Prefácio

Este trabalho está relacionado à Biologia Computacional, uma área da Ciência da Computação cuja existência é motivada pela busca de métodos computacionais que resolvam ou ajudem a resolver problemas de origem biológica. Esta ciência tem sido largamente utilizada no âmbito da genética, contribuindo essencialmente no seqüenciamento de cadeias de DNA e no mapeamento de genomas [11].

O foco do nosso projeto foi uma família de problemas denominada *Minimum Contig Problems* (*MCP*) [4], que é um modelo teórico para a abordagem da Montagem de Fragmentos de DNA [11] e que possui uma grande semelhança com um problema de grafos denominado Cobertura de Vértices por Caminhos (*CVC*) [4].

O principal resultado da nossa pesquisa foi a apresentação de provas formais da NP-dificuldade dos problemas de *MCP*. A partir daí, complementamos o nosso trabalho propondo um algoritmo de aproximação para instâncias restritas de cada problema de *MCP*.

No entanto, mesmo tendo concluído o nosso projeto de Mestrado, sabemos que existe um longo caminho a ser percorrido para que seja possível, a partir do nosso trabalho, encontrar métodos que venham a ser realmente utilizados em casos práticos de problemas de Biologia Computacional.

Agradecimentos

Aos meus pais e a todos os meus familiares, pela confiança constante.

Ao meu orientador e aos meus colegas, em especial ao Zanoni e à Lin, pelo indispensável apoio inicial.

Ao Marcos, pelo carinho e companheirismo.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Seqüências de DNA

O ácido desoxirribonucleico ou DNA é uma molécula orgânica de fundamental importância para os organismos, uma vez que é responsável pelo armazenamento dos seus respectivos genes (trechos de DNA que codificam proteínas) e pelos fatores hereditários que transferem esta carga genética através das gerações [11].

O DNA é composto de moléculas menores chamadas nucleotídeos, sendo que cada nucleotídeo é identificado por uma base nitrogenada presente em sua estrutura. Por causa disso, geralmente mencionamos apenas a base quando queremos nos referir ao nucleotídeo. No DNA existem somente quatro tipos diferentes de bases, que são adenina (`A`), citosina (`C`), guanina (`G`) e timina (`T`).

Na molécula de DNA, as bases são distribuídas em duas cadeias emparelhadas. Estas duas cadeias têm o mesmo tamanho e são unidas de forma que uma base do tipo `A` de uma cadeia liga-se sempre à uma base do tipo `T` da outra, assim como uma base do tipo `C` de uma cadeia liga-se sempre à uma base do tipo `G` da outra. Por causa disso chamamos os pares $\mathtt{A}-\mathtt{T}$, $\mathtt{C}-\mathtt{G}$ de pares de bases complementares (figura 1.1), geralmente utilizados como unidade de medida (*base pairs* ou **bp**) do comprimento de uma molécula de DNA.

Além da base, existem outras estruturas compondo cada nucleotídeo, dentre as quais ressaltamos uma molécula de açúcar que contém cinco átomos de carbono, numerados de $1'$ a $5'$. Dois destes átomos de carbono ($3'$ e $5'$) são responsáveis pela ligação de um nucleotídeo com os seus dois vizinhos na cadeia e determinam

```
A   C
|   |
T   G
```

Figura 1.1: Pares de bases complementares

a orientação em que a seqüência de bases desta deve ser lida, que, por convenção, começa na extremidade do carbono $5'$ e termina na extremidade do carbono $3'$ [11]. Sabe-se que, em uma mesma molécula, as duas cadeias possuem orientações opostas, sendo uma denominada o **complemento reverso** da outra [11] (figura 1.2).

```
5'  ACGCCGTTTACAGCTCT  3'
    |||||||||||||||||
3'  TGCGGCAAATGTCGAGA  5'
```

A cadeia superior deve ser lida da esquerda para a direita, resultando na própria seqüência `ACGCCGTTTACAGCTCT`, enquanto que a cadeia inferior, que é o seu complemento reverso, deve ser lida da direita para a esquerda, resultando na seqüência `AGAGCTGTAAACGGCGT`.

Figura 1.2: Trecho das cadeias emparelhadas de uma molécula de DNA

As propriedades descritas acima estabelecem uma relação muito estreita entre as duas cadeias de uma molécula de DNA, de modo que o conhecimento de apenas uma delas é suficiente para se determinar a outra. Tudo isto faz com que o DNA seja representado, de forma bastante apropriada, como uma grande seqüência de caracteres, composta pelos símbolos contidos no alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

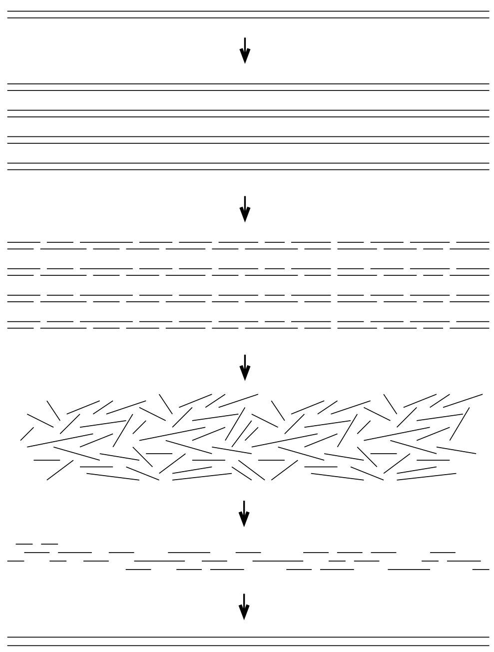
1.2 A montagem de fragmentos

As pesquisas para determinação e análise das seqüências de DNA têm sido conduzidas em ritmo acelerado, uma vez que esta molécula é a peça chave para se desvendar a evolução e as funções de um organismo. Entretanto, os trabalhos neste sentido esbarram sempre na limitação técnica das máquinas seqüenciadoras, que apenas são capazes de ler bases consecutivas de pequenos trechos de uma molécula. Atualmente, estes trechos possuem entre 200 e 1000bp.

Isto torna o seqüenciamento em larga escala de DNA uma tarefa muito complexa, que exige a utilização de estratégias que combinam a manipulação das moléculas em laboratório de biologia com o processamento computacional dos dados obtidos.

Uma destas estratégias é a **Montagem de Fragmentos** (figura 1.3), que pode ser descrita como se segue. Inicialmente, a molécula de DNA a ser seqüenciada é replicada diversas vezes em laboratório. Em seguida, todas as cópias obtidas são quebradas em pontos variados, dando origem a inúmeros pedaços menores. Estes pedaços são então seqüenciados e, a partir dos dados de suas seqüências, tenta-se determinar computacionalmente a ordem correta em que estes fragmentos devem ser agrupados e encaixados para se obter a seqüência original completa [11].

Portanto, do ponto de vista computacional, temos um problema que envolve imensos volumes de dados, além de diversas outras complicações, como a possível ocorrência de erros durante o seqüenciamento.



Inicialmente, a molécula, cuja seqüência é desconhecida, é copiada diversas vezes e fragmentada. Depois disso são determinadas as seqüências dos fragmentos, que serão processados e encaixados para se obter a seqüência completa da molécula original.

Figura 1.3: Abordagem da Montagem de Fragmentos

1.3 Motivação e objetivos

Assim como a Montagem de Fragmentos, os problemas abordados em Biologia Computacional possuem soluções geralmente difíceis e demoradas. Devido a este fato, as pesquisas realizadas nesta área visam estudar a fundo a complexidade destes problemas, buscando formas cada vez mais eficientes para resolvê-los.

Com esta motivação, decidimos estudar problemas baseados em seqüências de DNA e grafos construídos a partir deles. O principal foco da nossa pesquisa, relacionado ao problema da montagem de fragmentos de DNA, foi uma família de problemas denominada *Minimum Contig Problems* (*MCP*) [4].

Já havia sido determinada uma grande semelhança entre esta família e um problema de grafos NP-difícil chamado **Cobertura de Vértices por Caminhos** (*CVC*) [4]. Sabia-se também que alguns problemas da família *MCP* eram NP-difíceis [4], mas a complexidade de todos os demais ainda não havia sido determinada.

Dada a semelhança entre *CVC* e *MCP*, a nossa expectativa era de que os demais problemas desta família também seriam NP-difíceis. Assim, definimos os três objetivos principais do nosso projeto de pesquisa:

1. Determinar a complexidade de todos os problemas da família *MCP*.

2. Determinar a complexidade de todos os problemas da família *MCPr*, que leva em consideração os complementos reversos dos fragmentos de entrada;

3. Pesquisar aproximações para todos os problemas de *MCP*.

1.4 Resumo dos resultados obtidos

Sabendo que parte dos problemas da família *MCP* era NP-difícil [4], o nosso principal objetivo era determinar a complexidade de todos os demais problemas de *MCP* e, conforme o esperado, os estudos realizados mostraram então que estes problemas eram NP-difíceis, a não ser para o caso trivial de alfabetos unitários. Demonstramos também a NP-dificuldade dos problemas da família *MCPr*, que é uma variação de *MCP*.

A partir destes resultados, passamos a pesquisar aproximações para a família *MCP*, verificando então que, excluindo também o caso trivial, todo problema desta família não admitia aproximação que garantisse solução de custo menor que duas vezes o custo ótimo.

Por fim, nos concentramos na tentativa de elaborar algoritmos de aproximação para *MCP*. Mostramos que qualquer algoritmo de aproximação para *CVC* poderia ser adaptado para aproximar todos os problemas de *MCP*, mas, infelizmente, o algoritmo que propusemos se mostrou viável para aproximar apenas um subconjunto das instâncias de *CVC*. Entretanto, percebemos que, mesmo com esta restrição, a utilização do algoritmo poderia ser interessante nas situações geralmente encontradas nos casos práticos de Montagem de Fragmentos e decidimos, assim, aproveitá-lo como um resultado positivo do nosso trabalho.

1.5 Sumário

O DNA é uma molécula orgânica, em cuja estrutura destacamos a presença de duas cadeias de nucleotídeos compostas por quatro tipos de bases ($\mathtt{A}, \mathtt{C}, \mathtt{G}$ ou $\mathtt{T}$) e emparelhadas pelos pares de bases complementares ($\mathtt{A} - \mathtt{T}$, $\mathtt{C} - \mathtt{G}$). O fato das duas cadeias de uma molécula possuírem orientações opostas completam a estreita relação que existe entre elas. Por causa disso, uma é denominada **complemento reverso** da outra.

A abordagem da Montagem de Fragmentos [11] é uma estratégia utilizada para o seqüenciamento de DNA. Esta estratégia é base de uma família de problemas denominada *Minimum Contig Problems* (*MCP*) [4], que é o principal foco do nosso projeto de pesquisa.

Sabia-se que alguns problemas em *MCP* eram NP-difíceis [4], sendo que a complexidade dos demais ainda era desconhecida. Assim, o primeiro objetivo do nosso projeto era completar o estudo da complexidade dos problemas de *MCP* e os nossos resultados mostraram que todo problema desta família era NP-difícil. Em seguida, provamos que não era possível obter uma aproximação para qualquer problema de *MCP* que garantisse solução menor que duas vezes a solução ótima e propusemos um algoritmo de aproximação aplicável a instâncias restritas de todos os problemas de *MCP*.

Capítulo 2

Definições e notações

Palavras ou cadeias de caracteres, bem como o processo de construção destas, são elementos comuns nos problemas de Biologia Computacional, que se baseiam em seqüências de DNA. Agora, vamos revisar e introduzir algumas definições de alfabetos, cadeias e grafos, com as suas respectivas notações.

O conceito de conjunto, que é uma das principais bases matemáticas do nosso trabalho, aparecerá com muita freqüência nesta dissertação. De um modo geral, utilizaremos chaves ("{" e "}") para representar conjuntos não ordenados e parênteses ("(" e ")") para representar conjuntos ordenados. Dado um conjunto qualquer C, o tamanho de C, denotado por $|C|$, é o número de elementos em C. Assim, se $C = \{a, b, c, d, e\}$, temos $|C| = 5$.

Antes de mais nada, introduziremos a idéia de estender a aplicação de uma função a um subconjunto de seu domínio, que será bastante utilizada em nossas demonstrações.

Definição 2.1 *Dada uma função $f : D \longrightarrow I$, se um conjunto $A = \{a_1, a_2, \ldots a_{|A|}\}$ é tal que $A \subseteq D$, consideraremos $f(A) = \{f(a_1), f(a_2), \ldots f(a_{|A|})\}$, ressaltando que a ordem dos elementos em $f(A)$ é irrelevante. Mas se A for um conjunto ordenado, então teremos obrigatoriamente $f(A) = (f(a_1), f(a_2), \ldots, f(a_{|A|-1}), f(a_{|A|}))$.*

A definição acima vale para qualquer função em geral, mas sofre um pequena modificação, relacionada a conjuntos ordenados, no caso de funções que são homomorfismos reversos, como poderá ser visto na seção 2.1.1.

2.1 Alfabetos e cadeias

Um **alfabeto** é um conjunto finito e não vazio de caracteres. Usamos geralmente a letra grega Σ para designar alfabetos. Um exemplo de alfabeto é $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, com $|\Sigma| = 4$ [8].

Dizemos que uma **cadeia** ou **seqüência** de caracteres $s = c_1c_2\ldots c_q$ é escrita sobre um alfabeto Σ, quando $c_i \in \Sigma$, para todo $1 \leq i \leq q$. Um exemplo de cadeia escrita sobre o alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ é `TTCAGCATG`. O comprimento de uma cadeia s, denotado por $|s|$, é dado pelo seu número de caracteres. Quando $|s| = 0$, dizemos que s é a **seqüência vazia** e a designamos pelo símbolo especial ε [8].

Dado um conjunto de cadeias $C = \{s_1, s_2, \ldots, s_{|C|}\}$, denotamos por $\|C\|$ o inteiro dado por $\sum_{i=1}^{|C|} |s_i|$.

Um alfabeto pode dar origem a conjuntos formados por cadeias escritas sobre ele. Assim, dado um alfabeto Σ, o conjunto Σ^+ agrupa todas as cadeias de comprimento maior ou igual a um escritas sobre Σ, assim como o conjunto Σ^* agrupa todas as cadeias de comprimento maior ou igual a zero escritas sobre Σ e Σ^i é conjunto de todas as cadeias de comprimento i escritas sobre Σ, onde i é um inteiro tal que $i \geq 0$ [8]. Tomando como exemplo o alfabeto $\Sigma = \{0, 1\}$, temos $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$, $\Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, \ldots\}$ e $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

Utilizaremos com freqüência letras minúsculas para representar caracteres e cadeias. Em geral, letras que aparecem no princípio do nosso alfabeto, como a, b, c e d, representarão caracteres, enquanto que as letras que aparecem no final do alfabeto, como s, t, u, v, w e x, serão usadas para cadeias. Do mesmo modo, letras maiúsculas serão comumente usadas para representar conjuntos de cadeias.

Se $s = a_1a_2\ldots a_{|s|}$ e $t = b_1b_2\ldots b_{|t|}$, a cadeia $u = a_1a_2\ldots a_{|s|}b_1b_2\ldots b_{|t|}$ é denominada a **concatenação** de s e t, denotada por $u = st$ ou $u = s.t$. A cadeia `ACGTCTCCGGC`, por exemplo, é a concatenação de `ACGTC` e `TCCGGC`. Para concatenar mais de duas cadeias, basta fazer a concatenação das duas primeiras, em seguida fazer a concatenação do resultado com a terceira cadeia e assim por diante.

Dados uma cadeia s e um inteiro $k \geq 1$, definimos s^k como sendo a cadeia $sss\ldots s$ (k vezes).

Quando temos uma cadeia s, tal que $s = tuv$, a cadeia u é chamada de **subcadeia** de s. Se u é uma subcadeia de s e $|u| < |s|$, então u é denominada **subcadeia própria** de s. Além disso, se $s = uv$, a cadeia u é um **prefixo** de s. Da mesma forma, se

$s = tu$, a cadeia u é um **sufixo** de s. As cadeias TTAC e CTA são, respectivamente, um prefixo e um sufixo de TTACGCTA.

Dadas duas cadeias s e w, tais que $s = tu$ e $w = uv$, a cadeia u é dita **sobreposição** entre s e w. Note, porém, que a mesma cadeia u pode não ser sobreposição entre w e s. Se u é sobreposição entre s e w e não existe cadeia u', tal que $|u'| > |u|$ e u' também seja sobreposição entre s e w, então u é denominada **sobreposição máxima** entre s e w, denotada por $u = spmax(s, w)$.

As definições de subcadeia e sobreposição podem ser resumidas pelo conceito de **ocorrência**. Dizemos, portanto, que uma cadeia $t \neq \varepsilon$ **ocorre** em uma cadeia s quando t é subcadeia de s ou quando $spmax(s, t) \neq \varepsilon$. Portanto, a cadeia ε não ocorre em nenhuma cadeia, e nenhuma cadeia ocorre em ε. Se considerarmos as cadeias $s = \texttt{ACGTCGTAACG}$, $t = \texttt{TGACG}$ e $u = \texttt{GTA}$, temos ocorrências de s em t e em u e ocorrências de u em s e em t, mas t não ocorre nem em s, nem em u.

Dada uma sobreposição entre duas cadeias, podemos formar uma terceira cadeia simplesmente aglutinando as duas primeiras. Desta forma, se $s = tu$ e $w = uv$, a cadeia $x = tuv$ é uma aglutinação de s e w. Mais que isso, se $u = spmax(s, w)$, então x é denominada o **consenso** de s e w, denotado por $x = s : w$. A cadeia ACGTCCGGC, por exemplo, é o consenso de ACGTC e TCCGGC.

O conceito de consenso pode ser estendido para um conjunto ordenado de cadeias S. Tomando $S = (s_1, s_2, s_3 \dots s_{|S|})$ e escrevendo $s_i = v_i u_i$, onde $v_i = spmax(s_{i-1}, s_i)$, para todo $2 \leq i \leq |S|$, a cadeia s é dita o consenso de S, denotado por $s = s_1 : s_2 : s_3 : \dots s_{|S|}$, se $s = s_1 u_2 u_3 \dots u_{|S|}$.

