Instituto de Matemática e Estatística
Universidade de São Paulo

Trabalho de Formatura Supervisionado

O Problema da Subseqüência Comum Máxima sem Repetições

Christian Tjandraatmadja
Supervisora: Prof.ª Dr.ª Cristina Gomes Fernandes
Orientador: Prof. Dr. Carlos Eduardo Ferreira

São Paulo
2007

Apresentação

Esta monografia detalha um projeto de iniciação científica de que o autor fez parte. Este projeto também é seu trabalho de conclusão de curso[1] para o curso de Bacharelado em Ciência da Computação, no Instituto de Matemática e Estatística, Universidade de São Paulo. A iniciação científica teve apoio financeiro (por um semestre) da FAPESP, processo 07/54282-6.

Resumo

Uma método de se comparar dois genomas é determinar uma subseqüência comum máxima entre os genomas. O problema da subseqüência comum máxima consiste em, dadas duas seqüências s e t, encontrar uma seqüência que seja subseqüência tanto de s como de t e que tenha comprimento máximo. Apresentamos um estudo desse problema sob o ponto de vista de programação inteira e combinatória poliédrica. Uma de suas variantes, o problema da subseqüência comum máxima sem repetições, também é explorada sob o mesmo ponto de vista. Ela é uma outra forma de se medir similaridade entre genomas. Esta variante adiciona a restrição de que não podemos ter dois símbolos repetidos na solução. Após o estudo, descrevemos uma implementação de um algoritmo baseado na técnica *branch and cut* para resolvê-la.

Palavras-chave: subseqüência comum máxima sem repetições, subseqüência comum máxima, programação inteira, combinatória poliédrica.

Abreviações

O texto usa as abreviações a seguir.

- LCS: subseqüência comum máxima (*longest common subsequence*)
- RFLCS: subseqüência comum máxima sem repetições (*repetition free longest common subsequence*)

[1] Disciplina de código MAC0499 e título Trabalho de Formatura Supervisionado. Esta monografia foi entregue para avaliação no dia 3 de Dezembro de 2007.

Sumário

Capítulo 1

Introdução

O propósito deste projeto é estudar o problema da subseqüência comum máxima sem repetições (*repetition free longest common subsequence*, RFLCS). Ela é uma variante do problema da subseqüência comum máxima (*longest common subsequence*, LCS), que também é estudado.

Ambos são problemas de otimização combinatória. Antes de definir o problema, vale a pena dar uma noção dessa área e os pontos de vista que escolhemos para estudá-lo.

1.1 Abordagens

Os problemas do LCS e do RFLCS fazem parte da área de otimização combinatória, uma subárea de ciência da computação e matemática aplicada.

Problemas de otimização são aqueles em que buscamos a melhor solução em um espaço de soluções viáveis. Por exemplo, dado um conjunto de ruas, queremos encontrar o caminho mais curto entre dois locais, passando apenas por essas ruas. Nesse caso, as soluções viáveis seriam todos os caminhos possíveis entre esses dois pontos. Tal solução é chamada de solução ótima e não necessariamente é a única. Mais formalmente, procuramos maximizar ou minimizar uma função objetivo sujeito a um conjunto de restrições.

Otimização combinatória envolve problemas de otimização cujo espaço de soluções viáveis é discreto (ao oposto de contínuo, como nos números reais). O exemplo das ruas é um problema de otimização combinatória, pois podemos enumerar todos os possíveis caminhos. Ainda mais, uma forma de resolver esse exemplo é enumerar todas as soluções e escolher a que tem o caminho mais curto. Porém, o número de soluções é muito grande (exponencial no tamanho da entrada), o que torna tal estratégia inviável. Isso é o que ocorre na maioria dos problemas dessa área.

Nesta monografia, estudamos os problemas do LCS e do RFLCS sob o ponto de vista de programação inteira e combinatória poliédrica.

- **Programação inteira:** Muitos problemas de otimização podem ser formulados com uma função objetivo linear e um conjunto de restrições de igualdade ou desigualdade também lineares. Programação linear é a área em que resolvemos esses problemas tendo em mente essas formulações. Tais problemas podem ser resolvidos em tempo polinomial. Em programação inteira, permitimos, além das restriçoes lineares, restrições de que as variáveis sejam inteiras. Isso torna o problema NP-difícil (isto é, o problema é tão difícil quanto os problemas mais difíceis da classe de problemas NP e, como conseqüência, é improvável que exista algoritmo polinomial para o problema). Essa área nos interessa pois o LCS e o RFLCS podem ser modelados como problemas de programação inteira, e existem técnicas baseadas em programação linear para resolvê-los.

- **Combinatória poliédrica:** Quando olhamos para o espaço de soluções de um problema de programação linear, podemos vê-lo como uma série de restrições lineares, que formam um poliedro, e uma direção para a qual queremos avançar. Ao passar para programação inteira, o espaço de soluções se torna um conjunto de pontos, mas ainda é importante conhecer o poliedro associado à formulação sem a restrição de integralidade, tendo em mente que usamos técnicas baseadas em programação linear. Quanto mais perto esse poliedro está do espaço de soluções, melhor a formulação associada a ele é, e o ideal é que ele seja o casco convexo das soluções. Combinatória poliédrica é o estudo das características de tal poliedro, na tentativa de buscar a melhor formulação possível.

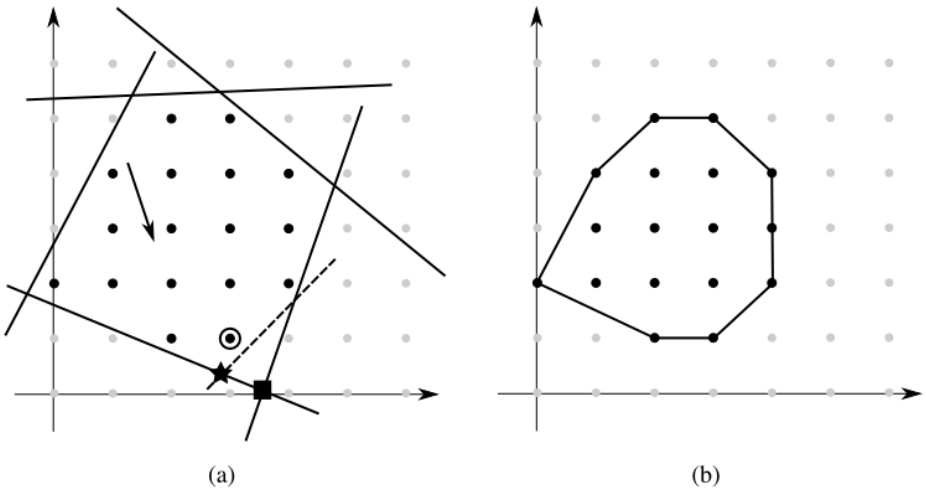

Figura 1.1: (a) Um poliedro associado a uma formulação. (b) O casco convexo das soluções viáveis inteiras. (Uma descrição melhor está a seguir.)

Na Figura 1.1(a), mostramos um poliedro no qual as retas cheias que o formam são as restrições lineares de uma formulação, a seta indica a direção da função objetivo e o ponto marcado com um círculo é a solução ótima inteira. Um outro poliedro é a união da reta tracejada com as retas cheias, que é mais forte que a primeira. Desconsiderando a restrição de integralidade, o quadrado é a solução ótima da primeira formulação e a estrela é a da segunda. A Figura 1.1(b) mostra o poliedro associado a melhor formulação possível. Observe que, mesmo se desconsiderarmos a restrição de integralidade, a solução ótima é inteira para qualquer função objetivo.

Outros dois pontos de vista que consideramos são grafos e algoritmos de aproximação.

- **Grafos:** Um grafo $G = (V, E)$ é uma estrutura de dados composta por um conjunto de vértices V e um conjunto de arestas E, onde cada aresta representa uma ligação entre dois vértices. Grafos são bastante úteis para representar e entender problemas. No exemplo das ruas, podemos montar um grafo onde os vértices são alguns locais (como cruzamentos), e as arestas são as ruas que as ligam, e cada uma pode ser associada a um peso que indica seu comprimento.

- **Algoritmos de aproximação:** Existem vários problemas para os quais não conseguimos encontrar solução em tempo polinomial. Para eles, pode valer a pena estudar algoritmos de aproximação, que procuram uma solução que seja perto da ótima. Geralmente buscamos provar a qualidade desses algoritmos, isto é, encontrar uma garantia de que o algoritmo devolve uma solução perto da ótima.

1.2 Definições

Um **alfabeto** é um conjunto finito e seus elementos são chamados de **símbolos**. Uma **seqüência** é uma lista ordenada de símbolos. Neste texto, todas as seqüências são finitas e sobre um alfabeto implícito, que pode ser considerado como o conjunto dos símbolos da seqüência em questão. O comprimento de uma seqüência s é denotado por $|s|$ e o i-ésimo símbolo de s é denotado por s_i.

Uma **subseqüência** de uma seqüência s é uma seqüência que pode ser obtida removendo zero ou mais elementos de s. De outra forma, podemos ver uma subseqüência de s como uma seqüência dada pela escolha de elementos de s mantendo a ordem relativa entre eles. Formalmente, t é uma subseqüência de s se existe uma seqüência estritamente crescente $i_1 i_2 \ldots i_k$ de índices de s tal que $s_{i_j} = t_j$.

Dadas duas seqüências s e t, dizemos que uma seqüência é uma **subseqüência comum** de s e t se ela é subseqüência tanto de s como de t.

Uma seqüência é uma **subseqüência comum máxima** (*longest common subsequence*, LCS) de seqüências s e t se ela é uma subseqüência comum de s e t e tem o maior comprimento possível entre todas as outras subseqüências comuns de s e t.

Figura 1.2: Uma seqüência s e uma subseqüência de s.

Dizemos que uma seqüência é **sem repetições** se cada símbolo ocorre nela no máximo uma vez, isto é, não há símbolos repetidos na seqüência. Definimos **subseqüência comum máxima sem repetições** (*repetition-free longest common subsequence*, RFLCS) como sendo uma subseqüência sem repetições comum de s e t que tenha comprimento máximo.

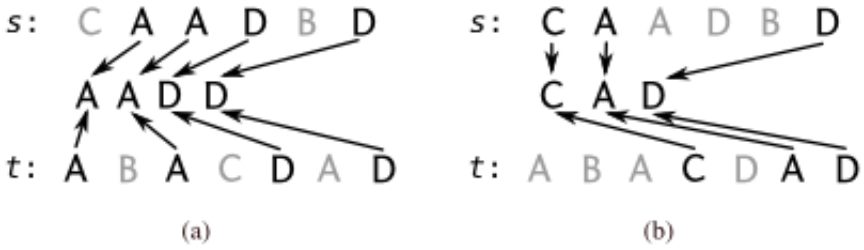

Figura 1.3: Duas seqüências s e t e (a) um LCS de s e t, (b) um RFLCS de s e t.

Definimos o **problema do LCS** como encontrar um LCS de duas seqüências e, analogamente, o **problema do RFLCS** como encontrar um RFLCS de duas seqüências.

1.3 Aplicações

O problema do LCS é um problema bem conhecido e estudado em computação. Ele é útil para comparar de seqüências, pois o comprimento do LCS é o número máximo de elementos iguais entre elas de forma a manter a ordem relativa entre eles. Isto é, quanto mais parecidas são as seqüências, mais comprido é o LCS.

Ele é usado em programas para encontrar diferenças entre arquivos, como o `diff`, do Unix. Corretores ortográficos também podem usar o LCS para determinar quais palavras de um dicionário são as mais próximas a uma palavra com um erro de ortografia.

Em sistemas de controle de versão, podemos armazenar um arquivo e conjuntos mínimos de modificações dele de forma que seja possível reconstruir versões modificadas desse arquivo. Para isso, resolver o problema do LCS é útil. Com essa mesma idéia, ele também aparece em alguns algoritmos de compressão de dados, nos quais é necessário encontrar o maior número possível de partes que são comuns entre seqüências de dados.

Mais recentemente, o interesse do LCS vem da área de Biologia Molecular Computacional. O LCS é uma forma de medir similaridade entre seqüências de DNA ou de proteínas. Isso ajuda a determinar se as seqüências compartilham um mesmo ancestral.

Devido a características dessas seqüências, diversas variantes do LCS surgiram. Estudamos neste texto uma delas, o RFLCS, cuja aplicação vem da comparação entre genomas, levando em conta a teoria de rearranjo de genomas. Nessa abordagem, representamos cada gene como um símbolo, sem se preocupar com sua estrutura interna, e consideramos a evolução de genomas baseada em processos de rearranjo.

Durante esses processos, duplicatas de genes podem surgir de diversas maneiras. Uma suposição comum feita para comparar rearranjos de genomas é que cada gene é homólogo a no máximo um gene de outro genoma. Em outras palavras, dadas duas seqüências s e t de genomas, essa suposição diz que, dado um símbolo em s, só pode ter no máximo uma ocorrência desse símbolo em t. Essa suposição tem fundamento para alguns genomas pequenos, como os de vírus ou de mitocôndrias, e torna o problema mais fácil, já que o LCS é uma medida boa para esse caso. No entanto, para genomas maiores, ela é problemática devido ao número de duplicatas de genes. No genoma humano, aproximadamente 15% dos genes de proteína são duplicatas [14]. Ainda mais, a situação é complicada pelo processo de divergência de seqüência, no qual as duplicatas podem se tornar estruturalmente e funcionalmente diferente até o ponto de deixarem de ser duplicatas, mas continuam sendo da mesma família de genes, que são homólogas e funcionalmente similares.

Com esse conceito de família de genes, ao invés de considerar cada gene separadamente, podemos considerar um representante de cada família, chamado de exemplar. Usando essa idéia, Sankoff [19] propõe uma medida de comparação de genes, denominado de distância exemplar. Ademais, com base nessa medida, Bonizzoni et al. [4] estuda uma variante do LCS, a subseqüência comum máxima exemplar (*exemplar longest common subsequence*, ELCS). O ELCS é uma generalização do RFLCS.

O RFLCS é uma outra medida que leva em consideração a noção de exemplar. Ele pode ser visto como uma medida de similaridade entre dois genomas na qual, para cada família de genes, desconsideramos todos os genes exceto um, o exemplar.

Uma noção melhor sobre rearranjo de genomas e duplicação de genes pode ser obtida em [21, 20].

Em particular, a variante do RFLCS que permite reversões de seqüências é uma medida boa para evidenciar uma conjectura que afirma que o cromossomo Y é derivado do cromossomo X a partir de cinco grandes reversões [13, 24]. Nessas referências,

observa-se que o cromossomo X é dividido em quatro ou cinco trechos (estratos) que são comparáveis com partes do cromossomo Y em termos da distância entre eles.

1.4 Organização do texto

O plano para este texto é o seguinte. No próximo capítulo, discutimos alguns conceitos importantes para o problema do LCS, apresentamos uma formulação simples, observamos alguns grafos associados ao problema, e chegamos a uma formulação completa e minimal do poliedro do LCS. No terceiro capítulo, apresentamos algumas formulações para o problema do RFLCS, observamos alguns grafos associados ao problema e depois discutimos alguns algoritmos de aproximação. No quarto capítulo, discutimos a implementação de um algoritmo *branch and cut* e mostramos uma forma de resolver em tempo polinomial o problema da separação, explicado no capítulo. O último capítulo é uma conclusão, na qual discutimos alguns possíveis trabalhos futuros.

Capítulo 2

Um estudo do LCS

2.1 Resultados conhecidos

Devido às diversas aplicações do LCS, em especial sua aplicação em Biologia Computacional, vários algoritmos para se resolver o problema foram estudados.

O mais conhecido é um algoritmo de programação dinâmica [25] de complexidade de tempo $O(nm)$, onde n e m são os comprimentos das duas seqüências. O algoritmo usa a seguinte recorrência:

$$R(i,j) = \begin{cases} 0, & \text{se } i = 0 \text{ ou } j = 0 \\ R(i-1,j-1)+1, & \text{se } s_i = t_j \\ \max\{R(i-1,j), R(i,j-1)\}, & \text{se } s_i \neq t_j \end{cases}$$

Uma descrição didática desse algoritmo pode ser encontrada em [8].

O algoritmo de melhor complexidade em pior caso é dado por Masek e Paterson [16], de complexidade de tempo $O(n \max\{1, m/\log n\})$ e de espaço $O(n^2/\log n)$. Uma comparação mais detalhada desses e outros algoritmos para o problema do LCS pode ser encontrada em [18, 3].

Considerando o problema do LCS para um conjunto de seqüências de tamanho arbitrário ao invés de apenas dois (as definições são análogas), Maier [15] prova que o problema de decisão do LCS (dado um inteiro k e um conjunto de seqüências R, o comprimento do LCS de R é maior que k?) é NP-completo para alfabetos de tamanho pelo menos 2. Se o tamanho do conjunto de seqüências for fixo (digamos, k), existe um algoritmo de programação dinâmica que resolve o problema com complexidade de tempo e de espaço $O(n^k)$ [22].

