Correspondência inexata entre grafos

Alexandre da Silva Freire

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO TÍTULO
DE
MESTRE EM CIÊNCIAS

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES.

São Paulo, julho de 2008.

Correspondência inexata entre grafos

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Alexandre da Silva Freire e aprovada pela Comissão Julgadora.

Banca Examinadora:

- Prof. Dr. Carlos Eduardo Ferreira - IME/USP (Presidente).
- Profa. Dra. Cristina Gomes Ferandes - IME/USP.
- Profa. Dra. Maria Claudia Silva Boeres - UFES.
- Prof. Dr. Roberto Marcondes César Júnior - IME/USP (Suplente).
- Prof. Dr. Flávio Keidi Miyazawa - UNICAMP (Suplente).

Agradecimentos

Para que a arte da computação continue se expandindo, é fundamental a existência de pessoas dedicadas a aprender e, principalmente, a ensinar. Por isso, meus agradecimentos vão para todos os profissionais da educação. A seguir, mencionarei alguns destes profissionais, com os quais eu tive o prazer e a honra de conviver por algum tempo: José Coelho de Pina, possuidor de didática e humor refinadíssimos. Suas aulas são divertidas e prazerosas. Uma pessoa extremamente solícita e simples; Yoshiko Wakabayashi, dedicadíssima e muito carinhosa com seu trabalho e também com seus alunos; Paulo Feofiloff, um grande elaborador de materiais didáticos. É praticamente impossível pesquisar sobre computação na internet sem se deparar com suas obras de grande qualidade na forma e no conteúdo; Cristina Gomes Fernandes, uma professora que conduz suas aulas de maneira agradável e desafiadora. Sempre simpática e alegre. Colaborou bastante conosco no início deste trabalho. Suas correções e sugestões após a defesa elevaram muito a qualidade desta dissertação; Mário Leston Rey e Aritanan Borges Garcia Gruber, foram responsáveis por meus primeiros contatos com a computação, e fizeram isso de forma brilhante. Nunca os esquecerei; Waldomiro Pereira, o professor que deu o pontapé final para que eu me encorajasse a fazer o mestrado. Segundo ele, “não se deve ser piloto de avião se não for para pilotar caças”. Foi através desta metáfora que entendi que para pilotar no imenso céu da ciência é preciso ter capacidade técnica sim, mas acima de tudo, fé e coragem!

Gostaria de agradecer ao pessoal da área de visão computacional (Roberto Marcondes Cesar Jr., Ana Beatriz Vicentim Graciano e Alexandre Noma), que foram fundamentais para que pudéssemos desenvolver este trabalho. Além disso, são pessoas muito agradáveis!

Tenho que reservar um parágrafo para agradecer ao Carlos Eduardo Ferreira (Carlinhos). Afinal de contas, ele, em seu papel de orientador, tentou exaustivamente me ensinar a escrever bem (ainda não consegui aprender, mas estou gostando bastante de tentar). O conheci através da Maratona de Programação, competição esta que me levou a estudar por diversos e divertidos finais de semana. O Carlinhos é um dos grandes incentivadores e colaboradores desta “brincadeira”. O tempo em que

trabalhamos juntos foi muito produtivo para mim. Espero continuar trabalhando com o Carlinhos por mais algum tempo, pelo menos até que ele consiga me vencer no jogo de pebolim!

Tenho muito a agradecer também aos amigos que fiz durante esta jornada rumo ao ∞ universo da computação. Durante o mestrado no IME-USP conheci pessoas fantásticas. Não listarei nomes para que minha memória falha não seja injusta com ninguém. Citarei apenas dois grandes amigos que foram meus companheiros de Maratona de Programação e que hoje são meus colegas de mestrado: Wanderley Guimarães da Silva, um guerreiro que se supera a cada dia. Ele já exerce a função de professor mesmo antes de ser nomeado para o cargo. Eu mesmo já aprendi diversos algoritmos com ele, em longas caminhadas, nas quais eu praticamente o forçava a me acompanhar; Thadeu Russo e Carmo, um jovem bastante talentoso, que estuda assuntos variados. Por uma imposição da vida teve que começar a trabalhar ainda muito jovem. Mas, isso fez com que ele se tornasse mais responsável e esforçado. Sempre está em busca de novos desafios.

Termino agradecendo à minha mãe (Maria Aparecida Cardoso da Silva), que tem tido uma vida muito dura, mas mesmo assim não deixou de lado o humor e a alegria de viver, e à minha namorada, Juliana Novochadlo De Oliveira (Djudja), que me faz queimar muito mais neurônios do que uma prova de análise de algoritmos, mas que me devolve em dobro todo o amor que lhe dou.

Resumo

Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ dois grafos simples. Um mapeamento de G_I para G_M é um conjunto de associações, tal que cada vértice de V_I está associado a um vértice de V_M , e cada aresta de A_I está associada a um par de vértices de V_M . A cada possível associação é atribuído um custo. O problema de correspondência inexata entre grafos (PCIG) consiste em encontrar um mapeamento de G_I para G_M , tal que a soma dos custos de suas associações seja mínima.

Nesta dissertação, resumimos os resultados encontrados na literatura sobre o PCIG e algumas de suas variações. Os resultados que incluímos aqui tratam sobre a questão de como formular o PCIG e algumas de suas variações, através de programação linear inteira. Provamos alguns resultados de complexidade computacional que relacionam variações do PCIG a problemas clássicos, como isomorfismo e partição de grafos. Fornecemos uma formulação através de programação linear inteira para o PCCA (uma variante do PCIG com conexidade e cobertura de arestas). Mostramos que o PCCA é $\mathcal{NP}$ -difícil quando os grafos de entrada são completos ou árvores (chamamos o segundo caso de PCCA *para árvores*). Apresentamos uma formulação linear inteira e um algoritmo – que é polinomial se o grau máximo dos vértices de V_M for limitado por uma constante – para o PCCA *para árvores*. Mostramos um caso especial em que o PCCA *para árvores* pode ser resolvido em tempo polinomial. Por último, exibimos alguns resultados experimentais, inclusive com instâncias reais de uma aplicação do problema.

Palavras-chave: teoria dos grafos, correspondência inexata entre grafos, partição de grafos, programação linear, programação inteira, otimização combinatória.

Abstract

Let $G_I = (V_I, A_I)$ and $G_M = (V_M, A_M)$ be two simple graphs. A mapping from G_I to G_M is an association set, such that each vertex in V_I is associated to a vertex in V_M , and each edge in A_I is associated to a pair of vertices of V_M . A cost is defined to each possible association. The inexact graph correspondence problem (IGCP) consists in finding a mapping from G_I to G_M , such that the sum of its associations costs is minimized.

In this dissertation, we summarize the results found in the literature about the IGCP and some variations. The results included here address the question of how to formulate the IGCP and some variations, using integer linear programming. We prove some computational complexity results which relate IGCP variations with classical problems, like graph isomorphism and partitioning. We give an integer linear programming formulation to the ICEC (IGCP with connectivity and edges cover). We show that the ICEC is $\mathcal{NP}$ -hard when the input graphs are complete or trees (we call the second case ICEC *for trees*). We introduce an integer linear formulation and an algorithm – which has polynomial running time if the vertices of V_M have maximum degree bounded by a constant – to the ICEC *for trees*. We show a especial case in which the ICEC *for trees* can be solved in polynomial time. Finally, we present some experimental results, also with instances of a real application of the problem.

Keywords: graph theory, inexact correspondence between graphs, graph partitioning, linear programming, integer programming, combinatorial optimization.

Sumário

1	Introdução e Definição do Problema	1
1.1	Definição do Problema	1
1.2	Variações do PCIG	2
1.3	Aplicação em Visão Computacional	3
1.4	Escolha do Problema de Interesse	7
1.5	Estrutura da Dissertação	7
2	Análise dos Resultados Encontrados na Literatura	9
2.1	O Problema de Isomorfismo de Grafos	9
2.1.1	Relação entre Isomorfismo e o PCCA	12
2.2	Problemas de Partição de Grafos	15
2.3	Formulação Não Linear do PCCC	20
2.4	Formulação Linear do PCCC <i>sem pesos nas arestas</i>	22
2.4.1	Corte-de-conexidade	23
2.4.2	Utilização de Fluxo em Redes para Tratar a Restrição de Conexidade	28
2.5	Considerações	32
3	PCIG com Cobertura de Arestas (PCCA)	35
3.1	Complexidade	35

3.2	Formulação	37
3.2.1	Redução da Quantidade de Inequações na Formulação	40
4	PCCA para árvores	47
4.1	Complexidade	47
4.2	Formulação	54
5	Algoritmos para o PCCA para árvores	63
5.1	Um Algoritmo de Programação Dinâmica	63
5.2	Um Caso Polinomial	94
6	Experimentos Computacionais	121
6.1	Desempenho do Algoritmo de Programação Dinâmica na Prática	121
6.2	Testes com Instâncias Geradas de Forma Automatizada	123
6.3	Testes com Instâncias Reais	129
7	Conclusão e Trabalhos Futuros	131
7.1	Aspectos Importantes com Relação ao PCIG	131
	Referências Bibliográficas	135

Capítulo 1

Introdução e Definição do Problema

Neste capítulo, definimos o Problema de Correspondência Inexata entre Grafos (PCIG) e algumas de suas variações. Mostramos uma aplicação do PCIG em visão computacional e, por último, descrevemos a estrutura geral dos capítulos seguintes.

Um problema clássico de correspondência **exata** entre grafos é o chamado “problema de isomorfismo entre grafos”. Este é um problema de decisão no qual deseja-se saber se dois grafos são “equivalentes” (ou isomorfos). Mais adiante, definimos formalmente o problema de isomorfismo. Já o PCIG, conforme definiremos a seguir, é um problema de otimização que tem como objetivo encontrar uma melhor forma de “casar” dois grafos, que não necessariamente são equivalentes. Por isso, empregamos o termo “correspondência **inexata**” no nome do problema.

1.1 Definição do Problema

Sejam $G_M = (V_M, A_M)$ e $G_I = (V_I, A_I)$ dois grafos simples não-orientados. O custo da associação de cada vértice $i \in V_I$ para cada vértice $k \in V_M$ é dado por uma função $c : V_I \times V_M \rightarrow \mathbb{R}_+$. O custo da associação de cada aresta $ij \in A_I$ para cada par de vértices (não necessariamente distintos) $k, l \in V_M$ é dado por uma função $d : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$.

Um *mapeamento* de G_I para G_M é um conjunto de associações feitas de cada vértice de V_I para algum vértice de V_M (as associações de cada aresta de A_I para algum par de vértices de V_M são “induzidas” pelas associações dos vértices). Dizemos que uma aresta $ij \in A_I$ está associada a um par de vértices $k, l \in V_M$ se uma das afirmações abaixo for verdadeira:

- (i) o vértice i está associado ao vértice k e o vértice j está associado ao vértice l .

- (ii) o vértice i está associado ao vértice l e o vértice j está associado ao vértice k .

Quando uma aresta $ij \in A_I$ está associada a um par de vértices $k, l \in V_M$, tal que $kl \in A_M$, dizemos que a aresta ij está *coberta* pela aresta kl ou, simplesmente, que ij está associada a kl . Se uma aresta $ij \in A_I$ está associada a um par de vértices $k, l \in V_M$, tal que $k = l$, dizemos que a aresta ij está coberta pelo vértice k ou, simplesmente, que ij está associada a k . Quando uma aresta $ij \in A_I$ está associada a um par de vértices $k, l \in V_M$, tal que $k \neq l$ e $kl \notin A_M$, dizemos que ij está *descoberta*.

Por conveniência, introduzimos o termo *cluster* para designar um vértice de V_M . Dessa forma, podemos dizer que um mapeamento é a associação dos vértices de V_I a clusters. Em um mapeamento, cada vértice de V_I está associado a um cluster e cada aresta de A_I está associada a um par de clusters (não necessariamente distintos). O custo de um mapeamento é a soma dos custos das associações dos vértices com os custos das associações das arestas. Dizemos que um mapeamento é ótimo se não existir nenhum outro mapeamento com custo menor.

Figura 1.1: À esquerda G_M e à direita G_I . As setas indicam o seguinte mapeamento: Os vértices 1, 2 e 3 estão associados ao cluster a , os vértices 4, 6 e 7 estão associados ao cluster b e os vértices 5, 8 e 9 estão associados ao cluster c . As arestas $\{1, 2\}$ e $\{1, 3\}$ estão associadas ao cluster a , as arestas $\{4, 7\}$ e $\{4, 6\}$ estão associadas ao cluster b , as arestas $\{5, 9\}$ e $\{5, 8\}$ estão associadas ao cluster c , a aresta $\{2, 4\}$ está associada à aresta $\{a, b\}$, a aresta $\{3, 5\}$ está associada à aresta $\{a, c\}$ e a aresta $\{7, 8\}$ está associada ao par de clusters b, c . Como $\{b, c\} \notin A_M$ então a aresta $\{7, 8\}$ está descoberta.

O PCIG consiste em encontrar um mapeamento ótimo de G_I para G_M .

1.2 Variações do PCIG

Existem variações mais restritas do PCIG. As que consideramos neste trabalho são as seguintes:

- (i) **PCIG com cobertura de arestas** – Nesta variação do problema exige-se que todo mapeamento preserve a seguinte relação: para toda aresta $ij \in A_I$ e todo par de clusters $k, l \in V_M$,

tal que $k \neq l$, vale que, se a aresta ij está associada ao par de clusters k, l , então existe uma aresta $kl \in A_M$. Ou seja, todas as arestas de A_I devem ser cobertas ou por arestas de A_M ou por clusters de V_M .

- (ii) **PCIG com conexidade** – Seja $\lambda(k)$ o conjunto de vértices de V_I associados a um cluster $k \in V_M$. Nesta variação do problema exige-se que $G_I[\lambda(k)]$ seja conexo, para todo $k \in V_M$, sendo que $G_I[\lambda(k)]$ é o subgrafo de G_I induzido por $\lambda(k)$.
- (iii) **PCIG com cobertura de clusters** – Nesta variação do problema exige-se que, para cada cluster $k \in V_M$, exista pelo menos um vértice de V_I associado a k .

Podemos combinar essas variações dependendo da aplicação de interesse. Alguns resultados encontrados na literatura [6, 16] tratam o caso em que é feita uma junção das restrições (ii) e (iii). Denominamos este problema de PCCC (PCIG com Conexidade e Cobertura de Clusters). Outros resultados encontrados [24, 30] tratam o caso “básico”, ou seja, nenhuma destas três restrições é considerada. Neste trabalho, investigaremos o problema que denominamos PCCA (PCIG com Conexidade e Cobertura de Arestas). Ou seja, tratamos do PCIG acrescido das restrições (i) e (ii). Note que podemos compor vários problemas variando as restrições (i), (ii) e (iii), sendo que cada um deles pode ter propriedades diferentes. Antes de justificar nossa escolha pelo PCCA, falaremos sobre uma aplicação do PCIG.

1.3 Aplicação em Visão Computacional

Uma aplicação do PCIG em visão computacional é o reconhecimento estrutural de objetos baseado em modelo ¹. São dados:

- (i) um modelo estrutural de um objeto, que descreve as partes em que o mesmo está subdividido e as relações entre essas partes;
- (ii) uma imagem digital contendo um objeto, que sabemos ser o mesmo do modelo.

O objetivo é atribuir cada ponto da imagem a uma parte do modelo (ver figura 1.2).

Antes de realizar efetivamente o reconhecimento estrutural do objeto, é necessária a realização de algumas etapas de pré-processamento, a fim de agrupar os pontos da imagem em sub-regiões de

¹Foi através desta aplicação que o problema chegou ao nosso conhecimento. Agradecemos ao professor Roberto Marcondes Cesar Jr. por compartilhar conosco esta aplicação.

Figura 1.2: Um modelo de uma face humana [24,30].

acordo com algum critério (por exemplo, uniformidade de cor ou texturas). Esse processo é chamado de segmentação de imagens (ver figura 1.3). Maiores detalhes sobre a etapa de pré-processamento fogem ao escopo deste trabalho e podem ser encontrados em [24,44].

Figura 1.3: Um exemplo de uma imagem segmentada [24,30].

Após realizada a segmentação, definimos os grafos $G_M = (V_M, A_M)$ e $G_I = (V_I, A_I)$ da seguinte maneira: V_M corresponde ao conjunto de sub-regiões do modelo, A_M corresponde às relações entre

as sub-regiões do modelo (tipicamente existe uma aresta $kl \in A_M$ se e somente se as sub-regiões do modelo correspondentes aos vértices k e l possuem parte de suas bordas em comum), V_I corresponde às sub-regiões obtidas na segmentação da imagem e A_I corresponde às relações entre as sub-regiões da imagem (analogamente a A_M). Portanto, G_M é o grafo que representa o modelo e G_I é o grafo que representa a imagem. Note que, da forma como definimos G_I e G_M , ambos os grafos são planares. Este fato pode ser levado em consideração na construção de métodos de resolução do PCIG que tenham como objetivo atender à aplicação de reconhecimento estrutural de objetos.

O custo da associação entre um vértice $i \in V_I$ e um cluster $k \in V_M$ pode ser definido de diversas maneiras. O mesmo ocorre com o custo das associações entre as arestas de A_I e os pares de clusters de V_M .

A seguir, mostramos um exemplo de como definir custos para as associações: atribuímos um peso $p_i \in \mathbb{R}$ a cada vértice $i \in V_I$ e um peso $P_k \in \mathbb{R}$ a cada cluster $k \in V_M$, bem como um peso w_{ij} a cada aresta $ij \in A_I$ e um peso W_{kl} a cada aresta $kl \in A_M$. Para atribuir os pesos aos vértices de V_I , podemos utilizar o nível médio de cinza dos pixels contidos na sub-região correspondente da imagem. Para atribuir os pesos aos clusters, podemos utilizar o nível de cinza esperado na sub-região correspondente do modelo. Os pesos das arestas de A_I e os pesos das arestas de A_M podem ser dados pela distância entre os pontos médios das sub-regiões adjacentes na imagem e no modelo, respectivamente.

A função de custo das associações dos vértices de V_I para os clusters de V_M é definida da seguinte maneira:

$$c(i, k) = |p_i - P_k|, \text{ para todo } i \in V_I \text{ e } k \in V_M.$$

A função de custo das associações das arestas de A_I para os pares de clusters de V_M é definida da seguinte maneira (para todo $ij \in A_I$ e $k, l \in V_M$):

$$d(ij, k, l) = \begin{cases} |w_{ij} - W_{kl}|, & \text{se } kl \in A_M \\ w_{ij}, & \text{se } kl \notin A_M. \end{cases}$$

Figura 1.4: À esquerda G_M e à direita G_I . Cada vértice possui um rótulo e um peso (rótulo:peso). Cada aresta possui um peso. As setas indicam um mapeamento ótimo de G_I para G_M com custo 68. Os custos das associações contidas em tal mapeamento serão mostrados nas tabelas a seguir.

Os custos das associações dos vértices de V_I para os clusters correspondentes:

Vértice $i \in V_I$	Cluster $k \in V_M$	$c(i, k)$
1	a	1
2	a	5
3	a	1
4	b	5
6	b	5
7	b	0
5	c	5
8	c	10
9	c	5

Os custos das associações das arestas de A_I para os correspondentes pares de clusters:

Aresta $ij \in A_I$	Par de clusters $k, l \in V_M$	$d(ij, k, l)$
$\{1, 2\}$	a, a	5
$\{1, 3\}$	a, a	2
$\{4, 6\}$	b, b	1
$\{4, 7\}$	b, b	2
$\{5, 8\}$	c, c	2
$\{5, 9\}$	c, c	4
$\{2, 4\}$	a, b	5
$\{3, 5\}$	a, c	3
$\{7, 8\}$	b, c	7

1.4 Escolha do Problema de Interesse

Nossa motivação inicial para estudar o PCIG veio da aplicação de reconhecimento estrutural de objetos. Optamos por tratar apenas o PCCA (PCIG com Conexidade e Cobertura de Arestas) por ser a variação que melhor se adequa a esta aplicação.

Exigir cobertura de clusters impossibilita o reconhecimento de objetos “incompletos” (objetos que não possuem algumas das partes definidas no modelo). Por isso, não incluímos a restrição de cobertura de clusters.

Exigir conexidade é razoável, já que as sub-regiões associadas a um mesmo cluster correspondem às sub-regiões internas a uma única parte de um objeto.

Toda informação estrutural do objeto está contida nas arestas, por isso, é razoável que esta estrutura seja mantida do modelo para a imagem. Note que não exigimos cobertura das arestas do modelo, apenas exigimos cobertura das arestas da imagem. Ou seja, as arestas da imagem não podem ser associadas a não arestas do modelo.

1.5 Estrutura da Dissertação

Começamos o segundo capítulo falando sobre dois problemas relacionados ao PCIG: o problema de isomorfismo em grafos e o problema de partição de grafos. Depois, mostramos algumas formulações do PCCC (PCIG com Conexidade e Cobertura de Clusters) encontradas na literatura.

Os capítulos 3 ao 5 são dedicados aos resultados que obtivemos para o PCCA (PCIG com Conexidade e Cobertura de Arestas). No capítulo 3, provamos que o PCCA é $\mathcal{NP}$ -difícil quando os grafos em questão são completos, e apresentamos uma formulação para o caso geral do problema. No capítulo 4, provamos que, mesmo no caso em que os grafos de entrada são árvores (chamamos este caso de PCCA *para árvores*), o problema continua $\mathcal{NP}$ -difícil. Em seguida, apresentamos uma formulação para o PCCA *para árvores*. No capítulo 5, apresentamos dois algoritmos para o PCCA *para árvores*. O primeiro é baseado em programação dinâmica. Caso seja fixada uma constante para limitar o grau máximo dos clusters (vértices de V_M), o algoritmo que apresentamos é polinomial no tamanho dos grafos de entrada. O segundo algoritmo que apresentamos é específico para uma determinada classe de árvores (o grafo G_I deve ser uma *estrela estendida*). Neste caso, mostramos que é possível resolver o PCCA em tempo polinomial.

No capítulo 6, apresentamos os resultados computacionais que obtivemos e, por fim, mostramos no capítulo 7 a conclusão do nosso trabalho e sugerimos possíveis trabalhos futuros.

Como pré-requisitos, é desejável que o leitor tenha noções elementares sobre teoria dos grafos [40], programação linear [11, 46] e complexidade computacional [23].

Capítulo 2

Análise dos Resultados Encontrados na Literatura

Neste capítulo, fazemos uma pequena resenha do material encontrado na literatura sobre o PCIG e dois problemas correlatos. Começamos falando sobre o problema de isomorfismo de grafos. Após definir o problema, dar um exemplo de aplicação e falar de sua complexidade, traçamos um paralelo entre isomorfismo e o PCIG. Depois, falamos um pouco sobre problemas de partição de grafos. Comparamos as características desses problemas com o PCIG. A literatura sobre estes dois problemas (isomorfismo e partição) é muito vasta. Apesar da semelhança entre eles e o PCCA, aparentemente não há nenhum resultado obtido para estes problemas que possa ser estendido para o PCCA. Por isso, optamos por não nos aprofundarmos na resenha sobre estes problemas. Apenas mencionamos alguns poucos resultados.

Por último, apresentamos algumas formulações do PCCC (PCIG com Conexidade e Cobertura de Clusters).

2.1 O Problema de Isomorfismo de Grafos

Sejam $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ dois grafos. O problema de isomorfismo de grafos (ou simplesmente isomorfismo) consiste em determinar se existe um mapeamento um-para-um do conjunto de vértices V_1 para V_2 , tal que as adjacências existentes em G_1 sejam preservadas em G_2 e vice-versa. Ou seja, o problema é determinar se existe uma bijeção $h : V_1 \rightarrow V_2$, tal que $\{u, v\} \in A_1$ se e somente se $\{h(u), h(v)\} \in A_2$. Uma bijeção h com tal propriedade é chamada de *isomorfismo*. Utilizamos a notação $G_1 \cong G_2$ para expressar que G_1 e G_2 são isomorfos.

De certo modo, pode-se dizer que o problema de isomorfismo consiste em testar quando dois grafos são iguais (ou equivalentes). Suponha que, dada uma coleção de grafos, necessitamos executar

Figura 2.1: Os dois grafos acima são isomorfos. O grafo mostrado na figura é conhecido como *grafo de Petersen*. Seu desenho mais comum é o mostrado ao lado esquerdo. Este grafo, apresentado por Julius Petersen em 1898, é muito famoso por servir de contra-exemplo para diversas conjecturas.

operações sobre cada um deles. Se pudermos identificar quais grafos desta coleção estão duplicados, podemos evitar trabalho redundante, executando tais operações uma única vez para cada grafo diferente.

Utilizando teoria dos grupos

Uma forma de abordar o problema de isomorfismo é utilizando teoria de grupos. Um mapeamento de um grafo para ele mesmo é chamado um *automorfismo*. Ou seja, um automorfismo é uma permutação de vértices que preserva as relações de adjacências, bem como as de não adjacências. Compondo automorfismos de um grafo G obtemos novamente um automorfismo de G . Logo, o conjunto de todos os automorfismos de G é um grupo. Grupos são usados na Matemática e nas ciências em geral para capturar a simetria interna de uma estrutura na forma de automorfismos de grupo. Uma simetria interna está normalmente associada com alguma propriedade invariante. Mais adiante, listamos alguns dos invariantes relacionados ao problema de isomorfismo de grafos.

Identificar simetrias é uma aplicação importante de isomorfismo. A coleção de automorfismos de um grafo (seu grupo de automorfismos) fornece bastante informação a respeito do quão simétrico é tal grafo. Por exemplo, um grafo completo K_n tem $n!$ automorfismos (qualquer mapeamento dele para ele mesmo será um automorfismo), enquanto um grafo arbitrário gerado aleatoriamente provavelmente terá poucos automorfismos (no mínimo um, considerando o automorfismo identidade, pois qualquer grafo G é igual a ele mesmo).

Seja $\text{AUT}(G)$ o conjunto de todos os automorfismos de um grafo G . Se pudéssemos determinar

$\text{AUT}(G)$ eficientemente, automaticamente poderíamos resolver o problema de isomorfismo de forma eficiente. Porém, o tamanho de $\text{AUT}(G)$ pode ser muito grande. No exemplo dado anteriormente vimos que $|\text{AUT}(\mathcal{K}_n)| = n!$. Entretanto, é possível escrever $\text{AUT}(G)$ utilizando até $O(n^2)$ geradores [48], sendo que n é a quantidade de vértices de G . Apesar disso, não se conhece algoritmo para determinar eficientemente esses geradores. Existem resultados que provam que determinar os geradores de $\text{AUT}(G)$, para um grafo G qualquer, e determinar se um grafo H é isomorfo a G , são problemas equivalentes [48].

Complexidade

Os primeiros trabalhos sobre isomorfismo surgiram na década de setenta. Um fato bastante intrigante acerca da complexidade do problema de isomorfismo é que ele está em $\mathcal{NP}$ (é possível fornecer um certificado, que pode ser verificado em tempo polinomial, de que dois grafos são isomorfos; basta exibir um isomorfismo entre os grafos em questão e podemos facilmente verificar que os respectivos grafos são isomorfos), mas não se sabe se o problema pode ser resolvido em tempo polinomial nem se sabe se ele é $\mathcal{NP}$ -completo. Este fato torna o problema de isomorfismo muito importante também do ponto de vista teórico. Existem estudos que cogitam a possibilidade do problema de isomorfismo estar contido em uma classe de complexidade intermediária entre $\mathcal{P}$ e $\mathcal{NP}$ -completo [3, 21]. Recentemente, chegou à nossa atenção um artigo depositado em um arquivo [15] em que o autor afirma ter obtido um algoritmo polinomial para verificar se dois grafos quaisquer são isomorfos.

Considere a seguinte generalização do problema de isomorfismo:

Isomorfismo de Subgrafo: Dados dois grafos $G = (V_1, A_1)$ e $H = (V_2, A_2)$, determinar se G contém um subgrafo induzido isomorfo a H .

Note que o problema de isomorfismo é um caso particular deste problema em que $|V_1| = |V_2|$ e $|A_1| = |A_2|$. Apesar da semelhança entre estes problemas, é sabido que o problema de isomorfismo de subgrafo é $\mathcal{NP}$ -completo [23], mesmo quando H é uma árvore (redução a partir de caminho hamiltoniano).

Invariantes

Uma forma de poupar bastante trabalho na enumeração de todas as possíveis permutações dos vértices de um grafo é separá-los em classes de equivalência. Nesta abordagem, os vértices são

separados em conjuntos, de forma que não existe um isomorfismo no qual um vértice está mapeado para um outro vértice contido em um conjunto diferente do seu. Todos os vértices em cada classe de equivalência devem possuir os mesmos valores para os invariantes que estão sendo levados em consideração. Como exemplo, temos dois invariantes de vértices que podem ser usados:

- **Grau dos vértices:** É fácil ver que não existe um isomorfismo no qual um vértice de grau k é mapeado para um vértice de grau $q \neq k$. Em grafos nos quais os graus dos vértices são bastante variados, esse invariante pode oferecer uma boa divisão do conjunto dos vértices de forma a diminuir consideravelmente o espaço de busca. Mas, nem sempre isso ocorre. Por exemplo, para grafos regulares este invariante simplesmente não ajuda em nada.
- **Matriz de caminhos mais curtos:** Para cada vértice v , a matriz dos caminhos mais curtos de todos os pares contém um subconjunto de $n - 1$ (n é a cardinalidade do conjunto de vértices do grafo em questão) valores representando as distâncias entre v e cada outro vértice do grafo. Quaisquer dois vértices associados em um isomorfismo definem exatamente o mesmo subconjunto de distâncias.

Existem outros invariantes relacionados ao problema de isomorfismo além destes citados acima [21].

Implementação e casos fáceis

Existem muitos algoritmos para resolver o caso geral do problema de isomorfismo que, apesar de seu pior caso ter complexidade exponencial, em geral resolvem instâncias grandes do problema em um tempo razoável. Experimentos computacionais sobre isomorfismo podem ser encontrados em [21]. Considerações sobre a dificuldade em se encontrar grafos “difíceis” (ou “quase isomorfos”) para a realização de testes de isomorfismo podem ser encontradas em [41].

Quando os graus dos vértices de ambos os grafos são limitados por uma constante, é possível resolver o problema de isomorfismo em tempo polinomial [35]. Para determinadas classes de grafos o problema de isomorfismo pode ser resolvido em tempo polinomial. Por exemplo, para grafos planares existe um algoritmo linear no número de vértices [28].

2.1.1 Relação entre Isomorfismo e o PCCA

Dados dois grafos $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$, tais que $|V_I| = |V_M|$ e $|A_I| = |A_M|$, a partir de uma solução ótima do PCCA para estes dois grafos, é possível saber se $G_I \cong G_M$. É claro que se

$|V_I| \neq |V_A|$ ou $|A_I| \neq |A_M|$, então G_I e G_M não são isomorfos.

Para fazer a redução do problema de isomorfismo para o PCCA, primeiro precisamos definir as funções de custo que utilizaremos no PCCA. Seja $g(u)$ o grau do vértice u . A função de custo das associações dos vértices de V_I para os clusters de V_M é assim definida:

$$c(i, k) = |g(i) - g(k)|, \text{ para todo } i \in V_I \text{ e } k \in V_M.$$

A função de custo das associações das arestas de A_I para os pares de clusters de V_M é assim definida (para todo $ij \in A_I$ e $k, l \in V_M$):

$$d(ij, k, l) = \begin{cases} 1, & \text{se } k = l \\ 0, & \text{caso contrário.} \end{cases}$$

Seja $\Phi(G_I, G_M)$ o custo de uma solução ótima do PCCA, utilizando as funções c e d para atribuir custos às associações dos vértices e arestas, respectivamente.

Proposição 2.1.1 *Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ dois grafos, tais que $|V_I| = |V_M|$ e $|A_I| = |A_M|$. Os grafos G_I e G_M são isomorfos se e somente se $\Phi(G_I, G_M) = 0$.*

Prova. ($\implies$) Se $G_I \cong G_M$, então existe uma bijeção $h : V_I \rightarrow V_M$, tal que $ij \in A_I$ se e somente se $\{h(i), h(j)\} \in A_M$. Considere um mapeamento $\mathcal{M}$ no qual $i \in V_I$ está associado a $k \in V_M$ se e somente se $h(i) = k$. Assim, $c(i, k) = 0$ para todo $i \in V_I$ e $k \in V_M$, tal que i está associado a k em $\mathcal{M}$, pois i e k possuem o mesmo grau. Note que da forma como $\mathcal{M}$ é construído, para toda aresta $ij \in A_I$ sabemos que ij está associada à aresta $kl \in A_M$, sendo que $k = h(i)$ e $l = h(j)$. Portanto, $\mathcal{M}$ atende à restrição de cobertura das arestas de A_I . Além disso, não existe aresta $ij \in A_I$ associada a um cluster. Portanto, temos que $d(ij, k, l) = 0$, para todas as associações de arestas contidas em $\mathcal{M}$.

Vimos que $\mathcal{M}$ é viável e tem custo zero. Note que o valor de $c(i, k)$ é não negativo para todo $i \in V_I$ e $k \in V_M$, e o valor de $d(ij, k, l)$ é não negativo para todo $ij \in A_I$ e $k, l \in V_M$. Ou seja, $\Phi(G_I, G_M) \geq 0$. Logo, $\mathcal{M}$ é ótimo.

($\impliedby$) Se $\Phi(G_I, G_M) = 0$, então existe um mapeamento ótimo $\mathcal{M}$ de G_I para G_M tal que o custo de $\mathcal{M}$ é zero. Considere os seguintes fatos:

fato 1: Sabemos que: $\Phi(G_I, G_M) = 0$; $c(i, k) \geq 0$, para todo $i \in V_I$ e $k \in V_M$; $d(ij, k, l) \geq 0$,

para todo $ij \in A_I$ e $k, l \in V_M$. Logo, nenhuma aresta $ij \in A_I$ está associada a algum cluster $k \in V_M$ em $\mathcal{M}$, pois $d(ij, k, k) = 1$, para todo $ij \in A_I$ e $k \in V_M$.

fato 2: Para todo cluster $k \in V_M$, vale que $|\lambda(k)| = 1$, sendo que $\lambda(k)$ é o conjunto de vértices de V_I associados a $k \in V_M$. Suponha que $|\lambda(k)| > 1$, para algum $k \in V_M$. Logo, ou existe pelo menos uma aresta associada a k , contradizendo o fato 1, ou $G_I[\lambda(k)]$ é desconexo, contradizendo a restrição de conexidade que toda solução viável do PCCA deve atender (sendo que $G_I[\lambda(k)]$ é o subgrafo de G_I induzido por $\lambda(k)$). Suponha agora que $|\lambda(k)| = 0$, para algum $k \in V_M$. Como $|V_M| = |V_I|$ e todos os vértices de V_I estão associados a algum cluster de V_M , então existe $|\lambda(l)| > 1$, para algum $l \in V_M$. Mas, já verificamos que não ocorre $|\lambda(k)| > 1$, para todo $k \in V_M$. Logo, $|\lambda(k)| = 1$, para todo cluster $k \in V_M$.

Vamos agora construir uma bijeção $h : V_I \rightarrow V_M$, e verificar que $ij \in A_I$ se e somente se $\{h(i), h(j)\} \in A_M$. Seja $h(i) = k$ se e somente se i está associado a k em $\mathcal{M}$, para todo $i \in V_I$ e $k \in V_M$. Pelo fato 2, sabemos que $h(i) \neq h(j)$, para todo $i, j \in V_I$. Portanto, h é um mapeamento um-para-um do conjunto V_I para V_M . Resta provar que as adjacências contidas em G_I são mantidas em G_M se utilizarmos a bijeção h .

Suponha que existe uma aresta $ij \in A_I$, tal que $\{h(i), h(j)\} \notin A_M$. Como em $\mathcal{M}$ o vértice i está associado ao cluster $h(i)$, bem como o vértice j está associado ao cluster $h(j)$, então o mapeamento $\mathcal{M}$ não atende a restrição de cobertura de arestas, pois a aresta ij está descoberta. Uma contradição com o fato de $\mathcal{M}$ ser um mapeamento viável do PCCA.

Suponha agora que existe uma aresta $\{h(i), h(j)\} \in A_M$ tal que $ij \notin A_I$. Como $|A_I| = |A_M|$, então ou (i) existe alguma aresta $kl \in A_M$ para a qual duas ou mais arestas de A_I estão associadas, ou (ii) existe um cluster $k \in V_M$ para o qual uma ou mais arestas de A_I estão associadas. Suponha que (i) é verdadeiro. Para que duas ou mais arestas de A_I sejam associadas a uma mesma aresta $kl \in A_M$, é necessário que $|\lambda(k)| > 1$ ou $|\lambda(l)| > 1$, contradizendo o fato 2. Suponha então que (ii) é verdadeiro. Temos agora uma contradição com o fato 1. Conclui-se então que $G_I \cong G_M$. ■

Pela proposição 2.1.1, concluímos que o PCCA é mais difícil, do ponto de vista de complexidade computacional, que o problema de isomorfismo. Pois, a existência de um algoritmo polinomial para o PCCA para uma determinada classe $\mathcal{C}$ de grafos implicaria na existência de um algoritmo polinomial para o problema de isomorfismo para a classe $\mathcal{C}$ de grafos. No entanto, o contrário não é verdadeiro. Ou seja, a existência de um algoritmo polinomial para o problema de isomorfismo para

alguma classe $\mathcal{C}$ de grafos não implicaria na existência de um algoritmo polinomial para o PCCA para a classe $\mathcal{C}$ de grafos. Por exemplo: conforme mencionado anteriormente, para grafos planares o problema de isomorfismo é solúvel em tempo polinomial, já o PCCA *para árvores* (um caso específico de grafo planar), conforme provaremos no capítulo 3, é $\mathcal{NP}$ -difícil.

Como até o momento não se conhece uma prova de que o problema de isomorfismo é $\mathcal{NP}$ -completo, a proposição 2.1.1 não nos ajuda a provar que o PCCA é $\mathcal{NP}$ -difícil.

2.2 Problemas de Partição de Grafos

O PCIG pode ser formulado como um problema de partição, pois a associação dos vértices de V_I a clusters nada mais é do que uma partição do conjunto V_I , na qual cada parte será associada a um cluster diferente de V_M . Além do PCIG, existem outros diversos problemas de partição. A seguir listamos alguns exemplos:

Partição em Florestas: Dado um grafo $G = (V, A)$ e um inteiro positivo $K \leq |V|$, determinar se é possível particionar V em $k \leq K$ conjuntos disjuntos $V_1, V_2, \dots, V_k$ tais que, para $i = 1, 2, \dots, k$, o subgrafo de G induzido por V_i é acíclico. Este problema é $\mathcal{NP}$ -completo [23].

Partição em Cliques: Dado um grafo $G = (V, A)$ e um inteiro positivo $K \leq |V|$, determinar se é possível particionar V em $k \leq K$ conjuntos disjuntos $V_1, V_2, \dots, V_k$, tais que, para $1 \leq i \leq k$, o subgrafo de G induzido por V_i é completo. Este problema é $\mathcal{NP}$ -completo, embora existam casos particulares nos quais é possível resolvê-lo em tempo polinomial [23].

Partição em Emparelhamentos Perfeitos: Dado um grafo $G = (V, A)$ e um inteiro positivo $K \leq |V|$, determinar se é possível particionar V em $k \leq K$ conjuntos disjuntos $V_1, V_2, \dots, V_k$, tais que, para $1 \leq i \leq k$, o subgrafo de G induzido por V_i é um emparelhamento perfeito. Este problema é $\mathcal{NP}$ -completo [23].

Geralmente, os problemas de partição de grafos têm como objetivo particionar o conjunto de vértices de um grafo em um número k de partes, de forma que cada parte possua uma determinada propriedade. Aqui cabe notar que o PCCC (PCIG com Conexidade e Cobertura de Clusters) é também um problema dessa natureza (cada parte deve ser conexa). Porém, é exigido que a partição do conjunto de vértices otimize uma função objetivo. Do ponto de vista de viabilidade, este problema é trivial. Pois, sempre existe uma partição de V_I em $|V_M|$ partes conexas (supondo que $|V_I| \geq |V_M|$ e que G possui um número de componentes não superior a k).