Definição 2.1.1 *Dados um inteiro $k \geq 0$ e um conjunto ordenado de cadeias $S = (s_1, s_2, \dots s_{|S|})$, S é chamado* **k-caminho** *(figura 2.1) se, para todo $2 \leq i \leq |S|$, o par de cadeias consecutivas s_{i-1}, s_i é tal que $|spmax(s_{i-1}, s_i)| \geq k$. Dizemos ainda que S é um k-caminho em um conjunto de cadeias C quando S é um subconjunto de C.*

Dados um conjunto de cadeias C e um k-caminho $S = (s_1, s_2, \dots s_{|S|})$ em C, se $|spmax(s_{|S|}, s_1)| \geq k$, dizemos que S é também um k-ciclo em C.

Por fim, podemos também estabelecer relações entre uma cadeia s e um conjunto de cadeias C. Dizemos que C **cobre** s quando s é subcadeia de uma cadeia $t \in C$. Portanto, o conjunto $\{\texttt{AACGTG}, \texttt{GTGCACCGT}, \texttt{GTAA}, \texttt{TGCAC}\}$ cobre cadeias como ACG e ACC, além das próprias cadeias que ele contém. Por outro lado, a cadeia s é dita **terminal** em C se $s \in C$ e, para toda cadeia $t \in C$, temos que s não é subcadeia própria de t. Considerando-se o mesmo conjunto {AACGTG, GTGCACCGT, GTAA, TGCAC}, podemos verificar que apenas as cadeias AACGTG, GTGCACCGT e GTAA são terminais.

Definição 2.1.2 *Dado um conjunto de cadeias C, chamamos de $T(C)$ o conjunto de cadeias terminais em C.*

```
        s =  .TAC.......

        x =  TTACGCT....
        t =  ....GCTCG..
        v =  .......CGAA
             -----------
x : t : v =  TTACGCTCGAA
```

Dado um conjunto $C = \{s, t, u, v, w, x\}$, tal que $s =$ TAC,
$t =$ GCTCG, $u =$ CACA, $v =$ CGAA, $w =$ CAATA e $x =$ TTACGCT,
o conjunto ordenado de cadeias (x, t, v) é um 2-caminho em C,
que cobre também a cadeia s, já que esta é subcadeia de x.

Figura 2.1: 2-caminho e seu consenso

2.1.1 Homomorfismos

Dados alfabetos Σ e Υ, um **homomorfismo** $g : \Sigma^* \longrightarrow \Upsilon^*$ é uma função tal que, para quaisquer $u, v \in \Sigma^*$, podemos dizer que $g(uv) = g(u)g(v)$. Como conseqüência desta propriedade, temos $g(\varepsilon) = \varepsilon$.

Podemos alterar a definição acima para criar o conceito de **homomorfismo reverso**. Uma função $g : \Sigma^* \longrightarrow \Upsilon^*$ é dita um homomorfismo reverso quando, para quaisquer $u, v \in \Sigma^*$, temos $g(uv) = g(v)g(u)$. Neste caso é preciso modificar também a definição 2.1, no trecho relacionado a conjuntos ordenados, de modo que, dado conjunto de cadeias ordenado $A = (a_1, a_2, \ldots a_{|A|})$, com $A \subseteq \Sigma^*$, obrigatoriamente $g(A) = (g(a_{|A|}), g(a_{|A|-1}), \ldots, g(a_2), g(a_1))$.

Definição 2.1.1.1 *Dados alfabetos Σ e Υ e um homomorfismo $g : \Sigma^* \longrightarrow \Upsilon^*$, dizemos que g* **preserva ocorrências** *quando g é função injetora onde, para todo $c \in \Sigma$, a cadeia $g(c)$ possui tamanho fixo, denotado por $|g|$ e, além disso, dados $s, t \in \Sigma^*$:*

- *$|spmax(s, t)||g| = |spmax(g(s), g(t))|$ (ou $|spmax(s, t)||g| = |spmax(g(t), g(s))|$, se g for um homomorfismo reverso);*

- *t é subcadeia própria de s se, e somente se, $g(t)$ é subcadeia própria de $g(s)$.*

Um exemplo de homomorfismo que preserva ocorrências pode ser encontrado na figura 4.2.

2.1.2 Seqüências de DNA

Cada fragmento de uma cadeia de DNA é representado por uma seqüência de caracteres escrita sobre o alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, denominada seqüência de DNA. Para toda seqüência de DNA, existe uma outra seqüência que corresponde ao seu complemento reverso. Denotamos por $\overline{s}$ o complemento reverso da seqüência s. Assim, se $s = \mathtt{AACTGACG}$, $\overline{s} = \mathtt{CGTCAGTT}$.

Do mesmo modo, dado um conjunto de seqüências de DNA C, designamos por $\overline{C}$ o conjunto formado pelos complementos reversos de todas as cadeias de C. Para exemplificar, se $C = \{\mathtt{GTGCACC}, \mathtt{GTAA}, \mathtt{TGCAC}, \mathtt{TTAC}\}$, então $\overline{C} = \{\mathtt{GGTGCAC}, \mathtt{TTAC}, \mathtt{GTGCA}, \mathtt{GTAA}\}$ e, portanto, $C \cup \overline{C} = C \cup \{\mathtt{GGTGCAC}, \mathtt{GTGCA}\}$.

2.2 Grafos

Um grafo é, basicamente, uma forma de modelar um problema. Este modelo é freqüentemente utilizado, já que as propriedades dos grafos são bem conhecidas e podem ser aproveitadas na construção de soluções e no estudo da complexidade dos problemas abordados [3].

Um grafo G é um conjunto de vértices (ou nós) ligados entre si por linhas denominadas arestas. Os conjuntos de vértices e arestas de um grafo G são dados, respectivamente, por $V(G)$ e $E(G)$.

Cada vértice pode ter muitas arestas incidindo sobre ele, e consideraremos aqui somente os grafos que não possuem arestas iguais e nos quais cada aresta sempre une dois vértices distintos. O número de arestas que incidem em um vértice determina o **grau** deste vértice. Dado um grafo G, denotaremos por $d(u)$ o grau de um vértice $u \in V(G)$. Denotaremos também por $\Delta(G)$ o grau máximo em G. Assim, temos $\Delta(G) = \max_{u \in V(G)} d(u)$.

Um grafo é dito **orientado** quando suas arestas são orientadas desde um vértice de origem até um vértice de destino. Caso contrário, o grafo é dito **não orientado**.

Por outro lado, um grafo é **ponderado** quando a cada aresta é associado um peso.

Como já foi visto, grafos são representados geralmente pela letra G. Utilizaremos então letras minúsculas, como s, t, u, v, w e x para representar vértices. Uma aresta, por sua vez, pode ser representada pelos dois vértices unidos por ela. Se uma aresta liga um vértice u a um vértice v em um grafo não-orientado, a sua representação é $\{u, v\}$. No caso de grafos orientados, a ordem dos vértices deve ser levada em consideração nesta representação, sendo que primeiro deve aparecer o vértice de origem e depois o vértice de destino da aresta. Portanto, em um grafo orientado, a aresta (u, v) seria distinta da aresta (v, u).

Definição 2.2.1 *Dois grafos G e G' são ditos* **isomorfos***, denotados por $G \simeq G'$ se existe uma bijeção $f : V(G) \longrightarrow V(G')$, tal que $E(G') = \{(f(u), f(v)) \mid (u, v) \in E(G)\}$. Além disso, a bijeção f é chamada de isomorfismo entre G e G'.*

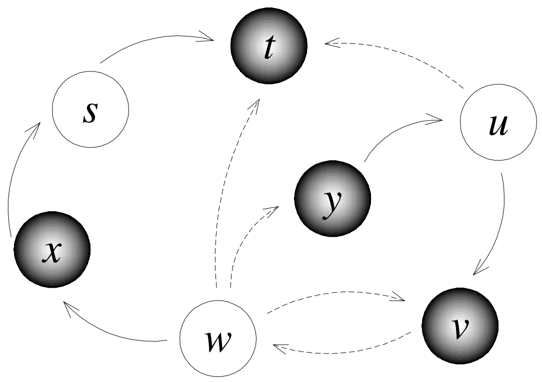
Um **caminho** em um grafo G é definido por uma seqüência alternada de vértices e arestas, sendo que tanto a sua origem quanto o seu destino devem ser vértices e, além disso, o caminho deve passar por cada vértice ou aresta do grafo no máximo uma vez. Dizemos que um elemento de um grafo (isto é, uma aresta ou um vértice) é coberto por um caminho quando este elemento faz parte da seqüência que compõe o caminho. Uma vez que cada aresta une apenas dois vértices distintos, a seqüência de vértices é suficiente para representar o caminho. Por uma questão de conveniência, definiremos um caminho como um conjunto ordenado de vértices $U = (u_1, u_2, \ldots, u_{|U|})$, de modo que, para todo $1 \leq i < |U|$, a aresta $(u_i, u_{i+1}) \in E(G)$.

Dado um grafo G e um caminho $U = (u_1, u_2, \ldots, u_{|U|})$ em G, se $(u_{|U|}, u_1) \in E(G)$, dizemos que U é também um ciclo em G.

Dizemos que um grafo G é **conexo** quando, para quaisquer $u, v \in V(G)$, existe um caminho que vai u para v. Dizemos ainda que G é grafo **completo** quando, também para quaisquer vértices $u, v \in V(G)$, temos $(u, v), (v, u) \in E(G)$, se o grafo for orientado, e $\{u, v\} \in E(G)$, se o grafo for não orientado.

Um **conjunto independente** em um grafo G é um subconjunto V' de $V(G)$ no qual, para todo par de vértices $u, v \in V'$, as arestas (u, v) e (v, u) não pertencem a $E(G)$. Dizemos que um grafo é **bipartido** (figura 2.2) quando todos os seus vértices podem ser agrupados em dois conjuntos independentes A e B tal que $A \cup B = V(G)$ e $A \cap B = \emptyset$.

Um **emparelhamento** (figura 2.3), por sua vez, é um subconjunto $E' \subseteq E(G)$ no



Como os vértices do grafo acima podem ser divididos nos conjuntos independentes $\{s, u, w\}$ e $\{t, v, x, y\}$, este é um grafo bipartido.

Além disso, os conjuntos (w, x, s, t) e (y, u, v) são caminhos disjuntos no mesmo grafo.

Figura 2.2: Caminhos em um grafo bipartido

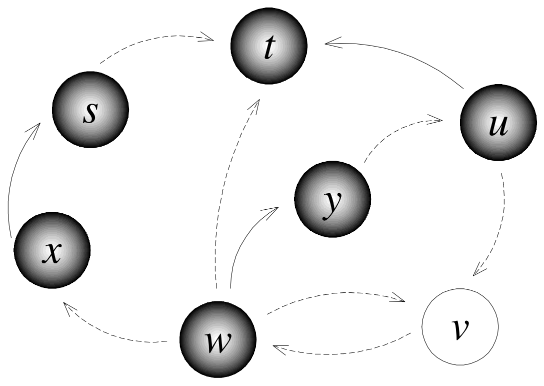
qual, para todo par de arestas $(u, v), (z, w) \in E'$, temos sempre $u \neq z$ e $u \neq w$, como também temos $v \neq z$ e $v \neq w$, ou seja, não existem duas arestas em E' incidindo sobre o mesmo vértice. Um emparelhamento E' é dito **maximal** se, para toda aresta (x, y) em $E(G) \setminus E'$, pelo menos um entre os vértices x e y é coberto por alguma aresta em E'.

Além disso, uma **coloração de arestas** em G é um conjunto de emparelhamentos disjuntos $\mathcal{C} = \{M_1, M_2, \ldots, M_{|\mathcal{C}|}\}$, tal que $M_1 \cup M_2 \cup M_3 \ldots \cup M_{|\mathcal{C}|} = E(G)$. Sabe-se, pelo teorema de Vizing [3, p.103], que uma coloração de arestas mínima $\mathcal{C}$ para um grafo G é tal que $|\mathcal{C}| = \Delta(G)$ ou $|\mathcal{C}| = \Delta(G) + 1$.

2.2.1 Grafos de sobreposição

Definição 2.2.1.1 *Dado um conjunto de cadeias C escrito sobre um alfabeto Σ, o grafo de k-sobreposição de C, denotado por $G_k(C)$, é o grafo G orientado tal que $V(G) = C$ e $E(G) = \{(u, v) \mid spmax(u, v) \geq k\}$.*

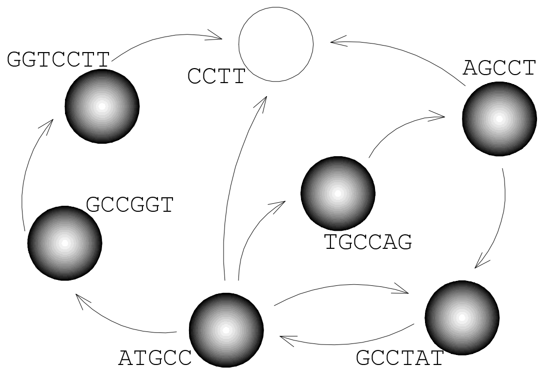
Um vértice v de $G_k(C)$ é denominado **terminal** quando $v \in T(C)$. Portanto, a propriedade de um vértice ser ou não terminal não depende de k.



O conjunto $\{(u,t),(x,s),(w,y)\}$ é um emparelhamento de arestas maximal.

Figura 2.3: Emparelhamento de arestas em um grafo

Um exemplo de grafo de sobreposição pode ser visto na figura 2.4.



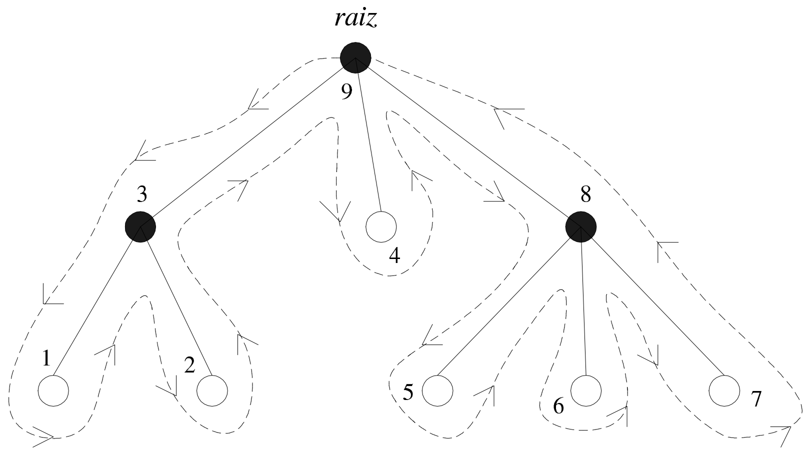
$C = \{\texttt{ATGCC}, \texttt{GCCTAT}, \texttt{AGCCT}, \texttt{CCTT}, \texttt{GGTCCTT}, \texttt{GCCGGT}, \texttt{TGCCAG}\}$
$T(C) = \{\texttt{ATGCC}, \texttt{GCCTAT}, \texttt{AGCCT}, \texttt{GGTCCTT}, \texttt{GCCGGT}, \texttt{TGCCAG}\}$

Figura 2.4: Grafo de 2-sobreposição e conjunto de terminais

2.2.2 Árvores

Um grafo não-orientado, conexo e acíclico pode ser também chamado de árvore [3]. Os vértices de grau um (ou zero, se a árvore for um único vértice) de uma árvore são chamados **folhas**, enquanto que os vértices de grau maior que um são chamados **nós**.

As árvores geralmente possuem um vértice especial, chamado **raiz**, segundo o qual se define o nível de profundidade de um vértice. Sabendo que a distância entre dois vértices em um grafo qualquer é dada pelo número de arestas presente no menor caminho existente entre eles, dizemos que um vértice v está no nível i de uma árvore quando a distância entre v e a raiz é i. Portanto, para todo $i \geq 1$ um vértice de nível i está sempre ligado a exatamente um vértice de nível $i-1$, podendo estar ligado a vários ou a nenhum vértice de nível $i+1$. Se o vértice u está no nível i, ligado aos vértices v e w, que estão no nível $i+1$, u é dito "pai" de v e w, do mesmo modo que v e w são ditos "filhos" de u.



As folhas são os vértices brancos e os nós são os vértices pretos.

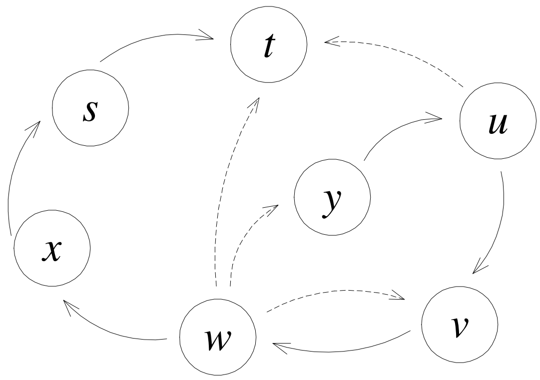
A linha tracejada indica o caminho percorrido durante uma busca em profundidade nesta árvore (os números representam a ordem em que os vértices são processados).

Figura 2.5: Busca em profundidade em uma árvore

Existem várias maneiras de se percorrer uma árvore, sendo uma delas em profundidade, que consiste em, partindo da raiz, processar cada vértice somente depois que todos os seus filhos tiverem sido processados (figura 2.5).

2.2.3 Alguns problemas clássicos

Existem problemas que, apesar de estarem relacionados a diferentes áreas da ciência ou mesmo do cotidiano, possuem exatamente as mesmas propriedades e compartilham o mesmo modelo em grafos.



O conjunto (y, u, v, w, x, s, t) é um caminho Hamiltoniano neste grafo, ou seja, é um caminho que passa por todos os vértices.

Figura 2.6: Caminho Hamiltoniano

Um destes problemas é o **Caminho Hamiltoniano** (HAM, figura 2.6), que consiste em determinar se existe um caminho que cobre todos os vértices de um grafo qualquer.

2.3 Sumário

Neste capítulo apresentamos definições e notações envolvendo alfabetos, cadeias de caracteres e grafos, que serão utilizadas ao longo desta dissertação. Alguns dos conceitos introduzidos aqui, como k-caminhos, homomorfismos que preservam ocorrências e grafos de k-sobreposição serão fundamentais na descrição dos problemas que estudamos e na demonstração de suas respectivas complexidades.

Capítulo 3

A família *Minimum Contig Problems* ou *MCP*

Um problema de otimização é um problema no qual, para cada instância I, temos um conjunto de soluções válidas $F(I)$ e ainda, para cada solução $K \in F(I)$, temos um valor $custo\,(K)$, inteiro e positivo. O custo ótimo é então definido como $OPT(I) = \min_{K \in F(I)} custo\,(K)$, se o problema for de minimização (ou $\max_{K \in F(I)} custo\,(K)$, se o problema for de maximização) [10].