Ainda considerando um conjunto de seqüências, a menos que P = NP, não existe algoritmo de aproximação para o LCS que garante uma solução boa, embora existam algoritmos que devolvem soluções boas em média [12]. Um algoritmo de aproximação simples de nome *Long Run algorithm* é dado por Jiang e Li em [12]. O algoritmo é o seguinte: para cada símbolo a do alfabeto, é fácil encontrar uma subseqüência

comum a^m (isto é, a repetido m vezes) com m máximo. Escolhemos o símbolo que maximiza tal m. Existe outro algoritmo de aproximação apresentado por Bonizzoni et al. [5] de nome *Expansion algorithm* que, em média, é melhor que o *Long Run*.

Uma outra forma de se resolver o problema do LCS é reduzi-lo ao problema da subseqüência crescente máxima, definido da seguinte forma: dada uma seqüência z de inteiros, encontrar uma subseqüência de z cujos elementos estão em ordem estritamente crescente. Para isso, dadas as seqüências s e t, construa a seqüência z como a seguir: para cada i de 1 até $|s|$, para cada j de $|t|$ a 1, coloque j no final da seqüência z se $s_i = t_j$. Não é difícil ver que uma subseqüência crescente máxima de z define um LCS de s e t.

2.2 Estrutura do problema e uma formulação simples

Dois conceitos importantes inerentes à estrutura do problema do LCS são casamentos e cruzamentos.

Dadas duas seqüências s e t, um **casamento** com símbolo associado a é um par ordenado de índices (i, j) tal que $s_i = t_j = a$. Em outras palavras, considerando um grafo cujos vértices são os elementos das seqüências s e t, uma aresta é chamada de casamento se ela liga o i-ésimo elemento de s ao j-ésimo elemento de t e esses dois elementos têm o mesmo símbolo associado. Vamos denotar o conjunto de casamentos em s e t de $\mathscr{C}$. Denotamos também o conjunto de casamentos em s e t com um mesmo símbolo associado a de $\mathscr{C}(a)$. A Figura 2.1(a) mostra um casamento e 2.1(b) mostra todos os casamentos (isto é, os elementos do conjunto $\mathscr{C}$) de s e t.

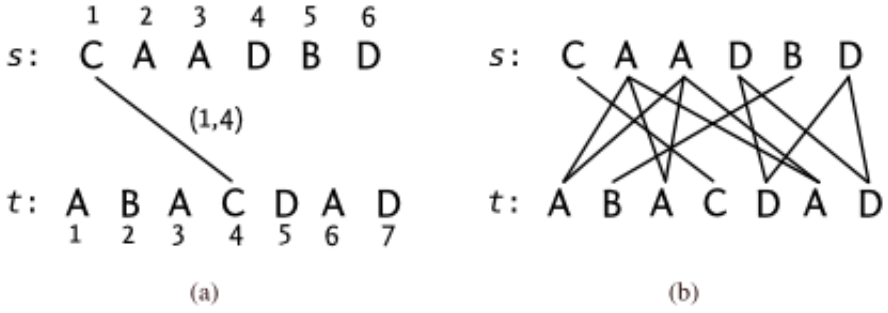

Figura 2.1: (a) Exemplo de um casamento. (b) Todos os casamentos de s e t.

Informalmente, dizemos que dois casamentos se cruzam se, na figura relacionada, eles se intersectam, tanto no meio como no extremo. Mais precisamente, dois casamentos **se cruzam** se ($i \leq k$ e $j \geq \ell$) ou ($k \leq i$ e $\ell \geq j$). Vale a pena destacar também o oposto dessa definição: dois casamentos não se cruzam se ($i < k$ e $j < \ell$) ou ($k < i$ e $\ell < j$). Dizemos que um conjunto de casamentos C é **sem cruzamento**

se os casamentos de C não se cruzam dois a dois.

Definimos também a relação $\prec_\times$ da seguinte forma. Sejam c e d dois casamentos. Então $c \prec_\times d$ se e só se ($i \leq k$ e $j \geq \ell$), isto é, uma "parte" da definição de cruzamento. Assim, dois casamentos c e d se cruzam se e só se $c \prec_\times d$ ou $d \prec_\times c$. De forma análoga, definimos uma relação $\prec_{||}$ para não-cruzamento. Dados dois casamentos c e d, $c \prec_{||} d$ se e só se ($i < k$ e $j < \ell$). Ou seja, dois casamentos c e d não se cruzam se e só se $c \prec_{||} d$ ou $d \prec_{||} c$. Essas relações são importantes mais adiante.

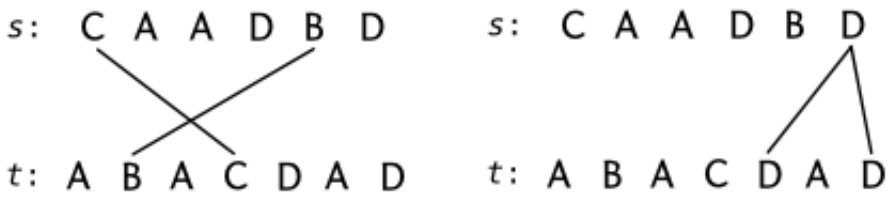

Figura 2.2: Exemplos de cruzamento.

Quando dois casamentos se cruzam, se eles não tiverem extremos em comum, a ordem relativa entre os elementos associados a eles em s é oposta à em t. Se eles tiverem extremos em comum, então estamos nos referindo a um mesmo elemento duas vezes. Essas observações são dicas à seguinte afirmação:

Fato 2.1. *Dadas duas seqüências s e t, um conjunto C de casamentos sem cruzamento em s e t define uma subseqüência comum de s e t de comprimento $|C|$.*

Prova. Seja C um conjunto de casamentos sem cruzamento. Tome a relação de não-cruzamento $\prec_{||}$ definida anteriormente. É fácil ver que, em $\mathscr{C}$, ela é irreflexiva ($c \prec_{||} c$) e transitiva (se $c \prec_{||} d$ e $d \prec_{||} e$ então $c \prec_{||} e$). Portanto, ela é uma relação de ordem parcial estrita em $\mathscr{C}$. Já em C, ela é uma relação de ordem total estrita pois, como C é sem cruzamento, vale que ou $c \prec_{||} d$, ou $d \prec_{||} c$ para quaisquer dois casamentos c e d de C.

Isso significa que, usando essa relação, podemos obter uma ordenação dos casamentos, digamos, $(i_1, j_1), (i_2, j_2), \ldots, (i_n, j_n)$ para $n = |C|$. Então as seqüências $i_1 i_2 \ldots i_n$ e $j_1 j_2 \ldots j_n$ são estritamente crescentes. Assim, a seqüência $s_{i_1} s_{i_2} \ldots s_{i_n}$, que é subseqüência de s, é igual à seqüência $t_{j_1} t_{j_2} \ldots t_{j_n}$, que é subseqüência de t. Essa seqüência é uma subseqüência comum de s e t de comprimento $n = |C|$. □

É claro que a afirmação acima implica que, se esse conjunto tiver cardinalidade máxima, então a subseqüência comum é um LCS. A Figura 2.3 ilustra melhor tal conjunto.

Figura 2.3: Um conjunto de casamentos sem cruzamento, que define um LCS.

O fato seguinte vem facilmente do fato anterior.

Fato 2.2. *Dadas duas seqüências s e t, para toda subseqüência comum u de s e t, existe pelo menos um conjunto de casamentos sem cruzamento em s e t de cardinalidade $|u|$ que a define.*

Prova. Seja u uma subseqüência comum de s e t de comprimento n. Então existem duas seqüências estritamente crescentes de índices $i_1 i_2 \dots i_n$ e $j_1 j_2 \dots j_n$ tal que $u_k = s_{i_k} = t_{j_k}$ para todo $k = 1 \dots n$. O conjunto de casamentos $(i_1, j_1), (i_2, j_2), \dots, (i_n, j_n)$ define u e tem cardinalidade $n = |u|$. □

Os Fatos 2.1 e 2.2 implicam que o espaço de todos os conjuntos de casamentos sem cruzamento em s e t define todo o espaço de subseqüências comuns de s e t, conforme representado na Figura 2.4. É fácil ver que podem existir dois conjuntos distintos de casamentos que definem uma mesma subseqüência comum.

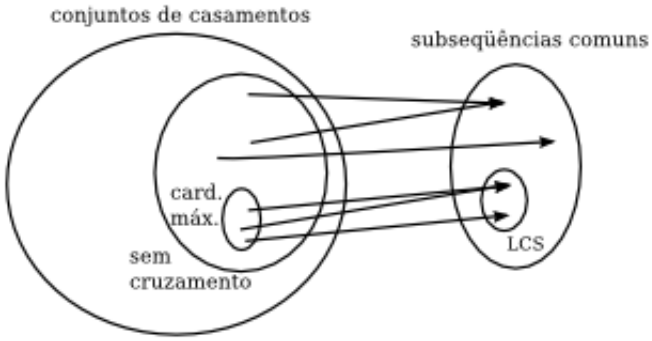

Figura 2.4: Um diagrama representando a relação entre conjuntos de casamentos e subseqüências comuns.

Isso significa que podemos resolver o problema do LCS escolhendo o maior número possível de casamentos que não se cruzam dois a dois. Assim, podemos formular o

problema do LCS como um problema de programação inteira. Explicamos a formulação em palavras primeiro.

maximizar	o número de casamentos escolhidos
sujeito às restrições	para cada dois casamentos que se cruzam, só podemos escolher um deles;
	para cada casamento, escolhemos ele ou 0 ou 1 vez.

A seguir, reescrevemos a formulação acima de forma matemática.

Lembre-se que $\mathscr{C}$ é o conjunto de todos os casamentos de s e t. Seja $z \in \{0,1\}^{\mathscr{C}}$ um vetor tal que $z_{ij} = 1$ se escolhemos o casamento $c = (i,j)$, e $z_{ij} = 0$ caso contrário.

$$\begin{array}{lrcll} \max & \displaystyle\sum_{(i,j)\in\mathscr{C}} z_{ij} & & & \\ \text{s. a} & z_{ij} + z_{k\ell} & \leq & 1 & \text{para todo } (i,j) \text{ e } (k,\ell) \text{ em } \mathscr{C} \text{ que se cruzam,} \\ & z_{ij} & \in & \{0,1\} & \text{para todo } (i,j) \text{ em } \mathscr{C}. \end{array}$$

A formulação acima nos permite construir um algoritmo baseado em programação inteira para encontrar um conjunto de casamentos de cardinalidade máxima sem cruzamento e, portanto, resolver o problema do LCS.

Vimos anteriormente que existem algoritmos polinomiais para se resolver o problema do LCS. Entretanto, problemas de programação inteira são, em geral, NP-difíceis, e seria interessante se pudéssemos trocar a restrição de integralidade por alguma restrição linear similar (como $z_{ij} \in \{0,1\}$ por $0 \leq z_{ij} \leq 1$). Isso reduziria o problema a um problema de programação linear, para o qual existe algoritmo polinomial.

No entanto, com a formulação acima, não podemos simplesmente fazer essa troca. De fato, considere o caso simples em que temos três casamentos, digamos com variáveis z_1, z_2, z_3, que se cruzam dois a dois. As restrições são $z_1 + z_2 \leq 1$, $z_1 + z_3 \leq 1$ e $z_2 + z_3 \leq 1$. No entanto, ao tentarmos maximizar $\sum_{(i,j)\in\mathscr{C}} z_{ij}$ sem considerar a restrição $z_{ij} \in \{0,1\}$ para todo $(i,j) \in \mathscr{C}$, temos que $(z_1, z_2, z_3) = \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$, como mostrado na Figura 2.5(a). É fácil ver que esse ponto satisfaz as restrições e maximiza a soma, mas ele é um ponto fracionário e, portanto, não define uma subseqüência comum máxima. Ainda mais, podemos eliminar esse ponto do poliedro com a restrição $z_1 + z_2 + z_3 \leq 1$, como na Figura 2.5(b). Nesse caso, existem soluções viáveis inteiras que maximizam a soma, como $(1,0,0)$.

Assim, somos motivados a procurar uma formulação melhor do ponto de vista de combinatória poliédrica (uma noção do que isso significa foi dada na Seção 1.1). Para isso, prosseguimos o estudo da estrutura do problema, considerando dois grafos que o representam.

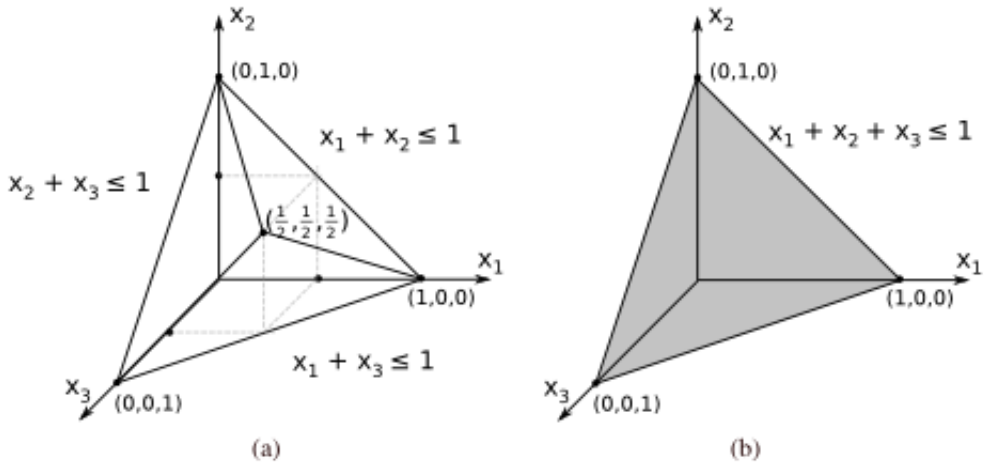

Figura 2.5: Poliedros válidos para o problema do LCS. (a) Com vértice fracionário $\left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$. (b) Com restrição que elimina esse vértice.

2.3 Grafos relacionados

Os grafos que definimos são baseados no conceito de cruzamento.

Denominamos $G_C = (V_C, E_C)$ de **grafo de cruzamento**. O conjunto de vértices V_C é o conjunto de casamentos $\mathscr{C}$. Uma aresta cd pertence a E_C se e somente se c e d se cruzam. Denotamos também $G_{NC} = (V_{NC}, E_{NC})$ de **grafo de não-cruzamento**, e ele é definido como o grafo complementar de G_C, isto é, V_{NC} também é $\mathscr{C}$ e uma aresta cd pertence a E_{NC} se e só se c e d não se cruzam. As Figuras 2.6(a) e 2.6(b) ilustram esses grafos.

Com esses grafos em mente, lembremos que um LCS pode ser representado como um conjunto de casamentos de cardinalidade máxima sem cruzamento. Em G_{NC}, isso equivale a um conjunto de vértices adjacentes dois a dois de cardinalidade máxima, que é exatamente a definição de um clique máximo (veja Figura 2.7). O problema de encontrar um clique máximo em um grafo qualquer, no entanto, é NP-difícil.

Lembremos da relação $\prec_\times$ ($c \prec_\times d$ se e só se ($i \leq k$ e $j \geq \ell$)). É fácil ver que a relação $\prec_\times$ é reflexiva ($c \prec_\times c$), antisimétrica ($c \prec_\times d$ e $d \prec_\times c$, então $c = d$) e transitiva ($c \prec_\times d$ e $d \prec_\times e$, então $c \prec_\times e$). Portanto, $\prec_\times$ é uma relação de ordem parcial. Um grafo definido por uma relação de ordem parcial como G_C (c e d são adjacentes se e só se $c \prec_\times d$ ou $d \prec_\times c$) é chamado de **grafo de comparabilidade**. O mesmo é verdade para G_{NC}, pois a relação $\prec_{||}$ é também uma relação de ordem parcial, apesar de estrita.

Considere a orientação $\vec{\mathcal{G}}_{NC}$ de G_{NC} onde c vai para d se $c \prec_{||} d$, como na Figura 2.8(a). Como essa orientação é transitiva, os vértices de todo caminho orientado

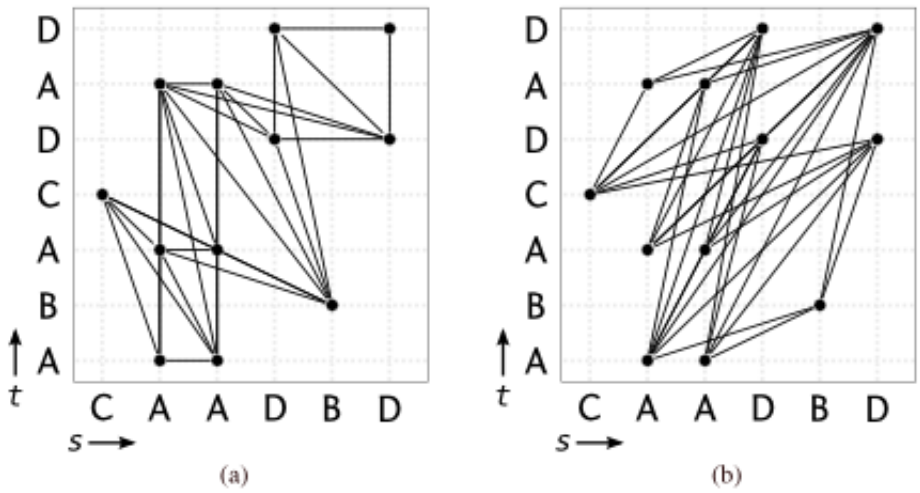

Figura 2.6: (a) Exemplo de grafo de cruzamento G_C. (a) Exemplo de grafo de não-cruzamento G_{NC}.