No caso do PCCA (PCIG com Conexidade e Cobertura de Arestas), mesmo existindo uma restrição a mais na estrutura da partição (a cobertura das arestas de A_I), o problema também é trivial do ponto de vista de viabilidade. O fato de não exigir-se que os clusters sejam cobertos, garante que sempre existirá pelo menos $|V_M|$ soluções viáveis, cada uma delas com todos os vértices de V_I associados a um único cluster.

Figura 2.2: À esquerda G_I e à direita G_M

Figura 2.3: $|V_M|$ Soluções viáveis.

Os três exemplos que fornecemos parecem não ser muito semelhantes ao PCCA, no sentido de que neles são exigidas estruturas muito mais complexas do que simplesmente conexidade nas partes. Além disso, o objetivo destes problemas é apenas verificar se uma determinada estrutura existe em um grafo, sem otimizar nenhuma função objetivo. Já no caso do PCCA, a estrutura exigida pode ser facilmente verificada quanto à existência. Porém, deseja-se otimizar uma função objetivo.

O próximo problema que veremos se assemelha mais ao PCCA, no sentido de que cada parte da partição deve ser conexa. Porém, o objetivo é encontrar uma partição que seja o mais balanceada possível.

Max 2-Partição Conexa Balanceada (PCB₂): Dados um grafo $G = (V, A)$ e uma função $w : V \rightarrow \mathbb{Z}_+$, encontrar uma (V_1, V_2) -partição de V , tal que os subgrafos de G induzidos por V_1 e V_2 sejam conexos, e que maximize $\min\{\sum_{u \in V_1} w(u), \sum_{u \in V_2} w(u)\}$.

Um caso particular desse problema é quando $w(v) = 1$, para todo $v \in V$. Esta versão do problema é denominada PCB₂ *sem pesos*.

Sabe-se que o PCB_2 é polinomial quando o grafo de entrada é uma árvore [5], e que o PCB_2 *sem pesos* é polinomial quando o grafo de entrada é 2-conexo [22, 33]. O PCB_2 é $\mathcal{NP}$ -difícil nos seguintes casos (para os seguintes grafos de entrada): grafos conexos [7]; grafos bipartidos [17]; grafos planares [4]; grafos 2-conexos [45] (inclusive, a versão de decisão deste problema é fortemente $\mathcal{NP}$ -completa, ou seja, o problema não admite algoritmo pseudopolinomial, a menos que $\mathcal{P} = \mathcal{NP}$). Nos dois primeiros casos, mesmo para o PCB_2 *sem pesos* o problema é $\mathcal{NP}$ -difícil.

Uma versão mais geral desse problema é o Max q -Partição Conexa Balanceada (PCB_q), no qual deseja-se particionar o conjunto de vértices em $q \geq 2$ partes conexas, tal que a cardinalidade da menor delas seja o maior possível. Como o PCB_2 é um caso particular do PCB_q , segue imediatamente que o PCB_q é $\mathcal{NP}$ -difícil. Um resultado não imediato e bastante interessante é que, assim como no PCB_2 , o PCB_q pode ser resolvido em tempo polinomial quando o grafo de entrada é uma árvore [43].

Para mais detalhes sobre o problema de partições conexas balanceadas (aplicações, heurísticas e algoritmos de aproximação) consulte [34, 45].

Existem diversas variações do problema de particionar grafos. No artigo *The partition problem* [10], de Chopra e Rao, é feito um estudo poliédrico de muitas destas variações. Uma das variações estudadas em tal artigo é a seguinte:

Problema de 2-Partição de Grafos (PP_2G): Dados um grafo $G = (V, A)$ e uma função $w : A \rightarrow \mathbb{Z}_+$, encontrar uma partição de V em dois conjuntos que maximize $\sum_{e \in E} w(e)$, sendo que $E \subseteq A$ é o conjunto das arestas com um extremo em cada conjunto da partição.

Ao contrário do PCB_2 , no PP_2G não é exigido conexidade nas partes. No artigo mencionado, os autores afirmam que o PP_2G pode ser formulado como um problema de encontrar um corte máximo em um grafo, cuja versão de decisão é a seguinte:

Corte Máximo: Dados um grafo $G = (V, A)$, pesos $w_a \in \mathbb{Z}_+$, para todo $a \in A$, e um inteiro positivo K , determinar se existe uma partição de V em dois conjuntos V_1 e V_2 , tais que a soma dos pesos das arestas de A que possuem um extremo em V_1 e outro em V_2 é pelo menos K .

As arestas que possuem um extremo em V_1 e outro em V_2 formam um *corte* em G . Dizemos que o peso de um corte é a soma dos pesos das arestas contidas nele. Um corte em G é máximo se não existe nenhum corte em G com peso maior. O problema “Corte Máximo” é $\mathcal{NP}$ -completo [23].

Curiosamente, o PCIG também possui uma certa relação com o problema de corte máximo. Vamos provar que o PCIG é $\mathcal{NP}$ -difícil, fazendo uma redução do problema “Corte Máximo”.

Proposição 2.2.1 *O PCIG é $\mathcal{NP}$ -difícil.*

Prova. Considere uma instância do problema “Corte Máximo”: um grafo $G = (V, A)$, um inteiro positivo K e pesos $w_a \in \mathbb{Z}_+$, para todo $a \in A$. A partir desta instância do problema “Corte Máximo”, construiremos uma instância do PCIG.

Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ dois grafos tais que, $|V_M| = 2$, $A_M = \emptyset$ e $G_I = G$. Definimos a função de custo das associações dos vértices de V_I para os clusters de V_M da seguinte maneira:

$$c(i, k) = 0, \text{ para todo } i \in V_I \text{ e } k \in V_M.$$

Definimos a função de custo das associações das arestas de A_I para os pares de clusters de V_M da seguinte maneira (para todo $ij \in A_I$ e $k, l \in V_M$):

$$d(ij, k, l) = \begin{cases} w_{ij}, & \text{se } k = l \\ 0, & \text{caso contrário.} \end{cases}$$

Seja $\Phi(G_I, G_M)$ o valor da função objetivo do PCIG para um mapeamento ótimo $\mathcal{M}$ de V_I para V_M , e seja $CM(G)$ o peso de um corte máximo em G . Provaremos que

$$\Phi(G_I, G_M) = \sum_{ij \in A_I} w_{ij} - CM(G), \quad (2.1)$$

e concluiremos que existe um corte de peso pelo menos K em G se e somente se

$$\Phi(G_I, G_M) \leq \sum_{ij \in A_I} w_{ij} - K.$$

Com isso, teremos mostrado que o PCIG é $\mathcal{NP}$ -difícil. Se $A_I = \emptyset$, claramente vale (2.1). Assim, vamos supor que $A_I \neq \emptyset$.

Suponha que $\Phi(G_I, G_M) < \sum_{ij \in A_I} w_{ij} - CM(G)$. Sejam k e l , tais que $k \neq l$, os dois clusters de V_M , e seja $S \subseteq A_I$ o conjunto das arestas que estão associadas ao par de clusters k e l em $\mathcal{M}$. Sejam $V_k \subseteq V_I$ e $V_l \subseteq V_I$, tais que V_k é o conjunto de vértices associados ao cluster k e V_l é o conjunto de

vértices associados ao cluster l .

Primeiro, provaremos que $V_k \neq \emptyset$ e $V_l \neq \emptyset$. Suponha, sem perda de generalidade, que $V_k = \emptyset$. Seja $ij \in A_I$ uma aresta associada ao cluster l . Considere o mapeamento $\mathcal{M}'$ no qual i está associado a k e j está associado a l . As demais associações contidas em $\mathcal{M}'$ são as mesmas de $\mathcal{M}$. Ou seja, em $\mathcal{M}'$ a aresta ij está associada ao par de clusters k e l . Como $k \neq l$, o custo da associação da aresta ij em $\mathcal{M}'$ é zero. Já em $\mathcal{M}$ o custo da associação da aresta ij é $w_{ij} > 0$. Dessa forma, segue que o custo de $\mathcal{M}'$ é igual ao custo de $\mathcal{M}$ menos o peso w_{ij} da aresta ij , contradizendo o fato de $\mathcal{M}$ ser um mapeamento ótimo. Logo $V_k \neq \emptyset$ e $V_l \neq \emptyset$.

Como $V_k \neq \emptyset$, $V_l \neq \emptyset$ e $G_I = G$, segue que S é um corte em G . Claramente, $\Phi(G_I, G_M) = \sum_{ij \in A_I} w_{ij} - p(S)$, sendo que $p(S)$ é o peso do corte S . Como estamos supondo que $\Phi(G_I, G_M) < \sum_{ij \in A_I} w_{ij} - CM(G)$, segue que

$$\sum_{ij \in A_I} w_{ij} - p(S) < \sum_{ij \in A_I} w_{ij} - CM(G).$$

Logo, $p(S) > CM(G)$. Ou seja, S é um corte com peso maior que $CM(G)$, contradizendo o fato de $CM(G)$ ser o peso de um corte máximo em G .

Agora suponha que $\Phi(G_I, G_M) > \sum_{ij \in A_I} w_{ij} - CM(G)$. Seja $C \subseteq A_I$ o conjunto das arestas de um corte máximo em $G = G_I$, que particiona V_I em V_1 e V_2 . Considere o mapeamento $\mathcal{M}^*$ no qual todos os vértices de V_1 estão associados a k , e todos os vértices de V_2 estão associados a l . Claramente o custo de $\mathcal{M}^*$ é dado por $\sum_{ij \in A_I} w_{ij} - CM(G) < \Phi(G_I, G_M)$, contradizendo o fato de que $\Phi(G_I, G_M)$ é o custo de um mapeamento ótimo.

Provamos que $\Phi(G_I, G_M) = \sum_{ij \in A_I} w_{ij} - CM(G)$. Como $CM(G)$ é o peso de um corte máximo em G , concluímos que G tem um corte de peso pelo menos K se e somente se $\Phi(G_I, G_M) \leq \sum_{ij \in A_I} w_{ij} - K$. Como o problema de determinar se um grafo possui ou não um corte de peso pelo menos K é $\mathcal{NP}$ -completo, segue que o PCIG é $\mathcal{NP}$ -difícil. ■

Existem inúmeras variações do problema de partição, bem como diversas abordagens diferentes de resolução do problema. Para o leitor interessado em mais informações sobre o assunto, recomendamos os seguintes textos [12, 13, 18, 19, 26, 27, 31].

Figura 2.4: Um corte máximo em G_I , sendo que neste exemplo o peso de todas as arestas são iguais. Ao lado esquerdo o cluster k e ao lado direito o cluster l . As arestas em negrito, que pertencem a um corte máximo de G_I , estão associadas ao par de clusters k e l .

2.3 Formulação Não Linear do PCCC

Boeres [6] propôs uma formulação do PCCC (PCIG com Conexidade e Cobertura de Clusters). Tal formulação foi inspirada na aplicação de reconhecimento estrutural de objetos, mas não se restringe somente a esta aplicação. A seguir, apresentamos a formulação não linear desenvolvida por Boeres [6] para o PCCC. Modificamos alguns detalhes da formulação original a fim de unificar as notações e manter uma coerência entre as diferentes formulações que apresentamos ao longo de todo o texto. Mesmo assim, as idéias originais foram essencialmente mantidas.

Dados e Variáveis

São dados dois grafos simples não-orientados $G_M = (V_M, A_M)$ e $G_I = (V_I, A_I)$ e duas funções $c : V_I \times V_M \rightarrow \mathbb{R}_+$ e $d : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$. Por simplicidade, utilizaremos a notação c_{ik} no lugar de $c(i, k)$, para todo $i \in V_I$ e $k \in V_M$. Utilizaremos d_{kl}^{ij} no lugar de $d(ij, k, l)$, para todo $ij \in A_I$ e $k, l \in V_M$.

A função c indica, através de um número real, uma medida de similaridade entre um vértice $i \in V_I$ e um cluster $k \in V_M$. Como o PCCC foi definido como um problema de minimização, a função c é definida de forma que, quanto maior for a similaridade entre o vértice i e o cluster k , menor será o custo de c_{ik} . Podemos interpretar que o custo de c_{ik} equivale ao grau de dissimilaridade entre i e k . A função d é definida analogamente a c , porém d equivale ao grau de dissimilaridade entre a aresta ij e o par de clusters k e l .

O termo similaridade é um tanto vago quando não especificamos a aplicação que está sendo levada em consideração. No caso da aplicação de reconhecimento estrutural de objetos, no capítulo 1 fornecemos um exemplo mais concreto de como definir as funções c e d .

Introduzimos as variáveis $x_{ik} \in \{0, 1\}$, para todo $i \in V_I$ e $k \in V_M$, tais que:

$$x_{ik} = \begin{cases} 1, & \text{se o vértice } i \text{ está associado ao cluster } k \\ 0, & \text{caso contrário.} \end{cases}$$

A menos de menção contrária, utilizaremos estes mesmos dados e variáveis ao longo de todo o texto.

Formulação

Utilizaremos a notação $\lambda(k)$ para designar o conjunto de vértices contidos em V_I associados ao cluster $k \in V_M$. Para representar o conjunto de clusters de V_M para os quais $i \in V_I$ está associado, utilizaremos $\lambda^{-1}(i)$. Temos a seguinte formulação para o PCCC:

$$\begin{aligned} \min z = & \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik} + \sum_{ij \in A_I} \sum_{k, l \in V_M} d_{kl}^{ij} \cdot x_{ik} \cdot x_{jl} \\ \text{sujeito a} & \\ & \text{(i) } |\lambda^{-1}(i)| = 1, \text{ para todo } i \in V_I. \\ & \text{(ii) } G_I[\lambda(k)] \text{ é conexo, para todo } k \in V_M. \\ & \text{(iii) } |\lambda(k)| \geq 1, \text{ para todo } k \in V_M. \\ & \text{(iv) } x_{ik} \in \{0, 1\}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \end{aligned}$$

O objetivo é minimizar o custo do mapeamento de V_I para V_M . Lembramos que o custo da associação entre um vértice e um cluster é o grau de dissimilaridade entre os mesmos. Da mesma forma, o custo da associação entre uma aresta e um par de clusters é o grau de dissimilaridade entre os mesmos.

Na restrição (i), exigimos que todo vértice de V_I seja associado a exatamente um cluster. Na restrição (ii), exigimos que o subgrafo de G_I induzido por $\lambda(k)$ deve ser conexo para todo $k \in V_M$ (restrição de conexidade). Na restrição (iii), exigimos que deve existir pelo menos um vértice de V_I associado a k , para todo cluster $k \in V_M$ (cobertura de clusters).

Figura 2.5: Exemplo de uma solução viável do PCCC. O grafo G_M aparece pontilhado. O grafo G_I está desenhado sobre o grafo G_M , indicando quais vértices estão associados a cada cluster de V_M . Repare que para cada cluster $k \in V_M$, existe pelo menos um vértice de V_I associado e $G_I[\lambda(k)]$ é conexo.

O estudo de formulações para o PCIG e suas variações, como por exemplo o PCCC, pode auxiliar no desenvolvimento de algoritmos para o problema. Mas, se quisermos utilizar técnicas de resolução que envolvem programação linear, precisamos de formulações lineares. Definir inequações lineares que representem as restrições (i) e (iii) parece ser uma tarefa bem simples. No entanto, para encontrar inequações que representem a restrição (ii) é necessário um pouco mais de trabalho. Outro ponto importante é a transformação da função objetivo apresentada nesta seção em uma função linear. Na próxima seção mostraremos uma formulação linear baseada na que acabamos de mostrar. Veremos como tal formulação trata cada uma dessas questões.

2.4 Formulação Linear do PCCC *sem pesos nas arestas*

Baseando-se na formulação não linear para o PCCC proposta por Boeres [6], Duarte [16] chegou a uma formulação linear que será apresentada nesta seção. Como a formulação desenvolvida por Duarte [16] não define uma função de custo para as associações entre as arestas e pares de clusters, vamos nos referir a esta versão do problema como PCCC *sem pesos nas arestas*.

Formulação

Temos a seguinte formulação para o PCCC *sem pesos nas arestas*:

$$\begin{aligned}
 \min z = & \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik} \\
 \text{sujeito a} & \\
 (F_1) \quad & \text{(i) } \sum_{k \in V_M} x_{ik} = 1, \text{ para todo } i \in V_I. \\
 & \text{(ii) } \sum_{i \in V_I} x_{ik} \geq 1, \text{ para todo } k \in V_M. \\
 & \text{(iii) } \sum_{i \in S} x_{ik} - \sum_{j \in V_I \setminus S} x_{jk} \leq |S| - 1, \forall k \in V_M \text{ e } S \subseteq V_I : G_I[S] \text{ é desconexo.} \\
 & \text{(iv) } x_{ik} \in \{0, 1\}, \text{ para todo } i \in V_I \text{ e } k \in V_M.
 \end{aligned}$$

O objetivo é minimizar o custo do mapeamento de V_I para V_M . Nesta formulação não é definido um custo para as associações entre arestas e pares de clusters. Isto torna mais simples a linearização da função objetivo. Em contrapartida, parte da informação contida nas arestas é desprezada. Conforme veremos no próximo capítulo, uma alternativa para considerar os custos das associações entre arestas e pares de clusters na função objetivo é introduzir novas variáveis na formulação. Por enquanto nos preocuparemos apenas com o custo das associações entre vértices e clusters.

Na restrição (i), exigimos que todo vértice de V_I seja associado a exatamente um cluster. Na restrição (ii), exigimos que exista pelo menos um vértice de V_I associado a k , para todo cluster k de V_M (cobertura de clusters). Na restrição (iii), exigimos que o subgrafo de G_I induzido por $\lambda(k)$ seja conexo, para todo $k \in V_M$ (restrição de conexidade – veja a figura 2.4).

Embora a formulação apresentada nesta seção seja linear, seu uso é impraticável, pois a quantidade de restrições do tipo (iii) pode ser exponencial, dependendo da estrutura do grafo G_I . Uma abordagem alternativa para contornar este problema é considerar inicialmente apenas as restrições (i) e (ii) e tratar a restrição (iii) através da inclusão de planos-de-cortes. A seguir, explicamos em mais detalhes esta outra abordagem.

2.4.1 Corte-de-conexidade

Considere o problema P , cuja formulação é equivalente à formulação (F_1) sem as restrições do tipo (iii). Claramente P é uma relaxação do PCCC *sem pesos nas arestas*, pois em P não é exigida a restrição de conexidade. Ou seja, toda solução viável do PCCC *sem pesos nas arestas* é viável de P . Porém, o contrário não é verdadeiro. As soluções que não atendem à restrição de conexidade só

Figura 2.6: Considere o exemplo no qual o grafo acima corresponde a G_I . Indicamos através dos círculos pontilhados um conjunto S , tal que $G_I[S]$ é desconexo. Note que para que a restrição (iii) seja satisfeita, deve valer que $\lambda(k) \neq S$, para todo $k \in V_M$. Para um $k \in V_M$ qualquer, dois exemplos que não violam a restrição (iii) seriam: $\lambda(k) = S \setminus \{i\}$ ou $\lambda(k) = S \cup \{j\}$.

são viáveis de P . Logo, uma solução ótima de P é um limitante inferior para o valor de uma solução do PCCC *sem pesos nas arestas*. Uma consequência disso é que se uma solução ótima de P atende à restrição de conexidade ($G_I[\lambda(k)]$ é conexo para todo $k \in V_M$), então tal solução também é ótima para o PCCC *sem pesos nas arestas*.

Vimos que a restrição de conexidade pode ser descrita através da seguinte inequação:

$$\sum_{i \in S} x_{ik} - \sum_{j \in V_I \setminus S} x_{jk} \leq |S| - 1, \text{ para todo } k \in V_M \text{ e } S \subseteq V_I, \text{ tal que } G_I[S] \text{ é desconexo.}$$

Qualquer subconjunto não vazio de V_I é um possível candidato a $\lambda(k)$, para qualquer $k \in V_M$. Este fato pode nos levar a concluir que a quantidade de cortes-de-conexidade necessários para obter uma solução que não viole a restrição de conexidade não seja polinomial no tamanho da entrada. Talvez não seja necessário incluir a restrição acima para todo $S \subseteq V_I$ e $k \in V_M$. Se tivermos sorte, possivelmente apenas um subconjunto não muito grande dessas inequações seja necessário para obter-se uma solução ótima que não viole a restrição de conexidade.

A idéia desta abordagem alternativa é utilizar inicialmente a formulação de P para obter uma solução ótima x^* . Caso a solução x^* viole a restrição de conexidade, então encontramos novas restrições violadas por x^* e as juntamos às restrições de P , gerando um novo problema P^1 , de modo que x^* não seja uma solução viável de P^1 . Chamaremos estas restrições de *cortes-de-conexidade*, pois tais restrições servem para “cortar” (ou descartar) as soluções que violam a restrição de conexidade.

Então, utilizamos P^1 para obter uma nova solução ótima x^1 . Caso x^1 viole a restrição de conexidade, juntamos algumas (pelo menos uma) inequações violadas por x^1 às restrições de P^1 , gerando um novo problema P^2 . E assim sucessivamente, até que seja encontrada uma solução ótima que não viole a restrição de conexidade. Vejamos como calcular tais cortes-de-conexidade.

Dada uma solução x^* , para todo $k \in V_M$, tal que $G_I[\lambda(k)]$ não seja conexo, temos a seguinte restrição violada por x^* :

$$\sum_{i \in \lambda(k)} x_{ik} - \sum_{j \in V_I \setminus \lambda(k)} x_{jk} \leq |\lambda(k)| - 1. \quad (4.2)$$

Vamos fixar um k tal que $G_I[\lambda(k)]$ não é conexo na solução x^* . Claramente a solução x^* viola a restrição (4.2), pois $\sum_{i \in \lambda(k)} x_{ik} = |\lambda(k)|$ e $\sum_{j \in V_I \setminus \lambda(k)} x_{jk} = 0$. Note que qualquer outra solução que contenha um conjunto de vértices associados a k diferente do atual (na solução x^*) não violará a restrição (4.2). Basta observar que qualquer mapeamento $\mathcal{M}$ que atenda a uma das condições abaixo, não viola a restrição (4.2) para o k fixado.

- $\mathcal{M}$ contém pelo menos um vértice $j \in V_I$ associado ao cluster k , tal que $j \notin \lambda(k)$ na solução x^* , aumentando assim o valor do termo $\sum_{j \in V_I \setminus \lambda(k)} x_{jk}$ em pelo menos uma unidade.
- $\mathcal{M}$ contém pelo menos uma associação de algum vértice $i \in V_I$ a um outro cluster $l \in V_M$, tal que $i \in \lambda(k)$ na solução x^* e $l \neq k$, diminuindo o valor do termo $\sum_{i \in \lambda(k)} x_{ik}$ em pelo menos uma unidade.

Dada uma solução x^* de P (possivelmente já acrescido de alguns cortes-de-conexidade), precisamos verificar se x^* atende à restrição de conexidade. Para efetuar esta tarefa, podemos utilizar o algoritmo de busca em largura, que é polinomial no tamanho do grafo no qual será feita a busca [14]. Basta executar, para cada $G_I[\lambda(k)]$, tal que $k \in V_M$, uma busca em largura partindo de um vértice qualquer de $G_I[\lambda(k)]$. Somente as arestas com os dois extremos em $\lambda(k)$ podem ser consideradas na busca. Se todos os vértices de $G_I[\lambda(k)]$ forem alcançados na busca, então $G_I[\lambda(k)]$ é conexo. Caso contrário, $G_I[\lambda(k)]$ é desconexo.

A figura 2.7 ilustra um exemplo onde $G_I[\lambda(l)]$ é conexo e $G_I[\lambda(k)]$ é desconexo, para dois clusters distintos $k, l \in V_M$. Note que o grafo $G_I[\lambda(k)]$ possui duas componentes, portanto, uma busca em largura partindo de um único vértice de $G_I[\lambda(k)]$ jamais alcançará todos os vértices de $G_I[\lambda(k)]$. Já o grafo $G_I[\lambda(l)]$ é conexo. Portanto, uma busca em largura partindo de um vértice qualquer de $G_I[\lambda(l)]$

Figura 2.7: À esquerda temos G_I . As áreas pontilhadas indicam que os vértices 1, 2, 3, e 7 estão associados a um mesmo cluster, digamos k . Os vértices 4, 5 e 8 estão associados a um cluster $l \neq k$. Ao centro temos $G_I[\lambda(k)]$ e à direita temos $G_I[\lambda(l)]$.

sempre alcançará todos os vértices de $G_I[\lambda(l)]$. Neste caso o corte-de-conexidade a ser inserido na formulação é $\sum_{i \in \lambda(k)} x_{ik} - \sum_{j \in V_I \setminus \lambda(k)} x_{jk} \leq 3$.

A seguir, mostramos um pseudo-código para a resolução do PCCC *sem pesos nas arestas* através da inclusão de cortes-de-conexidade. O algoritmo PCCC-SEM-PESOS-NAS-ARESTAS recebe como parâmetros dois grafos G_I e G_M e devolve uma solução ótima do PCCC *sem pesos nas arestas* para os grafos G_I e G_M .

PCCC-SEM-PESOS-NAS-ARESTAS(G_I, G_M)

- 1 Seja $\mathcal{F}$ o conjunto de restrições da formulação para P
- 2 devolva TRATA-CONEXIDADE($G_I, G_M, \mathcal{F}$)

TRATA-CONEXIDADE($G_I, G_M, \mathcal{F}$)

- 1 $x^* \leftarrow \text{RESOLVE-PI}(\mathcal{F})$
- 2 **se** x^* viola a restrição de conexidade
- 3 **então** Seja $\mathcal{C}$ o conjunto dos cortes-de-conexidade violados por x^*
- 4 $\mathcal{F}' \leftarrow \mathcal{F} \cup \mathcal{C}$
- 5 devolva TRATA-CONEXIDADE($G_I, G_M, \mathcal{F}'$)
- 6 **senão** devolva x^*

O algoritmo TRATA-CONEXIDADE faz uso de uma subrotina chamada RESOLVE-PI, que é encarregada de encontrar uma solução inteira que otimiza a função $\sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik}$, levando em

conta as restrições contidas em $\mathcal{F}$. Inicialmente, a formulação $\mathcal{F}$ contém apenas as restrições do problema P , que é uma relaxação do PCCC *sem pesos nas arestas*. A cada chamada recursiva feita a TRATA-CONEXIDADE, caso a solução x^* encontrada por RESOLVE-PI não atenda à restrição de conectividade, são gerados novos cortes-de-conexidade violados por x^* . Estes cortes são “acumulados” até que em alguma chamada a TRATA-CONEXIDADE a solução x^* encontrada por RESOLVE-PI não viole a restrição de conectividade.

Observe que, quando G_I é um grafo completo, não é necessária a inclusão de nenhum corte-de-conexidade, pois qualquer subgrafo $G_I[S]$, tal que $S \subseteq V_I$, é conexo. Neste caso, podemos utilizar um algoritmo guloso, que associa cada $i \in V_M$ a um cluster $k \in V_M$, tal que o custo de c_{ik} seja mínimo. Qualquer solução obtida desta forma é ótima porque cada vértice é associado de maneira ótima, e qualquer subconjunto de associações não influi no restante. Tal procedimento consumiria tempo $O(|V_I| \cdot |V_M|)$. De forma geral, quanto mais denso for o grafo G_I , menor tende a ser a quantidade de cortes-de-conexidade necessários para obter-se uma solução ótima para o PCCC *sem pesos nas arestas*.

A abordagem utilizada pelo algoritmo TRATA-CONEXIDADE é chamada de *método de planos-de-corte*. Dada uma classe de inequações $\mathcal{I}$, em cada iteração do método de planos-de-corte é necessário decidir se uma dada solução viola ou não alguma inequação de $\mathcal{I}$. Em caso afirmativo, deve-se exibir uma inequação de $\mathcal{I}$ violada por tal solução. Este problema é conhecido como o *problema da separação*. O problema de encontrar uma solução ótima que não viole nenhuma inequação de $\mathcal{I}$ é chamado de *problema de otimização*.

Grötschel, Lovász e Schrijver [25] provaram que os problemas de otimização e separação sobre um mesmo poliedro são computacionalmente equivalentes. Este resultado, ao contrário do que aparenta, não nos trás nenhuma informação sobre o comportamento do algoritmo TRATA-CONEXIDADE. Os cortes-de-conexidade que mostramos servem apenas para eliminar soluções inteiras que violam a restrição de conectividade. Repare que na linha 1 do algoritmo temos uma chamada à subrotina RESOLVE-PI, que encontra uma solução inteira para o problema composto pelas restrições de $\mathcal{F}$. Em geral, resolver um problema de programação inteira é $\mathcal{NP}$ -difícil. O ideal seria se soubéssemos resolver eficientemente o problema da separação para uma dada solução qualquer (fracionária ou inteira). Neste caso, através da equivalência entre otimização e separação, poderíamos afirmar que o PCCC *sem pesos nas arestas* está em $\mathcal{P}$.

A seguir, veremos que é possível formular o PCCC *sem pesos nas arestas* com uma quantidade

de restrições polinomial nos tamanhos de G_I e G_M . Esta outra formulação utiliza conceitos de fluxo em redes.

2.4.2 Utilização de Fluxo em Redes para Tratar a Restrição de Conexidade

Uma outra forma de garantir que a restrição de conexidade não seja violada por nenhum $G_I[\lambda(k)]$, tal que $k \in V_M$, é abordar o problema de conexidade como um problema de fluxo em redes. Tal abordagem foi proposta por Duarte [16]. Cada cluster $k \in V_M$ enviará uma certa quantidade de fluxo para um vértice $i \in \lambda(k)$. O fluxo total enviado dos clusters de V_M para V_I deve ser igual a $|V_I|$. Cada vértice de V_I deve consumir uma única unidade de fluxo, sendo que, um vértice $i \in \lambda(k)$ só pode consumir uma unidade de fluxo originado em um cluster $k \in V_M$ se uma das seguintes condições for satisfeita:

- (a) o vértice i recebe fluxo diretamente do cluster k ;
- (b) o vértice i recebe o fluxo de algum vértice $j \in \lambda(k)$ tal que a aresta ij pertence ao conjunto de arestas de $G_I[\lambda(k)]$.

Todo o fluxo gerado em V_M deve ser consumido. Para que todas as unidades de fluxo sejam consumidas, a quantidade máxima de fluxo que um cluster $k \in V_M$ pode enviar para $\lambda(k)$ é $|\lambda(k)|$. Se um cluster $k \in V_M$ envia $|\lambda(k)|$ unidades de fluxo para $\lambda(k)$, este fluxo será totalmente consumido se e somente se $G_I[\lambda(k)]$ é conexo, já que o balanceamento do fluxo é feito através das arestas que possuem seus dois extremos em $\lambda(k)$. Claramente $\sum_{k \in V_M} |\lambda(k)| = |V_I|$. Conforme dito anteriormente, a quantidade total de unidades de fluxo enviadas de V_M para V_I é igual a $|V_I|$. Como esse fluxo é totalmente consumido em V_I , respeitando as regras (a) e (b), temos que $G_I[\lambda(k)]$ é conexo para todo $k \in V_M$. A figura 2.8 ilustra um exemplo de uma solução viável que atende à restrição de conexidade, utilizando a abordagem de fluxo em redes.

Dados e Variáveis

Além dos dados e variáveis definidos na formulação anterior, precisamos de variáveis adicionais para tratar a restrição de conexidade utilizando a abordagem de fluxo. Seja $y_{ik} \in \{0, 1\}$, para todo

Figura 2.8: À esquerda G_M e G_I , respectivamente. As áreas pontilhadas indicam uma solução viável (com conexidade) para os grafos G_I e G_M . Os vértices 1, 2 e 3 estão associados ao cluster a , os vértices 4, 5, 6, 7 e 8 estão associados ao cluster b , e os vértices 9 e 10 estão associados ao cluster c . À direita as setas indicam os fluxos enviados entre os vértices. A quantidade de unidades de fluxo enviadas de G_M para G_I é $|V_I| = 10$, sendo que o vértice 1 recebe 3 unidades do cluster a , o vértice 7 recebe 5 unidades do cluster b e o vértice 9 recebe 2 unidades do cluster c . As setas em G_I indicam quantas unidades de fluxo passam por cada aresta. Apesar de G_I não ser orientado, a passagem do fluxo tem orientação, por isso, uma seta de $i \in V_I$ para $j \in V_I$ indica que uma certa quantidade de fluxo passou pela aresta $ij \in A_I$, saindo de i e chegando a j . Note que cada vértice de V_I consome exatamente uma unidade de fluxo e que as arestas de A_I só transmitem fluxo entre vértices associados a um mesmo cluster.

$i \in V_I$ e $k \in V_M$, tal que:

$$y_{ik} = \begin{cases} 1, & \text{se o vértice } i \in V_I \text{ recebe diretamente o fluxo originado em } k \in V_M \\ 0, & \text{caso contrário.} \end{cases}$$

Definimos também as variáveis $f_{ij}^k \in \mathbb{Z}_+$ que correspondem a quantas unidades do fluxo originado em $k \in V_M$ são enviados de i para j , tal que $k \in V_M$ e $ij \in A_I$. Apesar de G_I não ser orientado, as variáveis f_{ij}^k e f_{ji}^k são diferentes, pois a primeira indica quantas unidades de fluxo originadas em k que passam de i para j , já a segunda indica quantas unidades de fluxo originadas em k que passam de j para i .

Por último, definimos as variáveis $g_{ik} \in \mathbb{Z}_+$, para todo $i \in V_I$ e $k \in V_M$, que correspondem a quantas unidades de fluxo o vértice $i \in V_I$ recebe diretamente do cluster $k \in V_M$.

Formulação

Utilizaremos a notação $\delta_{G_I}(i)$ para designar o subconjunto de V_I que contém todos os vértices adjacentes a i em G_I . Ou seja, $\delta_{G_I}(i) = \{j \mid ij \in A_I\}$. Analogamente, $\delta_{G_M}(k)$ corresponde ao conjunto dos vértices adjacentes a k em G_M . Esta notação será utilizada ao longo de todo o texto.

Temos a seguinte formulação para o PCCC *sem pesos nas arestas*:

$$\begin{aligned} \min z = & \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik} \\ \text{sujeito a} & \\ & \text{(i)} \quad \sum_{k \in V_M} x_{ik} = 1, \text{ para todo } i \in V_I. \\ & \text{(ii)} \quad \sum_{i \in V_I} x_{ik} \geq 1, \text{ para todo } k \in V_M. \\ & \text{(iii)} \quad y_{ik} \leq x_{ik}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \\ & \text{(iv)} \quad \sum_{i \in V_I} y_{ik} \leq 1, \text{ para todo } k \in V_M. \\ & \text{(v)} \quad g_{ik} + \sum_{j \in \delta_{G_I}(i)} (f_{ji}^k - f_{ij}^k) = x_{ik}, \text{ para todo } k \in V_M \text{ e } i \in V_I. \\ & \text{(vi)} \quad \sum_{j \in \delta_{G_I}(i)} f_{ij}^k \leq |V_I| \cdot x_{ik}, \text{ para todo } k \in V_M \text{ e } i \in V_I. \\ & \text{(vii)} \quad \sum_{i \in V_I} \sum_{k \in V_M} g_{ik} = |V_I|. \\ & \text{(viii)} \quad g_{ik} \leq |V_I| \cdot y_{ik}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \\ & \text{(ix)} \quad g_{ik} \geq 0, \text{ para todo } i \in V_I \text{ e } k \in V_M. \\ & \text{(x)} \quad f_{ij}^k \geq 0, \text{ para todo } k \in V_M \text{ e } ij \in A_I. \\ & \text{(xi)} \quad x_{ik} \in \{0, 1\}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \\ & \text{(xii)} \quad y_{ik} \in \{0, 1\}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \end{aligned}$$

O objetivo é minimizar o custo do mapeamento de V_I para V_M . Na restrição (i), exigimos que todo vértice de V_I seja associado a exatamente um cluster. Na restrição (ii), exigimos que para todo cluster k de V_M exista pelo menos um vértice de V_I associado a k (cobertura de clusters). As restrições (iii) a (vii) juntas implicam na exigência de que $G_I[\lambda(k)]$ seja conexo, para todo $k \in V_M$ (restrição de conexidade).

Na restrição (iii), exigimos que um vértice $i \in V_I$ só pode receber fluxo diretamente de um cluster $k \in V_M$ se i estiver associado a k . Na restrição (iv), exigimos que todo cluster $k \in V_M$ deve enviar fluxo diretamente para no máximo um vértice $i \in V_I$. Note que a presença das restrições (ii), (vii) e (viii) implica em $\sum_{i \in V_I} y_{ik} \geq 1$. Logo, vale que $\sum_{i \in V_I} y_{ik} = 1$ em qualquer solução viável.

Na restrição (v), exigimos que, dados um cluster $k \in V_M$ e um vértice $i \in V_I$, caso i esteja associado a k , a soma das unidades do fluxo originado em k que chegam a i deve ter uma unidade a mais que a soma das unidades de fluxo que saem de i . Ou seja, i consome uma unidade do fluxo originado em k de acordo com as regras (a) e (b) descritas no início desta seção. Caso i não esteja associado a k , nenhuma unidade do fluxo originado em k deve passar por uma aresta que tenha i como uma de suas extremidades. Podemos reescrever a restrição (v) da seguinte maneira:

$$g_{ik} + \sum_{j \in \delta_{G_I}^-(i)} f_{ji}^k - \sum_{j \in \delta_{G_I}^+(i)} f_{ij}^k = x_{ik},$$

sendo que o termo $g_{ik} + \sum_{j \in \delta_{G_I}^-(i)} f_{ji}^k$ corresponde ao fluxo total que entra no vértice i , e o termo $\sum_{j \in \delta_{G_I}^+(i)} f_{ij}^k$ corresponde ao fluxo total que sai do vértice i . Note que se $g_{ik} > 0$, então $\sum_{j \in \delta_{G_I}^-(i)} f_{ji}^k = 0$, e se $\sum_{j \in \delta_{G_I}^-(i)} f_{ji}^k > 0$, então $g_{ik} = 0$. Ou seja, o fluxo que entra em um vértice é enviado ou por alguns de seus vizinhos, ou por uma fonte $k \in V_M$ (como um único vértice recebe fluxo diretamente de uma determinada fonte em V_M , se este mesmo vértice receber fluxo também de algum vizinho, este fluxo foi enviado por ele mesmo e, portanto, pode ser cancelado). Nas figuras 2.9 e 2.10 mostramos um exemplo de balanceamento de fluxo.

Figura 2.9: Suponha que o grafo à esquerda corresponde a $G_I[\lambda(k)]$ para algum $k \in V_M$ em uma solução viável qualquer. O vértice 1 recebe 5 unidades de fluxo diretamente do cluster k , logo $g_{1k} = 5$ e $\sum_{j \in \delta_{G_I}^-(1)} f_{j1}^k = 0$. Como $x_{1k} = 1$, para a restrição (v) ser satisfeita, é necessário que $\sum_{j \in \delta_{G_I}^+(1)} f_{1j}^k = 4$. Temos então $f_{12}^k = 2$ e $f_{13}^k = 2$.

Na restrição (vi), exigimos que o fluxo originado em um cluster $k \in V_M$ só pode passar por uma aresta $ij \in A_I$ se $i, j \in \lambda(k)$. Esta restrição garante que se $x_{ik} = 0$ então $f_{ij}^k = 0$. Caso $x_{ik} = 1$ então o valor de f_{ij}^k pode ser tão grande quando necessário, sendo que neste caso o valor de f_{ij}^k

Figura 2.10: Suponha que o grafo à esquerda corresponde a $G_I[\lambda(k)]$ para algum $k \in V_M$ em uma solução viável qualquer. O vértice 4 recebe 1 unidade de fluxo (originado em k) do vértice 2, e mais 1 unidade de fluxo (originado em k) do vértice 3. Ou seja, $f_{24}^k = f_{34}^k = 1$. Temos que $g_{4k} = 0$ e $\sum_{j \in \delta_{G_I}(4)} f_{j4}^k = 2$. Como $x_{4k} = 1$, para a restrição (v) ser satisfeita, é necessário que $\sum_{j \in \delta_{G_I}(1)} f_{1j}^k = 1$. Temos então $f_{15}^k = 1$.

será calculado através da restrição (v). Logo, no lugar de $|V_I|$, no termo $|V_I| \cdot x_{ik}$, podemos utilizar qualquer constante maior ou igual a $|V_I| - 1$, que corresponde à maior quantidade de fluxo que pode passar por uma aresta $ij \in A_I$ em uma solução viável.

