Um Estudo Computacional sobre o Problema de Decomposição de Grafos em Árvore

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Ana Shirley Ferreira da Silva e aprovada pela Banca Examinadora.

Fortaleza, 31 de Agosto de 2005.

Cláudia Linhares Sales (MDCC/UFC)

Dissertação apresentada ao programa Mestrado e Doutorado em Ciência da Computação, UFC, como requisito para a obtenção do título de Mestre em Ciência da Computação.

Mestrado e Doutorado em Ciência da Computação (MDCC)
Universidade Federal do Ceará (UFC)

Um Estudo Computacional sobre o Problema de Decomposição de Grafos em Árvore

Ana Shirley Ferreira da Silva
shirley@lia.ufc.br

Agosto de 2005

Banca Examinadora:

- Cláudia Linhares Sales (MDCC/UFC)
- Ricardo Cordeiro Corrêa (MDCC/UFC)
- Rafael de Castro Andrade (MDCC/UFC)
- Jayme Luiz Szwarcfiter (COPPE/UFRJ)

Sumário

Lista de Tabelas

Lista de Figuras

1

Introdução

A noção de largura em árvore, assim como a de decomposição em árvore, foi introduzida por Robertson e Seymour [55] ao longo de sua série de artigos sobre a teoria dos menores publicados nas décadas de 80 e 90. Logo após a introdução destes novos conceitos, notou-se a equivalência entre esse e vários outros conceitos já conhecidos e amplamente estudados, como por exemplo, o de triangularização e o de k-árvore parcial. Além disso, provou-se a influência da decomposição sobre a complexidade dos problemas, constatando-se que muitos problemas difíceis poderiam ser resolvidos polinomialmente dado que uma decomposição em árvore de largura limitada fosse fornecida. Dentre tais problemas estão alguns bastante conhecidos como: Conjunto Independente, Ciclo Hamiltoniano, Coloração de Vértices, Isomorfismo, Árvore de Steiner, entre outros. Muitos deles poderiam, na verdade, ser resolvidos linearmente dada tal decomposição. Existem ainda alguns importantes resultados acerca da complexidade de certos problemas usando lógica monádica de segunda ordem. Foi mostrado, basicamente, que qualquer problema que possa ser representado através de determinadas linguagens formais ou expressões lógicas, podem ser resolvidos polinomialmente dada uma decomposição de largura limitada. Dentre os vários trabalhos sobre esta perspectiva citamos Courcelle [24, 25, 26, 27], Arnborg et al.[5, 27], Borie et al. [17] e Courcelle e Mosbah [28].

Percebe-se, então, que encontrar uma decomposição de largura limitada torna-se parte

dominante na complexidade dos algoritmos que se utilizam destes fatos. Além disso, vale ressaltar que, de uma forma geral, tais algoritmos têm complexidade exponencial nesta largura. Daí a importância de obtermos uma decomposição com menor largura possível.

É de se esperar, então, que calcular a largura em árvore de um grafo qualquer seja um problema difícil. De fato, Arnborg, Corneil e Proskurowski [4] mostraram que determinar a largura em árvore de um grafo qualquer pertence a $\mathcal{NP}$-difícil. Porém, apesar desta complexidade, foi mostrado que um outro problema, derivado desse, pertence a $\mathcal{P}$. Tal problema é o de decidir, para k fixo, se um grafo $G = (V, E)$ qualquer possui largura em árvore menor ou igual a k.

Os primeiros algoritmos polinomiais para o problema com k fixo foram propostos, independentemente, por Arnborg, Corneil e Proskurowski [4] e por Ellis, Sudborough e Turner [32] e têm complexidade $O(n^{k+2})$ e $O(n^{2k^2+4k+8})$, respectivamente, onde $n = |V|$. Estes algoritmos, além de responder "sim" ou "não", também são construtivos, ou seja, também fornecem uma decomposição em árvore de largura ótima ao final.

Robertson e Seymour [55], baseados na sua vasta e complexa teoria acerca de menores, deram uma prova não-construtiva da existência de um algoritmo em duas fases que resolve o problema em $O(n^2)$. A primeira fase do algoritmo ou decide que a largura do grafo é maior do que k ou fornece uma decomposição de largura no máximo $4k$. A segunda utiliza a decomposição encontrada pela primeira para decidir se a largura é realmente menor ou igual a k.

Ainda no mesmo trabalho, Robertson e Seymour propuseram um algoritmo baseado no conceito de separadores que resolve a primeira fase em $O(n^2)$ e mostraram a existência de um algoritmo para a segunda fase de complexidade linear. Reed [54], modificando o algoritmo de Robertson e Seymour de forma a utilizar separadores aproximados, obteve um algoritmo para a primeira fase de complexidade $O(n \log n)$. Por ter utilizado uma aproximação para encontrar os separadores, a largura da decomposição encontrada é um pouco maior, mas há o benefício da melhor complexidade. Lagergren [46] propôs um algoritmo semelhante que, apesar de mais lento (complexidade $O(n \log^2 n)$), pode ser

aplicado em paralelo com complexidade de tempo $O(\log^3 n)$ utilizando $O(n)$ processadores em um CRCW PRAM.

A prova da existência da segunda fase do algoritmo foi baseada no resultado sobre menores proibidos obtido por Robertson e Seymour que afirma que toda classe de grafos fechada sobre menores possui um conjunto finito de menores proibidos. Bastou provar, então, que a classe de grafos cuja largura em árvore é menor ou igual a k é fechada sobre menores. A linearidade do algoritmo advém de resultados que propõem que, dada a decomposição em árvore de largura limitada, é possível verificar linearmente a relação de menor [7, 33].

Vê-se que tal prova é não construtiva sob dois aspectos: além de não fornecer o conjunto de menores proibidos propriamente dito, também não fornece uma decomposição ótima ao final. Com a ajuda de uma técnica de auto-redução ("self-reduction"), introduzida por Fellows e Langston [33], é possível obter um algoritmo construtivo de mesma complexidade amortizada, mas com um grande aumento nas constantes [11]. Porém, Lagergren e Arnborg[48], Abrahamson e Fellows [2] e Bodlaender e Kloks [14] propuseram, independentemente, algoritmos construtivos para a segunda fase. Dentre eles, o de Bodlaender e Kloks é o único para o qual se sabe que foi feito um estudo computacional (Röhrig [56]). Infelizmente, neste estudo Röhrig conclui que o algoritmo não tem um bom desempenho mesmo para k pequeno (por exemplo, $k = 4$). Supõe-se que os outros, apesar de não terem sido estudados na prática, também não possuam bom desempenho computacional, visto que possuem constantes escondidas na notação O ainda maiores do que aquelas do algoritmo de Bodlaender e Kloks. Na verdade, mesmo que as constantes fossem menores a ponto de tornar tais algoritmos computacionalmente viáveis, na prática não são tão úteis, visto que podem apenas retornar a resposta "não" ao fim dos cálculos, quando a largura em árvore do grafo pode estar a uma pequena distância da constante k fixa.

Uma alternativa para o problema é a utilização de algoritmos aproximativos. O primeiro algoritmo aproximativo proposto para o problema foi o de Bodlaender , Gilbert,

Hafsteinsson e Kloks [13] cuja aproximação fornecida é de $O(\log n)$ (veja também [42]). Bouchitté et al. [19] e Amir [3] melhoraram, independentemente, a razão de aproximação para $O(\log k)$, onde $k = la(G)$. Todos estes têm complexidade polinomial. Não se sabe, até então, se existe um algoritmo aproximativo polinomial cuja razão de aproximação seja uma constante. Porém, existem algoritmos de complexidade exponencial em k que fornecem uma razão constante [3, 9, 30, 47, 54, 55]. Eles funcionam, basicamente, como a primeira fase do algoritmo de Robertson e Seymour para o problema com k fixo, com a diferença de que k é dado como entrada: utilizando o conceito de separadores para construir uma decomposição em árvore, ou decidem que a largura é maior do que k ou retornam uma decomposição de largura no máximo ck, para uma determinada constante c. Esta constante c geralmente depende do tamanho máximo dos separadores utilizados, enquanto que a complexidade geralmente depende do algoritmo utilizado para encontrar os separadores. Utilizando também o conceito de separadores, porém sobre a abordagem de eliminação Gaussiana, Bornstein [18] apresenta um algoritmo cujo fator de aproximação é $O(\log^2 n)$.

Outras duas alternativas para o problema são: heurísticas para o cálculo de limites superiores ou métodos enumerativos. No que diz respeito ao cálculo de limites superiores, existem vários métodos na literatura (os mais conhecidos são apresentados no capítulo 5), muitos dos quais abordam o problema sob a perspectiva de triangularização. Porém, existe uma dificuldade na análise da qualidade destes métodos pois os limites superiores encontrados são, geralmente, muito distantes dos valores fornecidos pelas heurísticas de limites inferiores existentes (as principais são apresentadas no capítulo 6). Devido a isto, optamos pelo método enumerativo. Para isso utilizamos também a perspectiva de triangularização para abordar o problema. Ressaltamos que o algoritmo de segunda fase proposto por Bodlaender e Kloks [14] também pode ser utilizado para enumerar as soluções do problema, porém sabemos, graças ao estudo de Röhrig [56], que o algoritmo é computacionalmente inviável mesmo para k fixo e com valores pequenos (por exemplo, para k igual a 4). Além disso, este é o único método enumerativo do qual se tem

conhecimento.

A principal contribuição desta dissertação é a proposta de um método enumerativo para o problema de largura em árvore. Mais especificamente, propomos: 1) uma representação para as soluções do problema que utiliza ordens parciais (esta mesma representação é também utilizada para definir espaços de soluções); 2) um método de enumeração do espaço de soluções através da adaptação do algoritmo proposto por Corrêa e Szwarcfiter [23] para enumeração de todas as extensões de uma ordem parcial; e 3) uma forma de utilizar a enumeração para calcular um limite inferior e para explorar sub-espaços de solução.

O método enumerativo para o problema proposto nesta dissertação utiliza o conceito de ordens parciais. Inicialmente, a idéia era estudar o espaço de soluções definido por determinadas ordens parciais visando inferir quais espaços continham soluções melhores ou até ótimas, ou seja, qual o melhor a "caminho" seguir quando da ordenação dos vértices. Porém, como veremos no Capítulo 8, a falta de um bom método para calcular um limite inferior estreita a viabilidade de utilização da enumeração proposta. Ressaltamos que, apesar de não havermos obtido bons resultados com a enumeração, a abordagem do problema através de ordens parciais, também proposta no presente trabalho, ainda pode ser explorada para caracterizar espaços de soluções.

Além disso, propomos também a utilização da meta-heurística GRASP para a obtenção de limites superiores. Apesar de não ser o foco principal do trabalho, os estudos feitos mostram que a heurística GRASP pode ser uma boa opção para o cálculo de um limite superior. O trabalho inclui, também, um estudo experimental das principais heurísticas para obtenção de limites superiores.

O texto está organizado da seguinte forma. No Capítulo 2, definimos os principais conceitos e introduzimos a notação utilizada. No Capítulo 3, tratamos da complexidade dos problemas abordados (decomposição em árvore, decomposição em árvore com k fixo e triangularização). O Capítulo 4 expõe uma forma de representação de uma solução do problema de triangularização (utilizada pela maioria das heurísticas para limites su-

periores conhecidas), assim como um algoritmo exato para encontrar uma decomposição em árvore de um grafo triangularizado. Nos Capítulos 5 e 6 vemos as heurísticas mais conhecidas na literatura para calcular limites superiores e inferiores, respectivamente, assim como propomos no Capítulo 5 a aplicação da meta-heurística GRASP para obtenção de limites superiores. No Capítulo 7, propomos uma representação de uma solução para o problema que utiliza ordens parciais e um método enumerativo baseado no algoritmo proposto por Corrêa e Szwarcfiter [23]. Finalmente, o Capítulo 8 trata dos experimentos e da análise desses e o Capítulo 9 das conclusões e trabalhos futuros.

2

Definições

Neste capítulo, vemos os conceitos necessários para o entendimento do texto. Na verdade, é desejável que o leitor esteja familiarizado com os conceitos mais básicos da Teoria dos Grafos. Porém estes estão apresentados aqui a fim de suprir alguma carência, caso exista. Ao longo do texto, novos conceitos são introduzidos à medida que forem necessários.

Na Seção 2.1, vemos algumas definições acerca de grafos, além de enunciarmos a Propriedade de Helly aplicada a árvores. Na Seção 2.2 definimos os problemas de decomposição em árvore, assim como o problema de triangularização, sobre o qual iremos trabalhar mais diretamente. Finalmente, na Seção 2.3 introduzimos alguns conceitos que serão utilizados nas heurísticas vistas nos Capítulos 5 , assim como no método enumerativo visto no Capítulo 7.

2.1. Grafos

Definição 1. *Um* ***grafo*** G *é um par ordenado de conjuntos* (V, E)*, onde* $E \subseteq V^2$*. Chamamos de* ***vértices*** *os elementos do conjunto* V *e de* ***arestas*** *os elementos do conjunto* E*.*

Daqui em diante denotaremos $|V|$ por n e $|E|$ por m, caso não hajam ambigüidades.

No caso contrário, denotaremos o conjunto de vértices de um grafo G por $V(G)$ e conjunto de arestas por $E(G)$, e para a cardinalidade utilizaremos a notação matemática tradicional ($|V(G)|$ e $|E(G)|$).

Dizemos que dois vértices u e v são **adjacentes** se a aresta $e = (u, v) \in E$. Dizemos ainda que u e v são as **extremidades** da aresta e e que a aresta e é **incidente** aos vértices u e v.

A **vizinhaça** de v é o conjunto $N(v) = \{u \in V : (u, v) \in E\}$, enquanto que a **vizinhança fechada** de v é $N[v] = N(v) \cup \{v\}$. Chamamos de **grau** de v o valor $|N(v)|$ e denotamos por $d(v)$. O mínimo grau dentre todos os vértices do grafo é chamado de **grau mínimo de** G e é denotado por $\delta(G)$. Definimos o **grau máximo de** G analogamente e denotamos por $\Delta(G)$.

Se G é tal que $N[v] = V$, para todo $v \in V$, dizemos que G é um grafo **completo**.

Um **subgrafo** de $G = (V, E)$ é um grafo $G' = (V', E')$ tal que $V' \subseteq V$ e $E' \subseteq E$. Dizemos que G' é **próprio** se $G' \neq G$, e que é **trivial** se $G' = (\{v\}, \emptyset)$. Se G' é tal que $(u, v) \in E'$ se e somente se $\{u, v\} \subseteq V'$ e $(u, v) \in E$ dizemos que G' é o **subgrafo induzido** por V' em G. Denotamos um subgrafo de G induzido por um conjunto de vértices V' por $G[V']$.

Um **supergrafo** de $G = (V, E)$ é um grafo $G' = (V', E')$ tal que $V \subseteq V'$ e $E \subseteq E'$.

Seja $C \subseteq V$, se $G[C]$ é um grafo completo, dizemos que C é uma **clique** de G. O tamanho da maior clique de um grafo G é denotado por $\omega(G)$.

Se $N(v)$ é uma clique, dizemos que o vértice v é um vértice **simplicial**.

Definimos um **caminho** entre dois vértices u e v como sendo uma seqüência de vértices distintos $\langle v_1 = u, v_2, ..., v_q = v\rangle$ tal que $(v_i, v_{i+1}) \in E$, para $i = 1, ..., q - 1$. Um **caminho sem cordas** é tal que o subgrafo induzido pelos vértices do caminho possui apenas arestas do caminho, ou seja, $E(G[\{v_1, ..., v_q\}]) = \{(v_i, v_{i+1}) : 1 \leq i < q\}$.

Um grafo é dito **conexo** se para todo par de vértices u e v existe um caminho entre eles. Uma **componente conexa** de G é um subgrafo induzido conexo maximal em G, ou seja, um subgrafo conexo G' de G tal que não existe nenhum outro subgrafo conexo

de G, G'', tal que G' é subgrafo próprio de G''.

Se $C \subseteq V$ é tal que $G[V-C]$ tem mais componentes conexas do que o grafo G dizemos que C é um **corte** de G. Denotaremos $G[V-C]$ simplesmente por $G \setminus C$.

Um **ciclo** é uma seqüência de vértices $\langle v = v_1, v_2, ..., v_k = v \rangle$, com $k \geq 3$, tal que $(v_i, v_{i+1}) \in E$, para todo $1 \leq i < k$, e $v_i \neq v_j$, para todo $2 \leq i < j \leq k$. Se $k = 2$, v é adjacente a ele mesmo. Chamamos a isso de **laço**. Supomos, daqui em diante, que os grafos mencionados não possuem laços. De forma análoga à definição de caminho sem cordas, definimos um **ciclo sem cordas** como um ciclo tal que o subgrafo induzido pelos vértices do ciclo possui apenas arestas do ciclo.

