Nature Switzerland. Citado nas páginas 5 e 66.

Rocha, L. B., Adi, S. S., e Araujo, E. (2024). The de bruijn graph sequence mapping problem with changes in the graph. bioRxiv, páginas 2024–02. Citado na página 5.

Rocha, L. B., Adi, S. S., Stefanes, M. A., e Araujo, E. (2019). Heuristics for the specific substring problem with hamming distance. In 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering (BIBE), páginas 250–257. IEEE. Citado na página 1.

Schwartz, J., Steger, A., e Weißl, A. (2005). Fast algorithms for weighted bipartite matching. In Nikoletseas, S. E., editor, Experimental and Efficient Algorithms, páginas 476–487, Berlin, Heidelberg. Springer Berlin Heidelberg. Citado na página 13.

Vyverman, M., De Baets, B., Fack, V., e Dawyndt, P. (2013). essamem: finding maximal exact matches using enhanced sparse suffix arrays. Bioinformatics, 29(6):802–804. Citado na página 24.