Na restrição (vii), exigimos que a quantidade total de unidades de fluxo enviadas dos clusters de V_M aos vértices de V_I deve ser igual a $|V_I|$. Na restrição (viii), exigimos que a quantidade de fluxo recebido por um vértice $i \in V_I$ diretamente de um cluster $k \in V_M$ seja maior que zero somente se i recebe fluxo diretamente de k .

Duarte [16] apresentou resultados computacionais para algumas instâncias do PCCC *sem pesos nas arestas* utilizando a formulação apresentada nesta seção. A maior instância para a qual foi encontrada uma solução ótima tinha tamanho $|V_I| = 95$, $|A_I| = 1434$, $|V_M| = 12$ e $|A_M| = 42$.

2.5 Considerações

Nos próximos capítulos, apresentamos os resultados que obtivemos para o PCCA (PCIG com Conexidade e Cobertura de Arestas). Utilizamos algumas idéias dos trabalhos de Boeres [6] e Duarte [16] para desenvolver nossas formulações para o problema. Como a variação do PCIG que estudamos difere das estudadas por Boeres e Duarte, evidentemente tivemos que introduzir novas idéias. No decorrer do trabalho, optamos por considerar o caso em que os grafos de entrada são árvores (PCCA *para árvores*). A idéia de estudar este caso particular do problema surgiu da observação de que propriedades interessantes, relacionadas à estrutura das árvores, poderiam ser exploradas.

Nosso trabalho difere bastante dos trabalhos mencionados acima, no sentido de que nos focamos em estudar o PCCA como um problema de otimização combinatória, sem assumir nenhum compro-

misso com a aplicação de reconhecimento estrutural de objetos. Ao contrário, os trabalhos de Boeres e Duarte tiveram esta aplicação como foco principal.

Capítulo 3

PCIG com Cobertura de Arestas (PCCA)

Focamos este capítulo no estudo do PCCA (PCIG com Conexidade e Cobertura de Arestas). Primeiro provamos que o problema é $\mathcal{NP}$ -difícil e depois apresentamos uma formulação para o mesmo.

3.1 Complexidade

Através da redução de um problema $\mathcal{NP}$ -completo chamado *Clustering* (Agrupamento) [23], provaremos que o PCCA é $\mathcal{NP}$ -difícil.

Clustering: Dados um conjunto finito S , uma função $p : S \times S \rightarrow \mathbb{Z}_+$ que atribui custos para cada par $u, v \in S$, e dois inteiros positivos n e b , determinar se é possível particionar S em n conjuntos disjuntos $S_1, S_2, \dots, S_n$ tais que, para $1 \leq i \leq n$ e todo par $u, v \in S_i$, vale que $p(u, v) \leq b$.

Proposição 3.1.1 *O PCCA é $\mathcal{NP}$ -difícil.*

Prova. Considere uma instância $\mathcal{I}$ do problema de clustering, na qual são dados um conjunto S , uma função $p : S \times S \rightarrow \mathbb{Z}_+$ e dois inteiros positivos n e b . A partir de $\mathcal{I}$, construiremos uma instância do PCCA.

Seja $G_I = (V_I, A_I)$ um grafo completo, tal que $V_I = S$. Para cada aresta $ij \in A_I$, definimos um custo $w_{ij} = p(i, j)$. Seja $G_M = (V_M, A_M)$ um $\mathcal{K}_n$ ¹.

¹ $\mathcal{K}_n$ denota um grafo completo com n vértices. Esta notação é comumente encontrada na literatura.

Definimos a função $c : V_I \times V_M \rightarrow \mathbb{R}_+$ de custo das associações dos vértices de V_I para os clusters de V_M da seguinte maneira:

$$c(i, k) = 0, \text{ para todo } i \in V_I \text{ e } k \in V_M.$$

Definimos a função $d : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$ de custo das associações das arestas de A_I para os pares de clusters de V_M da seguinte maneira (para todo $ij \in A_I$ e $k, l \in V_M$):

$$d(ij, k, l) = \begin{cases} 1, & \text{se } k = l \text{ e } w_{ij} > b \\ 0, & \text{caso contrário.} \end{cases}$$

Pela definição de c e d , o custo de associar as arestas que possuem custo maior que b a qualquer cluster é 1. As demais associações, tanto de vértices como de arestas, têm custo zero. Como G_I e G_M são completos, qualquer mapeamento de V_I para V_M atende às restrições de conexidade e cobertura de arestas.

Seja $\Phi(G_I, G_M)$, o custo de uma solução ótima do PCCA para os grafos G_I e G_M , utilizando as funções c e d . Mostraremos que a resposta do problema de clustering para a instância $\mathcal{I}$ é *sim* se e somente se $\Phi(G_I, G_M) = 0$.

($\Rightarrow$) Suponha que a resposta do problema de clustering é *sim* para a instância $\mathcal{I}$. Para cada $S_k \subset S$, existe um cluster $k \in V_M$ correspondente. Como o custo de todas as associações possíveis de vértices e arestas são não negativos, segue que $\Phi(G_I, G_M) \geq 0$.

Considere o mapeamento $\mathcal{M}$ de V_I para V_M , tal que $i \in V_I$ está associado a $k \in V_M$ se e somente se $i \in S_k$. Sejam $A_1 = \{ij \mid ij \in A_I \text{ e } ij \text{ está associada a algum cluster } k \in V_M \text{ no mapeamento } \mathcal{M}\}$, e $A_2 = A_I \setminus A_1$. Da forma como definimos a função d , temos que a soma dos custos das associações das arestas de A_1 é zero, pois cada aresta $ij \in A_1$ corresponde a um par $i, j \in S_k$, tal que $p(i, j) \leq b$, para algum $k \in V_M$. A soma dos custos das associações das arestas de A_2 também é zero, pois nenhuma das arestas de A_2 está associada a algum cluster. Como $c(i, k) = 0$ para todo $i \in V_I$ e $k \in V_M$, vale que o custo de $\mathcal{M}$ é zero. Logo, $\mathcal{M}$ é ótimo. Ou seja, $\Phi(G_I, G_M) = 0$.

($\Leftarrow$) Suponha agora que $\Phi(G_I, G_M) = 0$ e $\mathcal{M}$ é um mapeamento ótimo de V_I para V_M . Seja $S_k = \lambda(k)$, para todo $k \in V_M$, sendo $\lambda(k)$ o conjunto dos vértices associados a k em $\mathcal{M}$. Como cada vértice só pode ser associado a um único cluster, segue que os conjuntos $S_1, S_2, \dots, S_n$ são disjuntos. Pela definição da função d , não existe em $\mathcal{M}$ nenhuma aresta $ij \in A_I$ associada a algum cluster

$k \in V_M$, tal que $w_{ij} > b$, pois tal associação custaria 1, implicando em $\Phi(G_I, G_M) > 0$. Portanto, para $1 \leq i \leq n$, e todo par $u, v \in S_i$, vale que $p(u, v) \leq b$. Logo, a resposta do problema de clustering para a instância $\mathcal{I}$ é *sim*.

Como o problema de clustering é $\mathcal{NP}$ -completo, segue que o PCCA é $\mathcal{NP}$ -difícil. ■

Figura 3.1: À esquerda temos $G_I = (V_I, A_I)$. As arestas $ij \in A_I$ em negrito representam os pares de elementos $i, j \in S$ tais que $p(i, j) > b$. À direita temos uma solução viável do PCCA, tal que $\Phi(G_I, G_M) = 0$. Observe que G_M é um $\mathcal{K}_n$, sendo que $n = 3$, e existe um mapeamento ótimo que é constituído das seguintes associações: os vértices 2 e 3 estão associados a um mesmo cluster, digamos a ; os vértices 1, 5 e 6 estão associados ao cluster b ; o vértice 4 está associado ao cluster c . Claramente $S_a = \lambda(a)$, $S_b = \lambda(b)$ e $S_c = \lambda(c)$ são disjuntos. Para todo $k \in V_M$, temos que, não existe aresta ij , com $i, j \in \lambda(k)$, tal que $p(i, j) > b$. Na figura à direita, omitimos as arestas ij que estão associadas a pares de clusters diferentes e $p(i, j) \leq b$.

3.2 Formulação

Nesta seção apresentamos uma formulação para o PCCA baseada nos trabalhos [6, 16]. Utilizamos principalmente as idéias contidas no trabalho de Duarte [16] para tratar a restrição de conexidade.

Dados e Variáveis

Os dados e variáveis são os mesmos contidos nas formulações anteriores, incluindo as variáveis para tratar a restrição de conexidade através de fluxo em redes.

Vimos que na formulação não linear proposta por Boeres [6], o custo das associações das arestas

de A_I para os pares de clusters é calculado na função objetivo através do termo não linear:

$$\sum_{ij \in A_I} \sum_{k, l \in V_M} d_{kl}^{ij} \cdot x_{ik} \cdot x_{jl}.$$

Nossa proposta para a linearização deste termo é introduzir variáveis adicionais w_{ij} , para cada aresta $ij \in A_I$, de forma que a igualdade $w_{ij} = \sum_{k, l \in V_M} d_{kl}^{ij} \cdot x_{ik} \cdot x_{jl}$ seja verdadeira em qualquer solução ótima. Definimos, então, uma variável $w_{ij} \in \mathbb{R}_+$, para todo $ij \in A_I$. Posteriormente, introduziremos restrições adicionais para calcular adequadamente os valores das variáveis w_{ij} . Como cada aresta $ij \in A_I$ será associada a um único par de clusters, podemos interpretar que w_{ij} é o custo da associação da aresta $ij \in A_I$.

Formulação

Temos a seguinte formulação para o PCCA:

$$\min z = \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik} + \sum_{ij \in A_I} w_{ij}$$

sujeito a

- (i) $\sum_{k \in V_M} x_{ik} = 1, \forall i \in V_I.$
- (ii) $d_{kl}^{ij} \cdot (x_{ik} + x_{jl} - 1) \leq w_{ij}, \forall ij \in A_I \text{ e } k, l \in V_M, \text{ tal que } kl \in A_M \text{ ou } k = l.$
- (iii) $x_{ik} + x_{jl} \leq 1, \forall ij \in A_I \text{ e } k, l \in V_M, \text{ tal que } k \neq l \text{ e } kl \notin A_M.$
- (iv) $y_{ik} \leq x_{ik}, \forall i \in V_I \text{ e } k \in V_M.$
- (v) $\sum_{i \in V_I} y_{ik} \leq 1, \forall k \in V_M.$
- (vi) $g_{ik} + \sum_{j \in \delta_{G_I}(i)} (f_{ji}^k - f_{ij}^k) = x_{ik}, \forall k \in V_M \text{ e } i \in V_I.$
- (vii) $\sum_{j \in \delta_{G_I}(i)} f_{ij}^k \leq |V_I| \cdot x_{ik}, \forall k \in V_M \text{ e } i \in V_I.$
- (viii) $\sum_{i \in V_I} \sum_{k \in V_M} g_{ik} = |V_I|.$
- (ix) $g_{ik} \leq |V_I| \cdot y_{ik}, \forall i \in V_I \text{ e } k \in V_M.$
- (x) $g_{ik} \geq 0, \forall i \in V_I \text{ e } k \in V_M.$
- (xi) $f_{ij}^k \geq 0, \forall k \in V_M \text{ e } ij \in A_I.$
- (xii) $x_{ik} \in \{0, 1\}, \forall i \in V_I \text{ e } k \in V_M.$
- (xiii) $y_{ik} \in \{0, 1\}, \forall i \in V_I \text{ e } k \in V_M.$

O objetivo é minimizar o custo do mapeamento de V_I para V_M . Na restrição (i), exigimos que todo vértice de V_I seja associado a exatamente um cluster de V_M .

Na restrição (ii), exigimos que cada variável w_{ij} contenha o custo da associação da aresta $ij \in A_I$. Considere a soma abaixo:

$$\sum_{k,l \in V_M} d_{kl}^{ij} \cdot x_{ik} \cdot x_{jl}, \text{ para alguma aresta } ij \in A_I. \quad (2.1)$$

Como uma aresta $ij \in A_I$ é associada a um único par de clusters, digamos $\{k', l'\}$, então $x_{ik} \cdot x_{jl} = 0$ sempre que $k \neq k'$ ou $l \neq l'$. Desta forma, os únicos termos não nulos de (2.1) correspondem a d_{kl}^{ij} e $-d_{kl}^{ij}$. Note que $w_{ij} = \max\{d_{kl}^{ij} \cdot (x_{ik} + x_{jl} - 1) \mid k, l \in V_M\}$. Portanto, $w_{ij} = \sum_{k,l \in V_M} d_{kl}^{ij} \cdot x_{ik} \cdot x_{jl}$. Como cada vértice de V_I é associado a exatamente um cluster, então cada aresta $ij \in A_I$ está associada a exatamente um par de clusters. Para cada restrição do tipo (ii) adicionada à formulação, temos uma das seguintes situações:

x_{ik}	x_{jl}	Inequação
0	0	$-d_{kl}^{ij} \leq w_{ij}$
0	1	$0 \leq w_{ij}$
1	0	$0 \leq w_{ij}$
1	1	$d_{kl}^{ij} \leq w_{ij}$

Temos na função objetivo o termo $\sum_{ij \in A_I} w_{ij}$. Como o PCCA é um problema de minimização, o valor de w_{ij} é o menor possível em qualquer solução ótima, para todo $ij \in A_I$. Logo $w_{ij} = d_{kl}^{ij}$, para algum par $k, l \in V_M$ tal que $x_{ik} = x_{jl} = 1$. Só definimos a restrição (ii) para todo $ij \in A_I$ e $k, l \in V_M$, tal que $kl \in A_M$ ou $k = l$, porque em qualquer solução viável, cada aresta $ij \in A_I$ está associada ou a uma aresta de A_M ou a um cluster (cobertura de arestas).

Na restrição (iii), exigimos que nenhuma aresta $ij \in A_I$ seja associada a um par de clusters k, l , tal que $kl \notin A_M$, a não ser quando $k = l$ (cobertura de arestas). Toda aresta $ij \in A_I$ deve ser coberta ou por uma aresta de A_M ou por um cluster de V_M . As inequações (iv) a (xi) tratam a restrição de conexidade, conforme explicado na seção 2.4.2.

A seguir mostraremos como reduzir a quantidade de inequações nesta formulação que apresentamos para o PCCA.

3.2.1 Redução da Quantidade de Inequações na Formulação

Para que a variável w_{ij} contenha o custo da associação da aresta ij , para todo $ij \in A_I$, são adicionadas as inequações do tipo (ii) à formulação do PCCA exibida na seção anterior. No entanto, em alguns casos não é necessário adicionar todas estas restrições. Dada uma aresta $ij \in A_I$ e um par de clusters $k, l \in V_M$, são adicionadas à formulação as seguintes restrições:

$$d_{kl}^{ij} \cdot (x_{ik} + x_{jl} - 1) \leq w_{ij}; \quad (2.2)$$

$$d_{kl}^{ij} \cdot (x_{il} + x_{jk} - 1) \leq w_{ij}; \quad (2.3)$$

$$d_{kk}^{ij} \cdot (x_{ik} + x_{jk} - 1) \leq w_{ij}; \quad (2.4)$$

$$d_{ll}^{ij} \cdot (x_{il} + x_{jl} - 1) \leq w_{ij}. \quad (2.5)$$

Note que, para cada aresta $ij \in A_I$, cada restrição do tipo (2.4) e (2.5) é adicionada no máximo uma vez para os clusters k e l , respectivamente.

Dependendo do caso, podemos reduzir a quantidade de restrições a serem adicionadas para ij e k, l . Introduzimos a seguinte inequação:

$$d_{kl}^{ij} \cdot (x_{ik} + x_{il} + x_{jk} + x_{jl} - 1) \leq w_{ij}. \quad (2.6)$$

Observe que, das quatro variáveis contidas na inequação (2.6), no máximo duas assumem o valor 1 ao mesmo tempo. Outro detalhe a ser observado é que w_{ij} assumirá o maior valor que for imposto por alguma restrição, pois o termo $\sum_{ij \in A_I} w_{ij}$, contido na função objetivo, será minimizado.

Assumiremos que a função d foi definida de forma que os custos das associações de uma aresta $ij \in A_I$ para todos os clusters são iguais. Logo, assumiremos que $d_{kk}^{ij} = d_{ll}^{ij}$. Na seção 1.3, fornecemos um exemplo de definição de uma função d para a qual esta propriedade é válida.

Quando $d_{kl}^{ij} = d_{kk}^{ij} = d_{ll}^{ij}$, podemos substituir as restrições (2.2), (2.3), (2.4) e (2.5) pela restrição (2.6). Quando $d_{kl}^{ij} < d_{kk}^{ij} = d_{ll}^{ij}$, podemos substituir as restrições (2.2) e (2.3) pela restrição (2.6). Somente quando $d_{kl}^{ij} > d_{kk}^{ij} = d_{ll}^{ij}$ é que precisamos das quatro restrições (2.2), (2.3), (2.4) e (2.5).

Estas substituições não geram uma diminuição substancial no tamanho da formulação do PCCA e tampouco fornecem um limitante melhor para a relaxação linear. Porém, do ponto de vista de

implementação, é desejável que os recursos de máquina sejam utilizados de forma enxuta.

Uma alternativa para calcular o custo das associações entre arestas e pares de clusters, sem definir as variáveis w_{ij} , é definir variáveis de decisão para as associações entre arestas e pares de clusters. Para que tal abordagem valha a pena, é necessário que os grafos G_I e G_M sejam bem esparsos. O ideal para utilização desta abordagem é que G_I e G_M sejam árvores. Por enquanto, não faremos esta suposição.

A seguir, mostramos outro ponto onde a formulação do PCCA pode ser melhorada. Considere a inequação de cobertura de arestas (restrição (iii) da formulação que apresentamos para o PCCA):

$$x_{ik} + x_{jl} \leq 1, \forall ij \in A_I \text{ e } k, l \in V_M, \text{ tal que } k \neq l \text{ e } kl \notin A_M. \quad (2.7)$$

Introduzimos a seguinte inequação:

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} \geq |\delta_{G_I}(i)| \cdot x_{ik}, \text{ para todo } i \in V_I \text{ e } k \in V_M. \quad (2.8)$$

Mostraremos que as inequações (2.7) podem ser substituídas por (2.8) na formulação do PCCA. Antes de provar isto formalmente, apresentamos nas figuras 3.2 e 3.3 uma idéia geral do que está por trás das restrições (2.8).

Figura 3.2: À esquerda um vértice $i \in V_I$ e seus vizinhos. À direita um cluster $k \in V_M$ e seus vizinhos. A restrição (2.8) exige que, se i está associado a k , então todos os vértices vizinhos de i devem ser associados ou ao cluster k ou a algum cluster vizinho de k .

Lema 3.2.1 *Seja $x_{ik} \in \{0,1\}$, tal que $x_{ik} = 1$ se o vértice $i \in V_I$ está associado ao cluster $k \in V_M$ e $x_{ik} = 0$ caso contrário. Sejam $P_1 = \{x \in \{0,1\}^t \mid x \text{ atende à restrição (2.7)}\}$ e*

Figura 3.3: À esquerda um vértice $i \in V_I$ e um vizinho seu $j \in V_I$. À direita dois clusters $k, l \in V_M$, tal que $kl \notin A_M$. A restrição (2.8) exige que, se i está associado a k , então todos os vértices vizinhos de i devem ser associados ou ao cluster k ou a algum cluster vizinho de k , o que implica que j não pode ser associado a l . Caso j fosse associado a l , teríamos que $ij \in A_I$ estaria descoberta, pois $kl \notin A_M$.

$P_2 = \{x \in \{0, 1\}^t \mid x \text{ atende à restrição (2.8)}\}$ dois poliedros inteiros, sendo que $t = |V_I| \cdot |V_M|$. Um ponto x pertence a P_1 se e somente x pertence a P_2 (ou seja, $P_1 = P_2$).

Prova. ($\Rightarrow$) Suponha que existe um ponto x , tal que $x \in P_1$ e $x \notin P_2$. Então, existe alguma restrição do tipo (2.8) violada por x . Sejam $i \in V_I$ e $k \in V_M$, tais que

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} < x_{ik} \cdot |\delta_{G_I}(i)|.$$

Por definição, $x_{jl} \geq 0$, para todo $j \in V_I$ e $l \in V_M$. Portanto, $x_{ik} > 0$. Como P_1 é um poliedro inteiro, temos que $x_{ik} = 1$ (i está associado a k). Dessa forma, sabemos que

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} < |\delta_{G_I}(i)|.$$

Logo, existe algum vizinho de i , digamos $j \in \delta_{G_I}(i)$, que está associado a algum cluster l , tal que $l \neq k$ e l não é vizinho de k . Ou seja, $x_{jl} = 1$, para algum $l \in V_M$, tal que $l \notin \delta_{G_M}(k) \cup k$. Mas se $l \notin \delta_{G_M}(k) \cup k$, então $kl \notin A_M$ e conseqüentemente a restrição $x_{ik} + x_{jl} \leq 1$ é violada por x , o que é uma contradição com o fato de que $x \in P_1$. Portanto, se $x \in P_1$, então $x \in P_2$.

($\Leftarrow$) Suponha agora que existe um ponto x , tal que $x \in P_2$ e $x \notin P_1$. Então, existe alguma restrição do tipo (2.7) violada por x . Sejam $ij \in A_I$ e $k, l \in V_M$, tais que $kl \notin A_M$ e $x_{ik} + x_{jl} > 1$. Como P_2 é um poliedro inteiro, temos que $x_{ik} = x_{jl} = 1$. Pela definição da restrição (2.7), sabemos

que $k \neq l$ e que $kl \notin A_M$. Ou seja, $l \notin \delta_{G_M}(k) \cup k$. Logo,

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} \leq x_{ik} \cdot (|\delta_{G_I}(i)| - 1),$$

o que é uma contradição com o fato de que $x \in P_2$. Portanto, se $x \in P_2$, então $x \in P_1$. ■

A quantidade de inequações do tipo (2.7) que são adicionadas à formulação é precisamente $2 \cdot |A_I| \cdot \left(\frac{|V_M|^2 - |V_M|}{2} - |A_M| \right)$. Pelo lema 3.2.1, podemos substituir as restrições (2.7) pelas restrições (2.8), na formulação do PCCA. A quantidade de inequações do tipo (2.8) a serem adicionadas na formulação no lugar das de tipo (2.7) é precisamente $|V_I| \cdot |V_M|$. Para saber se vale ou não a pena substituir (2.7) por (2.8), é preciso levar em conta a expectativa que se tem com relação à quantidade de vértices e arestas dos grafos de entrada.

Para fornecer um exemplo concreto de comparação entre as restrições (2.7) e (2.8), vamos supor que G_I e G_M são grafos conexos e planares, situação que ocorre na aplicação de reconhecimento estrutural de objetos, mostrada na seção 1.3. Lembramos que, se G é um grafo conexo e planar com n vértices, m arestas e f faces, então: $n - m + f = 2$. Em consequência disso, conclui-se que G tem no máximo $3n - 6$ arestas.

Supondo que G_I e G_M são planares e que $|V_I| \leq c \cdot |V_M|$, sendo que c é uma constante¹, temos que:

$$|V_I| = O(|V_M|)$$

$$|A_I| \leq 3 \cdot |V_I| - 6 = O(|V_M|)$$

$$|A_M| \leq 3 \cdot |V_M| - 6 = O(|V_M|)$$

Logo, a quantidade de inequações do tipo (2.7) necessárias é

$$2 \cdot |A_I| \cdot \left(\frac{|V_M|^2 - |V_M|}{2} - |A_M| \right) =$$

$$O(|V_M|) \cdot (O(|V_M|^2) - O(|V_M|)) = O(|V_M|^3) - O(|V_M|^2) = O(|V_M|^3).$$

Já a quantidade de inequações do tipo (2.8) necessárias é: $|V_I| \cdot |V_M| = \Theta(|V_M|^2)$. Temos um exemplo numérico, utilizando uma instância real do problema de reconhecimento estrutural de objetos, na

¹Nas instâncias do problema de reconhecimento estrutural de objetos que possuímos $c < 7$.

qual $|V_I| = 276$, $|A_I| = 1352$, $|V_M| = 43$ e $|A_M| = 228$. Para esta instância seriam necessárias 317.856 inequações do tipo (2.7), contra 11.868 do tipo (2.8).

Vimos que, de forma geral, a quantidade de inequações do tipo (2.8) necessárias para tratar a restrição de conexidade é menor do que a quantidade de inequações do tipo (2.7) necessárias para desempenhar a mesma tarefa. No entanto, veremos a seguir que nem todas as inequações do tipo (2.8) são necessárias para tratar a restrição de conexidade. Ou seja, podemos reduzir ainda mais a quantidade de restrições na formulação do PCCA.

Para todo $i \in V_I$, definimos a seguinte inequação do tipo (2.8):

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} \geq |\delta_{G_I}(i)| \cdot x_{ik}, \text{ para todo } k \in V_M. \quad (2.9)$$

Considere um vértice $i \in V_I$, tal que, para todo $j \in \delta_{G_I}(i)$, a inequação (2.9) foi incluída na formulação do PCCA. A seguir, provaremos que não é necessário incluir a inequação (2.9) para o vértice i .

Antes de enunciar o próximo lema, definiremos o que é um *conjunto independente*. Um conjunto independente em um grafo $G = (V, A)$ é um conjunto $S \subseteq V$ de vértices tal que não existem dois vértices adjacentes em G contidos em S . Em outras palavras, se $u \in V$ e $v \in V$ são vértices quaisquer de um conjunto independente, então não existe aresta entre u e v em A .

Lema 3.2.2 *Dados dois grafos conexos $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$, e um conjunto independente $S \subset V_I$, definimos os seguintes poliedros:*

$$P_1 = \{x \in \{0, 1\}^t \mid \text{para todo } i \in V_I, \text{ as restrições (2.9) são satisfeitas}\} \text{ e}$$

$$P_2 = \{x \in \{0, 1\}^t \mid \text{para todo } i \in V_I \setminus S, \text{ as restrições (2.9) são satisfeitas}\},$$

sendo que $t = |V_I| \cdot |V_M|$. Um ponto inteiro x pertence a P_1 se e somente se x pertence a P_2 (ou seja, $P_1 = P_2$).

Prova. ($\implies$) Seja x um ponto inteiro, tal que $x \in P_1$. Como todas as restrições definidas em P_2 também foram definidas em P_1 , segue que $x \in P_2$.

($\Leftarrow$) Seja x um ponto inteiro, tal que $x \in P_2$. Suponha que $x \notin P_1$. Seja $i \in V_I$, tal que

$$\sum_{j \in \delta_{G_I}(i)} \sum_{l \in \delta_{G_M}(k) \cup k} x_{jl} < |\delta_{G_I}(i)| \cdot x_{ik}, \text{ para algum } k \in V_M.$$

Sabemos que as restrições (2.9) são satisfeitas para todos os vértices de $V_I \setminus S$. Logo, $i \in S$. Seja $j \in \delta_{G_I}(i)$, tal que j está associado a um cluster $l \notin \delta_{G_M}(k) \cup k$ (tal vértice existe, senão a restrição acima não estaria violada). Como S é um conjunto independente, segue que todo vértice de $\delta_{G_I}(i)$ está contido em $V_I \setminus S$. Logo, a restrição (2.9) foi definida em P_2 para o vértice j . Como $i \in \delta_{G_I}(j)$ e a restrição (2.9) foi definida para j , então i está associado a algum cluster contido em $\delta_{G_M}(l) \cup l$. Mas i está associado a k , o que é uma contradição, pois $k \notin \delta_{G_M}(l) \cup l$. Note que $l \notin \delta_{G_M}(k) \cup k$. Isto implica em duas coisas: (1) $k \neq l$; (2) se l não é vizinho de k , então obviamente k não é vizinho de l . Portanto, se $x \in P_2$, então $x \in P_1$. ■

Pelo lema 3.2.2, se S é um conjunto independente em G_I , então não é necessário incluir a restrição (2.9) para nenhum vértice de S . Para reduzir ao máximo a quantidade de inequações necessárias para tratar a restrição de conexidade, seria necessário encontrar um conjunto independente em G_I de cardinalidade máxima. Infelizmente, o problema de encontrar um conjunto independente de cardinalidade máxima é $\mathcal{NP}$ -difícil [23]. Note que o lema 3.2.2 não exige que o conjunto independente S tenha cardinalidade máxima. Sendo assim, seria razoável utilizar algum método eficiente (heurísticas ou algoritmos de aproximação) para encontrar um conjunto independente de maior cardinalidade possível. Em alguns casos, podemos facilmente identificar um bom limitante. Por exemplo, para um grafo bipartido $G = (V_1, V_2, A)$, é fácil ver que um limitante inferior para o tamanho de um conjunto independente de cardinalidade máxima é $\max\{|V_1|, |V_2|\}$, pois tanto V_1 como V_2 são conjuntos independentes.

Capítulo 4

PCCA *para árvores*

Um caso particular do PCCA é quando os grafos de entrada são árvores. Chamamos esta variação do problema de PCCA *para árvores*. Este capítulo é dedicado ao estudo do PCCA *para árvores*. Provamos que o problema é $\mathcal{NP}$ -difícil e apresentamos uma formulação para o mesmo.

4.1 Complexidade

Antes de tratar sobre a complexidade do PCCA *para árvores*, enunciaremos um lema que estabelece uma relação entre a restrição de conexidade e a quantidade de arestas de A_I associadas a uma aresta de A_M em uma solução viável do PCCA *para árvores*.

Dado um grafo $G = (V, A)$ não orientado, utilizaremos a notação $P_G(u, v)$, tal que $u, v \in V$, para designar um caminho qualquer de u a v em G . Quando implicitamente sabe-se qual é o grafo em questão, utilizaremos apenas $P(u, v)$. Esta notação será utilizada ao longo de todo o texto.

Lema 4.1.1 *Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ duas árvores. Seja $\mathcal{M}$ um mapeamento de V_I para V_M , tal que $\mathcal{M}$ é viável do PCCA *para árvores*. Para cada aresta $kl \in A_M$, existe no máximo uma aresta $ij \in A_I$ associada a kl em $\mathcal{M}$.*

Prova. Suponha que existe uma aresta $kl \in A_M$, para a qual mais de uma aresta de A_I estão associadas em $\mathcal{M}$. Sejam $i, i' \in \lambda(k)$ e $j, j' \in \lambda(l)$, tais que $ij \in A_I$, $i'j' \in A_I$ e $ij \neq i'j'$. Note que as arestas ij e $i'j'$ têm no máximo um extremo em comum. Temos, então, dois casos: (i) ij e $i'j'$ não têm nenhum extremo em comum; (ii) ij e $i'j'$ têm um extremo em comum.

Como $\mathcal{M}$ é viável, então $G_I[\lambda(k)]$ e $G_I[\lambda(l)]$ são conexos. Portanto, no caso (i) existe um circuito

$P(i, i') \cup \{i'j'\} \cup P(j', j) \cup \{ji\}$ em G_I , contradizendo o fato de que G_I é uma árvore. No caso (ii), suponha, sem perda de generalidade, que $i = i'$. Então, existe um circuito $\{i'j'\} \cup P(j', j) \cup \{ji\}$ em G_I , contradizendo o fato de que G_I é uma árvore.

Figura 4.1: À esquerda, um circuito $P(i, i') \cup \{i'j'\} \cup P(j', j) \cup \{ji\}$ em G_I . À direita, um circuito $\{i'j'\} \cup P(j', j) \cup \{ji\}$ em G_I (supondo que $i = i'$).

■

A propriedade mostrada no lema 4.1.1 será utilizada diversas vezes nos próximos capítulos. Algumas vezes faremos uso desta propriedade sem mencionar diretamente o lema 4.1.1.

A seguir, provaremos que o PCCA para árvores é $\mathcal{NP}$ -difícil. Faremos uma redução de um problema $\mathcal{NP}$ -completo chamado *emparelhamento tridimensional* [23].

Emparelhamento Tridimensional: Dado um conjunto $E \subseteq W \times X \times Y$, sendo que W , X e Y são disjuntos, e $|W| = |X| = |Y| = t$, determinar se E contém um emparelhamento perfeito (um conjunto $E' \subseteq E$, tal que $|E'| = t$ e para qualquer par $(w_1, x_1, y_1), (w_2, x_2, y_2) \in E'$, temos que $w_1 \neq w_2$, $x_1 \neq x_2$ e $y_1 \neq y_2$).

Proposição 4.1.1 *O PCCA para árvores é $\mathcal{NP}$ -difícil.*

Prova. Dada uma instância $\mathcal{I}$ do problema de emparelhamento tridimensional, na qual são dados X, W, Y e $E \subseteq X \times W \times Y$, tal que $E = \{1, 2, \dots, n\}$ e $|X| = |W| = |Y| = t$, definimos os grafos G_I e G_M da seguinte maneira:

- Em V_I temos: um vértice de rótulo r ; para cada tripla $e_j = (x, w, y) \in E$, tal que $1 \leq j \leq |E|$, definimos um vértice de rótulo a_j e três vértices de rótulos a_x^j , a_w^j e a_y^j .
- Em A_I temos: para cada tripla $e_j = (x, w, y) \in E$, tal que $1 \leq j \leq |E|$, definimos uma aresta $\{r, a_j\}$ e três arestas $\{a_w^j, a_j\}$, $\{a_x^j, a_j\}$ e $\{a_y^j, a_j\}$.

- Em V_M temos: um cluster de rótulo q ; para cada coordenada $j \in X \cup W \cup Y$, definimos um cluster de rótulo j ; para cada tripla $e_j = (x, w, y) \in E$, tal que $1 \leq j \leq |E|$, definimos um cluster de rótulo d_j e três clusters de rótulos d_x^j , d_w^j e d_y^j .
- Em A_M temos: para cada tripla $e_j = (x, w, y) \in E$, tal que $1 \leq j \leq |E|$, definimos uma aresta $\{q, d_j\}$ e três arestas $\{d_w^j, d_j\}$, $\{d_x^j, d_j\}$ e $\{d_y^j, d_j\}$; para cada coordenada $j \in X \cup W \cup Y$, definimos uma aresta $\{j, q\}$.

Claramente G_I e G_M são árvores. As figuras 4.2 e 4.3 ilustram a construção de G_I e G_M para $X = \{1, 2, 3\}$, $W = \{4, 5, 6\}$, $Y = \{7, 8, 9\}$ e $E = \{(1, 4, 7), (1, 4, 8), (2, 5, 7), (3, 6, 9), (3, 6, 8)\}$.

Figura 4.2: O grafo G_I gerado a partir de uma instância do problema de emparelhamento tridimensional.

Figura 4.3: O grafo G_M gerado a partir de uma instância do problema de emparelhamento tridimensional.

Dividiremos o conjunto de todas as possíveis associações dos vértices de V_I para os clusters de V_M em três conjuntos: $Z \subseteq V_I \times V_M$, $U \subseteq V_I \times V_M$ e $P \subseteq V_I \times V_M$. O conjunto Z será composto por todas as associações para as quais atribuiremos custo zero. O conjunto U será composto por todas as associações para as quais atribuiremos custo um. O conjunto P será um conjunto de associações “proibidas”. Dizemos que uma associação a é proibida quando a está contida em uma solução ótima

se e somente se não existe nenhuma solução viável que não contenha a . Para proibir uma associação $a = (j, l)$, tal que $j \in V_I$ e $l \in V_M$, basta atribuir um custo $c(j, l) = \alpha$, para uma constante α suficientemente grande, de forma que tal associação só seja escolhida em uma solução ótima se não existir nenhuma outra solução viável que não contenha a associação a . Um exemplo seria definir α como sendo a soma dos custos de todas as associações não proibidas. Por simplicidade definiremos o custo das associações proibidas como sendo infinito.

Seja $Z \subseteq V_I \times V_M$, tal que $Z = \{(r, q)\} \cup \{(a_j, q) \mid 1 \leq j \leq |E|\} \cup \{(a_x^j, x), (a_w^j, w), (a_y^j, y)\} \mid 1 \leq j \leq |E|$ e $e_j = (x, w, y) \in E\} \cup \{(a_x^j, d_x^j), (a_w^j, d_w^j), (a_y^j, d_y^j)\} \mid 1 \leq j \leq |E|$ e $e_j = (x, w, y) \in E\}$. Seja $U \subseteq V_I \times V_M$, tal que $U = \{(a_j, d_j) \mid 1 \leq j \leq |E|\}$. Seja $P \subseteq V_I \times V_M$, tal que $P = S \setminus Z \setminus U$, sendo que S é o conjunto de todas as associações possíveis entre os vértices de V_I e os clusters de V_M .

A função de custo das associações dos vértices de V_I para os clusters de V_M é definida da seguinte maneira (para todo $j \in V_I$ e $l \in V_M$):

$$c(j, l) = \begin{cases} 0, & \text{se } (j, l) \in Z \\ 1, & \text{se } (j, l) \in U \\ \infty, & \text{se } (j, l) \in P. \end{cases}$$

A função de custo das associações das arestas de A_I para os pares de clusters é definida da seguinte maneira:

$$d(ij, k, l) = 0, \text{ para todo } ij \in A_I \text{ e } k, l \in V_M.$$

As figuras 4.5 e 4.4 ilustram os custos das associações de $G'_I = (V'_I, A'_I)$ para $G'_M = (V'_M, A'_M)$, tal que $G'_I = G_I[\{r, a_1, a_1^1, a_4^1, a_7^1\}]$ e $G'_M = G_M[\{q, d_1, d_1^1, d_4^1, d_7^1, 1, 4, 7\}]$, sendo que G_I e G_M são os grafos definidos nas figuras 4.2 e 4.3, respectivamente.

Seja $\Phi(G_I, G_M)$, o custo de uma solução ótima do PCCA *para árvores*, utilizando as funções c e d , para os grafos G_I e G_M , gerados a partir da instância $\mathcal{I}$ do problema de emparelhamento tridimensional. Mostraremos que a resposta do problema de emparelhamento tridimensional, para a instância $\mathcal{I}$, é *sim* se e somente se $\Phi(G_I, G_M) = |E| - t$, sendo que t corresponde à cardinalidade de um emparelhamento perfeito em E , se houver.

($\implies$) Suponha que existe um mapeamento ótimo $\mathcal{M}$, de V_I para V_M , de custo $\Phi(G_I, G_M) =$

Figura 4.4: Considere os grafos G'_I e G'_M . À esquerda temos as associações de custo zero. À direita temos as associações de custo um.

Figura 4.5: Considere os grafos G'_I e G'_M . Nas quatro figuras temos as associações proibidas. Observe que as associações de qualquer vértice de V'_I para qualquer cluster de $V_M \setminus V'_M$ também são proibidas.

$|E| - t$, para os grafos G_I e G_M , gerados a partir da instância $\mathcal{I}$ do problema de emparelhamento tridimensional, conforme especificado anteriormente.

Seja $H = \{a_j \mid 1 \leq j \leq |E| \text{ e } a_j \in V_I\}$. Pelo valor de $\Phi(G_I, G_M)$, sabemos que não existe nenhuma associação proibida em $\mathcal{M}$. Logo, o vértice de rótulo r está associado ao cluster de rótulo q e cada vértice a_j de H está associado ou a q ou a d_j . Sabemos que $\Phi(G_I, G_M) = |E| - t$ e que $c(a_j, q) = 0$ e $c(a_j, d_j) = 1$, para todo $a_j \in H$. Como todas as associações de vértices de $V_I \setminus H$ para

qualquer cluster custam ou zero ou infinito, temos que exatamente t vértices de H estão associados a q .

Sejam $H' = \{a_j \mid 1 \leq j \leq |E|, a_j \in H \text{ e } a_j \text{ está associado a } q \text{ em } \mathcal{M}\}$, $F' = \{\{a_x^j, a_w^j, a_y^j\} \mid 1 \leq j \leq |E|, e_j = (x, w, y) \in E \text{ e } a_j \in H'\}$ e $L = \{l \mid l \in X \cup W \cup Y \text{ e } l \in V_M\}$. Observe que $|L| = |X| + |W| + |Y| = 3t$ e que, pelo valor de $\Phi(G_I, G_M)$ e pela restrição de cobertura de arestas, os vértices de F' estão todos associados a algum cluster de L . Como o vértice r está associado ao cluster q , se existisse um cluster $l \in L$ para o qual dois ou mais vértices de F' estivessem associados a l , teríamos mais de uma aresta de A_I associada à aresta $kl \in A_M$, o que seria uma contradição com o lema 4.1.1. Portanto, para cada cluster $l \in L$, existe exatamente um vértice de F' associado a l . Como cada cluster de L corresponde a uma coordenada distinta de $X \cup W \cup Y$ e, para cada vértice $a_j \in H'$, temos os três vértices a_x^j , a_w^j e a_y^j em F' associados aos clusters x , w e y de L , respectivamente, que correspondem a uma tripla $(x, w, y) \in E$, temos que a resposta do problema de emparelhamento tridimensional para a instância $\mathcal{I}$ é *sim*.