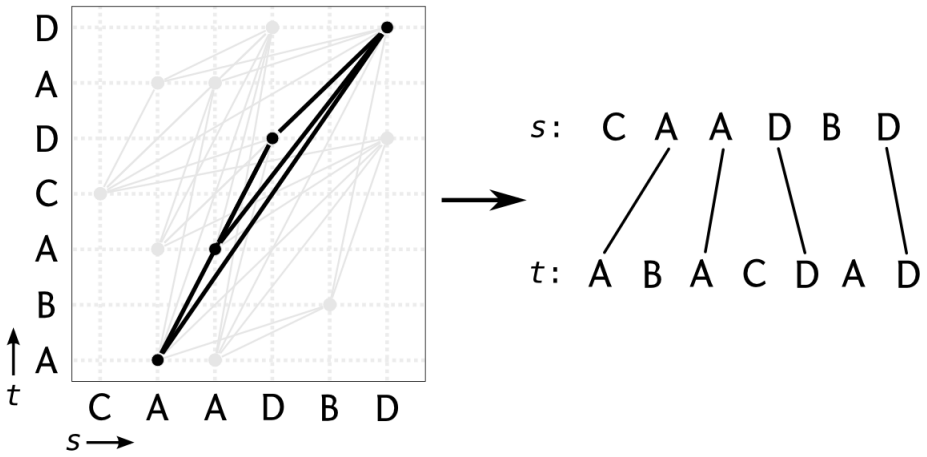

Figura 2.7: Um clique máximo no grafo de não-cruzamento G_{NC}.

formam um clique. Portanto, um caminho orientado de comprimento máximo define um clique máximo, ou seja, define um LCS (veja Figura 2.8(b)). Isso consiste com o fato que existe algoritmo polinomal para se resolver o problema do LCS, já que existe algoritmo polinomial para se encontrar um caminho orientado de comprimento máximo em um grafo orientado acíclico como G_{NC}.

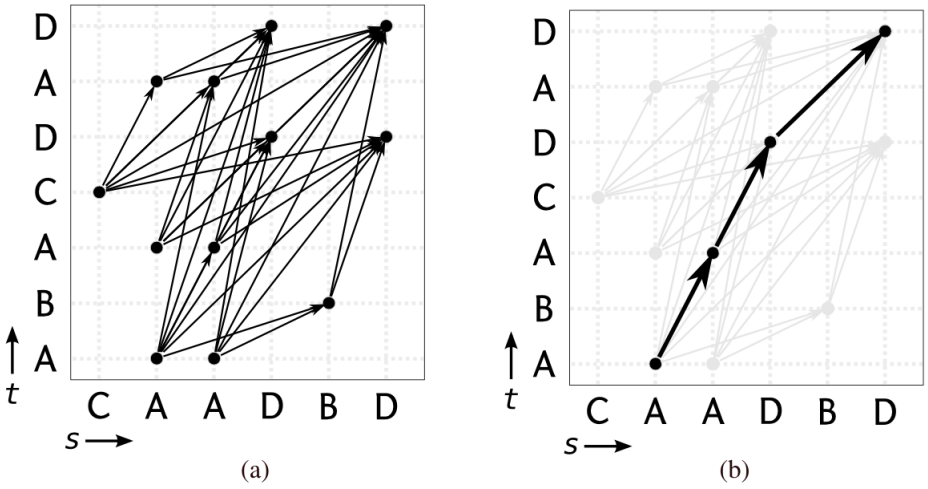

Figura 2.8: (a) Orientação de G_{NC}. (b) Caminho orientado que define um LCS.

Tome o grafo G_C. Um clique em G_C é um conjunto de casamentos que se cruzam dois a dois. Denominamos tal conjunto de **estrela**. Nos interessamos em **estrelas maximais** (cliques maximais em G_C), isto é, um conjunto de casamentos que se cruzam dois a dois tal que não seja possível adicionar mais nenhum casamento ao conjunto de forma que ele continue sendo uma estrela. A Figura 2.9 mostra um exemplo de uma estrela maximal. Como G_C também é um grafo de comparabilidade e, portanto, tem uma orientação transitiva, essas estrelas podem ser encontradas procurando caminhos orientados de comprimento maximal. Vemos na seção seguinte que podemos usar a idéia de estrelas maximais para construir uma formulação melhor.

2.4 Uma formulação melhor

Lembremos da definição de estrela maximal na seção anterior: um conjunto maximal de casamentos que se cruzam dois a dois. Lembremos também que, dadas as seqüências s e t, um conjunto C de casamentos sem cruzamento em s e t define um LCS de s e t. Portanto, um conjunto C' de casamentos tal que, para cada estrela maximal em $\mathscr{C}$, no máximo um casamento pertence a C', também define um LCS.

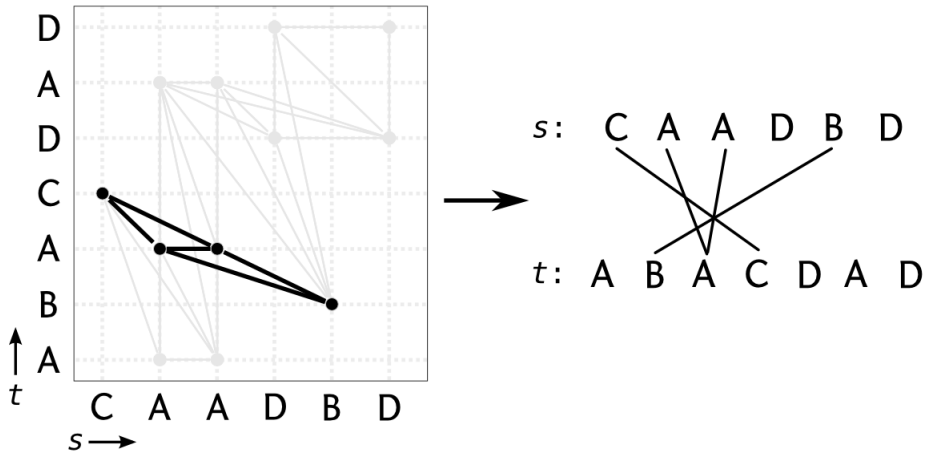

Figura 2.9: Um clique maximal no grafo de cruzamento G_C.

De fato, C' também é um conjunto de casamentos sem cruzamento.

Isso nos leva à seguinte formulação, explicada em palavras primeiro.

maximizar	o número de casamentos escolhidos
sujeito às restrições	para cada estrela maximal, podemos escolher no máximo um casamento dela;
	para cada casamento, escolhemos ele ou 0 ou 1 vez.

Reescrevemos matematicamente a formulação a seguir.

Assim como na formulação anterior, tome $\mathscr{C}$ o conjunto de todos os casamentos de s e t, e seja $z \in \{0,1\}^{\mathscr{C}}$ um vetor tal que $z_{ij} = 1$ se escolhemos o casamento $c = (i,j)$, e $z_{ij} = 0$ caso contrário.

$$\begin{aligned} \max \quad & \sum_{(i,j)\in\mathscr{C}} z_{ij} \\ \text{s. a} \quad & \sum_{(i,j)\in S} z_{ij} \;\leq\; 1 \qquad \text{para toda estrela maximal } S \text{ contida em } \mathscr{C}, \\ & z_{ij} \;\in\; \{0,1\} \quad \text{para todo } (i,j) \text{ em } \mathscr{C}. \end{aligned}$$

Lembremos que, do ponto de vista de combinatória poliédrica, a melhor formulação de programação inteira para um certo problema é aquela cujo poliedro associado é o casco convexo dos vértices. A seguir, definimos um poliedro que é o casco convexo das soluções viáveis do problema do LCS (subseqüências comuns), e mostramos que esse poliedro é igual ao poliedro associado a uma relaxação linear da formulação

acima, isto é, trocando as restrições de integralidade ($z_{ij} \in \{0,1\}$) por restrições lineares análogas ($z_{ij} \geq 0$).

2.5 Poliedro do LCS

Definimos o poliedro do LCS como

$$P_{\text{LCS}} := \text{conv}\{z \in \{0,1\}^{\mathscr{C}} \mid z \text{ representa uma subseqüência comum de } s \text{ e } t\},$$

isto é, o casco convexo de todas as subseqüências comuns de s e t.

É claro que encontrar um LCS é equivalente a maximizar $\sum_{(i,j)\in\mathscr{C}} z_{ij}$ restrito a P_{LCS}. Nesta seção, provamos que o poliedro baseado na relaxação linear da formulação anterior,

$$P_{form} = \{z \in \mathbb{R}^{\mathscr{C}} \quad | \quad \begin{array}{ll} \sum_{(i,j)\in S} z_{ij} \leq 1 & \text{para toda estrela maximal } S \subseteq \mathscr{C} \text{ e} \\ z_{ij} \geq 0 & \text{para todo } (i,j) \in \mathscr{C}\} \end{array}$$

é minimal e equivalente a P_{LCS}. Para isso, precisamos do conceito de faces e facetas.

Considere um poliedro $P \subseteq \mathbb{R}^n$ (no nosso caso, $n = |\mathscr{C}|$, pois temos $|\mathscr{C}|$ variáveis z_{ij}). As inequações de P podem ser dadas por restrições lineares de uma formulação. Dizemos que F define uma **face** de P se existe inequação válida $\pi x \leq \pi_0$ de P e F é o conjunto de pontos de P para os quais essa inequação vale com igualdade (isto é, vale $\pi x = \pi_0$ para todo $x \in F$). Nesse caso, dizemos que a inequação $\pi x \leq \pi_0$ define F. Se essa face F tem dimensão igual à dimensão do poliedro menos um ($\dim(F) = \dim(P) - 1$), então ela é uma **faceta**. Dimensão de X pode ser definida como $m - 1$ onde m é o número máximo de pontos afim independentes de X, e um conjunto de pontos $x_1, \ldots, x_k$ são afim independentes se os vetores $x_2 - x_1, \ldots, x_k - x_1$ ou os vetores $(x_1, 1), \ldots, (x_k, 1)$ são linearmente independentes. Um conjunto de vetores $x_1, \ldots, x_k$ são linearmente independentes se nenhum deles pode ser escrito como uma combinação linear dos outros, ou seja, se $a_1 x_1 + \ldots + a_k x_k = 0$, então $a_\ell = 0$ para todo ℓ. A Figura 2.10 ilustra uma face e uma faceta de um poliedro.

O conceito de faceta é relevante por causa da afirmação seguinte.

Fato 2.3. *Seja P um poliedro. Se P tem dimensão plena, uma inequação válida $\pi x \leq \pi_0$ é necessária na descrição de P se e somente se ela define faceta em P.*

$P \subseteq \mathbb{R}^n$ tem dimensão plena se tem n direções linearmente independentes. Todo poliedro de dimensão plena tem uma descrição única e minimal na qual toda desigualdade é única a menos de um múltiplo positivo. Assim, se P_{LCS} tiver dimensão plena e provarmos que todas as inequações de P_{LCS} definem facetas, então sabemos que elas são necessárias.

É fácil ver que P_{LCS} tem dimensão plena. Um vetor unitário denotado por e^k é tal que $e^k_k = 1$ e $e^k_\ell = 0$ para todo $k \neq \ell$. Os n vetores e^{ij}, para todo $ij \in \mathscr{C}$, pertencem a P_{LCS} (pois representam subseqüências comuns) e são linearmente independentes.

Provamos agora que as inequações dadas em P_{form} são facetas de P_{LCS}.

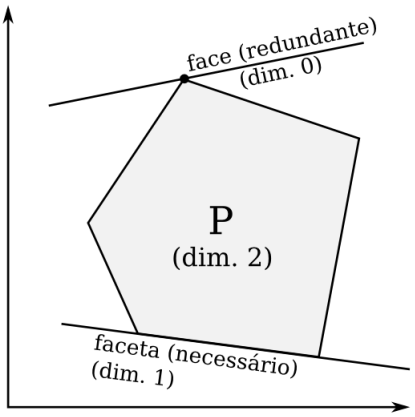

Figura 2.10: Exemplo de uma face e uma faceta em um poliedro P.

Fato 2.4. *A inequação $z_{ij} \leq 1$ define faceta de P_{LCS} para todo $(i,j) \in \mathscr{C}$.*

Prova. Fixe um $(i,j) \in \mathscr{C}$. Tome o vetor nulo e os $|\mathscr{C}| - 1$ vetores $e^{k\ell}$ para todo $(k,\ell) \in \mathscr{C}$, $(k,\ell) \neq (i,j)$. É fácil ver que esses vetores são linearmente independentes. Esses $|\mathscr{C}|$ vetores satisfazem a inequação com igualdade, isto é, $z_{ij} = 0$, e, portanto, a inequação define faceta. □

Provamos o fato a seguir de forma indireta. Escolhemos n pontos $z^1, \ldots, z^n \in P_{\text{LCS}}$ que satisfazem a inequação com igualdade, isto é, no caso, $\sum_{(i,j)\in S} z_{ij} = 1$. Supomos que os pontos estão em um hiperplano $az = \alpha$ para algum $a \in \mathbb{R}^{\mathscr{C}}, \alpha \in \mathbb{R}$, e mostramos que esse hiperplano deve ser a inequação com igualdade.

Fato 2.5. *A inequação $\sum_{(i,j)\in S} z_{ij} \leq 1$ define faceta de P_{LCS} para toda estrela maximal $S \subseteq \mathscr{C}$.*

Prova. Fixe uma estrela maximal $S \subseteq \mathscr{C}$. Observe primeiro que, como S é maximal, para todo $(k,\ell) \in \mathscr{C} \setminus S$, existe um $(i,j) \in S$ que não se cruza com (k,ℓ). Tome os pontos $e^{ij} + e^{k\ell}$ para todo $(k,\ell) \in \mathscr{C} \setminus S$ e um $(i,j) \in S$ que não se cruza com (k,ℓ). Também tome os pontos e^{ij} para todo $(i,j) \in S$. É fácil ver que esses pontos pertencem a P_{LCS} e satisfazem $\sum_{(i,j)\in S} z_{ij} = 1$. Considere um hiperplano $az = \alpha$ e suponha que esses n pontos estão nele. Então podemos substituir os pontos na equação. Assim, temos que $ae^{ij} = \alpha$, para todo $(i,j) \in S$, e $a(e^{ij} + e^{k\ell}) = \alpha$, para todo $(k,\ell) \in \mathscr{C} \setminus S$ e para algum $(i,j) \in S$ que não cruza com (k,ℓ). Aplicando a primeira equação na segunda, vale que $\alpha + ae^{k\ell} = \alpha$, isto é, $ae^{k\ell} = 0$, para todo $(k,\ell) \in \mathscr{C} \setminus S$. Portanto, $a_{ij} = \alpha, \forall (i,j) \in S$, e $a_{k\ell} = 0, \forall (k,\ell) \in \mathscr{C} \setminus S$. Logo, temos que $az = \alpha$ é o mesmo que $\alpha \sum_{(i,j)\in S} z_{ij} + 0 \sum_{(i,j)\in \mathscr{C}\setminus S} z_{ij} = \alpha$, ou seja, $\sum_{(i,j)\in S} z_{ij} = 1$. Portanto, a inequação $\sum_{(i,j)\in S} z_{ij} \leq 1$ define faceta. □

O teorema a seguir é o resultado mais interessante deste capítulo.

Teorema 2.1.

$$P_{\text{LCS}} = P_{form},$$

isto é,

$$\{z \in \mathbb{R}^{\mathscr{C}} \quad | \quad \begin{array}{ll} \sum_{(i,j)\in S} z_{ij} \leq 1 & \textit{para toda estrela maximal } S \subseteq \mathscr{C} \textit{ e} \\ z_{ij} \geq 0 & \textit{para todo } (i,j) \in \mathscr{C}\} \end{array}$$

é o casco convexo das soluções viáveis do problema do LCS *(as subseqüências comuns de s e t).*

Prova. Esta prova utiliza o seguinte teorema dado por Chvátal [7].

Antes, mostramos algumas definições necessárias para o teorema. Cocliques, ou conjuntos estáveis, são conjuntos de vértices que não são adjacentes dois a dois. Dizemos que G é um grafo perfeito se, para cada subgrafo induzido G' de G, o tamanho do clique máximo de G' é igual ao número mínimo de cores necessárias para colorir G' de forma que dois vértices adjacentes não tenham a mesma cor.