3.1 Definição

Recentemente, Ferreira, Souza e Wakabayashi [4] estudaram uma família de problemas de minimização que chamamos *Minimum Contig Problems* (*MCP*). Estes problemas têm sua origem na área biológica, servindo para modelar o problema da Montagem de Fragmentos descrito anteriormente e que é parte integrante de qualquer projeto de seqüenciamento em larga escala de DNA [11].

Na verdade, durante o nosso trabalho, ao revisar as definições dos problemas de *MCP*, decidimos alterar a sua denominação, que originalmente era *Minimum k-Contig Problems* [4]. Além disso, passamos a designar por k-caminhos o que os autores do trabalho original chamavam de k-contigs, mas decidimos manter a palavra *contig* na denominação da família. Esta substituição foi feita pelo fato da palavra *contig* ser comumente utilizada em biologia computacional como referência a alinhamentos múltiplos de seqüências [11]. Deste modo, diferentemente do que foi definido por Ferreira, Souza e Wakabayashi [4], é mais apropriado dizer que um k-contig é um alinhamento múltiplo de cadeias que exige que a sobreposição entre duas cadeias vizinhas seja de pelo menos k caracteres, ou que uma seja subcadeia da outra.

Cada $MCP_n(k)$ é definido para um alfabeto Σ, com $|\Sigma| = n$, e um inteiro k (figuras 3.1 e 3.2).

Entrada: Conjunto de seqüências de caracteres C, escrito sobre Σ.

Objetivo: Sabendo que $F(C)$ é o conjunto formado pelos conjuntos de k-caminhos disjuntos em C que cobrem todas as seqüências de C, o objetivo do $MCP_n(k)$ é encontrar $H \in F(C)$, tal que $|H| = \min_{K \in F(C)} |K|$.

Saída: Conjunto de k-caminhos obtidos.

Em outras palavras, dada uma instância C, o objetivo do $MCP_n(k)$ é encontrar o menor conjunto de k-caminhos em C, tal que cada cadeia de C deve pertencer a exatamente um k-caminho, se for terminal em C, e a no máximo um k-caminho, se não for terminal em C. Note, portanto, que os k-caminhos obtidos na solução podem conter apenas as cadeias terminais em C.

```
s   TTACGCTA.........
t   TTACGC...........
u   ....GCTAGAGATC...
v   ......TAGAGAT....
w   ...........ATCGTT
    -----------------
    TTACGCTAGAGATCGTT
```

Neste caso, todas as seqüências de entrada são cobertas por um único 2−caminho, que é (s, u, w) e cujo consenso é `TTACGCTAGAGATCGTT`.

Figura 3.1: $MCP_4(2)$

3.2 A versão de decisão da família *MCP*

Chamamos de *DMCP* a família que é a versão de decisão de *MCP*. Cada $DMCP_n(k)$ também é definido para um alfabeto Σ, com $|\Sigma| = n$, e um inteiro $k \geq 1$ e determina se todas as cadeias de um conjunto de entrada C, escrito sobre Σ, podem ser cobertas por no máximo i k-caminhos disjuntos em C, onde i é parte da entrada do problema.

```
s   TTACGCTA...............
t   TTACGC.................
u   ....GCTAGAGATC.........
v   ......TAGAGAT..........
w   .................ATCGTT
    -----------------------
    TTACGCTAGAGATC...ATCGTT
```

Neste caso, as seqüências s, t, u e v são cobertas pelo $4-$caminho (s,u), cujo consenso é `TTACGCTAGAGATC` e a seqüência w é coberta pelo $4-$caminho (w), cujo consenso é `ATCGTT`.

Figura 3.2: $MCP_4(4)$

3.3 Variações da família MCP

No problema real encontrado em biologia, que é a motivação do nosso projeto, diversos outros fatores devem ser levados em consideração na família MCP, especialmente a ocorrência de erros na determinação da seqüência dos fragmentos e a questão da complementaridade [4].

3.3.1 A família $MCP\varepsilon$

Para lidar com os erros, poderíamos estender a definição da família MCP para a família $MCP\varepsilon$, onde admitimos que uma ocorrência de uma cadeia t em uma cadeia s possua alguns erros, sendo que um erro ocorre quando um caracter de uma cadeia é alinhado com um caracter diferente ou com um espaço inserido na outra [11].

Cada $MCP\varepsilon_n(k)$ é definido para um alfabeto Σ, com $|\Sigma| = n$, um inteiro $k \geq 1$ e um real $\varepsilon \geq 0$ (figura 3.3).

Entrada: Conjunto de seqüências de caracteres C, escrito sobre Σ.

Objetivo: Sabendo que $F(C)$ é o conjunto formado pelos conjuntos de k-caminhos disjuntos em C que cobrem todas as seqüências de C, considerando que uma ocorrência de uma cadeia t em uma cadeia s pode ter até $\varepsilon.|u|$ erros, se a ocorrência for uma sobreposição u, e até $\varepsilon.|t|$ erros, se a ocorrência for de subcadeia, o objetivo do $MCP\varepsilon_n(k)$ é encontrar $H \in F(C)$, tal que $|H| = \min_{K \in F(C)} |K|$.

Saída: conjunto de k-caminhos obtidos.

Podemos observar que a família *MCP* é um caso especial da família $MCP\varepsilon$, onde temos $\varepsilon = 0$.

```
s   TTACGCTA.........
t   TTGCGG...........
u   ....GCAAGAGATC...
v   ......TAGAGAC....
w   ...........ACCGTT
```

Neste caso, temos $\varepsilon = 1/3$ e todas as seqüências de entrada são cobertas por um único 2−caminho, que pode ser (s, u, w).

Figura 3.3: $MCP\varepsilon_4(2)$

3.3.2 A família *MCPr*

Na questão da complementaridade, o problema reside no fato de que, pela própria natureza do processo de fragmentação, não é possível saber a princípio qual das duas cadeias da molécula de DNA originou cada fragmento. A estratégia adotada neste caso é estender a definição de *MCP* para *MCPr*, uma família de problemas que leva em consideração os complementos reversos de todos os fragmentos incluídos em suas instâncias [11].

A família *MCPr* é definida apenas para o alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ e cada $MCPr(k)$ é definido para um inteiro $k \geq 1$ (figura 3.4).

Entrada: Conjunto de seqüências de caracteres C, escrito sobre $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

Objetivo: Denotaremos por $F(C)$ o conjunto formado por conjuntos de k-caminhos disjuntos em $C \cup \overline{C}$, de modo que, se $H \in F(C)$, para toda cadeia $s \in C$, se s for terminal em $C \cup \overline{C}$, precisamente uma cadeia entre s e $\overline{s}$ deve pertencer a algum k-caminho de H e, se s não for terminal em $C \cup \overline{C}$, no máximo uma cadeia entre s e $\overline{s}$ pode pertencer a algum k-caminho de H. Assim, o objetivo do $MCPr(k)$ é encontrar $H \in F(C)$, tal que $|H| = \min_{K \in F(C)} |K|$.

Saída: conjunto de k-caminhos obtidos.

Podemos perceber, então, que um conjunto de k-caminhos em $C \setminus \overline{C}$ que cubram todas as cadeias de $C \setminus \overline{C}$, ou vice-versa, é uma solução válida de $MCPr(k)$ para C.

```
s   TTACGCTA.........        s̄   .........TAGCGTAA
u   ....GCTAGAGATC...        ū   ...GATCTCTAGC....
w   ...........ATCGTT        w̄   AACGAT...........
    -----------------            -----------------
    TTACGCTAGAGATCGTT            AACGATCTCTAGCGTAA
```

Consideramos apenas as cadeias terminais no conjunto mostrado na figura 3.1. Neste caso, uma solução pode ser o 2−caminho $(\overline{w}, \overline{u}, \overline{s})$, cujo consenso é a seqüência AACGATCTCTAGCGTAA. Outra possibilidade seria a própria solução (s, u, w) da figura 3.1.

Figura 3.4: $MCPr(2)$

3.4 Cobertura de Vértices por Caminhos ou *CVC*

Dado um grafo orientado G e um conjunto $T \subseteq V(G)$, chamamos de *Cobertura de Vértices por Caminhos* (CVC) [4] o problema de determinar um conjunto U de caminhos orientados disjuntos em G, tal que U cobre todos os vértices em T e $|U|$ é mínimo (figuras 3.5 e 3.6).

Entrada: Grafo orientado G, e um conjunto de vértices $T \subseteq V(G)$.

Objetivo: Sabendo que $F(G,T)$ é um conjunto formado pelos conjuntos de caminhos orientados disjuntos em G que cobrem todos os vértices de T, o objetivo do CVC é encontrar $H \in F(G,T)$, tal que $|H| = \min_{K \in F(G,T)} |K|$.

Saída: Conjunto de caminhos obtidos.

Denominamos $DCVC$ a versão de decisão de CVC, tal que, dados um inteiro i, um grafo G e um conjunto $T \subseteq V(G)$, o objetivo de $DCVC$ é determinar se T pode ser coberto por no máximo i caminhos orientados disjuntos.

Teorema 3.4.1 *CVC é NP-difícil.*

Prova:

Tomamos um grafo G, orientado, e utilizamos $DCVC$ para resolver o problema do caminho hamiltoniano para G. Para isso, resolvemos $DCVC$ para $(G, V(G), 1)$, ou seja, determinamos se todos os vértices de G podem ser cobertos por no máximo um caminho. Se a resposta for SIM, então G é hamiltoniano e, se a resposta for NÃO, então G não é hamiltoniano. Portanto, CVC é NP-difícil.

□

Teorema 3.4.2 *Dados um inteiro $k \geq 1$, um alfabeto finito Σ e um conjunto de seqüências C, escrito sobre Σ, podemos construir, em tempo $O(\|C\| + |C|^2)$, o grafo $G_k(C)$ e o conjunto $T(C)$.*

Prova:

Gusfield, Landau e Schieber [7] propuseram um algoritmo eficiente para calcular $spmax(s,t)$ para todo par de cadeias $s, t \in C$, em $O(\|C\| + |C|^2)$, o que é suficiente para construir $G_k(C)$. Percebemos que, fazendo uma pequena modificação no algoritmo, era possível também determinar o conjunto $T(C)$, sem perder eficiência (ver detalhes no apêndice A.1).

□

Lema 3.4.1 *Dados um inteiro $k \geq 1$, um conjunto de cadeias C e o grafo $G = G_k(C)$, o conjunto $S = (s_1, s_2, \ldots, s_{|S|})$ é um k-caminho em C se, e somente se, o conjunto S é também um caminho em G.*

Prova:

Se $S = (s_1, s_2, \ldots, s_{|S|})$ é um caminho em G, para cada par de vértices consecutivos s_i, s_{i+1} em S, temos $(s_i, s_{i+1}) \in E(G)$. Portanto, temos também $|spmax(s_i, s_{i+1})| \geq k$. Como isto ocorre para todo par de vértices consecutivos em S, então S é também um k-caminho em C.

Por outro lado, se $S = (s_1, s_2, \ldots, s_{|S|})$ é um k-caminho em C, para cada par de cadeias consecutivas s_i, s_{i+1} em S, temos $|spmax(s_i, s_{i+1})| \geq k$. Portanto, temos também $(s_i, s_{i+1}) \in E(G)$. Como isto ocorre para todo par de cadeias consecutivas em S, então S é também um caminho em G.

□

Teorema 3.4.3 *Dados inteiros $n \geq 2$ e $k \geq 1$ e uma instância C de $MCP_n(k)$, seja $G = G_k(C)$ e $T = T(C)$ e considere a instância (G, T) para CVC. Dado um conjunto $H = \{S_1, S_2, \ldots, S_{|H|}\}$, dizemos que H é solução válida de CVC para (G, T) se, e somente se, o conjunto H é também solução válida de $MCP_n(k)$ para C.*

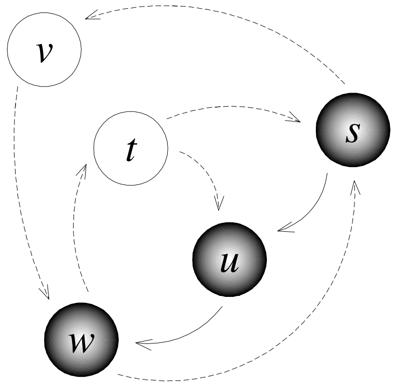
Prova:

Se H é solução válida de CVC para (G, T), sabemos que, para todo $1 \leq i \leq |H|$, o conjunto S_i é um caminho em G e, do lema 3.4.1, S_i é também um k-caminho em

C. Podemos afirmar que H cobre toda seqüência s em C, pois se $s \in T(C)$, então s pertence a algum caminho em H e, portanto, faz parte de um k-caminho em C. Se s não pertence a $T(C)$, existe $t \in C$ tal que s é subcadeia própria de t e, portanto, s é coberta pelo menos pelo k-caminho que cobre t. Logo, o conjunto H é solução válida de $MCP_n(k)$ para C.

Por outro lado, se H é solução válida de $MCP_n(k)$ para C, sabemos que, para todo $1 \leq i \leq |H|$, o conjunto S_i é um k-caminho em C e, do lema 3.4.1, S_i é também um caminho em G. Podemos afirmar que H cobre todo vértice v em $T(C)$, pois, por definição, o vértice v pertence a algum k-caminho de H e, portanto, faz parte de um caminho em G. Logo, o conjunto H é solução válida de CVC para (G, T).

□



s	`TTACGCTA`	v	`TAGAGAT`
t	`TTACGC`	w	`ATCGTT`
u	`GCTAGAGATC`		

Os vértices terminais, de fundo preto, são s, u e w. Neste caso, uma solução para o problema seria um único caminho orientado, formado pelas arestas (s, u) e (u, w), que resultaria no 2-caminho (s, u, w) cujo consenso é `TTACGCTAGAGATCGTT`.

Figura 3.5: Grafo de 2-sobreposição para o $MCP_4(2)$ da figura 3.1

Teorema 3.4.4 *Dados inteiros $n \geq 2$ e $k \geq 1$, e um conjunto de cadeias C, o custo de toda solução ótima de $MCP_n(k)$ para C é igual ao custo de toda solução ótima de CVC para $(G_k(C), T(C))$.*

Prova:

Consideraremos $G = G_k(C)$ e $T = T(C)$ e denotaremos por $OPT(G,T)$ o custo de toda solução ótima de CVC para (G,T). Denotaremos também por $OPT(C)$ o custo de toda solução ótima de $MCP_n(k)$ para C. Tomaremos ainda H_1, solução ótima de CVC para (G,T) (com $|H_1| = OPT(G,T)$) e H_2, solução ótima de $MCP_n(k)$ para C (com $|H_2| = OPT(C)$).

Do teorema 3.4.3, sabemos que se H_1 é solução válida de CVC para (G,T), então H_1 é solução válida de $MCP_n(k)$ para C. Portanto, $|H_1| = OPT(G,T) \geq OPT(C)$. Mas sabemos também que, se H_2 é solução válida de $MCP_n(k)$ para C, então H_2 é solução válida de CVC para (G,T), ou seja, $|H_2| = OPT(C) \geq OPT(G,T)$. Conseqüentemente, temos $OPT(C) = OPT(G,T)$.

$\square$

Teorema 3.4.5 *Dados inteiros $n \geq 2$ e $k \geq 1$, existe uma redução polinamial de $MCP_n(k)$ a CVC.*

Prova:

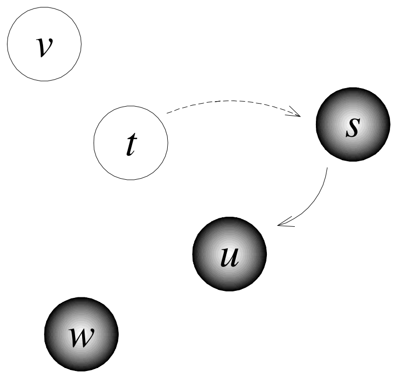
Dada uma instância C de $MCP_n(k)$, construímos em tempo polinomial o grafo $G_k(C)$ e o conjunto $T(C))$ (teorema 3.4.2), e tomamos $(G_k(C), T(C))$ como uma instância de CVC.

Consideraremos $OPT(G_k(C), T(C))$ o custo de toda solução ótima de CVC para $(G_k(C), T(C))$ e $OPT(C)$ o custo de toda solução ótima de $MCP_n(k)$ para C.

Se H é solução válida de CVC para $(G_k(C), T(C))$, com $|H| = OPT(G_k(C), T(C))$, então H é solução válida de $MCP_n(k)$ para C (teorema 3.4.3) e $|H| = OPT(C)$ (teorema 3.4.4).

$\square$

O CVC, se considerarmos somente as instâncias em que $T = V(G)$, é muito semelhante ao problema *Directed Hamiltonian Path Completion* ($DHPC$) [6], cujo objetivo, dado um grafo G, é determinar o menor conjunto E' tal que $E(G) \subseteq E'$ e exista um caminho hamiltoniano no grafo G', onde $V(G') = V(G)$ e $E(G') = E'$. Observe que, dado um grafo G, se h é o número de caminhos de uma solução ótima de CVC para $(G, V(G))$ e E' é solução de $DHPC$ para G, então $h = |E'| - |E(G)| + 1$.



s	TTACGCTA	v	TAGAGAT
t	TTACGC	w	ATCGTT
u	GCTAGAGATC		

Os vértices terminais, de fundo preto, são s, u e w. Neste caso, uma solução para o problema seria composta por dois caminhos orientados, um formado pela aresta (s, u) (resultando no 4-caminho (s, u), cujo consenso é `TTACGCTAGAGATC`) e outro formado apenas pelo vértice w (resultando no 4-caminho (w), cujo consenso é `ATCGTT`).

Figura 3.6: Grafo de 4-sobreposição para o $MCP_4(4)$ da figura 3.2

3.5 Supercadeia Comum Mínima

Um problema mais conhecido e que possui alguma semelhança com a família *MCP* é o problema da Supercadeia Comum Mínima (*SCM*) [5]. Este problema também foi estudado, e algumas técnicas utilizadas na determinação de sua complexidade foram adaptadas ao *MCP* e aproveitadas na demonstração de sua NP-dificuldade.

Uma **supercadeia** de um conjunto de cadeias $S = \{s_1, s_2, \ldots, s_n\}$ é uma cadeia s que contenha cada s_i como subcadeia. O problema da Supercadeia Comum Mínima consiste em, dado um conjunto de cadeias S, determinar uma supercadeia s para S, tal que $|s|$ seja mínimo (figura 3.7).