($\Leftarrow$) Suponha que a resposta do problema de emparelhamento tridimensional para a instância $\mathcal{I}$ é *sim*. Sejam G_I e G_M dois grafos construídos a partir de $\mathcal{I}$, conforme especificado anteriormente.

Seja $E' \subseteq E$ um emparelhamento perfeito. Construiremos um mapeamento ótimo $\mathcal{M}$, de V_I para V_M , a partir de E' . Associamos o vértice de rótulo r ao cluster de rótulo q , no mapeamento $\mathcal{M}$. Caso associássemos r a algum outro cluster, teríamos $\Phi(G_I, G_M) = \infty$.

Primeiramente, mostraremos que $\Phi(G_I, G_M) \geq |E| - t$. Sejam $H = \{a_j \mid 1 \leq j \leq |E| \text{ e } a_j \in V_I\}$, $L = \{l \mid l \in X \cup W \cup Y \text{ e } l \in V_M\}$ e $F = \{\{a_x^j, a_w^j, a_y^j\} \mid 1 \leq j \leq |E|, e_j = (x, w, y) \in E \text{ e } a_j \in H\}$. Note que $\Phi(G_I, G_M) \neq \infty$ (basta considerar a solução em que r está associado a q , a_j está associado a d_j , para todo $a_j \in V_I$, e a_x^j está associado a d_x^j , para todo $a_x^j \in V_I$). Tal solução tem custo $|E|$. Ou seja, não existe nenhuma associação proibida em qualquer solução ótima. Logo, cada vértice a_j de H está associado ou a q ou a d_j . Observe que no máximo t vértices de H podem ser associados a q em uma solução ótima qualquer. Se tivéssemos mais do que t vértices associados a q , teríamos mais de $3t$ vértices de F obrigatoriamente associados a algum cluster de L . Do contrário, teríamos ou associações proibidas ou a restrição de cobertura de arestas violada. Como $|L| = 3t$, teríamos algum cluster $l \in L$ para o qual dois ou mais vértices de F estivessem associados a l . Logo, teríamos mais de uma aresta de A_I associada à aresta $ql \in A_M$, o que seria uma contradição com o lema 4.1.1. Portanto, pelo menos $|E| - t$ vértices de H não estão associados a q . Como $c(a_j, d_j) = 1$, para $1 \leq j \leq |E|$, segue que $\Phi(G_I, G_M) \geq |E| - t$.

Figura 4.6: Acima G_I e abaixo G_M . $H = \{a_1, a_2, \dots, a_5\}$, $H' = \{a_3, a_4, a_5\}$, F' corresponde aos vértices de V_I que estão dentro do retângulo sublinhado e L corresponde aos clusters de V_M que estão dentro do retângulo sublinhado.

Sem perda de generalidade, assuma que $E' = (1, 2, \dots, m)$ e que $E = (1, 2, \dots, m, m+1, \dots, n)$. Para $1 \leq j \leq |E'|$, temos uma tripla $e_j = (x, w, y) \in E'$. Associamos, no mapeamento $\mathcal{M}$, o vértice a_j ao cluster q e os vértices a_x^j , a_w^j e a_y^j aos clusters x , w e y , respectivamente, para $1 \leq j \leq |E'|$. Como não existem coordenadas iguais em E' , temos que, para cada cluster $l \in L$, existe exatamente um vértice de F associado a l em $\mathcal{M}$. Claramente, a soma dos custos das associações contidas em $\mathcal{M}$, até o momento, é zero. Tais associações respeitam a restrição de conexidade, pois o cluster q é o único para o qual existem dois ou mais vértices associados e, para cada vértice $a_j \in V_I$ associado a q em $\mathcal{M}$, temos uma aresta $\{a_j, r\} \in A_I$ (lembrando que r está associado a q em $\mathcal{M}$). A restrição de cobertura de arestas também está respeitada por $\mathcal{M}$, até o momento, pois, para $1 \leq j \leq |E'|$, temos as arestas $\{a_j, a_x^j\}$, $\{a_j, a_w^j\}$ e $\{a_j, a_y^j\}$ em A_I que estão cobertas pelas arestas qx , qw e qy em A_M , respectivamente. Para cada a_j associado a q em $\mathcal{M}$, temos uma aresta $\{a_j, r\} \in V_I$ coberta pelo cluster q .

Seja H'' o conjunto dos vértices de H que não estão associados a q em $\mathcal{M}$. Como já associamos t vértices de H ao cluster q , não podemos associar nenhum vértice de H'' ao cluster q . Nos resta apenas associar cada vértice $a_j \in H''$ ao respectivo $d_j \in V_M$. Para cada $a_j \in H''$, temos uma tripla $e_j = (x, w, y) \in E$. Associamos, no mapeamento $\mathcal{M}$, o vértice a_j ao cluster d_j e os vértices a_x^j , a_w^j e a_y^j aos clusters d_x^j , d_w^j e d_y^j , respectivamente, para cada $a_j \in H''$. Claramente $\mathcal{M}$ é viável e o custo de $\mathcal{M}$ é $|H''| = |E| - t$. Portanto, $\Phi(G_I, G_M) = |E| - t$.

Como o problema de emparelhamento tridimensional é $\mathcal{NP}$ -completo, segue que o PCCA *para árvores* é $\mathcal{NP}$ -difícil. A figura 4.7 mostra uma solução ótima do PCCA para os grafos G_I e G_M definidos nas figuras 4.2 e 4.3, respectivamente.

Figura 4.7: Uma solução ótima do PCCA para os grafos definidos nas figuras 4.2 e 4.3.

■

4.2 Formulação

Nesta seção, exploraremos algumas características específicas da estrutura das árvores para desenvolver uma formulação para o PCCA *para árvores*.

Na formulação que apresentamos para o PCCA, definimos variáveis de decisão relacionadas somente às associações de vértices a clusters. As associações de arestas a pares de clusters eram induzidas pelas associações de vértices a clusters. Na formulação que apresentaremos a seguir, definiremos variáveis de decisão relacionadas somente às associações de arestas a pares de clusters. Nesta outra representação, as associações de vértices a clusters serão induzidas pelas associações de arestas a pares de clusters. Se uma aresta $ij \in A_I$ está associada a um par de clusters $k, l \in V_M$, então dizemos que i está associado a k e j está associado a l . Mais adiante, mostraremos em mais detalhes como isso será feito. Note que há uma distinção entre associar ij a k, l e associar ij a l, k . No primeiro caso, i está associado a k e j está associado a l . No segundo caso, i está associado a l e j está associado a k .

Em vez de definir as funções de custo c e d , conforme fizemos nas formulações anteriores, definiremos uma única função de custo h . Suponha que temos duas funções $c : V_I \times V_M \rightarrow \mathbb{R}_+$ e $d : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$, tais que a função c representa o custo das associações de vértices a clusters e a função d representa o custo das associações de arestas a pares de clusters. Definimos a função $h : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$ da seguinte maneira:

$$h(ij, k, l) = \frac{c(i, k)}{g(i)} + \frac{c(j, l)}{g(j)} + d(ij, k, l), \quad \forall ij \in A_I \text{ e } k, l \in V_M,$$

sendo que $g(u)$ corresponde ao grau de um vértice u . Por simplicidade, utilizaremos h_{kl}^{ij} , c_{ik} e d_{kl}^{ij} no lugar de $h(ij, k, l)$, $c(i, k)$ e $d(ij, k, l)$, respectivamente.

Seja $x_{ik} \in \{0, 1\}$, para todo $i \in V_I$ e $k \in V_M$, tal que $x_{ik} = 1$ se o vértice $i \in V_I$ está associado ao cluster $k \in V_M$, e $x_{ik} = 0$, caso contrário. Seja $y_{kl}^{ij} = x_{ik} \cdot x_{jl}$, para todo $ij \in A_I$ e $k, l \in V_M$. Considere as seguintes funções:

$$\sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik} + \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} d_{kl}^{ij} \cdot y_{kl}^{ij}, \quad (2.1)$$

$$\sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} h_{kl}^{ij} \cdot y_{kl}^{ij}. \quad (2.2)$$

Note que a função (2.1) corresponde à função objetivo do PCCA.

Lema 4.2.1 *As funções (2.1) e (2.2) são equivalentes para qualquer vetor x viável do PCCA.*

Prova. Observe que (2.2) pode ser reescrita da seguinte forma:

$$\begin{aligned} & \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} \left(\frac{c_{ik}}{g(i)} + \frac{c_{jl}}{g(j)} + d_{kl}^{ij} \right) \cdot y_{kl}^{ij} = \\ & \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} d_{kl}^{ij} \cdot y_{kl}^{ij} + \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} \left(\frac{c_{ik}}{g(i)} + \frac{c_{jl}}{g(j)} \right) \cdot y_{kl}^{ij} = \\ & \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} d_{kl}^{ij} \cdot y_{kl}^{ij} + \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} \left(\frac{c_{ik}}{g(i)} + \frac{c_{jl}}{g(j)} \right) \cdot (x_{ik} \cdot x_{jl}). \end{aligned}$$

Precisamos mostrar que

$$\sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} \left(\frac{c_{ik}}{g(i)} + \frac{c_{jl}}{g(j)} \right) \cdot (x_{ik} \cdot x_{jl}) = \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik},$$

para qualquer vetor x viável do PCCA.

Seja x um vetor viável do PCCA. Para cada aresta $ij \in A_I$, sejam $k_i \in V_M$ e $l_j \in V_M$, tais que $x_{ik_i} = x_{jl_j} = 1$.

Da forma como definimos as variáveis y_{kl}^{ij} , se $y_{kl}^{ij} = 1$, significa que a aresta ij está associada ao par de clusters k, l . Logo, vale que $\sum_{k \in V_M} \sum_{l \in V_M} y_{kl}^{ij} = 1$, para todo $ij \in A_I$. Ou seja, cada aresta de A_I está associada a exatamente um par de clusters em qualquer solução viável. Portanto, temos que:

$$\sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} \left(\frac{c_{ik}}{g(i)} + \frac{c_{jl}}{g(j)} \right) \cdot (x_{ik} \cdot x_{jl}) = \sum_{ij \in A_I} \left(\frac{c_{ik_i}}{g(i)} + \frac{c_{jl_j}}{g(j)} \right) \cdot (x_{ik_i} \cdot x_{jl_j}).$$

Como $x_{ik_i} = x_{jl_j} = 1$, vale que:

$$\sum_{ij \in A_I} \left(\frac{c_{ik_i}}{g(i)} + \frac{c_{jl_j}}{g(j)} \right) \cdot (x_{ik_i} \cdot x_{jl_j}) = \sum_{ij \in A_I} \left(\frac{c_{ik_i}}{g(i)} \cdot x_{ik_i} + \frac{c_{jl_j}}{g(j)} \cdot x_{jl_j} \right).$$

Cada vértice $i \in V_I$ aparece $g(i)$ vezes na soma acima, portanto:

$$\sum_{ij \in A_I} \left(\frac{c_{ik_i}}{g(i)} \cdot x_{ik_i} + \frac{c_{jl_j}}{g(j)} \cdot x_{jl_j} \right) = g(i) \cdot \sum_{i \in V_I} \frac{c_{ik_i}}{g(i)} \cdot x_{ik_i} = \sum_{i \in V_I} c_{ik_i} \cdot x_{ik_i}.$$

Como $\sum_{k \in V_M} x_{ik} = 1$, para todo $i \in V_I$, então $x_{ik} = 0$, para todo $i \in V_I$ e $k \in V_M$, tal que $k \neq k_i$. Logo,

$$\sum_{i \in V_I} c_{ik_i} \cdot x_{ik_i} = \sum_{i \in V_I} \sum_{k \in V_M} c_{ik} \cdot x_{ik}.$$

Concluimos que as funções (2.1) e (2.2) são equivalentes para qualquer vetor x viável do PCCA. ■

Repare que poderíamos ter substituído (2.1) por (2.2) nas formulações anteriores. No entanto, tal substituição não alteraria em nada o problema. Em outras palavras, não ganharíamos nada com tal substituição. A grande vantagem em usar a função (2.2) no lugar de (2.1) ocorre quando as variáveis de decisão são definidas apenas para as arestas (nos casos anteriores, definimos variáveis de decisão apenas para os vértices). Utilizaremos a função (2.2) na formulação que apresentaremos a seguir.

Dados e Variáveis

São dadas duas árvores $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$, e uma função $h : A_I \times V_M \times V_M \rightarrow \mathbb{R}_+$, para atribuir custos às associações de arestas a pares de clusters. Assumiremos que a função h foi definida como mostrado anteriormente.

Seja $y_{kl}^{ij} \in \{0, 1\}$, para todo $ij \in A_I$ e $k, l \in V_M$, tal que ou $k = l$ ou $kl \in A_M$, sendo que $y_{kl}^{ij} = 1$ se a aresta $ij \in A_I$ está associada ao par de clusters $k, l \in V_M$, e $y_{kl}^{ij} = 0$, caso contrário. Observe que não há necessidade de tratar explicitamente a restrição de *cobertura de arestas*, pois definimos somente as variáveis y_{kl}^{ij} para as quais ou $kl \in A_M$ ou $k = l$. Para uma aresta $ij \in A_I$ estar descoberta em uma solução, tal aresta teria que estar associada a um par de clusters k, l , tal que $kl \notin A_M$. Note que a variável que corresponderia a esta associação nem sequer foi definida. Por simplicidade, vamos supor que y_{kl}^{ij} é definido para todo $ij \in A_I$ e $k, l \in V_M$, e que o valor de y_{kl}^{ij} é fixado em zero quando $kl \notin A_M$ e $k \neq l$.

Note que G_M não é orientado. Logo, $kl \in A_M$ e $lk \in A_M$ são a mesma aresta. Mesmo assim, precisamos diferenciar x_{kl}^{ij} de x_{lk}^{ij} . Pois, o custo de $h(ij, k, l)$ pode ser diferente de $h(ij, l, k)$. Por essa razão, para evitar ambigüidades no texto, quando dissermos que ij está associado ao par k, l , será equivalente a dizer que $x_{kl}^{ij} = 1$. Quando quisermos dizer que $x_{lk}^{ij} = 1$, diremos que ij está associado ao par l, k .

Formulação

Para todo $ij \in A_I$ e $k, l \in V_M$, definimos os conjuntos S_i , S_j , S_k e S_l da seguinte maneira: $S_i = \delta_{G_I}(i) \setminus j$ e $S_j = \delta_{G_I}(j) \setminus i$; se $kl \notin A_M$, então $S_l = \delta_{G_M}(l) \cup l$ e $S_k = \delta_{G_M}(k) \cup k$, senão (ou seja, se $kl \in A_M$), então $S_k = (\delta_{G_M}(k) \cup k) \setminus l$ e $S_l = (\delta_{G_M}(l) \cup l) \setminus k$.

Temos a seguinte formulação para o PCCA *para árvores*:

$$\begin{aligned} \min z = & \sum_{ij \in A_I} \sum_{k \in V_M} \sum_{l \in V_M} y_{kl}^{ij} \cdot h_{kl}^{ij} \\ \text{sujeito a} & \\ \text{(i)} & \sum_{k \in V_M} \sum_{l \in V_M} y_{kl}^{ij} = 1, \text{ para todo } ij \in A_I. \\ \text{(ii)} & \sum_{i' \in S_i} \sum_{k' \in S_k} y_{k'i}^{j'k} \geq (|\delta_{G_I}(i)| - 1) \cdot y_{kl}^{ij}, \\ & \sum_{j' \in S_j} \sum_{l' \in S_l} y_{l'j}^{i'l} \geq (|\delta_{G_I}(j)| - 1) \cdot y_{kl}^{ij}, \text{ para todo } ij \in A_I \text{ e } k, l \in V_M. \\ \text{(iii)} & \sum_{ij \in A_I} (y_{kl}^{ij} + y_{lk}^{ij}) \leq 1, \text{ para todo } kl \in A_M. \\ \text{(iv)} & y_{kl}^{ij} \in \{0, 1\}, \text{ para todo } ij \in A_I \text{ e } k, l \in V_M. \end{aligned}$$

O objetivo é encontrar um mapeamento de A_I para $V_M \times V_M$ de custo mínimo.

Na restrição (i), exigimos que toda aresta de A_I seja associada a exatamente um par de clusters de V_M . Conforme dito anteriormente, as associações de vértices a clusters serão induzidas pelas associações de arestas a pares de clusters. Por simplicidade, comentaremos o funcionamento das restrições do tipo (ii) em termos de associações de vértices a clusters. No entanto, deve ficar subentendido que as associações de arestas a pares de clusters devem induzir associações de vértices a clusters que estejam de acordo com o que for especificado.

A restrição (ii) implica na exigência de que:

- no caso em que $k \neq l$, se a aresta $ij \in A_I$ estiver associada à aresta $kl \in A_M$, então cada vértice de S_i deve ser associado a algum cluster de S_k e cada vértice de S_j deve ser associado a algum cluster de S_l . Neste caso, não incluímos l em S_k , pois, pelo lema 4.1.1, só pode haver uma única aresta de A_I associada a kl . Pelo mesmo motivo, não incluímos k em S_l . Um exemplo é fornecido na figura 4.8;
- no caso em que $k = l$, se a aresta ij estiver associada ao cluster $k = l$, então cada vértice de S_i deve ser associado a algum cluster de S_k e cada vértice de S_j deve ser associado a algum

cluster de S_l . Neste caso, vale que $S_k = S_l$. Um exemplo é fornecido na figura 4.8.

Pelo lema 3.2.2, conclui-se que nem todas as inequações do tipo (ii) são necessárias na formulação do PCCA *para árvores*. Vimos que, para um grafo bipartido $G = (V_1, V_2, A)$, um limitante inferior para o tamanho de um conjunto independente de cardinalidade máxima é $\max\{|V_1|, |V_2|\}$. Como toda árvore é um grafo bipartido, então, pelo menos metade das inequações do tipo (ii) não precisam ser incluídas na formulação do PCCA *para árvores*.

Figura 4.8: A primeira figura ilustra o caso em que $k \neq l$ e $kl \in A_M$. A segunda figura ilustra o caso em que $k = l$.

Conforme dito anteriormente, pelo fato de definirmos as variáveis y_{kl}^{ij} somente para todo $ij \in A_I$ e $k, l \in V_M$, tais que $kl \in A_M$ ou $k = l$, não há necessidade de tratar explicitamente a restrição de *cobertura de arestas*. Mesmo assim, precisamos da restrição (ii). Suponha que existam duas arestas adjacentes ij e ji' em A_I , e duas arestas não adjacentes kl e $k'l'$ em A_M . Sem a restrição (ii), uma solução com $y_{kl}^{ij} = 1$ e $y_{k'l'}^{ji'} = 1$ seria viável. Neste caso, seria impossível determinar se o vértice j está associado ao cluster l ou ao cluster k , conforme figura 4.9. Diremos que uma solução em que tal situação não ocorre é uma solução que respeita as adjacências das arestas.

Suponha agora que, além das arestas mencionadas acima, existe uma aresta $kl' \in A_M$ adjacente a kl . Sem a restrição (ii), uma solução com $y_{kl}^{ij} = 1$ e $y_{k'l'}^{ji'} = 1$ seria viável. Embora as arestas kl e kl' sejam adjacentes, é fundamental manter a coerência entre as associações de vértices a clusters. Neste exemplo, seria impossível determinar se o vértice j está associado ao cluster l ou ao cluster k , conforme figura 4.10. Diremos que uma solução em que tal situação não ocorre é uma solução que

Figura 4.9: Uma solução que não respeita as adjacências das arestas.

mantém a coerência entre as associações de vértices a clusters.

Figura 4.10: Uma solução que não mantém a coerência entre as associações de vértices a clusters.

Observe que não incluímos na formulação do PCCA *para árvores* nenhuma restrição para tratar explicitamente a conexidade. No lema a seguir, mostramos que qualquer solução que atende às restrições (i) a (iv) tem $G_I[\lambda(k)]$ conexo, para todo $k \in V_M$.

Lema 4.2.2 *Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ duas árvores. Qualquer solução que atende às restrições (i) a (iv) tem $G_I[\lambda(k)]$ conexo, para todo $k \in V_M$.*

Prova. As restrições (i) a (iv) garantem que qualquer solução viável tem as seguintes propriedades:

- (1) para cada aresta $kl \in A_M$, existe no máximo uma aresta de A_I associada a kl .
- (2) todas as arestas de A_I estão cobertas. Ou seja, para todo $ij \in A_I$ e $k, l \in V_M$, se $y_{kl}^{ij} = 1$, então $k = l$ ou $kl \in A_M$.
- (3) tal solução respeita as adjacências das arestas e mantém a coerência entre as associações de vértices a clusters.

Suponha que existe uma solução que atende às restrições (i) a (iv) com $G_I[\lambda(k)]$ desconexo para algum $k \in V_M$. Sejam S e T duas componentes distintas de $G_I[\lambda(k)]$. Como G_I é uma árvore, então

existe um único caminho $P(i, j)$ em G_I para qualquer par i, j , tal que $i \in S$ e $j \in T$. Como $G_I[\lambda(k)]$ é desconexo e considerando as propriedades (2) e (3), então existem pelo menos duas arestas em $P(i, j)$ com um extremo associado a k e o outro extremo associado a outro cluster diferente de k .

Sejam uv e $u'v'$ duas arestas de $P(i, j)$, tais que os vértices u e u' estão associados ao cluster k e os vértices v e v' estão associados a outro(s) cluster(s) diferente(s) de k . Pela propriedade (1), os vértices v e v' estão associados a clusters distintos, do contrário teríamos duas arestas de A_I associadas a uma mesma aresta de A_M . Sejam $l \in V_M$ e $m \in V_M$, tais que v está associado a l e v' está associado a m .

Sem perda de generalidade, assuma que $P(i, j) = P(i, u) \cup \{uv\} \cup P(v, v') \cup \{v'u'\} \cup P(u', j)$. Pela propriedade (2), existem as arestas $kl \in A_M$, $lm \in A_M$ e $mk \in A_M$, tais que a aresta uv está associada a kl , a aresta $u'v'$ está associada a km e alguma aresta de $P(v, v')$ está associada a lm . Portanto, existe o ciclo formado pelas arestas kl , lm , mk em G_M , contradizendo o fato de que G_M é uma árvore. Portanto, qualquer solução que atende às restrições (i) a (iv) possui $G_I[\lambda(k)]$ conexo, para todo $k \in V_M$.

■

Capítulo 5

Algoritmos para o PCCA *para árvores*

Neste capítulo, apresentamos dois algoritmos para o PCCA *para árvores*. O primeiro algoritmo é exponencial no grau máximo dos vértices de V_M (se fixarmos uma constante para limitar o grau máximo dos vértices de V_M , o algoritmo passa a ser polinomial nos tamanhos dos grafos de entrada). O segundo algoritmo é polinomial nos tamanhos de G_I e G_M , mas só pode ser aplicado se G_I for um tipo específico de árvore.

5.1 Um Algoritmo de Programação Dinâmica

Nesta seção, apresentamos alguns algoritmos que juntos formam um método de programação dinâmica para resolver o PCCA *para árvores*. Fazemos a análise da complexidade do método e provamos que, se o grau máximo dos vértices de V_M for limitado por uma constante, o mesmo é polinomial.

Estrutura de uma solução ótima

Dada uma instância do PCCA *para árvores*, em um caso específico podemos dividir o problema em questão em dois subproblemas independentes. Ou seja, podemos utilizar as soluções destes dois subproblemas para resolver o problema original.

Lema 5.1.1 *Dadas duas árvores $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$, sejam $ij \in A_I$ e $kl \in A_M$, e seja $\mathcal{M}$ um mapeamento viável do PCCA **para árvores** de V_I para V_M , tal que o vértice i está associado ao cluster k e o vértice j está associado ao cluster l . Sejam C_i e C_j as duas componentes de $G_I[A_I \setminus \{ij\}]$, tais que $i \in V(C_i)$ e $j \in V(C_j)$. Sejam D_k e D_l as duas componentes de $G_M[A_M \setminus \{kl\}]$,*

tais que $k \in V(D_k)$ e $l \in V(D_l)$.

Cada vértice de $V(C_i)$ está associado a algum cluster de $V(D_k)$ em $\mathcal{M}$ e cada vértice de $V(C_j)$ está associado a algum cluster de $V(D_l)$ em $\mathcal{M}$.

Prova. Sejam $S_k \subseteq V(C_i)$ e $S_l \subseteq V(C_i)$, tais que S_k contém todos os vértices de $V(C_i)$ que estão associados a algum cluster de $V(D_k)$ em $\mathcal{M}$, e S_l contém todos os vértices de $V(C_i)$ que estão associados a algum cluster de $V(D_l)$ em $\mathcal{M}$. Claramente, $V(C_i) = S_k \cup S_l$ e $S_k \cap S_l = \emptyset$. Mostraremos que $S_l = \emptyset$ e, portanto, $S_k = V(C_i)$. Ou seja, vamos mostrar que cada vértice de $V(C_i)$ está associado a algum cluster de $V(D_k)$ em $\mathcal{M}$.

Suponha que $S_l \neq \emptyset$. Sabemos que $S_k \neq \emptyset$, pois $i \in V(C_i)$ está associado a $k \in V(D_k)$ em $\mathcal{M}$. Como C_i é conexo, existe pelo menos um vértice de S_l com um vizinho em S_k . Seja $uv \in A_I$, tal que $u \in S_l$ e $v \in S_k$. Sejam $k' \in V(D_k)$ e $l' \in V(D_l)$, tais que u está associado a k' e v está associado a l' . Logo, a aresta uv está associada ao par de clusters k', l' .

Suponha que $k'l' \notin A_M$. Como a aresta uv está associada ao par de clusters k', l' , então a aresta $uv \in A_I$ está descoberta, violando a restrição de cobertura de arestas definida no PCCA *para árvores*. Ou seja, temos que o mapeamento $\mathcal{M}$ é inviável, contradizendo o fato de que $\mathcal{M}$ é viável. Portanto, $k'l' \in A_M$.

Se $k' \neq k$ ou $l' \neq l$ ou ambos, então temos um circuito da forma $P_{G_M}(k', k) \cup \{kl\} \cup P_{G_M}(l, l') \cup \{k', l'\}$ em G_M , contrariando o fato de que G_M é uma árvore. Se $k' = k$ e $l' = l$, temos que $k'l' = kl$, ou seja, a aresta uv está associada à aresta kl . Pelo lema 4.1.1, em um mapeamento viável do PCCA *para árvores* existe no máximo uma aresta de A_I associada a uma mesma aresta de A_M . Como a aresta ij também está associada à aresta kl , temos que o mapeamento $\mathcal{M}$ é inviável, contradizendo o fato de que $\mathcal{M}$ é viável.

Concluimos que $S_l = \emptyset$ e, portanto, $S_k = V(C_i)$. Ou seja, cada vértice de $V(C_i)$ está associado a algum cluster de $V(D_k)$ em $\mathcal{M}$. É fácil ver que a prova de que cada vértice de $V(C_j)$ está associado a algum cluster de $V(D_l)$ em $\mathcal{M}$ é exatamente igual a que acabamos de apresentar. ■

Figura 5.1: À esquerda G_I e à direita G_M . O vértice i está associado ao cluster k , e o vértice j está associado ao cluster l . As áreas contornadas indicam as possíveis associações para os demais vértices de V_I .

Corolário 5.1.1 Dadas duas árvores $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$, sejam $ij \in A_I$ e $kl \in A_M$ e seja $\mathcal{M}$ um mapeamento viável do PCCA **para árvores** de V_I para V_M , tal que o custo de $\mathcal{M}$ é mínimo, considerando todos os mapeamentos nos quais o vértice i está associado ao cluster k e o vértice j está associado ao cluster l . Sejam C_i e C_j as duas componentes de $G_I[A_I \setminus \{ij\}]$, tais que $i \in V(C_i)$ e $j \in V(C_j)$. Sejam D_k e D_l as duas componentes de $G_M[A_M \setminus \{kl\}]$, tais que $k \in V(D_k)$ e $l \in V(D_l)$.

Sejam z , α_1 e α_2 , tais que z é o custo do mapeamento $\mathcal{M}$, α_1 é o custo de um mapeamento ótimo de C_i para D_k , sujeito à restrição de que i deve ser associado a k , e α_2 é o custo de um mapeamento ótimo de C_j para D_l , sujeito à restrição de que j deve ser associado a l . Vale que $z = \alpha_1 + \alpha_2 + d_{kl}^{ij}$, sendo que d_{kl}^{ij} é o custo da associação da aresta ij à aresta kl .

Prova. Pelo lema 5.1.1, vale que cada vértice de $V(C_i)$ está associado a algum cluster de $V(D_k)$ em $\mathcal{M}$ e cada vértice de $V(C_j)$ está associado a algum cluster de $V(D_l)$ em $\mathcal{M}$. Logo, podemos decompor $\mathcal{M}$ em duas partes:

- um mapeamento ótimo $\mathcal{M}_i^k$ de C_i para D_k , sujeito à restrição de que i deve ser associado a k ;
- um mapeamento ótimo $\mathcal{M}_j^l$ de C_j para D_l , sujeito à restrição de que j deve ser associado a l .

Sabemos que a aresta ij está associada à aresta kl em $\mathcal{M}$. Portanto, o custo de d_{kl}^{ij} certamente está “embutido” no custo de $\mathcal{M}$.

Suponha que $z < \alpha_1 + \alpha_2 + d_{kl}^{ij}$. Neste caso, ou o custo de $\mathcal{M}_i^k$ é maior que α_1 , ou o custo de $\mathcal{M}_j^l$ é maior que α_2 ou ambos, contrariando o fato de que $\mathcal{M}_j^l$ e $\mathcal{M}_i^k$ são mapeamentos ótimos.

Suponha que $z > \alpha_1 + \alpha_2 + d_{kl}^{ij}$. Neste caso, podemos compor uma solução viável com a junção das associações contidas em $\mathcal{M}_j^l$ e $\mathcal{M}_i^k$. Tal solução tem custo $\alpha_1 + \alpha_2 + d_{kl}^{ij}$, que é menor do que z , contrariando o fato de z ser o custo de um mapeamento ótimo. ■

Através do corolário 5.1.1, vimos que é possível compor uma solução ótima para uma instância do PCCA *para árvores* a partir de soluções ótimas de subproblemas desta instância, desde que estes subproblemas estejam restritos à condição de que um determinado vértice deve ser associado a um determinado cluster. Portanto, o problema de resolver uma instância do PCCA *para árvores* para duas subárvores, digamos $\mathcal{T}_I \subseteq G_I$ e $\mathcal{T}_M \subseteq G_M$, sujeito à restrição de que um vértice $i \in V(\mathcal{T}_I)$ deve ser associado a um cluster $k \in V(\mathcal{T}_M)$, é um problema de interesse.

Notações

Dada uma árvore $\mathcal{T}$, denotamos por $\mathcal{T}^r$ a árvore $\mathcal{T}$ enraizada em r . Denotamos por $\mathcal{T}^r(i)$ a subárvore de $\mathcal{T}^r$ que está enraizada em i , sendo que $i \in V(\mathcal{T}^r)$. Por simplicidade, em algumas partes do texto não faremos distinção explícita entre *árvore* e *árvore enraizada*, mas tal distinção deve ficar subentendida.

Dadas duas árvores enraizadas $\mathcal{T}^r$ e $\mathcal{T}^q$, diremos que um mapeamento $\mathcal{M}$ de $\mathcal{T}^r$ para $\mathcal{T}^q$ é um (r, q) -mapeamento se r está associado a q em $\mathcal{M}$.

Dada uma árvore enraizada $\mathcal{T}^r$, denotamos por $\delta_{\mathcal{T}^r}^+(u)$, para um vértice $u \in V(\mathcal{T}^r)$, o conjunto dos filhos de u em $\mathcal{T}^r$. Quando implicitamente estiver evidente qual é a árvore $\mathcal{T}^r$, utilizaremos simplesmente $\delta^+(u)$. Usualmente, a notação $\delta^+(u)$ é utilizada somente para grafos orientados, indicando o conjunto dos arcos que saem do vértice u . Podemos imaginar uma árvore enraizada como sendo um grafo orientado, no qual os arcos são direcionados da raiz para as folhas. Dessa forma, o conjunto $\delta^+(u)$ corresponde aos filhos de u .

A idéia do algoritmo “PCCA-para-árvores”

O algoritmo que chamaremos de PCCA-PARA-ÁRVORES utiliza-se do corolário 5.1.1 para construir uma solução ótima através de *divisão e conquista*. Uma tabela de memoização¹ é utilizada a fim de impedir que as soluções de subproblemas iguais sejam calculadas mais de uma vez (técnica conhecida como *programação dinâmica*). A seguir, mostramos informalmente o funcionamento deste algoritmo.

Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ duas árvores. Inicialmente, o algoritmo toma um vértice $r \in V_I$ qualquer como sendo a raiz de G_I . Para cada cluster $q \in V_M$, é construída uma melhor solução na qual r está associado a q . Uma melhor solução dentre estas $|V_M|$ construídas é uma solução ótima para a instância do problema. Ou seja, a eficiência do algoritmo depende da forma como estas $|V_M|$ soluções são construídas.

A cada passo é escolhido um novo cluster $q \in V_M$, sendo que no passo atual o cluster q é considerado como sendo a raiz de G_M . Temos então o seguinte problema: dadas as árvores G_I^r e G_M^q , resolver o PCCA *para árvores* para G_I^r e G_M^q , sujeito à restrição de que r deve ser associado a q .

Sejam $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$ árvores enraizadas, tais que $\mathcal{T}_I \subseteq G_I$, $\mathcal{T}_M \subseteq G_M$, $r \in V(\mathcal{T}_I)$ e $q \in V(\mathcal{T}_M)$. Definimos $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ como sendo o custo de uma solução ótima do PCCA *para árvores* de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q .

Observe que se um vértice $j \in V(\mathcal{T}_I^r)$ está associado ao cluster q , e um vértice i , filho de j , está associado a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, temos que a aresta $ij \in A(\mathcal{T}_I^r)$ está associada à aresta $kq \in A(\mathcal{T}_M^q)$. Pelo corolário 5.1.1, podemos utilizar o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. Ou seja, o problema original pode ser dividido em diversos subproblemas menores e independentes. O algoritmo PCCA-PARA-ÁRVORES trabalhará de forma a garantir que antes de calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já tenha sido calculado, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Mais adiante, ficará clara a razão da exigência de se calcular previamente estes valores. Diremos que a tarefa de calcular os valores de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$, será a etapa de divisão.

Após a etapa de divisão, temos a etapa de conquista, na qual precisamos combinar as soluções dos subproblemas para resolver o problema original. A etapa de conquista deste algoritmo é enumerativa e sua compreensão depende de algumas propriedades que serão provadas no final desta seção. Por isso, neste primeiro momento, daremos apenas uma idéia de como esta etapa funciona, sem explicar porque ela funciona. Veremos posteriormente que, dependendo da estrutura da árvore $\mathcal{T}_I$, é possível

¹Este termo ainda não é totalmente padronizado, sendo confundido às vezes com o termo “memorização”.

formular esta etapa como um problema de emparelhamento máximo de custo mínimo.

Dados $i \in \mathcal{T}_I^r$ e um mapeamento $\mathcal{M}$ de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, denotaremos por $\omega(i, \mathcal{M})$ o cluster para o qual i está associado, considerando o mapeamento $\mathcal{M}$. Definimos ainda os conjuntos $Q(\mathcal{M})$ e $H(\mathcal{M})$, e uma função c^* da seguinte maneira:

$$Q(\mathcal{M}) := \{i \in V(\mathcal{T}_I^r) \mid \omega(i, \mathcal{M}) = q\},$$

$$H(\mathcal{M}) := \{i \in V(\mathcal{T}_I^r) \mid \omega(i, \mathcal{M}) \in \delta_{\mathcal{T}_M^q}^+(q) \text{ e se } j \text{ é ancestral de } i \text{ então } \omega(j, \mathcal{M}) \notin \delta_{\mathcal{T}_M^q}^+(q)\},$$

$$c^*(\mathcal{M}) = c_{rq} + \sum_{i \in Q(\mathcal{M}) \setminus \{r\}} (c_{iq} + d_{qq}^{i, \text{pai}[i]}) + \sum_{i \in H(\mathcal{M})} (\Phi(\mathcal{T}_I^r(i), \mathcal{T}_I^q(k)) + d_{kq}^{i, \text{pai}[i]}), \text{ sendo que } k = \omega(i, \mathcal{M}).$$

O custo de $\mathcal{M}$ será denotado por $c(\mathcal{M})$. Repare que nem sempre vale que $c(\mathcal{M}) = c^*(\mathcal{M})$. Conseqüentemente, para mapeamentos diferentes, digamos $\mathcal{M}'$ e $\mathcal{M}''$, pode valer que $c(\mathcal{M}') \neq c(\mathcal{M}'')$ e $c^*(\mathcal{M}') = c^*(\mathcal{M}'')$. Um de nossos principais objetivos nesta seção é mostrar que se $\mathcal{M}$ é ótimo, então $c(\mathcal{M}) = c^*(\mathcal{M})$.

Seja $\mathcal{M}^*$ um mapeamento ótimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q . Se soubermos qual é o vértice de menor nível de $V(\mathcal{T}_I^r) \setminus \{r\}$ que está associado a cada cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ (quando houver algum vértice associado ao respectivo cluster k), já temos informação suficiente para calcular o custo de $\mathcal{M}^*$. Usaremos este fato para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

Suponha que um vértice $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ está associado a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, tal que i é o vértice de menor nível associado a k (provaremos que se existe tal vértice, então ele é único). Neste caso, utilizaremos o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ no cálculo de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. Todos os vértices de $V(\mathcal{T}_I^r) \setminus \{r\}$ são candidatos a serem associados a cada um dos clusters de $\delta_{\mathcal{T}_M^q}^+(q)$. Por isso, antes de calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, temos que calcular o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$, para depois enumerar todas as possibilidades.

Para cada cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, consideraremos cada um dos vértices de $V(\mathcal{T}_I^r) \setminus \{r\}$ como sendo o vértice de menor nível associado a k . Sendo assim, temos $O(n^w)$ possibilidades a serem consideradas, sendo que $n = |V(\mathcal{T}_I^r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$. Repare que w corresponde ao grau de q . Se fixarmos uma constante para estipular qual será o grau máximo dos vértices de $V(\mathcal{T}_M^q)$, então teremos uma quantidade polinomial de possibilidades a serem consideradas. O algoritmo que chamaremos de COMBINA-SOLUÇÕES se encarregará de fazer a enumeração dessas $O(n^w)$ soluções. Por enquanto,

utilizaremos o algoritmo COMBINA-SOLUÇÕES como uma “caixa-preta” que devolve o valor de um mapeamento ótimo entre duas árvores, tal que a raiz de uma deve ser associada à raiz de outra. O algoritmo COMBINA-SOLUÇÕES recebe como parâmetros as árvores em questão.

Definimos $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ através da seguinte recorrência:

$$\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = \begin{cases} c_{rq}, & \text{se } r \text{ é folha.} \\ c_{rq} + \sum_{i \in V(\mathcal{T}_I^r) \setminus \{r\}} (c_{iq} + d_{qq}^{ij}), & \text{se } q \text{ é folha (sendo que } j \text{ é o pai de } i\text{).} \\ \min\{c^*(\mathcal{M}) \mid \mathcal{M} \text{ é um } (r, q)\text{-mapeamento}\}, & \text{caso contrário.} \end{cases}$$

Sabemos que r deve ser associado a q , e que r e q são as raízes de $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$, respectivamente. Quando r é folha, o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ corresponde ao custo da associação entre r e q . Quando q é folha, a única solução viável é a que possui todos os vértices de $V(\mathcal{T}_I^r)$ associados a q . Ou seja, neste caso o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ corresponde à soma dos custos das associações de todos os vértices de $V(\mathcal{T}_I^r)$ para q . Além disso, deve-se somar a $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ a soma dos custos das associações de todas as arestas de $A(\mathcal{T}_I^r)$ para q , pois as arestas de $A(\mathcal{T}_I^r)$ estão associadas a q no mapeamento correspondente. Quando nem r e nem q são folhas, calcularemos $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ a partir dos valores de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}(q)$, utilizando o algoritmo COMBINA-SOLUÇÕES.