A um grafo G sem ciclos chamamos de **acíclico**.

Um grafo T é uma **árvore** se é acíclico e conexo. Chamamos de **subárvore** de T a qualquer subgrafo de uma árvore T que também é uma árvore, ou seja, que é conexo.

Existem algumas propriedades interessantes a respeito de árvores. A mais básica delas é a de que uma árvore com n vértices possui exatamente $n-1$ arestas. A seguir enunciamos a propriedade de Helly e em seguida o lema que enuncia que esta propriedade pode ser aplicada a árvores.

Propriedade de Helly *Uma família $\mathcal{F}$ de subconjuntos de um conjunto S satisfaz a propriedade de Helly se toda subfamília $\mathcal{F}'$ de $\mathcal{F}$ satisfaz:*

$$S' \cap S'' \neq \emptyset, \text{ para todo par } S', S'' \in \mathcal{F}' \Rightarrow \bigcap_{S' \in \mathcal{F}'} S' \neq \emptyset$$

Lema 2.1.1. *Dada uma árvore T, temos que qualquer família $\mathcal{F}$ de subárvores de T satisfaz a propriedade de Helly.*

2.2. Problemas

O conceito que veremos a seguir é como que uma tentativa de organizar um grafo qualquer em uma estrutura de árvore a fim de resolver alguns problemas facilmente.

Definição 2. *Uma **decomposição em árvore** de G é uma dupla $D = (\mathcal{X}, T)$ onde*

$T = (I, F)$ é uma árvore e $\mathcal{X} = \{X_i | \ i \in I\}$ é uma família de subconjuntos de $V(G)$ relacionados aos vértices de T. Além disso:

1. $\bigcup_{i \in I} X_i = V(G)$;
2. *para toda aresta $(u, v) \in E(G)$, existe $i \in I$ tal que $\{u, v\} \subseteq X_i$;*
3. *para i, j e k em I, se j está no (i, k)-caminho em T, então $X_i \cap X_k \subseteq X_j$.*

*T é chamada de **árvore de decomposição**. A **largura** de D é dada por $max_{i \in I} |X_i| - 1$ e é denotada por $la(D)$.*

Definição 3. *A **largura em árvore** de um grafo G, denotada por $la(G)$, é igual à menor largura dentre todas as decomposições em árvore de G.*

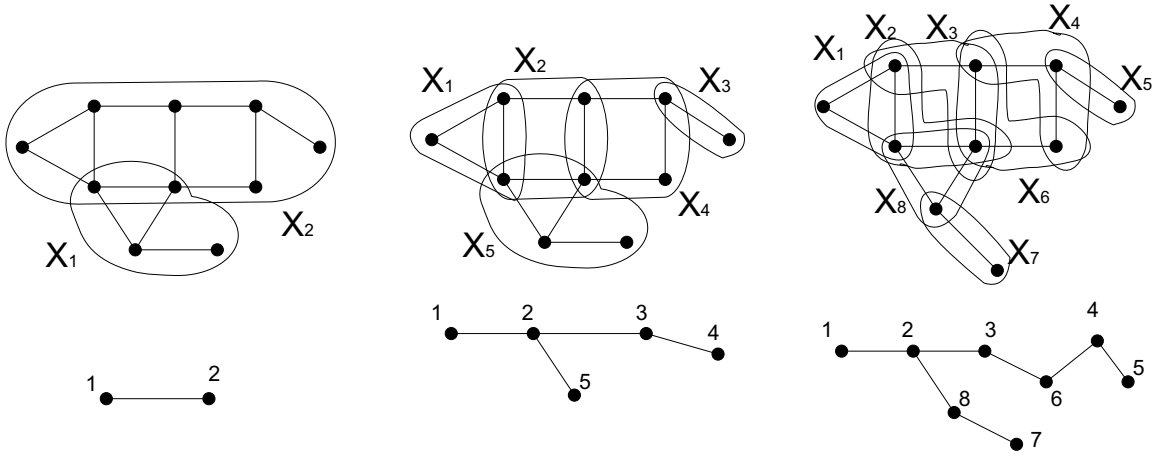

Figura 2.1. *Exemplo de decomposições em árvore.*

Daqui em diante, nos referiremos aos vértices da árvore de decomposição como **nós**, e simplesmente como vértices aos vértices do grafo de entrada.

A seguir definimos os problemas que serão abordados ao longo do texto.

Problema 1. *(**DEA**) Dados um grafo G qualquer e um inteiro k positivo, responder se $la(G) \leq k$.*

Problema 2. *(**DEA com** k **fixo**)Para um inteiro k fixo, decidir, dado um grafo $G = (V, E)$ qualquer, se $la(G) \leq k$.*

Robertson e Seymour [55] provaram que DEA com k fixo pode ser resolvido em tempo $O(n^2)$. No próximo capítulo, damos uma explicação geral de como isso é feito. Apesar da complexidade obtida para DEA com k fixo, DEA continua difícil devido às constantes envolvidas, exponenciais em k.

A seguir, definimos um problema equivalente a DEA sobre o qual iremos trabalhar.

Definição 4. *Um grafo é dito **triangularizado** (ou cordal), se todo ciclo de tamanho maior ou igual a quatro possui pelo menos uma corda.*

O seguinte resultado será bastante utilizado e devido à sua simplicidade, optamos por enunciá-lo neste capítulo:

Lema 2.2.1. *Se G é cordal, então $G[V - \{v\}]$ também o é, para todo vértice $v \in V$.*

Chama-se uma **triangularização** (ou cordalização) de $G = (V, E)$ a um supergrafo cordal $G' = (V, E \cup E')$ de G. A **triangularização mínima** de um grafo é tal que o número de arestas adicionadas é mínimo. Porém, como veremos no capítulo 3, a largura em árvore de um grafo triangularizado G' é igual a $\omega(G') - 1$. Logo, estamos mais interessados em encontrar a triangularização de G cujo tamanho da maior clique é mínimo. Denotaremos por G_{MIN} uma tal triangularização e definimos o problema a seguir também levando em conta o valor $\omega(G_{MIN})$.

Problema 3. *(**Triang**) Dados um grafo G qualquer e um inteiro k positivo, decidir se a triangularização mínima G_{MIN} de G é tal que $\omega(G_{MIN}) \leq k$.*

2.3. Ordens Parciais

A seguir definimos conceitos relacionados a ordenação dos vértices de G. Mais adiante, no Capítulo 4, vemos a relação entre esses conceitos e o problema de triangularização. Por enquanto, nos restringiremos às definições.

Definição 5. *Uma **ordem parcial** sobre um conjunto finito C é uma relação binária sobre C que é também reflexiva, transitiva e anti-simétrica.*

Daqui em diante, denotaremos o par ordenado (a, b) simplesmente por ab.

Seja $P \subseteq C^2$ uma ordem parcial. Se $ab \in P$, dizemos que a precede b em P.

Dizemos que dois elementos a e b de C são **comparáveis** em P se $ab \in P$ ou $ba \in P$ (note que este ou é exclusivo uma vez que P é anti-simétrica). Caso contrário, os elementos a e b são ditos **incomparáveis**. O conjunto de pares ordenados incomparáveis em P será denotado por $I(P)$. Note que $ab \in I(P)$ se e somente se $ba \in I(P)$.

Se $I(P) = \emptyset$ (ou seja, cada dois elementos de C são comparáveis em P), dizemos que P é uma **ordem total** sobre os elementos de C. Note que uma ordem total $P = \{c_i c_j : i \leq j\}$ pode ser vista também como uma seqüência $\langle c_1, ..., c_q \rangle$, onde $q = |C|$.

Definição 6. *O **fecho transitivo** de um par xy em uma ordem parcial P é dado por:*

$$fecho(P, uv) = \begin{cases} \{xy : x \in X, y \in Y\}, & \text{se } uv \in I(P) \\ \emptyset, & \text{c.c.} \end{cases} \tag{2.1}$$

onde $X = \{x : xu \in P \text{ e } xv \in I(P)\}$ e $Y = \{y : vy \in P \text{ e } uy \in I(P)\}$.

Note que se $uv \in I(P)$, então $u \in X$ e $v \in Y$, uma vez que a relação é reflexiva. Logo, temos que $fecho(P, uv) = \emptyset$ se e somente se $uv \notin I(P)$.

Uma ordem parcial P' sobre C é uma **extensão** de P se $P \subseteq P'$. Caso P' seja, simultaneamente, uma extensão de P e uma ordem total sobre os elementos de C, então P' é uma **extensão total** de P (ou extensão linear de P). O conjunto das extensões de P é denotado por $Ext(P)$ e o conjunto das extensões totais de P, por $Ext_t(P)$.

Um elemento $c \in C$ é **minimal** em P se não existe $c' \in C$ tal que $c'c \in P$. Denotamos por $Min(P)$ o conjunto dos elementos minimais em P. Note que $Min(P)$ sempre será um conjunto não vazio (a menos que $C = \emptyset$). Note ainda que se P é uma ordem total sobre os elementos de C, então existirá exatamente um elemento minimal em P (caso contrário, sejam a, $b \in Min(P)$, certamente a e b são incomparáveis em P , absurdo pois $I(P) = \emptyset$ já que P é uma ordem total).

Sejam P uma ordem parcial sobre C e $C' \subseteq C$. A **ordem parcial induzida por C' em** P é a ordem $P[C']$ tal que $uv \in P[C']$ se e somente se $uv \in P$ e $\{u, v\} \subseteq C'$. Note que $P \in Ext(P[C'])$.

Os conceitos vistos até aqui dizem respeito somente a ordens parciais. A seguir, introduzimos alguns conceitos que relacionam uma ordem parcial a um grafo.

Sejam um grafo $G = (V, E)$ e uma ordem parcial $P \subseteq V^2$.

Definição 7. *O **grafo de eliminação segundo a ordem parcial** P é o grafo G_P obtido a partir de G da seguinte forma:*

1. $G_P \leftarrow G$
2. $S \leftarrow \emptyset$;
3. *Escolha u minimal em $P[V - S]$;*
4. *Seja $N = \{v \in N_{G_P}(u) : \ uv \in P\}$. Faça com que N induza um subgrafo completo em G_P;*
5. *Acrescente u em S e volte ao passo 2 caso $S \neq V$; caso contrário, pare.*

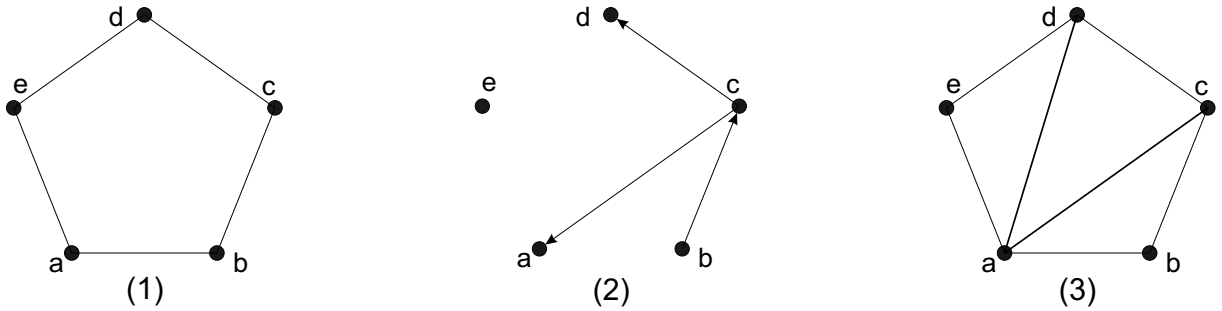

Figura 2.2. *Exemplo de grafo de eliminação: **(1)** grafo $G = (V, E)$; **(2)** ordem parcial $P \subseteq V^2$ (omitimos as transitividades e reflexividades); **(3)** G_P.*

A **pós-vizinhança** de um vértice $u \in V$ é dada por $N_P(u) = \{v \in N_{G_P}(u) : uv \in P\}$.

Se P é uma ordem total dizemos que P é um **esquema de eliminação** de V e algumas vezes denotamos P pela seqüência equivalente.

Seja $\pi = \langle v_1, ..., v_n \rangle$ um esquema de eliminação de V. Daqui em diante, denotaremos por $\pi(v)$ a posição do vértice v no esquema π, ou seja, $\pi(v) = i$ tal que $v = v_i$.

3

Complexidade dos Problemas

Neste capítulo, analisaremos as complexidades dos problemas DEA com k fixo e Triang. O motivo de estarmos também analisando este último é o fato de ele e DEA serem equivalentes, além de que será a abordagem tomada para "resolver" DEA nos capítulos seguintes.

Como veremos na seção a seguir, apesar de DEA com k fixo poder ser resolvido em tempo considerado polinomial, as constantes envolvidas no algoritmo apresentado são muito altas, tornando-o inviável computacionalmente. Além disso, a informação de que a largura de um grafo G é menor ou igual a um inteiro k fixo não é de nenhuma utilidade para encontrar uma decomposição em árvore para G. Apesar do conhecimento de um algoritmo que ao final fornece tal decomposição e cuja complexidade teórica é de $O(n)$ (Bodlaender e Kloks [14]), o tempo consumido para encontrá-la na prática é desmotivador (Röhrig [56]). Por isso escolhemos abordar o problema pela ótica da triangularização.

O capítulo está organizado da seguinte maneira: na Seção 3.1, mostramos que DEA com k fixo pertence a $\mathcal{P}$; na Seção 3.2, vemos que DEA equivale a $Triang$; finalmente, na Seção 3.3, mostramos que $Triang \in \mathcal{NP}$-completo (implicando que DEA também).

3.1. DEA com k fixo pertence a $\mathcal{P}$

Robertson e Seymour [55] deram uma prova não-construtiva de que existe um algoritmo em duas fases que resolve DEA com k fixo em tempo $O(n^2)$. Dado um grafo G qualquer, a primeira fase ou informa que a largura de G é maior do que k (ou seja, responde "NÃO") ou fornece uma decomposição em árvore de G de largura menor ou igual a $4k$. A segunda fase utiliza a decomposição encontrada pela primeira para finalmente decidir se a largura em árvore de G é no máximo k.

Nesta seção mostraremos os teoremas que fundamentaram a prova da existência do algoritmo em duas fases.

3.1.1. Primeira Fase

Na verdade, ainda no artigo em que provaram a existência do algoritmo de decisão, Robertson e Seymour também propuseram um algoritmo para a primeira fase de complexidade $O(n^2)$. Tal algoritmo utiliza o conceito de separadores dado a seguir:

Definição 8. *Seja S um conjunto de vértices de G. Um S-**separador** em G é um conjunto X de vértices de G tal que para toda componente G' de $G \backslash X$, $|V(G') \cap S| \leq \frac{2}{3}|S|$. A **ordem** de X é dada por $|X|$.*

O teorema a seguir relaciona separadores e decomposição em árvore.

Teorema 3.1.1 (Robertson e Seymour [55]). *Se $G = (V, E)$ tem largura em árvore no máximo k, então para cada $S \subseteq V$, G contém um S-separador de ordem no máximo $k+1$. Inversamente, se para cada $S \subseteq V$, G contém um separador de ordem no máximo $k+1$, então existe uma decomposição em árvore de G de largura no máximo $4k$.*

Prova: Provaremos apenas a primeira parte do enunciado do teorema. A segunda trata-se do algoritmo para a primeira fase, visto na subseção seguinte.