Teorema 2.2. *(Chvátal) Seja $G = (V, E)$ um grafo. Considere o poliedro*

$$P = \{x \in \mathbb{R}^V \quad | \quad \begin{array}{ll} \sum_{v\in S} x_v \leq 1 & \textit{para todo coclique } S \subseteq V \textit{ e} \\ x_v \geq 0 & \textit{para todo } v \in V\}. \end{array}$$

Então o poliedro de cliques é dado por P se e somente se G é perfeito, onde um poliedro de cliques é definido como o casco convexo dos cliques de G.

Uma prova desse teorema pode ser encontrada em [23].

Tome G_{NC} como na Seção 2.3 (G_{NC} é o grafo de não-cruzamento, onde os vértices são casamentos e dois casamentos são adjacentes se e só se eles não se cruzam). Observe que P é igual a P_{form} para o grafo G_{NC}: $V = \mathscr{C}$ e cocliques são estrelas. Provamos agora que G_{NC} é um grafo perfeito.

Vimos anteriormente que G_{NC} pode ser definido por uma relação de ordem parcial, isto é, G_{NC} é um grafo de comparabilidade. Mostramos então que qualquer grafo de comparabilidade é perfeito (isso sai diretamente do dual do teorema de Dilworth [9, 17], mas reescrevemos a prova no contexto de grafos).

Fato 2.6. *Todo grafo de comparabilidade é perfeito.*

Prova. Seja G um grafo de comparabilidade. Considere a orientação transitiva de G. Para cada vértice v, encontre o caminho orientado mais comprido, digamos de comprimento v_k, para o qual v é seu último vértice. É claro que o v_k máximo é o comprimento máximo de um caminho orientado e, portanto, é o tamanho de um clique máximo devido à transitividade de G. Também não é difícil ver que podemos colorir minimamente cada vértice com seu v_k tal que nenhum par de vértices adjacentes tenham a mesma cor (o número de cores é o v_k máximo). Portanto, o tamanho do clique máximo é igual ao número mínimo de cores necessárias para colorir G sem que dois vértices adjacentes tenham a mesma cor. Como isso vale para todos os subgrafos induzidos de G (pois um subgrafo induzido de um grafo de comparabilidade também é de comparabilidade), G é perfeito. □

Com isso, temos que G_{NC} é um grafo perfeito. Portanto, P_{form} é o poliedro dado pelo casco convexo dos cliques de G_{NC}. Isso não só diz que P_{form} tem vértices inteiros (o que já implica que $P_{form} = P_{\text{LCS}}$ pois P_{form} é válido para o problema do LCS), como também diz diretamente que P_{form} é o casco convexo das subseqüências comuns de s e t (um clique em G_{NC} é uma subseqüência comum). □

Uma prova alternativa mais direta para esse teorema pode ser encontrada em [10] (esta iniciação científica colaborou com o desenvolvimento dessa referência). Ela mostra que toda inequação que define faceta pode ser escrita como alguma das inequações de P_{form}.

Como conseqüência disso, na nossa formulação de programação inteira anterior, podemos trocar a restrição $z_{ij} \in \{0, 1\}$ por $z_{ij} \geq 0$, uma vez que a otimização dessa formulação sempre resulta em uma solução inteira (binária). Assim, a formulação de programação linear seguinte é válida e minimal para o problema do LCS.

$$\begin{aligned} \max \quad & \sum_{(i,j)\in\mathscr{C}} z_{ij} & & \\ \text{s. a} \quad & \sum_{(i,j)\in S} z_{ij} \;\leq\; 1 & & \text{para toda estrela maximal } S \text{ contida em } \mathscr{C}, \\ & z_{ij} \;\geq\; 0 & & \text{para todo } (i,j) \text{ em } \mathscr{C}. \end{aligned}$$

2.6 Dualidade

Em programação linear, um problema dual é um problema relacionado ao original (que chamamos de primal), que pode ser definido seguinte forma.

Dado o problema primal

$$\begin{aligned} \max \quad & cx \\ \text{s. a} \quad & Ax \leq b \\ & x \geq 0, \end{aligned}$$

seu dual é

$$\begin{aligned} \min \quad & yb \\ \text{s. a} \quad & yA \geq c \\ & y \geq 0. \end{aligned}$$

O teorema fraco da dualidade afirma que o valor da função objetivo do problema dual é sempre maior ou igual ao valor da função objetivo do problema primal (se for de maximização). O teorema forte da dualidade diz que o valor ótimo da função objetivo dos dois problemas são iguais.

Um problema de programação inteira também tem um par dual, o dual da relaxação linear dele, mas para ele não necessariamente vale o teorema forte da dualidade, embora valha o fraco. No caso do problema do LCS, o fato de que existe uma formulação de programação linear para ele implica que podemos obter um problema dual, para o qual vale que os valores ótimos das suas funções objetivos são iguais.

Consideramos a definição do par primal-dual no problema do LCS. Seja $\mathscr{S}$ o conjunto estrelas maximais em s e t, e $m = |\mathscr{S}|$, $n = |\mathscr{C}|$. A matriz A é uma matriz binária $m \times n$ cujas linhas representam estrelas maximais e colunas representam casamentos. O elemento $a_{ij} = 1$ se o casamento i pertence à estrela maximal j, ou $a_{ij} = 0$ caso contrário. Ademais, os vetores c de tamanho n e b de tamanho m são $(1, 1, \ldots, 1)$. Assim, podemos formular o dual do problema do LCS da seguinte forma:

Seja uma variável $y \in \mathbb{R}^m$. Apesar de y não ser binária, podemos ver y como $y_S = 1$ se escolhemos a estrela maximal S, ou $y_S = 0$ caso contrário.

$$\begin{aligned} \min \quad & \sum_{S \in \mathscr{S}} y_S & & \\ \text{s. a} \quad & \sum_{S \in K_c} y_S \geq 1 & & \text{para todo casamento } c \text{ em } \mathscr{C}, \\ & y_S \geq 0 & & \text{para todo } S \text{ em } \mathscr{S}, \end{aligned}$$

onde K_c é o conjunto de estrelas maximais que cobre c, isto é, $K_c = \{S \in \mathscr{S} \mid c \in S\}$.

O poliedro associado ao problema acima tem vértices inteiros, pois o problema primal também o tem. Ou seja, uma solução ótima do problema é inteira.

Assim, pelo teorema forte da dualidade, temos que o comprimento de um LCS equivale ao número mínimo de estrelas necessárias para cobrir todos os casamentos.

Ademais, podemos mostrar que essa afirmação é verdadeira sem usar programação linear. Um método é usar o teorema de Dilworth [9] ou seu dual [17], mas vamos aproveitar que provamos que G_{NC} é perfeito no final da Seção 2.5 (usando o dual do teorema de Dilworth). Relembrando da definição de grafo perfeito, G é perfeito se, para todo subgrafo induzido G' de G, o tamanho do clique máximo de G' é igual ao número mínimo de cores necessárias para cobrir G' de forma que não hajam dois vértices adjacentes com a mesma cor (que é o mesmo que o número mínimo de cocliques necessários para cobrir G'). Ou seja, como em G_{NC} um clique máximo representa um LCS e um coclique representa uma estrela, temos que o comprimento de um LCS é igual ao número mínimo de estrelas necessárias para cobrir os casamentos.

Note que como o complementar de um grafo perfeito é perfeito, isso também vale para G_C. Isto é, o número mínimo de conjuntos de casamentos necessários para cobrir todos os casamentos que definem subseqüências comuns é igual ao comprimento de uma estrela máxima.

Existem algoritmos para o problema do LCS que exploram essa dualidade [11].

Capítulo 3

Um estudo do RFLCS

3.1 Resultados conhecidos

O RFLCS é um problema pouco estudado na literatura.

Adi et al. [1] prova que o problema do RFLCS é APX-difícil mesmo se cada símbolo aparece no máximo duas vezes em cada uma das seqüências dadas. Isto é, não existe um algoritmo de aproximação de complexidade de tempo polinomial que garante uma solução perto da ótima para o problema do RFLCS (tal algoritmo é chamado de *polynomial-time approximation scheme*, ou PTAS). Conseqüentemente, o problema do RFLCS é NP-difícil. Fernandes et al. [10] realiza um estudo poliédrico do problema do RFLCS, assim como é feito neste texto. Esta iniciação científica contribuiu para o desenvolvimento dessas duas referências.

Bonizzoni et al. [4] estuda uma generalização dos problemas do LCS e do RFLCS, o problema da subseqüência comum máxima exemplar (*exemplar longest common subsequence*). Esse problema considera um conjunto de símbolos opcionais e um conjunto de símbolos obrigatórios e permite que, dependendo da versão do problema, haja exatamente uma ou pelo menos uma ocorrência dos símbolos obrigatórios, e no máximo uma ou um número irrestrito de ocorrências dos símbolos opcionais. Assim, o problema do LCS equivale ao que todos os símbolos são opcionais e há um número irrestrito de ocorrências deles, e o problema do RFLCS equivale ao que todos os símbolos são opcionais e há no máximo uma ocorrência de cada um. Um dos resultados nesse artigo é uma prova de que as versões que permitem no máximo uma ocorrência dos símbolos opcionais são APX-difícil mesmo no caso em que cada símbolo aparece no máximo duas vezes em cada seqüência.

3.2 Formulações para o RFLCS

Como já mencionado na introdução, o RFLCS é uma variante do LCS na qual cada símbolo do alfabeto pode aparecer no máximo uma vez na subseqüência comum que

procuramos.

Pela definição do problema, é natural modificar as formulações do LCS adicionando uma restrição nova que impede que dois casamentos com o mesmo símbolo associado sejam escolhidos.

maximizar	o número de casamentos escolhidos
sujeito às restrições	para cada símbolo do alfabeto, escolhemos no máximo um casamento associado a ele;
	(restrição de cruzamentos ou estrelas do LCS);
	para cada casamento, escolhemos ele ou 0 ou 1 vez.

Assim, as duas restrições a seguir são possíveis. A próxima formulação usa casamentos.

Seja $\mathscr{C}$ o conjunto de todos os casamentos de s e t e $\mathscr{C}(a)$ o conjunto de casamentos de s e t que têm o símbolo associado a. Seja $z \in \{0,1\}^{\mathscr{C}}$ um vetor tal que $z_{ij} = 1$ se escolhemos o casamento $c = (i,j)$, e $z_{ij} = 0$ caso contrário.

$$
\begin{array}{lrcll}
\max & \displaystyle\sum_{(i,j)\in\mathscr{C}} z_{ij} & & & \\
\text{s. a} & \displaystyle\sum_{(i,j)\in\mathscr{C}(a)} z_{ij} & \leq & 1 & \text{para todo } a \text{ no alfabeto,} \\
& z_{ij} + z_{k\ell} & \leq & 1 & \text{para todo } (i,j) \text{ e } (k,\ell) \text{ em } \mathscr{C} \text{ que se cruzam,} \\
& z_{ij} & \in & \{0,1\} & \text{para todo } (i,j) \text{ em } \mathscr{C}.
\end{array}
$$

A seguir, temos a formulação usando estrelas.

$$
\begin{array}{lrcll}
\max & \displaystyle\sum_{(i,j)\in\mathscr{C}} z_{ij} & & & \\
\text{s. a} & \displaystyle\sum_{(i,j)\in\mathscr{C}(a)} z_{ij} & \leq & 1 & \text{para todo } a \text{ no alfabeto,} \\
& \displaystyle\sum_{(i,j)\in S} z_{ij} & \leq & 1 & \text{para toda estrela maximal } S \text{ contida em } \mathscr{C}, \\
& z_{ij} & \in & \{0,1\} & \text{para todo } (i,j) \text{ em } \mathscr{C}.
\end{array}
$$

Assim como foi feito no problema do LCS, vale a pena procurar uma formulação melhor. No entanto, o problema do RFLCS é NP-difícil. Como conseqüência disso, a menos que P = NP, não é possível descrever explicitamente o casco convexo das soluções viáveis inteiras do problema como foi feito para o do LCS. Mesmo assim, ainda podemos procurar uma formulação melhor.

A definição de estrela para o problema do LCS é um conjunto de casamentos que se cruzam dois a dois. Estendemos essa definição. Definimos **estrela estendida** como

um conjunto de casamentos que ou se cruzam ou têm o mesmo símbolo associado dois a dois. Assim, uma estrela estendida maximal é um conjunto maximal com essa propriedade, isto é, para qualquer casamento de símbolo a de fora da estrela estendida maximal, existe algum casamento de símbolo diferente de a que esteja nessa estrela e que não se cruze com ela.

Figura 3.1: Exemplo de uma estrela estendida maximal.

Dois casamentos distintos que pertencem ao conjunto de casamentos que define um RFLCS não podem pertencer a uma mesma estrela estendida, pois, caso contrário, eles se cruzariam ou teriam o mesmo símbolo associado. Portanto, a seguinte formulação é válida.

Seja $\mathscr{C}$ o conjunto de todos os casamentos de s e t e $\mathscr{C}(a)$ o conjunto de casamentos de s e t que têm o símbolo associado a. Seja $z \in \{0,1\}^{\mathscr{C}}$ um vetor tal que $z_{ij} = 1$ se escolhemos o casamento $c = (i,j)$, e $z_{ij} = 0$ caso contrário.

$$
\begin{aligned}
\max \quad & \sum_{(i,j)\in\mathscr{C}} z_{ij} & & \\
\text{s. a} \quad & \sum_{(i,j)\in\mathscr{C}(a)} z_{ij} \;\leq\; 1 & & \text{para todo } a \text{ no alfabeto,} \\
& \sum_{(i,j)\in S} z_{ij} \;\leq\; 1 & & \text{para toda estrela estendida maximal } S \subseteq \mathscr{C}, \\
& z_{ij} \;\in\; \{0,1\} & & \text{para todo } (i,j) \text{ em } \mathscr{C}.
\end{aligned}
$$

Observe que, em particular, o conjunto $\mathscr{C}(a)$ é uma estrela estendida. Portanto, as restrições de que só pode ter uma ocorrência de cada símbolo são redundantes. Assim, a formulação a seguir é válida.

$$
\begin{aligned}
\max \quad & \sum_{(i,j)\in\mathscr{C}} z_{ij} & & \\
\text{s. a} \quad & \sum_{(i,j)\in S} z_{ij} \;\leq\; 1 & & \text{para toda estrela estendida maximal } S \subseteq \mathscr{C}, \\
& z_{ij} \;\in\; \{0,1\} & & \text{para todo } (i,j) \text{ em } \mathscr{C}.
\end{aligned}
$$

Agora, provamos que a restrição $\sum_{(i,j)\in S} z_{ij} \leq 1$ para toda estrela estendida maximal S é necessária para o problema do RFLCS, isto é, ela define faceta de P_{RFLCS}. Definimos

$$P_{\text{RFLCS}} := \text{conv}\{z \in \{0,1\}^{\mathscr{C}} \mid \quad z \text{ representa uma subseqüência comum sem repetições de } s \text{ e } t\}.$$

Usamos o mesmo método indireto usado na Seção 2.5 para provar que as restrições de estrela estendida definem facetas. A prova é análoga à prova para as restrições de estrela do LCS.

Fato 3.1. *A inequação $\sum_{(i,j)\in S} z_{ij} \leq 1$ define faceta de P_{RFLCS} para toda estrela estendida maximal $S \subseteq \mathscr{C}$.*

Prova. Fixe uma estrela estendida maximal $S \subseteq \mathscr{C}$. Como S é maximal, para todo $(k,\ell) \in \mathscr{C} \setminus S$ com símbolo associado a, existe um $(i,j) \in S$ com símbolo associado $b \neq a$ que não se cruza com (k,ℓ). Tome os pontos $e^{ij} + e^{k\ell}$ para todo $(k,\ell) \in \mathscr{C} \setminus S$ e um $(i,j) \in S$ que não se cruza com (k,ℓ) e tem símbolo associado diferente do símbolo associado a (k,ℓ), e os pontos e^{ij} para todo $(i,j) \in S$. É fácil ver que esses pontos pertencem a P_{RFLCS} e satisfazem $\sum_{(i,j)\in S} z_{ij} = 1$. Considere um hiperplano $az = \alpha$ e suponha que esses n pontos estão nele. Então podemos substituir os pontos na equação. Assim, $ae^{ij} = \alpha, \forall (i,j) \in S$, e $a(e^{ij} + e^{k\ell}) = \alpha, \forall (k,\ell) \in \mathscr{C} \setminus S$, $(i,j) \in S$ que não se cruza com (k,ℓ) e tem símbolo associado diferente do símbolo associado a (k,ℓ). Aplicando a primeira equação na segunda, vale que $\alpha + ae^{k\ell} = \alpha$, isto é, $ae^{k\ell} = 0$, para todo $(k,\ell) \in \mathscr{C} \setminus S$. Portanto, $a_{ij} = \alpha, \forall (i,j) \in S$, e $a_{k\ell} = 0, \forall (k,\ell) \in \mathscr{C} \setminus S$. Logo, temos que $az = \alpha$ é o mesmo que $\alpha \sum_{(i,j)\in S} z_{ij} + 0 \sum_{(i,j)\in \mathscr{C}\setminus S} z_{ij} = \alpha$, ou seja, $\sum_{(i,j)\in S} z_{ij} = 1$. Portanto, a inequação $\sum_{(i,j)\in S} z_{ij} \leq 1$ define faceta. □

Essa é a melhor formulação que obtivemos.