A seguir, apresentamos uma versão preliminar do algoritmo PCCA-PARA-ÁRVORES. O algoritmo recebe como parâmetros duas árvores G_I e G_M e devolve o custo de um mapeamento ótimo de G_I para G_M .

PCCA-PARA-ÁRVORES(G_I, G_M)

- 1 Seja $r \in V_I$
- 2 $menorCusto \leftarrow \infty$
- 3 **para** todo $q \in V_M$
- 4 **faça** $\alpha \leftarrow \text{CALCULA}\Phi(G_I^r, G_M^q)$
- 5 **se** $\alpha < menorCusto$
- 6 **então** $menorCusto \leftarrow \alpha$
- 7 devolva $menorCusto$

CALCULA $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$

- 1 **se** $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ está memoizado
- 2 **então** $\alpha \leftarrow$ o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ que está memoizado
- 3 **senão se** r é folha
- 4 **então** $\alpha \leftarrow c_{rq}$
- 5 **senão se** q é folha
- 6 **então** $\alpha \leftarrow c_{rq} + \sum_{i \in V(\mathcal{T}_I^r) \setminus \{r\}} (c_{iq} + d_{qq}^{ij}) \triangleright$ sendo que j é o pai de i
- 7 **senão para** cada $i \in V(\mathcal{T}_I^r) \setminus \{r\}$
- 8 **faça para** cada $k \in \delta_{\mathcal{T}_M^q}^+(q)$
- 9 **faça** CALCULA $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$
- 10 $\alpha \leftarrow$ COMBINA-SOLUÇÕES($\mathcal{T}_I^r, \mathcal{T}_M^q$)
- 11 memoize α como sendo o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$
- 12 devolva α

A justificativa para a utilização de programação dinâmica é que se considerarmos duas raízes diferentes para G_M , podem existir diversas subárvores iguais nestas duas árvores “diferentes”. Sejam G_M^q e G_M^p , tais que $p, q \in V_M$ e $p \neq q$. Seja $k \in V_M$ um vértice qualquer. Note que a subárvore $G_M^q(k)$ é igual a $G_M^p(k)$ se e somente se os pais de k em G_M^q e G_M^p são iguais (veja as figuras 5.2 e 5.3). Intuitivamente, percebe-se que uma mesma subárvore de G_M ocorre considerando-se diversas raízes diferentes para G_M . Desta forma, uma vez calculado o custo de um mapeamento ótimo de uma subárvore de G_I para uma subárvore de G_M , esta solução pode ser reaproveitada diversas vezes.

Figura 5.2: Considerando raízes diferentes para G_M , se o pai de um vértice $k \in V_M$ é o mesmo nas duas árvores, então as subárvores enraizadas em k nas duas árvores são iguais.

A idéia geral do algoritmo PCCA-PARA-ÁRVORES foi descrita. Para formalizar os conceitos vistos até aqui, precisamos definir mais algumas notações e mostrar alguns algoritmos auxiliares. Faremos isso nas próximas páginas. No final desta seção, apresentamos de forma completa o algoritmo

Figura 5.3: Considerando raízes diferentes para G_M , se o pai de um vértice $k \in V_M$ não é o mesmo nas duas árvores, então as subárvores enraizadas em k nas duas árvores são diferentes.

PCCA-PARA-ÁRVORES.

Representação de árvores

Dado um grafo $G = (V, A)$, tal que G é uma árvore enraizada, definimos os seguintes atributos:

- $pai[v] \in V$, para todo $v \in V$, sendo o vértice pai de v em G . Por conveniência, definimos que se v é a raiz de G , então $pai[v] = v$;
- $folha[v] \in \{\text{VERDADEIRO}, \text{FALSO}\}$, para todo $v \in V$, sendo $folha[v] = \text{VERDADEIRO}$ se v é uma folha (um vértice que não tem filhos) de G , e caso contrário $folha[v] = \text{FALSO}$.

É importante observar que os valores dos atributos descritos anteriormente dependem de qual vértice está sendo considerado como a raiz de G . Não definimos nenhum atributo específico para determinar se um vértice v é ou não a raiz de G . Porém, temos esta informação através do atributo $pai[v]$ que é igual a v se e somente se v é a raiz de G .

A seguir, apresentamos um algoritmo específico para atribuir os devidos valores a esses atributos. O algoritmo que chamaremos de MONTA-ÁRVORE recebe os seguintes parâmetros: um grafo $G = (V, A)$, tal que G é um grafo acíclico e conexo; um vértice $r \in V$, que será a raiz de G .

Após a execução do algoritmo, os atributos dos vértices contidos em V estarão com seus respec-

tivos valores corretos, considerando r como sendo a raiz de G . Ou seja, G será uma árvore enraizada em r .

MONTA-ÁRVORE(G, r)

1 MONTA-SUBÁRVORE(G, r, r)

O algoritmo MONTA-SUBÁRVORE recebe os seguintes parâmetros: um grafo $G = (V, A)$; dois vértices $u, p_u \in V$.

Após a execução do algoritmo, o vértice u e todos os seus descendentes estão com seus atributos corretos, considerando u como sendo a raiz da respectiva subárvore e p_u como sendo o pai de u em G . Se u é a raiz de G , então deve valer que $p_u = u$, conforme definimos no início desta seção.

MONTA-SUBÁRVORE(G, u, p_u)

1 $pai[u] \leftarrow p_u$

2 $folha[u] \leftarrow \text{VERDADEIRO}$

3 **para** cada $v \in \delta^+(u)$

4 **faça** $folha[u] \leftarrow \text{FALSO}$

5 MONTA-SUBÁRVORE(G, v, u)

Lema 5.1.2 *O algoritmo MONTA-SUBÁRVORE está correto, ou seja, ao final de sua execução os atributos de u e de todos os seus descendentes estão corretos, considerando o vértice $u \in V$ como sendo a raiz da respectiva subárvore e p_u como sendo o pai de u em G .*

Prova. Por indução em n , sendo que n corresponde à quantidade de vértices contidos na subárvore enraizada em u , considerando p_u como sendo o pai de u em G .

O caso base é quando $n = 1$. Neste caso, o único vértice da subárvore é o u . Todos os atributos de u são atribuídos corretamente nas linhas 1 e 2. Como u não tem vizinhos, então as demais linhas não são executadas e o algoritmo termina.

Suponha que $n > 1$. Todos os atributos de u são atribuídos corretamente nas linhas 1 e 2 (supondo que u não tem filhos). Então o laço das linhas 3 a 5 é executado para cada filho de u . Se o laço das linhas 3 a 5 for executado alguma vez, então o atributo $folha$ do vértice u é “corrigido” na linha 4. Para cada filho de u , na linha 5 é feita uma chamada recursiva a MONTA-SUBÁRVORE que, por hipótese de indução, ao final de sua execução os atributos de v e todos os seus descendentes estão

corretos, considerando o vértice $v \in V$ como sendo a raiz da respectiva subárvore e u como sendo o pai de v em G . Como os atributos de u e de todos os seus descendentes estão corretos, então o algoritmo MONTA-SUBÁRVORE está correto. ■

Corolário 5.1.2 *O algoritmo MONTA-ÁRVORE está correto, ou seja, ao final de sua execução os atributos de V estão corretos, considerando o vértice $u \in V$ como sendo a raiz de G .*

Lema 5.1.3 *O algoritmo MONTA-SUBÁRVORE executa $\Theta(n)$ operações, sendo que n corresponde à quantidade de vértices contidos na subárvore enraizada em u .*

Prova. Considere a seguinte recorrência:

$$F(n) = \begin{cases} 2, & \text{se } n = 1 \\ F(n-1) + 2, & \text{se } n > 1 \end{cases}$$

Mostraremos que o algoritmo MONTA-SUBÁRVORE executa $F(n)$ operações. Quando $n = 1$, ou seja, o vértice u não possui nenhum filho, somente as linhas 1 e 2 são executadas. Neste caso, temos que o algoritmo MONTA-SUBÁRVORE executa 2 operações e $F(1) = 2$. Quando $n > 1$ temos $m = g(u) - 1$ (ou $m = g(u)$ se u for a raiz de G , sendo que $g(u)$ corresponde ao grau de u) chamadas recursivas a MONTA-SUBÁRVORE, além das 2 operações executadas nas linhas 1 e 2. Logo, quando $n > 1$, a quantidade de operações que o algoritmo MONTA-SUBÁRVORE executa é $2 + \sum_{v \in \delta^+(u)} F(t(v))$, sendo que $t(v)$ corresponde ao tamanho da subárvore enraizada em v . Somando o tamanho de todas as m subárvores diferentes, temos $n - 1$ vértices. Mostraremos que $\sum_{v \in \delta^+(u)} F(t(v)) = F(n - 1)$. Mas antes, precisamos resolver a recorrência.

Provaremos, por indução em n , que $F(n) = 2n$, para $n \geq 1$. O caso base da indução é quando $n = 1$. Neste caso $F(1) = 2 = 2n$. Suponha que $n > 1$. Por hipótese de indução $F(n) = 2(n-1) + 2 = 2n$. Ou seja, para qualquer $n \geq 1$, vale que $F(n) = 2n$.

Como $F(n)$ é linear, então $\sum_{v \in \delta^+(u)} F(t(v)) = F(n - 1)$. Logo, a quantidade de operações executadas pelo algoritmo MONTA-SUBÁRVORE é dada por $F(n)$, sendo que n corresponde ao número de vértice contidos na subárvore enraizada em u . Claramente, $F(n) = \Theta(n)$. Portanto, o algoritmo MONTA-SUBÁRVORE executa $\Theta(n)$ operações. ■

Corolário 5.1.3 *O algoritmo MONTA-ÁRVORE executa $\Theta(n)$ operações, sendo que $n = |V|$.*

A seguir, apresentamos o algoritmo que chamaremos de MESMO-CAMINHO. Dada uma árvore enraizada G^r , tal que G é uma árvore qualquer, o algoritmo MESMO-CAMINHO recebe os seguintes parâmetros: três vértices $r', v, u \in V$, tais que, $u \neq r'$ e os vértices u e v estão contidos em $G^r(r')$ (assumimos que os atributos dos vértices de $V(G^r)$ foram atribuídos previamente através do algoritmo MONTA-ÁRVORE). O algoritmo MESMO-CAMINHO devolve VERDADEIRO se o vértice u estiver contido no caminho de v até r' em G^r . Caso contrário, o algoritmo devolve FALSO.

MESMO-CAMINHO(u, v, r')

```

1  se  $v = r'$ 
2    então devolva FALSO
3  senão se  $v = u$ 
4    então devolva VERDADEIRO
5  senão devolva MESMO-CAMINHO( $u, \text{pai}[v], r'$ )
```

Lema 5.1.4 *O algoritmo MESMO-CAMINHO está correto, ou seja, devolve VERDADEIRO se e somente se o vértice u estiver contido no caminho de v a r' em G^r .*

Prova. Por indução em n , sendo que n é o comprimento do caminho de v a r' em G^r .

O caso base é quando $n = 0$, ou seja, $v = r'$ é o único vértice contido no caminho de v a r' . Como $u \neq r'$ (por definição), temos que u não está contido no caminho de v a r' . Neste caso, a linha 2 é executada e o algoritmo devolve FALSO. Portanto, o algoritmo está correto para o caso base.

Suponha que $n > 0$. Podemos então dividir os vértices contidos no caminho de v a r' nas seguintes duas partes: o vértice v , e os vértices contidos no caminho $P(\text{pai}[v], r')$. Se $u = v$, temos que u está contido no caminho de v a r' em G^r . Neste caso, a linha 4 é executada e o algoritmo devolve VERDADEIRO. Caso $u \neq v$, resta verificar se u está contido em $P(\text{pai}[v], r')$. Por hipótese de indução, a chamada recursiva feita na linha 5 devolve VERDADEIRO se e somente se u está contido em $P(\text{pai}[v], r')$. Ou seja, o algoritmo devolve o valor correto.

Portanto, o algoritmo MESMO-CAMINHO está correto. ■

Lema 5.1.5 *O algoritmo MESMO-CAMINHO executa $O(n)$ operações no pior caso, sendo que n corresponde à quantidade de vértices contidos no caminho de v a r' .*

Prova. Seja $h(n)$ a quantidade de operações que o algoritmo MESMO-CAMINHO executa quando o comprimento do caminho de v até r' é n . Quando $n = 0$, temos que o comprimento do caminho de v até r' em G^r é zero ($v = r'$). Neste caso, as linhas 1 e 2 são executadas e, portanto, $h(0) = 2$. Quando $n > 0$, temos duas possibilidades. Se $v = u$, as linhas 1, 3 e 4 são executadas. Neste caso, $h(n) = 3$. Caso contrário, as linhas 1, 2 e 5 são executadas. Na linha 5 é feita uma chamada recursiva para “procurar” o vértice u no caminho de $\text{pai}[v]$ a r' , que é um caminho de comprimento $n - 1$. Neste caso, temos que $h(n) = h(n - 1) + 2$. Como $h(n - 1)$ corresponde à execução de pelo menos uma operação, quando $n > 0$, temos que $h(n) \leq h(n - 1) + 2$.

Portanto, a quantidade de instruções executadas pelo algoritmo MESMO-CAMINHO é limitada pela seguinte recorrência:

$$F(n) = \begin{cases} 2, & \text{se } n = 0 \\ F(n - 1) + 2 & \end{cases}$$

(ou seja, $h(n) \leq F(n)$).

Mostraremos, por indução em n , que $F(n) \leq 4n$, para qualquer $n \geq 1$. O caso base é quando $n = 1$. Neste caso $F(1) = F(0) + 2 = 4 \leq 4n$. Suponha que $n > 1$. Então $F(n) = F(n - 1) + 2$. Por hipótese de indução, $F(n - 1) \leq 4(n - 1)$, logo $F(n) \leq 4(n - 1) + 2 = 4n - 2 \leq 4n$.

Portanto, o algoritmo MESMO-CAMINHO é $O(n)$. ■

Algumas propriedades do PCCA para árvores

A seguir, observaremos algumas propriedades do PCCA para árvores que serão úteis na construção do algoritmo que apresentaremos no final desta seção.

Lema 5.1.6 *Sejam $G_I = (V_I, A_I)$ e $G_M = (V_M, A_M)$ duas árvores e sejam $i, j \in V_I$ e $k, l \in V_M$, tais que $i \neq j$ e $k \neq l$. Seja $\mathcal{M}$ um mapeamento viável do PCCA para árvores de G_I para G_M , tal que i está associado a k em $\mathcal{M}$ e j está associado a l em $\mathcal{M}$.*

Considere os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$, e sejam $n = |V(P_{G_I}(i, j))|$ e $m = |V(P_{G_M}(k, l))|$. Suponha que os vértices de $P_{G_I}(i, j)$ possuem índices $(1, 2, \dots, n)$, tais que o índice de cada vértice de $V(P_{G_I}(i, j))$ corresponde à ordem em que ele se encontra no caminho $P_{G_I}(i, j)$. Analogamente, suponha que os vértices de $P_{G_M}(k, l)$ possuem índices $(1, 2, \dots, m)$. Seja $\text{ind}(u)$ o índice de um vértice u em um dado caminho (exemplo: considerando os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$, vale que $\text{ind}(i) = \text{ind}(k) = 1$, $\text{ind}(j) = n$ e $\text{ind}(l) = m$).

Sejam u e v dois vértices contidos no caminho $P_{G_I}(i, j)$, tais que $uv \in A_I$ e $\text{ind}(u) = \text{ind}(v) - 1$. Sejam u' e v' dois clusters de V_M , tais que u está associado a u' e v está associado a v' (considerando o mapeamento $\mathcal{M}$). As seguintes afirmações são verdadeiras: (i) os clusters u' e v' estão contidos no caminho $P_{G_M}(k, l)$; (ii) ou $\text{ind}(u') = \text{ind}(v')$ ou $\text{ind}(u') = \text{ind}(v') - 1$.

Prova. Por indução em n , sendo que n corresponde à quantidade de vértices contidos no caminho $P_{G_I}(i, j)$.

O caso base da indução é quando $n = 2$. Neste caso, o caminho $P_{G_I}(i, j)$ é constituído pelos vértices i e j , sendo que $ij \in A_I$. Como i está associado a k e j está associado a l , então a aresta ij está associada ao par de clusters k, l . Como $k \neq l$, pela restrição de cobertura de arestas, existe uma aresta $kl \in A_M$. Ou seja, o caminho $P_{G_M}(k, l)$ é constituído pelos vértices k e l , sendo que $kl \in A_M$. Claramente, para o caso base, as afirmações (i) e (ii) são verdadeiras.

Suponha que $n > 2$. Seja $i' \in V_I$, tal que $ii' \in A_I$ e i' está contido no caminho $P_{G_I}(i, j)$. Observe que a quantidade de vértices contidos no caminho $P_{G_I}(i', j)$ é $n - 1$. Seja $k' \in V_M$, tal que i' está associado a k' . Por hipótese de indução, as afirmações (i) e (ii) são verdadeiras para os caminhos $P_{G_I}(i', j)$ e $P_{G_M}(k', l)$. Vamos mostrar que as afirmações (i) e (ii) são verdadeiras para os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$.

Temos dois casos a serem analisados:

- (1) Se $k = k'$, então $P_{G_M}(k, l) = P_{G_M}(k', l)$.
- (2) O outro caso é quando $k \neq k'$. Mostraremos que neste caso $kk' \in A_M$ e o cluster k não está contido no caminho $P_{G_M}(k', l)$, e concluiremos que $P_{G_M}(k, l) = \{kk'\} \cup P_{G_M}(k', l)$.

Como $ii' \in A_I$ e a aresta ii' está associada ao par de clusters kk' , pela restrição de cobertura de arestas, existe uma aresta $kk' \in A_M$. Suponha que k está contido no caminho $P_{G_M}(k', l)$. Por hipótese de indução as afirmações (i) e (ii) são verdadeiras para os caminhos $P_{G_I}(i', j)$ e $P_{G_M}(k', l)$. Logo, é fácil ver que para cada aresta a contida no caminho $P_{G_M}(k', l)$ existe alguma aresta de $P_{G_I}(i', j)$ associada a a . Como a aresta ii' (que não está contida no caminho $P_{G_I}(i', j)$) está associada à aresta kk' (que está contida no caminho $P_{G_M}(k', l)$), temos uma contradição com o lema 4.1.1, que diz que em um mapeamento viável do PCCA para árvores existe no máximo uma aresta de A_I associada a uma mesma aresta de A_M . Portanto, $kk' \in A_M$ e o cluster k não está contido no caminho $P_{G_M}(k', l)$. Logo, $P_{G_M}(k, l) = \{kk'\} \cup P_{G_M}(k', l)$.

Nos dois casos, a afirmação (i) vale para os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$ (note que $P_{G_I}(i, j) = \{ii'\} \cup P_{G_I}(i', j)$, que i está associado a k , e que por hipótese de indução a afirmação (i) vale para os caminhos $P_{G_I}(i', j)$ e $P_{G_M}(k', l)$). Resta mostrar que a afirmação (ii) é verdadeira para os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$. Considerando os casos (1) e (2), por hipótese de indução a afirmação (ii) é verdadeira se tomarmos quaisquer dois vértices consecutivos contidos no caminho $P_{G_I}(i', j)$. Resta mostrar que a afirmação (ii) é verdadeira se tomarmos i e i' .

Sabemos que $P_{G_I}(i, j) = \{ii'\} \cup P_{G_I}(i', j)$. Logo, $ind(i) = 1$ e $ind(i') = 2$ (ou seja, $ind(i) = ind(i') - 1$). Como i está associado a k e i' está associado a k' , precisamos mostrar que ou $ind(k) = ind(k')$ ou $ind(k) = ind(k') - 1$. No caso (1), onde $k = k'$, temos que $ind(k) = ind(k')$. No caso (2), onde $k \neq k'$, como $P_{G_M}(k, l) = \{kk'\} \cup P_{G_M}(k', l)$, temos que $ind(k) = 1$ e $ind(k') = 2$. Ou seja, $ind(k) = ind(k') - 1$. Portanto, a afirmação (ii) é verdadeira para os caminhos $P_{G_I}(i, j)$ e $P_{G_M}(k, l)$ (veja a figura 5.4). ■

Figura 5.4: Pelo lema 5.1.6, se i está associado a k e j está associado a l , então todos os vértices no caminho $P_{G_I}(i, j)$ estão associados a vértices no caminho $P_{G_M}(k, l)$. Vale também que para cada vértice k' em $P_{G_M}(k, l)$, existe pelo menos um vértice de $P_{G_I}(i, j)$ associado a k' . Observe que, como consequência da propriedade enunciada no lema 5.1.6, não há cruzamento entre as setas.

Lema 5.1.7 *Sejam $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$ árvores enraizadas. Seja $\mathcal{M}$ um mapeamento viável de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, tal que r está associado a q . Suponha que existe um vértice $i \in V(\mathcal{T}_I^r) \setminus \{r\}$, tal que i está associado a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}$, e nenhum ancestral de i está associado a k . Temos que, cada vértice de $V(\mathcal{T}_I^r(i))$ está associado a algum cluster contido em $V(\mathcal{T}_M^q(k))$ e todos os ancestrais de i em $\mathcal{T}_I^r$ estão associados a q .*

Prova. Sabemos que o vértice r está associado ao cluster q e o vértice i está associado ao cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Logo, a aresta $\{i, \text{pai}[i]\}$ está associada à aresta $qk \in A_M$, pois o vértice $\text{pai}[i]$ não está associado a k por hipótese. Como $\mathcal{M}$ é viável, pelo lema 5.1.1, cada vértice de $V(\mathcal{T}_I^r(i))$ está

associado a algum cluster de $V(\mathcal{T}_M^q(k))$ em $\mathcal{M}$. Resta provar que todos os ancestrais de i em $\mathcal{T}_I^r$ estão associados a q em $\mathcal{M}$.

Como r está associado a q e i está associado a k , pelo lema 5.1.6, todos os vértices contidos no caminho $P_{\mathcal{T}_I^r}(i, r)$ (que é composto por i e todos seus ancestrais em $\mathcal{T}_I^r$) estão associados a clusters contidos no caminho $P_{\mathcal{T}_M^q}(k, q)$ (que contém apenas os clusters k e q , pois $kq \in A(\mathcal{T}_M^q)$). Como nenhum ancestral de i está associado a k , então todos os ancestrais de i estão associados a q em $\mathcal{M}$. ■

Ilustramos na figura 5.5 as afirmações do lema 5.1.7.

Figura 5.5: À esquerda $\mathcal{T}_I^r$ e à direita $\mathcal{T}_M^q$, sendo que r e q são as raízes de $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$, respectivamente. O vértice r está associado ao cluster q e o vértice i está associado ao cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$. As regiões contornadas e as setas indicam as afirmações do lema 5.1.7.

Lema 5.1.8 *Sejam $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$ árvores enraizadas. Seja $\mathcal{M}$ um mapeamento viável de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, tal que r está associado a q . Se dois vértices distintos $i, j \in V(\mathcal{T}_I^r) \setminus \{r\}$ estão associados a algum cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}$, então uma das seguintes afirmações é verdadeira: (i) i é ancestral de j em $\mathcal{T}_I^r$; (ii) i é descendente de j em $\mathcal{T}_I^r$; (iii) i e j possuem um ancestral em comum associado a k em $\mathcal{M}$.*

Prova. Suponha que as afirmações (i), (ii) e (iii) são falsas.

Sabemos que r está associado a q . Pelo lema 5.1.6, existe uma aresta uv , tal que uv está contida

no caminho $P_{\mathcal{T}_I^r}(i, r)$, e uv está associada à aresta kq . Da mesma forma, existe uma aresta $u'v'$, tal que $u'v'$ está contida no caminho $P_{\mathcal{T}_I^r}(j, r)$, e $u'v'$ está associada à aresta kq .

Suponha que $uv \neq u'v'$. Temos então uma contradição com o lema 4.1.1, que diz que em um mapeamento viável do PCCA *para árvores* existe no máximo uma aresta de G_I associada a uma mesma aresta de G_M . Logo, $uv = u'v'$. Como estamos supondo que as afirmações (i) e (ii) são falsas, então tanto u como v são ancestrais de i e de j . Sem perda de generalidade, temos que u está associado a q e v está associado a k . Temos então uma contradição com a suposição de que a afirmação (iii) é falsa.

Portanto, pelo menos uma das afirmações (i), (ii) e (iii) é verdadeira. ■

Utilizando os últimos três lemas e o corolário 5.1.1, mostraremos que é possível calcular o custo de um mapeamento ótimo, mesmo sem saber explicitamente quais são todas as associações contidas em tal mapeamento. Faremos isto com o auxílio de um vetor que representará as associações “importantes” contidas em um mapeamento.

Vetores de associações

Sejam $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$ árvores enraizadas. Suponha que os vértices de $V(\mathcal{T}_I^r) \setminus \{r\}$ possuam rótulos $(1, 2, \dots, n)$ e que os clusters de $\delta_{\mathcal{T}_M^q}^+(q)$ possuam rótulos $(1, 2, \dots, w)$, sendo que $n = |V(\mathcal{T}_I^r)| - 1$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$. Seja $\mathcal{A} = \{a_1, a_2, \dots, a_w\}$ um vetor de inteiros de w posições, tal que $0 \leq a_k \leq n$, para todo $1 \leq k \leq w$. O vetor $\mathcal{A}$ representará um conjunto de associações. Cada elemento a_k de $\mathcal{A}$ corresponde ao rótulo do vértice $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ que está associado ao cluster de rótulo k , tal que o nível de i é mínimo. Quando $a_k = 0$, significa que nenhum vértice de $V(\mathcal{T}_I^r) \setminus \{r\}$ está associado ao cluster k . Por simplicidade, nos referiremos a a_k como sendo um vértice, no lugar de “o vértice de rótulo a_k ” (supondo que $a_k \neq 0$). Analogamente, no lugar de “o cluster de rótulo k ”, diremos “o cluster k ”.

Sejam k e l , tais que $k \neq l$, $1 \leq k, l \leq w$, $a_k \neq 0$ e $a_l \neq 0$. Então, deve valer que $a_k \neq a_l$ (um vértice não pode ser associado a dois clusters distintos) e os vértices a_k e a_l não estão no mesmo caminho até a raiz r , ou seja, a_k não é ancestral e nem descendente de a_l . Esta exigência vem do fato de que, conforme vimos, se um vértice $i \in V(\mathcal{T}_I^r)$ está associado a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, tal que o nível de i é mínimo, então todos os ancestrais de i estão associados a q e cada descendente de i está associado a algum cluster contido na subárvore $\mathcal{T}_M^q(k)$. Ou seja, em nenhuma solução viável o

vértice a_k é descendente ou ancestral do vértice a_l .

Diremos que um vetor que atende às propriedades enunciadas anteriormente é um *vetor de associações válido*, ou simplesmente um *vetor de associações*. Utilizaremos vetores de associações com a finalidade de representar mapeamentos. Note que apenas parte das associações de um determinado mapeamento estão contidas no vetor de associações que o representa. A seguir, provaremos que é possível calcular o custo de um mapeamento ótimo se soubermos qual é o vetor de associações que o representa.

Seja $\mathcal{M}$ um mapeamento viável do PCCA para árvores de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, tal que r está associado a q , e o custo de $\mathcal{M}$ é mínimo. Considerando $\mathcal{M}$, seja $S \subseteq \{V(\mathcal{T}_I^r) \setminus \{r\}\}$ o conjunto de todos os vértices que não estão associados a clusters de $\delta_{\mathcal{T}_M^q}^+(q)$ e que não são descendentes de vértices associados a clusters de $\delta_{\mathcal{T}_M^q}^+(q)$.

Lema 5.1.9 *Sejam $\mathcal{T}_I^r$, $\mathcal{T}_M^q$, $\mathcal{M}$ e S , conforme acima. Seja $\mathcal{A} = \{a_1, a_2, \dots, a_w\}$ um vetor de associações representando $\mathcal{M}$.*

O custo de $\mathcal{M}$ é dado por:

$$c_{rq} + \sum_{a_k \in \mathcal{A} | a_k \neq 0} (\Phi(\mathcal{T}_I^r(a_k), \mathcal{T}_M^q(k)) + d_{kq}^{a_k, pai[a_k]}) + \sum_{i \in S} (c_{iq} + d_{qq}^{i, pai[i]}).$$

Prova. Considerando $\mathcal{M}$, seja $\alpha(i)$ o custo da associação do vértice $i \in V(\mathcal{T}_I^r)$ ao seu respectivo cluster, e seja $\beta(ij)$ o custo da associação da aresta $ij \in A(\mathcal{T}_I^r)$ ao seu respectivo par de clusters. Seja $c(X) = \sum_{i \in X} \alpha(i) + \sum_{ij | i, j \in X} \beta(ij)$, tal que $X \subseteq V(\mathcal{T}_I^r)$. Seja $d(Y) = \sum_{ij \in Y} \beta(ij)$, tal que $Y \subseteq A(\mathcal{T}_I^r)$.

Sejam $S' = S \cup \{r\}$, $H = V(\mathcal{T}_I^r) \setminus S'$ e $Q = \{ij \in A(\mathcal{T}_I^r) \mid i \in H, j \in S'\}$. Como $V(\mathcal{T}_I^r) = H \cup S'$, é fácil ver que o custo de $\mathcal{M}$ é dado por $c(H) + c(S') + d(Q)$.

Tome um vértice $i \in H$. Pela definição de H , sabemos que ou $i = a_k$ ou i é descendente de a_k , para algum $a_k \in \mathcal{A}$. Como a_k é um vértice de menor nível associado a k , pelo lema 5.1.8, não existe nenhum vértice com nível igual ao de a_k associado a k e todos os vértices associados a k são descendentes de a_k . Logo,

$$c(H) = \sum_{a_k \in \mathcal{A} | a_k \neq 0} \Phi(\mathcal{T}_I^r(a_k), \mathcal{T}_M^q(k)).$$

Figura 5.6: No exemplo acima, temos que $w = 5$ e $\mathcal{A} = \{0, 7, 0, 0, 13\}$ (lembrando que $w = |\mathcal{A}|$). Nesta prova estamos mostrando que, dado que r está associado a q , saber apenas a configuração de $\mathcal{A}$ já é suficiente para calcular o custo de $\mathcal{M}$.

Da forma como H é definido (H é composto pelos vértices de $\mathcal{A}$ e seus descendentes), temos que todas as arestas de Q possuem um extremo em $\mathcal{A}$. Pelo lema 5.1.7, todos os ancestrais dos vértices de $\mathcal{A}$ estão associados a q . Logo,

$$d(Q) = \sum_{a_k \in \mathcal{A} | a_k \neq 0} d_{kq}^{a_k, \text{pai}[a_k]}.$$

Tome um vértice $j \in S'$. Sabemos que se $j = r$, então j está associado a q . Pela definição de S , vale que j não está associado a nenhum cluster de $\delta_{T_M^q}^+(q)$ e j não é descendente de nenhum vértice associado a um cluster de $\delta_{T_M^q}^+(q)$. Como r está associado a q , pelo lema 5.1.6, se j está associado a algum cluster $l \in V(\mathcal{T}_M^q) \setminus (\{q\} \cup \delta_{T_M^q}^+(q))$, então existe algum ancestral de j que está associado a algum cluster $k \in \delta_{T_M^q}^+(q)$. Temos então uma contradição com o fato de que j não é descendente de nenhum vértice associado a um cluster de $\delta_{T_M^q}^+(q)$. Logo, j só pode estar associado a q . Portanto,

$$c(S') = \sum_{i \in S'} c_{iq} + \sum_{ij \in A(\mathcal{T}_I^r) | i, j \in S'} d_{qq}^{ij} = c_{rq} + \sum_{i \in S} (c_{iq} + d_{qq}^{ij}).$$

Concluimos que o custo de $\mathcal{M}$ é

$$c(H) + c(S') + d(Q) = c_{rq} + \sum_{a_k \in \mathcal{A} | a_k \neq 0} (\Phi(\mathcal{T}_I^r(a_k), \mathcal{T}_M^q(k)) + d_{kq}^{a_k, \text{pai}[a_k]}) + \sum_{i \in S} (c_{iq} + d_{qq}^{i, \text{pai}[i]}).$$

■

Dadas duas árvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$, pelo lema 5.1.9, para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, precisamos apenas saber qual é o vértice de menor nível associado a k , para todo $k \in \delta_{\mathcal{T}_M^q}^+(q)$ (se houver algum vértice associado a k). Estamos assumindo que o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$.

Todos os $|V(\mathcal{T}_I^r)| - 1$ vértices de $V(\mathcal{T}_I^r) \setminus \{r\}$ são candidatos a serem associados a cada um dos $w = |\delta_{\mathcal{T}_M^q}^+(q)|$ clusters de $\delta_{\mathcal{T}_M^q}^+(q)$ (se q é a raiz de G_M , então $w = g(q)$, caso contrário $w = g(q) - 1$). Considerando que existem mapeamentos viáveis nos quais nenhum vértice de $V(\mathcal{T}_I^r)$ está associado a um ou mais clusters de $\delta_{\mathcal{T}_M^q}^+(q)$, temos $n = |V(\mathcal{T}_I^r)|$ possibilidades diferentes para cada cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ ($\mathcal{A}[k]$ contém um dos $|V(\mathcal{T}_I^r)| - 1$ vértices candidatos ou o elemento zero). Portanto, temos $O(n^w)$ configurações possíveis (nem todas representam um mapeamento viável). A idéia que utilizaremos para encontrar um mapeamento ótimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que a raiz r de $\mathcal{T}_I^r$ deve ser associada à raiz q de $\mathcal{T}_M^q$, é enumerar todas estas $O(n^w)$ possibilidades. Se o grau máximo dos vértices de V_M for limitado por uma constante, teremos uma quantidade polinomial de possibilidades a serem enumeradas. Antes de mostrar o algoritmo que faz esta enumeração, mostraremos dois algoritmos que serão úteis na manipulação de vetores de associações.

Dados um vetor $\mathcal{A}$ de $w = |\delta_{\mathcal{T}_M^q}^+(q)|$ posições e um inteiro k , tal que $1 \leq k \leq w$ e $\mathcal{A}[1, 2, \dots, k-1]$ é um vetor de associações válido, o algoritmo que chamaremos de MANTÉM-VALIDADE verifica se $\mathcal{A}[1, 2, \dots, k]$ é um vetor de associações válido. Assumimos que $0 \leq a_k \leq n$, sendo que $n = |V(\mathcal{T}_I^r)| - 1$.

MANTÉM-VALIDADE($r, k, \mathcal{A}$)

```

1  se  $a_k = 0$ 
2      então devolva VERDADEIRO
3      senão para  $l \leftarrow 1$  até  $k - 1$ 
4          faça se  $a_l \neq 0$ 
5              então se MESMO-CAMINHO( $a_k, a_l, r$ )
6                  então devolva FALSO
7              senão se MESMO-CAMINHO( $a_l, a_k, r$ )
8                  então devolva FALSO
9  devolva VERDADEIRO

```

Lema 5.1.10 *O algoritmo MANTÉM-VALIDADE está correto, ou seja, devolve VERDADEIRO se e somente se $\mathcal{A}[1, 2, \dots, k]$ é um vetor de associações válido.*

Prova. Conforme especificamos na lista de parâmetros do algoritmo, vale que $\mathcal{A}[1, 2, \dots, k-1]$ é um vetor de associações válido. Logo, para que $\mathcal{A}[1, 2, \dots, k]$ seja válido, deve valer que: (i) nenhum vértice de $\mathcal{A}[1, 2, \dots, k-1]$ é igual a a_k ; (ii) nenhum vértice de $\mathcal{A}[1, 2, \dots, k-1]$ é descendente a_k ; (iii) nenhum vértice de $\mathcal{A}[1, 2, \dots, k-1]$ é ancestral de a_k .

Claramente, se $a_k = 0$, então $\mathcal{A}[1, 2, \dots, k]$ é válido. Logo, para este caso o algoritmo está correto. Suponha que $a_k \neq 0$ e considere o seguinte invariante do laço das linhas 3 a 8: *as propriedades (i), (ii) e (iii) são válidas para $\mathcal{A}[1, 2, \dots, l-1]$ e a_k .*

Inicialização

Imediatamente antes da primeira iteração, temos que $l = 1$. Neste caso, o intervalo $\mathcal{A}[1, 2, \dots, l-1]$ é vazio e, portanto, vale o invariante.

Manutenção

Suponha que imediatamente antes de uma iteração em que $1 \leq l \leq k-1$ o invariante é válido. Seja $l' = l+1$ o valor de l na próxima iteração. Se $a_l = 0$, vale que $\mathcal{A}[1, 2, \dots, l'-1]$ é um vetor de associações válido. Neste caso, o invariante é válido imediatamente antes da próxima iteração. Se $a_l \neq 0$, então é necessário checar as propriedades (i), (ii) e (iii). Conforme o lema 5.1.4, a linha 6 só é executada se ou $a_k = a_l$ ou a_k for ancestral de a_l (observe que $a_k \neq r$). Analogamente, a linha 8 só é executada se ou $a_k = a_l$ ou a_l for ancestral de a_k . Suponha que haverá uma próxima iteração. Logo as linhas 6 e 8 não foram executadas na iteração atual, então as propriedades (i), (ii) e (iii) são válidas para l' e o invariante é válido imediatamente antes da próxima iteração.

Término

Se alguma das linhas 6 ou 8 for executada em alguma iteração, o algoritmo devolve FALSO, pois uma das propriedades (i), (ii) ou (iii) foi violada. Neste caso o algoritmo está correto. Caso contrário, ou seja, as linhas 6 e 8 não foram executadas em nenhuma iteração, então na última iteração $l = k$. Logo, as propriedades (i), (ii) e (iii) são válidas para $\mathcal{A}[1, 2, \dots, k-1]$ e a_k , e o algoritmo devolve VERDADEIRO. Portanto, o algoritmo está correto. ■

Lema 5.1.11 *O algoritmo MANTÉM-VALIDADE executa $O(w \cdot n)$ operações no pior caso, sendo que $n = |V(\mathcal{T}_I^r)|$ e w corresponde ao tamanho do vetor $\mathcal{A}$.*

Prova. No pior caso o laço das linhas 3 a 8 é executado $k-1 \leq w-1 = O(w)$ vezes. Um caminho mais longo em G_I contém no máximo n vértices. Logo, pelo lema 5.1.5, as linhas 5 e 7 executam $O(n)$ operações a cada vez que são executadas. Portanto, o algoritmo MANTÉM-VALIDADE executa $O(w \cdot n)$ operações no pior caso. ■

Dado um vetor de associações, calculamos seu custo através do algoritmo que chamaremos de CALCULA-CUSTO. O algoritmo CALCULA-CUSTO recebe os seguintes parâmetros: duas árvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$; um vetor de associações $\mathcal{A}[1, 2, \dots, w]$ representando um mapeamento de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q ; um inteiro w indicando o tamanho de $\mathcal{A}$.

O valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já deve ter sido previamente calculado, para todo $i \in \mathcal{T}_I^r \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$. O algoritmo CALCULA-CUSTO devolve o custo do mapeamento $\mathcal{M}$, representado através de $\mathcal{A}$.