Para isso, mostraremos que para todo $S \subseteq V$ e toda decomposição em árvore (X, T) de G, existe $t \in V(T)$ tal que X_t é um S-separador. Por absurdo, sejam (X, T) uma

decomposição em árvore de G de largura ótima e $S \subseteq V$ para o qual (X, T) não possui um S-separador. Seja ainda $(y, z) \in E(T)$. Sejam Y a componente de $T - yz$ contendo y e V_Y o conjunto de vértices do subgrafo de G obtido por $\bigcup_{t \in Y} X_t$. Analogamente, sejam Z a subárvore de $T - yz$ contendo z e V_Z obtido de forma análoga. Note que $(V_Y \setminus X_z, V_Z \setminus X_y)$ é uma partição dos vértices de $G \setminus (X_y \cap X_z)$, pois, dado $v \in G \setminus (X_y \cap X_z)$, pela propriedade 3 da Definição 2 teremos que ou $v \in V_Y \setminus X_z$ ou $v \in V_Z \setminus X_y$. Sejam $v \in V_Y \setminus X_z$ e $u \in V_Z \setminus X_y$. Como $(V_Y \setminus X_z, V_Z \setminus X_y)$ é uma partição de $G \setminus (X_y \cap X_z)$, temos que $v \notin V_Z \setminus X_y$ e $u \notin V_Y \setminus X_z$. Logo, $(u, v) \notin E(G)$, pois caso contrário, deveria ser coberta por $X_y \cap X_z$, ou seja, $\{u, v\}$ deveria estar contido em $X_y \cap X_z$, o que não ocorre. Temos então que $X_y \cap X_z$ é um corte. Ademais, pela premissa, sabemos que $X_y \cap X_z$ não é um S-separador, o que nos indica que ou $|(V_Y \setminus X_z) \cap S| > \frac{2}{3}|S|$ ou $|(V_Z \setminus X_y) \cap S| > \frac{2}{3}|S|$. Suponha, sem perda de generalidade, que $|(V_Y \setminus X_z) \cap S| > \frac{2}{3}|S|$. Então, direcione a aresta (y, z) de z para y. Repita o processo feito para (y, z) em todas as arestas de T. Obviamente, o digrafo obtido é acíclico. Logo, existe um sumidouro t. Desta forma, nenhuma componente de $G \setminus X_t$ contém mais do que $\frac{2}{3}|S|$ vértices de S, absurdo pois neste caso X_t seria um S-separador. Podemos concluir, então, que todo $S \subset V$ possui um S-separador de tamanho no máximo $la(G) + 1 \leq k + 1$. □

A prova da segunda parte do teorema é o algoritmo proposto por Robertson e Seymour que será visto a seguir. Consiste, basicamente, em tentar construir uma decomposição em árvore com largura no máximo $4k$, encontrando separadores para alguns subconjuntos. Se em determinado momento for encontrado um subconjunto que não possua um separador de ordem no máximo $k+1$, pode-se concluir, pela primeira parte do teorema, que a largura de G é maior do que k.

3.1.1.1. Algoritmo para Primeira Fase: Robertson e Seymour

Dados um grafo $G = (V, E)$ e um subconjunto $W \subseteq V$ tal que $|W| \leq 3k$, o algoritmo consiste, basicamente, em tentar encontrar um W-separador de ordem menor ou igual $k + 1$. Caso não exista tal separador, pelo Teorema 3.1.1 podemos concluir que a largura

de G é maior do que k. Caso exista, o problema é dividido. Caso todos os subproblemas encontrem uma decomposição para os subgrafos, essas são combinadas para formarem uma decomposição em árvore do grafo original com largura no máximo igual a $4k$.

Para k fixo, o algoritmo tem como entrada um grafo $G = (V, E)$ e um subconjunto $W \subseteq V$ de cardinalidade no máximo $3k$. São duas as saídas possíveis:

1. uma decomposição em árvore $(\mathcal{X}, T)$ de G de largura no máximo $4k$, caso exista, juntamente com um vértice t tal que $W \subseteq X_t$

2. um conjunto $W^* \subseteq V$ para o qual G não possui um W^*-separador de ordem menor ou igual a $k + 1$.

A função $Separador(G, W, k+1)$ retorna um W-separador S de ordem no máximo $k+1$, caso exista, ou o conjunto vazio.

Note que os subconjuntos calculados na linha 10 do Algoritmo 1 são tais que $|W_i| \leq 3k$, para cada $i \in \{1, ..., l\}$, já que S é um W-separador, $|W| \leq 3k$ e $|S| \leq k$. Logo, é possível aplicar recursivamente o algoritmo considerando os subconjuntos W_i e os subgrafos G_i. Além disso, cada novo nó t, adicionado nas linhas 15 a 18 para compor as sub-decomposições, é tal que $|X_t| \leq 4k + 1$. Daí a largura da decomposição retornada. É fácil notar que é possível obter-se conjuntos menores do que os conjuntos W_i, bastando para isso não aumentar o conjunto W na linha 3. Porém, a constante $3k$ facilita os cálculos.

A seguir, damos uma visão geral de como a função $Separador(G, W, k+1)$ encontra um W-separador de ordem no máximo $k + 1$.

Lema 3.1.1. *G possui um W-separador de ordem no máximo $k+1$ se e somente se existe uma partição de $V(G)$ em conjuntos A, S, B tais que $|A \cap W| \leq \frac{2}{3}|W|$, $|B \cap W| \leq \frac{2}{3}|W|$, $|S| = k + 1$ e não existe arestas entre A e B.*

Prova: $\Rightarrow$ Seja S um W-separador de ordem no máximo igual a $k+1$. Sejam G_1, ..., G_q as componentes conexas de $G \setminus S$ ordenadas em ordem não decrescente de acordo com o

Algoritmo 1. $Fase1 : RS$

Entrada: $G = (V, E)$, $W \subseteq V(G)$ tal que $|W| \leq 3k$

Saída: 1. (X, T) e nó $t \in V(T)$ tal que $W \subseteq X_t$; ou **2.** $W^* \subseteq V(G)$

se $|V(G)| \leq 4k$ **então**

retorne $(X_t = \{v\}, (t, \emptyset))$

Adicione vértices em W de forma a tornar $|W| = 3k$

$S \leftarrow Separador(G, W, k + 1)$

se $S = \emptyset$ **então**

retorne W

Sejam $U_1, ..., U_l$ as componentes conexas de $G - S$

para $i = 1...l$ **faça**

$G_i \leftarrow G[S \cup U_i]$

$W_i \leftarrow (W \cap U_i) \cup S$

se $Fase1 : RS(G_i, W_i)$ retornar um conjunto S^i **então**

retorne S^i

senão

Sejam (X^i, T^i) e t_i a decomposição e o nó retornados, respectivamente

$V(T) \leftarrow \bigcup_{i=1...l} V(T^i) \cup \{t\}$

$E(T) \leftarrow \bigcup_{i=1...l} E(T^i) \ \cup \bigcup_{1 \leq i \leq l} (t, t_i)$

$X_t \leftarrow S \cup W$

$X \leftarrow \bigcup_{i=1...l} X^i \cup X_t$

retorne $((X, T), t)$

tamanho dos seus conjuntos de vértices. Denotaremos o conjunto de vértices da componete G_i por V_i. Sabe-se que $|V_i \cap W| \leq \frac{2}{3}|W|$, para $1 \leq i \leq q$. É fácil ver que existe $j \in \{1, ..., q\}$ tal que $|\bigcup_{1 \leq i \leq j} V_i| \leq \frac{2}{3}|W|$ e $|\bigcup_{j < i \leq q} V_i| > \frac{2}{3}|W|$. Logo, basta tomar a partição $A = \bigcup_{1 \leq i \leq j} V_i$, $B = \bigcup_{j < i \leq q} V_i$ e o próprio S para satisfazer o lema.

$\Leftarrow$ Trivial, pois neste caso o próprio S é um W-separador de ordem no máximo $k+1$.

□

Devido ao lema, ao invés de procurarmos especificamente por um W-separador, procuraremos por uma partição como a descrita. Faremos isto considerando partições de W em conjuntos W_A, W_B e W_C e verificando se existe uma partição de $V(G)$ em A, B e S tal que $A \cap W = W_A$, $B \cap W = W_B$ e $S \cap W = W_C$. Analisaremos apenas partições de W tais que $|W_A| \leq \frac{2}{3}|W|$, $|W_B| \leq \frac{2}{3}|W|$ e $|W_C| \leq k+1$, por motivos óbvios. Para cada uma destas partições, verifica-se a existência de mais do que $k + 1 - |W_C|$ caminhos disjuntos por vértices entre W_A e W_B em $G \setminus W_C$, pois caso não existam, é possível encontrar um conjunto S tal que $|S| \leq k+1$ e não existem caminhos entre W_A e W_B em $G \setminus S$ (basta adicionarmos a W_C um vértice de cada um dos (W_A, W_B)-caminhos disjuntos de $G \setminus W_C$). Além disso, se fizermos A conter todas as componentes de $G \setminus S$ que possui pelo menos um vértice de W_A, e W_B ser igual a $(V \setminus S)s \setminus A$, teremos a partição desejada. Logo, podemos retornar S. Caso exista mais do que $k + 1 - |W_C|$ (W_A, W_B)-caminhos disjuntos em $G \setminus W_C$ para toda partição de W analisada, certamente G não possui um W-separador de ordem no máximo k e podemos retornar $\emptyset$.

No Algoritmo 2, a função $CaminhosDisjuntos(W_1, W_2, G')$ retorna um conjunto de vértices de G' de forma que cada um pertence a exatamente um caminho entre W_1 e W_2 em G', todos os caminhos disjuntos entre si. Isto pode ser feito em tempo $O(k \cdot m)$, utilizando caminhos aumentantes.

Os três lemas a seguir nos fornecem a complexidade do algoritmo:

Lema 3.1.2. *O algoritmo para a primeira fase de Robertson e Seymour produz no máximo* $max\{1, 2n - 6k - 3\}$ *subproblemas.*

Algoritmo 2. $Separador(G, W, k+1)$

para todo particionamento W_A, W_B, W_C de W **faça**
  se $|W_A \cap W| \leq \frac{2}{3}|W|$, $|W_B \cap W| \leq \frac{2}{3}|W|$ **e** $|W_C| \leq k+1$ **então**
    $S \leftarrow CaminhosDisjuntos(W_A, W_B, G \setminus W_C)$
    se $|S| \leq k+1-|W_C|$ **então**
      retorne $S \cup W_C$
retorne $\emptyset$

Lema 3.1.3. *A complexidade de* $Separador(G', W, k+1)$*, dado que* $|W| \leq 3k$*, é* $O(3^{3k}km)$.

Lema 3.1.4. *Se* $m \geq k \cdot n$*, então* $la(G) \geq k+1$.

Temos, pelos Lemas 3.1.2 e 3.1.3 acima, que a complexidade do algoritmo é $O(2 \cdot 3^{3k}knm - 2 \cdot 3^{3k}k^2m - 3^{3k}km)$. Além disso, por 3.1.4 podemos assumir que $m \leq kn$, pois caso contrário já poderíamos concluir que $la(G) > k$. Como o inteiro k é considerado uma constante, temos que $m = O(n)$ e que a complexidade do algoritmo se resume a $O(n^2)$.

Reed [54] propõe uma modificação no algoritmo que utiliza separadores aproximados. Há uma piora na largura da decomposição fornecida ao final da execução, passando a ser no máximo $5k$, porém a complexidade melhora para $O(n \log n)$.

3.1.2. Segunda Fase

A segunda fase do algoritmo teve demonstrada apenas sua existência. Consiste, basicamente, em verificar uma caracterização dos grafos de largura no máximo k. A seguir definimos algumas operações sobre um grafo $G = (V, E)$:

1. Remoção de vértices: dado um vértice $v \in V$, consiste em remover v e as arestas incidentes em v (denotamos esta operação por $G - v$);

2. Remoção de arestas: dada uma aresta $e \in E$, consiste em remover a aresta e (denotamos esta operação por $G - e$);

3. Contração de arestas: dada uma aresta $e = (u, v) \in E$, consiste em acrescentar um vértice w, fazer com que w torne-se adjacente a todo vizinho de u ou de v e, finalmente, remover os vértices u e v.

Definição 9. *Sejam os grafos G e H. Se H pode ser obtido a partir de G através de uma seqüência de operações do tipo descrito acima, dizemos que H é um* ***menor*** *de G e denotamos por $H \preccurlyeq G$.*

Definição 10. *Dada uma classe de grafos Γ, dizemos que ela é* ***fechada sobre menores*** *se para todo grafo H em Γ, temos que se $H' \preccurlyeq H$ então H' também está em Γ.*

Pelo teorema a seguir, vemos que a classe de grafos de largura no máximo k, Γ_k, é fechada sobre menores.

Teorema 3.1.2 (Robertson e Seymour [55]). *Se H' é um menor de H, então $la(H') \leq la(H)$.*

O teorema a seguir constata que toda classe de grafos fechada sobre menores pode ser relacionada a um conjunto finito chamado de conjunto de menores proibidos.

Teorema 3.1.3 (Robertson e Seymour [55]). *Seja Γ uma classe de grafos, fechada sobre menores. Então, existe um conjunto finito de grafos Ψ, chamado conjunto de menores proibidos de Γ, tal que para todo H temos: $H \in \Gamma$ se e somente se não existe $H' \in \Psi$ tal que H' é menor de H.*

Podemos concluir pelo Teorema 3.1.3, que existe um conjunto finito de menores proibidos para Γ_k. Porém, este conjunto não é conhecido. Logo, apenas a existência de um algoritmo de verificação foi mostrada. A linearidade do algoritmo advém de resultados

que propõem que, dada a decomposição em árvore de largura limitada fornecida pela primeira fase do algoritmo, é possível verificar a relação de menores linearmente [7, 33].

O conjunto de menores proibidos é conhecido apenas para $k = 1$, $k = 2$ e $k = 3$. Para $k = 1$ e $k = 2$ os únicos menores proibidos são K_3 e K_4, respectivamente, onde K_n denota o grafo completo com n vértices. Para $k = 3$, o conjunto de menores proibidos é formado pelos grafos da Figura 3.1.

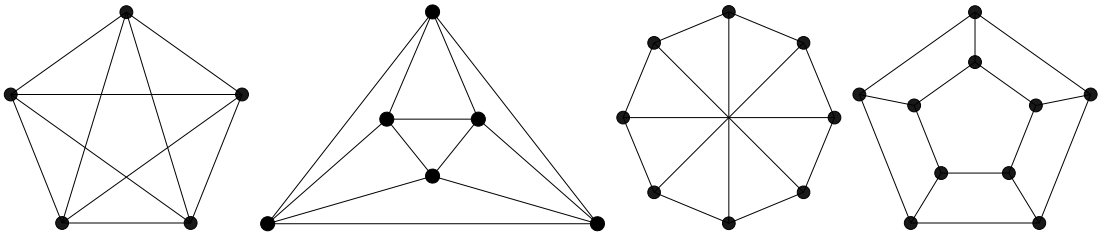

Figura 3.1. *Conjunto dos menores proibidos para a classe de grafos cuja largura é no máximo $k = 3$.*

A largura em caminho de um grafo é dada pela largura da menor decomposição em árvore cuja árvore associada trata-se de um caminho. Takahashi, Ueno e Kajitani [60] mostraram que o tamanho do conjunto de menores proibidos para a classe de grafos cuja largura em caminho é no máximo k contém pelo menos $k!^2$ árvores, cada uma contendo $(5 \cdot 3^k - 1)/2$ vértices. Outro resultado relacionado à cardinalide destes conjuntos, obtido por Ramachandramurthi [53], estuda os menores proibidos de Γ_k com $k+1$, $k+2$ e $k+3$ vértices.

3.2. $Triang$ é equivalente a DEA

O seguinte resultado define grafos triangularizados sob a visão de decomposição em árvores.

Teorema 3.2.1 (Gavril [37]). *Um grafo $G = (V, E)$ é triangularizado se e somente se existe uma árvore $T = (I, F)$ tal que, para cada vértice $v \in V$, podemos associar uma*

subárvore $T_v = (I_v, F_v)$ *de* T *tal que* $(v, w) \in E$ *se e somente se* $I_v \cap I_w \neq \emptyset$.

Note que a segunda parte do Teorema acima equivale a uma decomposição em árvore $D = (\mathcal{X}, T = (I, F))$ do grafo G tal que, dado um nó $t \in I$ qualquer, $\{v, w\} \subseteq X_t$ se e somente se $(v, w) \in E$. Logo, para todo $t \in I$ temos que X_t é uma clique em G. Como veremos no Lema 3.2.1, toda clique do grafo deve estar contida em algum nó. Devido a isso e ao fato de que nenhum nó contém mais do que uma clique, podemos concluir que esta decomposição é ótima. Além disso, vemos também que a largura da decomposição D é dada por $\omega(G) - 1$. Logo, o seguinte colorário é válido:

Corolário 3.2.1. *Para todo grafo triangularizado* G*, temos que* $la(G) = \omega(G) - 1$

Pelo colorário acima é fácil notar que $la(G_{MIN})$ é mínima dentre as larguras de todas as triangularizações de G.

O lema abaixo mostra que toda clique do grafo deve estar contida em uma das partes de D, para toda decomposição em árvore D de G.

Lema 3.2.1. *Sejam um grafo* $G = (V, E)$ *e* $D = (\mathcal{X}, T)$ *uma decomposição em árvore de* G*. Se* $C \subseteq V$ *é uma clique de* G*, então existe um nó* $t \in V(T)$ *tal que* $C \subseteq X_t$.

Prova: Para cada $v \in V$, defina T_v como sendo o subgrafo de T formado pelos nós que contém v. Certamente, T_v trata-se de uma árvore, pois é acíclico, toda componente conexa de T_v contém v e pela condição 3 da Definição 2. Seja $(u, v) \in E$. Como (u, v) deve ser coberta, existe $t \in V(T)$ tal que $\{u, v\} \subseteq X_t$. Obviamente, $t \in V(T_u)$ e $t \in V(T_v)$. Logo, $V(T_u) \cap V(T_v) \neq \emptyset$, para toda aresta $(u, v) \in E$. Então, dado o conjunto $\Gamma_C = \{T_v : v \in C\}$ das árvores relacionadas aos vértices de uma clique C, sabemos que quaisquer duas árvores $T', T'' \in \Gamma_C$ são tais que $V(T') \cap V(T'') \neq \emptyset$. Pela Propriedade de Helly aplicada a árvores, podemos concluir que existe $\bigcap_{T' \in \Gamma_C} V(T') \neq \emptyset$, ou seja, existe $t \in V(T)$ que contém todos os vértices de C.