$S = \{\texttt{TTACGCTA}, \texttt{TTACGC}, \texttt{GCTAGAGATC}, \texttt{TAGAGAT}, \texttt{ATCGTT}\}$
$s' = \texttt{GCTAGAGATCGTTACGCTA}$ e $|s'| = 19$
$s = \texttt{TTACGCTAGAGATCGTT}$ e $|s| = 17$

Neste caso, o conjunto S é formado pelas mesmas cadeias da figura 3.1. A cadeia s' é uma supercadeia em S, enquanto que a cadeia s é a supercadeia mínima em S.

Figura 3.7: Supercadeia Comum Mínima

Gallant, Maier e Storer [5] reduziram o problema do caminho hamiltoniano em grafos orientados ao problema da Supercadeia Comum Mínima, comprovando que o último é NP-difícil, mesmo quando restrito a cadeias formadas sobre alfabetos pequenos (3 ou 4 letras).

3.6 Sumário

Neste capítulo fizemos uma definição detalhada da família de problemas denominada *Minimum Contig Problems* (*MCP*) [4], foco da nossa pesquisa. Devido às dificuldades geralmente encontradas na utilização prática da abordagem da montagem de fragmentos, na qual se baseia os nossos problemas, foram também definidas duas variações de *MCP*, que são as famílias $MCP\varepsilon$ e $MCPr$.

Além disso, foi mostrado que todo problema de *MCP* pode ser reduzido a um problema de grafos NP-difícil chamado *Cobertura de Vértices por Caminhos* (*CVC*).

Citamos também um outro problema utilizado nas análises realizadas no decorrer do projeto, que é a *Supercadeia Comum Mínima* (*SCM*) [5].

Capítulo 4

A complexidade da família MCP

Neste capítulo demonstraremos que os problemas que estudamos são NP-difíceis, utilizando o recurso da redução polinomial e denotaremos por $A \,\alpha\, B$ quando um problema A se reduzir polinomialmente a um problema B.

4.1 Resultados anteriores

Apesar de ter surgido no contexto de seqüências de DNA, a família *MCP* pode ser generalizada para seqüências em geral, sobre qualquer alfabeto. Com esta generalidade, Ferreira, Souza e Wakabayashi [4] mostraram que $MCP_n(k)$ é NP-difícil para todo $k \geq 1$ e $n = 3n' + 1$ caracteres, onde n' é o número de vértices terminais do grafo de k-sobreposição correspondente ao conjunto de seqüências dado. Nesta demonstração, eles partiram de um grafo G orientado, bipartido e de grau máximo igual a três e criaram um método para gerar um conjunto de cadeias C, tal que $G \simeq G_1(C)$ (aproveitando a propriedade de que grafos deste tipo possuem no máximo três emparelhamentos disjuntos em suas arestas). Eles mostraram, então, que a solução de $DMCP_n(1)$ para $(C, 1)$ seria pelo menos tão difícil quanto a determinação de um caminho hamiltoniano em G, mostrando assim que o $MCP_n(1)$ é NP-difícil.

Posteriormente, esta prova foi estendida para qualquer $k \geq 1$, mas o alfabeto utilizado ainda precisava ter pelo menos $n = 3n' + 1$ caracteres. Portanto, se o alfabeto fosse restrito às quatro letras `A`, `C`, `G` e `T`, que formam as seqüências de DNA, a demonstração dada por Ferreira, Souza e Wakabayashi [4] não garantia que qualquer problema da família *MCP* continuasse sendo NP-difícil. Isto nos motivou a determinar a complexidade dos problemas da família MCP, considerando alfabetos de quatro letras ou menores.

Empregando técnicas semelhantes às utilizadas por Gallant, Maier e Storer [5] para codificar cadeias no problema da Supercadeia Comum Mínima, conseguimos mostrar que $HAM\ \alpha\ MCP_n(k)$, para todo $n \geq 2$ e $k \geq 1$, provando que $MCP_n(k)$ é NP-difícil. Esta demonstração, que será descrita a seguir, vale para a família $MCP\varepsilon$, já que, como foi dito anteriormente, a família MCP é um caso especial desta com $\varepsilon = 0$. A partir deste resultado, conseguimos demonstrar também a NP-dificuldade da família $MCPr$, mostrando que $DMCP_4(k')\ \alpha\ DMCPr(k)$, para todo par de inteiros k, k', tais que $7(k'-1) \leq k \leq 7k'$.

4.2 $MCP_n(k)$ é NP-difícil para $n \geq 2$ e $k \geq 1$

Para demonstrar a NP-dificuldade dos problemas da família MCP, criamos uma notação especial para os alfabetos que iremos utilizar. Dado inteiro $m \geq 1$, chamamos de Σ_m o alfabeto $\{0, 1, 2, \ldots, m-1\}$, de modo que $|\Sigma_m| = m$.

Além disso, dados inteiros m e n, tais que $2 \leq n \leq m$, definimos a função injetora b_{mn} de modo que, para todo $c \in \Sigma_m$, $b_{mn}(c)$ é a representação de c na base n com $\lceil \log_n m \rceil$ caracteres.

A base do nosso raciocínio está nas definições de homomorfismos que preservam ocorrências (2.1.1.1) e grafos de $k-$sobreposição (2.2.1.1), sendo muito importante que o leitor esteja familiarizado com estes conceitos.

A nossa demonstração será dividida em cinco partes:

1. Mostraremos que, dado um grafo $G = (V, E)$, podemos construir, em tempo polinomial, um conjunto de cadeias C'', escrito sobre $\Sigma_{|V|+|E|}$, tal que $G \simeq G_1(C'')$ e $T(C'') = C''$;
2. dados inteiros $k \geq 1$ e $n \geq 2$, tomaremos um homomorfismo que preserva ocorrências $g : \Sigma^*_{|V|+|E|} \longrightarrow \Sigma^*_n$ e calcularemos k', tal que $(k'-1)|g| < k \leq k'|g|$;
3. tomaremos um homomorfismo que preserva ocorrências $f : \Sigma^*_{|V|+|E|} \longrightarrow \Sigma^*_{|V|+|E|}$, tal que $|f| = k'$ e criaremos, a partir de C'', o conjunto de cadeias $C' = f(C'')$, tal que $G_1(C'') \simeq G_{k'}(C')$ e $T(C') = f(T(C'')) = f(C'') = C'$;

4. tomaremos o homomorfismo g, do passo 2, e criaremos, a partir de C', o conjunto de cadeias $C = g(C')$, tal que $G_{k'}(C') \simeq G_k(C)$ e $T(C) = g(T(C')) = g(C') = C$;

5. resolveremos $DMCP_n(k)$ para $(C, 1)$. Se a resposta for **sim**, então saberemos que os grafos isomorfos $G_k(C)$, $G_{k'}(C')$, $G_1(C'')$ e G são hamiltonianos e se a resposta for **não**, então saberemos que estes grafos não são hamiltonianos.

Apesar desta divisão não ser totalmente explícita, iremos utilizá-la para separar os nossos teoremas e facilitar a leitura da nossa demonstração.

Parte 1:

Teorema 4.2.1 *Dado um grafo G, orientado, com $|V(G)| = n$ e $|E(G)| = m$, podemos construir, em tempo polinomial em n e m e exponencial em $\Delta(G)$, um conjunto de seqüências de caracteres C, escrito sobre o alfabeto Σ_{m+n}, tal que $G \simeq G_1(C)$, com $T(C) = C$ e, para toda cadeia $s \in C$, $|s| \leq 2^{\Delta(G)+2} - 1$.*

Prova:

Algoritmo 4.2.1 *Construção de C (figura 4.1):*

1. *Para cada $v \in V(G)$ faça:*
   1.1. *$pref(v) \leftarrow \varepsilon$;*
   1.2. *$suf(v) \leftarrow \varepsilon$;*
2. *Determine uma coloração de arestas $\mathcal{C} = \{M_1, M_2, \ldots, M_{|\mathcal{C}|}\}$ para G, tal que $|\mathcal{C}| = \Delta(G)$ ou $|\mathcal{C}| = \Delta(G) + 1$;*
3. *$I \leftarrow 0$;*
4. *Para cada $M_i \in \mathcal{C}$, faça:*
   4.1. *Para cada aresta $(u, v) \in M_i$ faça:*
      4.1.1. *$pref(v) \leftarrow pref(v).I.suf(u)$;*
      4.1.2. *$suf(u) \leftarrow pref(v)$;*
      4.1.3. *$I \leftarrow I + 1$;*
5. *$C \leftarrow \emptyset$;*

6. *Para cada* $v \in V(G)$*:*

    6.1. $C \leftarrow C \cup \{pref(v).I.suf(v)\}$*;*

    6.2. $I \leftarrow I + 1$*;*

7. *Retorne* C*;*

Para provar o nosso teorema será necessário mostrar que, ao término da execução do algoritmo, teremos:

(i) $T(C) = C$;

(ii) $G \simeq G_1(C)$;

(iii) $|s| \leq 2^{\Delta(G)+2} - 1$, para toda cadeia $s \in C$.

Precisamos mostrar também que:

(iv) a complexidade do algoritmo é polinomial em n e m e exponencial em $\Delta(G)$.

Denotaremos por $A(u, v)$ o instante correspondente ao processamento de uma dada aresta (u, v) no passo 4 do algoritmo e por $B(w)$ o instante correspondente ao processamento de um dado vértice w no passo 6 do algoritmo. Sabemos que, para toda $(u, v) \in E(G)$, temos $0 \leq A(u, v) \leq m - 1$ e que, se $(u, v) \neq (w, x)$, então $A(u, v) \neq A(w, x)$. Sabemos ainda que, para todo $w \in V(G)$, temos $m \leq B(w) \leq m + n - 1$ e que, se $w \neq v$, então $B(w) \neq B(v)$.

Definiremos também a função $f : V(G) \longrightarrow C$, tal que, dado $u \in V(G)$, $f(u) \in C$ é a cadeia correspondente ao vértice u. De acordo com o passo 6 do algoritmo, $f(u) = pref(u).B(u).suf(u)$.

(i) Primeiro vamos mostrar que $T(C) = C$.

Para todo par de vértices $u, v \in V(G)$, podemos ver que $B(u)$ só aparece em $f(u)$ e que $B(v)$ só aparece em $f(v)$, portanto $f(u)$ não é subcadeia própria de $f(v)$ e $f(v)$ não é subcadeia própria de $f(u)$.

(ii) Em seguida, vamos mostrar que a função f é um isomorfismo tal que $G \simeq G_1(C)$.

Podemos afirmar que f é uma bijeção, pois, para todo vértice $u \in V(G)$, o caracter $B(u)$ só aparece em $f(u)$.

Dado um instante $I = A(u, v)$, para todo vértice w em $V(G)$, definimos como $\mathit{preftmp}(w, I+1)$ a cadeia $\mathit{pref}(w)$ no instante I e $\mathit{suftmp}(w, I+1)$ a cadeia $\mathit{suf}(w)$ no instante I. Assim, se $w = v$, temos $\mathit{preftmp}(w, I+1) = \mathit{preftmp}(w, I).I.\mathit{suftmp}(u, I)$ e, senão, temos $\mathit{preftmp}(w, I+1) = \mathit{preftmp}(w, I)$. Do mesmo modo, se $w = u$, temos $\mathit{suftmp}(w, I+1) = \mathit{preftmp}(v, I).I.\mathit{suftmp}(w, I)$ e, senão, temos $\mathit{suftmp}(w, I+1) = \mathit{suftmp}(w, I)$. Além disso, para todo vértice w, as cadeias $\mathit{suftmp}(w, 0)$ e $\mathit{preftmp}(w, 0)$ são vazias.

Portanto, quando todas as m arestas tiverem sido processadas, teremos:

$$\mathit{preftmp}(w, m) = I_1.\mathit{suftmp}(u_1, I_1).I_2.\mathit{suftmp}(u_2, I_2) \ldots I_j.\mathit{suftmp}(u_j, I_j), e$$

$$\mathit{suftmp}(w, m) = \mathit{preftmp}(v_i, I'_i).I'_i \ldots \mathit{preftmp}(v_2, I'_2).I'_2.\mathit{preftmp}(v_1, I'_1).I'_1,$$

onde u_1 a u_j são os vértices de onde saem as arestas que entram em w, na ordem em que estas são processadas no passo 4 do algoritmo, de forma que $I_1 = A(u_1, w)$, $I_2 = A(u_2, w)$, ..., $I_j = A(u_j, w)$; do mesmo modo, denotamos por v_1 a v_i os vértices onde entram as arestas que saem de w, tal que $I'_1 = A(w, v_1)$, $I'_2 = A(w, v_2)$, ... $I'_i = A(w, v_i)$.

Assim, temos que $f(u) = \mathit{preftmp}(u, m).B(u).\mathit{suftmp}(u, m)$. Sabe-se que, se $u \neq v$, $B(u) \neq B(v)$. Portanto, uma sobreposição de l caracteres entre $f(u)$ e $f(v)$ é, na verdade, uma sobreposição de l caracteres entre $\mathit{suftmp}(u, m)$ e $\mathit{preftmp}(v, m)$.

- Primeiro vamos mostrar que, dada aresta (u, v) em $V(G)$, temos sobreposição maior ou igual a um entre $\mathit{suftmp}(u, m)$ e $\mathit{preftmp}(v, m)$:

Sabendo que um dado instante I corresponde ao processamento da aresta (u, v), sendo $I = A(u, v)$, temos $\mathit{suftmp}(u, I+1) = \mathit{preftmp}(v, I+1) = \mathit{preftmp}(v, I).I.\mathit{suftmp}(u, I)$. Portanto, há sobreposição maior ou igual a um entre $\mathit{suftmp}(u, I+1)$ e $\mathit{preftmp}(v, I+1)$.

Notamos que, para todo par de inteiros I', I'', tal que $1 \leq I' < I'' \leq m$, a cadeia $\mathit{preftmp}(v, I'+1)$ é prefixo de $\mathit{preftmp}(v, I''+1)$ e a cadeia $\mathit{suftmp}(u, I'+1)$ é sufixo de $\mathit{suftmp}(u, I''+1)$. Assim, podemos afirmar que $\mathit{suftmp}(u, I+1)$ é um sufixo de $\mathit{suftmp}(u, m)$ e $\mathit{preftmp}(v, I+1)$ é um prefixo de $\mathit{preftmp}(v, m)$, o que garante que a sobreposição estabelecida não foi alterada.

- Agora vamos mostrar que, dada sobreposição maior ou igual a um entre entre *suftmp* (u,m) e *preftmp* (v,m), existe a aresta (u,v) em $V(G)$.

Denotaremos por j o grau de entrada de v e por i o grau de saída de u, sendo v_1 a v_i os vértices onde entram as arestas que saem de u, na ordem em que estas são processadas, e u_1 a u_j os vértices de onde saem as arestas que entram em v, também na ordem em que estas são processadas:

Podemos observar que, em *suftmp* (u,m), o caracter $A(u,v_i)$ é único e corresponde ao maior número (é o instante mais recente deste intervalo); além disso, para um dado $1 \leq i' < i$, se considerarmos apenas os caracteres à direita de $A(u,v_{i'+1})$, veremos que o caracter $A(u,v_{i'})$ é único e corresponde ao maior número (é o instante mais recente do outro intervalo). Da mesma forma, em *preftmp* (v,m), o caracter $A(u_j,v)$ é único e corresponde ao maior número; além disso, para um dado $1 \leq j' < j$, se considerarmos apenas os caracteres à esquerda de $A(u_{j'+1},v)$, veremos que o caracter $A(u_{j'},v)$ é único e corresponde ao maior número.

Tomaremos *sufstr* (c,l) como uma função que, dada uma cadeia c, retorna o sufixo de c com l caracteres e *prefstr* (c,l) como uma função que retorna o prefixo de c com l caracteres.

Assim, dado $l \geq 1$, temos *sufstr* (*suftmp* $(u,m),l)$ = *prefstr* (*preftmp* $(v,m),l)$.

Mas *sufstr* (*suftmp* $(u,m),l) = ...A(u,v_{i'})......A(u,v_2)...A(u,v_1)$ (onde $i' \leq i$ e, para todo $i'' > i'$, o caracter $A(u,v_{i''})$ não aparece em *sufstr* (*suftmp* $(u,m),l)$) e *prefstr* (*preftmp* $(v,m),l) = A(u_1,v)...A(u_2,v)......A(u_{j'},v)...$ (onde $j' \leq j$ e, para todo $j'' > j'$, o caracter $A(u_{j''},v)$ não aparece em *sufstr* (*suftmp* $(u,m),l)$).

Sabemos então que $A(u,v_{i'})$ é único e é o maior caracter em *sufstr* (*suftmp* $(u,m),l)$ e $A(u_{j'},v)$ é único e é o maior caracter em *prefstr* (*preftmp* $(v,m),l)$. Logo $A(u,v_{i'}) = A(u_{j'},v)$, $u = u_{j'}$ e $v = v_{i'}$ e, portanto, existe a aresta (u,v) em $V(G)$.

Isto prova que $G \simeq G_1(C)$.

(iii) Precisamos provar ainda que, para toda cadeia $s \in C$, temos $|s| \leq 2^{\Delta(G)+2} - 1$.

Sabemos que foi determinada uma coloração de arestas $\mathcal{C} = \{M_1, M_2, \ldots, M_{|\mathcal{C}|}\}$ para $E(G)$.

Para todo $1 \leq i \leq |\mathcal{C}|$, dado vértice $v \in V(G)$, denotamos por $pref_i(v)$ e por $suf_i(v)$, respectivamente, as cadeias $pref(v)$ e $suf(v)$ no final do processamento das arestas de M_i.

Note que, para todo $1 \leq i \leq |\mathcal{C}|$, temos $\max_{v \in V(G)} |pref_i(v)| = \max_{v \in V(G)} |suf_i(v)|$ e, em razão deste fato, definimos $tam(i) = \max_{v \in V(G)} |pref_i(v)|$.

Inicialmente, temos $tam(0) = 0$, e, no final do processamento das arestas de M_i, temos $tam(i) = |pref_{i-1}(v).I.suf_{i-1}(u)|$, onde $|pref_{i-1}(v)| + |suf_{i-1}(u)| = \max_{(u,v) \in M_i}(|pref_{i-1}(v)| + |suf_{i-1}(u)|)$ e $I = A(u,v)$. Portanto, $tam(i) \leq 2.tam(i-1) + 1 \leq 2^i - 1$.