CALCULA-CUSTO($\mathcal{T}_I^r, \mathcal{T}_M^q, \mathcal{A}, w$)

1 devolva CALCULA-CUSTO-REC($\mathcal{T}_I^r, \mathcal{T}_M^q, r, \mathcal{A}, w$)

```

CALCULA-CUSTO-REC( $\mathcal{T}_I^r, \mathcal{T}_M^q, i, \mathcal{A}, w$ )
1   $j \leftarrow \text{pai}[i]$ 
2  para  $k \leftarrow 1$  até  $w$ 
3      faça se  $a_k = i$ 
4          então devolva  $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k)) + d_{kq}^{ij}$ 
5   $\alpha \leftarrow c_{iq}$ 
6  se  $i \neq r$ 
7      então  $\alpha \leftarrow \alpha + d_{qq}^{ij}$ 
8  para cada  $i' \in \delta_{\mathcal{T}_I^r}^+(i)$ 
9      faça  $\alpha = \alpha + \text{CALCULA-CUSTO-REC}(\mathcal{T}_I^r, \mathcal{T}_M^q, i', \mathcal{A}, w)$ 
10 devolva  $\alpha$ 

```

O algoritmo CALCULA-CUSTO-REC devolve a soma dos custos das associações de vértices e arestas de $\mathcal{T}_I^r(i)$ para $\mathcal{T}_M^q$, considerando o mapeamento $\mathcal{M}$ representado através de $\mathcal{A}$. Caso $i \neq r$ (somente nas chamadas recursivas), o custo da associação da aresta $\{i, \text{pai}[i]\}$ para seu respectivo par de clusters, considerando $\mathcal{M}$, também é computado. Neste caso, deve valer que o vértice $\text{pai}[i]$ deve estar associado a q em $\mathcal{M}$.

Baseando-se no lema 5.1.9, conclui-se que é possível calcular o custo de $\mathcal{M}$ da seguinte maneira: percorre-se a árvore $\mathcal{T}_I^r$ do menor para o maior nível (da raiz até as folhas). A cada vértice i encontrado no percurso, verificamos se i está associado a algum cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, tal que $\mathcal{T}_M^q$. Em caso afirmativo, sabemos que i é o vértice de menor nível associado a k . Logo, não precisamos percorrer a subárvore enraizada em i , pois sabemos que todos os vértices descendentes de i estarão associados a clusters da subárvore enraizada em k . Neste caso, basta utilizar o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, que estamos assumindo que já foi calculado. Caso o vértice i não esteja associado a nenhum cluster de $\delta_{\mathcal{T}_M^q}^+(q)$, então i só pode estar associado a q . Para verificar esta afirmação, basta observar que se i estivesse associado a algum cluster $l \notin \delta_{\mathcal{T}_M^q}^+(q) \cup q$, então, pelo lema 5.1.6, existiria um ancestral de l em $\delta_{\mathcal{T}_M^q}^+(q)$ para o qual algum ancestral de i está associado. Mas, se tal situação tivesse ocorrido, então o vértice i não teria sido alcançado no percurso.

O algoritmo CALCULA-CUSTO-REC funciona exatamente como explicado no parágrafo anterior, sendo que o “percurso” mencionado é feito recursivamente. Portanto, a prova da correção do algoritmo CALCULA-CUSTO-REC, e conseqüentemente do algoritmo CALCULA-CUSTO, pode ser derivada do lema 5.1.9.

Lema 5.1.12 *O algoritmo CALCULA-CUSTO-REC executa $O(n \cdot w)$ operações, sendo que $n = |V(\mathcal{T}_I^r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$.*

Prova. Considere a seguinte recorrência:

$$F(n, w) \leq \begin{cases} 0, & \text{se } n = 0 \\ F(n - 1, w) + w, & \text{se } n > 0 \end{cases}$$

Quando $n > 0$, o laço das linhas 2 a 4 executa no máximo w operações. Caso a linha 4 não seja executada, o laço das linhas 8 e 9 executa $\sum_{i' \in \delta_{G_I}(i) \setminus \{j\}} F(H_{i'}, w)$ operações, sendo que $H_{i'}$ corresponde à quantidade de vértices da subárvore enraizada em i' . Por isso, quando $n > 0$, temos que $F(n, w) \leq \sum_{i' \in \delta_{G_I}(i) \setminus \{j\}} F(H_{i'}, w) + w$. Note que as subárvores em questão são disjuntas. Logo, se mostrarmos que $F(n, w)$ é linear, teremos que $\sum_{i' \in \delta_{G_I}(i) \setminus \{j\}} F(H_{i'}, w) = F(n - 1, w)$. De fato, mostraremos a seguir que $F(n, w)$ é linear e concluiremos que $F(n, w)$ é um limitante superior para a quantidade de operações executadas pelo algoritmo CALCULA-CUSTO-REC.

Mostraremos, por indução em n , que $F(n, w) = nw$ (observe que w é uma constante dentro da recorrência). A base é quando $n = 0$. Neste caso $F(0, w) = w \cdot 0 = 0$. Suponha que $n > 0$. Então $F(n, w) \leq F(n - 1, w) + w$. Por hipótese de indução, temos que $F(n, w) \leq (n - 1)w + w = nw - w + w = nw$.

Portanto, o algoritmo CALCULA-CUSTO-REC executa $O(n \cdot w)$ operações. ■

Corolário 5.1.4 *O algoritmo CALCULA-CUSTO executa $O(n \cdot w)$ operações, sendo que n corresponde à quantidade de vértices contidos na subárvore enraizada em r , e w corresponde à quantidade de filhos de q .*

Agora que já definimos como representaremos um mapeamento, estamos prontos para mostrar o algoritmo COMBINA-SOLUÇÕES, que enumera todas as possíveis combinações de um vetor de associações para determinar o custo de um mapeamento ótimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q .

O algoritmo “combina-soluções”

O problema que estamos interessados em resolver é o seguinte:

Dadas duas subárvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$, o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, calcular o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

Conforme vimos anteriormente, um algoritmo chamado COMBINA-SOLUÇÕES foi utilizado como “caixa-preta” na definição da recorrência para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. A seguir, exibimos este algoritmo.

O algoritmo COMBINA-SOLUÇÕES recebe como parâmetros duas árvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$. Após enumerar todas as configurações possíveis para um vetor de associações que representa um mapeamento de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q , o algoritmo devolve o custo de um mapeamento ótimo. Assumimos que $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ foi previamente calculado para todo i e k , tais que $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$.

COMBINA-SOLUÇÕES($\mathcal{T}_I^r, \mathcal{T}_M^q$)

- 1 $n \leftarrow |V(\mathcal{T}_I^r)| - 1$
- 2 $w \leftarrow |\delta_{\mathcal{T}_M^q}^+(q)|$
- 3 $\alpha^* \leftarrow \infty$
- 4 COMBINA-SOLUÇÕES-REC($\mathcal{T}_I^r, \mathcal{T}_M^q, 1, n, w, \mathcal{A}, \alpha^*$)
- 5 devolva α^*

COMBINA-SOLUÇÕES-REC($\mathcal{T}_I^r, \mathcal{T}_M^q, k, n, w, \mathcal{A}, \alpha^*$)

- 1 **para** $t \leftarrow 0$ até n
- 2 **faça** $a_k \leftarrow t$
- 3 **se** MANTÉM-VALIDADE($r, k, \mathcal{A}$) = VERDADEIRO
- 4 **então se** $k = w \triangleright$ completou o vetor
- 5 **então** $\alpha \leftarrow$ CALCULA-CUSTO($\mathcal{T}_I^r, \mathcal{T}_M^q, \mathcal{A}, w$)
- 6 **se** $\alpha < \alpha^*$
- 7 **então** $\alpha^* \leftarrow \alpha$
- 8 **senão** COMBINA-SOLUÇÕES-REC($\mathcal{T}_I^r, \mathcal{T}_M^q, k + 1, n, w, \mathcal{A}, \alpha^*$)

O parâmetro α^* , que é passado ao COMBINA-SOLUÇÕES-REC, corresponde ao custo de uma melhor solução dentre as enumeradas. Ou seja, após o término da execução do algoritmo COMBINA-SOLUÇÕES, o valor de α^* corresponde ao custo de uma melhor solução dentre todas as soluções viáveis que têm r associado a q . Do ponto de vista de implementação, o parâmetro α^* é como se fosse um ponteiro (ou referência) para um valor. Inicialmente, o conteúdo apontado por α^* é ∞ , pois nenhuma solução viável foi calculada até o momento.

O algoritmo COMBINA-SOLUÇÕES-REC recebe os seguintes parâmetros: duas árvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$; um inteiro w que corresponde ao tamanho de $\mathcal{A}$; um inteiro n que corresponde ao maior rótulo que pode ser atribuído a uma posição qualquer de $\mathcal{A}$; um inteiro k que corresponde a uma posição dentro do vetor $\mathcal{A}$; um vetor de associações $\mathcal{A}$; um ponteiro (ou referência) para um número $\alpha^* \in \mathbb{R}_+$.

O algoritmo COMBINA-SOLUÇÕES-REC guarda na variável α^* o custo de uma melhor solução dentre as que possuem o prefixo $\mathcal{A}[1, 2, \dots, k-1]$. Como $k = 1$ na primeira chamada feita a COMBINA-SOLUÇÕES-REC, ou seja, o segmento $\mathcal{A}[1, 2, \dots, k-1]$ é vazio, temos que o algoritmo COMBINA-SOLUÇÕES devolve o custo de uma melhor solução dentre todas as viáveis.

Pelo lema 5.1.9, sabemos que é possível representar de forma única um mapeamento através de um vetor de associações válido. O procedimento de combinar as soluções é feito através da enumeração de todas as possíveis configurações de um vetor de associações. Portanto, a prova da correção do algoritmo COMBINA-SOLUÇÕES-REC, e consequentemente do algoritmo COMBINA-SOLUÇÕES, pode ser derivada do lema 5.1.9.

Lema 5.1.13 *O algoritmo COMBINA-SOLUÇÕES-REC executa $O(w \cdot n^{s+1})$ operações, sendo que $s = w - k + 1$ (note que n , w e k são parâmetros do algoritmo).*

Prova. As linhas 1 e 2 executam $\Theta(n)$ operações. Pelo lema 5.1.11, a cada iteração a linha 3 executa $O(n \cdot w)$ operações. Como a linha 3 é executada $\Theta(n)$ vezes, o total de operações executadas é $O(n^2 \cdot w)$. A linha 4 executa $O(n)$ operações. Em uma chamada a COMBINA-SOLUÇÕES-REC, temos uma das duas seguintes situações: (1) quando $k = w$, vale que $s = 1$; (2) quando $k \neq w$, vale que $1 \leq k < w$ (por definição) e consequentemente $s > 1$.

No caso (1), as linhas 5 a 7 são executadas e a linha 8 não é executada. A linha 5 é executada $O(n)$ vezes. Pelo corolário 5.1.4, a cada execução da linha 5 são executadas $O(n \cdot w)$ operações. Logo, a linha 5 executa $O(n^2 \cdot w)$ operações. As linhas 6 e 7 são executadas $O(n)$ vezes. Somando as operações executadas em todas as linhas, temos que o algoritmo COMBINA-SOLUÇÕES-REC executa $O(n^2 \cdot w)$ operações, quando $s = 1$.

No caso (2), a linha 8 é executada $O(n)$ vezes e as linhas 5 a 7 não são executadas. A cada execução da linha 8 é feita uma chamada recursiva na qual $s' = s - 1$ (pois o parâmetro k é acrescido em uma unidade), sendo que s' é o valor de s nas respectivas chamadas recursivas. Somando as operações executadas em todas as linhas, quando $s > 1$, temos que o algoritmo COMBINA-SOLUÇÕES-REC

executa $O(n^2 \cdot w) + O(n\gamma)$ operações, sendo que γ é a quantidade de operações executadas pela chamada recursiva feita na linha 8.

De acordo com a análise dos casos (1) e (2), podemos limitar a quantidade de operações executadas pelo algoritmo COMBINA-SOLUÇÕES-REC através da seguinte recorrência.

$$F(s) = \begin{cases} O(n^2 \cdot w), & \text{se } s = 1 \\ n \cdot [F(s-1) + O(n \cdot w)], & \text{se } s > 1. \end{cases}$$

Mostraremos, por indução em s , que $F(s) = O(w \cdot n^{s+1})$. O caso base é quando $s = 1$. Neste caso, temos que $F(1) = O(w \cdot n^2) = O(w \cdot n^{s+1})$. Suponha que $s > 1$. Neste caso, temos que $F(s) = n \cdot [F(s-1) + O(nw)]$. Por hipótese de indução, vale que

$$\begin{aligned} n \cdot [F(s-1) + O(nw)] &= n \cdot [O(w \cdot n^s) + O(nw)] \\ &= O(w \cdot n^{s+1} + n^2 \cdot w). \end{aligned}$$

Como $s > 1$, vale que

$$O(w \cdot n^{s+1} + n^2 \cdot w) = O(w \cdot n^{s+1}).$$

Portanto, o algoritmo COMBINA-SOLUÇÕES-REC executa $O(w \cdot n^{s+1})$ operações. ■

Corolário 5.1.5 *O algoritmo COMBINA-SOLUÇÕES executa $O(w \cdot n^{w+1})$ operações, sendo que $n = |V(\mathcal{T}_I^r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$.*

Um algoritmo para o cálculo de Φ

Conforme definido anteriormente, o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ corresponde ao custo de um mapeamento ótimo de $\mathcal{T}_I^r \subseteq G_I$ para $\mathcal{T}_M^q \subseteq G_M$, sujeito à restrição de que r deve ser associado a q . No início

desta seção, apresentamos a seguinte recorrência para calcular o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$:

$$\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = \begin{cases} c_{rq}, & \text{se } r \text{ é folha.} \\ c_{rq} + \sum_{i \in V(\mathcal{T}_I^r) \setminus \{r\}} (c_{iq} + d_{qq}^{ij}), & \text{se } q \text{ é folha (sendo que } j = \text{pai}[i]). \\ \min\{c^*(\mathcal{M}) \mid \mathcal{M} \text{ é um } (r, q)\text{-mapeamento}\}, & \text{caso contrário.} \end{cases}$$

O algoritmo que chamaremos de $\text{CALCULA}\Phi$ recebe os seguintes parâmetros: duas árvores enraizadas G_I^r e G_M^q ; um vértice $r' \in V_I$; um cluster $q' \in V_M$; uma matriz $\mathcal{M}$ bidimensional (a primeira dimensão será indexada pelos vértices de V_I , e a segunda dimensão será indexada pelas arestas de A_M), que será utilizada como uma tabela de memoização.

O algoritmo calcula e memoiza o valor de $\Phi(G_I^r(r'), G_M^q(q'))$. A memoização é feita de forma que, para $i \in V_I$, $k \in V_M$ e $l = \text{pai}[k]$ (note que o pai de k depende da raiz de G_M , que é modificada durante a execução), a entrada $\mathcal{M}[i][kl]$ contém o valor de $\Phi(G_I^r(i), G_M^q(k))$, após tal valor ter sido calculado. Quando $\mathcal{M}[i][kl] = \infty$, significa que o valor de $\Phi(G_I^r(i), G_M^q(k))$ ainda não foi calculado.

Sejam $p \in V_M$ e $q \in V_M$, tais que $p \neq q$. Vimos no início desta seção que $G_M^q(k) = G_M^p(k)$ se e somente se os pais de k são iguais em G_M^q e G_M^p . Por isso, ao invés de guardar quem é a raiz de G_M , guardamos somente a aresta que liga k a $\text{pai}[k]$. No contexto onde este algoritmo será utilizado, será considerada uma única raiz para a árvore G_I . Por isso, não precisamos nos preocupar com quem é o pai de i .

Para calcular $\min\{c^*(\mathcal{M}) \mid \mathcal{M} \text{ é um } (r, q)\text{-mapeamento}\}$, o algoritmo $\text{CALCULA}\Phi$ calculará o valor de $\Phi(G_I^r(i), G_M^q(k))$ para todo $i \in V(G_I^r(r')) \setminus \{r'\}$ e $k \in \delta_{G_M^q}^+(q')$. Após calcular estes valores, basta chamar o algoritmo $\text{COMBINA-SOLUÇÕES}(G_I^r(r'), G_M^q(q'))$ para obter um (r, q) -mapeamento ótimo.

```

CALCULA $\Phi$ ( $G_I^r, r', G_M^q, q', \mathcal{M}$ )
1   $l \leftarrow \text{pai}[q']$ 
2  se  $\mathcal{M}[r'][lq'] \neq \infty$ 
3      então  $\alpha^* \leftarrow \mathcal{M}[r'][lq']$ 
4  senão se  $\text{folha}[r']$ 
5      então  $\alpha^* \leftarrow c_{r'q'}$ 
6  senão se  $\text{folha}[q']$ 
7      então  $\alpha^* \leftarrow c_{r'q'}$ 
8      para cada  $i \in \delta_{G_I^r}^+(r')$ 
9          faça  $j \leftarrow \text{pai}[i]$ 
10              $\alpha^* \leftarrow \alpha^* + c_{iq'} + d_{q'q'}^{ij}$ 
11  senão para cada  $i \in \delta_{G_I^r}^+(r')$ 
12      faça para cada  $k \in \delta_{G_M^q}^+(q')$ 
13          faça CALCULA $\Phi$ ( $G_I^r, i, G_M^q, k, \mathcal{M}$ )
14       $\alpha^* \leftarrow \text{COMBINA-SOLUÇÕES}(G_I^r(r'), G_M^q(q'))$ 
15   $\mathcal{M}[r'][lq'] \leftarrow \alpha^*$ 
16  devolva  $\alpha^*$ 

```

O algoritmo CALCULA Φ funciona exatamente como definido na recorrência. Portanto, a verificação de sua correção pode ser derivada dos argumentos que usamos na construção da recorrência. O único detalhe que difere da definição da recorrência é o uso de memoização.

Para que possamos fazer uma análise justa da complexidade do algoritmo CALCULA Φ , levaremos em conta o contexto onde ele é utilizado. Por isso, a análise do algoritmo CALCULA Φ ficará “embutida” na análise do próximo algoritmo. Intuitivamente, percebe-se que nas primeiras chamadas, nas quais a tabela de memoização ainda está esparsa, o algoritmo precisará trabalhar mais. Quando a tabela de memoização estiver mais densa, o algoritmo tende a trabalhar menos.

O algoritmo “PCCA-para-árvores”

Por fim, apresentamos o algoritmo PCCA-PARA-ÁRVORES, que recebe duas árvores G_I e G_M , e devolve o custo de um mapeamento ótimo de G_I para G_M .

PCCA-PARA-ÁRVORES(G_I, G_M)

```

1  para todo  $i \in V_I$ 
2      faça para todo  $kl \in A_M$ 
3          faça  $\mathcal{M}[i][kl] \leftarrow \infty$ 
4  Seja  $r \in V_I$ 
5  MONTA-ÁRVORE( $G_I, r$ )
6   $menorCusto \leftarrow \infty$ 
7  para todo  $q \in V_M$ 
8      faça MONTA-ÁRVORE( $G_M, q$ )
9          CALCULA $\Phi(G_I^r, G_M^q, \mathcal{M})$ 
10          $l' \leftarrow pai[q]$ 
11         se  $\mathcal{M}[r][ql'] < menorCusto$ 
12             então  $menorCusto \leftarrow \mathcal{M}[r][ql']$ 
13  devolva  $menorCusto$ 

```

Proposição 5.1.1 *O algoritmo PCCA-PARA-ÁRVORES está correto, ou seja, devolve o custo de um mapeamento ótimo de G_I para G_M , sendo que G_I e G_M são árvores.*

Prova. Assuma que os clusters de V_M possuem rótulos $1, 2, \dots, |V_M|$ e são percorridos em ordem crescente, no laço das linhas 7 a 12 (o laço pára quando $q = |V_M| + 1$). Considere o seguinte invariante do laço das linhas 7 a 12:

*A variável **menorCusto** contém o custo de um mapeamento de custo mínimo de G_I para G_M dentre todos os mapeamentos viáveis nos quais o vértice r está associado a algum cluster $l \in V_M$, tal que $1 \leq l \leq q - 1$.*

Inicialização

Imediatamente antes de entrar a primeira vez no laço da linha 7 (quando $q = 1$), vale que $menorCusto = \infty$. Como não existe cluster l , tal que $1 \leq l \leq q - 1$, então não existe nenhum mapeamento viável no qual o vértice r está associado a algum cluster $l \in V_M$, tal que $1 \leq l \leq q - 1$. Portanto, o invariante é válido na inicialização.

Manutenção

Suponha que o invariante é válido imediatamente antes de uma iteração em que $1 \leq q - 1 < |V_M|$. Seja $q' = q + 1$ o valor de q na próxima iteração. Após a execução da linha 9, vale que $\mathcal{M}[r][ql']$

contém o custo de um mapeamento de custo mínimo no qual r está associado a q . Após a execução das linhas 11 e 12, vale que a variável *menorCusto* contém o custo de um mapeamento de custo mínimo de G_I para G_M dentre todos os mapeamentos viáveis nos quais o vértice r está associado a algum cluster $l \in V_M$, tal que $1 \leq l \leq q' - 1$. Ou seja, o invariante é válido imediatamente antes da próxima iteração.

Término

Quando o laço termina, vale que $q = |V_M| + 1$. Pelo invariante do laço, vale que a variável *menorCusto* contém o custo de um mapeamento de custo mínimo de G_I para G_M dentre todos os mapeamentos viáveis nos quais o vértice r está associado a algum cluster $l \in V_M$, tal que $1 \leq l \leq |V_M|$. Logo, na linha 13 é devolvido o valor correto e, portanto, o algoritmo está correto. ■

Proposição 5.1.2 *O algoritmo PCCA-PARA-ÁRVORES executa $O(w \cdot m \cdot n^{w+2})$ operações, sendo que $n = |V_I|$, $m = |V_M|$ e $w = \max\{g(k) \mid k \in V_M\}$.*

Prova. Na tabela abaixo, mostramos a quantidade de operações executadas por cada linha do algoritmo:

Linha(s)	Operações	Obs.
1 a 3	$\Theta(n \cdot m)$	(i)
4, 6, 10 e 13	$\Theta(1)$	
5	$\Theta(n)$	(ii)
7, 11 e 12	$O(m)$	
8	$\Theta(m^2)$	(iii)
9	$O(w \cdot m \cdot n^{w+2})$	(iv)

sendo que: (i) o laço é repetido $|A_M|$ vezes, e como G_M é uma árvore, então $|A_M| = O(m)$; (ii) pelo corolário 5.1.3; (iii) pelo corolário 5.1.3, são consumidas $O(m)$ operações cada vez que a linha 8 é executada; (iv) para provar esta afirmação, usaremos o fato de que a tabela de memoização possui $O(nm)$ posições (a primeira dimensão é indexada pelos vértices de V_I e a segunda dimensão é indexada pelas arestas de A_M), e que para preencher cada uma destas posições executa-se $O(w \cdot n^{w+1})$ operações.

Claramente, cada posição da tabela é preenchida no máximo uma única vez. No pior caso, para preencher cada uma das $O(nm)$ posições da tabela de memoização, é necessário executar uma vez o

algoritmo COMBINA-SOLUÇÕES, que pelo corolário 5.1.5 executa $O(w \cdot n^{w+1})$ operações. Concluimos que a linha 9 executa $O(w \cdot m \cdot n^{w+2})$ operações. Portanto, o algoritmo PCCA-PARA-ÁRVORES executa $O(w \cdot m \cdot n^{w+2})$ operações. ■

Supondo que w é limitado por uma constante, o algoritmo PCCA-PARA-ÁRVORES é polinomial, sendo que w corresponde ao grau máximo dos vértices de V_M .

5.2 Um Caso Polinomial

Nesta seção, apresentamos um algoritmo polinomial para um caso especial do PCCA *para árvores*. Será imposta uma estrutura específica para uma das árvores de entrada.

Estrutura das árvores de entrada

Dizemos que uma árvore é uma *estrela* se todos os seus vértices, exceto um único, possuem grau um. Em outras palavras, um único vértice da árvore, digamos u , é adjacente a todos os outros, e todos os vértices (exceto u) são adjacentes somente a u . Dizemos que u é o *núcleo* da estrela.

Figura 5.7: Uma estrela. O vértice pintado de preto é o núcleo da estrela.

Dizemos que uma árvore G é uma *estrela estendida*² se G possui um subgrafo H , tal que H é uma estrela cujo núcleo é um vértice u , e o subgrafo de G induzido por $V(G) \setminus \{u\}$ é composto por $g(u)$ componentes, sendo que cada uma delas é um caminho (uma componente com um único vértice também é um caminho). Dizemos que u é o *núcleo* da estrela estendida (ou núcleo de G), e todos os vértices com grau igual a dois são *internos* (supondo que $g(u) > 2$). Quando G é um caminho com pelo menos três arestas, podemos tomar qualquer vértice de grau dois como sendo o núcleo e os demais vértices de grau dois como sendo internos. Os vértices de grau um são chamados de *pontas*. Repare que uma estrela também é uma estrela estendida.

²Esta classe de grafo é também conhecida como *spider graph* (grafo aranha).

Figura 5.8: Uma estrela estendida. O vértice pintado de preto é o núcleo da estrela estendida. O subgrafo H de G , que é uma estrela, está mais escuro.

Ao longo desta seção assumiremos que o grafo $G_I = (V_I, A_I)$ é uma estrela estendida e que o grafo $G_M = (V_M, A_M)$ é uma árvore qualquer (o grau máximo dos vértices de V_M não é limitado por nenhuma constante).

Modificação do algoritmo “combina-soluções”

Conforme visto na seção anterior, o algoritmo COMBINA-SOLUÇÕES resolve o seguinte problema:

Dadas duas subárvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$, o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, calcular o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

A idéia que será apresentada nesta seção é utilizar o algoritmo PCCA-PARA-ÁRVORES exatamente como mostrado na seção anterior. Porém, o passo em que a subrotina CALCULA Φ chama o algoritmo COMBINA-SOLUÇÕES será modificado. Ao invés de utilizar o algoritmo COMBINA-SOLUÇÕES, será utilizado o algoritmo que chamaremos de COMBINA-SOLUÇÕES-EMP, que tirará proveito do fato de G_I ser uma estrela estendida. Em um determinado passo, o algoritmo COMBINA-SOLUÇÕES-EMP precisa resolver o problema de encontrar um emparelhamento máximo de custo mínimo, por isso o sufixo “emp” em seu nome.

Lembramos que a complexidade do algoritmo COMBINA-SOLUÇÕES, mostrado na seção anterior, é $O(w \cdot n^{w+1})$, sendo que $n = |V(\mathcal{T}_I^r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$. Analisando a prova da complexidade do algoritmo PCCA-PARA-ÁRVORES (proposição 5.1.2), conclui-se que se o algoritmo COMBINA-SOLUÇÕES-EMP

for polinomial, então o algoritmo PCCA-PARA-ÁRVORES também será.

Definições preliminares

Duas arestas de um grafo são adjacentes se têm um extremo em comum. Um emparelhamento num grafo é um conjunto de arestas duas a duas não-adjacentes. Seja $G = (V, A)$ um grafo. Um emparelhamento $E \subseteq A$ é *máximo* se $|E| \geq |F|$, para todo emparelhamento $F \subseteq A$. Um emparelhamento $E \subseteq A$ é *perfeito* se cada vértice de V é extremo de alguma aresta de E . Dizemos que um emparelhamento E cobre um subconjunto S de vértices se, para cada $v \in S$, existe alguma aresta de E que incide em v . Suponha que existe um custo $\lambda_a \in \mathbb{R}$ associado a cada aresta $a \in A$. Então, o custo de $E \subseteq A$ é dado por $\lambda(E) = \sum_{e \in E} \lambda_e$. Um emparelhamento máximo $E \subseteq A$ tem custo mínimo se nenhum outro emparelhamento máximo $F \subseteq A$ possui custo menor que o seu.

Um grafo é *bipartido* se existe uma bipartição do seu conjunto de vértices, tal que cada aresta tem um extremo em uma das partes da bipartição e o outro extremo na outra parte. Seja $B = (V_B, A_B)$ um grafo bipartido. Sejam X e Y as duas partes de V_B . Dizemos que B é *bipartido completo* se para cada par de vértices u, v , tal que $u \in X$ e $v \in Y$, existe uma aresta $uv \in A_B$. Suponha que $|X| < |Y|$. Neste caso, B não possui um emparelhamento perfeito, pois um emparelhamento máximo $E \subseteq A_B$ possui no máximo $|X|$ arestas. Ou seja, uma condição necessária para que B tenha um emparelhamento perfeito é $|X| = |Y|$.

O algoritmo conhecido como *método húngaro* [32] encontra um emparelhamento perfeito de custo mínimo em um grafo bipartido completo $B = (V_B, A_B)$. Seu consumo de tempo é $O(n^3)$, sendo que $n = |V_B|$.

Estaremos interessados em encontrar um emparelhamento **máximo** de custo mínimo em um grafo bipartido $B = (V_B, A_B)$. Conforme mencionamos, o método húngaro encontra um emparelhamento **perfeito** de custo mínimo em um grafo bipartido **completo**. Ou seja, precisamos “completar” o grafo B , de forma que ele contenha um emparelhamento perfeito “equivalente” ao emparelhamento máximo que estamos interessados em encontrar.

Sejam X e Y , tais que $X \cup Y = V_B$ e $X \cap Y = \emptyset$. Suponha que $|X| \neq |Y|$ e, sem perda de generalidade, $|X| < |Y|$. Neste caso, o primeiro passo para “completar” o grafo B é adicionar $|Y| - |X|$ vértices, e adicionar arestas que ligam cada um deles a todos os vértices de Y . Tais arestas devem ter custo zero. Caso $|X| = |Y|$, não precisamos adicionar nenhum vértice.

Agora que $|X| = |Y|$, o próximo passo é verificar se falta inserir alguma aresta em A_B para que B seja bipartido completo. Suponha que alguns vértices de X não sejam adjacentes a todos os vértices de Y . Então, adicionamos as arestas que restam para que B seja um grafo bipartido completo. Tais arestas devem ter custo infinito.

Figura 5.9: À esquerda temos um grafo B , tal que $|X| < |Y|$. À direita temos o grafo B acrescido de vértices e arestas para torná-lo bipartido completo. Os vértices pontilhados foram inseridos para que as duas partes de V_B tenham a mesma cardinalidade. As arestas pontilhadas têm custo zero. As arestas em negrito têm custo infinito.

Sejam $B = (V_B, A_B)$ e $B' = (V'_B, A'_B)$ dois grafos, tais que B é um grafo bipartido e B' corresponde à construção mencionada (adicionando arestas e vértices em B). Um emparelhamento perfeito $E' \subset A'_B$ de custo mínimo em B' corresponde a um emparelhamento máximo $E \subset A_B$ de custo mínimo em B . Basta definir E da seguinte forma: $E = E' \setminus H$, sendo que H corresponde às arestas que incidem nos vértices de $V'_B \setminus V_B$. Como as arestas de H possuem custo zero, vale que $\lambda(E) = \lambda(E')$ (claramente, E não contém nenhuma aresta de custo infinito).

Por simplicidade, assumiremos que o algoritmo que denotaremos por MÉTODO-HÚNGARO encontra um emparelhamento máximo de custo mínimo em B . Os “ajustes” que mencionamos podem ser implementados como rotinas de pré e pós-processamento dentro do algoritmo MÉTODO-HÚNGARO, sem que a sua complexidade seja alterada.

Sejam $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$ árvores enraizadas, sendo que G_I é uma estrela estendida e G_M é uma árvore. Mostraremos que é possível utilizar o algoritmo MÉTODO-HÚNGARO para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. Basicamente a estratégia que apresentaremos consiste em gerar um grafo bipartido $B = (V_B, A_B)$ a partir de $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$. Após associar os custos corretos às arestas de A_B , o custo de um emparelhamento máximo de custo mínimo em B corresponderá a um valor que será utilizado para calcular, de forma simples, o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

Como $\mathcal{T}_I^r$ é uma subárvore de G_I , o núcleo de G_I pode ou não estar contido em $\mathcal{T}_I^r$. Seja u o núcleo de G_I . Considere os seguintes casos: (1) $u \notin V(\mathcal{T}_I^r)$. Neste caso, temos que $\mathcal{T}_I^r$ é um caminho;

(2) $u \in V(\mathcal{T}_I^r)$ e $r = u$; (3) $u \in V(\mathcal{T}_I^r)$ e r é ancestral de u em $\mathcal{T}_I^r$.

Classificaremos a subárvore $\mathcal{T}_I^r$ através de três rótulos diferentes, sendo eles: IMEDIATO, nos casos (1) e (2) (ver figura 5.10); SEMI-IMEDIATO, no caso (3), mas somente quando r tem apenas um filho em $\mathcal{T}_I^r$ (neste caso, o vértice r é ou interno ou ponta em G_I ; ver figura 5.11); DIVISÍVEL, no caso (3), mas somente quando r tem dois filhos em $\mathcal{T}_I^r$ (neste caso, r é um vértice interno em G_I ; ver figura 5.11).

Figura 5.10: À esquerda o caso (1). À direita o caso (2). Repare que no caso (2), a raiz r pode ou não ter um pai fora de $\mathcal{T}_I^r$. As duas subárvores são classificadas com o rótulo IMEDIATO.

Figura 5.11: A figura ilustra as duas possibilidades do caso (3). À esquerda a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO (r só possui um filho em $\mathcal{T}_I^r$), e à direita com o rótulo DIVISÍVEL (r possui dois filhos em $\mathcal{T}_I^r$).

Os nomes dos rótulos estão relacionados com o funcionamento do algoritmo que apresentaremos a diante. Quando $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO, o algoritmo construirá um grafo bipartido e encontrará um emparelhamento máximo de custo mínimo em tal grafo. Quando $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO, o funcionamento será igual ao do caso anterior, porém, o

algoritmo considerará algumas soluções separadamente. O último caso ocorre quando $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL. Neste caso, o algoritmo construirá dois grafos bipartidos e encontrará um emparelhamento máximo de custo mínimo em cada um deles.

Antes de mostrar os algoritmos para a construção dos grafos bipartidos, mostraremos um algoritmo para classificar a subárvore $\mathcal{T}_I^r$. O algoritmo que chamaremos de CLASSIFICA-SUBÁRVORE recebe como parâmetro uma árvore $\mathcal{T}_I^r \subseteq G_I$, tal que G_I é uma estrela estendida. Assumiremos que $\mathcal{T}_I^r$ possui pelo menos dois vértices (no contexto em que o algoritmo será utilizado, esta condição é sempre satisfeita).

O algoritmo devolve um rótulo indicando qual a classificação de $\mathcal{T}_I^r$ (IMEDIATO, SEMI-IMEDIATO ou DIVISÍVEL). No último caso, além do rótulo DIVISÍVEL, o algoritmo devolve três vértices s_1 , s_2 e s_3 (ver figura 5.12), tais que $s_1, s_2 \in \delta_{\mathcal{T}_I^r}^+(r)$, a subárvore $\mathcal{T}_I^r(s_1)$ é um caminho e s_3 é o núcleo de G_I (conseqüentemente, ou $s_2 = s_3$ ou s_2 é ancestral de s_3 em $\mathcal{T}_I^r$).

Figura 5.12: Além do rótulo DIVISÍVEL, o algoritmo devolve os três vértices s_1 , s_2 e s_3 .

CLASSIFICA-SUBÁRVORE($\mathcal{T}_I^r$)

```

1  se  $|\delta_{\mathcal{T}_I^r}^+(r)| > 2$ 
2    então devolva IMEDIATO
3  senão se  $|\delta_{\mathcal{T}_I^r}^+(r)| = 1$ 
4    então Seja  $i$  o único filho de  $r$  em  $\mathcal{T}_I^r$ 
5        enquanto  $|\delta_{\mathcal{T}_I^r}^+(i)| = 1$ 
6            faça  $i \leftarrow$  o único filho de  $i$  em  $\mathcal{T}_I^r$ 
7        se  $|\delta_{\mathcal{T}_I^r}^+(i)| = 0$ 
8            então devolva IMEDIATO
9        senão devolva SEMI-IMEDIATO
10  senão Sejam  $s_1$  e  $s_2$  os dois filhos de  $r \triangleright$  Assumimos que  $|\delta_{\mathcal{T}_I^r}^+(r)| \geq 1$ . Logo,  $|\delta_{\mathcal{T}_I^r}^+(r)| = 2$ .
11    para cada  $s_3 \in \delta_{\mathcal{T}_I^r}^+(r)$ 
12        faça se  $s_2 \neq s_3$ 
13            então  $s_1 \leftrightarrow s_2$ 
14            enquanto  $|\delta_{\mathcal{T}_I^r}^+(s_3)| = 1$ 
15                faça  $s_3 \leftarrow$  o único filho de  $s_3$  em  $\mathcal{T}_I^r$ 
16            se  $|\delta_{\mathcal{T}_I^r}^+(s_3)| > 1$ 
17                então devolva DIVISÍVEL,  $s_1, s_2, s_3$ 
18    devolva IMEDIATO  $\triangleright$  Esta linha só é executada se  $G_I$  é um caminho e  $r$  não é ponta.
```

Devido à simplicidade do algoritmo CLASSIFICA-SUBÁRVORE, consideramos desnecessário provar sua correção e complexidade. Assumiremos que o algoritmo CLASSIFICA-SUBÁRVORE está correto, e que sua complexidade de tempo é $O(n)$, sendo que $n = |V(\mathcal{T}_I^r)|$. Este algoritmo será utilizado no final desta seção, onde precisaremos classificar $\mathcal{T}_I^r$ com um dos rótulos que definimos. Antes disso, voltemos à construção dos grafos bipartidos.

Resolução do caso em que $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO

O algoritmo que chamaremos de MGB-IMEDIATO (MGB - Monta Grafo Bipartido) recebe como parâmetros duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO.

O algoritmo devolve um grafo bipartido $B = (V_B, A_B)$ e uma função λ que associa custos às aresta de A_B , tais que o custo de um emparelhamento máximo de custo mínimo em B corresponde a $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) - c_{rq}$.

MGB-IMEDIATO($\mathcal{T}_I^r, \mathcal{T}_M^q$)

- 1 Sejam $t = |\delta_{\mathcal{T}_I^r}^+(r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$
- 2 $\triangleright$ Assuma que os vrtices de $\delta_{\mathcal{T}_I^r}^+(r)$ possuem rtulos $1, 2, \dots, t$
- 3 $\triangleright$ Assuma que os clusters de $\delta_{\mathcal{T}_M^q}^+(q)$ possuem rtulos $1, 2, \dots, w$
- 4 $V_B \leftarrow \{u_1, u_2, \dots, u_t\} \cup \{p_1, p_2, \dots, p_t\} \cup \{v_1, v_2, \dots, v_w\}$
- 5 $A_B \leftarrow \emptyset$
- 6 **para** $i \leftarrow 1$ at t
- 7 **faa** $e \leftarrow \{u_i, p_i\}$
- 8 $A_B \leftarrow A_B \cup \{e\}$
- 9 $\lambda_e \leftarrow \sum_{j \in V(\mathcal{T}_I^r(i))} (c_{jq} + d_{qq}^{j, \text{pai}[j]})$
- 10 **para** $k \leftarrow 1$ at w
- 11 **faa** $e \leftarrow \{u_i, v_k\}$
- 12 $A_B \leftarrow A_B \cup \{e\}$
- 13 Seja $H(j)$ o conjunto dos ancestrais de j em $\mathcal{T}_I^r(i)$, $\forall j \in V(\mathcal{T}_I^r(i))$
- 14 $\lambda_e \leftarrow \min_{j \in V(\mathcal{T}_I^r(i))} [\Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})]$
- 15 Seja $B = (V_B, A_B)$
- 16 devolva B, λ

Figura 5.13: Construo feita pelo algoritmo MGB-IMEDIATO. esquerda as rvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$. direita o grafo bipartido B devolvido pelo algoritmo.

Antes de mostrar a correo do algoritmo MGB-IMEDIATO, introduziremos a noto que ser utilizada.