3.3 Grafos relacionados

Consideremos os grafos de cruzamento G_C e não-cruzamento G_{NC} como no problema do LCS (Seção 2.3). Lembremos que um grafo de cruzamento é aquele cujos vértices são os casamentos e dois casamentos são adjacentes se e só se eles se cruzam, e um grafo de não-cruzamento é o complementar desse grafo.

No caso do problema do RFLCS, vamos chamar esses grafos de G'_C e G'_{NC} respectivamente. Colorimos os grafos G'_C e G'_{NC} de forma que cada casamento é colorido com o símbolo associado a ele. As Figuras 3.2(a) e 3.2(b) representam uma tal coloração para G'_C e G'_{NC} respectivamente (como a figura é preto-e-branco, a coloração é representada através de símbolos).

Vimos que, em G'_C, um LCS pode ser representado por um clique máximo. Um RFLCS não pode ter dois casamentos de mesma cor. Assim, um RFLCS pode ser

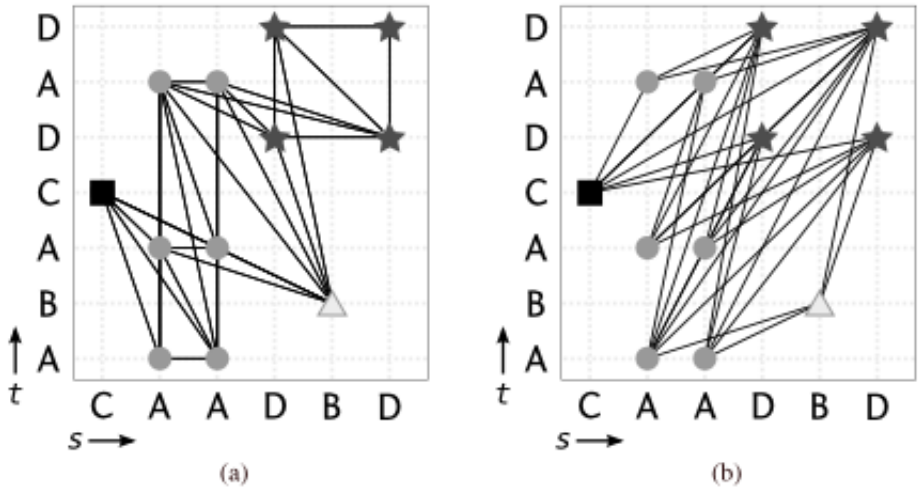

Figura 3.2: (a) Coloração do grafo G'_C. (b) Coloração do grafo G'_{NC}.

representado por um clique máximo que não tenha dois casamentos de mesma cor. Não perdemos generalidade ao afirmar que um clique mais colorido possível (isto é, com um número máximo de cores distintas) representa um RFLCS, uma vez que podemos eliminar casamentos de cores repetidas, deixando apenas um de cada cor. Da mesma forma que fizemos anteriormente com o problema do LCS, podemos considerar uma orientação tal que os cliques sejam dados por caminhos orientados. Assim, o problema do RFLCS pode ser reduzido para encontrar um caminho orientado mais colorido possível em G'_C colorido.

Vamos considerar dois grafos similares a G'_C e G'_{NC}. Definimos o grafo G_C para o problema do RFLCS como o grafo com o conjunto de vértices igual ao conjunto de casamentos e dois casamentos são adjacentes se e somente se ou eles se cruzam, ou eles têm o mesmo símbolo associado. Definimos também G_{NC} como o grafo complementar de G_C (dois casamentos são adjacentes se e só se eles não se cruzam nem têm o mesmo símbolo associado).

Esses dois grafos são mais interessantes que G'_C e G'_{NC} no sentido de que um clique máximo em G_{NC} (conjunto de casamentos que não se cruzam nem têm o mesmo símbolo associado dois a dois) representa um RFLCS, e um clique maximal em G_C (conjunto de casamentos que se cruzam ou têm o mesmo símbolo associado dois a dois) representa uma estrela estendida maximal.

Observemos agora algumas relações entre os grafos e a dificuldade do problema. Lembremos a prova da integralidade do poliedro via um teorema de Chvátal da Seção 2.5: o poliedro de cliques de um grafo G é dado pela formulação por cocliques se e somente se G é perfeito. O grafo G_{NC} é o grafo para o qual os cocliques são

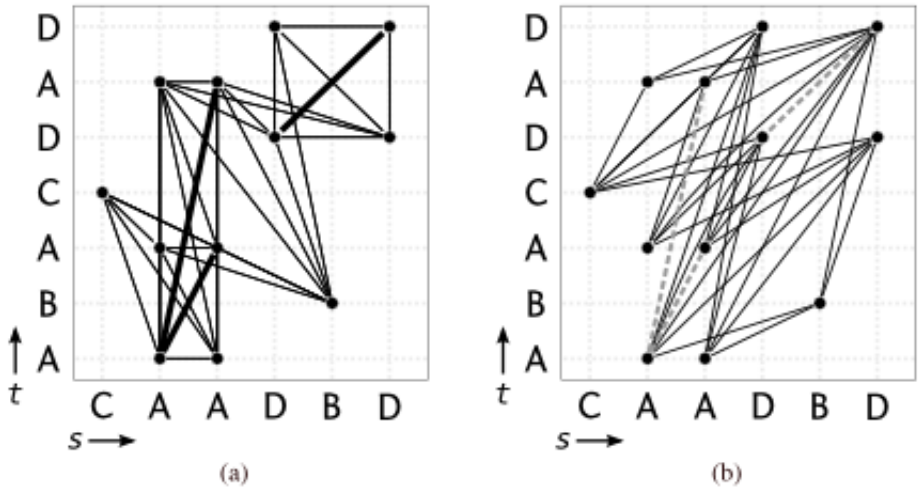

Figura 3.3: (a) Grafo G_C. (b) Grafo G_{NC}. Em destaque estão as diferenças dos grafos G_C e G_{NC} do LCS: em (a), as arestas novas, em (b), as arestas que não existem mais.

estrelas estendidas e os cliques são subseqüências comuns sem repetições. Portanto, dado P_{form} o poliedro associado à relaxação linear da formulação anterior, $P_{\text{RFLCS}} = P_{form}$ se e somente se G_{NC} é perfeito. O que isso diz é que, de uma certa forma, a dificuldade do problema do RFLCS é ditada pelo fato de G_{NC} ser ou não perfeito.

Um caso fácil simples é quando $G_{NC} = G'_{NC}$. Isso ocorre quando, para cada símbolo a, existe no máximo uma ocorrência de a ou em s ou em t. Na verdade, nesse caso, todo LCS de s e t é sem repetições.

Com essa idéia em mente, podemos encontrar um contra-exemplo para uma afirmação relacionada à dualidade do problema do RFLCS (verdadeira no caso do LCS): o comprimento de um LCS é igual ao número mínimo de estrelas estendidas necessárias para cobrir os casamentos.

Já vimos que se G_{NC} é perfeito, então vale a afirmação acima. A recíproca nem sempre é verdadeira, pois a definição de grafo perfeito envolve todos os subgrafos induzidos de G. Uma idéia é encontrar um grafo G_{NC} válido que não seja perfeito.

O teorema forte do grafo perfeito afirma que um grafo é perfeito se e somente se ele é um grafo de Berge [6]. Um grafo de Berge é um grafo que não contém nenhum buraco ímpar ou anti-buraco ímpar. Um buraco ímpar é um circuito induzido de comprimento ímpar maior ou igual a 5, e um anti-buraco ímpar é o complemento de um buraco ímpar. Mostramos então um grafo G_{NC} com anti-buraco ímpar de comprimento 7, ou melhor, para uma visualização mais fácil, um grafo G_C com buraco ímpar de comprimento 7.

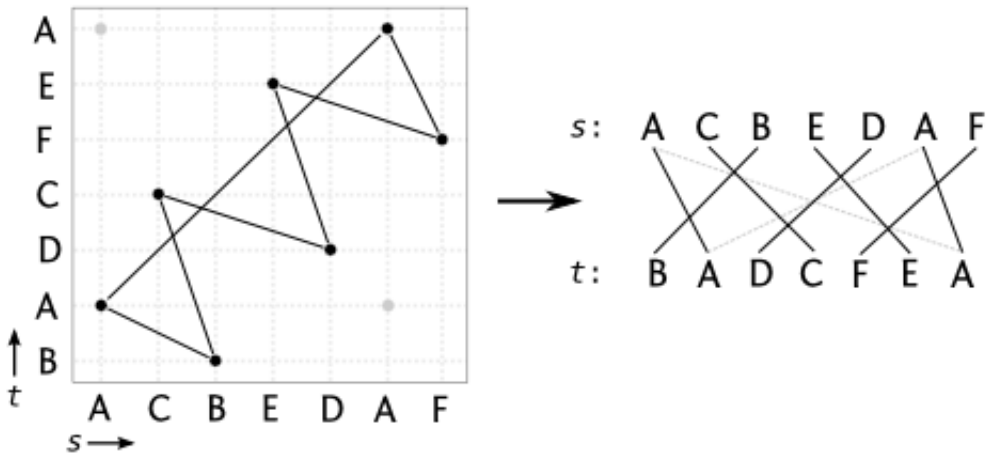

Figura 3.4: Um buraco ímpar de comprimento 7 em G_C.

Por inspeção, o comprimento de um RFLCS nesse exemplo é 3, e o número mínimo de estrelas estendidas necessárias para cobrir os casamentos é 4.

3.4 Limitantes

Nesta seção, discutimos alguns limitantes inferiores e superiores do problema do RFLCS.

Um limitante superior óbvio é o tamanho do alfabeto, pois não pode haver símbolos repetidos em um RFLCS. Outro limitante superior é o comprimento de um LCS das seqüências. De fato, um RFLCS também é um LCS e, portanto, o comprimento de um LCS é pelo menos o comprimento de um RFLCS.

Para encontrar limitantes inferiores, nos baseamos em algoritmos de aproximação.

Algoritmos de aproximação são algoritmos que procuram uma solução perto da solução ótima. Infelizmente, o problema do RFLCS é APX-difícil, o que significa que não há PTAS para o problema (a menos que P = NP), isto é, não há algoritmo de aproximação que garante uma solução perto da ótima e roda em tempo polinomial. Apesar disso, ainda vale a pena buscar algoritmos de aproximação que talvez sejam bons em média.

Um algoritmo de aproximação simples é rodar algum algoritmo qualquer que resolve o problema do LCS e eliminar símbolos repetidos, deixando apenas uma ocorrência de cada símbolo. É claro que esse algoritmo roda em tempo polinomial, supondo que o algoritmo para o LCS também roda em tempo polinomial.

Um segundo algoritmo de aproximação é probabilístico. Para cada símbolo a, encontre a seqüência s ou t que tem menos ocorrências de a (se for igual, escolha

uma qualquer). Nessa seqüência, escolha aleatoriamente (de forma uniforme) uma das ocorrências de a e elimine o resto das ocorrências de a nessa seqüência. O LCS das seqüências resultantes s' e t' é um RFLCS.

Um terceiro algoritmo de aproximação é uma variante do anterior que usa menos bits aleatórios. Um número é escolhido no intervalo entre 0 e 1 e fazemos o mesmo que o segundo algoritmo, mas, na escolha das ocorrências, esse número é usado. Para desaleatorizar esse algoritmo, podemos considerar o intervalo de 0 a 1 e dividi-lo em faixas, onde cada faixa representa as ocorrências dadas por esse número aleatório. Assim, aplicamos o algoritmo para as ocorrências de todas as faixas e pegamos o melhor resultado. É claro que esse algoritmo desaleatorizado é sempre melhor ou igual ao algoritmo original.

Seja $m(a)$ o menor entre o número de ocorrências de a em s e o número de ocorrências de a em t. Seja m o maior $m(a)$ para todo símbolo a do alfabeto. Esses três algoritmos são m-aproximações, isto é, o comprimento da seqüência que cada algoritmo devolve é maior que $\frac{1}{m}$ do comprimento de um RFLCS [1].

Outro algoritmo de aproximação probabilístico se baseia no estado atual do problema. Construímos uma matriz de probabilidade baseado na solução viável atual, onde damos pesos para cada casamento, e, usando essa matriz, escolhemos de forma análoga aos dois algoritmos anteriores as ocorrências dos símbolos que queremos manter.

Esses algoritmos de aproximação probabilísticos são rodados várias vezes e o melhor dos resultados é selecionado, para se obter uma solução boa.

Capítulo 4

Implementação e resultados experimentais

Um algoritmo exato para resolver o problema do RFLCS foi implementado usando o pacote GLPK (GNU Linear Programming Kit), que é uma biblioteca de funções em linguagem C para a resolução de problemas de programação linear. O GLPK inclui implementações do simplex, do método de pontos interiores e de funcionalidades para programação inteira, além de ser um resolvedor *stand-alone* de problemas de programação linear e inteira. Ele foi escolhido por ser o pacote de programação linear livre mais conhecido. Existem pacotes comerciais, como o CPLEX, que provavelmente utilizam técnicas melhores.

Neste capítulo, discutimos a implementação desse algoritmo e, ao fim, observamos alguns resultados experimentais.

4.1 Algoritmo *branch and cut*

O algoritmo é baseado na técnica *branch and cut*, que é uma extensão do *branch and bound*. Explicamos o *branch and bound* primeiro. Tudo o que é explicado nesta seção já está implementado no GLPK.

O *branch and bound* é uma técnica para se resolver de forma exata um problema de programação inteira decompondo-o em problemas de programação linear. Podemos construir uma árvore que enumera os subproblemas de um problema, onde em cada subproblema fixamos uma variável, como na Figura 4.1.

Realizar tal enumeração de forma completa é difícil, pois o número de vértices da árvore cresce muito rapidamente. A técnica *branch and bound* é uma enumeração implícita dessa árvore. A idéia do algoritmo é construir uma árvore, armazenando para cada vértice os limitantes superiores e inferiores do problema do vértice, e usar esses limitantes para podar a árvore, isto é, desconsiderar vértices (e conseqüentemente ramos inteiros) que sabemos que não precisam ser examinados. A ramificação

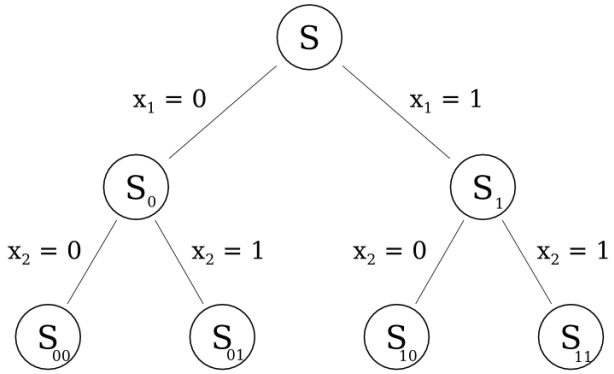

Figura 4.1: Uma árvore de enumeração com duas variáveis binárias x_1 e x_2.

de um vértice é feita escolhendo uma variável do problema, digamos x, e, para algum inteiro k, criamos um vértice filho em que $x \leq k$, e outro em que $x \geq k+1$. Existem vários detalhes e estratégias para esse algoritmo que estão fora do escopo deste texto; dentre eles, a escolha dessa variável. Usamos as heurísticas e estratégias implementadas no GLPK.

O *branch and cut* estende o *branch and bound* permitindo a inclusão de planos de corte no problema durante a execução do algoritmo. Planos de corte são restrições novas que "cortam" fora alguma solução fracionária, melhorando o aspecto poliédrico do problema.

No caso do problema do RFLCS, temos um número exponencial no tamanho do problema de estrelas estendidas maximais. Isto é, temos um número exponencial de restrições, o que nos impede de colocar todas as restrições no começo e rodar um algoritmo *branch and bound*. Porém, nem todas as restrições são necessárias.

Assim, fazemos o seguinte. Começamos com nenhuma restrição ou com alguma restrição simples. Após o passo de encontrar uma solução ótima para a relaxação linear, procuramos uma estrela estendida maximal que viole a solução ótima atual. Se houver, adicionamos o plano de corte dessa estrela estendida maximal, reotimizamos o problema, e procuramos outra estrela estendida maximal que viole a solução. Se não houver, prosseguimos com o problema.

O problema de encontrar tal estrela estendida maximal é chamado de **problema da separação**, que tratamos na próxima seção.

Para mais informações sobre essas técnicas, vale a pena procurar um bom livro de programação inteira. As explicações acima são baseadas no livro [26].