Ao término do processamento de todas as arestas, temos $tam(|\mathcal{C}|) \leq 2^{|\mathcal{C}|} - 1$. Sabemos que $|\mathcal{C}| = \Delta(G)$ ou $|\mathcal{C}| = \Delta(G) + 1$ e podemos dizer ainda que $tam(|\mathcal{C}|) \leq 2^{\Delta(G)+1} - 1$.

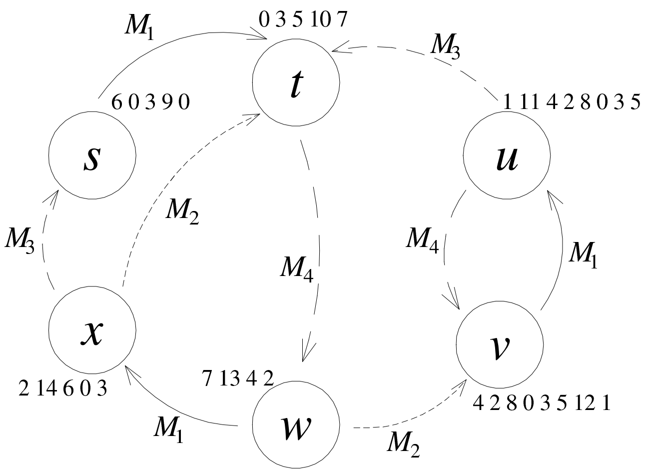
Portanto, para todo vértice $v \in V(G)$, temos $|f(v)| \leq 2.tam(|\mathcal{C}|) + |B(v)| \leq 2(2^{\Delta(G)+1} - 1) + 1 = 2^{\Delta(G)+2} - 1$.

(iv) Por fim, vamos determinar a complexidade do algoritmo.

Para o passo 2, aproveitamos o algoritmo de coloração proposto por Misra e Gries [9] em sua prova construtiva do teorema de Vizing [3, p.103]. Este passo possui complexidade $O(m^2 + m\Delta^2(G))$, como pode ser verificado no apêndice A.2.

Sabemos ainda que o passo 1 é $O(n)$, que o passo 6 é $O(2^{\Delta(G)}n)$ e que o passo 4 é $\sum_{i=1}^{|\mathcal{C}|} |M_i|.(2^{\Delta(G)+2} - 1) = O(2^{\Delta(G)}m)$, portanto a complexidade final é $O(2^{\Delta(G)}(m + n) + m^2)$.

□



$V(G) = 6$, $E(G) = 9$ e $\Delta(G) = 4$

$$\mathcal{C} = \{M_1, M_2, M_3, M_4\}$$
$$M_1 = \{(s,t), (v,u), (w,x)\},\ M_2 = \{(x,t), (w,v)\},$$
$$M_3 = \{(u,t), (x,s)\},\ M_4 = \{(t,w), (u,v)\}$$

$A(s,t) = 0$	$A(x,t) = 3$	$A(x,s) = 6$	$B(s) = 9$	$B(v) = 12$
$A(v,u) = 1$	$A(w,v) = 4$	$A(t,w) = 7$	$B(t) = 10$	$B(w) = 13$
$A(w,x) = 2$	$A(u,t) = 5$	$A(u,v) = 8$	$B(u) = 11$	$B(x) = 14$

Iterações do passo 4 do algoritmo:

Processamento de M_1:

(i) $suf(s) = pref(t) = 0$
(ii) $suf(v) = pref(u) = 1$
(iii) $suf(w) = pref(x) = 2$

Processamento de M_2:

(iv) $suf(x) = pref(t) = 0\ 3$
(v) $suf(w) = pref(v) = 4\ 2$

Processamento de M_3:

(vi) $suf(u) = pref(t) = 0\ 3\ 5$
(vii) $suf(x) = pref(s) = 6\ 0\ 3$

Processamento de M_4:

(viii) $suf(t) = pref(w) = 7$
(ix) $suf(u) = pref(v) = 4\ 2\ 8\ 0\ 3\ 5$

Figura 4.1: Execução do algoritmo 4.2.1

Parte 2:

Teorema 4.2.2 *Dados inteiros $m \geq n \geq 2$, um homomorfismo $g : \Sigma_m^* \longrightarrow \Sigma_n^*$ que preserva ocorrências, e um conjunto de seqüências de caracteres C', escrito sobre Σ_m, construímos $C = g(C')$, satisfazendo:*

1. *$T(C) = g(T(C'))$ e*
2. *para todo par de inteiros k e k', tais que $(k'-1)|g| < k \leq k'|g|$, $G_k(C) \simeq G_{k'}(C')$.*

Prova:

Como $C = g(C')$, podemos restringir o homomorfismo g para a bijeção $g : C' \longrightarrow C$.

(1) Como g preserva ocorrências, g preserva em C as mesmas relações de subcadeia que existiam em C'. Automaticamente, $T(g(C')) = g(T(C'))$, ou seja, g comuta com T. Portanto, $T(C) = g(T(C'))$.

(2) Tomaremos $u, v \in C'$ e sabemos que $g(u), g(v) \in C$. Por definição sabemos que $(u, v) \in E(G_{k'}(C'))$ implica em $|spmax(u, v)| \geq k'$. Como g preserva ocorrências, temos também $|spmax(g(u), g(v))| \geq k'|g|$, ou seja, para todo inteiro $k \leq k'|g|$, temos $(g(u), g(v)) \in E(G_k(C))$. Por outro lado, se $(u, v) \notin E(G_{k'}(C'))$, temos $|spmax(u, v)| \leq k' - 1$. Conseqüentemente, temos $|spmax(g(u), g(v))| \leq (k' - 1)|g|$, ou seja, para todo inteiro $k > (k' - 1)|g|$, temos $(g(u), g(v)) \notin E(G_k(C))$.

Portanto, para todo par de inteiros k e k', tais que $(k' - 1)|g| < k \leq k'|g|$, temos $G_k(C) \simeq G_{k'}(C')$.

□

Lema 4.2.1 *Dado um homomorfismo $g : \Sigma^* \longrightarrow \Upsilon^*$, tal que, para todo $a \in \Sigma$, $g(a)$ tem tamanho fixo $|g|$, se $g(t) = vw$ e $|v|$ é múltiplo de $|g|$, então $v \in Im(g)$.*

Prova:

Sabemos que, para um dado inteiro k, $|v| = k|g|$. Vamos tomar uma cadeia x, tal que x seja o prefixo de t com k caracteres. Assim, temos $|g(x)| = k|g|$. Mas tanto v quanto $g(x)$ são prefixos de $g(t)$ e, além disso, estes dois prefixos tem o mesmo tamanho. Portanto, $v = g(x)$, ou seja, $v \in \mathrm{Im}(g)$.

□

Lema 4.2.2 *Seja $g : \Sigma^* \longrightarrow \Upsilon^*$ um homomorfismo, tal que, para todo $c \in \Sigma$, $g(c)$ tem tamanho fixo $|g|$ e é definido como $g(c) = g'(c).y$, onde y é uma cadeia constante e $g'(c)$ é cadeia de tamanho fixo, de modo que não exista ocorrência de $g'(c)$ em y, nem de y em $g'(c)$. Se, para um dado $u \in \Sigma^*$, temos $g(u) = vw$ e y é sufixo de v, então $|v|$ é múltiplo de $|g|$.*

Prova:

Sabemos que $g(u) = g'(u_1).y.g'(u_2).y.g'(u_3).y \dots g'(u_{|u|}).y$ e que não existe ocorrência de $g'(u_i)$ em y, nem de y em $g'(u_i)$ para todo $1 \leq i \leq |u|$. Portanto, se y é sufixo de v, então existe $1 \leq j \leq |u|$ tal que $v = g'(u_1).y.g'(u_2).y \dots g'(u_j).y$, ou seja, $|v|$ é múltiplo de $|g|$.

□

Teorema 4.2.3 *Dado um homomorfismo $g : \Sigma^* \longrightarrow \Upsilon^*$, tal que, para todo $c \in \Sigma$, $g(c)$ tem tamanho fixo $|g|$ e é definido como $g(c) = g'(c).y$, onde y é uma cadeia constante e $g'(c)$ é cadeia de tamanho fixo, de modo que não exista ocorrência de $g'(c)$ em y, nem de y em $g'(c)$, podemos dizer que g preserva ocorrências.*

Prova:

Para mostrar que g preserva ocorrências, precisamos provar que (de 2.1.1.1):

- $|spmax(s,t)||g| = |spmax(g(s), g(t))|$ (ou $|spmax(s,t)||g| = |spmax(g(t), g(s))|$ [1])

[1] se g for homomorfismo reverso

Vamos supor que $s = uv$ e $t = vw$, com $v = spmax(s,t)$. Sabemos que $g(s) = g(u)g(v)$ e $g(t) = g(v)g(w)$ (ou $g(s) = g(v)g(u)$ e $g(t) = g(w)g(v)$ [1]). Portanto, existe sobreposição $g(v)$ entre $g(s)$ e $g(t)$ (ou entre $g(t)$ e $g(s)$ [1]). Logo, temos $|g(v)| = |v||g| = |spmax(s,t)||g| \leq |spmax(g(s), g(t))|$ (ou $|spmax(s,t)||g| \leq |spmax(g(t), g(s))|$ [1]).

Por outro lado, podemos supor $g(s) = uv$ e $g(t) = vw$, com $v = spmax(g(s), g(t))$ (ou $g(s) = vu$ e $g(t) = wv$, com $v = spmax(g(t), g(s))$ [1]). Por definição, podemos observar que, para todo $c \in \Sigma$, a cadeia $g'(c)$ tem tamanho fixo e y é uma cadeia constante, sendo que não existe ocorrência de $g'(c)$ em y, nem de y em $g'(c)$. Além disso, podemos afirmar seguramente que v termina em y, pois $g(s)$ (ou $g(t)$ [1]) termina em y. Portanto $|v|$ é múltiplo de $|g|$ (lema 4.2.2) e $v \in \mathrm{Im}(g)$ (lema 4.2.1). Assim, temos também que $u, w \in \mathrm{Im}(g)$, com $s = g^{-1}(u)g^{-1}(v)$ e $t = g^{-1}(v)g^{-1}(w)$. Portanto existe sobreposição $g^{-1}(v)$ entre s e t. Logo, temos $|g^{-1}(v)| = |v|/|g| = |spmax(g(s), g(t))|/|g| \leq |spmax(s,t)|$ (ou $|spmax(g(t), g(s))|/|g| \leq |spmax(s,t)|$ [1]).

Como $|spmax(s,t)||g| \leq |spmax(g(s), g(t))|$ e $|spmax(g(s), g(t))|/|g| \leq |spmax(s,t)|$, temos $|spmax(s,t)||g| = |spmax(g(s), g(t))|$ (ou $|spmax(s,t)||g| = |spmax(g(t), g(s))|$ [1]).

- t é subcadeia própria de s se, e somente se, $g(t)$ é subcadeia própria de $g(s)$.

Vamos agora supor que $s = utv$. Sabemos que $g(s) = g(u)g(t)g(v)$ (ou $g(s) = g(v)g(t)g(u)$ [1]) e, portanto, podemos dizer que $g(t)$ é subcadeia de $g(s)$. Do mesmo modo, vamos supor que $g(s) = u.g(t).v$. Podemos afirmar seguramente que v termina em y, pois $g(s)$ termina em y. Além disso, o prefixo de $g(s)$ que precede $g(t)$ também termina em y, já que $g(t)$ começa com $g'(t_1)$ e sabemos ainda que $g(s) = g'(s_1).y.g'(s_2).y \ldots g'(s_{|s|}).y$ (ou $g(s) = g'(s_{|s|}).y.g'(s_{|s|-1}).y \ldots g'(s_1).y$ [1]), sendo que, para todo $c \in \Sigma_m$, não existe ocorrência de $g'(c)$ em y, nem de y em $g'(c)$. Portanto, podemos dizer que u também termina em y. Com isso, sabemos que $|u|$ e $|v|$ são múltiplos de $|g|$ (lema 4.2.2) e temos $u, v \in \mathrm{Im}(g)$ (lema 4.2.1). Assim, temos $s = g^{-1}(u)g^{-1}(g(t))g^{-1}(v) = g^{-1}(u).t.g^{-1}(v)$ (ou $s = g^{-1}(v)g^{-1}(g(t))g^{-1}(u) = g^{-1}(v).t.g^{-1}(u)$ [1]). Portanto, podemos dizer que t é subcadeia de s.

□

[1] se g for homomorfismo reverso

Teorema 4.2.4 *Para todo par de inteiros m, n tais que $m \geq n \geq 2$, existe um homomorfismo $g : \Sigma_m^* \longrightarrow \Sigma_n^*$ que preserva ocorrências.*

Prova:

Tomaremos inteiros m, n, com $m \geq n$, e consideraremos dois casos, um no qual $n \geq 3$, e outro no qual $n = 2$.

1. No primeiro caso, para qualquer par de inteiros m, n, com $m \geq n \geq 3$, tomamos $p = n - 1$. Para todo $c \in \Sigma_m$, definimos $g(c) = b_{mp}(c).p$, de modo que $|g| = \lceil \log_p m \rceil + 1$. Como b_{mp} é injetora, então g também é injetora, pois p é uma cadeia constante.

   Podemos observar que, para todo $c \in \Sigma_m$, a cadeia $b_{mp}(c)$ tem tamanho fixo e p é uma cadeia constante, sendo que não existe ocorrência de $b_{mp}(c)$ em p, nem de p em $b_{mp}(c)$.

   Portanto, pelo teorema 4.2.3, o homomorfismo g preserva ocorrências.

2. No segundo caso, para qualquer $m \geq 2$, definimos $g(c) = 1.b_{m2}(c).1.0^{i+1}$, para todo $c \in \Sigma_m$, tal que $i = \log_2 m$ e $|g| = 2i + 3$. Como b_{m2} é injetora, então g também é injetora, pois as cadeias 1 e 1.0^{i+1} são constantes.

   Podemos observar que, para todo $c \in \Sigma_m$, a cadeia $1.b_{m2}(c).1$ tem tamanho fixo e 0^{i+1} é uma cadeia constante, sendo que não existe ocorrência de $1.b_{m2}(c).1$ em 0^{i+1}, nem de 0^{i+1} em $1.b_{m2}(c).1$.

   Portanto, também pelo teorema 4.2.3, o homomorfismo g preserva ocorrências.

□

Na figura 4.2 apresentamos um exemplo de homomorfismo que preserva ocorrências.

Parte 3:

Dados um inteiro $k \geq 1$ e um alfabeto Σ definimos o homomorfismo $f_k : \Sigma^* \longrightarrow \Sigma^*$, tal que $f_k(c) = c^k$ para todo $c \in \Sigma$.

$$\Sigma_{15} = \{0, 1, 2, \ldots, 14\};\ \Sigma_5 = \{0, 1, 2, 3, 4\}$$

$$g : \Sigma_{15}^* \longrightarrow \Sigma_5^* \text{ é tal que}$$
$$g(a) = b_{15\,4}(a).4, \text{ para todo } a \in \Sigma_{15}$$

$$|b_{15\,4}| = \lceil \log_4 15 \rceil = 2 \text{ e } |g| = 3$$

$g(0) = 0\,0\,4$	$g(5) = 1\,1\,4$	$g(10) = 2\,2\,4$
$g(1) = 0\,1\,4$	$g(6) = 1\,2\,4$	$g(11) = 2\,3\,4$
$g(2) = 0\,2\,4$	$g(7) = 1\,3\,4$	$g(12) = 3\,0\,4$
$g(3) = 0\,3\,4$	$g(8) = 2\,0\,4$	$g(13) = 3\,1\,4$
$g(4) = 1\,0\,4$	$g(9) = 2\,1\,4$	$g(14) = 3\,2\,4$

Figura 4.2: Homomorfimo $g : \Sigma_{15}^* \longrightarrow \Sigma_5^*$ que preserva ocorrências

Teorema 4.2.5 *Dado um alfabeto Σ, o homomorfismo $f_k : \Sigma^* \longrightarrow \Sigma^*$ preserva ocorrências, para todo $k \geq 1$.*

Prova:

Tomando $k \geq 1$, sabemos que f_k é injetora, com $|f_k| = k$.

Consideraremos cadeias $s, t \in \Sigma^*$ tal que $s = c_1 c_2 \ldots c_{|s|}$ e $t = d_1 d_2 \ldots d_{|t|}$. Assim, temos $f_k(s) = f_k(c_1) f_k(c_2) \ldots f_k(c_{|s|})$ e $f_k(t) = f_k(d_1) f_k(d_2) \ldots f_k(d_{|t|})$.

Para mostrar que f_k preserva ocorrências, precisamos provar que (de 2.1.1.1):

- $|spmax(s,t)||f_k| = |spmax(f_k(s), f_k(t))|$

Vamos supor que temos $s = uv$ e $t = vw$, com $v = spmax(s,t)$. Podemos então dizer que $f_k(s) = f_k(u) f_k(v)$ e $f_k(t) = f_k(v) f_k(w)$. Portanto, existe sobreposição $f_k(v)$ entre $f_k(s)$ e $f_k(t)$. Logo, temos $|f_k(v)| = |v||f_k| = |spmax(s,t)||f_k| \leq |spmax(f_k(s), f_k(t))|$.

Por outro lado, podemos supor $f_k(s) = uv$ e $f_k(t) = vw$, com $v = spmax(f_k(s), f_k(t))$. Supondo ainda que $u, v, w \notin \mathrm{Im}(f_k)$, teríamos a cadeia $u = c_1^k c_2^k \ldots c_{i-1}^k c_i^j$, a cadeia $v = c_i^{k-j} c_{i+1}^k c_{i+2}^k \ldots c_{|s|}^k = d_1^k d_2^k \ldots d_{l-1}^k d_l^m$ e a cadeia $w = d_l^{k-m} d_{l+1}^k d_{l+2}^k \ldots d_{|t|}^k$, onde consideramos $1 \leq j < k$ e $1 \leq i < |s|$, e, do mesmo modo, consideramos $1 \leq m < k$ e $1 \leq l < |t|$.