Suponha que temos duas subrvores $\mathcal{T}_I^r$ e $\mathcal{T}_M^q$, e um mapeamento $\mathcal{M}$ de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, tal que r est associado a q em $\mathcal{M}$. Utilizaremos a noto $\Delta(i, \mathcal{M})$ e $\beta(a, \mathcal{M})$, para todo $i \in V(\mathcal{T}_I^r)$ e $a \in A(\mathcal{T}_I^r)$, de forma que $\Delta(i, \mathcal{M})$ corresponde ao custo da associao do vrtice i a seu respectivo cluster, considerando o mapeamento $\mathcal{M}$, e $\beta(a, \mathcal{M})$ corresponde ao custo da associao da aresta a

ao seu respectivo par de clusters, considerando o mapeamento $\mathcal{M}$. Definimos $c(\mathcal{T}_I^r(i), \mathcal{M})$, para todo $i \in \delta_{\mathcal{T}_I^r}^+(r)$, tal que

$$c(\mathcal{T}_I^r(i), \mathcal{M}) = \beta(ir, \mathcal{M}) + \sum_{j \in V(\mathcal{T}_I^r(i))} \Delta(j, \mathcal{M}) + \sum_{a \in A(\mathcal{T}_I^r(i))} \beta(a, \mathcal{M}).$$

Repare que $c(\mathcal{T}_I^r(i), \mathcal{M})$ contém, além do custo das associações dos vértices e arestas de $\mathcal{T}_I^r(i)$, o custo da associação da aresta ir . Utilizando esta definição, se $\mathcal{M}$ tem custo mínimo dentre todos os mapeamentos que têm r associado a q , então

$$\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = c_{rq} + \sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M}).$$

Lema 5.2.1 *O algoritmo MGB-IMEDIATO está correto, ou seja, devolve um grafo bipartido $B = (V_B, A_B)$ e uma função λ que associa custos às arestas de A_B , tais que o custo de um emparelhamento máximo de custo mínimo em B corresponde a $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) - c_{rq}$.*

Prova. Seja $E \subset A_B$ um emparelhamento máximo de custo mínimo em B , sendo que B é o grafo devolvido pelo algoritmo MGB-IMEDIATO. A partir de E , construiremos um mapeamento viável $\mathcal{M}$ de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, tal que r está associado a q , o custo de $\mathcal{M}$ é dado por $\lambda(E) + c_{rq}$ e $\mathcal{M}$ tem custo mínimo.

Primeiramente, definimos que r está associado a q em $\mathcal{M}$. Para cada aresta $e \in E$, temos dois casos:

- (1) Suponha que a aresta e é da forma $\{u_i, p_i\}$, com $1 \leq i \leq t$. De acordo com a rotulação que estamos utilizando, o vértice i é filho do vértice r . Definimos que todos os vértices de $V(\mathcal{T}_I^r(i))$ estão associados a q em $\mathcal{M}$. Dessa forma, todas as arestas de $A(\mathcal{T}_I^r(i))$ estão associadas a q em $\mathcal{M}$. Como os vértices r e i estão associados a q em $\mathcal{M}$, então a aresta ir está associada a q em $\mathcal{M}$. Pela forma de e , sabemos que seu custo foi atribuído na linha 9 do algoritmo. Logo, o custo desta aresta é equivalente a $c(\mathcal{T}_I^r(i), \mathcal{M})$.
- (2) Suponha que a aresta e é da forma $\{u_i, v_k\}$, com $1 \leq i \leq t$ e $1 \leq k \leq w$. De acordo com a rotulação que estamos utilizando, o vértice i é filho do vértice r , e o cluster k é filho do cluster q . Seja $H(j)$ o conjunto dos ancestrais de j em $\mathcal{T}_I^r(i)$, para todo $j \in V(\mathcal{T}_I^r(i))$. Seja $j \in V(\mathcal{T}_I^r(i))$,

tal que o valor de $z = \Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})$ é mínimo. Definimos que todos os vértices de $H(j)$ estão associados a q em $\mathcal{M}$, o vértice j está associado ao cluster k em $\mathcal{M}$ e os demais vértices de $V(\mathcal{T}_I^r(j))$, se houver, estão associados ou a k ou a algum descendente de k em $\mathcal{M}$, de forma que a soma dos custos das associações dos vértices e das arestas de $\mathcal{T}_I^r(j)$ seja $\Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k))$ (para encontrar tal mapeamento basta resolver o PCCA para as árvores $\mathcal{T}_I^r(j)$ e $\mathcal{T}_M^q(k)$, sujeito à restrição de que j deve ser associado a k . Assumimos que este valor foi previamente calculado). Pela forma de e , sabemos que seu custo foi atribuído na linha 14 do algoritmo. Logo, o custo desta aresta é equivalente a $c(\mathcal{T}_I^r(i), \mathcal{M})$.

Figura 5.14: Na figura acima, um exemplo da construção de $\mathcal{M}$. À esquerda um emparelhamento máximo E de custo mínimo. À direita temos a construção de $\mathcal{M}$ da seguinte forma: as arestas $\{u_1, p_1\}, \{u_2, p_2\} \in E$ indicam que todos os vértices de $\mathcal{T}_I^r(1)$ e $\mathcal{T}_I^r(2)$ estão associados a q ; a aresta $\{u_3, v_3\} \in E$ indica que existe um vértice j em $\mathcal{T}_I^r(3)$ associado ao cluster de rótulo 3 (o vértice mais escuro em $\mathcal{T}_I^r(3)$). A escolha de j foi feita na linha 14 do algoritmo. Os ancestrais de j estão associados a q e os descendentes de j estão associados ao cluster de rótulo 3 ou a algum de seus descendentes; a aresta $\{u_4, v_2\} \in E$ indica que existe um vértice j em $\mathcal{T}_I^r(4)$ associado ao cluster de rótulo 2 (o vértice mais escuro em $\mathcal{T}_I^r(4)$). As associações dos ancestrais e dos descendentes de j são feitas da mesma forma como no caso anterior); a aresta $\{u_5, v_1\} \in E$ indica que existe um vértice j em $\mathcal{T}_I^r(5)$ associado ao cluster de rótulo 1 (o vértice mais escuro em $\mathcal{T}_I^r(5)$). As associações dos ancestrais e dos descendentes de j são feitas da mesma forma como nos dois casos anteriores). De forma geral, todos os vértices contidos nas regiões mais escuras estão associados a q , e as associações dos vértices contidos nas regiões mais claras já foram calculadas em outros subproblemas.

Seja $V_1 = \{u_1, u_2, \dots, u_t\} \subset V_B$. Tomando um vértice $u_j \in V_1$, podemos imaginar que cada aresta que incide em u_j corresponde a uma forma de associar cada vértice da subárvore $\mathcal{T}_I^r(j)$ a exatamente um cluster. Como E é um emparelhamento máximo, então E cobre todos os vértices de V_1 , que corresponde à parte de menor cardinalidade da bipartição de V_B (para cada $u_i \in V_1$, existe

uma aresta $\{u_i, p_i\} \in A_B$, ou seja, existe pelo menos um emparelhamento máximo em B que cobre V_1). Logo, todos os vértices de $V(\mathcal{T}_I^r)$ são associados a algum cluster de $V(\mathcal{T}_M^q)$ em $\mathcal{M}$.

Da forma como construímos $\mathcal{M}$, se considerarmos cada subárvore $\mathcal{T}_I^r(j)$ isoladamente, para todo $j \in \delta_{\mathcal{T}_I^r}^+(r)$, não há violação de nenhuma das restrições do PCCA *para árvores*. Logo, a única possibilidade de $\mathcal{M}$ ser inviável seria se tivéssemos a seguinte situação: sejam $\mathcal{T}_I^r(j)$ e $\mathcal{T}_I^r(j')$, tais que $j, j' \in \delta_{\mathcal{T}_I^r}^+(r)$ e $j \neq j'$. Suponha que dois vértices $u \in V(\mathcal{T}_I^r(j))$ e $u' \in V(\mathcal{T}_I^r(j'))$ estão associados ao mesmo cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Neste caso, teríamos duas ou mais arestas de $A(\mathcal{T}_I^r)$ associadas a qk em $\mathcal{M}$, violando a condição do lema 4.1.1. Precisamos, então, provar que tal situação não ocorre.

Suponha que a situação mencionada ocorre em $\mathcal{M}$, para algum $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Da forma como $\mathcal{M}$ é construído, teríamos que duas ou mais arestas de E incidem em $v_k \in V_B$, o que é uma contradição com o fato de E ser um emparelhamento. Portanto, o mapeamento $\mathcal{M}$ é viável. Resta mostrar que $\mathcal{M}$ é ótimo.

Toda aresta $e \in E$ é da forma $\{u_i, p_i\}$ ou $\{u_i, v_k\}$, para $1 \leq i \leq t$ e $1 \leq k \leq w$. Acabamos de mostrar (casos 1 e 2) que, para toda aresta $e \in E$, vale que $\lambda_e = c(\mathcal{T}_I^r(i), \mathcal{M})$. Logo, $\lambda(E) = \sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M})$. Precisamos mostrar que

$$\sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M}) = \Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) - c_{rq}.$$

Suponha que $\sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M}) < \Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) - c_{rq}$. Seja α o custo de $\mathcal{M}$. Como r está associado a q em $\mathcal{M}$, então $\alpha = \sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M}) + c_{rq}$. Logo, $\alpha < \Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, o que é uma contradição, pois $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$ corresponde ao custo de um mapeamento ótimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r seja associado a q .

Suponha que $\sum_{i \in \delta_{\mathcal{T}_I^r}^+(r)} c(\mathcal{T}_I^r(i), \mathcal{M}) > \Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) - c_{rq}$. Seja $\mathcal{M}'$ um mapeamento ótimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q . Se o custo de $\mathcal{M}'$ for menor do que o custo de $\mathcal{M}$, mostraremos que é possível construir um emparelhamento máximo $E' \subset A_B$, tal que $\lambda(E') < \lambda(E)$. Como E é máximo e tem custo mínimo, concluiremos que tal situação não ocorre, ou seja, $\mathcal{M}$ é ótimo. Vejamos então a construção de E' .

Conforme definimos na lista de parâmetros do algoritmo, a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO. Conseqüentemente, para todo $i \in \delta_{\mathcal{T}_I^r}^+(r)$, a subárvore $\mathcal{T}_I^r(i)$ é um caminho e,

para todo $j \in V(\mathcal{T}_I^r(i))$, se j é o vértice de menor nível, dentre todos os vértices de $V(\mathcal{T}_I^r(i))$, que está associado a algum cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, então, pelo lema 5.1.7, todos os vértices de $H(j)$ estão associados a q , e cada vértice de $\mathcal{T}_I^r(j)$ está associado ou a k ou a algum descendente de k . Logo, para todo $i \in \delta_{\mathcal{T}_I^r}^+(r)$, temos uma das duas seguintes situações: (i) todos os vértices de $V(\mathcal{T}_I^r(i))$ estão associados a q em $\mathcal{M}'$; (ii) existe algum $j \in V(\mathcal{T}_I^r(i))$ associado a algum $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}'$, tal que todos os vértices de $H(j)$ estão associados a q , e cada vértice de $V(\mathcal{T}_I^r(j))$ está associado ou a k ou a algum descendente de k .

No caso (i), temos que existe uma aresta $e = \{u_i, p_i\}$ em A_B , tal que $\lambda_e = c(\mathcal{T}_I^r(i), \mathcal{M}')$. No caso (ii), como $\mathcal{M}'$ é ótimo, então o valor de

$$z = \Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})$$

é mínimo. Logo, existe uma aresta $e = \{u_i, v_k\}$ em A_B , tal que $\lambda_e = c(\mathcal{T}_I^r(i), \mathcal{M}')$. Estabelecemos que nos dois casos a aresta e é incluída em E' , para todo $i \in \delta_{\mathcal{T}_I^r}^+(r)$. Mostraremos que E' é um emparelhamento máximo tal que $\lambda(E') < \lambda(E)$.

Pela construção de E' , é fácil ver que todos os vértices de V_1 são cobertos por E' , e no máximo uma aresta incide em cada um dos vértices de V_1 . Para mostrar que E' é um emparelhamento, precisamos mostrar que no máximo uma aresta incide em cada um dos vértices de $V_2 = V_B \setminus V_1$.

Cada vértice de V_2 é da forma p_i ou v_k , tal que $1 \leq i \leq t$ e $1 \leq k \leq w$. Para todo $p_i \in V_2$, tal que $1 \leq i \leq t$, existe exatamente uma aresta adjacente a p_i em A_B . Logo, se existem duas ou mais arestas em E' que incidem em um mesmo vértice de V_2 , tal vértice tem a forma v_k , para algum $1 \leq k \leq w$. Considerando a forma como o algoritmo constrói B , se duas ou mais arestas de E' incidem em um mesmo vértice $v_k \in V_2$, temos que dois vértices j de $\mathcal{T}_I^r(i)$ e j' de $\mathcal{T}_I^r(i')$, tais que $i, i' \in \delta_{\mathcal{T}_I^r}^+(r)$ e $i \neq i'$, estão associados a um mesmo cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}'$. Mostraremos que tal situação não ocorre em $\mathcal{M}'$.

Pelo lema 5.1.8, se dois vértices distintos $i, j \in V(\mathcal{T}_I^r) \setminus \{r\}$ estão associados a algum cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}'$, então uma das seguintes afirmações é verdadeira: (a) i é ancestral de j em $\mathcal{T}_I^r$; (b) i é descendente de j em $\mathcal{T}_I^r$; (c) i e j possuem um ancestral em comum associado a k em $\mathcal{M}'$. Como $\mathcal{T}_I^r(i)$ é um caminho, para todo $i \in \delta_{\mathcal{T}_I^r}^+(r)$, e r está associado a q em $\mathcal{M}'$, então a afirmação (c) não é verdadeira para $\mathcal{M}'$. Logo, se dois vértices j e j' estão associados a um mesmo cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em $\mathcal{M}'$, então j e j' pertencem a uma mesma subárvore (um mesmo caminho) $\mathcal{T}_I^r(i)$,

para algum $i \in \delta_{T_I^r}^+(r)$. Portanto, E' é um emparelhamento máximo.

Como E' foi construído a partir de $\mathcal{M}'$, que é um mapeamento ótimo, temos que

$$\lambda(E') = \sum_{i \in \delta_{T_I^r}^+(r)} c(T_I^r(i), \mathcal{M}) = \Phi(T_I^r, T_M^q) - c_{rq} < \lambda(E).$$

Acabamos de mostrar que E' é um emparelhamento máximo com custo menor do que $\lambda(E)$, o que é uma contradição com o fato de E ser um emparelhamento máximo de custo mínimo.

Concluimos que o custo de um emparelhamento máximo de custo mínimo em B corresponde a $\Phi(T_I^r, T_M^q) - c_{rq}$, como queríamos mostrar. ■

Lema 5.2.2 *O algoritmo MGB-IMEDIATO executa $O(t \cdot w \cdot n)$ operações, sendo que $t = |\delta_{T_I^r}^+(r)|$, $w = |\delta_{T_M^q}^+(q)|$ e $n = |V(T_I^r)|$.*

Prova. Na tabela abaixo, mostramos o consumo de tempo de cada linha do algoritmo:

Linha(s)	Operações	Obs.
1 a 5	$\Theta(1)$	
6 a 9	$\Theta(t)$	
9	$\Theta(t \cdot n)$	
10 a 13	$\Theta(t \cdot w)$	
14	$O(t \cdot w \cdot n)$	(i)
15 e 16	$\Theta(1)$	

(i) – na linha 14 não detalhamos como o algoritmo encontra o valor de λ_e . Sabemos que, para todo $i \in V(T_I^r) \setminus \{r\}$ e $k \in \delta_{T_M^q}^+(q)$, o valor de $\Phi(T_I^r(i), T_M^q(k))$ foi previamente calculado. Logo, é possível realizar tal cálculo em tempo $O(n)$ a cada vez que esta linha é executada. Basta escolher os vértices na ordem em que eles aparecem no caminho de i até a respectiva folha. A cada vértice j escolhido em tal caminho, é possível calcular em tempo constante o valor de

$$\Phi(T_I^r(j), T_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']}),$$

bastando reaproveitar o último valor calculado.

Somando os valores da tabela, temos que o algoritmo executa $O(t \cdot w \cdot n)$ operações. ■

Utilizando o lema 5.2.1, podemos facilmente escrever um algoritmo para resolver o problema de interesse, no caso em que a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO.

O algoritmo que chamaremos de COMBINA-SOLUÇÕES-IMEDIATO recebe como parâmetros duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO.

Assumimos que $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$. O algoritmo devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

COMBINA-SOLUÇÕES-IMEDIATO($\mathcal{T}_I^r, \mathcal{T}_M^q$)

- 1 $B, \lambda \leftarrow \text{MGB-IMEDIATO}(\mathcal{T}_I^r, \mathcal{T}_M^q)$
- 2 $E \leftarrow \text{MÉTODO-HÚNGARO}(B, \lambda)$
- 3 devolva $\lambda(E) + c_{rq}$

Do lema 5.2.1, podemos derivar o seguinte corolário.

Corolário 5.2.1 *O algoritmo COMBINA-SOLUÇÕES-IMEDIATO está correto, ou seja, devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.*

Lema 5.2.3 *O algoritmo COMBINA-SOLUÇÕES-IMEDIATO executa $O(t \cdot w \cdot n + (t + w)^3)$ operações, sendo que $n = |V(\mathcal{T}_I^r)|$, $t = |\delta_{\mathcal{T}_I^r}^+(r)|$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$.*

Prova. Pelo lema 5.2.2, a linha 1 executa $O(t \cdot w \cdot n)$ operações. Claramente, $|V_B| = 2(t + w) = O(t + w)$, sendo que $B = (V_B, A_B)$ foi construído na linha 1. Conforme vimos no início desta seção, o algoritmo MÉTODO-HÚNGARO executa $O(|V_B|^3)$ operações. Logo, a linha 2 executa $O((t + w)^3)$ operações. Portanto, o algoritmo COMBINA-SOLUÇÕES-IMEDIATO executa $O(t \cdot w \cdot n + (t + w)^3)$ operações. ■

Mostramos como resolver o problema de interesse quando a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO. Resta mostrar como fazer isso para os demais rótulos.

Resolução do caso em que $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO

Quando $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO (quando o núcleo u de G_I está contido em $\mathcal{T}_I^r$, o vértice r tem apenas um filho em $\mathcal{T}_I^r$, e $r \neq u$), temos as duas seguintes situações: (i) o vértice u está associado a q , conseqüentemente, seus ancestrais também estão associados a q ; (ii) o vértice u não está associado a q , conseqüentemente, seus descendentes também não estão associados a q . Ilustramos na figura 5.15 os casos (i) e (ii).

Figura 5.15: À esquerda o caso (i), e à direita o caso (ii).

Para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, precisamos considerar os casos (i) e (ii). Vejamos o que pode ser feito em cada um dos dois casos:

caso (i): como u está associado a q , sabemos que todos os ancestrais de u também estão associados a q . Resta decidir quais serão as associações dos descendentes de u . Mas u é o núcleo de G_I . Logo, a subárvore $\mathcal{T}_I^r(u)$ é classificada com o rótulo IMEDIATO. Ou seja, reduzimos o problema original a um problema que sabemos resolver.

Figura 5.16: Caso (i). O problema ilustrado na figura 5.15 é reduzido ao ilustrado acima.

caso (ii): como u não está associado a q , sabemos que todos os descendentes de u também não estão associados a q . Além disso, não precisamos decidir quais serão as associações dos descendentes de u , pois, ao associar u a um cluster $k \in \delta_{T_M^q}^+(q)$, podemos reaproveitar o valor de $\Phi(\mathcal{T}_I^r(u), \mathcal{T}_M^q(k))$, que já foi calculado. Da mesma forma, se associarmos algum ancestral de u , digamos i , a um cluster $k \in \delta_{T_M^q}^+(q)$, não precisamos decidir quais serão as associações dos descendentes de i (isso inclui u e seus descendentes), pois, podemos reaproveitar o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, que já foi calculado. Portanto, só precisamos resolver quais serão os clusters para os quais associaremos o vértice u e seus ancestrais. Faremos isso de forma enumerativa.

Figura 5.17: Caso (ii). O problema ilustrado na figura 5.15 é reduzido ao ilustrado acima.

O algoritmo que chamaremos de COMBINA-SOLUÇÕES-SEMI-IMEDIATO recebe como parâmetros duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO.

Assumimos que $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{T_M^q}^+(q)$. O algoritmo devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. No algoritmo abaixo, utilizaremos a notação $\bar{H}(i)$ para designar o conjunto que contém $i \in V(\mathcal{T}_I^r)$ e seus ancestrais em $\mathcal{T}_I^r$, desconsiderando r .

COMBINA-SOLUÇÕES-SEMI-IMEDIATO($\mathcal{T}_I^r, \mathcal{T}_M^q$)

```

1   $z^* \leftarrow \infty$ 
2   $i \leftarrow r$ 
3  enquanto  $|\delta_{\mathcal{T}_I^r}^+(i)| = 1$ 
4      faça  $j \leftarrow$  o filho único de  $i$ 
5      para  $k \leftarrow 1$  até  $w$ 
6          faça  $z \leftarrow \Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{ij} + \sum_{v \in H(i)} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$ 
7           $z^* \leftarrow \min\{z^*, z\}$ 
8       $i \leftarrow j$ 
9   $B, \lambda \leftarrow \text{MGB-IMEDIATO}(\mathcal{T}_I^r(i), \mathcal{T}_M^q) \triangleright i$  agora é o núcleo de  $G_I$ 
10  $E \leftarrow \text{MÉTODO-HÚNGARO}(B, \lambda)$ 
11  $z \leftarrow \lambda(E) + \sum_{v \in H(i)} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$ 
12 devolva  $\min\{z^*, z\}$ 

```

Lema 5.2.4 *O algoritmo COMBINA-SOLUÇÕES-SEMI-IMEDIATO está correto, ou seja, devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.*

Prova. Seja $u \in V(\mathcal{T}_I^r)$ o núcleo de G_I . Em um mapeamento viável de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$ (sujeito à restrição de que r deve ser associado a q), temos duas possibilidades com relação a u : (i) u não está associado a q ; (ii) u está associado a q . O algoritmo considera as duas possibilidades. O caso (i) é tratado nas linhas 3 a 8, e o caso (ii) é tratado nas linhas 9 a 11.

Sejam $j \in H(u)$ e $i = \text{pai}[j]$. Claramente, se j é o vértice de menor nível que está associado a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$ em um mapeamento ótimo, então

$$\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = \Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{ij} + \sum_{v \in H(i)} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}.$$

Ou seja, para enumerar todas as soluções nas quais u não está associado a q , basta considerar cada um dos vértices de $H(u)$ como sendo o vértice de menor nível associado a k , para todo $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Isso é feito nas linhas 3 a 8. Considere o seguinte invariante do laço que começa na linha 3:

a variável z^ contém o custo de um melhor mapeamento de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$ (sujeito à restrição de que r deve ser associado a q), dentre todas as soluções nas quais consideramos cada um dos vértices de $H(j)$ como sendo o vértice de menor nível associado a k , para todo $k \in \delta_{\mathcal{T}_M^q}^+(q)$.*

Inicialização

Imediatamente antes da primeira iteração do laço, vale que $j = r$, $H(j) = \emptyset$ e $z^* = \infty$. Portanto, o invariante é válido imediatamente antes da primeira iteração do laço.

Manutenção

Suponha que o invariante é válido imediatamente antes de uma iteração qualquer, tal que $i \neq r$ e $\delta_{\mathcal{T}_I^r}^+(i) = 1$. Ou seja, a variável z^* contém o custo de um melhor mapeamento de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$ (sujeito à restrição de que r deve ser associado a q), dentre todas as soluções nas quais consideramos cada um dos vértices de $H(i)$ como sendo o vértice de menor nível associado a k , para todo $k \in \delta_{\mathcal{T}_M^q}^+(q)$. Como j é o único filho de i , resta apenas considerar as soluções nas quais j é o vértice de menor nível que está associado a cada um dos clusters de $\delta_{\mathcal{T}_M^q}^+(q)$. Isto é feito no laço das linhas 5 a 7. Portanto, o invariante é válido imediatamente antes da próxima iteração.

Término

Pela condição de parada do laço, vale que a variável z^* contém o custo de um melhor mapeamento de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$ (sujeito à restrição de que r deve ser associado a q), dentre todas as soluções nas quais consideramos cada um dos vértices de $H(u)$ como sendo o vértice de menor nível associado a k , para todo $k \in \delta_{\mathcal{T}_M^q}^+(q)$.

Provamos que o caso (i) é tratado corretamente no algoritmo. Resta provar que o caso (ii) também é tratado corretamente.

Pela condição de parada do laço que começa na linha 3, vale que $i = u$, ou seja, i é o núcleo de G_I . Pelo lema 5.1.6, se i está associado a q , então todos os ancestrais de i também estão associados a q . Claramente, a subárvore $\mathcal{T}_I^r(i)$ é classificada com o rótulo IMEDIATO. Portanto, se i estiver associado a q em um mapeamento ótimo, então

$$\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = \lambda(E) + \sum_{v \in H(i)} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq},$$

sendo que E foi obtido na linha 9.

Portanto, o valor devolvido na linha 12 corresponde a $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. ■

Lema 5.2.5 *O algoritmo COMBINA-SOLUÇÕES-IMEDIATO executa $O(t \cdot w \cdot n + (t + w)^3)$ operações,*

sendo que $n = |V(\mathcal{T}_I^r)|$, $t = |\delta_{\mathcal{T}_I^r}^+(u)|$ ($u \in V(\mathcal{T}_I^r)$ é o núcleo de G_I) e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$.

Prova. As linhas 1 a 2 executam $\Theta(1)$ operações. O laço que começa na linha 3 é executado $O(n)$ vezes. O laço que começa na linha 5 é executado $\Theta(w)$ vezes. Logo, as linhas 3 a 8 executam $O(n \cdot w)$ operações. Pelo lema 5.2.2, a linha 9 executa $O(t \cdot w \cdot n)$ operações. Claramente, $|V_B| = 2(t + w) = O(t + w)$, sendo que $B = (V_B, A_B)$ foi construído na linha 9. Conforme vimos no início desta seção, o algoritmo MÉTODO-HÚNGARO executa $O(|V_B|^3)$ operações. Logo, a linha 2 executa $O((t + w)^3)$ operações. Portanto, o algoritmo COMBINA-SOLUÇÕES-SEMI-IMEDIATO executa $O(n \cdot w + t \cdot w \cdot n + (t + w)^3) = O(t \cdot w \cdot n + (t + w)^3)$ operações. ■

Vimos que quando $\mathcal{T}_I^r$ é classificada com o rótulo SEMI-IMEDIATO, podemos utilizar basicamente a mesma idéia do caso em que $\mathcal{T}_I^r$ é classificada com o rótulo IMEDIATO. Basta enumerar algumas soluções separadamente.

Resolução do caso em que $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL

O último caso que precisamos analisar ocorre quando $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL (o núcleo u de G_I está contido em $\mathcal{T}_I^r$, o vértice r tem dois filhos em $\mathcal{T}_I^r$ e $r \neq u$). Sejam s_1, s_2 e s_3 , tais que $s_1, s_2 \in \delta_{\mathcal{T}_I^r}^+(r)$, a subárvore $\mathcal{T}_I^r(s_1)$ é um caminho e s_3 é o núcleo de G_I (conseqüentemente, ou $s_2 = s_3$ ou s_2 é ancestral de s_3 em $\mathcal{T}_I^r$). Novamente, temos as duas seguintes situações: (i) o vértice s_3 está associado a q , conseqüentemente, seus ancestrais também estão associados a q ; (ii) o vértice s_3 não está associado a q , conseqüentemente, seus descendentes também não estão associados a q .

Figura 5.18: À esquerda o caso (i), e à direita o caso (ii).

Para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$, precisamos considerar os casos (i) e (ii). Vejamos o que pode ser feito em cada um dos dois casos:

caso (i): como s_3 está associado a q , sabemos que todos os ancestrais de s_3 também estão associados a q . Resta decidir quais serão as associações dos descendentes de s_3 e dos vértices da subárvore $\mathcal{T}_I^r(s_1)$. Podemos construir uma nova árvore $\mathcal{T}_I^{r'}$ a partir de $\mathcal{T}_I^r$. Seja H o conjunto dos ancestrais de s_3 em $\mathcal{T}_I^r$, e seja $A_H \subset A(\mathcal{T}_I^r)$ as arestas com algum extremo em H . Definimos $V(\mathcal{T}_I^{r'}) = V(\mathcal{T}_I^r) \setminus H$ e $A(\mathcal{T}_I^{r'}) = \{s_1, s_3\} \cup A(\mathcal{T}_I^r) \setminus A_H$. A aresta $\{s_1, s_3\}$ “fará o papel” da aresta $\{s_1, r\}$. Repare que a árvore $\mathcal{T}_I^{r'}$ é classificada com o rótulo IMEDIATO.

Figura 5.19: Caso (i). A árvore $\mathcal{T}_I^{r'}$ é classificada com o rótulo IMEDIATO.

É claro que a construção da árvore $\mathcal{T}_I^{r'}$ é apenas simbólica. Utilizamos tal construção apenas para ilustrar as associações que precisamos decidir quando s_3 está associado a q , e também para mostrar que este caso pode ser reduzido a um caso que sabemos como resolver. O algoritmo MGB-DIVISÍVEL-1 será responsável por construir um grafo bipartido representando este caso.

caso (ii): como s_3 não está associado a q , sabemos que todos os descendentes de s_3 também não estão associados a q . Além disso, não precisamos decidir quais serão as associações dos descendentes de s_3 , pois, ao associar s_3 a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, podemos reaproveitar o valor

de $\Phi(\mathcal{T}_I^r(s_3), \mathcal{T}_M^q(k))$, que já foi calculado. Da mesma forma, se associarmos algum ancestral de s_3 , digamos i , a um cluster $k \in \delta_{\mathcal{T}_M^q}^+(q)$, não precisamos decidir quais serão as associações dos descendentes de i (isso inclui s_3 e seus descendentes), pois, podemos reaproveitar o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$, que já foi calculado. Portanto, só precisamos resolver quais serão os clusters para os quais associaremos os vértices da subárvore $\mathcal{T}_I^r(s_1)$, o vértice s_3 e os ancestrais de s_3 . Se considerarmos apenas estes vértices, a subárvore resultante é classificada com o rótulo IMEDIATO. O algoritmo MGB-DIVISÍVEL-2 será responsável por construir um grafo bipartido representando este caso.

Figura 5.20: Considerando apenas os vértices da subárvore $\mathcal{T}_I^r(s_1)$, o vértice s_3 e os ancestrais de s_3 , a subárvore resultante é classificada com o rótulo IMEDIATO.

De forma geral, o que fazemos quando $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL é dividir o problema em dois grafos bipartidos, nos quais deseja-se encontrar um emparelhamento máximo de custo mínimo. Posteriormente, tomaremos a solução proveniente do emparelhamento de menor custo dentre estes dois. A seguir, mostramos os algoritmos MGB-DIVISÍVEL-1 e MGB-DIVISÍVEL-2, responsáveis por construir estes grafos bipartidos.

O algoritmo MGB-DIVISÍVEL-1 recebe os seguintes parâmetros: duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que G_I é uma estrela estendida e $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL; um vértice $s_1 \in \delta_{\mathcal{T}_I^r}^+(r)$, tal que $\mathcal{T}_I^r(s_1)$ é um caminho; um vértice $s_3 \in V(\mathcal{T}_I^r)$, tal que s_3 é o núcleo de G_I (repare que s_3 não está contido na subárvore $\mathcal{T}_I^r(s_1)$).

O algoritmo devolve um grafo bipartido $B = (V_B, A_B)$ e uma função λ que associa custos às arestas de A_B . Seja E um emparelhamento máximo de custo mínimo em B , e seja z o custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e o vértice s_3 deve ser associado a q . Seja H o conjunto que contém s_3 e seus ancestrais em $V(\mathcal{T}_I^r)$, desconsiderando r . Então, vale que $z = \lambda(E) + \sum_{v \in H} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$.

MGB-DIVISÍVEL-1($\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_3$)

- 1 Sejam $t = |\delta_{\mathcal{T}_I^r}^+(s_3)| + 1$ e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$
- 2 $V_B \leftarrow \{u_1, u_2, \dots, u_t\} \cup \{p_1, p_2, \dots, p_t\} \cup \{v_1, v_2, \dots, v_w\}$
- 3 $A_B \leftarrow \emptyset$
- 4 Assuma que os vértices de $\delta_{\mathcal{T}_I^r}^+(s_3)$ possuem os rótulos $1, 2, \dots, t-1$, e que $s_1 = t$
- 5 **para** $i \leftarrow 1$ até t
- 6 **faça** $e \leftarrow \{u_i, p_i\}$
- 7 $A_B \leftarrow A_B \cup \{e\}$
- 8 $\lambda_e \leftarrow \sum_{j \in V(\mathcal{T}_I^r(i))} (c_{jq} + d_{qq}^{j, \text{pai}[j]})$
- 9 **para** $k \leftarrow 1$ até w
- 10 **faça** $e \leftarrow \{u_i, v_k\}$
- 11 $A_B \leftarrow A_B \cup \{e\}$
- 12 $\triangleright$ Seja $H(j)$ o conjunto dos ancestrais de j em $\mathcal{T}_I^r(i)$, $\forall j \in V(\mathcal{T}_I^r(i))$
- 13 $\lambda_e \leftarrow \min_{j \in V(\mathcal{T}_I^r(i))} [\Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})]$
- 14 Seja $B = (V_B, A_B)$
- 15 devolva B, λ

O algoritmo MGB-DIVISÍVEL-2 recebe os seguintes parâmetros: duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que G_I é uma estrela estendida e $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL; um vértice $s_1 \in \delta_{\mathcal{T}_I^r}^+(r)$, tal que $\mathcal{T}_I^r(s_1)$ é um caminho; um vértice $s_2 \in \delta_{\mathcal{T}_I^r}^+(r)$, tal que $s_2 \neq s_1$; um vértice $s_3 \in V(\mathcal{T}_I^r)$, tal que s_3 é o núcleo de G_I (assumimos que $g(s_3) > 2$) e, conseqüentemente, ou $s_3 = s_2$ ou s_3 é descendente de s_2 em $\mathcal{T}_I^r$.

O algoritmo devolve um grafo bipartido $B = (V_B, A_B)$ e uma função λ que associa custos às arestas de A_B . Seja E um emparelhamento máximo de custo mínimo em B , e seja z o custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e o vértice s_3 **não** deve ser associado a q . Então, vale que $z = \lambda(E) + c_{rq}$.

MGB-DIVISÍVEL-2($\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_2, s_3$)

- 1 Seja $w = |\delta_{\mathcal{T}_M^q}^+(q)|$
- 2 $V_B \leftarrow \{u_1, u_2\} \cup \{p_1\} \cup \{v_1, v_2, \dots, v_w\}$
- 3 $e \leftarrow \{u_1, p_1\}$
- 4 $A_B \leftarrow \{e\}$
- 5 $\lambda_e \leftarrow \sum_{i \in V(\mathcal{T}_I^r(s_1))} (c_{iq} + d_{qq}^{i, \text{pai}[i]})$
- 6 **para** $k \leftarrow 1$ até w
- 7 **faça** $e_1 \leftarrow \{u_1, v_k\}$
- 8 $A_B \leftarrow A_B \cup \{e_1\}$
- 9 $\triangleright$ Seja $H(j)$ o conjunto dos ancestrais de j em $\mathcal{T}_I^r(s_1)$, $\forall j \in V(\mathcal{T}_I^r(s_1))$
- 10 $\lambda_{e_1} \leftarrow \min_{j \in V(\mathcal{T}_I^r(s_1))} [\Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})]$
- 11 $e_2 \leftarrow \{u_1, v_k\}$
- 12 $A_B \leftarrow A_B \cup \{e_2\}$
- 13 Seja S o conjunto de todos os descendentes de s_3 em $\mathcal{T}_I^r$
- 14 $\mathcal{T}_I^{s_2} \leftarrow \mathcal{T}_I^r(s_2) - S$
- 15 Seja $H'(j)$ o conjunto dos ancestrais de j em $\mathcal{T}_I^{s_2}$, $\forall j \in V(\mathcal{T}_I^{s_2})$
- 16 $\lambda_{e_2} \leftarrow \min_{j \in V(\mathcal{T}_I^{s_2})} [\Phi(\mathcal{T}_I^r(j), \mathcal{T}_M^q(k)) + d_{kq}^{j, \text{pai}[j]} + \sum_{j' \in H(j)} (c_{j'q} + d_{qq}^{j', \text{pai}[j']})]$
- 17 Seja $B = (V_B, A_B)$
- 18 devolva B, λ

Lema 5.2.6 *O algoritmo MGB-DIVISÍVEL-1 executa $O(w \cdot n)$ operações, sendo que $w = |\delta_{\mathcal{T}_M^q}^+(q)|$ e $n = |V(\mathcal{T}_I^r)|$.*

Lema 5.2.7 *O algoritmo MGB-DIVISÍVEL-2 executa $O(t \cdot w \cdot n)$ operações, sendo que $t = |\delta_{\mathcal{T}_I^r}^+(s_3)|$ (s_3 é o núcleo de G_I), $w = |\delta_{\mathcal{T}_M^q}^+(q)|$ e $n = |V(\mathcal{T}_I^r)|$.*

Não provaremos estes dois lemas, pois, do ponto de vista de complexidade computacional, essencialmente estes dois algoritmos são praticamente idênticos ao algoritmo MGB-IMEDIATO. No algoritmo MGB-DIVISÍVEL-1, temos que $t = 2$, por isso seu consumo de tempo é $O(w \cdot n)$. No algoritmo MGB-DIVISÍVEL-2, o grafo bipartido é construído a partir dos filhos de $s_3 \neq r$, que é o núcleo de G_I .

Lema 5.2.8 *Sejam $B = (V_B, A_B)$ e λ devolvidos pelo algoritmo MGB-DIVISÍVEL-1. Seja E um emparelhamento máximo de custo mínimo em B , e seja z o custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e o vértice s_3 deve ser*

associado a q . Seja H o conjunto que contém s_3 e seus ancestrais em $V(\mathcal{T}_I^r)$, desconsiderando r . Então, vale que $z = \lambda(E) + \sum_{v \in H} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$.

Lema 5.2.9 *Sejam $B = (V_B, A_B)$ e λ devolvidos pelo algoritmo MGB-DIVISÍVEL-2. Seja E um emparelhamento máximo de custo mínimo em B , e seja z o custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e o vértice s_3 **não** deve ser associado a q . Então, vale que $z = \lambda(E) - c_{rq}$.*

A idéia da prova destes dois lemas é similar à prova do lema 5.2.2. Para não tornar o texto repetitivo, omitiremos a prova destes dois lemas.

Suponha que $B^1 = (V_B^1, A_B^1)$, λ^1 , $B^2 = (V_B^2, A_B^2)$ e λ^2 foram construídos pelos algoritmos MGB-DIVISÍVEL-1 e MGB-DIVISÍVEL-2, respectivamente. Sejam $E^1 \subset A_B^1$ e $E^2 \subset A_B^2$ dois emparelhamentos máximos de custo mínimo. Seja $z^1 = \lambda^1(E^1) + \sum_{v \in H} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$, sendo que H corresponde ao conjunto que contém o vértice s_3 e todos os seus ancestrais em $V(\mathcal{T}_I^r)$, desconsiderando r , e seja $z^2 = \lambda^2(E^2) + c_{rq}$. Sabemos que z^1 corresponde ao custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e s_3 deve ser associado a q . Analogamente, z^2 corresponde ao custo de um mapeamento de custo mínimo de $\mathcal{T}_I^r$ para $\mathcal{T}_M^q$, sujeito à restrição de que r deve ser associado a q e s_3 **não** deve ser associado a q . Pelos lemas 5.2.8 e 5.2.9, temos o seguinte corolário:

Corolário 5.2.2 *Sejam G_I e G_M duas árvores, tais que G_I é uma estrela estendida. Sejam r e q , tais que $\mathcal{T}_I^r$, $\mathcal{T}_M^q$ e a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL. Sejam z^1 e z^2 conforme definidos acima. Temos que $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q) = \min\{z^1, z^2\}$.*

Com este corolário, fica simples a construção de um algoritmo para resolver o problema de interesse, no caso em que $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL.

O algoritmo que chamaremos de COMBINA-SOLUÇÕES-DIVISÍVEL recebe os seguintes parâmetros: uma estrela estendida G_I ; uma árvore G_M ; um vértice r , tal que a subárvore $\mathcal{T}_I^r$ é classificada com o rótulo DIVISÍVEL; um cluster q , tal que $\mathcal{T}_M^q$; três vértices s_1 , s_2 e s_3 , tais que $s_1, s_2 \in \delta_{\mathcal{T}_I^r}^+(r)$, a subárvore $\mathcal{T}_I^r(s_1)$ é um caminho e s_3 é o núcleo de G_I (conseqüentemente, ou $s_2 = s_3$ ou s_2 é ancestral de s_3 em $\mathcal{T}_I^r$).

Assumimos que $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$. O algoritmo devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.