□

O lema seguinte nos fornece o fato necessário para provar a equivalência entre $Triang$ e DEA.

Lema 3.2.2. *Para todo grafo $G = (V, E)$ existe uma triangularização de G, $G' = (V, E \cup E')$, tal que $la(G') = la(G)$.*

Prova: Seja $D = (X, T)$ uma decomposição em árvore de G de largura mínima. Seja $G' = (V, E \cup E')$ construído a partir de G da seguinte forma: para todo par de vértices (v, w) de V, se $\{v, w\} \subseteq X_t$, para algum $t \in V(T)$, e $(v, w) \notin E$, então acrescente (v, w) em E'. Pelo Teorema 3.2.1, temos que G' é um grafo triangularizado, logo, trata-se obviamente de uma triangularização de G. Além disso, D também é uma decomposição em árvore para G' de largura mínima. Logo $la(G') = la(G)$. □

Lema 3.2.3. *Seja G_{MIN} a triangularização mínima de G, então $la(G_{MIN}) = la(G)$.*

Prova: Temos que $G_{MIN} = (V, E \cup E_{MIN})$. Dada uma decomposição em árvore D de G_{MIN}, temos que D também será uma DA de G, pois basta retirar as arestas de E_{MIN}. Logo $la(G_{MIN}) \geq la(G)$. Além disso, temos que $la(G_{MIN}) \leq la(G')$, para toda triangularização G' de G. Isto, adicionado ao Lema 3.2.2, nos leva ao resultado $la(G_{MIN}) \leq la(G)$. □

Como veremos a seguir, $Triang \in \mathcal{NP}$-completo. Porém, dado um grafo G e uma triangularização G' de G qualquer, o tamanho da maior clique de G' menos um nos fornece um limite superior para a largura em árvore de G. Ademais, o problema de encontrar uma decomposição em árvore ótima de um grafo triangularizado é polinomial, assim como encontrar uma triangularização qualquer de G pode ser feito também polinomialmente. Desta forma, uma alternativa fácil para obter uma decomposição em árvore de G é encontrar uma triangularização G' e computar uma decomposição D ótima para G'. Certamente, D também será uma decomposição para G. A qualidade de D dependerá unicamente da qualidade da triangularização.

3.3. $Triang$ é $\mathcal{NP}$-completo

Nesta seção, iremos mostrar a equivalência entre $k-Tree$ e $Triang$. Arnborg, Corneil e Proskurowski [4] mostraram que $k-Tree \in \mathcal{NP}$-completo; como conseqüência teremos

que $Triang$ também.

Definição 11. *A classe de k-árvores é definida recursivamente como segue:*

1. *O grafo completo com k vértices é uma k-Árvore;*
2. *Uma k-árvore com $n+1$ vértices pode ser construída a partir de uma k-árvore com n vértices, adicionando-se um vértice e fazendo com que este seja adjacente a todos os vértices de uma das cliques de tamanho k, e somente a estes vértices.*

Uma **k-árvore parcial** é um subgrafo de uma k-árvore.

Problema 4. *$(k-Tree)$ Dados um grafo G qualquer e um inteiro k positivo, decidir se G é uma k-árvore parcial.*

Um grafo $G = (V, E)$ é **k-decomponível** se e somente se:

1. $n \leq k+1$ ou
2. existe $S \subset V$ tal que $|S| \leq k$, $G \setminus S$ é desconexo e $G_{K_S}[V(U) \cup S]$ é k-decomponível, para cada componente U de $G \setminus S$, onde G_{K_S} denota o grafo obtido com a adição de arestas de forma a fazer com que S seja uma clique.

Note que uma k-árvore é k-decomponível.

Um grafo **k-cordal** é um grafo cordal G tal que $\omega(G) = k+1$ (ou seja, $la(G) = k$). Um grafo **k-cordal parcial** é um subgrafo de um grafo k-cordal.

Lema 3.3.1. *Todo grafo k-cordal é k-decomponível.*

Prova: Dado um grafo k-cordal G, pelo Lema 4.1.1 no capítulo seguinte, sabemos que ele possui pelo menos um vértice simplicial u. Como $\omega(G) \leq k+1$, temos que $|N(u)| \leq k$. Sabemos também que $N(u)$ é uma clique que separa u do resto do grafo. Ademais, $N[u]$ é um grafo completo com no máximo $k+1$ vértices e $G-u$ trata-se também de um

grafo cordal cuja maior clique terá tamanho no máximo $k+1$. Logo, temos que G é k-decomponível. □

Dado um grafo G qualquer, definimos $k_c(G)$ como o valor mínimo de k tal que G é um grafo k-cordal parcial e $k_t(G)$ como o valor mínimo de k tal que G é uma k-árvore parcial. Note que $k_c(G)$ é o menor valor possível passado para $Triang$ cuja resposta será sim. Da mesma forma, $k_t(G)$ é o menor possível para $k - Tree$ cuja resposta será sim. O lema e o teorema a seguir nos fornece o resultado que queríamos.

Lema 3.3.2. *Para qualquer grafo G, $k_t(G) = k_c(G)$.*

Prova:

1. $k_c(G) \leq k_t(G)$: Seja G' um supergrafo $k_t(G)$-árvore de G. É fácil ver que uma k-árvore é também um grafo k-cordal, pois a vizinhança de qualquer vértice induz uma clique de tamanho exatamente k, ou seja, todo vértice é simplicial e o tamanho da maior clique é $k+1$. Sendo assim, G' é também $k_t(G)$-cordal. Logo, $k_t(G) \geq k_c(G)$.

2. $k_t(G) \leq k_c(G)$: Seja G' um supergrafo $k_c(G)$-cordal de G. Sabemos, pelo Lema 3.3.1, que G' é $k_c(G)$-decomponível. Logo, por Arnborg e Proskurowski [6], temos que G' também é uma $k_c(G)$-árvore. Logo, G é uma $k_c(G)$-árvore parcial, e $k_c(G) \geq k_t(G)$.

□

4

D.A. de Grafos Triangularizados

O principal objetivo deste capítulo é mostrar como encontrar uma decomposição em árvore de um grafo triangularizado em tempo polinomial. A importância disto deve-se ao fato de que passaremos a trabalhar em torno do problema de triangularizar um grafo, ao invés de trabalhar diretamente sobre uma decomposição. A idéia para encontrar uma boa decomposição de um grafo G qualquer é: primeiramente encontrar uma boa triangularização G' de G e, em seguida, decompor G' utilizando o algoritmo visto na Seção 4.2. Como vimos anteriormente, esta decomposição também será uma decomposição em árvore de G.

Na primeira seção deste capítulo, expomos a idéia geral de como utilizamos o conceito de esquema de eliminação visto em 2.3 para encontrar uma triangularização qualquer. Logo em seguida, apresentamos o algoritmo a que se propõe este capítulo.

4.1. Esquema de Eliminação

O lema e o teorema a seguir relacionam esquemas de eliminação e grafos triangularizados, dando uma idéia de como utilizar os esquemas para encontrar triangularizações de um grafo qualquer.

Lema 4.1.1 (Dirac[31]). *Todo grafo cordal possui um vértice simplicial. Além disso, se o grafo não for completo, então possui pelo menos dois vértices simpliciais não adjacentes.*

Teorema 4.1.1 (Fulkerson e Gross[36]). *Um grafo G é triangularizado se e somente se existe um esquema de eliminação perfeito para os vértices de G.*

Pelo teorema acima e pelo Lema 4.1.1 vemos que uma forma simples de reconhecer grafos triagularizados é construindo um esquema de eliminação dos vértices da seguinte forma: inicialmente o esquema é vazio; toma-se um vértice simplicial e coloca-se no final do esquema até então obtido; retira-se tal vértice do grafo e repete-se o procedimento anterior sobre o grafo obtido até que o todos os vértices tenham sido postos no esquema. No capítulo seguinte veremos que existem outros métodos para reconhecimento de grafos triangularizados mais eficientes do que o descrito.

Seja G_π o grafo de eliminação do esquema π. Note que a eliminação dos vértices é feita de acordo com o esquema π, logo a eliminação de v_i só poderá ser feita após a eliminação de $\{v_1, ..., v_{i-1}\}$. Note ainda que π é um esquema de eliminação perfeito dos vértices de G_π. Logo, pelo Teorema 4.1.1, G_π é triangularizado.

Pela observação feita no parágrafo acima, temos então que uma forma de encontrar uma triangularização de um grafo G é construindo um esquema de eliminação π dos vértices de G e, após isso, construir o grafo G_π. As heurísticas apresentadas no Capítulo 5.1 determinam um esquema de eliminação. Algumas delas são baseadas em algoritmos de reconhecimento de grafos cordais, logo garantem que, se G é triangularizado, então o esquema encontrado é perfeito.

4.2. Algoritmo de Decomposição

Nesta seção veremos como encontrar uma decomposição em árvore de largura mínima para um grafo triangularizado qualquer.

Teorema 4.2.1 (Dirac[31]). *$G = (V, E)$ é triangularizado se e somente se admite uma decomposição $D = (\mathcal{X}, T)$ tal que X_t é uma clique maximal de G, para todo nó $t \in V(T)$.*

Seja $G = (V, E)$ um grafo triangularizado. Note que dado um esquema de eliminação perfeito de V, π, temos que $G = G_\pi$. Note ainda que as cliques são facilmente encontradas, bastando tomar um vértice e a sua pós-vizinhança em π. Logo, um algoritmo para encontrar uma decomposição em árvore de G, dado este esquema, consiste em:

1. Tome i tal que $N_\pi(v_i)$ é um corte em G;
2. Sejam $\{G_1, ..., G_q\}$ as componentes conexas de $G \setminus N_\pi(v_i)$ (denotaremos o conjunto de vértices da componente G_k por V_k);
3. Encontre uma decomposição $D^k = (\mathcal{X}^k, T^k)$ de $G[V_k \cup N_\pi(v_i)]$, para $k = 1, ..., q$ (note que estes grafos também são cordais, logo estas decomposições podem ser obtidas usando este mesmo esquema);
4. Tome $t_k \in V(T^k)$ tal que $N_\pi(v_i) \subseteq X^k_{t_k}$ (existe pois $N_\pi(v_i)$ induz uma clique e pelo Lema 3.2.1);
5. Construa uma decomposição criando um nó t tal que $X_t = N_\pi(v_i)$ e fazendo t adjacente a cada t_k.

Para encontrar as decomposições exigidas no passo 3, aplicamos este mesmo esquema utilizando π restrito aos vértices do subgrafo $G[V_k \cup N_\pi(v_i)]$, para $k = 1, ..., q$.

A seguir mostramos o pseudocódigo do algoritmo. Este tem como entrada um grafo triangularizado $G = (V, E)$ e um esquema de eliminação perfeito π dos vértices de G, e como saída uma decomposição em árvore de G. No algoritmo, para uma decomposição D, T^D denota a árvore da decomposição de D e $\mathcal{X}^D$ os subconjuntos relacionados. Além disso, a função $Prox(\pi)$ retorna um vértice do esquema cuja pós-vizinhança desconecta o grafo G (sempre existirá a não ser que o grafo seja completo). É interessante escolher aquele vértice cuja pós-vizinhança divide o grafo em um maior número de componentes. Assim, a cada iteração estaremos trabalhando com um problema significativamente menor.

Algoritmo 3. *DAGrafoTriang*

Entrada: Grafo triangularizado $G = (V, E)$, Esquema perfeito π de V

Saída: Decomposição em árvore $D = (\mathcal{X}, T)$ de G

se G é completo **então**

retorne $(\{V\}, (\{t\}, \emptyset))$

$v \leftarrow Prox(\pi)$

$D \leftarrow (\{N_\pi(v)\}, (\{t\}, \emptyset))$

para cada componente conexa G' de $G \setminus N_\pi(v)$ **faça**

Seja G' o subgrafo de G induzido por $V(G') \cup N_\pi(v)$

$D' \leftarrow DAGrafoTriang(G', \pi)$

Seja $t' \in V(T^{D'}) : N_\pi(v) \subseteq X_{t'}^{D'}$ //existe pelo Lema 3.2.1

$\mathcal{X}^D \leftarrow \mathcal{X}^D \cup \mathcal{X}^{D'}$

$V(T^D) \leftarrow V(T^D) \cup V(T^{D'})$

$E(T^D) \leftarrow E(T^D) \cup E(T^{D'}) \cup \{(t, t')\}$

retorne D

5

Limites Superiores

Neste capítulo apresentamos as heurísticas mais amplamente conhecidas para o problema DEA.

A maioria destas heurísticas utilizam-se de rótulos atribuídos aos vértices para construir uma ordem de eliminação. Dada esta semelhança, decidimos agrupar tais heurísticas numa mesma seção (Seção 5.1).

A heurística apresentada na Seção 5.2 aborda diretamente o problema de decomposição. Tomando uma decomposição em árvore qualquer dada como entrada, ela tenta a cada iteração melhorar a largura da decomposição, usando para isso cortes mínimos dos vértices do grafo.

A heurística apresentada na última seção deste capítulo trata-se de uma aplicação da meta-heurística $GRASP$ para o problema em questão. Ressaltamos que não é conhecido nenhum estudo computacional que utilize o $GRASP$ para obter um limite superior para o problema DEA. O intuito da utilização desta meta-heurística foi o de obter um parâmetro comparativo para as heurísticas que utilizam rotulação.

5.1. Heurísticas de Rotulação

Todas as heurísticas apresentadas nesta seção encontram um esquema de eliminação basicamente da seguinte maneira: todo vértice possui um rótulo a ele atribuído seguindo certos critérios; a cada passo, dentre os vértices ainda não postos no esquema, o de maior ou menor rótulo, dependendo do método de rotulação utilizado, é escolhido para próximo vértice a ser posto no esquema. Além disso, o esquema pode ser construído da primeira para a última posição, ou vice-versa, dependendo novamente do método de rotulação utilizado.

O Algoritmo 4 fornece uma visão geral do funcionamento de uma heurística de triangularização baseada em rotulação. Ele deve receber um método de rotulação L que implemente as seguintes funções e procedimentos (S representa o conjunto de vértices ainda não postos no esquema):

- $InicializaRotulos(G)$ - inicializa os rótulos dos vértices de G;
- $Crescente()$ - deve retornar $verdadeiro$ se o esquema deve ser construído da primeira para a última posição, $falso$ caso contrário;
- $ProxVertice(S)$ - retorna o próximo vértice a ser ordenado dentre os elementos do conjunto S, ou seja, o vértice em S de melhor rótulo segundo L;
- $AtualizaRotulos(S, u)$ atualiza os rótulos dos vértices em S quando da ordenação de u.

Além disso, o algoritmo recebe como entrada um grafo $G = (V, E)$ e fornece como saída um esquema de eliminação π de V.

5.1.1. Busca Lexicográfica

A Busca Lexicográfica [57] foi proposta originalmente para o reconhecimento de grafos cordais. O algoritmo original tenta construir um esquema de eliminação perfeito

Algoritmo 4. *HeurRotula*

Entrada: Grafo $G = (V, E)$, Método de rotulação L

Saída: Esquema de eliminação π de V

1: $L.InicializaRotulos(G)$
2: $S \leftarrow V$ //conjunto de vértices não ordenados
3: **se** $L.Crescente()$ **então**
4: $\quad i \leftarrow 1$
5: $\quad inc = 1$
6: **senão**
7: $\quad i \leftarrow n$
8: $\quad inc = -1$
9: **enquanto** $S \neq \emptyset$ **faça**
10: $\quad u \leftarrow L.ProxVertice(S)$
11: $\quad \pi(i) \leftarrow u$
12: $\quad S \leftarrow S - \{u\}$
13: $\quad L.AtualizaRotulos(S, u)$
14: $\quad i \leftarrow i + inc$
15: **retorne** π

para os vértices do grafo dado como entrada; se em algum momento se verificar que o próximo vértice no esquema construído não é um vértice simplicial, o algoritmo retorna com a resposta "o grafo não é cordal". Para adequar o algoritmo às nossas necessidades, simplesmente continuaremos a execução até que um esquema de eliminação de todos os vértices seja encontrado.