Assim, teríamos obrigatoriamente $l - 1 = |s| - (i+1) + 1 = |s| - i$ e $m = k-(j+1)+1 = k-j$, de modo que $v = c_i^m c_{i+1}^k c_{i+2}^k \dots c_{|s|}^k = d_1^k d_2^k \dots d_{|s|-i}^k d_{|s|-i+1}^m$ e, portanto, teríamos $c_i = d_1 = c_{i+1} = d_2 = c_{i+2} = \dots = d_{|s|-i} = c_{|s|} = d_{|s|-i+1}$. Mas, neste caso, poderíamos tomar a cadeia $u' = c_1^k c_2^k \dots c_{i-1}^k$, a cadeia $v' = c_i^k c_{i+1}^k c_{i+2}^k \dots c_{|s|}^k = d_1^k d_2^k \dots d_{l-1}^k d_l^k$ e a cadeia $w' = d_{l+1}^k d_{l+2}^k \dots d_{|t|}^k$, tal que $|v'| > |v|$, com $f_k(s) = u'v'$ e $f_k(t) = v'w'$, o que seria uma contradição.

Portanto, $u, v, w \in \mathrm{Im}(f_k)$, com $s = f_k^{-1}(u) f_k^{-1}(v)$ e $t = f_k^{-1}(v) f_k^{-1}(w)$, ou seja, existe sobreposição $f_k^{-1}(v)$ entre s e t. Logo, temos $|f_k^{-1}(v)| = |v|/|f_k| = |spmax(f_k(s), f_k(t))|/|f_k| \leq |spmax(s,t)|$.

Como $|spmax(s,t)||f_k| \leq |spmax(f_k(s), f_k(t))|$ e $|spmax(f_k(s), f_k(t))|/|f_k| \leq |spmax(s,t)|$, temos $|spmax(s,t)||f_k| = |spmax(f_k(s), f_k(t))|$.

- t é subcadeia própria de s se, e somente se, $f_k(t)$ é subcadeia própria de $f_k(s)$.

Vamos agora supor que $s = utv$. Sabemos que $f_k(s) = f_k(u) f_k(t) f_k(v)$ e, portanto, podemos dizer que $f_k(t)$ é subcadeia de $f_k(s)$.

Do mesmo modo, vamos supor que $f_k(s) = u.f_k(t).v$. Supondo ainda que $u, v \notin \mathrm{Im}(f_k)$, teríamos $u = c_1^k c_2^k \dots c_i^k c_{i+1}^j$ e $v = c_{l-1}^{k-j} c_l^k c_{l+1}^k \dots c_{|s|}^k$, onde consideramos $1 \leq j < k$ e $i < l$, com $i + |t| + l + 1 = |s|$.

Assim, teríamos $f_k(s) = c_1^k c_2^k \dots c_i^k c_{i+1}^j . d_1^k d_2^k \dots d_{|t|}^k c_{l-1}^{k-j} c_l^k c_{l+1}^k \dots c_{|s|}^k$ e, portanto, teríamos $c_{i+1} = d_1 = d_2 = \dots = d_{|t|} = c_{l-1} = c$. Mas, neste caso, poderíamos tomar a cadeia $u' = c_1^k c_2^k \dots c_i^k$, a cadeia $x = c^k$ e a cadeia $v' = c_l^k c_{l+1}^k \dots c_{|s|}^k$, tal que $f_k(s) = u' f_k(t) x v' = u' x f_k(t) v'$. Portanto, sempre podemos tomar $u, v \in \mathrm{Im}(f_k)$, com $s = f_k^{-1}(u).t.f_k^{-1}(v)$. Em outras palavras, podemos dizer que t é subcadeia de s.

□

Partes 4 e 5:

Teorema 4.2.6 *Para todo $n \geq 2$ e todo $k \geq 1$, $MCP_n(k)$ é NP-difícil.*

Prova:

Dado um inteiro positivo d, chamaremos de HAM_d o problema do caminho hamiltoniano cujas instâncias são grafos nos quais d é o grau máximo permitido. Sabemos que, para todo $d \geq 3$, o problema HAM_d é NP-completo [6]. Na nossa demonstração mostraremos que, dados inteiros $d \geq 3$, $n \geq 2$ e $k \geq 1$, temos $HAM_d \leq DMCP_n(k)$. Os passos desta redução podem ser acompanhados na figura 4.3.

Dados um inteiro $d \geq 3$ e um grafo orientado G, com $\Delta(G) \leq d$, construímos um conjunto de seqüências de caracteres C'', escrito sobre o alfabeto Σ_m, onde $m = |V(G)| + |E(G)|$, tal que $T(C'') = C''$ e $G \simeq G_1(C'')$. Além disso, para toda $s \in C$, temos $|s| \leq 2^{d+2} - 1$ (Teorema 4.2.1).

Sabemos que existe um homomorfismo $g : \Sigma_m^* \longrightarrow \Sigma_n^*$ que preserva ocorrências (teorema 4.2.4) e tomamos inteiro k', tal que $(k'-1)|g| < k \leq k'|g|$.

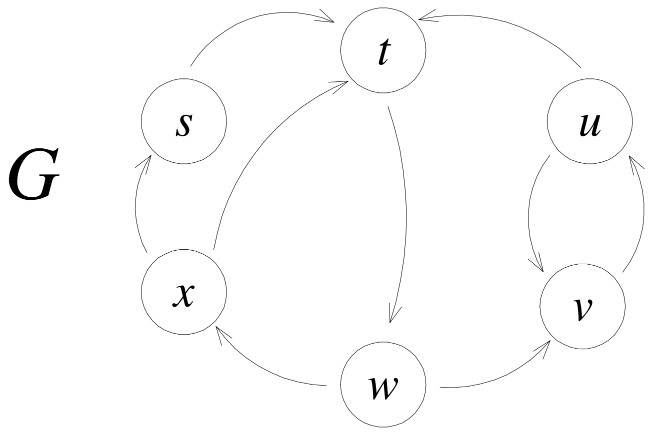
Sabemos que existe também um homomorfismo $f_{k'} : \Sigma_m^* \longrightarrow \Sigma_m^*$ que preserva ocorrências. (Teorema 4.2.5) e construímos $C' = f_{k'}(C'')$. Note que $G_1(C'') \simeq G_{k'}(C')$ e $T(C') = f_k(T(C'')) = f_k(C'') = C'$ (teorema 4.2.2).

Usamos então g para construir um conjunto de seqüências de caracteres $C = g(C')$, escrito sobre Σ_n, e, novamente do teorema 4.2.2, temos $G_{k'}(C') \simeq G_k(C)$ e $T(C) = g(T(C')) = g(C') = C$.

Assim, supondo que D é solução de $DMCP_n(k)$ para $(C, 1)$, então D é solução de HAM_d para $(G_k(C), 1)$, de HAM_d para $(G_{k'}(C'), 1)$, de HAM_d para $(G_1(C''), 1)$ e de HAM_d para $(G, 1)$ (teoremas 3.4.4). Mas se $D =$ **sim**, sabemos que existe um único caminho que cobre todos os vértices de G (G é hamiltoniano) e se $D =$ **não**, sabemos que não é possível cobrir todos os vértices de G com apenas um caminho (G não é hamiltoniano).

Esta redução mostra que, para todo $n \geq 2$ e todo $k \geq 1$, $MCP_n(k)$ é NP-difícil.

$\square$



Queremos reduzir HAM a $MCP_5(4)$ ($k = 4$), partindo do grafo G, ou seja, queremos construir conjunto de cadeias C, escrita sobre $\Sigma_5 = \{0, 1, 2, 3, 4\}$, tal que $G \simeq G_4(C)$

Usamos o algoritmo 4.2.1 para construir um conjunto de cadeias C'' a partir de G e $f : V(G) \longrightarrow C''$ é uma bijeção que relaciona os vértices do grafo às cadeias de C''

$C'' = \{f(y) \mid y \in V(G)\}$; pelo algoritmo 4.2.1, como $|V(G)| + |E(G)| = 15$, as cadeias de C'' são escritas sobre $\Sigma_{15} = \{0, 1, 2, \ldots, 14\}$ (ver figura 4.1)

$f(s) = 6\ 0\ 3\ 9\ 0$ $\quad$ $f(u) = 1\ 11\ 4\ 2\ 8\ 0\ 3\ 5$ $\quad$ $f(w) = 7\ 13\ 4\ 2$
$f(t) = 0\ 3\ 5\ 10\ 7$ $\quad$ $f(v) = 4\ 2\ 8\ 0\ 3\ 5\ 12\ 1$ $\quad$ $f(x) = 2\ 14\ 6\ 0\ 3$

$g : \Sigma_{15}^* \longrightarrow \Sigma_5^*$ é tal como foi definido na figura 4.2, com $|g| = 3$

Calculamos o valor $k' = \lceil k/|g| \rceil = \lceil 4/3 \rceil = 2$.

$f_2 : \Sigma_{15}^* \longrightarrow \Sigma_{15}^*$ é tal que $f_2(a) = aa$, para todo $a \in \Sigma_{15}$, com $|f_2| = 2$

$$C' = \{f_2(z) \mid z \in C''\}$$

$f_2(f(s)) = 6\ 6\ 0\ 0\ 3\ 3\ 9\ 9\ 0\ 0$ $\quad$ $f_2(f(v)) = 4\ 4\ 2\ 2\ 8\ 8\ 0\ 0\ 3\ 3\ 5\ 5\ 12\ 12\ 1\ 1$
$f_2(f(t)) = 0\ 0\ 3\ 3\ 5\ 5\ 10\ 10\ 7\ 7$ $\quad$ $f_2(f(w)) = 7\ 7\ 13\ 13\ 4\ 4\ 2\ 2$
$f_2(f(u)) = 1\ 1\ 11\ 11\ 4\ 4\ 2\ 2\ 8\ 8\ 0\ 0\ 3\ 3\ 5\ 5$ $\quad$ $f_2(f(x)) = 2\ 2\ 14\ 14\ 6\ 6\ 0\ 0\ 3\ 3$

$$C = \{g(z) \mid z \in C'\}$$

$g(f_2(f(s))) = 1\ 2\ 4\ 1\ 2\ 4\ 0\ 0\ 4\ 0\ 0\ 4\ 0\ 3\ 4\ 0\ 3\ 4\ 2\ 1\ 4\ 2\ 1\ 4\ 0\ 0\ 4\ 0\ 0\ 4$
$g(f_2(f(t))) = 0\ 0\ 4\ 0\ 0\ 4\ 0\ 3\ 4\ 0\ 3\ 4\ 1\ 1\ 4\ 1\ 1\ 4\ 2\ 2\ 4\ 2\ 2\ 4\ 1\ 3\ 4\ 1\ 3\ 4$
$g(f_2(f(u))) = 0\ 1\ 4\ 0\ 1\ 4\ 2\ 3\ 4\ 2\ 3\ 4\ 1\ 0\ 4\ 1\ 0\ 4\ 0\ 2\ 4\ 0\ 2\ 4\ 2\ 0\ 4\ 2\ 0\ 4\ 0\ 0\ 4\ 0\ 0\ 4\ 0\ 3\ 4\ 0\ 3\ 4\ 1\ 1\ 4\ 1\ 1\ 4$
$g(f_2(f(v))) = 1\ 0\ 4\ 1\ 0\ 4\ 0\ 2\ 4\ 0\ 2\ 4\ 2\ 0\ 4\ 2\ 0\ 4\ 0\ 0\ 4\ 0\ 0\ 4\ 0\ 3\ 4\ 0\ 3\ 4\ 1\ 1\ 4\ 1\ 1\ 4\ 3\ 0\ 4\ 3\ 0\ 4\ 0\ 1\ 4\ 0\ 1\ 4$
$g(f_2(f(w))) = 1\ 3\ 4\ 1\ 3\ 4\ 3\ 1\ 4\ 3\ 1\ 4\ 1\ 0\ 4\ 1\ 0\ 4\ 0\ 2\ 4\ 0\ 2\ 4$
$g(f_2(f(x))) = 0\ 2\ 4\ 0\ 2\ 4\ 3\ 2\ 4\ 3\ 2\ 4\ 1\ 2\ 4\ 1\ 2\ 4\ 0\ 0\ 4\ 0\ 0\ 4\ 0\ 3\ 4\ 0\ 3\ 4$

$$G \simeq G_1(C'') \simeq G_2(C') \simeq G_4(C)$$

Figura 4.3: Redução de HAM a $MCP_5(4)$, utilizando o grafo da figura 4.1

4.3 $MCPr(k)$ é NP-difícil para todo $k \geq 1$

Como se sabe, a família *MCPr* está definida apenas para o alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, que denotamos aqui por Σ_{dna}.

Definimos ainda o homomorfismo $\varrho : \Sigma_{dna}^* \longrightarrow \Sigma_{dna}^*$ e o homomorfismo reverso $\overline{\varrho} : \Sigma_{dna}^* \longrightarrow \Sigma_{dna}^*$, tais que, para todo $c \in \Sigma_{dna}$, $\varrho(c) = \mathtt{AAG}c\mathtt{TCC}$ e $\overline{\varrho}(c) = \mathtt{GGA}\overline{c}\mathtt{CTT}$.

Chamaremos de $DMCPr(k)$ a versão de decisão de $MCPr(k)$, de modo que, dado um inteiro $i \geq 1$ e um conjunto de cadeias C, escrito sobre Σ_{dna}, o problema $DMCPr(k)$ determina se C pode ser coberto por no máximo i k-caminhos disjuntos em $C \cup \overline{C}$.

Para demonstrar a NP-dificuldade dos problemas da família *MCPr*, utilizaremos os homomorfismos ϱ e $\overline{\varrho}$ para fazer $DMCP_4(k') \leq DMCPr(k)$, para todo par de inteiros k, k', com $7(k'-1) < k \leq 7k'$.

Teorema 4.3.1 *Os homomorfismos ϱ e $\overline{\varrho}$ preservam ocorrências.*

Prova:

Para todo $c \in \Sigma_{dna}$, temos $\varrho(c) = \mathtt{AAG}c\mathtt{TCC}$, onde $\mathtt{AAG}c\mathtt{T}$ é uma cadeia de tamanho fixo e $\mathtt{CC}$ é uma cadeia constante, de modo que não existe ocorrência de $\mathtt{AAG}c\mathtt{T}$ em $\mathtt{CC}$, nem de $\mathtt{CC}$ em $\mathtt{AAG}c\mathtt{T}$.

Da mesma forma, para todo $c \in \Sigma_{dna}$, temos $\overline{\varrho}(c) = \mathtt{GGA}\overline{c}\mathtt{CTT}$, onde $\mathtt{GGA}\overline{c}\mathtt{C}$ é uma cadeia de tamanho fixo e $\mathtt{TT}$ é uma cadeia constante, de modo que não existe ocorrência de $\mathtt{GGA}\overline{c}\mathtt{C}$ em $\mathtt{TT}$, nem de $\mathtt{TT}$ em $\mathtt{GGA}\overline{c}\mathtt{C}$.

Portanto, pelo teorema 4.2.3, os homomorfismos ϱ e $\overline{\varrho}$ preservam ocorrências.

□

Lema 4.3.1 *Dada qualquer cadeia $s \in \Sigma_{dna}^*$, $\overline{\varrho(s)} = \overline{\varrho}(s)$.*

Prova:

Supondo $s = c_1c_2c_3\ldots c_{|s|}$, temos:

1. $\varrho(s) = \mathtt{AAG}c_1\mathtt{TCCAAG}c_2\mathtt{TCCAAG}c_3\mathtt{TCC}\ldots\mathtt{AAG}c_{|s|}\mathtt{TCC}$ e
   $\overline{\varrho(s)} = \mathtt{GGA}\overline{c_{|s|}}\mathtt{CTTGGA}\overline{c_{|s|-1}}\mathtt{CTT}\ldots\mathtt{GGA}\overline{c_2}\mathtt{CTTGGA}\overline{c_1}\mathtt{CTT}$.
2. $\overline{\varrho}(s) = \overline{\varrho}(c_{|s|}).\overline{\varrho}(c_{|s|-1})\ldots\overline{\varrho}(c_2).\overline{\varrho}(c_1)$, ou seja,
   $\overline{\varrho}(s) = \mathtt{GGA}\overline{c_{|s|}}\mathtt{CTTGGA}\overline{c_{|s|-1}}\mathtt{CTT}\ldots\mathtt{GGA}\overline{c_2}\mathtt{CTTGGA}\overline{c_1}\mathtt{CTT}$.

Portanto, $\overline{\varrho(s)} = \overline{\varrho}(s)$.

□

Lema 4.3.2 *Dadas quaisquer cadeias* $s, t \in \Sigma^*_{dna}$, *nunca haverá ocorrência de* $\varrho(s)$ *em* $\overline{\varrho}(t)$ *ou de* $\overline{\varrho}(t)$ *em* $\varrho(s)$.

Prova:

Supondo $s = c_1c_2 \ldots c_{|s|}$ e $t = d_1d_2 \ldots d_{|t|}$ e tomando caracteres c e d tais que, para quaisquer inteiros i e j, com $1 \leq i \leq |s|$ e $1 \leq j \leq |t|$, temos $c = c_i$ e $d = d_j$, se examinarmos todos os possíveis casos, teremos:

$$\begin{array}{llll}
\texttt{......AAG}c\texttt{TCC} & \texttt{.....AAG}c\texttt{TCC.} & \texttt{.....AAG}c\texttt{TCC.} & \texttt{....AAG}c\texttt{TCC..} \\
\texttt{GGA}\overline{d}\texttt{CTT......} & \texttt{GGA}\overline{d}\texttt{CTT......} & \texttt{.GGA}\overline{d}\texttt{CTT.....} & \texttt{.GGA}\overline{d}\texttt{CTT.....} \\
 & & & \\
\texttt{....AAG}c\texttt{TCC..} & \texttt{...AAG}c\texttt{TCC...} & \texttt{...AAG}c\texttt{TCC...} & \texttt{..AAG}c\texttt{TCC....} \\
\texttt{..GGA}\overline{d}\texttt{CTT....} & \texttt{..GGA}\overline{d}\texttt{CTT....} & \texttt{...GGA}\overline{d}\texttt{CTT...} & \texttt{...GGA}\overline{d}\texttt{CTT...} \\
 & & & \\
\texttt{..AAG}c\texttt{TCC....} & \texttt{.AAG}c\texttt{TCC.....} & \texttt{.AAG}c\texttt{TCC.....} & \texttt{AAG}c\texttt{TCC......} \\
\texttt{....GGA}\overline{d}\texttt{CTT..} & \texttt{....GGA}\overline{d}\texttt{CTT..} & \texttt{.....GGA}\overline{d}\texttt{CTT.} & \texttt{.....GGA}\overline{d}\texttt{CTT.} \\
 & & & \\
\texttt{AAG}c\texttt{TCC......} & & & \\
\texttt{......GGA}\overline{d}\texttt{CTT} & & &
\end{array}$$

Portanto, é impossível haver ocorrência de $\varrho(c)$ em $\overline{\varrho}(d)$ ou de $\overline{\varrho}(d)$ em $\varrho(c)$ e, conseqüentemente, é impossível haver ocorrência de $\varrho(s)$ em $\overline{\varrho}(t)$ ou de $\overline{\varrho}(t)$ em $\varrho(s)$.