COMBINA-SOLUÇÕES-DIVISÍVEL($\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_2, s_3$)

- 1 $B^1, \lambda^1 \leftarrow \text{MGB-DIVISÍVEL-1}(\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_3)$
- 2 $E^1 \leftarrow \text{MÉTODO-HÚNGARO}(B^1, \lambda^1)$
- 3 $z^1 \leftarrow \lambda(E^1) + c_{rq}$
- 4 $B^2, \lambda^2 \leftarrow \text{MGB-DIVISÍVEL-2}(\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_2, s_3)$
- 5 $E^2 \leftarrow \text{MÉTODO-HÚNGARO}(B^2, \lambda^2)$
- 6 Seja H o conjunto que contém s_3 e seus ancestrais em $V(\mathcal{T}_I^r)$, desconsiderando r
- 7 $z^2 \leftarrow \lambda^2(E^2) + \sum_{v \in H} (c_{vq} + d_{qq}^{v, \text{pai}[v]}) + c_{rq}$
- 8 devolva $\min\{z^1, z^2\}$

Corolário 5.2.3 *O algoritmo COMBINA-SOLUÇÕES-DIVISÍVEL está correto, ou seja, devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.*

Lema 5.2.10 *O algoritmo COMBINA-SOLUÇÕES-DIVISÍVEL executa $O(t \cdot w \cdot n + (t + w)^3)$ operações, sendo que $n = |V(\mathcal{T}_I^r)|$, $t = |\delta_{\mathcal{T}_I^r}^+(s_3)|$ ($s_3 \in V(\mathcal{T}_I^r)$ é o núcleo de G_I) e $w = |\delta_{\mathcal{T}_M^q}^+(q)|$.*

Prova. Na tabela abaixo, mostramos o consumo de tempo de cada linha do algoritmo:

Linha(s)	Operações	Obs.
1	$O(w \cdot n)$	lema 5.2.6
2	$O(w^3)$	$ V_B^1 = O(w)$
3	$\Theta(1)$	
4	$O(t \cdot w \cdot n)$	lema 5.2.7
5	$O((t + w)^3)$	$ V_B^2 = O(t + w)$
7	$O(n)$	
8	$\Theta(1)$	

Somando a quantidade de operações executadas por todas as linhas, temos que o algoritmo COMBINA-SOLUÇÕES-DIVISÍVEL executa $O(t \cdot w \cdot n + (t + w)^3)$ operações. ■

O algoritmo “combina-soluções-emp”

Nosso objetivo nesta seção é apresentar um algoritmo polinomial que possa “substituir” o algoritmo COMBINA-SOLUÇÕES, caso G_I seja uma estrela estendida. Vimos que os algoritmos

MGB-IMEDIATO, MGB-DIVISÍVEL-1 e MGB-DIVISÍVEL-2 constroem, em tempo polinomial, grafos bipartidos para os quais o custo de um emparelhamento máximo de custo mínimo pode ser utilizado para calcular $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. Conforme mencionamos, o algoritmo MÉTODO-HÚNGARO resolve o problema de encontrar um emparelhamento máximo de custo mínimo em um grafo bipartido em tempo polinomial. Resta apenas mostrar o algoritmo que chamaremos de COMBINA-SOLUÇÕES-EMP, que faz uso adequado destes algoritmos.

O algoritmo COMBINA-SOLUÇÕES-EMP recebe como parâmetros duas árvores $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q \subseteq G_M$, tais que G_I é uma estrela estendida. O algoritmo devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$. Assumimos que, para todo $i \in V(\mathcal{T}_I^r) \setminus \{r\}$ e $k \in \delta_{\mathcal{T}_M^q}^+(q)$, o valor de $\Phi(\mathcal{T}_I^r(i), \mathcal{T}_M^q(k))$ já foi calculado.

COMBINA-SOLUÇÕES-EMP($\mathcal{T}_I^r, \mathcal{T}_M^q$)

```

1  rótulo,  $s_1, s_2, s_3 \leftarrow$  CLASSIFICA-SUBÁRVORE( $\mathcal{T}_I^r$ )
2  se rótulo = IMEDIATO
3      então devolva COMBINA-SOLUÇÕES-IMEDIATO( $\mathcal{T}_I^r, \mathcal{T}_M^q$ )
4  senão se rótulo = SEMI-IMEDIATO
5      então devolva COMBINA-SOLUÇÕES-SEMI-IMEDIATO( $\mathcal{T}_I^r, \mathcal{T}_M^q$ )
6  senão devolva COMBINA-SOLUÇÕES-DIVISÍVEL( $\mathcal{T}_I^r, \mathcal{T}_M^q, s_1, s_2, s_3$ )
```

Considerando os corolários 5.2.1 e 5.2.3, e o lema 5.2.4, temos o seguinte corolário:

Corolário 5.2.4 *O algoritmo COMBINA-SOLUÇÕES-EMP está correto, ou seja, devolve o valor de $\Phi(\mathcal{T}_I^r, \mathcal{T}_M^q)$.*

Dos lemas 5.2.3, 5.2.5 e 5.2.10, podemos derivar o seguinte corolário:

Corolário 5.2.5 *O algoritmo COMBINA-SOLUÇÕES-EMP executa $O(t \cdot w \cdot n + (t + w)^3)$ operações, sendo que $n = |V_I|$, $t = \max\{g(i) \mid i \in V_I\}$ e $w = \max\{g(k) \mid k \in V_M\}$.*

Concluimos que, quando G_I é uma estrela estendida, podemos utilizar uma versão do algoritmo PCCA-PARA-ÁRVORES na qual a chamada ao algoritmo COMBINA-SOLUÇÕES (cujo consumo de tempo é $O(w \cdot n^{w+1})$) é substituída por uma chamada ao algoritmo COMBINA-SOLUÇÕES-EMP (cujo consumo de tempo é $O(t \cdot w \cdot n + (t + w)^3)$). Tal substituição tornaria o algoritmo PCCA-PARA-ÁRVORES polinomial nos tamanhos dos grafos de entrada. Mais precisamente, o consumo de tempo desta versão modificada do algoritmo PCCA-PARA-ÁRVORES é $O(nm^2 + n^2mtw + nm(t + w)^3)$.

Capítulo 6

Experimentos Computacionais

Neste capítulo, apresentamos uma síntese dos experimentos computacionais realizados com o algoritmo de programação dinâmica, apresentado na seção 5.1, para resolver o PCCA *para árvores*. Na verdade, este algoritmo apenas calcula o custo de um mapeamento ótimo, sem exibir um mapeamento com tal custo. No entanto, adaptá-lo de forma a exibir um mapeamento ótimo é muito simples (o consumo de tempo é essencialmente o mesmo).

Tivemos três focos em nossos experimentos, sendo eles: medir o desempenho do algoritmo na prática; medir a eficácia do método com instâncias geradas de forma automatizada; medir a eficácia do método com instâncias reais.

6.1 Desempenho do Algoritmo de Programação Dinâmica na Prática

Na seção 5.1, vimos que o algoritmo PCCA-PARA-ÁRVORES executa $O(w \cdot m \cdot n^{w+2})$ operações, sendo que $n = |V_I|$, $m = |V_M|$ e $w = \max\{g(k) \mid k \in V_M\}$. Vimos também que o desempenho deste algoritmo está fortemente relacionado com o desempenho do algoritmo COMBINA-SOLUÇÕES, que executa $O(w' \cdot n'^{w'+1})$ operações, sendo que $n' = |V(\mathcal{T}_I^r)|$ e $w' = |\delta_{\mathcal{T}_M^q}^+(q)|$ (supondo que $\mathcal{T}_I^r \subseteq G_I$ e $\mathcal{T}_M^q(q) \subseteq G_M$). O algoritmo COMBINA-SOLUÇÕES funciona de forma enumerativa, por isso seu desempenho não é bom. Na prática, podemos implementar estratégias como *branch-and-bound*¹ para diminuir o espaço de busca. Em nossa implementação da rotina COMBINA-SOLUÇÕES, formulamos o problema como um PLI (programa linear inteiro) e utilizamos um resolvidor para resolver este PLI.

¹*Branch-and-bound* é um algoritmo genérico para encontrar uma solução ótima para um problema de otimização combinatória qualquer. O método consiste na geração da árvore de enumeração de todas as soluções viáveis, sendo que diversas subárvores de soluções podem ser eliminadas (ou seja, não precisam ser calculadas) através da utilização de limitantes inferior e superior para o valor de uma solução ótima.

Por isso, nosso interesse nesta seção é medir o comportamento do algoritmo na prática.

De forma geral, o que nos interessa são três parâmetros, sendo eles: a quantidade de vértices do grafo G_I ; a quantidade de clusters do grafo G_M ; o grau máximo dos clusters de V_M . Geramos as entradas aleatoriamente (utilizando um algoritmo baseado no código de Prüfer [8, 9, 37]), de acordo com esses parâmetros. Separamos os resultados dos testes em três tabelas, sendo que cada uma delas contém os resultados dos testes feitos variando um dos parâmetros e deixando os outros dois fixos.

Configurações do ambiente utilizado nos testes.

Computador	Notebook DELL Latitude D510
Sistema operacional	Microsoft Windows XP Professional 2002
Processador	Intel Pentium 1.73 GHz 795 MHz
Memória RAM	512MB
Linguagem de programação	Java SE 6 [38]
Resolvedor de PLIs	GLPK 4.8 [36]
Conector para o resolvedor de PLIs	GLPK 4.8 Java Interface [49]

Cada linha das próximas três tabelas corresponde ao resultado da execução do nosso algoritmo sobre uma instância gerada aleatoriamente.

Tabela 6.1: Variando o parâmetro $|V_M|$.

$ V_I $	$ V_M $	$\max_{k \in V_M} g(k)$	tempo gasto
100	50	20	15 s
100	100	20	32 s
100	200	20	62 s
100	400	20	142 s
100	800	20	457 s

Tabela 6.2: Variando o parâmetro $|V_I|$.

$ V_I $	$ V_M $	$\max_{k \in V_M} g(k)$	tempo gasto
50	100	20	11 s
100	100	20	32 s
200	100	20	106 s
400	100	20	262 s
800	100	20	1101 s

Tabela 6.3: Variando o parâmetro $\max_{k \in V_M} g(k)$.

$ V_I $	$ V_M $	$\max_{k \in V_M} g(k)$	tempo gasto
500	200	≥ 20	787 s
500	200	≥ 40	795 s
500	200	≥ 80	4296 s
500	200	≥ 160	3541 s

Na tabela 6.1, podemos notar que à medida que dobramos o valor de $|V_M|$, o consumo de tempo também dobra. Exceto na última linha, onde o consumo de tempo é aproximadamente o triplo do valor da linha anterior. Neste caso em particular, acreditamos que isto esteja ligado às limitações do ambiente (mais especificamente, memória RAM) e não ao comportamento assintótico do algoritmo. Ou seja, variando só o parâmetro V_M , o algoritmo é linear, o que já estava evidente na análise assintótica de sua complexidade de tempo.

Na tabela 6.2, podemos observar que a medida que dobramos o valor de $|V_I|$, o consumo de tempo triplica (exceto na última linha, como na tabela anterior). Comparando as tabelas 6.1 e 6.2, percebemos que o consumo de tempo contido em cada linha da tabela 6.2 é aproximadamente o dobro do consumo de tempo obtido invertendo os valores dos parâmetros $|V_I|$ e $|V_M|$, exceto quando $|V_I| \leq |V_M|$. Podemos concluir que o algoritmo é mais sensível ao aumento do parâmetro $|V_I|$ do que ao aumento do parâmetro $|V_M|$, o que também já estava evidente na análise assintótica de sua complexidade. Outra observação é que o algoritmo apresentou comportamento quadrático para o valor que fixamos para o parâmetro $\max_{k \in V_M} g(k)$ (supondo que $|V_M|$ também é fixo).

Na tabela 6.3, a medida que aumentamos o valor do parâmetro $\max_{k \in V_M} g(k)$, não ficou evidente qual é a taxa de crescimento do consumo de tempo do algoritmo. Nas duas primeiras linhas o consumo é aproximadamente o mesmo. O mesmo ocorre nas duas últimas linhas. A quarta linha apresenta um consumo de tempo mais de 5 vezes maior do que o apresentado na linha anterior.

Com os resultados apresentados aqui, concluímos que é possível fazer uma implementação do algoritmo PCCA-PARA-ÁRVORES que na prática consiga resolver instâncias relativamente grandes em tempo relativamente curto, mesmo quando os graus dos clusters de V_M são altos.

6.2 Testes com Instâncias Geradas de Forma Automatizada

Uma instância do PCCA *para árvores* é composta por duas árvores $G_I = (V_I, A_I)$ e $G_M = (A_M, V_M)$, e duas funções $c : V_I \times V_M \rightarrow \mathbb{Z}_+$ e $d : A_I \times V_M \times V_M \rightarrow \mathbb{Z}_+$. Antes de mostrar como

geramos as árvores, primeiro mostraremos como construímos as funções c e d , a partir de um par de árvores G_I e G_M .

Para cada par de árvores G_M e G_I , são dados um peso $p_i \in \mathbb{Z}_+$ atribuído a cada vértice $i \in V_I$ e um peso $P_k \in \mathbb{Z}_+$ atribuído a cada cluster $k \in V_M$, bem como um peso $w_{ij} \in \mathbb{Z}_+$ atribuído a cada aresta $ij \in A_I$ e um peso $W_{kl} \in \mathbb{Z}_+$ atribuído a cada aresta $kl \in A_M$. A função de custo das associações dos vértices de V_I para os clusters de V_M foi definida da seguinte maneira:

$$c(i, k) = |p_i - P_k|, \text{ para todo } i \in V_I \text{ e } k \in V_M.$$

A função de custo das associações das arestas de A_I para os pares de clusters de V_M foi definida da seguinte maneira (para todo $ij \in A_I$ e $k, l \in V_M$):

$$d(ij, k, l) = \begin{cases} |w_{ij} - W_{kl}|, & \text{se } kl \in A_M \\ w_{ij}, & \text{se } kl \notin A_M. \end{cases}$$

A seguir, descrevemos como cada par G_I e G_M foi gerado. Fixamos uma constante qualquer para ser a quantidade de vértices da árvore G_M (mais especificamente $|V_M| = 15$ em todos os testes que fizemos). As arestas de A_M foram geradas aleatoriamente, contanto que o grafo G_M fosse uma árvore. Os pesos dos clusters de V_M e das arestas de A_M foram atribuídos aleatoriamente, obedecendo a seguinte relação: $0 \leq P_k \leq \text{MAXPESOVERTICE}$, para todo $k \in V_M$, e $\frac{\text{MAXPESOARESTA}}{2} \leq W_{kl} \leq \text{MAXPESOARESTA}$, para todo $kl \in A_M$, sendo que MAXPESOVERTICE e MAXPESOARESTA são dois parâmetros cujos valores foram variados durante os testes.

A árvore G_I foi obtida em duas etapas. A primeira etapa refere-se à topologia de G_I . Para cada cluster k de V_M , geramos aleatoriamente uma árvore T_k com s vértices, sendo que s é um número aleatório entre 1 e o parâmetro VARIACAOTPOLOGIA . Depois, conectamos todas estas $|V_M|$ árvores de acordo com a topologia de G_M . Ou seja, ligamos, através de uma única aresta, as árvores T_k e T_l se e só se $kl \in A_M$. Então, G_I é o grafo resultante desta construção. Claramente, o grafo G_I é uma árvore (veja figura 6.1). A segunda etapa refere-se à atribuição dos pesos aos vértices e às arestas de G_I . Todo vértice i contido em uma árvore T_k recebeu o mesmo peso do cluster k acrescido (ou decrescido) de um “ruído” (RUIDOVERTICE) aleatório, tal que

$$p_i = P_k \pm \frac{P_k * \text{RUIDOVERTICE}}{100},$$

Figura 6.1: À esquerda uma árvore G_M gerada aleatoriamente. À direita uma árvore G_I gerada a partir de G_M . Note que cada círculo pontilhado indica a árvore correspondente a um cluster $k \in V_M$. Observe ainda que as arestas entre os círculos pontilhados seguem a topologia de G_M .

sendo que $0 \leq \text{RUIDOVERTICE} \leq \text{MAXRUIDOVERTICE} \leq 100$. O parâmetro MAXRUIDOVERTICE foi variado ao longo dos testes.

Sejam $A_I^1 \subseteq A_I$ e $A_I^2 \subset A_I$, tais que A_I^1 é composto pelas arestas de A_I que ligam duas árvores distintas T_k e T_l , para todo $kl \in A_M$, e $A_I^2 = A_I \setminus A_I^1$. Toda aresta ij contida em A_I^1 , que liga as árvores T_k e T_l , para algum $kl \in A_M$, recebeu o mesmo peso da aresta kl acrescido (ou decrescido) de um “ruído” (RUIDOARESTA) aleatório, tal que

$$w_{ij} = W_{kl} \pm \frac{W_{kl} * \text{RUIDOARESTA}}{100},$$

sendo que $0 \leq \text{RUIDOARESTA} \leq \text{MAXRUIDOARESTA} \leq 100$. O parâmetro MAXRUIDOARESTA foi variado ao longo dos testes.

Para cada aresta $ij \in A_I^2$, que está contida em uma árvore T_k , para algum $k \in V_M$, calculamos um peso aleatório f , tal que $0 \leq f \leq \frac{1}{2} \cdot \frac{\text{MAXPESOARESTA}}{|V(T_k)|}$. Então, o peso da aresta ij foi acrescido (ou decrescido) de um “ruído” (RUIDOARESTA) aleatório, tal que

$$w_{ij} = f \pm \frac{f * \text{RUIDOARESTA}}{100}.$$

O parâmetro RUIDOARESTA é o mesmo utilizado nas arestas de A_I^1 . Note que o peso de uma aresta qualquer de A_I^2 , desconsiderando o ruído, é sempre menor ou igual ao peso de uma aresta qualquer

de A_I^1 , e quanto maior for o valor de $|V(T_k)|$, para algum $k \in V_M$, menor será o peso das arestas contidas na árvore T_k .

Nas figuras 6.2 e 6.3, mostramos um exemplo no qual os grafos G_I e G_M possuem uma relação equivalente a que simulamos em nossos testes¹.

Figura 6.2: À esquerda um modelo de um objeto que queremos reconhecer. À direita o grafo G_M representando o modelo.

Figura 6.3: À esquerda uma imagem, segmentada, de um objeto similar ao do modelo. À direita o grafo G_I representando a imagem segmentada. Note que quanto maior a quantidade de vértices em uma árvore T_k , para algum $k \in V_M$, menor é o comprimento (que pode ser considerado como o peso) das arestas de T_k .

Como os testes foram feitos

O mapeamento ótimo esperado nas instâncias geradas da forma como descrevemos é o que contém

¹As figuras 6.2 e 6.3 ilustram um exemplo de representação de modelo e imagem através de grafos que são árvores. No entanto, nas aplicações práticas estes grafos podem não ser árvores.

todos os vértices de T_k associados ao cluster k , para todo $k \in V_M$. Intuitivamente, percebe-se que quando os ruídos aplicados nas arestas e nos vértices são muito baixos, a tendência é que o mapeamento encontrado pelo algoritmo seja igual ao esperado. O objetivo principal dos testes foi medir até que ponto a influência de ruídos prejudica a qualidade das soluções encontradas pelo algoritmo. De forma geral, o que fizemos foi, dado um mapeamento encontrado pelo algoritmo, contar quantos vértices foram associados a clusters diferentes dos que se esperava.

Resultados dos testes

A seguir, apresentamos três tabelas contendo os resultados dos testes realizados. Cada linha de cada tabela é o resultado de cem entradas diferentes, geradas da forma como descrevemos anteriormente. A coluna “% máx ruído vértices” indica o percentual máximo de ruído aplicado nos vértices (corresponde ao valor do parâmetro MAXRUIDOVERTICE). A coluna “% máx ruído arestas” indica o percentual máximo de ruído aplicado nas arestas (corresponde ao valor do parâmetro MAXRUIDO-ARESTA). A coluna “% médio de erros” indica o percentual médio de vértices associados a clusters diferentes dos esperados. Esta coluna está subdividida em duas colunas, cada uma delas indica um valor diferente para o parâmetro VARIACAO TOPOLOGIA, abreviado como “var top”. Ao lado do percentual médio de erros, colocamos o desvio padrão entre parênteses. A coluna “% máx de erros” indica o percentual máximo de vértices associados a clusters diferentes dos esperados em uma única solução. De certa forma, esta coluna indica qual foi a pior solução encontrada com os parâmetros contidos na respectiva linha. Esta coluna, assim como a anterior, está subdividida em duas colunas que indicam o valor do parâmetro VARIACAO TOPOLOGIA.

Tabela 6.4: MAXPESOARESTA = 5, MAXPESOVERTICE = 256.

% máx. ruido		% médio de erros		% máx. de erros	
arestas	vértices	var. top. = 5	var. top. = 15	var. top. = 5	var. top. = 15
0%	0%	0% (0%)	0% (0%)	0%	0%
0%	50%	18% (15%)	16% (12%)	97%	57%
0%	100%	37% (23%)	34% (20%)	100%	94%
50%	0%	0% (0%)	0% (0%)	0%	0%
50%	50%	15% (11%)	12% (11%)	48%	49%
50%	100%	36% (20%)	31% (18%)	100%	100%
100%	0%	0% (0%)	0% (1%)	0%	14%
100%	50%	18% (14%)	13% (10%)	73%	51%
100%	100%	45% (24%)	33% (19%)	100%	96%

Tabela 6.5: MAXPESOARESTA = 256, MAXPESOVERTICE = 256.

% máx. ruído		% médio de erros		% máx. de erros	
arestas	vértices	var. top. = 5	var. top. = 15	var. top. = 5	var. top. = 15
0%	0%	0% (0%)	0% (0%)	0%	0%
0%	50%	1% (4%)	0% (1%)	28%	15%
0%	100%	2% (6%)	2% (5%)	41%	28%
50%	0%	0% (1%)	0% (0%)	10%	0%
50%	50%	2% (6%)	1% (4%)	34%	23%
50%	100%	8% (11%)	5% (8%)	47%	34%
100%	0%	4% (5%)	1% (3%)	23%	16%
100%	50%	13% (13%)	6% (8%)	63%	43%
100%	100%	24% (18%)	15% (18%)	86%	97%

Tabela 6.6: MAXPESOARESTA = 9600, MAXPESOVERTICE = 256.

% máx. ruído		% médio de erros		% máx. de erros	
arestas	vértices	var. top. = 5	var. top. = 15	var. top. = 5	var. top. = 15
0%	0%	0% (0%)	0% (0%)	0%	0%
0%	50%	0% (0%)	0% (0%)	0%	0%
0%	100%	0% (0%)	0% (0%)	0%	0%
50%	0%	8% (15%)	3% (8%)	100%	37%
50%	50%	10% (13%)	5% (9%)	69%	44%
50%	100%	12% (14%)	6% (10%)	61%	48%
100%	0%	47% (27%)	32% (24%)	100%	100%
100%	50%	43% (28%)	39% (25%)	100%	100%
100%	100%	46% (26%)	39% (25%)	100%	100%

Na tabela 6.4, ficou claro que os ruídos aplicados nas arestas tiveram influência praticamente nula nos resultados. Isto justifica-se pelo fato do valor do parâmetro MAXPESOARESTA ser muito inferior ao valor do parâmetro MAXPESOVERTICE. Já na tabela 6.6, temos a situação inversa. Ou seja, pelo fato do valor do parâmetro MAXPESOARESTA ser muito superior ao valor do parâmetro MAXPESOVERTICE, os ruídos aplicados nos vértices praticamente não influenciaram nos resultados.

Na tabela 6.5 os valores dos parâmetros MAXPESOARESTA e MAXPESOVERTICE são iguais. Podemos notar que neste caso os resultados são muito melhores. Além disso, a influência dos ruídos aplicados tanto nos vértices como nas arestas parecem ter aproximadamente a mesma influência

sobre o resultado. Mas, não concluímos que este equilíbrio sempre ocorre quando MAXPESOARESTA e MAXPESOVERTICE são iguais. Quando G_M é maior, o comportamento pode não ser exatamente o mesmo. Intuitivamente, podemos concluir que existe uma proporção entre estes dois parâmetros de forma que ambos tenham aproximadamente a mesma influência sobre o resultado. Porém, identificar quais são estes valores pode não ser imediato.

Em alguns aspectos os resultados se mantiveram iguais em todas as tabelas. Quanto maior o valor do parâmetro VARIACAO_TOPOLOGIA, melhores foram os resultados. Através da coluna “% máx. de erros”, podemos notar que mesmo quando para determinados parâmetros os resultados na média podem ser considerados satisfatórios, o algoritmo pode produzir resultados ruins em algumas instâncias geradas com os mesmos parâmetros. Os valores de média de erros estão bem próximos aos respectivos valores de desvio padrão. Isto indica que quando os percentuais de erros são grandes, os resultados estão bem dispersos, e quando os percentuais de erros são pequenos, os resultados estão bem concentrados.

O algoritmo demonstrou ser bastante tolerante a ruídos, quando os parâmetros são devidamente ajustados. Na formulação proposta por Boeres [6], é definido um parâmetro para “regular” a influência das associações de vértices para clusters com relação à influência das associações de arestas para pares de clusters. Para incorporar esta idéia, basta fazer uma pequena adaptação nas formulações que apresentamos, ou então fazer um pré-processamento nos grafos de entrada.

6.3 Testes com Instâncias Reais

Nas instâncias reais do problema, os grafos G_I e G_M não são árvores. O que fizemos foi extrair árvores geradoras de G_I e G_M , através de três heurísticas diferentes, sendo elas: extrair uma árvore geradora de custo mínimo de cada grafo; extrair uma árvore geradora de custo máximo de cada grafo; extrair uma árvore geradora aleatoriamente de cada grafo.

Os testes apresentados nesta seção foram feitos no mesmo ambiente descrito na seção 6.1. A maior instância que resolvemos tinha 557 vértices em V_I e 255 clusters em V_M . Nosso algoritmo gastou 43 minutos para resolver esta instância. A seguir, mostramos o resultado visual de um dos testes que fizemos.

Figura 6.4: O resultado visual do teste feito com uma instância real do problema.

A imagem à esquerda serviu como fonte para gerar uma instância com 393 vértices em V_I e 206 clusters em V_M . Nosso algoritmo gastou 11 minutos para resolver esta instância. A heurística utilizada neste teste foi a de extrair árvores geradoras de custo máximo a partir de G_I e G_M . A figura ao centro indica, através das cores, o casamento esperado, baseado no modelo construído para esta imagem. A figura ao lado direito indica, através das cores, o casamento produzido pelo algoritmo PCCA-PARA-ÁRVORES.

Os testes que fizemos não foram suficientes para concluir qual das heurísticas que utilizamos para extrair as árvores geradoras de G_I e G_M apresenta melhores resultados. Dependendo da instância, uma determinada heurística apresenta resultados melhores do que as outras.

Através dos testes apresentados nesta seção, tivemos nosso primeiro contato com instâncias reais do problema. Por isso, consideramos que os resultados foram satisfatórios. Para que os resultados sejam melhorados, é necessário um estudo mais detalhado de como extrair árvores geradoras de G_I e G_M de forma que os dois grafos continuem tendo basicamente a mesma estrutura.

Outro aspecto importante para a melhoria dos resultados é a necessidade de que investiguemos mais afundo algumas questões do domínio da aplicação, como por exemplo: que atributos, além do nível médio de cinza, podem ser utilizados para atribuir pesos aos vértices; que atributos, além da distância, podem ser utilizados para atribuir pesos às arestas. Estas questões não foram foco deste trabalho. Como mencionamos, nosso foco foi o de estudar o PCCA como um problema de otimização combinatória, sem assumir nenhum compromisso com a aplicação.

As instâncias que utilizamos nos testes mostrados nesta seção foram cedidas pelo Alexandre Noma¹, que detalha em [39] como a instância que aparece na figura 6.4 foi obtida (como os grafos G_I e G_M foram gerados e como o casamento mostrado na figura 6.4 ao centro foi obtido).

¹Agradecemos ao Alexandre Noma, ao Roberto Marcondes Cesar Jr. e à Ana Beatriz Vicentim Graciano, que foram fundamentais para a realização destes testes.

Capítulo 7

Conclusão e Trabalhos Futuros

Neste capítulo, apontamos os principais tópicos abordados nesta dissertação e mostramos algumas questões que ainda necessitam de um estudo mais profundo.

7.1 Aspectos Importantes com Relação ao PCIG

A seguir, fazemos algumas considerações sobre cinco aspectos importantes com relação ao PCIG. Com relação a cada um destes aspectos, comentamos o que encontramos na literatura, o que obtivemos durante o mestrado e o que ainda precisa ser investigado.

Complexidade computacional

Não encontramos nada na literatura a respeito da complexidade do PCIG. Apresentamos alguns resultados interessantes, sendo que o principal deles é a proposição 4.1.1, na qual provamos que o PCCA *para árvores* é $\mathcal{NP}$ -difícil. Não estudamos a complexidade de todas as variações do PCIG. Será que todas as variações do PCIG são difíceis?

Formulações

Encontramos algumas formulações do PCIG (e algumas de suas variações) na literatura [6, 16]. Utilizamos algumas idéias contidas em tais formulações (utilização de fluxo em redes para tratamento da restrição de conexidade) para desenvolver a nossa formulação para o PCCA. Introduzimos uma formulação específica para o PCCA *para árvores*, que tira proveito de características específicas deste caso particular do PCIG. Tal formulação é muito mais simples e compacta que as demais. Como

continuação do trabalho que fizemos, um caminho natural seria associar poliedros a esses problemas, depois estudar tais poliedros utilizando ferramentas de combinatória poliédrica [1, 2, 20, 29, 46, 50];

Algoritmos exatos e heurísticas

Encontramos na literatura apenas heurísticas para o PCIG (e suas variações) [6, 16, 30]. Fornecemos um algoritmo exato (que é polinomial se o grau máximo dos vértices de V_M for limitado por uma constante) para o PCCA *para árvores*, e um algoritmo (exato) polinomial para um caso especial do PCCA *para árvores*. Uma pergunta natural que surge é: será que existem mais casos especiais para os quais é possível obter um algoritmo polinomial? Um outro ponto a ser explorado seria o desenvolvimento de novas heurísticas que se utilizassem dos algoritmos que apresentamos. Tais heurísticas poderiam extrair uma árvore geradora de cada um dos grafos de entrada (exemplo: árvore geradora de custo mínimo ou máximo) e ignorar as demais arestas desses grafos. Outra possibilidade seria desenvolver uma heurística que extraísse uma árvore geradora de G_M e uma estrela estendida geradora de G_I . Neste último caso, o desempenho do algoritmo PCCA-PARA-ÁRVORES, modificado conforme explicamos na seção 5.2, seria muito melhor.

Algoritmos de aproximação

Não obtivemos nenhum algoritmo de aproximação para nenhuma das variações do PCIG, e também não encontramos nada na literatura a respeito disso. Certamente, algum resultado neste sentido seria bastante interessante. Possivelmente, os resultados apresentados aqui (formulações e/ou algoritmos) podem servir como ponto de partida para desenvolver algum algoritmo de aproximação para o PCCA *para árvores*.

Geradores de grafos

Dados um real positivo t e um grafo G com pesos positivos nas arestas, considere o seguinte problema: encontrar um subgrafo esparso G' de G , tal que para todo par $u, v \in G$, vale que $d_{G'}(x, y) \leq td_G(x, y)$, sendo que $d_G(x, y)$ denota a distância entre x, y em G .

Dizemos que G' é um t -gerador. O número t é chamado de fator de ampliação, e dá uma medida da qualidade da aproximação que G' fornece para as distâncias em G . É claro que G é um 1-gerador de si próprio. O objetivo em estudar este problema é encontrar um grafo G' que seja o mais esparso possível (ou que otimize alguma função objetivo).

Existem diversos resultados na literatura sobre geradores de grafos (graph spanners) [42, 47]. Este problema parece ter grande utilidade no desenvolvimento de novos métodos de resolução (exata ou aproximada) do PCIG. Os grafos de entrada seriam submetidos a um pré-processamento que os tornariam mais esparsos (preferencialmente árvores), mas tentando manter ao máximo suas estruturas originais (isso dependeria do fator de ampliação).

Aplicação em reconhecimento estrutural de objetos

Mostramos que o algoritmo PCCA-PARA-ÁRVORES resolve instâncias reais do problema em tempo relativamente curto. Conforme dissemos, através dos testes apresentados na seção 6.3, tivemos nosso primeiro contato com instâncias reais do problema. Portanto, ainda não estamos aptos a tirar muitas conclusões sobre o que pode ser melhorado. Nos textos [6, 16, 24, 30, 39], podem ser encontrados mais detalhes sobre esta aplicação.

Por questões de desempenho, talvez o que seja mais adequado para um determinado tipo de aplicação seja o uso de heurísticas. Mesmo assim, a existência de um algoritmo exato para resolver o problema pode ser muito útil na avaliação de resultados produzidos por heurísticas (através da comparação entre os resultados produzidos das duas formas).

Como trabalhos futuros, visualizamos duas direções. A primeira seria o aperfeiçoamento das heurísticas para extrair árvores geradoras de G_I e G_M . A segunda seria trabalhar em cima das formulações que apresentamos para tentar resolver instâncias “grandes” do caso geral do problema (ou seja, quando G_I e G_M não são árvores).

Referências Bibliográficas

- [1] K. Aardal e C.P.M. van Hoesel. Polyhedral techniques in combinatorial optimization i: Theory. *Statistica Neerlandica*, 50:3–26, 1996.
- [2] K. Aardal e C.P.M. van Hoesel. Polyhedral techniques in combinatorial optimization ii: Applications and computations. *Statistica Neerlandica*, 53:131–177, 1999.
- [3] V. Arvind e Piyush P. Kurur. Graph isomorphism is in spp. *Proceedings of the 43rd Symposium on Foundations of Computer Science*, pages 743–750, 2002.
- [4] R. Becker, I. Lari, M. Lucertini, e B. Simeone. Max-min partitioning of grid graphs into connected components. *Networks*, 32(2):115–125, 1998.
- [5] R. Becker, I. Lari, M. Lucertini, e B. Simeone. A polynomial-time algorithm for max-min partitioning of ladders. *Theory of Computing Systems*, 34(4):353–374, 2001.
- [6] M.C.S. Boeres. *Heurísticas para reconhecimento de cenas por correspondência de grafos*. PhD thesis, Universidade Federal do Rio de Janeiro, 2002.
- [7] P. Camerini, G. Galbiati, e F. Maffioli. On the complexity of finding multi-constrained spanning trees. *Discrete Applied Mathematics*, 5:39–50, 1983.
- [8] S. Caminiti, I. Finocchi, e R. Petreschi. A unified approach to coding labeled trees. In *Proceedings of 6th Latin American Symposium on Theoretical Informatics*, pages 339–348, 2004.
- [9] H. C. Chen e Y. L. Wang. An efficient algorithm for generating prüfer codes from labelled trees. *Theory Computing Systems*, pages 97–105, 2000.
- [10] S. Chopra e M. R. Rao. The partition problem. *Mathematical Programming*, 59:87–115, 1993.
- [11] V. Chvátal. *Linear Programming*. W. H. Freeman, 1983.
- [12] M. Conforti, M. Rao, e A. Sassano. The equipartition polytope i. *Mathematical Programming*, pages 49–70, 1990.

- [13] M. Conforti, M. Rao, e A. Sassano. The equipartition polytope ii. *Mathematical Programming*, pages 71–90, 1990.
- [14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, e C. Stein. *Introduction to Algorithms*. MIT Press, 2001.
- [15] R. Czerwinski. A polynomial time algorithm for graph isomorphism. Technical report, Institut für Softwaretechnik und Theoretische Informatik, feb 2008. arXiv:0711.2010v4 [cs.CC].
- [16] A.R. Duarte. Novas heurísticas e uma abordagem por programação inteira para um problema de correspondência inexata de grafos. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2004.
- [17] M. Dyer e A. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10:139–153, 1985.
- [18] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, e L.A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *CORE Discussion Paper 9437*, Université Catholique de Louvain, Louvain-la-Neuve, 1994.
- [19] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, e L.A. Wolsey. The node capacitated graph partitioning problem: A computacional study. *Mathematical Programming*, 81:229–256, 1998.
- [20] C.E. Ferreira e Y. Wakabayashi. *Combinatória Poliédrica e Planos-de-Corte Faciais*. UNICAMP - Instituto de Computação, 1996.
- [21] S. Fortin. The graph isomorphism problem. Technical report, University of Alberta, july 1996.
- [22] G. Galbiati, F. Maffioli, e A. Morzenti. On the approximability of some maximum spanning tree problems. *Theoretical Computer Science*, 181(1):107–118, 1997.
- [23] M.R. Garey e D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [24] A.B.V. Graciano. Rastreamento de objetos baseado em reconhecimento estrutural de padrões. Master's thesis, Universidade de São Paulo - Instituto de Matemática e Estatística, 2007.
- [25] M. Grötschel, L. Lovász, e A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.
- [26] M. Grötschel e Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989.

- [27] M. Grötschel e Y. Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1-3):367–387, 1990.
- [28] J.E. Hopcroft e J.K. Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184, New York, NY, USA, 1974. ACM Press.
- [29] M. Jünger, G. Reinelt, e S. Thienel. Pratical problem solving with cutting plane algorithms in combinatorial optimization. *Angewandte Mathematik und Informatik, Universität zu Köln*, Report No. 94.156, 1994.
- [30] R.M. Cesar Jr., E. Bengoetxea, I. Bloch, e P. Larrañaga. Inexact graph matching for model-based recognition: Evaluation and comparison of optimization algorithms. *Pattern Recognition*, 38:2099–2113, 2005.
- [31] W. Kernighan e S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, pages 291–307, 1970.
- [32] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, pages 83–97, 1955.
- [33] L. Lovász. *Combinatorial Problems and Exercises*. North-Holland Publishing Co., 1979.
- [34] R.P.F.L. Lucindo. Partição de grafos em subgrafos conexos balanceados. Master's thesis, Universidade de São Paulo – Instituto de Matemática e Estatística, 2007.
- [35] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, pages 42–65, 1982.
- [36] A. Makhorin. Glpk (gnu linear programming kit). Technical report, Moscow Aviation Institute, 2003. <http://www.gnu.org/software/glpk/>.
- [37] L. Markenzon, O. Vernet, e P. R. Pereira. Codificação de Árvores: Conceitos e algoritmos. Technical report, 2005.
- [38] Sun Microsystems. Java platform, standard edition 6 release. Technical report, 2008. <http://java.sun.com/javase/6/>.
- [39] A. Noma, A. B. V. Graciano, L. A. C., R. M. Cesar-Jr, e I. Bloch. A new algorithm for interactive structural image segmentation. Technical report, may 2008. arXiv:0805.1854v2 [cs.CV].
- [40] Y. Wakabayashi P. Feofiloff, Y. Kohayakawa. *Uma Introdução Sucinta à Teoria dos Grafos*. II Bienal da SBM (Sociedade Brasileira de Matemática), 2004.

- [41] M. Vento P. Foggia, C. Sansone. A database of graphs for isomorphism and sub-graph isomorphism benchmarking. Technical report, Dipartimento di Informatica e Sistemistica - Università di Napoli “Federico II”, 2001.
- [42] D. Peleg e A. A. Schäffer. Graph spanners. *Journal Graph Theory*, pages 99–116, 1989.
- [43] Y. Perl e S. Schach. Max-min tree partitioning. *Journal of the ACM*, 28(1):5–15, 1981.
- [44] K. Saarinen. Color image segmentation by a watershed algorithm and region adjacency graph processing. *Image Processing, IEEE International Conference*, 1994.
- [45] L.R.B. Salgado. Algoritmos de aproximação para partições conexas em grafos. Master’s thesis, Universidade de São Paulo – Instituto de Matemática e Estatística, 2003.
- [46] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.
- [47] J. Soares. Graph spanners: a survey. *Congresus Numeratium*, pages 225–238, 1992.
- [48] E. Tengan. Automorfismo de grafos. Master’s thesis, Universidade de São Paulo – Instituto de Matemática e Estatística, 2002.
- [49] Y. Victorovich e B. Frank. Glpk 4.8 java interface. Technical report, 2003. <http://bjoern.dapnet.de/glpk/index.htm>.
- [50] L.A. Wolsey. *Integer Programming*. Wiley-Interscience Publication, 1998.