A busca lexicográfica ordena os vértices da última para a primeira posição, escolhendo o próximo vértice a ser ordenado dentre aqueles de rótulo máximo. A rotulação é feita da seguinte forma: inicialmente o rótulo de todos os vértices é vazio (ou zero); em seguida, toda vez que um vértice é escolhido, seus vizinhos ainda não ordenados têm seu rótulo modificado colocando-se a posição em que o vértice escolhido foi posto no esquema no final do rótulo.

Observe, por exemplo, a Figura 5.1. Entre parênteses estão indicados os rótulos dos vértices; os vértices pretos já foram postos em π e o cinza é o próximo a ser posto. Os rótulos dos vértices já ordenados são na verdade os rótulos que possuiam quando da sua ordenação. Ao final, as arestas destacadas foram adicionadas pela eliminação dos vértices de acordo com π.

Enfatizamos que, quando da inicialização e atualização, os rótulos são tratados como uma cadeia de caracteres (tais caracteres serão apenas dígitos de 0 a 9). Porém, na função $ProxVertice$ ao compararmos seus valores para saber qual o de maior rótulo, consideramos como inteiros não-negativos. Logo, a cadeia de caracteres "9" seria considerada inferior à "10".

5.1.2. Cardinalidade Máxima

A heurística de Cardinalidade Máxima [61] ordena os vértices da última para a primeira posição, escolhendo o próximo vértice a ser ordenado dentre aqueles de rótulo máximo. O rótulo de um vértice v é dado pela quantidade de vizinhos de v já ordenados.

Observe, por exemplo, a Figura 5.2. A notação usada é semelhante à da Figura 5.1.

O Teorema 5.1.1 é de grande importância nas provas de alguns dos lemas enunciados

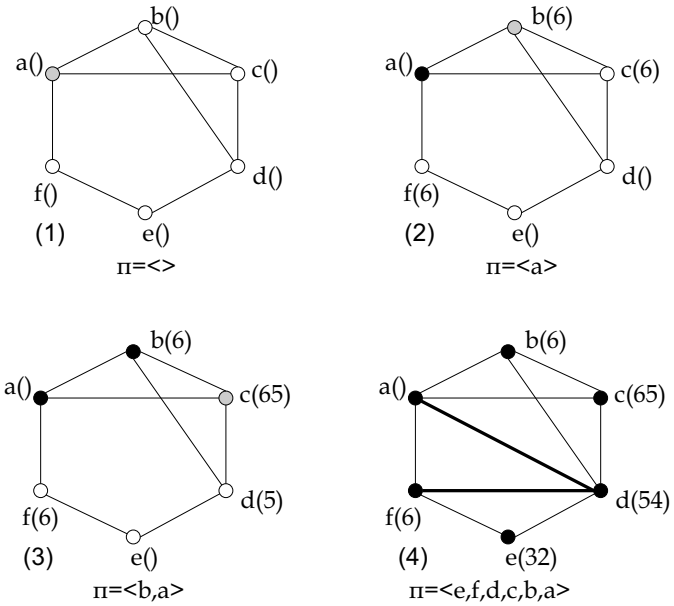

Figura 5.1. *(1)Inicialização dos rótulos;(2)Modificação após inclusão de a;(3)Modificação após inclusão de b;(4)Esquema final π e G_π.*

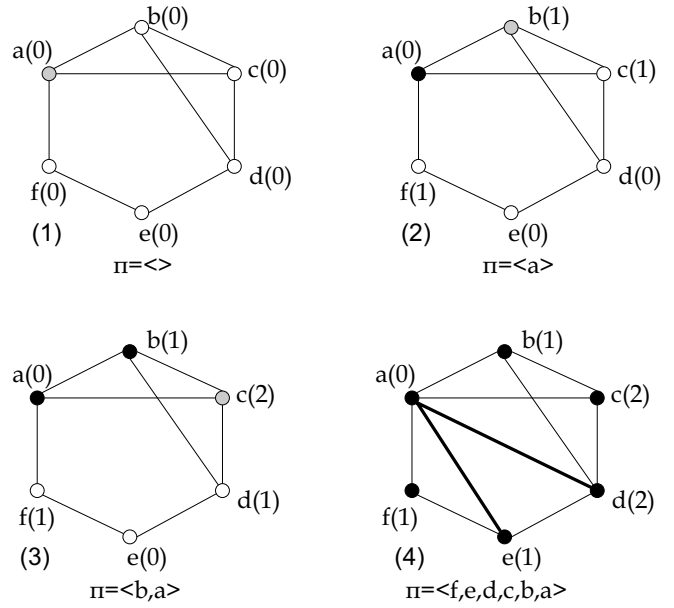

Figura 5.2. *(1)Inicialização dos rótulos;(2)Modificação após inclusão de a;(3)Modificação após inclusão de b;(4)Esquem final π e G_π.*

neste capítulo. Daqui em diante, denotaremos os métodos de rotulação Busca Lexicográfica e Cardinalidade Máxima por $LEXP$ e MCS, respectivamente.

Teorema 5.1.1 (Rose, Tarjan e Lueker [57]). *Dado um grafo $G = (V, E)$ e um esquema de eliminação π dos vértices de G, temos que $(u, v) \in G_\pi$ se e somente se $(u, v) \in G$ ou existe um caminho $\langle u, x_1, ..., x_q, v\rangle$ em G tal que $\pi(x_i) < min\{\pi(u), \pi(v)\}$, para $1 \leq i \leq q$.*

Lema 5.1.1. *Se G é cordal, o Algoritmo 4 executado com o método $LEXP$ ou MCS produz um esquema de eliminação perfeito.*

5.1.3. Caminhos de Menores Rótulos

As heurísticas LEXP e MCS podem ser alteradas de forma a retornarem um esquema de eliminação minimal. Tal modificação reside basicamente em quais vértices terão seu rótulo atualizado quando da ordenação de um determinado vértice. Apesar da mudança ser a mesma sobre ambas heurísticas, foram propostas separadamente. Primeiramente, no mesmo artigo em que propuseram a Busca Lexicográfica, Rose, Tarjan e Lueker [57] propuseram a modificação que levaria a uma triangularização minimal. Posteriormente, Tarjan e Yannakakis [61] mostraram que para o reconhecimento de grafos cordais era necessário apenas contar a quantidade de vértices vizinhos já ordenados, ou seja, propuseram a heurística Cardinalidade Máxima. Apenas recentemente é que Berry, Blair, Heggernes e Peyton [10] mostraram que aquela modificação também poderia ser aplicada ao algoritmo Cardinalidade Máxima de forma a obter uma triangularização minimal. Como veremos adiante, na verdade é possível utilizar a mesma prova de corretude para ambas heurísticas.

Um esquema de eliminação π é dito **minimal** se não existe outro esquema de eliminação σ tal que G_σ seja subgrafo próprio de G_π, ou seja, G_π é tal que $G_\pi - e$ não é cordal, para toda aresta $e \in E_\pi$, onde $E_\pi = E(G_\pi) \setminus E(G)$.

A um caminho $\langle u, v_1, ..., v_q, v\rangle$ em G onde $r(v_i) < min\{r(u), r(v)\}$, $i = 1, ..., q$, chamaremos de **caminho de menores rótulos**.

Note que em ambas heurísticas, ao ordenar um vértice v, todos os vizinhos de v ainda não ordenados deveriam ter seus rótulos modificados. A alteração necessária consiste em, ao ordenar um vértice v, todo vértice ainda não ordenado alcançável por v através de um caminho de menores rótulos terá seu rótulo modificado de acordo com a forma de rotulação escolhida (se busca lexicográfica ou cardinalidade máxima).

Por exemplo, observe a Figura 5.3 onde aplicamos esta modificação à heurística Busca Lexicográfica. A notação é a mesma usada nos exemplos anteriores.

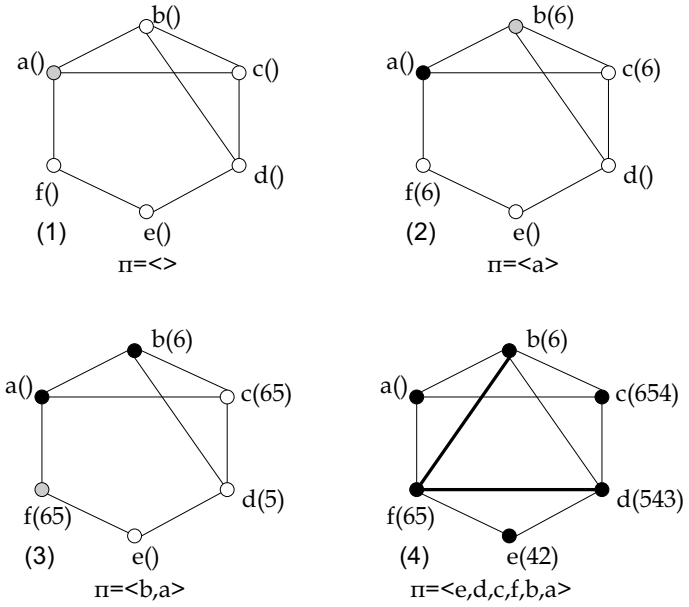

Figura 5.3. *(1)Inicialização dos rótulos;(2)Modificação após inclusão de a;(3)Modificação após inclusão de b;(4)Esquema final π e G_π.*

Daqui em diante, representaremos por LEXM e MCSM as heurísticas LEXP e MCS modificadas de forma a utilizarem os caminhos de menores rótulos, respectivamente.

A seguir apresentamos a prova de que esta simples modificação faz com que o esquema encontrado seja minimal. Para um vértice v qualquer, v^- denota o momento imediamente anterior à escolha do vértice v para ser posto no esquema e v^+, o momento imediatamente posterior à atualização dos rótulos após v ser posto.

O lema a seguir enuncia que se um caminho de menores rótulos entre dois vértices y e z existe em um dado momento v^-, então cada um dos vértices internos a este caminho será escolhido para ser posto no esquema após a escolha de y e de z, ou seja, aparecerão no esquema π antes dos vértices y e z.

Lema 5.1.2. *Seja π o esquema encontrado pela heurística utilizando $LEXM$ ou $MCSM$. Para um passo qualquer do algoritmo, seja v o vértice escolhido. Dentre os vértices ainda não postos no esquema, se $r_{v^-}(x_i) < min\{r_{v^-}(y), r_{v^-}(z)\}$, para todo x_i no caminho $\langle y, x_1, ..., x_q, z\rangle$ em G, então $\pi(x_i) < min\{\pi(y), \pi(z)\}$.*

Prova: Seja $\langle y, x_1, ..., x_q, z\rangle$ um caminho que satisfaz a premissa. Como, para $1 \leq i \leq q$, temos $r_{v^-}(x_i) < min\{r_{v^-}(y), r_{v^-}(z)\}$, se existe um caminho de menores rótulos entre v e algum x_j, este pode ser estendido em caminhos de menores rótulos até y e até z. Ou seja, se algum x_j tem o rótulo modificado, tanto z quanto y também terão. Logo, $r_{v^+}(x_i) < min\{r_{v^+}(y), r_{v^+}(z)\}$. É fácil ver que isso continuará sendo válido para as próximas iterações do algoritmo, até que um dentre y ou z seja escolhido. Suponha, sem perda de generalidade, que $\pi(y) > \pi(z)$. Logo, $r_{y^-}(x_i) < r_{y^-}(z) \leq r_{y^-}(y)$, para $1 \leq i \leq q$. Note que o mesmo argumento usado anteriormente pode ser usado agora, pois a partir da ordenação de y, qualquer caminho de menores rótulos até x_j, $1 \leq j \leq q$, pode ser estendido para um caminho até z. Assim, certamente z será escolhido antes de qualquer x_j, para $1 \leq j \leq q$. □

O lema a seguir define as arestas que serão acrescentadas no grafo de eliminação G_π.

Lema 5.1.3. *Sejam π o esquema encontrado pela heurística utilizando MCSM ou LEXM e $G_\pi = (V, E \cup E_\pi)$ o grafo de eliminação correspondente. Seja ainda o conjunto E^* definido como a seguir:*

$$E^* = \{(u, v) : \exists \langle u, x_1, ..., x_q, v\rangle \text{ tal que } r_{v^-}(x_i) < r_{v^-}(u),\ 1 \leq i \leq q\}$$

Temos que $E^* = E_\pi$.

Prova: Note que se no momento v^- o rótulo $r_{v^-}(x)$ é definido para um vértice x então, obviamente, $r_{v^-}(x) \leq r_{v^-}(v)$. Por isso omitimos na fórmula acima o fato de que $r_{v^-}(u) \leq r_{v^-}(v)$. A seguir, mostramos que $E^* = E_\pi$.

- $(u, v) \in E^* \rightarrow (u, v) \in E_\pi$: trivial, pelo Lema 5.1.2 e pelo Teorema 5.1.1.
- $(u, v) \in E_\pi \rightarrow (u, v) \in E^*$: Analisemos o momento v^-. Por absurdo, suponha que não existem caminhos de menores rótulos entre u e v neste momento. Sejam então $p_1, ..., p_r$ todos os caminhos entre u e v em G cujos vértices internos aparecem no esquema π antes de u e de v. Certamente $r \geq 1$, devido ao Teorema 5.1.1. Pela suposição, existe x_i em cada caminho p_i tal que $r_{v^-}(x_i) \geq r_{v^-}(u)$. Note que pelos menos o vértice x_i mais próximo de v no caminho p_i tal que isso ocorre é tal que $r_{v^+}(x_i) > r_{v^+}(u)$. Definimos, então, x_i^* como sendo o vértice mais próximo de u no caminho p_i tal que $r_{v^+}(x_i^*) > r_{v^-}(u)$. Note que, como não existe um caminho de menores rótulos de u até v no momento v^-, temos que $r_{v^+}(u) = r_{v^-}(u) < r_{v^+}(x_i^*)$, para $1 \leq i \leq r$. Como cada x_i^* é definido como o mais próximo de u, temos que para qualquer w escolhido após v, se existir um caminho de menores rótulos até u, este caminho poderá ser estendido em um caminho de menores rótulos até x_i^*. Desta forma, sempre que o rótulo de u aumentar, o de x_i^* também aumentará da mesma quantidade. Como a heurística sempre escolhe o vértice de maior rótulo, temos que todo vértice x_i^* será escolhido antes de u, absurdo pois neste caso (u, v) não estaria em E_π de acordo com o Teorema 5.1.1.

□

Lema 5.1.4. *Sejam π o esquema encontrado pela heurística utilizando MCSM ou LEXM e $G_\pi = (V, E \cup E_\pi)$ o grafo de eliminação correspondente. Para toda aresta $(u, v) \in E_\pi$, (u, v) é a única corda de algum ciclo de tamanho quatro em G_π.*

Prova: Suponha, inicialmente, que $E_\pi \neq \emptyset$, pois caso contrário não há o que mostrar. Note, pelo lema anterior, que para cada aresta $(u, v) \in E_\pi$ existe pelo menos um caminho

de menores rótulos associado $\langle u, x_1, ..., x_q, v\rangle$. Note ainda que, como inicialmente todos os rótulos são iguais (zero ou vazio), não existe nenhum caminho de menores rótulos com extremidade em v_n, onde $\pi(v_n) = n$, logo nenhuma aresta em E_π será incidente a v_n. Desta forma, uma aresta (u, v) ocorre em E_π sempre que um caminho de menores rótulos entre u e v for "criado".

Seja, então, $(u, v) \in E_\pi$ e z o vértice tal que não existe um caminho de menores rótulos entre u e v em G no momento z^-, porém existe pelo menos um em z^+. Seja $p = \langle u, x_1, ..., x_q, v\rangle$ um caminho de menores rótulos em z^+. Sem perda de generalidade, assumimos que $r_{z^-}(u) \leq r_{z^-}(v)$. É fácil ver que se p tornou-se um caminho de menores rótulos após a inserção de z no esquema, então o rótulo de u fora modificado, ou seja, existe um caminho de menores rótulos p' entre u e z no momento z^-. Além disso, não existe $x_i \in p$ tal que $r_{z^-}(x_i) > r_{z^-}(u)$, pois caso existisse o caminho p' poderia ser estendido em um caminho de menores rótulos de z a x_i, ou seja, x_i também teria o rótulo modificado e p não seria um caminho de menores rótulos em z^+ (note que o argumento é válido para o vértice x_i *mais próximo* de u no caminho p tal que $r_{z^-}(x_i) > r_{z^-}(u)$). Logo, temos que para todo x_i no caminho p, $r_{z^-}(x_i) \leq r_{z^-}(u)$. Utilizaremos estes fatos mais adiante.

Seja $x_k \in p$ tal que $\pi(x_k) \geq \pi(x_i)$, para todo $i \in p$. Mostraremos que (u, v) é a única corda no ciclo $\langle u, x_k, v, z, u\rangle$.