□

Lema 4.3.3 *Dadas quaisquer cadeias* $s, t \in \Sigma^*_{dna}$, *podemos dizer que* $|spmax(\varrho(s), \varrho(t))| = |spmax(\overline{\varrho}(t), \overline{\varrho}(s))|$.

Prova:

Sabemos que ϱ e $\overline{\varrho}$ preservam ocorrências. Da definição 2.1.1.1, sabemos que $|spmax(s,t)||\varrho| = |spmax(\varrho(s), \varrho(t))|$ e que $|spmax(s,t)||\overline{\varrho}| = |spmax(\overline{\varrho}(t), \overline{\varrho}(s))|$. Como $|\varrho| = |\overline{\varrho}|$, temos $|spmax(\varrho(s), \varrho(t))| = |spmax(\overline{\varrho}(t), \overline{\varrho}(s))|$.

□

Teorema 4.3.2 *Dados inteiros $k \geq 1$ e $k' \geq 1$ e conjuntos de cadeias C e C', tais que exista um isomorfismo $f : C \longrightarrow C'$, pelo qual $G_k(C) \simeq G_{k'}(C')$ e $T(C') = f(T(C))$, o custo de toda solução ótima de CVC para a instância $(G_k(C), T(C))$ é igual ao custo de toda solução ótima de CVC para a instância $(G_{k'}(C'), T(C'))$.*

Prova:

Consideraremos $OPT(G, T)$ o custo de toda solução ótima de CVC para a instância (G, T).

Vamos supor que $H = \{S_1, S_2, \ldots, S_{|H|}\}$ seja solução ótima de CVC para $(G_k(C), T(C))$. Sabemos que, para todo $1 \leq i \leq |H|$, o conjunto S_i é um caminho em $G_K(C)$ e, portanto, o conjunto $f(S_i)$ é um caminho em $G_{k'}(C')$. Podemos então afirmar que $H' = \{f(S_1), f(S_2), \ldots, f(S_{|H|})\}$ cobre todo vértice v em $T(C')$, pois $f^{-1}(v) \in T(C)$ e, sendo assim, temos $f^1(v) \in S$, para algum $S \in H$. Conseqüentemente, temos $v \in f(S)$ e $f(S) \in H'$. Logo, o conjunto H' é solução válida de CVC para $(G_{k'}(C'), T(C'))$, ou seja, temos $|H'| = OPT(G_k(C), T(C)) \geq OPT(G_{k'}(C'), T(C'))$.

Vamos agora supor que $K' = \{R_1, R_2, \ldots, R_{|K'|}\}$ seja solução ótima de CVC para $(G_{k'}(C'), T(C'))$. Sabemos que, para todo $1 \leq i \leq |K'|$, o conjunto R_i é um caminho em $G_{K'}(C')$ e, portanto, o conjunto $f^{-1}(R_i)$ é um caminho em $G_k(C)$. Podemos então afirmar que $K = \{f^{-1}(R_1), f^{-1}(R_2), \ldots, f^{-1}(R_{|K'|})\}$ cobre todo vértice v em $T(C)$, pois $f(v) \in T(C')$ e, sendo assim, temos $f(v) \in R$, para algum $R \in K'$. Conseqüentemente, temos $v \in f^{-1}(R)$ e $f^{-1}(R) \in K$. Logo, o conjunto K é solução válida de CVC para $(G_k(C), T(C))$, ou seja, temos $|K| = OPT(G_{k'}(C'), T(C')) \geq OPT(G_k(C), T(C))$.

Mas se $OPT(G_k(C), T(C)) \geq OPT(G_{k'}(C'), T(C'))$ e $OPT(G_{k'}(C'), T(C')) \geq OPT(G_k(C), T(C))$, então $OPT(G_k(C), T(C)) = OPT(G_{k'}(C'), T(C'))$.

□

Teorema 4.3.3 *Dado um homomorfismo $f : \Sigma^* \longrightarrow \Upsilon^*$ que preserva ocorrências e um conjunto de cadeias C', escrito sobre Σ, para todo par de inteiros k, k', tais que $(k'-1)|f| < k \leq k'|f|$, se tomarmos $C = f(C')$, podemos dizer que o custo de toda solução ótima de $MCP_{|\Upsilon|}(k)$ para C é igual ao custo de toda solução ótima de $MCP_{|\Sigma|}(k')$ para C'.*

Prova:

Denotaremos por $OPT(G,T)$ o custo de toda solução ótima de CVC para a instância (G,T) e por $OPT_{n,k}(C)$ o custo de toda solução ótima de $MCP_n(k)$ para a instância C.

Do teorema 3.4.4, sabemos que $OPT_{|\Upsilon|,k}(C) = OPT(G_k(C), T(C))$. Mas, de 4.3.2, sabemos que $OPT(G_k(C), T(C)) = OPT(G_{k'}(C'), T(C'))$ e, novamente de 3.4.4, sabemos que $OPT(G_{k'}(C'), T(C')) = OPT_{|\Sigma|,k'}(C')$.

Portanto, temos $OPT_{|\Upsilon|,k}(C) = OPT(G_k(C), T(C)) = OPT(G_{k'}(C'), T(C')) = OPT_{|\Sigma|,k'}(C')$.

□

Lema 4.3.4 *Dados um inteiro $k \geq 1$ e um conjunto de cadeias C, podemos dizer que toda solução válida D de MCPr(k) para $\varrho(C)$ é tal que, para todo k-caminho $S \in D$, ou S cobre apenas cadeias de $\varrho(C)$ ou S cobre apenas cadeias de $\overline{\varrho}(C)$.*

Prova:

Sabemos que, para todo $S \in D$, o conjunto S é um k-caminho em $\varrho(C) \cup \overline{\varrho}(C)$. Mas, do lema 4.3.2, sabemos que não há ocorrência de $\varrho(s)$ em $\overline{\varrho}(t)$, nem de $\overline{\varrho}(t)$ em $\varrho(s)$, para quaisquer $s, t \in \Sigma_{dna}^*$, portanto podemos dizer ainda que ou S é um k-caminho em $\varrho(C)$ (e cobre apenas cadeias de $\varrho(C)$) ou S é um k-caminho em $\overline{\varrho}(C)$ (e cobre apenas cadeias de $\overline{\varrho}(C)$).

□

Teorema 4.3.4 *Dado um inteiro $k \geq 1$ e um conjunto de cadeias C, escrito sobre Σ_{dna}, podemos dizer que o custo de toda solução ótima de $MCP_4(k)$ para $\varrho(C)$ é igual ao custo de toda solução ótima de MCPr(k) para $\varrho(C)$.*

Prova:

Denotaremos por $OPT_{4,k}(\varrho(C))$ o custo de toda solução ótima de $MCP_4(k)$ para a instância $\varrho(C)$ e por $OPT_k(\varrho(C))$ o custo de toda solução ótima de $MCPr(k)$ para a instância $\varrho(C)$.

Vamos supor que H seja solução ótima de $MCP_4(k)$ para $\varrho(C)$. Por definição, sabemos que H é um conjunto de k-caminhos em $\varrho(C)$ que cobrem todas as cadeias de $\varrho(C)$. Pelo lema 4.3.2, sabemos que, dadas quaisquer cadeias $s, t \in C$, nunca há ocorrência de $\varrho(s)$ em $\overline{\varrho}(t)$ ou de $\overline{\varrho}(t)$ em $\varrho(s)$ e, portanto, $\varrho(C) \setminus \overline{\varrho}(C) = \varrho(C)$. Conseqüentemente, o conjunto H é solução válida de $MCPr(k)$ para $\varrho(C)$, ou seja, temos $|H| = OPT_{4,k}(\varrho(C)) \geq OPT_k(\varrho(C))$.

Vamos agora supor que D seja solução ótima de $MCPr(k)$ para $\varrho(C)$.

Do lema 4.3.4, sabemos que $D = D_n \cup D_r$, onde D_n contém k-caminhos que cobrem apenas cadeias de $\varrho(C)$ e D_r contém k-caminhos que cobrem apenas cadeias de $\overline{\varrho}(C)$.

Tomamos um k-caminho $S \in D_r$, tal que $S = (\overline{\varrho}(t_1), \overline{\varrho}(t_2), \ldots, \overline{\varrho}(t_{|S|}))$, onde t_1, t_2, ..., $t_{|S|}$ são cadeias de C, e construímos o conjunto $S' = \varrho(\overline{\varrho}^{-1}(S)) = (\varrho(t_{|S|}), \varrho(t_{|S|-1}), \ldots, \varrho(t_2), \varrho(t_1))$. Podemos afirmar que S' também é um k-caminho em $\varrho(C) \cup \overline{\varrho}(C)$, pois do lema 4.3.3, temos que $|spmax(\overline{\varrho}(t_i), \overline{\varrho}(t_{i+1}))| = |spmax(\varrho(t_{i+1}), \varrho(t_i))|$. Além disso, podemos dizer que o conjunto $D_1 = (D \setminus \{S\}) \cup \{S'\}$ é solução válida de $MCPr(k)$ para $\varrho(C)$ pois, para toda cadeia t coberta por S, temos $\overline{t}$ coberta por S' e, por definição, uma solução de $MCPr(k)$ para $\varrho(C)$ deve cobrir ou $s \in \varrho(C)$ ou $\overline{s}$. Como $|D_1| = |D|$, D_1 é também solução ótima de $MCPr(k)$ para $\varrho(C)$.

Assim, construímos um conjunto de k-caminhos D'_n de modo que, para todo $S \in D_r$, incluímos $\varrho(\overline{\varrho}^{-1}(S))$ em D'_n. Como D_r cobre apenas cadeias de $\overline{\varrho}(C)$, então D'_n cobre apenas cadeias de $\varrho(C)$.

Tomamos então o conjunto $D' = (D \setminus D_r) \cup D'_n = D_n \cup D'_n$ e, como acabamos de argumentar, sabemos que D' é solução ótima de $MCPr(k)$ para $\varrho(C)$. Sabemos ainda que D' cobre apenas cadeias de $\varrho(C)$, pois D_n e D'_n cobrem apenas cadeias de $\varrho(C)$. Mais que isso, uma vez que D' é solução de $MCPr(k)$ para $\varrho(C)$ e que $\varrho(C) \setminus \overline{\varrho}(C) = \varrho(C)$, podemos dizer que D' cobre todas as cadeias de $\varrho(C)$. Portanto, o conjunto D' é solução válida de $MCP_4(k)$ para $\varrho(C)$, ou seja, temos $|D'| = OPT_k(\varrho(C)) \geq OPT_{4,k}(\varrho(C))$.

Mas se $OPT_k(\varrho(C)) \geq OPT_{4,k}(\varrho(C))$ e $OPT_{4,k}(\varrho(C)) \geq OPT_k(\varrho(C))$, então $OPT_{4,k}(\varrho(C)) = OPT_k(\varrho(C))$.

$\square$

Teorema 4.3.5 *$MCPr(k)$ é NP-difícil para todo $k \geq 1$.*

Prova:

Calculamos o inteiro k', de modo que $7(k'-1) < k \leq 7k'$.

Tomamos um conjunto de cadeias C, escrito sobre Σ_{dna}, instância de $MCP_4(k')$, e construímos $\varrho(C)$, instância de $MCP_4(k)$ e de $MCPr(k)$, através da função ϱ definida anteriormente.

Tomamos um inteiro $i \geq 1$ e uma solução D de $DMCPr(k)$ para $(\varrho(C), i)$. Do teorema 4.3.4, sabemos que D é também solução de $DMCP_4(k)$ para $(\varrho(C), i)$ e, do teorema 4.3.3, sabemos que D é também solução de $DMCP_4(k')$ para (C, i).

$\square$

4.4 Sumário

Neste capítulo demonstramos a NP-dificuldade das famílias MCP (mesmo considerando alfabetos com apenas dois caracteres) e $MCPr$.

Para demonstrar a NP-dificuldade de cada $MCP_n(k)$, partimos de grafo com propriedades especiais, instância do problema do Caminho Hamiltoniano (HAM), e construímos instância de $MCP_n(k)$, fazendo $HAM \ \alpha \ DMCP_n(k)$.

Em seguida demonstramos também a NP-dificuldade de cada $MCPr(k)$, fazendo $DMCP_4(k') \ \alpha \ DMCPr(k)$, com $7(k'-1) < k \leq 7k'$.

Capítulo 5

Aproximações para a família MCP

Tendo comprovado que os problemas $MCP_n(k)$ eram $NP-$difíceis para quaisquer inteiros $n \geq 2$ e $k \geq 1$, decidimos pesquisar aproximações para esta família e, se possível, propor um algoritmo de aproximação para cada um de seus problemas.

Primeiramente, mostramos que qualquer algoritmo de aproximação para CVC poderia ser adaptado para todo $MCP_n(k)$. Conseguimos também provar que nenhum problema $MCP_n(k)$, com $n \geq 2$ e $k \geq 1$, nem CVC admitiam aproximação que garantisse solução menor que duas vezes a ótima.

Em seguida apresentamos um algoritmo de aproximação para CVC, que, infelizmente, é válido apenas para um subconjunto restrito de suas instâncias.

5.1 Algoritmo de $\varepsilon-$aproximação e esquema de aproximação

Vamos supor que A seja um problema de otimização, no qual, para cada instância I, temos um conjunto de soluções válidas $F(I)$ e cujo custo ótimo seja definido como $OPT(I) = \min_{K \in F(I)} custo\,(K)$, se o problema for de minimização, ou como $OPT(I) = \max_{K \in F(I)} custo\,(K)$, se o problema for de maximização.

Seja então M um algoritmo, tal que, dada qualquer instância I de A, M retorne uma solução válida $M(I) \in F(I)$. Dizemos que M é um algoritmo de $\varepsilon-$aproximação, com $\varepsilon \geq 0$, se para toda intância I de A temos:

$$\frac{|custo\,(M(I)) - OPT(I)|}{\max\{OPT(I), custo\,(M(I))\}} \leq \varepsilon$$

Portanto, uma heurística é $\varepsilon-$aproximada se, intuitivamente, o "erro relativo" da solução encontrada é no máximo ε, sendo que ε sempre assume valores entre 0 e 1. Para problemas de maximização, a solução encontrada nunca é menor que $1-\varepsilon$ vezes a ótima, enquanto que, para problemas de minimização, a solução nunca é maior que $\frac{1}{1-\varepsilon}$ vezes a ótima [10].

Teorema 5.1.1 *Para todo $n \geq 2$ e todo $k \geq 1$, $MCP_n(k)$ é $\varepsilon-$aproximável se CVC for $\varepsilon-$aproximável.*

Prova:

Dado conjunto C, escrito sobre Σ_n, instância de um $MCP_n(k)$, existe função R, tal que $R(C,k) = (G,T)$, onde $G = G_k(C)$ e $T = T(C)$ (teorema 3.4.2).

Denotaremos por $OPT(C)$ o tamanho de toda solução ótima de $MCP_n(k)$ para C, por $OPT(G,T)$ o tamanho de toda solução ótima de CVC para (G,T), por $custo\,(S)$ o tamanho de uma solução de $MCP_n(k)$ e por $custo\,(S')$ o tamanho de uma solução de CVC para (G,T).

Supondo que existe algoritmo A de $\varepsilon-$aproximação para CVC, temos que:

$$\frac{|custo\,(A(G,T)) - OPT(G,T)|}{custo\,(A(G,T))} \leq \varepsilon$$

De acordo com o teorema 3.4.3, $A(G,T)$ é também solução de $MCP_n(k)$ para C e, além disso, $OPT(G,T) = OPT(C)$ (teorema 3.4.4).

Assim, temos:

$$\frac{|custo\,(A(R(C,k))) - OPT(C)|}{custo\,(A(R(C,k)))} = \frac{|custo\,(A(G,T)) - OPT(G,T)|}{custo\,(A(G,T))} \leq \varepsilon$$

Portanto, $MCP_n(k)$ é $\varepsilon-$aproximável se CVC for $\varepsilon-$aproximável.

$\square$

Um esquema de aproximação em tempo polinomial [10] de um problema de otimização A é um algoritmo que, dados uma instância x de A e um fator de aproximação $\varepsilon > 0$, retorna uma solução com erro relativo máximo ε, num tempo limitado por um polinômio em $|x|$ que depende de ε. Se adicionalmente o tempo for limitado por um polinômio em $|x|$ e em $1/\varepsilon$, o esquema de aproximação é chamado completamente polinomial.

5.2 Limite de aproximação para MCP e CVC

Com a análise a respeito da possibilidade de aproximar MCP (e CVC), percebemos que poderíamos estabelecer um limite de aproximação para ambos os casos, mostrando assim que os dois problemas não admitem esquema de aproximação polinomial.

Teorema 5.2.1 *Dados inteiros $n \geq 2$ e $k \geq 1$, e um real positivo $\varepsilon < 1/2$ não existe $\varepsilon-$aproximação para $MCP_n(k)$, a menos que $P = NP$.*

Prova:

Dados inteiros $n \geq 2$ e $k \geq 1$, se existisse uma $\varepsilon-$aproximação para $MCP_n(k)$ tal que $\varepsilon < 1/2$, teríamos:

$$\frac{custo\,(S) - OPT(C)}{custo\,(S)} \leq \varepsilon < 0,5,$$

onde C é uma instância qualquer de $MCP_n(k)$, S é uma solução válida de $MCP_n(k)$ para C e $OPT(C)$ é o custo de toda solução ótima de $MCP_n(k)$ para C.

Logo, $custo\,(S) - OPT(C) < 0,5.custo\,(S)$ ou ainda $custo\,(S) - 0,5.custo\,(S) < OPT(C)$. Assim, $0,5.custo\,(S) < OPT(C)$ e $custo\,(S) < 2.OPT(C)$. Conseqüentemente, temos $OPT(C) \leq custo\,(S) < 2.OPT(C)$.