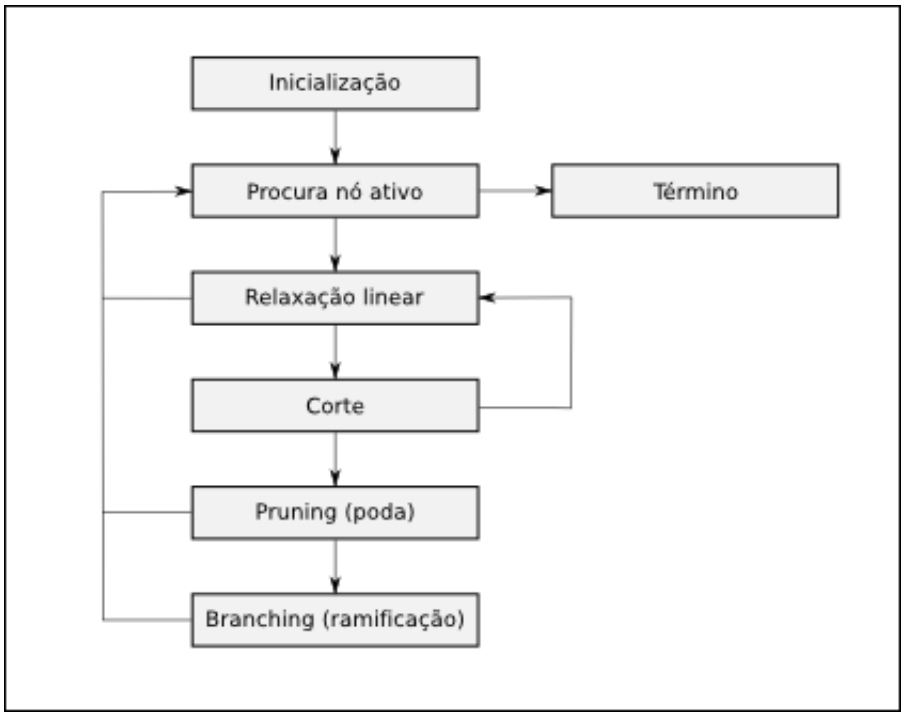

Figura 4.2: Diagrama de fluxo de um algoritmo *branch and cut*.

4.2 Problema da separação

O problema da separação consiste em, dado um ponto fracionário, encontrar um plano de corte (uma inequação) que a elimine do poliedro associado ao problema. Como explicado na seção anterior, desejamos encontrar uma estrela maximal (no caso do LCS) ou estrela estendida maximal (no caso do RFLCS) tal que a restrição associada a ela é violada por um certo ponto, que é uma solução viável da formulação anterior.

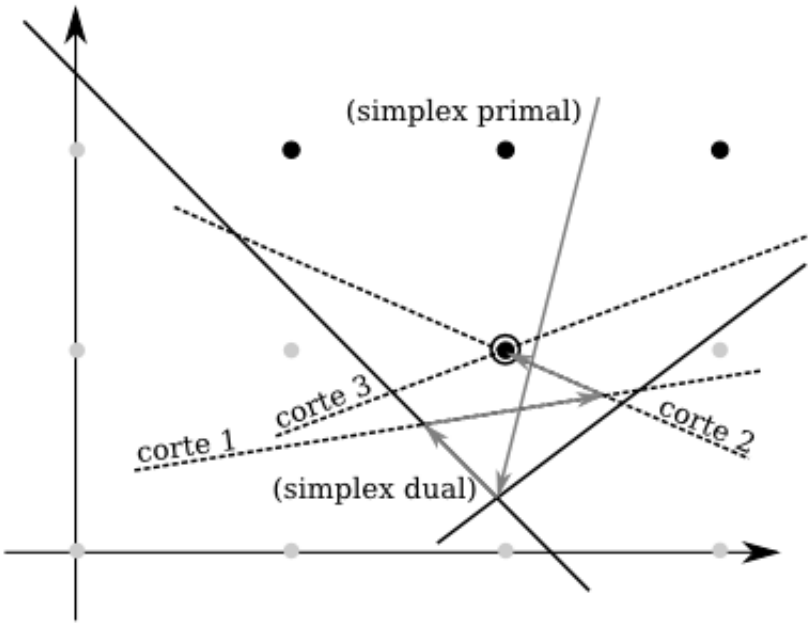

Figura 4.3: Exemplo de planos de corte que, nesse caso, resultam em uma solução inteira viável para a direção de otimização dada.

Tratamos primeiro o problema do LCS. Sejam s e t as seqüências do problema e z o ponto viável para o qual queremos verificar se existe uma restrição de estrela maximal violada por ele. É claro que $0 \leq z_{ij} \leq 1$ para todo $(i, j) \in \mathscr{C}$. Considere a seqüência s reverso, s^r, isto é, s com a ordem dos símbolos invertida. Um LCS de s^r e t para o qual permitimos casamentos com mesma ponta é uma estrela de cardinalidade máxima em s e t, como ilustrado pela Figura 4.4. De fato, a ordem relativa dos elementos em s é invertida em s^r. Além disso, os casamentos com mesma ponta também entram na estrela máxima como deveriam.

Assim, modificamos o algoritmo de programação dinâmica para o LCS para levar em conta z como os pesos e permitir casamentos com mesmo extremo.

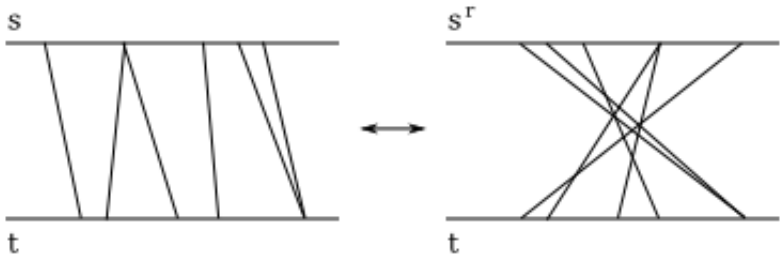

Figura 4.4: O reverso de uma seqüência e a estrela máxima resultante.

Primeiro, considere apenas os pesos. A recursão da Seção 2.1 se torna:

$$R(i,j) = \begin{cases} 0, & \text{se } i = 0 \text{ ou } j = 0 \\ \max\{R(i-1,j-1) + z_{ij}, R(i-1,j), R(i,j-1)\} & \text{se } s_i^r = t_j \\ \max\{R(i-1,j), R(i,j-1)\}, & \text{se } s_i^r \neq t_j \end{cases}$$

Com essa recursão, o algoritmo devolve uma subseqüência comum de s^r e t que maximiza a soma dos pesos (dado por z) dos casamentos escolhidos. Agora, para permitir casamentos com mesma ponta, a cada vez que $s_i^r = t_j$, olhamos não só para o próprio casamento, mas também para os casamentos de mesmo extremo que vêm antes. Em termos da matriz de programação dinâmica, eles são os elementos (k,ℓ) da mesma linha ou da mesma coluna que (i,j) em que $s_k^r = t_\ell$.

$$R(i,j) = \begin{cases} 0, & \text{se } i = 0 \text{ ou } j = 0 \\ \max\{m(i,j), R(i-1,j), R(i,j-1)\} & \text{se } s_i^r = t_j \\ \max\{R(i-1,j), R(i,j-1)\}, & \text{se } s_i^r \neq t_j \end{cases}$$

onde

$$\begin{array}{rcl} m(i,j) & = & \max\{m_{lin}(i,j), m_{col}(i,j)\}, \\ m_{lin}(i,j) & = & \max_{1 \leq \ell \leq j}\{R(i-1,\ell-1) + \sum_{(p,q) \in S_{lin}(\ell,i,j)} z_{pq}\}\}, \\ & & \text{com } S_{lin}(\ell,i,j) = \{(i,q) \in \mathscr{C} \mid \ell \leq q \leq j\}, \\ m_{col}(i,j) & = & \max_{1 \leq k \leq i}\{R(k-1,j-1) + \sum_{(p,q) \in S_{col}(k,i,j)} z_{pq}\}, \\ & & \text{com } S_{col}(k,i,j) = \{(p,j) \in \mathscr{C} \mid k \leq p \leq i\}. \end{array}$$

Em outras palavras, $m(i,j)$ é a melhor escolha do (k,ℓ) (em termos da soma dos pesos dos casamentos entre (i,j) e (k,ℓ)) que está ou na mesma linha, ou na mesma coluna que o elemento (i,j).

Logo, o resultado do algoritmo de programação dinâmica é uma estrela S de s e t que maximiza essa soma. Essa soma é $\sum_{(i,j) \in S} z_{ij}$ e, portanto, se ela for maior que 1, então a restrição dessa estrela é violada por z. Caso contrário, não existe restrição de estrela violada por z, pois maximizamos essa soma.

É um tanto estranho usar um algoritmo para o problema do LCS para resolver o problema da separação para o LCS. No entanto, para o RFLCS, um método baseado nele para resolver o problema da separação é interessante, pois ele é polinomial.

O problema da separação para o RFLCS pode ser resolvido de forma similar. Sejam s e t as seqüências do problema e z o ponto viável que queremos separar. Uma estrela estendida em s e t equivale a um conjunto de estrelas estendidas disjuntas em s^r e t. Na verdade, para cada uma dessas estrelas estendidas S, os símbolos associados aos casamentos de S são iguais, isto é, eles formam bipartidos completos, como ilustrado pela Figura 4.5.

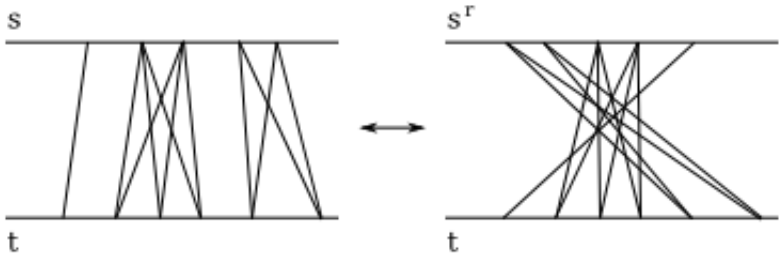

Figura 4.5: O reverso de uma seqüência e a estrela estendida máxima resultante.

Podemos modificar o algoritmo de programação dinâmica para encontrar esse conjunto de estrelas estendidas disjuntas cuja soma dos pesos seja máxima. Vale a seguinte recursão:

$$R(i,j) = \begin{cases} 0, & \text{se } i = 0 \text{ ou } j = 0 \\ \max\{m_a(i,j), R(i-1,j), R(i,j-1)\} & \text{se } s^r_i = t_j = a \\ \max\{R(i-1,j), R(i,j-1)\}, & \text{se } s^r_i \neq t_j \end{cases}$$

onde

$$\begin{array}{lcl} m_a(i,j) & = & \max_{1\leq k\leq i, 1\leq \ell\leq j}\{R(k-1,\ell-1) + \sum_{(p,q)\in S_a(k,i,\ell,j)} z_{pq}\} \text{ e} \\ S_a(k,i,\ell,j) & = & \{(p,q) \in \mathscr{C} \mid k \leq p \leq i,\ \ell \leq q \leq j,\ (p,q) \text{ tem símbolo } a\}. \end{array}$$

Observe que S_a é uma dessas estrelas estendidas de mesmo símbolo. A idéia da recursão é considerar essas estrelas, formando um conjunto delas. Note que a cada vez que $s^r_i = t_j$, isto é, (i,j) é um casamento, procuramos a estrela estendida S_a de maior peso que inclui (i,j) e vem antes dele. A Figura 4.6 descreve melhor a recursão acima no caso $s^r_i = t_j$.

O algoritmo de programação dinâmica que usa a recursão acima tem complexidade de tempo $O(n^2m^2)$ e de espaço $O(nm)$. Tanto no caso do LCS como no do RFLCS, para recuperar as estrelas estendidas disjuntas (ou conjunto de casamentos

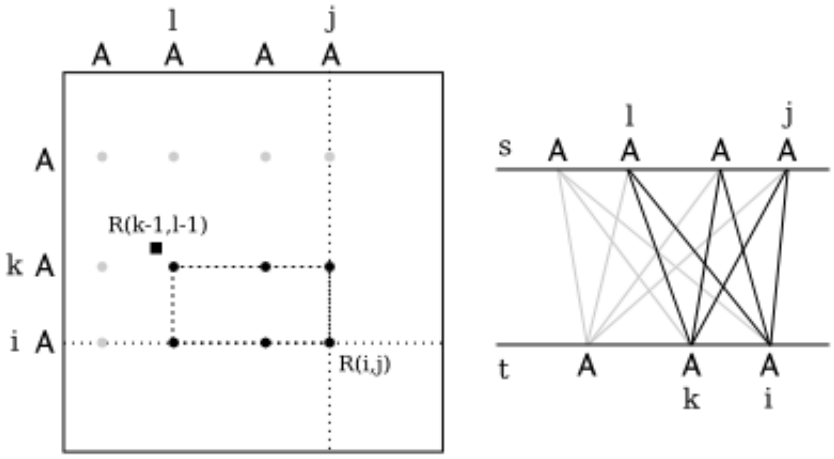

Figura 4.6: Uma matriz de programação dinâmica. Estamos olhando para o elemento (i, j) e considerando um (k, ℓ), que define uma estrela estendida até (i, j). A recursão escolhe o (k, ℓ) que maximiza o peso da estrela.

de mesma ponta no caso do LCS) a partir da matriz de programação dinâmica, precisamos armazenar os (k, ℓ) que maximizam a recursão no caso $s_i^r = t_j$. Cada um desses (k, ℓ) representa uma dessas estrelas estendidas.

Assim como no caso do LCS, se o resultado desse algoritmo for maior que 1, então a solução viável z viola a restrição dada pela estrela estendida maximal que achamos. Caso contrário, não existe estrela estendida cuja restrição é violada por z, pois ela é máxima.

Uma observação interessante sobre esses planos de corte é que eles equivalem a planos de corte já estudados na literatura, chamado cortes por clique [2]. Um grafo de conflito é um grafo no qual os vértices são as variáveis binárias e seus complementos. Ademais, dois vértices são adjacentes se e só se no máximo uma das variáveis representadas pelos vértices pode ser igual a 1. No caso de um problema de programação inteira genérico, podemos encontrar cliques maximais nesse grafo e gerar inequações novas a partir desses cliques. A Figura 4.7 representa um grafo de conflito.

No caso do LCS e do RFLCS, devido à aparência das restrições, podemos desprezar os vértices que representam o complemento da variável. O grafo resultante é G_C e os cliques maximais justamente são as estrelas maximais ou estrelas estendidas maximais.

No entanto, como os nossos cortes se aproveitam da estrutura do problema, eles são mais eficientes que esse método mais genérico.

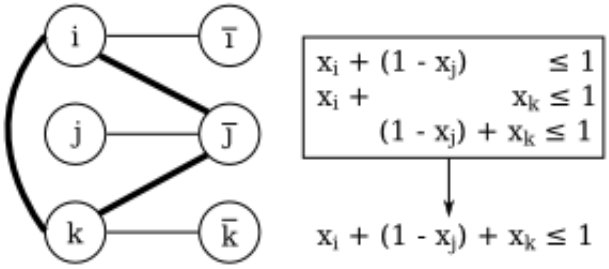

Figura 4.7: Um grafo de conflito. As arestas destacadas formam um clique.

4.3 Pré-processamento

Antes de rodar o algoritmo baseado em *branch and cut*, as seqüências são pré-processadas. Um método para pré-processar as seqüências é descrito abaixo.

Fato 4.1. *Dada uma seqüência qualquer z, seja $x = z^k$ com $k \geq |z|$ (z^k denota z repetido consecutivamente k vezes) e $x' = z^\ell a$ com $\ell = |z| - 1$ e a é o primeiro símbolo de z. Então y é uma subseqüência sem repetições de x se e só se ela é uma subseqüência sem repetições de x'.*

Por exemplo, para $|z| = 1$, podemos substituir aaaa por a, e para $|z| = 2$, podemos substituir abababab por aba. Um detalhe adicional é que a última repetição de z em x pode ser um prefixo de z ao invés do z inteiro; por exemplo, ababa ainda se torna aba.

Prova. Basta observar que x' contém todas as subseqüências sem repetição possíveis para o alfabeto composto pelos símbolos de x' (e conseqüentemente os símbolos de x).

De fato, vamos supor o pior caso, ou seja, $z = a_1 a_2 \ldots a_n$ tem todos os símbolos distintos, $n = |z|$. A subseqüência sem repetição que mais se "estende" em x é $z^r = a_n a_{n-1} \ldots a_1$. Mais formalmente, seja $m(z)$ o maior índice dos elementos de x que z escolhe da melhor forma possível (escolhendo os símbolos de z assim que eles aparecem da esquerda para direita). Então $m(z^r)$ é máximo. Vale que z^r é subseqüência sem repetições de x', pois z é repetido $n-1$ vezes (cobrindo $a_n \ldots a_2$) e há um a_1 adicional no fim de x'. Portanto, todas as subseqüências sem repetição de x também estão em x' (em particular, elas são todas do alfabeto dado pelos símbolos de z). □

Podemos ver x como um trecho de uma seqüência maior s e, portanto, um RFLCS de s e t é também um RFLCS de s' e t' eliminando essas repetições consecutivas como descrito acima. Na implementação, esse pré-processamento é feito apenas para $|z| \leq 2$, pois, em geral, há poucas ocorrências de casos em que $|z| > 2$, a menos que sejam específicos.