- $(z, u) \in E_\pi$ - Trivial, devido à existência do caminho p' mencionado acima;

- $(v, z) \in E_\pi$ - Se $r_{z^-}(u) = r_{z^-}(v)$, então $(v, z) \in E\pi$ pelo mesmo argumento usado para mostrar que $(u, z) \in E_\pi$; caso contrário, se $r_{z^-}(v) > r_{z^-}(u)$, como $r_{z^-}(z) \geq r_{z^-}(v)$ e $r_{z^-}(x_i) \leq r_{z^-}(u)$, para todo x_i em p, temos que o caminho p acrescido de p' define um caminho de menores rótulos entre v e z no momento z^-, definindo assim a aresta (v, z) em E_π.

- $(x_k, v) \in E_\pi$ e $(u, x_k) \in E_\pi$ - Trivial, pelo Teorema 5.1.1 e pela existência dos caminhos $\langle u, x_1, ..., x_k\rangle$ e $\langle x_k, ..., x_q, v\rangle$.

- $(x_k, z) \notin E_\pi$ - Suponha, por absurdo, que $(x_k, z) \in E_\pi$. Logo, deve existir um

caminho de menores rótulos entre x_k e z no momento z^-, pois se ainda não existe, não mais existirá já que z está sendo posto no esquema. Desta forma, temos que o rótulo de x_k é modificado, ou seja, $r_{z^-}(x_k) < r_{z^+}(x_k)$. Como p é um caminho de menores rótulos em z^+, temos que $r_{z^-}(x_k) < r_{z^-}(u)$. Sejam x_i e x_j os vértices mais próximos de x_k em p tais que $r_{z^-}(x_i) = r_{z^-}(u)$, $r_{z^-}(x_j) \geq r_{z^-}(u)$ e $i < k < j$ (no pior caso, esses vértices serão os próprios u e v). Obviamente, $\langle x_i, ..., x_k ..., x_j \rangle$ define um caminho de menores rótulos em z^-, absurdo pois, neste caso, pelo Lema 5.1.2, teríamos que $\pi(x_k) < min\{\pi(x_i), \pi(x_j)\}$.

□

Como conseqüência do lema anterior temos que o esquema de eliminação obtido é minimal, já que toda aresta $e \in E_\pi$ é tal que $G_\pi - e$ possui um ciclo de tamanho quatro sem cordas.

5.1.4. Grau Mínimo

A heurística de Grau Mínimo constrói o esquema a partir da primeira posição (ou seja, os vértices selecionados são postos no final do esquema até então construído). Além disso, o próximo vértice a ser ordenado é um dentre aqueles de menor rótulo.

Considere a i-ésima iteração, no momento anterior à escolha do i-ésimo vértice do esquema. Seja $\pi_{i-1} = \langle v_1, ..., v_{i-1} \rangle$ o esquema construído até então e seja G_{i-1} o grafo obtido pela eliminação dos vértices $\{v_1, ..., v_{i-1}\}$ de acordo com π. Para cada vértice w ainda não ordenado no início da i-ésima iteração, o rótulo de w é dado pelo grau de w no grafo G_{i-1}. Desta forma, o próximo vértice escolhido é sempre o de menor grau no grafo obtido pela eliminação dos vértices já ordenados.

Observe o exemplo da Figura 5.4. Os vértices pretos já foram postos no esquema π e o cinza indica o próximo a ser posto. As arestas destacadas foram adicionadas pela eliminação de algum vértice e as pontilhadas não estão sendo consideradas para o valor do rótulo (ou seja, não fazem parte de G_{i-1} como definido acima). Ao final, o grafo com as arestas pontilhadas equivale a G_π.

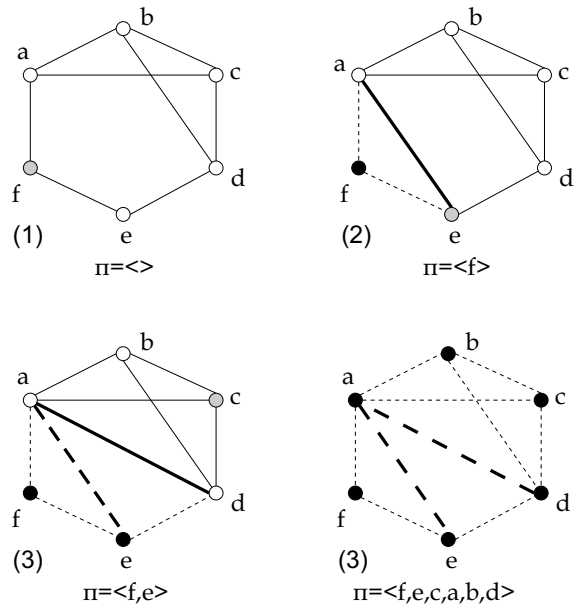

Figura 5.4. *(1)G;(2)Inclusão e "eliminação" de a;(3)Inclusão e "eliminação" de b;(4)Esquema final π e G_π.*

Ressaltamos que este método de rotulação não produz um esquema de eliminação perfeito caso o grafo de entrada seja triangularizado. Como exemplo, observe o grafo da Figura 5.5. Note que o único vértice de menor grau é o vértice destacado de preto e que tal vértice não é simplicial, logo não pode iniciar um esquema de eliminação perfeito.

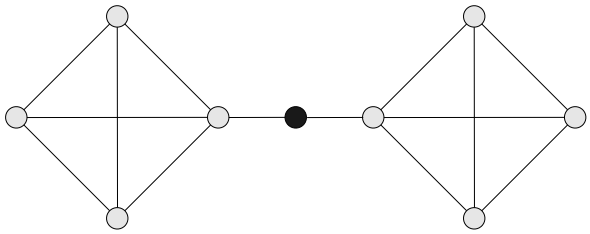

Figura 5.5. *Exemplo em que Grau Mínimo não encontra um esquema perfeito para um grafo triangularizado.*

5.1.5. Preenchimento Mínimo

Dado $v \in V$, o **preenchimento** ("fill-in") de v é o número de arestas que precisam ser adicionadas à vizinhança de v de forma a transformá-la numa clique.

Este método de rotulação adiciona os vértices no esquema a partir da primeira posição, escolhendo o vértice de menor rótulo a cada iteração. O rótulo de um vértice w ainda não ordenado no início da iteração i é igual ao seu valor de preenchimento no grafo G_{i-1} construído como enunciado na subseção anterior.

É fácil ver que se G é cordal, então o esquema de eliminação obtido utilizando este método é perfeito. Basta notar que o rótulo de um vértice qualquer é maior ou igual a zero, e que será zero se e somente se este vértice é simplicial. Logo, como todo grafo triangularizado tem pelo menos um vértice simplicial e ao retirarmos um vértice qualquer deste grafo, o grafo obtido também será triangularizado, temos que a cada iteração i será tomado um vértice simplicial em G_{i-1}.

5.2. Heurística dos Cortes Mínimos

A heurística dos cortes mínimos foi proposta por Koster em sua tese de doutorado [44] e trata-se de uma "heurística de melhoramento", onde partindo de uma decomposição em árvore qualquer dada como entrada, tenta-se melhorar iterativamente sua largura utilizando cortes mínimos.

Dada uma decomposição $D = (\mathcal{X}, T)$, a idéia é substituir cada nó $t \in V(T)$ tal que $|X_t| = la(D) + 1$ por um conjunto de nós cujos conjuntos relacionados sejam menores do que X_t. Se for possível tal substituição, então a largura da decomposição obtida será menor do que $la(D)$. A seguir explicamos como substituir um nó qualquer da árvore de decomposição.

Considera-se que a decomposição de entrada D é minimal no seguinte sentido: para todo $t \in V(T)$, se retirarmos um vértice qualquer do conjunto X_t, D deixa de ser uma decomposição em árvore.

Seja $t \in V(T)$ qualquer. Deseja-se substituir t em D por um nó t^* e um conjunto de nós I^*. Para isso, constrói-se primeiramente o grafo G_t a partir de $G[X_t]$ adicionando-se o seguinte conjunto de arestas:

$$E_t = \{(u, v) : \{u, v\} \subseteq X_t \cap X_{t'},\ \textit{para algum } t' \in N_T(t)\}$$

Koster mostrou que se G_t não é um grafo completo, então é possível substituir o nó t por um conjunto de outros nós $\{t^*, I^*\}$, todos relacionados a subconjuntos de V de cardinalidade inferior a $|X_t|$. Estes nós são definidos a seguir:

- O nó t^* será relacionado com um corte mínimo de vértices do grafo G_t, X^*;
- Sejam $G_1, ..., G_q$ as componentes conexas de $G_t \setminus X^*$. Para cada G_i teremos um novo nó t_i em I^* cujo conjunto relacionado será $X_i^* = V_i \cup X^*$, onde $V_i = V(G_i)$.

Note que para qualquer $t' \in N_T(t)$ temos que $X_\cap = X_t \cap X_{t'}$ é uma clique em G_t. Logo, o conjunto X^* não separa $X_\cap$, ou seja, ou $X_\cap \subseteq X^*$ ou $X_\cap \subseteq X_i^*$ para um único $t_i \in I^*$. Sendo assim, a nova decomposição é obtida da seguinte forma:

1. Retire de T o nó t, guardando seus vizinhos no conjunto N_t;
2. Adicione os nós t^* e $t_i \in I^*$ descritos acima;
3. Adicione em T as arestas (t^*, t_i), para todo $t_i \in I^*$;
4. Para cada $t' \in N_t$, se $X_\cap = X_t \cap X_{t'} \subseteq X^*$, adicione a aresta (t', t^*); caso contrário, adicione a aresta (t', t_i), onde $t_i \in I^*$ é tal que $X_\cap \subseteq X_i^*$.

Para maiores detalhes sobre a heurística, como por exemplo o algoritmo utilizado para encontrar o corte mínimo, recomendamos a leitura de [44, 45].

Algoritmo 5. *MSVS*

Entrada: decomposição em árvore inicial $D = (\mathcal{X}, T)$

Saída: decomposição $D' = (\mathcal{X}', T')$ modificada tal que $la(D') \leq la(D)$

$D' \leftarrow D$ //Escrevemos $D' = (\mathcal{X}', T')$

enquanto existe $t \in T'$ tal que $|X'_t| = la(D') + 1$ e G_t não é uma clique **faça**

$N \leftarrow N_{T'}(t)$

Retire de T' toda aresta incidente em t

Seja S um corte mínimo em G_t

Adicione em $V(T')$ o nó t^* relacionado ao conjunto S

Sejam $G_1, ..., G_q$ as componentes conexas de $G_t \setminus S$

para $i = 1, ..., q$ **faça**

Adicione em $V(T')$ o nó t_i relacionado ao conjunto $V(G_i) \cup S$

$E(T') \leftarrow E(T') \cup \{(t_i, t^*)\}$

para cada $t' \in N$ **faça**

se existe t_i tal que $X_{t'} \cap V(G_i) \neq \emptyset$ **então**

$E(T') \leftarrow E(T') \cup \{(t_i, t')\}$

senão

$E(T') \leftarrow E(T') \cup \{(t^*, t')\}$

retorne D'

5.3. Heurística GRASP

O método GRASP [34, 35] (Greedy Randomized Adaptive Search Procedure) é uma meta-heurística iterativa onde cada iteração constitui-se de duas fases: na primeira, uma solução s é construída de maneira gulosa e aleatória (levando-se também em consideração determinadas características adaptativas); na segunda, define-se um espaço de soluções, denominado vizinhança de s, e faz-se uma busca neste espaço até que uma solução ótima local seja encontrada. Estas duas fases são executadas até que um dado critério de parada seja alcançado.

Nesta seção, introduzimos primeiramente como funciona o método GRASP propriamente dito e, em seguida, explicamos como o método foi aplicado para encontrar um esquema de eliminação dos vértices do grafo dado como entrada.

5.3.1. A meta-heurística Grasp

A seguir, apresentamos um algoritmo genérico para o método. A função $BuscaLocal(s)$ retorna uma solução ótima local alcançada a partir de s. Representamos por s^* a melhor solução até então encontrada.

Algoritmo 6. *GRASP*

Entrada: Instância do problema em questão
Saída: Melhor solução encontrada
  enquanto critério de parada não é alcançado **faça**
    Encontre uma solução gulosa aleatória adaptativa s
    $s \leftarrow BuscaLocal(s)$
    se valor de s é melhor do que o valor de s^* **então**
      $s^* \leftarrow s$
  retorne s^*

O critério de parada pode basear-se no tempo, número de iterações, ou mesmo no valor da solução, parando quando um valor determinado for alcançado. A vizinhança de

uma solução s é geralmente constituída de um conjunto de soluções facilmente obtidas a partir de s. Além disso, s é considerada uma solução ótima local se nenhuma das soluções vizinhas de s tem valor inferior a s, ou superior, caso o problema considerado seja de maximização. Daqui em diante, consideraremos que o problema tratado é de minimização.

A obtenção de uma solução no início de cada iteração do método deve ser, como já foi dito anteriormente, gulosa, aleatória e adaptativa. Basicamente, uma solução s é vista como um conjunto onde cada elemento tem influência sobre o valor final de s. A esta influência chamaremos de custo do elemento. Desta forma, uma maneira de construir uma solução s gulosa é tomar um elemento que aumente minimamente o valor de s (ou de menor custo) até que s se torne uma solução viável. Ao tomar-se um elemento, o custo dos outros pode ser modificado, logo faz-se necessária uma reavaliação dos custos (este é o caráter adaptativo). Porém, como não queremos uma solução puramente gulosa, o que é feito na verdade é construir uma lista de elementos candidatos a entrarem na solução baseados nos valores de custo e escolher aleatoriamente um dentre tais elementos. A seguir apresentamos um algoritmo genérico para a obtenção de uma solução gulosa aleatória adaptativa.

Algoritmo 7. *Solução GAA*

$s \leftarrow \emptyset$

Avalie o custo de cada elemento e

enquanto s não é uma solução viável **faça**

  Construa uma lista de candidatos RCL contendo os elementos com menores custos

  Escolha randomicamente um elemento e em RCL

  $s \leftarrow s \cup \{e\}$

  Reavalie os custos dos elementos que não estão em s

retorne s

5.3.2. A heurística GRASP para triangularização

Novamente, abordaremos o problema de triangularização, ao invés de trabalhar diretamente com decomposições, e uma solução considerada viável nada mais é do que uma permutação dos vértices de G, que pode também ser vista como um esquema de eliminação. O valor de uma solução π é dado pela largura em árvore do grafo G_π (ou seja, por $max_{v \in V} N_\pi(v)$). Denotaremos o valor de π por $valor(\pi)$.

Na obtenção de uma solução gulosa aleatória adaptativa utilizamos novamente os métodos de rotulação (mais precisamente, utilizamos o método $MCSM$), fazendo com que os custos sejam exatamente os rótulos atribuídos a cada vértice. Desta forma, a lista de candidatos a cada iteração é formada pelos vértices cujo custo (ou rótulo) tem valor máximo, um dentre estes será escolhido aleatoriamente para entrar no esquema e uma reavaliação dos custos é feita, ou seja, os rótulos são atualizados (como é feito em $MCSM$, o vértice é posto no começo do esquema até então construído).

Note que a diferença entre a obtenção de um solução inicial como descrito no parágrafo anterior difere da heurística de rotulação unicamente pela aleatoriedade da escolha do vértice. Pode-se argumentar que essa aleatoriedade também está presente na rotulação, já que dentre os vértices de melhor rótulo não se adota nenhum critério de escolha. Porém, se não for empregado um método de escolha aleatória nesta fase, é bastante provável que em todas as iterações da heurística GRASP seja gerada a mesma solução inicial. Em outras palavras, a aleatoriedade garante a diversidade de soluções iniciais investigadas.

Definimos agora como é feita a busca local. Seja π um esquema de eliminação qualquer. Dizemos que π' está na vizinhança de π, $N(\pi)$, se pode ser obtido trocando-se exatamente dois elementos de posição em π. O esquema π será uma solução ótima local se não possuir nenhum esquema vizinho cujo valor seja melhor que o seu. Logo, a busca trata-se simplesmente de percorrer a vizinhança de π por uma solução melhor. Caso exista, passa-se a buscar na vizinhança desta solução. Caso contrário, podemos concluir que π é uma solução ótima local.

Porém, como apenas nos interessa percorrer as soluções vizinhas cujo valor seja menor

do que o valor de π, tentaremos evitar aquelas cujo valor sabe-se que é maior ou igual ao de π. Para isso, definiremos índices no esquema que nos indicarão que pares de elementos podem valer a pena transpor.

Seja $u \in V$ qualquer e $v \in N_\pi(u)$, Seção 4.1). Pelo Teorema 5.1.1, se $v \notin N_G(u)$ então existe um caminho $\langle v, v_1, ..., v_q, u\rangle$ em G tal que $\pi(v_i) < \pi(u)$, para $i = 1, ..., q$. Logo, a única forma de fazer com que v não esteja mais na pós-vizinhança de u é fazendo uma transposição de algum dos vértices internos deste caminho com algum outro vértice w tal que $\pi(w) > \pi(u)$ (na verdade, seria necessário transpor pelo menos um vértice de cada caminho deste tipo).