Portanto, se $OPT(C) = 1$, $1 \leq custo\,(S) < 2$. Como $custo\,(S)$ é sempre um valor inteiro, teríamos $custo\,(S) = 1$. Por outro lado, se $OPT(C) \geq 2$, $custo\,(S) \geq 2$. Em outras palavras, um algoritmo de aproximação com $\varepsilon < 1/2$ seria capaz de identificar se a solução ótima de uma instância qualquer de $MCP_n(k)$ possui custo ótimo 1 ou não e, neste caso, este algoritmo poderia ser usado para resolver o problema do Caminho Hamiltoniano, já que este se reduz a $MCP_n(k)$ (teorema 4.2.6). Concluímos, então, que, para todo par de inteiros $n \geq 2$ e $k \geq 1$, não é possível encontrar um algoritmo de $\varepsilon-$aproximação para $MCP_n(k)$, tal que $\varepsilon < 1/2$, ou seja, $MCP_n(k)$ não possui esquema de aproximação.

□

Teorema 5.2.2 *Dado um real positivo $\varepsilon < 1/2$ não existe $\varepsilon-$aproximação para CVC.*

Prova:

A prova deste teorema é análoga à do teorema 5.2.1.

□

5.3 O algoritmo *AlgMatch*

Chamamos de Cobertura de Vértices por Caminhos em Grafos Acíclicos (*CVCGA*) o problema de determinar o menor conjunto de caminhos orientados disjuntos que cobrem um grafo orientado e acíclico (isto é, sem ciclos orientados, podendo ter ciclos não-orientados) e chamamos de *AlgMatch* o algoritmo que resolve *CVCGA*, utilizando emparelhamento máximo em grafos bipartidos [2, p.626-ex.27-2].

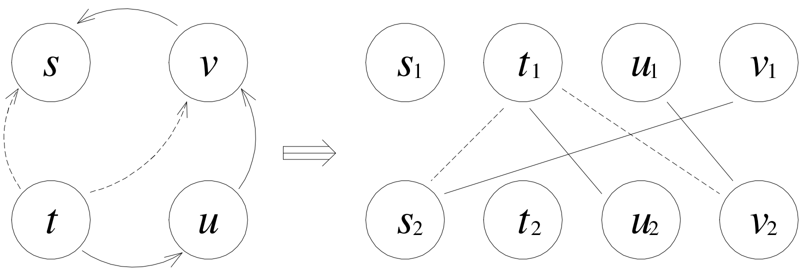
Sabendo que, quanto menor o número de caminhos em uma solução de *CVCGA*, maior a quantidade de arestas do grafo que são cobertas por estes caminhos, o algoritmo *AlgMatch* procura maximizar o número de arestas cobertas pela solução e o seu funcionamento está descrito a seguir.

A partir de um grafo G, orientado e acíclico, *AlgMatch* gera um grafo H, bipartido, no qual, para cada vértice u em G, temos dois vértices u_1 e u_2. Além disso, para cada aresta (u, v) em G, temos a aresta $\{u_1, v_2\}$ em H. Assim, se $V(G) = n$, então $V(H) = 2n$ e uma das duas partições de H é formada apenas por vértices de índice 1, enquanto a outra é formada apenas por vértices de índice 2.

Além disso, como G é acíclico, um emparelhamento máximo em H possui no máximo $n - 1$ arestas (assim como a melhor solução possível para G — um único caminho — possui também $n - 1$ arestas). Desta forma, com a determinação de um emparelhamento máximo em H encontramos uma solução ótima de *CVCGA* para G [1]. Na figura 5.1 temos um exemplo de funcionamento de *AlgMatch*.

5.3.1 Restrição e fator de aproximação

A base de *AlgMatch* é escolher o maior número de arestas possível no grafo, minimizando o número de caminhos. Quando a entrada de *AlgMatch* é um grafo que contém ciclos, o algoritmo retorna como solução um conjunto de caminhos e ciclos disjuntos. Para um mesmo número de vértices, um ciclo possui mais arestas que um caminho,



Neste caso, a solução retornada seria o emparelhamento $\{\{t_1, u_2\}, \{u_1, v_2\}, \{v_1, s_2\}\}$, resultando no caminho (t, u, v, s) no grafo original.

Figura 5.1: *AlgMatch*

e, sendo assim, existe uma chance muito grande das arestas dos ciclos serem escolhidas. Por causa disso, grafos de entrada com número ilimitado de ciclos inviabilizam a aproximação.

Teorema 5.3.1.1 *Dado um inteiro c, AlgMatch é algoritmo de aproximação para $CVCCL(c)$ de fator $c/(1+c)$, onde $CVCCL(c)$ é uma versão de CVC cujas instâncias são grafos orientados, nos quais c é o número máximo de ciclos disjuntos.*

Prova:

Dado um grafo G orientado, $custo(AlgMatch(G)) = b + a$, onde b é o número de caminhos e a é o número de ciclos retornado. Definimos como $edge(AlgMatch(G))$ a função que retorna o número de arestas presentes em $AlgMatch(G)$, como $OPT(G)$ o custo de toda solução ótima de CVC para G e como $edge_{OPT}(G)$ o número de arestas presentes em qualquer solução ótima de CVC para G. Podemos verificar que $edge_{OPT}(G) = |V(G)| - OPT(G)$ e que $edge(AlgMatch(G)) = |V(G)| - b$. Como $AlgMatch$ maximiza o número de arestas na solução, $edge(AlgMatch(G)) \geq edge_{OPT}(G)$ e, então, $|V(G)| - b \geq |V(G)| - OPT(G)$. Portanto, $OPT(G) \geq b$, ou seja, dada uma instância G, o número de caminhos retornado em $AlgMatch(G)$ nunca é maior que o número de caminhos na solução ótima para G.

Assim, se denotarmos por $dcyc(G)$ o número de ciclos disjuntos em um grafo G, temos $custo(AlgMatch(G)) \leq OPT(G) + dcyc(G)$. Logo, no pior caso, teríamos:

$$\frac{custo(AlgMatch(G)) - OPT(G)}{custo(AlgMatch(G))} \leq \frac{OPT(G) + c(G) - OPT(G)}{OPT(G) + dcyc(G)} \leq \frac{dcyc(G)}{1 + dcyc(G)}$$

Portanto, se $dcyc(G) \leq c$, onde c é uma constante, *AlgMatch* é algoritmo de aproximação de $CVCCL(c)$ com fator $c/(1+c)$.

$\square$

Lema 5.3.1.1 *Dados um inteiro $k \geq 1$, um conjunto de cadeias C e o grafo $G = G_k(C)$, o conjunto $S = (s_1, s_2, \ldots, s_{|S|})$ é um k-ciclo em C se, e somente se, S for também um ciclo em G.*

Prova:

Se $S = (s_1, s_2, \ldots, s_{|S|})$ é um ciclo em G, S é também um caminho em G e $(s_{|S|}, s_1) \in E(G)$. Do lema 3.4.1, sabemos que S é um k-caminho em C e, como $(s_{|S|}, s_1) \in E(G)$, $|spmax(s_{|S|}, s_1)| \geq k$ e. Logo, S é também um k-ciclo em C.

Por outro lado, se $S = (s_1, s_2, \ldots, s_{|S|})$ é um k-ciclo em C, S é também um k-caminho em C e $|spmax(s_{|S|}, s_1)| \geq k$. Do lema 3.4.1, sabemos que S é um caminho em G e, como $|spmax(s_{|S|}, s_1)| \geq k$, $(s_{|S|}, s_1) \in E(G)$. Logo, S é um ciclo em G.

$\square$

Teorema 5.3.1.2 *Dados um inteiro $k \geq 1$ e um conjunto de cadeias C, C possui no máximo n k-ciclos disjuntos se, e somente se, $G_k(C)$ possui no máximo n ciclos disjuntos.*

Prova:

Denotaremos por $CYC(G)$ o número máximo de ciclos disjuntos em G e por $CYC_k(C)$ o número máximo de k-ciclos disjuntos em C.

Se C possui no máximo n k-ciclos disjuntos, consideraremos $B = \{S_1, S_2, \ldots, S_n\}$, um conjunto de k-ciclos disjuntos em C. Do lema 5.3.1.1, para cada $S_i \in B$, sabemos que S_i é um ciclo em G, portanto B é um conjunto de ciclos disjuntos em G. Logo, $n = CYC_k(C) \leq CYC(G)$.

Do mesmo modo, se G possui no máximo n ciclos disjuntos, consideraremos $B = \{S_1, S_2, \ldots, S_n\}$, um conjunto de ciclos disjuntos em G. Do lema 5.3.1.1, para cada $S_i \in B$, sabemos que S_i é um k-ciclo em C, portanto B é um conjunto de k-ciclos disjuntos em G. Logo, $n = CYC(G) \leq CYC_k(C)$.

Se $CYC_k(C) \leq CYC(G)$ e $CYC(G) \leq CYC_k(C)$, $CYC_k(C) = CYC(G)$.

$\square$

Teorema 5.3.1.3 *Dados inteiros $k \geq 1$, $n \geq 2$ e $c \geq 0$, existe algoritmo de aproximação para $MCPCL_n(k, c)$ de fator $c/(1 + c)$, onde $MCPCL_n(k, c)$ é uma versão de MCP cujas instâncias são grafos orientados, nos quais c é o número máximo de k-ciclos disjuntos.*

Prova:

Tomamos instância C de $MCPCL_n(k, c)$ e sabemos que c é o número máximo de k-ciclos disjuntos em C. Do teorema 5.3.1.2, sabemos também que c é o número máximo de ciclos disjuntos em $G_k(C)$, ou seja, $G_k(C)$ é instância de $CVCCL(c)$.

Mas *AlgMatch* é algoritmo de aproximação de fator $c/1 + c$ para $CVCCL(c)$ (teorema 5.3.1.1). Pelo teorema 5.1.1, podemos concluir que existe algoritmo de aproximação para $MCPCL_n(k, c)$ de fator $c/(1 + c)$.

$\square$

5.3.2 Observações

Apesar das restrições e do fator de aproximação não ser tão bom, sabendo que, em casos práticos de montagem de fragmentos de DNA, os conjuntos de fragmentos dão origem a grafos que geralmente não apresentam muitos ciclos disjuntos, consideramos este algoritmo de aproximação como um resultado positivo da nossa pesquisa.

5.4 Sumário

Neste capítulo, pesquisamos aproximações para CVC e todos os problemas de MCP, determinando um limite de aproximação de $1/2$ e apresentando também um algoritmo de aproximação para instâncias restritas destes problemas.

Capítulo 6

Conclusões

Tendo em vista os nossos objetivos iniciais, consideramos que, no término destes dois anos de pesquisas, chegamos a resultados bastante interessantes.

Na primeira etapa do trabalho, conforme nossa expectativa, conseguimos elaborar a demonstração de NP-dificuldade de todos os problemas das famílias MCP e $MCPr$ e, já na segunda etapa, passamos a pesquisar aproximações para MCP, tendo como base o problema CVC, que é seu modelo de grafos.

Nesta segunda fase, as dificuldades acabaram sendo muito maiores, talvez em razão da pequena familiaridade que tínhamos com o tema. Conseguimos, no entanto, mostrar que os problemas analisados, incluindo CVC e todos os problemas da família MCP, não admitiam aproximação que garantisse solução menor que duas vezes a solução ótima e apresentamos uma primeira idéia de aproximação para instâncias restritas destes problemas.

Para projetos futuros, porém, pensamos que ainda há muito a ser explorado em pesquisas de aproximação para MCP.

Apêndice A

Demonstrações complementares

A.1 Prova do teorema 3.4.2

Gusfield, Landau e Schieber [7] propuseram um algoritmo eficiente para calcular $spmax(s,t)$ para todo par de cadeias $s,t \in C$, em $O(\|C\| + |C|^2)$, o que é suficiente para construir $G_k(C)$, mas não para construir $T(C)$.

Sabemos que, para construir a árvore generalizada de sufixos para C [7], é preciso adicionar um caracter inédito ao final de cada cadeia de C. Consideraremos que, para cada cadeia $s \in C$, este caracter é $\mathtt{s}$.

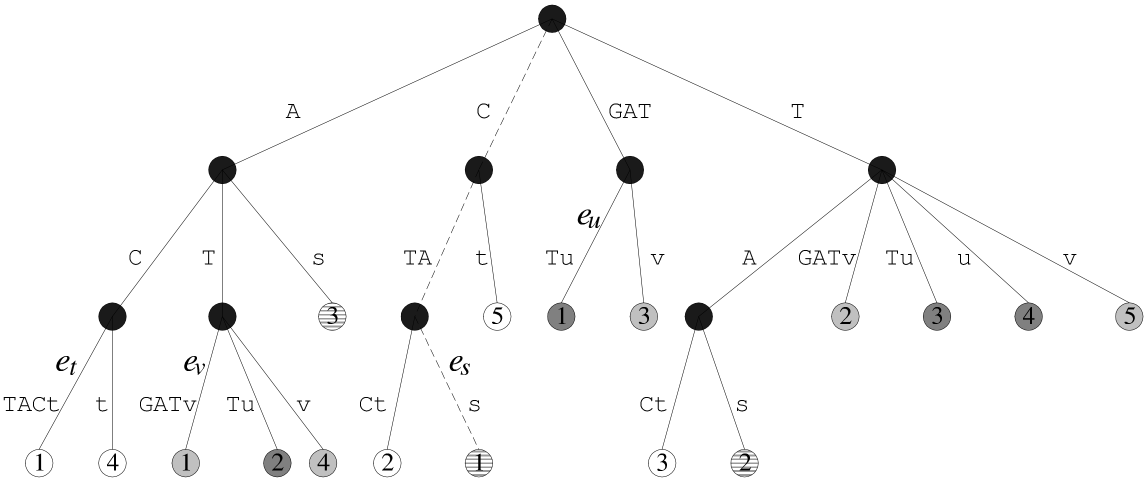
Um caminho em uma árvore de sufixos sempre começa na raiz e termina em uma folha qualquer, representando o sufixo de alguma cadeia, sendo que este sufixo é dado pela concatenação dos rótulos das arestas do caminho.

Cada folha da árvore é rotulada com um número que ajuda a identificar o sufixo que o seu caminho representa. Assim, se uma folha está rotulada com o número i e a última aresta de seu caminho apresenta o caracter especial $\mathtt{v}$, podemos dizer que o seu caminho representa o sufixo de v que começa a partir do caracter da posição i em v [7].

Sabendo que o "caminho de uma cadeia s" na árvore de sufixos é tal que a concatenação dos rótulos de suas arestas corresponde à própria cadeia s (concatenada com o caracter especial $\mathtt{s}$), podemos definir formalmente este caminho como aquele que (1) termina com uma folha rotulada com 1 e (2) apresenta o caracter especial $\mathtt{s}$ no rótulo de sua última aresta.

Chamamos de e_s a última aresta do caminho da cadeia s e de n_s o nó que antecede e_s. Caracterizamos então uma cadeia s como subcadeia própria de alguma outra cadeia de C quando e_s é rotulada apenas com o caracter especial $\mathtt{s}$, pois, se isto ocorre, então o vértice n_s está ligado também a pelo menos mais uma aresta, que está no mesmo nível de e_s, mas que é diferente desta, significando que n_s faz parte do caminho de um sufixo de alguma outra cadeia de C e, conseqüentemente, que s faz parte de alguma outra cadeia de C (figura A.1).

O algoritmo de Gusfield, Landau e Schieber [7] realiza uma busca em profundidade na árvore para calcular as sobreposições. Portanto, para construir $T(C)$, basta fazer um pequena modificação neste algoritmo e verificar, durante a busca por profundidade, a caracterização acima para todas as cadeias de C. Assim, para cada cadeia $s \in C$, se s não apresentar esta caracterização, então a cadeia s não é subcadeia própria de nenhuma outra cadeia de C e deve ser incluída em $T(C)$. Esta modificação não altera a complexidade assintótica do algoritmo, que, portanto, continua sendo $O(\|C\| + |C|^2)$.



s	`CTA`	u	`GATT`
t	`ACTAC`	v	`ATGAT`

$$C = \{s, t, u, v\}$$

Os caracteres `s`, `t`, `u` e `v` foram acrescentados ao final das cadeias s, t, u e v, respectivamente.

As cores das folhas indicam de qual cadeia é o sufixo que elas representa m.

Como o caminho da cadeia s, cujas arestas estão tracejadas, termina com uma aresta rotulada apenas com `s`, então s é subcadeia própria de alguma outra cadeia de C.

$$T(C) = \{t, u, v\}$$

Figura A.1: Árvore generalizada de sufixos

A.2 A complexidade do algoritmo de coloração de arestas (teorema 4.2.1)

Para o passo 2 do algoritmo 4.2.1, aproveitamos o algoritmo de coloração proposto por Misra e Gries [9] em sua prova construtiva do teorema de Vizing [3, p.103].

Deduzimos aqui a complexidade deste algoritmo, cujos passos dominantes estão resumidos a seguir, uma vez que esta não foi calculada pelos seus autores.

Dados um grafo G, com $|V(G)| = n$ e $|E(G)| = m$, e uma aresta não colorida $(u, v) \in E(G)$, o algoritmo constrói uma lista especial chamada *fan* em $O(\Delta^2(G))$, inverte as cores em um certo caminho bicolorido em $O(m)$ e faz a rotação da *fan* em $O(\Delta(G))$. Como este algoritmo de coloração deve ser repetido m vezes, a complexidade do passo 2 é $O(m^2 + m\Delta^2(G))$.

Bibliografia

[1] F. R. Cerqueira. Montagem de Fragmentos de DNA. Master's thesis, Instituto de Computação, UNICAMP, Campinas, SP, January 2000.

[2] T.H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[3] R. Diestel. *Graph Theory.* Springer-Verlag New York, Inc, 1997.

[4] C. E. Ferreira, C. C. de Souza, and Y. Wakabayshi. Rearrangement of DNA fragments: a branch-and-cut algorithm. Versão revisada a ser publicada. *Discrete Applied Mathematics*, 2001.

[5] J. Gallant, D. Maier, and J. A. Storer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, February 1980.

[6] M. R. Garey and D. S. Johnson. *Computer and Intractability - A Guide to the Theory of NP-Completeness.* Freeman, New York, 1979.

[7] D. Gusfield, G. M. Landau, and B. Schieber. An efficient algorithm for the all pairs suffix-prefix problem. *Information Processing Letters*, 41(4):181–185, March 1992.

[8] D. Kelly. *Automata and Formal Languages.* Prentice-Hall, 1995.

[9] J. Misra and D. Gries. A constructive proof of Vizing's Theorem. *Information Processing Letters*, 41(3):131–133, March 1992.

[10] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[11] J. C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, 1997.