4.4 Entrada e saída

Por padrão, o programa recebe os comprimentos das duas seqüências s e t, gera uniformemente as seqüências usando um alfabeto de tamanho padrão, e aplica o algoritmo descrito acima. Através de parâmetros de linha de comando, as opções de entrada podem ser alteradas. Elas são as seguintes:

- O tamanho do alfabeto pode ser especificado.
- O método de geração de seqüências pode ser alterado. Os métodos que foram implementados são os seguintes:
  - os símbolos do alfabeto são distribuídos uniformemente, ou
  - metade do alfabeto tem uma probabilidade maior de ocorrer nas seqüências, ou
  - ao invés dos comprimentos das seqüências, o programa recebe dois valores x e y e gera as seqüências tal que hajam no máximo x e y ocorrências de cada símbolo para s e t respectivamente.

  Para a implementação desse segundo método, foi implementada uma função que recebe uma distribuição de probabilidades dos símbolos do alfabeto, e ela pode ser usada para outras distribuições de probabilidade.
- A semente (*seed*) para a geração das seqüências pode ser especificada, caso queiramos gerar as mesmas seqüências que em uma outra execução. Essa semente também é impressa na saída.
- Um arquivo com os dados de entrada (tamanho do alfabeto, comprimentos das seqüências, e as seqüências em si) pode ser indicado. Nesse caso, o programa ignora os dois parâmetros obrigatórios de linha de comando (os comprimentos das seqüências).

A saída consiste nos dados iniciais, nas informações de execução do algoritmo e na solução final. A informações de execução incluem a impressão do peso da estrela máxima, um limitante inferior (o algoritmo de aproximação que se baseia na solução viável) e um limitante superior (dado pela otimização após a inclusão de novas restrições de estrela estendida).

O programa também pode gerar um arquivo MetaPost contendo o desenho dos casamentos das seqüências e os grafos associados G_C e G_{NC}, permitindo também eliminar as arestas transitivas desses grafos. MetaPost é uma linguagem similar a LaTeX e um arquivo MetaPost contém informações para o desenho de uma imagem, que pode ser gerada através do interpretador de mesmo nome (via o comando `mpost`). Essas imagens podem ser coloridas de acordo com o símbolo. As Figuras 4.8, 4.9, 4.10 mostram exemplos dessas imagens.

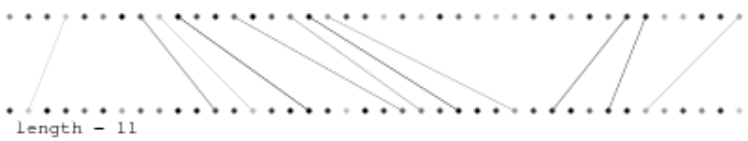

Figura 4.8: Exemplo de uma saída do programa: representação de um RFLCS.

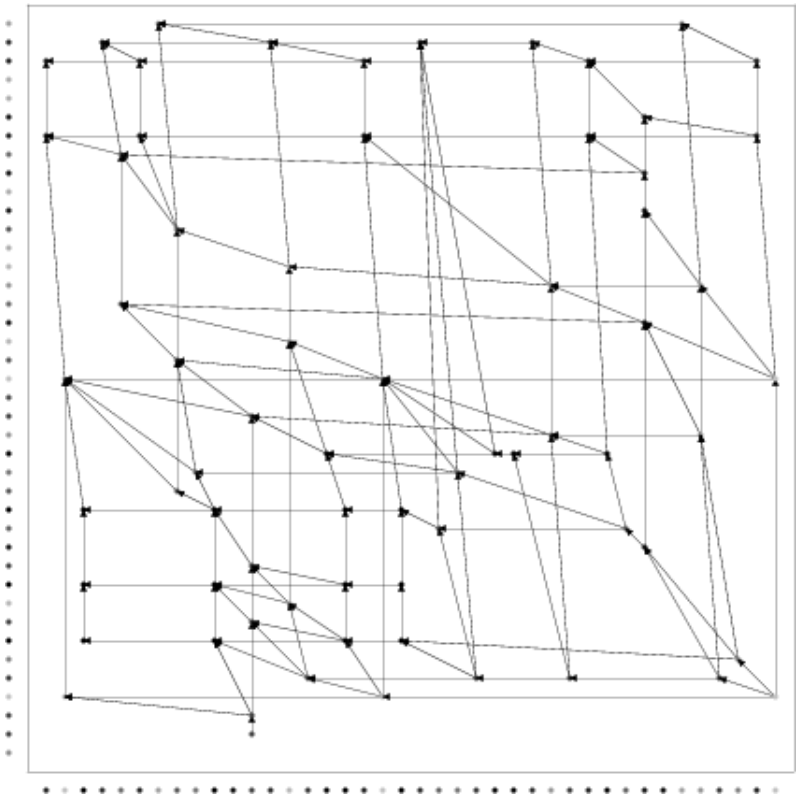

Figura 4.9: Exemplo de uma saída do programa: grafo de cruzamento sem arestas transitivas.

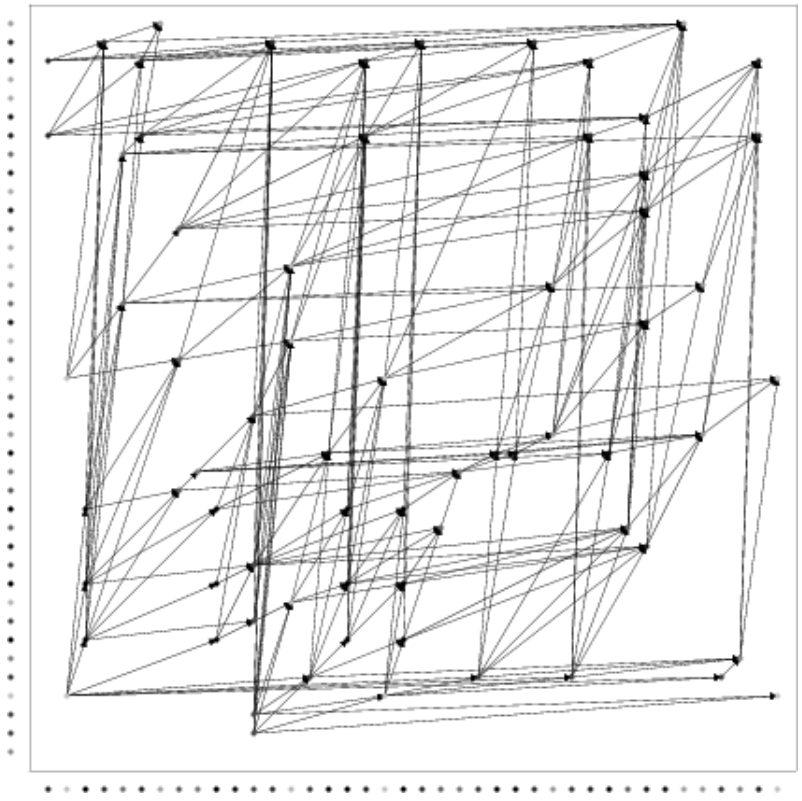

Figura 4.10: Exemplo de uma saída do programa: grafo de não-cruzamento sem arestas transitivas.

4.5 Resultados experimentais

Os dados experimentais mostrados nesta seção são os mesmos disponíveis nas referências [1] e [10].

t. alf.	tempo	(mín/máx)	cortes	resol.	l. inf.	l. sup.	ativos	visit.
64	3604.5	(3600/3609)	1162.5	0	44.4	63.9	1.0	0.0
128	3601.6	(3600/3604)	1927.7	0	56.8	75.2	1.0	0.0
192	2761.8	(762/3600)	2532.1	6	58.8	59.7	1.7	10.7
256	604.4	(224/1582)	1474.0	10	54.2	54.2	0.0	5.6
320	245.9	(93/498)	1077.1	10	46.5	46.5	0.0	1.8
384	113.0	(72/200)	797.0	10	43.0	43.0	0.0	1.0
448	76.0	(52/108)	660.6	10	40.7	40.7	0.0	1.4

Tabela 4.1: Resultados experimentais usando restrições de estrelas estendidas.

A Tabela 4.1 mostra os resultados da execução do programa para $|s| = |t| = 512$ usando a implementação descrita neste capítulo. As seqüências foram geradas de forma uniformemente aleatória, com alfabetos de tamanho $\frac{1}{8}n, \frac{2}{8}n, \ldots, \frac{7}{8}n$. Para cada tamanho, rodamos 10 instâncias, com um limite de tempo de uma hora.

Na primeira coluna, temos o tamanho do alfabeto. Na segunda, o tempo médio das 10 instâncias em segundos, considerando o limite de tempo de uma hora (3600 segundos), e, em parênteses, o tempo mínimo e máximo em segundos. Na terceira, o número médio de cortes de estrela estendida gerados. Na quarta, o número de vezes que o programa chegou em uma solução exata ótima dentro do limite de tempo. Na quinta e na sexta, os limitantes inferiores e superiores médios respectivamente. Nas duas últimas, os números médios de nós ativos da árvore *branch and bound* no momento que o programa terminou e nós visitados ao decorrer da execução.

Nessa tabela, podemos notar que é mais difícil resolver instâncias cujo alfabeto é pequeno em relação ao tamanho das seqüências. Até $\frac{2}{8}n$, nenhuma das instâncias foi resolvida até o final. Em $\frac{3}{8}n$, 6 das instâncias foram resolvidas até o final, e para os alfabetos de tamanho pelo menos $\frac{1}{2}n$, todas as instâncias foram resolvidas dentro do tempo limite de uma hora. É possível que isso aconteça devido ao número grande de repetições e, portanto, número grande de soluções viáveis.

Uma outra hipótese é que, com menos símbolos, existem mais arestas "novas" (comparando com e sem repetições) que têm o potencial de formar buracos ou anti-buracos ímpares no grafo de não-cruzamento, o que torna o problema difícil (como visto na Seção 3.3). Para ser um pouco mais preciso, para cada símbolo a, seja $|s(a)|$ o número de ocorrências de a em s, e $|t(a)|$ o número de ocorrências de a em t. O número de arestas "novas" para cada símbolo a é $O(|s(a)|^2|t(a)|^2)$, isto é, esse número cresce bastante se temos símbolos com ambos $|s(a)|$ e $|t(a)|$ grandes. Assim, quanto

mais repetições de um mesmo símbolo nas duas seqüências, mais arestas "novas" temos e, portanto, mais chance de se formar buracos ou anti-buracos ímpares.

Em geral, para os problemas de alfabeto maior, o número de nós visitados é perto de 1, o que significa que quase não ocorreram ramificações, isto é, as primeiras soluções já foram inteiras. É possível que, em alguns desses casos, o grafo de não-cruzamento associado ao problema seja perfeito e, portanto, o poliedro do RFLCS tem vértices inteiros.

Para verificar experimentalmente o quanto a formulação com restrições de estrela estendida é mais forte que a formulação mais simples (com restrições de cruzamentos), rodamos testes com um algoritmo *branch and bound* usando esta formulação simples. Configuramos o GLPK para gerar cortes de Gomory e por clique, e aplicamos pré-processamento assim como na formulação anterior. A Tabela 4.2 mostra esses resultados.

t. alf.	tempo	(mín/máx)	cortes	resol.	l. inf.	l. sup.	ativos	visit.
256	3669.3	(3642/3724)	300.0	0	13.3	56.0	29.0	3.0
320	2739.3	(771/3631)	282.6	4	45.7	47.3	45.6	91.7
384	1159.9	(534/2886)	271.9	10	43.0	43.0	0.0	87.2
448	746.1	(254/2537)	236.5	10	40.7	40.7	0.0	117.0

Tabela 4.2: Resultados experimentais usando *branch and bound* com a formulação simples (restrições de cruzamentos) e cortes gerados pelo GLPK.

Podemos ver que apenas nos casos $\frac{6}{8}n$ e $\frac{7}{8}n$ o algoritmo resolveu todos os 10 casos dentro do tempo limite de uma hora e demorou aproximadamente 10 vezes mais que na outra formulação.

A Tabela 4.3 mostra a qualidade, em média, dos algoritmos de aproximação.

Para essa tabela, geramos seqüências de comprimento n de forma uniformemente aleatória. Assim como nas tabelas anteriores, foram rodadas 10 instâncias por linha. A primeira coluna é o tamanho do alfabeto, a segunda é o comprimento da seqüência, as três próximas são os resultados médios dos três primeiros algoritmos de aproximação explicados na Seção 3.4 (LCS removendo repetições e dois que pré-processam as seqüências probabilisticamente), a próxima é o máximo entre esses três resultados, e a última é o valor ótimo médio das instâncias (apenas para as instâncias que não demoram muito). Entre parênteses, para cada algoritmo de aproximação e para o máximo, temos, respectivamente, o número de vezes em que o resultado foi igual ao máximo (nas colunas que não é o máximo), e o número de vezes em que o resultado foi igual ao ótimo.

Os dados experimentais indicam que o algoritmo 3 devolve os resultados piores, e o algoritmo 1 é o melhor dos três para instâncias de tamanho grande. Notamos também que, para os dados que temos, a razão entre o resultado ótimo e o máximo

dos resultados dos algoritmos de aproximação é sempre menor que $\frac{5}{4}$. Assim como nas tabelas anteriores, observamos que as instâncias com alfabeto pequeno (em relação ao comprimento das seqüências) são mais difíceis, inclusive em termos de algoritmos de aproximação.

Uma outra tabela similar à Tabela 4.3 que usa um método de geração de seqüências diferente pode ser encontrada em [1].

t. alf.	n	alg. 1		alg. 2		alg. 3		máx.		ót.
	32	4.0	(10/10)	4.0	(10/10)	4.0	(10/10)	4.0	(10)	4.0
	64	7.8	(8/8)	8.0	(10/10)	7.9	(9/9)	8.0	(10)	8.0
$n/8$	128	15.3	(7/6)	15.7	(9/7)	14.2	(1/0)	15.8	(8)	16.0
	256	25.8	(9/-)	23.1	(1/-)	21.3	(0/-)	25.9	(-)	—
	512	52.1	(10/-)	40.5	(0/-)	36.5	(0/-)	52.1	(-)	—
	32	6.5	(4/4)	7.2	(10/10)	6.9	(7/7)	7.2	(10)	7.2
	64	12.7	(3/0)	13.9	(10/1)	12.9	(5/0)	13.9	(1)	15.3
$n/4$	128	21.7	(8/0)	20.5	(3/0)	19.2	(0/0)	22.0	(0)	26.2
	256	36.2	(10/0)	31.0	(0/0)	28.9	(0/0)	36.2	(0)	43.7
	512	58.2	(10/-)	46.2	(0/-)	43.2	(0/-)	58.2	(-)	—
	32	7.8	(3/3)	8.7	(9/7)	7.8	(2/2)	8.8	(8)	9.0
	64	13.9	(4/0)	14.7	(7/3)	13.3	(1/0)	15.0	(3)	16.1
$3n/8$	128	22.5	(8/0)	21.9	(5/0)	20.6	(1/0)	22.8	(0)	25.1
	256	35.7	(10/0)	31.6	(1/0)	30.3	(0/0)	35.7	(0)	39.6
	512	53.7	(10/0)	44.9	(0/0)	43.3	(0/0)	53.7	(0)	59.0
	32	8.2	(6/4)	8.6	(10/8)	7.9	(3/1)	8.6	(8)	8.8
	64	13.0	(2/1)	13.9	(9/3)	12.7	(1/0)	14.0	(3)	14.7
$n/2$	128	21.3	(7/0)	21.0	(5/1)	19.6	(1/0)	21.8	(1)	23.2
	256	33.5	(10/0)	30.7	(1/0)	29.3	(0/0)	33.5	(0)	35.8
	512	50.3	(10/0)	44.7	(0/0)	42.3	(0/0)	50.3	(0)	54.2
	32	7.6	(6/4)	7.8	(8/6)	7.5	(5/4)	8.1	(8)	8.3
	64	12.8	(6/4)	12.9	(7/3)	12.5	(4/4)	13.2	(6)	13.7
$5n/8$	128	20.4	(8/1)	19.8	(5/1)	19.3	(1/0)	20.6	(2)	21.6
	256	31.5	(9/2)	29.6	(2/0)	27.9	(0/0)	31.6	(2)	32.8
	512	46.2	(9/2)	42.4	(1/0)	41.3	(0/0)	46.4	(2)	48.3
	32	6.7	(1/1)	7.6	(10/9)	7.1	(5/4)	7.6	(9)	7.7
	64	11.9	(4/3)	12.5	(9/7)	11.9	(3/3)	12.6	(8)	12.8
$3n/4$	128	19.4	(8/6)	19.2	(6/3)	18.1	(2/1)	19.7	(7)	20.0
	256	28.4	(9/2)	28.0	(5/2)	26.9	(3/1)	28.7	(3)	29.9
	512	42.5	(10/2)	39.8	(0/0)	39.4	(1/0)	42.5	(2)	43.8
	32	7.2	(8/8)	7.4	(9/9)	7.1	(6/6)	7.5	(10)	7.5
	64	11.6	(6/5)	11.8	(8/7)	11.3	(4/4)	12.0	(9)	12.1
$7n/8$	128	18.4	(8/7)	18.4	(8/7)	17.9	(3/3)	18.6	(9)	18.8
	256	26.8	(9/6)	26.0	(4/1)	25.2	(1/0)	26.9	(6)	27.4
	512	39.2	(10/0)	37.4	(1/0)	36.6	(0/0)	39.2	(0)	40.7

Tabela 4.3: Valores dos algoritmos de aproximação e comparação com as soluções exatas correspondentes.