Seja $M = \{\pi(v) : |N_\pi(v)| = valor(\pi)\}$ o conjunto formado pelos índices em π dos vértices cuja pós-vizinhança tem cardinalide igual ao valor de π. Definimos os índices: $min(\pi) = min\{i : i \in M\}$ e $max(\pi) = max\{i : i \in M\}$. Pelo o que foi dito no parágrafo anterior e pelo fato de que $valor(\pi) = max_{v \in V} N_\pi(v)$, percebemos que transpor quaisquer dois vértices u e v tais que $\pi(u) < max(\pi)$ e $\pi(v) < max(\pi)$ não irá melhorar o valor de π, pois a pós-vizinhança do vértice em $max(\pi)$ continuará a mesma. Analogamente, transpor dois vértices tais que $\pi(u) > min(\pi)$ e $\pi(v) > min(\pi)$ também não diminui o valor de π. Temos então que os vizinhos de π que podem ter valor inferior ao de π são tais que diferem de π pela transposição de dois vértices u e v tais que $\pi(u) \leq min(\pi)$ e $\pi(v) \geq max(\pi)$. O tamanho da vizinhança $N(\pi)$ a ser percorrida é agora tal que $|N(\pi)| = min(\pi) * (n - max(\pi) + 1) = O(n^2/4)$.

Finalmente, utilizamos também um método em conjunto com GRASP que explora caminhos entre duas soluções. Este método é conhecido como "Path Relinking" e foi primeiramente proposto por Glover [38] para utilização com busca tabu. Trata-se de percorrer um caminho partindo de uma determinada solução, efetuando determinadas operações (no nosso caso, transposições) até chegar-se à solução destino. Ao final, o método retorna a melhor solução intermediária. O momento de executar o Path Relinking é arbitrário; pode-se optar por fazer uma vez a cada iteração do GRASP, a cada intervalo de x iterações, sempre que uma solução melhor for encontrada, ou mesmo rodar a cada

iteração com uma probabilidade p. Decidimos executar este método sempre que uma solução melhor for encontrada, ou a cada 15 iterações.

6

Limites Inferiores

Neste capítulo veremos os métodos mais conhecidos para o cálculo de limites inferiores para a largura em árvore de um grafo qualquer.

6.1. Clique Máxima

Pelo Lema 3.2.1, sabemos que para toda clique C de G e toda decomposição em árvore $D = (X, T)$ de G, existe um nó $t \in T$ tal que $C \subseteq X_t$. Logo, certamente $la(D) \geq \omega(G)-1$. Desta forma, o tamanho da clique máxima fornece um limite inferior para a largura de qualquer decomposição de G. Temos, porém, que calcular $\omega(G)$ para um grafo G qualquer é NP-difícil, apesar de haver muitos resultados que mostram que para grafos não tão grandes ou densos, $\omega(G)$ pode ser calculado em um tempo viável [8, 52].

6.2. Máximo Grau Mínimo

Dado um grafo G qualquer, nesta seção apresentamos um método para calcular um limite inferior para $la(G)$ baseado no fato de que $\delta(G) \leq la(G)$ e no Teorema 3.1.2. Basicamente, a idéia é encontrar o subgrafo de G cujo grau mínimo é máximo dentre os graus mínimos de todos os subgrafos de G. No parágrafo a seguir mostramos que $\delta(G)$ é realmente um limite inferior para $la(G)$.

Seja $D = (X, T)$ uma decomposição em árvore de G de largura mínima. Sejam $t \in T$ uma folha qualquer de T e t' o nó vizinho a t em T. Se $X_t \subseteq X_{t'}$, é fácil ver que ao retirar-se o nó t, ainda resta uma decomposição em árvore de largura ótima, pois não há arestas ou vértices cobertos por t que também não o são por t'. Pode-se considerar, então, que $X_t \setminus X_{t'} \neq \emptyset$. Logo, temos que para v qualquer em $X_t \setminus X_{t'}$, certamente $N_G(v) \subseteq X_t$, ou seja, temos que $d(v) \leq la(D) = la(G)$. Como $d(v) \geq \delta(G)$, temos que $la(G) \geq \delta(G)$.

Pelo Teorema 3.1.2, sabemos que para todo menor H de G temos que $la(H) \leq la(G)$. Como um subgrafo qualquer G' de G também é um menor de G, temos que $la(G) \geq la(G')$. Além disso, sabemos que $la(G') \geq \delta(G')$. Logo, temos que $la(G) \geq \delta(G')$, para todo subgrafo G' de G.

Seja Λ o conjunto de todos os subgrafos de G. Definimos $MMD(G) = max_{G' \in \Lambda} \delta(G')$ (este limite é conhecido na literatura como "Maximum Minimum Degree"). O lema a seguir nos mostra que este limite pode ser calculado de forma mais simples do que parece a primeira vista.

Lema 6.2.1. *$MMD(G) = max\{\delta(G), MMD(G - v)\}$, para todo $v \in V$ tal que $d(v) = \delta(G)$.*

Prova: Sejam Λ o conjunto de todos os subgrafos de G e $v \in V$ um vértice qualquer de grau mínimo em G. É fácil ver que Λ pode ser particionado em Λ_1 e Λ_2, onde Λ_1 contém todos os subgrafos que contém v e Λ_2 todos os que não contém. Obviamente, para qualquer subgrafo $G' \in \Lambda_1$ temos que $\delta(G') \leq d(v) = \delta(G)$. Além disso, para qualquer $G' \in \Lambda_2$ temos que G' é subgrafo de $G - v$, logo $\delta(G') \leq MMD(G - v)$. □

O Algoritmo 8 segue diretamente do lema acima.

Como uma clique também é um subgrafo de G, temos que o limite fornecido por $MMD(G)$ é sempre maior ou igual ao tamanho da clique máxima, $\omega(G)$. Na verdade, é possível mostrar que $MMD(G) \geq \chi(G) - 1$ [59], onde $\chi(G)$ é o número cromático do grafo G (recomendamos [29] para a definição de número cromático).

Algoritmo 8. *MMD*

Entrada: Grafo G

Saída: máximo grau mínimo de G (ou seja, MMD(G))

$G' \leftarrow G$

$mmd \leftarrow 0$

enquanto $V(G') \neq \emptyset$ **faça**

Seja $v \in V(G')$ de grau mínimo em G'

$mmd \leftarrow max\{mmd, d(v)\}$

$Remove(G', v)$ //remove v do grafo G'

retorne mmd

6.3. Melhorando o Limite Inferior

Clautiaux, Carlier, Moukrim e Négre [21] propuseram um método que pode ser utilizado juntamente com algum outro método para encontrar um limite inferior de forma a melhorar tal limite. Para desenvolverem tal método, eles se basearam em propriedades encontradas por Bodlaender [12].

Sejam um grafo $G = (V, E)$ qualquer e k um inteiro positivo. O **grafo melhorado com x vizinhos comuns** é o grafo G_x^{imp} obtido de G adicionando uma aresta (u, v) para todo par $u, v \in V$ tal que u e v têm pelo menos $x + 1$ vizinhos comuns.

Lema 6.3.1 (Bodlaender [12]). *Se $la(G) \leq k$, então $la(G_k^{imp}) \leq k$. Além disso, qualquer decomposição em árvore para G de largura no máximo k é também uma decomposição em árvore para G_k^{imp} de largura no máximo k, e vice-versa.*

Sejam LB um método qualquer para calcular um limite inferior para $la(G)$ e lb o limite calculado usando LB. O método apresentado utiliza-se do lema acima para melhorar este limite da seguinte forma:

1. Suponha que $la(G) = lb$

2. Calcule G_{lb}^{imp}

3. Calcule um limite inferior lb' para $la(G_{lb}^{imp})$ usando o método LB

4. Se $lb' > lb$, temos uma contradição em 1, ou seja, $la(G) > lb$. Logo podemos atualizar lb para $lb + 1$

5. Repete-se os passos acima até que não seja mais encontrada uma contradição

Um outro tipo de grafo, também definido primeiramente por Bodlaender [12], pode ser usado no algoritmo descrito acima. Este adiciona arestas (u, v) para todo par de vértices conectados por pelo menos $x + 1$ caminhos disjuntos em vértices. Chamamos tal grafo de **grafo melhorado com x caminhos disjuntos**. Um lema análogo ao Lema 6.3.1 também é válido, logo o algoritmo pode ser corretamente empregado utilizando este outro tipo de grafo.

Em [21] foi feito um estudo computacional para analisar a qualidade deste novo método aplicando-o juntamente com MMD. Para muitas instâncias o limite inferior encontrado é superior ao anteriormente conhecido quando o grafo melhorado com caminhos disjuntos é utilizado. Utilizando o grafo melhorado com vizinhos comuns houve também algumas melhoras, porém em um menor número de instâncias, além do aumento ter sido menos significativo.

7

Método Enumerativo

Neste capítulo, mostramos como enumerar as soluções do problema utilizando o conceito de ordens parciais visto na Seção 2.3. Além disso, expomos como obter um limite inferior para $la(G)$ a partir desta enumeração.

Dado um grafo qualquer $G = (V, E)$, seja P_V a ordem parcial contendo apenas os pares reflexivos vv, para todo $v \in V$. Obviamente, $Ext_t(P_V)$ é igual ao conjunto de todas as ordens totais de V (ou todos os esquemas de eliminação de V). Logo, uma maneira de resolver o problema de forma exata é enumerar todo o conjunto $Ext_t(P_V)$ e tomar P^* tal que $\omega(G_{P^*})$ é mínimo.

Porém, sabe-se que o tamanho de $Ext_t(P)$ para uma ordem P qualquer pode ser muito grande, tornando este método inviável. De fato, determinar o tamanho desse conjunto é um problema $\#\mathcal{P}$-completo [20].

Apesar disso, se for possível enumerar este conjunto em uma estrutura de árvore, a aplicação do método "branch and bound" pode tornar a enumeração viável (para um estudo mais detalhado acerca do método recomendamos [51]). Ressaltamos que o motivo pelo qual escolhemos o algoritmo de Corrêa e Szwarcfiter [23] é justamente o fato desse gerar as extensões em uma estrutura de árvore. Além desse, estudamos também o algoritmo de Ruskey [58], porém descartamos a sua utilização pois gera as extensões totais de uma ordem parcial P percorrendo um ciclo hamiltoniano no grafo de transposição de P, que

possui dois vértices relacionados a cada uma das extensões totais de P (recomendamos a leitura do referido artigo para maiores detalhes). Logo, para concluir que uma solução é ótima seria necessário que todo o grafo de transposição fosse percorrido (ou seja, que todo o conjunto $Ext_t(P)$ fosse gerado duas vezes).

Como já falamos anteriormente, ao contrário do algoritmo de Ruskey, o algoritmo de Corrêa e Swarcfiter permite a utilização do método "branch and bound". Este método possibilita que subárvores inteiras não precisem ser geradas, diminuindo a quantidade de soluções percorridas. Na verdade, mesmo que o método não seja executado até que se conheça uma solução ótima, um limite inferior pode ser obtido. A seguir explicamos brevemente como isso é possível.

Dada uma ordem parcial P de V, primeiramente definiremos um limite inferior para $la(G_{P'})$, onde P' é uma extensão total de P (relembramos que, neste caso, $la(G_{P'}) = \omega(G_{P'}) - 1 = max_{v \in V}|N_{P'}(v)|$). Como $P \subseteq P'$, para toda ordem $P' \in Ext_t(P)$, temos que $N_P(u) \subseteq N_{P'}(u)$, para todo $u \in V$. Logo, $max_{u \in V}|N_P(u)|$ é um limite inferior para $la(G_{P'})$, para toda ordem $P' \in Ext_t(P)$. Denotaremos este valor por $inf(P)$.

Considere que seja possível enumerar o conjunto $Ext_t(P_V)$ percorrendo uma árvore enraizada $\mathcal{T}$ tal que a raiz da árvore representa a ordem P_V, cada nó representa uma extensão de P_V e cada folha representa uma extensão total de P_V. Além disso, dado um nó interno P da árvore, a subárvore enraizada em P contém todas as extensões totais de P. Na seção seguinte, veremos que isto é possível com a modificação proposta sobre o algoritmo de Corrêa e Szwarcfiter [23].

Percorremos esta árvore da seguinte forma: marcamos a raiz para ser visitada; a cada iteração, escolhemos um nó P marcado para visita tal que $inf(P)$ é mínimo; em seguida, desmarcamos P e marcamos seus filhos para serem visitados.

Uma forma de diminuir o número de nós percorridos é a seguinte (método "branch and bound"): inicialmente tomamos um limite superior sup^* para $la(G)$ (este valor pode ser $la(G_\pi)$, para um esquema de eliminação π qualquer, ou mesmo $|V|$); a cada iteração, marca-se apenas os nós cujos limites inferiores são no máximo sup^*; se chegarmos em uma

folha, atualizamos o valor de sup^* e desmarcamos todo nó P tal que $inf(P) > sup^*$. Se isto for feito, temos que todo nó P visitado é tal que $inf(P) \leq la(G)$. O parágrafo a seguir prova esta afirmação.

Suponha, por absurdo, que em uma iteração qualquer o nó tomado P é tal que $inf(P) > la(G)$. Então, certamente não existe P' marcado para visita tal que $inf(P') \leq la(G)$. Como a árvore contém todas as ordens totais de V, existe uma folha P^* tal que $la(G_{P^*}) = la(G)$. Além disso, certamente toda extensão P' no caminho entre P^* e a raiz P_V é tal que $inf(P') \leq inf(P^*) \leq la(G_{P^*}) = la(G)$. Pela forma como estamos percorrendo $\mathcal{T}$ e pelo fato de $\mathcal{T}$ ser conexa, temos que este caminho deve ter sido totalmente percorrido. Logo, a folha P^* já foi visitada e o valor sup^* foi atualizado, ou seja, no momento da escolha de P temos que $sup^* \leq la(G_{P^*})$, absurdo pois neste caso P teria sido desmarcado para visita.

Infelizmente, o método descrito acima pode não ser suficiente para diminuir o número de nós a serem percorridos de forma a tornar o método viável. Além disso, de um nível para outro na árvore, a diferença no limite inferior cresce lentamente, fazendo necessário que uma grande quantidade de nós precise ser percorrida até que obtenha-se um limite inferior significativo. Uma solução seria melhorar o limite inferior de cada nó. Porém, como os métodos existentes muitas vezes fornecem um limite inferior muito distante do possível limite superior inicial (obtido com uma das heurísticas para limite superior descritas no capítulo anterior), provavelmente isto não resolveria o problema. Algo interessante seria, então, limitar o espaço de busca, ou seja, não percorrer todo o conjunto $Ext_t(P_V)$, mas apenas um subconjunto $Ext_t(P) \subseteq Ext_t(P_V)$, para uma dada ordem parcial P. Isso certamente não fornece um limite inferior para $la(G)$, porém pode ser uma boa ferramenta para estudar que melhor "caminho" tomar quando da construção de uma ordem total. Além disso, se for possível provar que existe uma solução ótima tal que a ordem total relacionada P^* contém determinados pares ordenados, então pode-se partir da ordem parcial contendo estes pares sem eliminar o ótimo, ou seja, sem prejudicar a obtenção da solução ótima. Infelizmente, o único resultado neste sentido é o de que se existir um

vértice simplicial u então existirá uma ordem total ótima tal que u é o vértice minimal [15].

A Seção 7.1 descreve o algoritmo de Corrêa e Szwarcfiter para gerar o conjunto de extensões de uma ordem P qualquer, além da modificação proposta para gerar apenas as ordens totais, e a Seção 7.2 propõe alguns métodos para o cálculo de uma ordem parcial inicial.

7.1. Geração de ordens totais

Nesta seção, veremos o algoritmo para geração de extensões de uma ordem parcial proposto por Corrêa e Szwarcfiter [23]. Tal algoritmo gera todas as extensões de uma dada ordem parcial $P \subseteq C^2$ utilizando o conceito de pares passivos e tem complexidade $O(m + n\mu)$, onde $m = |P|$, $n = |C|$ e $\mu = |Ext(P)|$. Aos leitores que desejarem conhecer as provas da corretude deste algoritmo, recomendamos a leitura do artigo previamente citado.

Precisaremos de alguns novos conceitos relacionados a ordens parciais, vistos a seguir.

O **inverso** do par ordenado $p = uv$ é o par $\overline{p} = vu$.

Seja $P \subseteq C^2$ uma ordem parcial. Um par ordenado $uv \in I(P)$ é dito **passivo** se $P \cup \{uv\}$ também é uma ordem parcial sobre C. Observe o exemplo na Figura 7.1. Denotamos o conjunto de todos os pares passivos de P por $W(P)$ e por $P + uv$ a ordem $P \cup \{uv\}$.