Capítulo 5

Conclusão

O problema do LCS é bem conhecido e tem várias aplicações, então uma descrição completa e minimal do poliedro associado a ele é interessante. Também é interessante ver que algumas as características poliédricas do problema do LCS podem ser estendidas para a variante RFLCS e possivelmente para outras variantes.

Uma relação interessante observada no problema do LCS e do RFLCS pode ser resumida pelo diagrama a seguir.

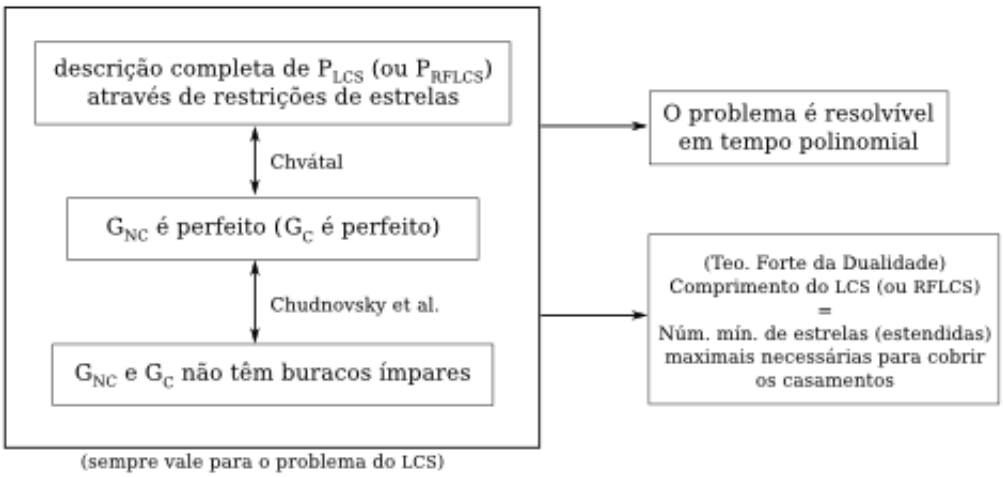

Figura 5.1: Relação entre algumas afirmações.

Como trabalhos futuros, podemos estudar a variante do RFLCS em que são permitidas as reversões, como ocorre nas aplicações em comparação de genoma. Permitir reversões significa permitir que faixas das seqüências sejam trocadas pelo reverso delas. Por exemplo, se $s = abcdefghij$, uma reversão em s pode ser $s = abcihgfej$

(invertemos o trecho $efghi$).

Também pode valer a pena procurar conseqüências interessantes do fato de que o poliedro do LCS equivale ao poliedro de clique de G_{NC} ou estudar métodos que funcionem melhor para alfabetos pequenos no problema do RFLCS. Seria interessante também tentar procurar casos fáceis estudando grafos perfeitos no contexto do problema do RFLCS.

Os algoritmos de aproximação para o problema do RFLCS que mostramos são bem simples, mas, experimentalmente, eles se mostram razoavelmente bons. Seria legal conseguir alguns algoritmos de aproximação melhores do que os que apresentamos, apesar do problema do RFLCS ser APX-difícil.

A implementação do algoritmo descrito no capítulo anterior pode ser encontrada em `http://www.ime.usp.br/~christj/rflcs`.

Essa implementação foi usada neste texto e nas referências [1] e [10] para obter resultados experimentais. Ela requer o pacote GLPK de versão pelo menos 4.20, que pode ser encontrado em `http://www.gnu.org/software/glpk`.

Referências Bibliográficas

[1] S.S. Adi, M. Braga, C.G. Fernandes, C.E. Ferreira, F.H.V. Martinez, M.-F. Sagot, M.A. Stefanes, C. Tjandraatmadja, and Y. Wakabayashi. Repetition-free longest common subsequence. In *Proc. IV Latin-American Algorithms, Graphs and Optimization Symposium*, 2007. to appear.

[2] A. Atamturk, G.L. Nemhauser, and M.W.P Savelsbergh. Conflict graphs in solving integer programming problems. *European Journal of Operational Research*, 121:40–55(16), 2000.

[3] L. Bergroth, H. Hakonen, and T. Raita. A survey of longest common subsequence algorithms. In *SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, page 39, Washington, DC, USA, 2000. IEEE Computer Society.

[4] P. Bonizzoni, G. Della Vedova, R. Dondi, G. Fertin, and S. Vialette. Exemplar longest common subsequence. In *Proceedings of ICCS*, volume 3992 of *Lecture Notes in Computer Science*, pages 622–629. Springer, 2006.

[5] P. Bonizzoni, G. Della Vedova, and G. Mauri. Experimenting an approximation algorithm for the LCS. *Discrete Applied Mathematics*, 110(1):13–24, 2001.

[6] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. Progress on perfect graphs, 2003.

[7] V. Chvátal. On certain polytopes associated with graphs. *J. Combinatorial Theory Ser. B*, 18:138–154, 1975.

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

[9] R.P̃. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[10] C.G. Fernandes, C.E. Ferreira, C. Tjandraatmadja, and Y. Wakabayashi. A polyhedral investigation of the lcs problem and a repetition-free variant. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, 2008. accepted.

[11] H. Goeman and M. Clausen. A new practical linear space algorithm for the longest common subsequence problem. In J. Holub and M. Šimánek, editors, *Proceedings of thePrague Stringology Club Workshop '99*, pages 40–60, Czech Technical University, Prague, Czech Republic, 1999.

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

[13] B.T. Lahn and D.C. Page. Four evolutionary strata on the human X chromosome. *Science*, 286:964–967, 1999.

[14] W.-H. Li, Z. Gu, H. Wang, and A. Nekrutenko. Evolutionary analyses of the human genome. *Nature*, 409:847–849, Feb 2001.

[15] D. Maier. The complexity of some problems on subsequences and supersequences. *J. ACM*, 25(2):322–336, 1978.

[16] W.J. Masek and M. Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.

[17] L. Mirsky. A dual of dilworth's decomposition theorem. *The American Mathematical Monthly*, 78(8):876–877, 1971.

[18] M. Paterson and V. Dancik. Longest common subsequences. In *Mathematical Foundations of Computer Science*, pages 127–142, 1994.

[19] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

[20] D. Sankoff. Gene and genome duplication. *Current Opinion in Genetics and Development*, 11(6):681–684, 2001.

[21] D. Sankoff and N. El-Mabrouk. Genome rearrangement. In T. Jiang, T. Smith, Y. Xu, and M. Zhang, editors, *Current Topics in Computational Biology*, pages 135–155. MIT Press, 2002.

[22] D. Sankoff and J.B. Kruskal. *Time warps, string edits, and macromolecules: The theory and practice of sequence comparison.* Addison-Wesley, 1983.

[23] A. Schrijver. Min-max results in combinatorial optimization. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming - The state of the Art*, pages 439–500. Springer-Verlag, 1983.

[24] H. Skaletsky et al. The male-specific region of the human Y chomosome is a mosaic of discrete sequence classes. *Nature*, 423:825–837, 2003.

[25] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[26] L.A. Wolsey. *Integer Programming.* Wiley-Interscience, 1998.

Parte subjetiva

Como requisitado pela disciplina MAC0499 - Trabalho de Formatura Supervisionado, esta parte da monografia relata algumas de minhas experiências pessoais durante o desenvolvimento do projeto de iniciação científica.

A iniciação científica e seus desafios

No início do 3º ano, decidi procurar uma iniciação científica. Tendo acabado de fazer apenas 2 anos de BCC, e nenhuma disciplina optativa eletiva, não conhecia nem as áreas em que procurar uma iniciação científica. Aconselhado pelo o Prof. Carlos Eduardo Ferreira (Carlinhos) e lendo um pouco de alguns livros, reduzi a minha escolha a duas áreas: Processamento de Imagens/Visão Computacional e Otimização Combinatória/Grafos. O primeiro, eu gostava principalmente das aplicações; o segundo, eu gostava do pouco que eu conhecia.

Na época, decidi pela área de Processamento de Imagens/Visão Computacional e procurei o Prof. Roberto Hirata Jr., que também me aconselhou com atenção e me introduziu aos professores da área. Acabei me interessando pelos tópicos em reconhecimento de gestos que o Prof. Carlos Hitoshi Morimoto havia sugerido e fiz uma iniciação científica orientada por ele durante o 3º ano. No entanto, no meio do ano, o Prof. Hitoshi viajou por um ano para a universidade de Maryland, o que dificultou a comunicação. Ainda mais, tive dificuldades em entender alguns artigos que o Prof. Hitoshi havia me indicado. Apesar dessas dificuldades, a experiência adquirida na iniciação científica com o Prof. Hitoshi certamente valeu a pena. Mesmo que não continuei nessa área, o melhor jeito de saber em que área estudar é tentando.

Influenciado ainda no 3º ano pelas matérias MAC0328 - Algoritmos em Grafos e MAC0452 - Tópicos de Otimização Combinatória (a qual fiz antes de MAC0325 - Otimização Combinatória pois não havia sido oferecido na época), me interessei cada vez mais em Otimização Combinatória e Grafos. Com a decisão de escolher um tópico para o TCC (Trabalho de Conclusão de Curso, ou MAC0499 - Trabalho de Formatura Supervisionado) no início do 4º ano, resolvi procurar o Prof. Carlinhos novamente para uma iniciação científica na área de Otimização Combinatória. Ele me sugeriu um tópico que ele, a Prof.ª Cristina Gomes Fernandes

e a Prof.ª Yoshiko Wakabayashi estavam estudando, que é o problema do RFLCS. Como ele é o responsável pela disciplina do TCC, a Prof.ª Cristina ficou como minha supervisora do TCC e tive a sorte de poder aproveitar a experiência de dois orientadores. Também tive a sorte de fazer parte do estudo de um problema bem interessante e pouco estudado na literatura, e, como resultado, o Prof. Carlinhos, a Prof.ª Cristina e a Prof.ª Yoshiko desenvolveram dois artigos sobre o problema, um para o LAGOS'07 (*IV Latin-American Algorithms, Graphs and Optimization Symposium*) e outro para o LATIN'08 (*8th Latin American Theoritical Informatics Symposium*), ambos aceitos.

Não tendo experiência em escrever textos científicos grandes, um desafio foi escrever esta monografia. No entanto, foi uma experiência interessante e com certeza valeu a pena organizar as idéias da iniciação científica em papel. Outro desafio um tanto recorrente foi a falta de tempo para lidar com as matérias mais a iniciação científica. Porém, ao fim, acabei conseguindo lidar com isso de uma forma ou outra.

Devido ao excelente apoio de meus orientadores, superamos uma boa parte dos desafios inerentes ao estudo dos problemas do LCS e RFLCS. Talvez o maior desafio tenha sido a implementação. De vez em quando, optei por resolver problemas no código do jeito mais fácil ao invés do jeito mais flexível. Outros desafios interessantes foram provar que a formulação do LCS que obtivemos tem vértices inteiros (a prova direta foi dada pelo Prof. Carlinhos) e resolver os problemas da separação para o LCS e o RFLCS.

Interação com os orientadores

Como eu havia mencionado na seção anterior, tive a sorte de ser orientado por dois professores excelentes. Ambos foram bastante dispostos a me ajudar e bem atenciosos, e me ajudaram não só na iniciação científica como também em outros assuntos acadêmicos, como, por exemplo, na obtenção de bolsa na FAPESP, na sugestão de disciplinas optativas e na recomendação de fazer matérias como aluno de pós. Reuni-me semanalmente com eles e isso manteve o andamento do projeto bem fluido.

Disciplinas relevantes ao projeto

Além de citar as disciplinas relevantes à iniciação científica, também vale a pena mencionar os ótimos professores que as ministraram, pois eles são mais importantes que a disciplina em si.

- MAC0122 - Princípios de Desenvolvimento de Algoritmos (Prof.ª Dr.ª Nami Kobayashi). Enquanto que na matéria MAC0110 - Introdução à Computação aprendi a sintaxe e alguns conceitos básicos de programação,

foi apenas na disciplina MAC0122 que esses conceitos se solidificaram e aprendi a construir algoritmos não triviais.

- MAC0323 - Estruturas de Dados (Prof. Dr. Carlos Eduardo Ferreira). Essa é outra matéria básica que me permitiu construir programas mais eficientes e flexíveis. Uma boa manipulação de estruturas de dados é importante para qualquer programa.

- MAC0211 - Laboratório de Programação I (Prof. Dr. Roberto Hirata Jr.). A experiência mais importante nessa disciplina foi o desenvolvimento de um projeto de grande porte pela primeira vez no curso de BCC. Além disso, aprendi algumas ferramentas que foram bastante úteis mais adiante, como LaTeX e Makefiles.

- MAC0328 - Algoritmos em Grafos (Prof. Dr. Paulo Feofiloff). Como já dito na introdução desta monografia, grafos são bastante interessantes para se representar problemas. Uma lida na monografia mostra como o entendimento de grafos foi importante para o estudo dos problemas abordados.

- MAC0338 - Análise de Algoritmos (Prof. Dr. Yoshiharu Kohayakawa). Aprendi nessa disciplina como reconhecer algoritmos teoricamente eficientes ou não eficientes. Também aprendi algumas técnicas úteis em computação, como programação dinâmica, além de ser a matéria em que vi o problema do lcs pela primeira vez (apesar de, naquela época, eu ainda não saber que eu o estudaria).

- MAC0325 - Otimização Combinatória (Prof.ª Dr.ª Yoshiko Wakabayashi). Os problemas do lcs e do rflcs são problemas de otimização combinatória e é claro por que essa matéria é importante para a iniciação científica. Um exemplo em particular é que vi nessa matéria o teorema de Dilworth, que pode ser aplicado ao problema do lcs.

- MAC0452 - Tópicos de Otimização Combinatória (Prof. Dr. Carlos Eduardo Ferreira). Nessa disciplina, o tópico abordado foi programação inteira, e é óbvio por que ela foi importante para o desenvolvimento da iniciação científica. Não só aprendi os conceitos de programação inteira, como também aprendi a modelar na prática um problema no GLPK, o que foi bastante útil para este projeto.

Com um pouco menos de ênfase, outras matérias também são relevantes para esse projeto. Elas são MAC0110 - Introdução a Computação, que me deu as primeiras noções de programação; MAT0138 - Álgebra I para Computação, que me ensinou conceitos fundamentais de matemática, além de eu aprender a provar teoremas; MAT0139 - Álgebra Linear para Computação, que me ensinou

outros conceitos matemáticos importantes, que inclusive foram usados no projeto; MAC0315 - Programação Linear, que me introduziu a conceitos importantes para problemas de programação linear; e MAC0327 - Desafios de Programação, que me ofereceu bastante experiência prática em resolver probleminhas de programação, além de ser divertido.

Muitas outras disciplinas do curso de BCC foram bastante interessantes e bem dadas, mas não as citarei aqui por não serem relacionadas ao projeto.

Passos futuros

A iniciação científica foi uma experiência muito boa e pretendo prosseguir como orientado do Prof. Carlinhos em um mestrado cujo tópico é o estudo da variante do rflcs que permite reversões de trechos das seqüências. Isso vem da idéia de que houveram reversões na evolução de genomas mais complexos e, portanto, uma medida que considera isso pode ser mais precisa para essa aplicação. Além disso, na conclusão da parte técnica desta monografia, também menciono alguns outros aspectos do problema do rflcs que podem ser interessantes estudar.

As matérias que cursarei durante a pós-graduação também devem contribuir para o meu aprofundamento na área.

Agradecimentos

Em nenhuma ordem em particular, agradeço aos meus orientadores, por terem sido bastante atenciosos e dedicados comigo; aos meus professores, por tudo que aprendi durante o curso; aos meus amigos (especialmente os do IME), pela valiosa amizade e pelo apoio durante todo o curso, além de torná-lo mais divertido; e à minha família, pelo suporte durante todos os anos de minha vida.

Enfim, obrigado por ler minha monografia.