Seja $R \subseteq I(P)$. Denotamos por $Ext(P, R)$ o conjunto de extensões de P que não contêm pares pertencentes a R. Desta forma, $Ext(P, \emptyset)$ é o conjunto de todas as extensões de P.

O teorema seguinte descreve a idéia central do método proposto por Corrêa e Szwarcfiter para calcular $Ext(P, \emptyset)$.

Teorema 7.1.1. *Sejam $R \subseteq I(P)$ e $Ext(P, R)$ o conjunto de extensões de P excluindo R. Então,*

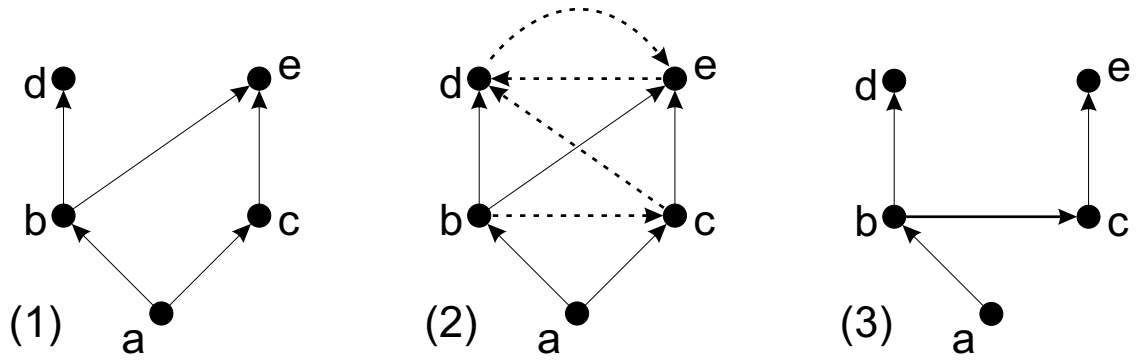

Figura 7.1. *Exemplo de pares passivos (omitimos as transitividades):* ***(1)*** *Ordem parcial P;* ***(2)*** *Pares passivos representados pelas setas pontilhadas;* ***(3)*** *Ordem parcial $P + bc$.*

- *Se $W(P) \subseteq R$, então $Ext(P, R) = \{P\}$*
- *Caso contrário, seja $W - R = \{w_1, ..., w_\ell\}$,*

$$Ext(P, R) = \{P\} \cup_{q=1...\ell} Ext(P + w_q, R \cup \{w_1, ..., w_{q-1}\}) \tag{7.1}$$

Além disso, os subconjuntos acima formam uma partição de $Ext(P, R)$.

O algoritmo para calcular as extensões de P consiste então em calcular o conjunto de pares passivos e, recursivamente, calcular as extensões da ordem $P + w$, para cada $w \in W(P)$, excluindo determinados pares passivos. Observe, por exemplo, a figura 7.2.

Porém, não estamos interessados em conhecer todo o conjunto $Ext(P)$, mas apenas um subconjunto deste, $Ext_t(P)$. Devido a isto, adaptamos o teorema para se adequar melhor às nossas necessidades.

Seja P uma ordem parcial qualquer. Considere $R = \emptyset$ e $W(P) = \{w_1, ..., w_\ell\} \neq \emptyset$. Note que dado uma extensão total $P' \in Ext_t(P)$ e $ab \in I(P)$ qualquer, temos que ou o par ordenado $ab \in P'$ ou $ba \in P'$, mas não ambos. Desta forma, para qualquer par passivo $w_q \in W(P)$, se $P' \in Ext_t(P + w_q, \{w_1, ..., w_{q-1}\})$, então $\overline{w}_j \in P'$, para $j = 1, ..., q - 1$. A seguir, reformulamos (7.1) em função de P_q e Γ que serão definidos mais precisamente em (7.3) e (7.4).

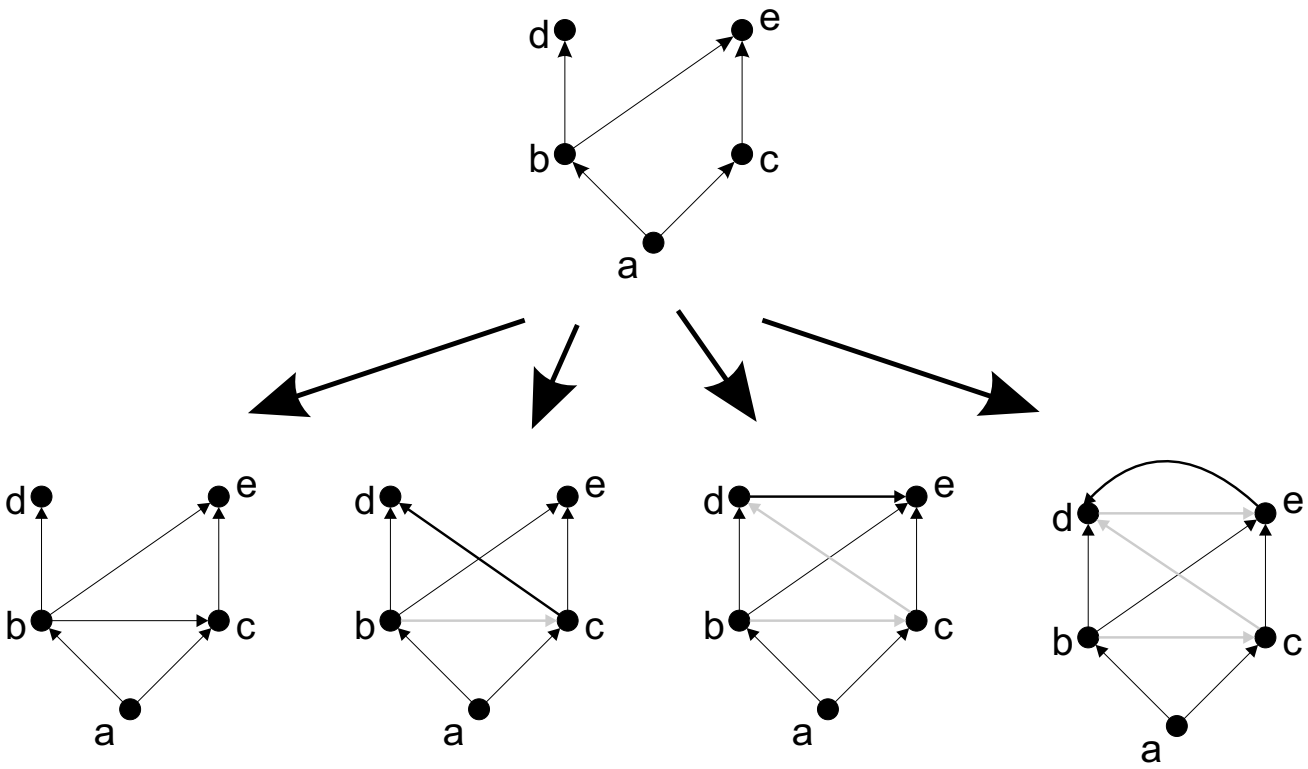

Figura 7.2. *Exemplo de geração dos filhos de uma extensão P na árvore.*

$$Ext_t(P) = \begin{cases} \{P\}, & se\ W(P) = \emptyset \\ \bigcup_{q\in\Gamma} Ext_t(P_q), & c.c. \end{cases} \tag{7.2}$$

onde $\Gamma \subseteq \{1, ..., \ell\}$ e, para todo $q \in \Gamma$, P_q é uma ordem parcial que contém P, w_q e $\overline{w}_j$, para todo $j \in \{1, ..., q-1\}$. Note que o conjunto W é vazio se e somente se P é uma ordem total .

A definição precisa de P_q e Γ depende da seguinte relação definida recursivamente:

$$P'_q = \begin{cases} P, & se\ q = 1 \\ P'_{q-1} + \overline{w}_{q-1}, & c.c \end{cases}$$

Observe que, por definição, $P'_q \subseteq P_q$. Observe ainda que, para $q > 1$, não se sabe se P'_q é ou não uma ordem parcial, pois mesmo que P'_{q-1} seja uma ordem parcial, não se tem certeza que $\overline{w}_{q-1}$ seja passivo nessa ordem. Mais precisamente, considerando-se que P'_{q-1} seja uma ordem parcial (a base é correta pois $P'_1 = P$), os seguintes casos podem ocorrer, para $q > 1$:

1. P'_q viola transitividade: obviamente o par ordenado que viola transitividade trata-se de $\overline{w}_{q-1}$, já que P'_{q-1} é uma ordem parcial.

2. P'_q viola anti-simetria: neste caso, temos que $q > 2$ e $w_{q-1} \in P'_{q-1}$. Logo, P'_q não define uma ordem parcial e, obviamente, não existe uma extensão total P' de P contendo P'_q. Como $P'_q \subseteq P'_k \subseteq P_k$, para todo $k \in \{q, ..., \ell\}$, temos que P_k também não define uma ordem parcial, para todo $k \in \{q, ..., \ell\}$.

Pelo o que vimos acima, definimos o inteiro κ e a ordem parcial P^*_q como a seguir. Estes parâmetros servirão para o cálculo do conjunto Γ e da ordem P_q.

$$\kappa = \begin{cases} min\{q : w_{q-1} \in P'_{q-1}\}, & se\ isso\ ocorrer\ para\ algum\ q \in \{1, ..., \ell\} \\ \ell + 1, & c.c \end{cases}$$

$$P_q^* = \begin{cases} P, & se\ q = 1 \\ P_{q-1}^* \cup fecho(P_{q-1}^*, \overline{w}_{q-1}), & se\ q \in \{2, ..., \kappa\} \end{cases}$$

Analisaremos agora o que é necessário fazer para calcular corretamente P_q a partir de P_q^*.

- $P_q^* + w_q$ viola transitividade: basta adicionarmos o fecho transitivo do par w_q.
- $P_q^* + w_q$ viola anti-simetria: neste caso, não existe uma ordem total P' de P tal que $P' \in Ext_t(P + w_q, \{w_1, ..., w_{q-1}\})$, pois se $\overline{w}_q \in P_q^*$, certamente fora adicionado por estar contido no fecho transitivo de algum dos pares $\{\overline{w}_1, ..., \overline{w}_{q-1}\}$, ou seja, o par $\overline{w}_q$ é imprescindível para manter transitividade.

Finalmente, podemos definir P_q e Γ:

$$P_q = P_q^* \cup fecho(P_q^*, w_q) \tag{7.3}$$

$$\Gamma = \{q \leq \kappa : P_q^* + w_q\ viola\ anti - simetria\} \tag{7.4}$$

Note que o uso de 7.2 acelera a geração das extensões totais, já que deixa de enumerar as extensões parciais em cujas subárvores não existem folhas que sejam ordens totais, além de que adiciona mais pares ordenados às extensões quando da subdivisão do problema, diminuindo assim o número de pares incomparáveis em cada subproblema gerado.

A Figura 7.3 mostra a subdivisão vista na Figura 7.2 com a modificação. A seta cinza indica que o filho não será gerado e que nenhum que viria após este também será gerado. Note que isso ocorre pois após a inclusão do par cb, o par cd torna-se necessário. Fornecemos o conjunto $Ext_t(P)$. Note que $\langle a, b, c, d, e\rangle$, $\langle a, b, c, e, d\rangle$ e $\langle a, b, d, c, e\rangle$ são extensões do filho mais à esquerda, enquanto que $\langle a, c, b, d, e\rangle$ é extensão do filho do meio.

O Algoritmo 9 calcula o conjunto de todas as extensões totais de uma ordem parcial P. No algoritmo, a função $ParesPassivos(P)$ computa o conjunto de pares passivos da ordem P e $Fecho(Q, uv)$ retorna o fecho transitivo do par uv na ordem fornecida, Q.

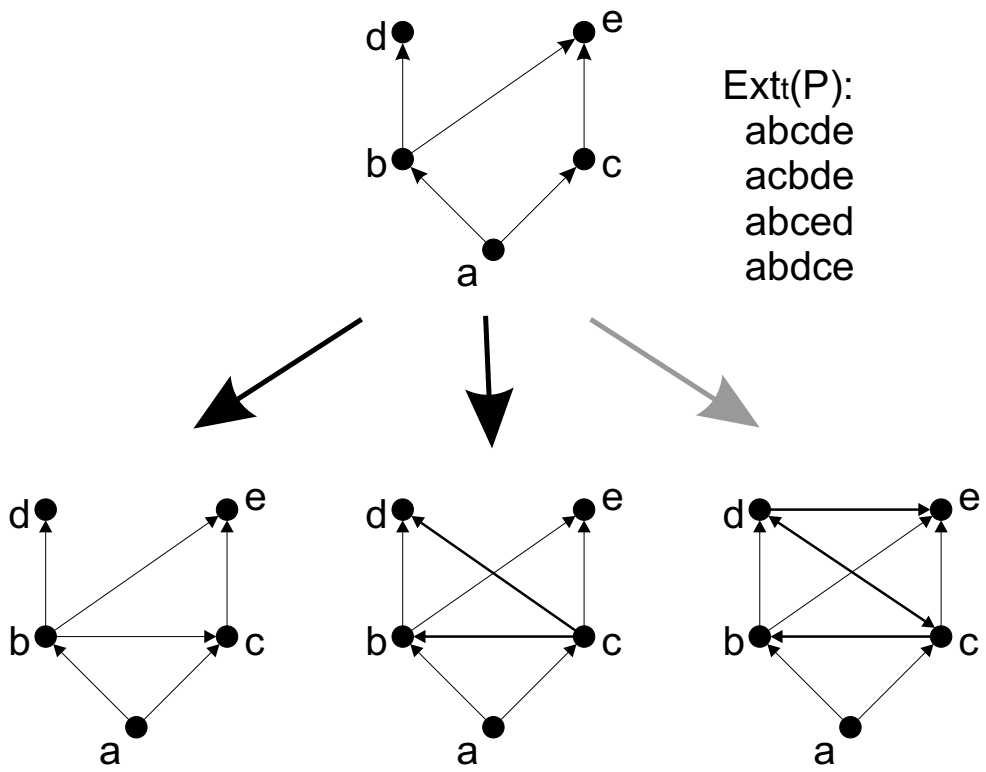

Figura 7.3. *Exemplo de geração dos filhos de uma extensão P na árvore após a modificação proposta.*

Algoritmo 9. *ExtTotaisParesPass*

Entrada: ordem parcial P

Saída: conjunto E com as extensões totais de P

se P é uma ordem total **então**

retorne $\{P\}$

$W \leftarrow ParesPassivos(P)$

$P^* \leftarrow P$

$E \leftarrow \emptyset$

para todo $uv \in W$ **faça**

se $vu \notin P^*$ **então**

$E \leftarrow E \cup ExtTotaisParesPass(P^* \cup Fecho(P^*, uv))$

se $uv \notin P^*$ **então**

$P^* \leftarrow P^* \cup Fecho(P^*, vu)$

senão

retorne E // Neste caso, κ foi "encontrado"

retorne E

A dificuldade agora reside em calcular os pares passivos a cada iteração. Os conceitos e resultados a seguir servem para obter um melhor desempenho no cálculo dos pares passivos das extensões geradas e foi proposto por Corrêa e Szwarcfiter ainda no mesmo artigo em que propuseram o algoritmo.

Sejam $\{u, v, w\} \subseteq C$ elementos distintos tais que $uv \in I(P)$. Dizemos que w é um elemento **ativo** para uv caso uma das situações abaixo ocorra:

$$wv \in P \text{ e } uw \in I(P) \tag{7.5}$$

$$wv \in I(P) \text{ e } wu \in P \tag{7.6}$$

Observe os dois casos em que um elemento é considerado ativo na Figura 7.4 e note que a inclusão do par uv violaria a transitividade da relação. Ou seja, um elemento ativo

é um elemento que "impede" a inclusão do par uv na ordem. Denotamos por $A_{uv}(P)$ o conjunto dos elementos ativos do par uv na ordem P.

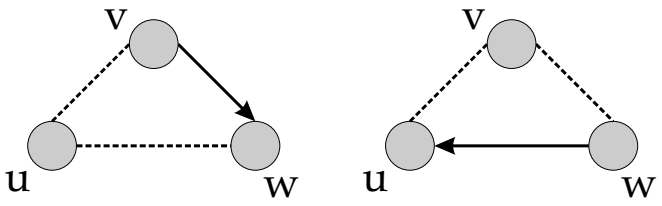

Figura 7.4. *Ilustração das Equações 7.5 e 7.6.*

O lema a seguir define os pares passivos de uma ordem em termos dos respectivos conjuntos de elementos ativos.

Lema 7.1.1. *Um par $uv \in I(P)$ é passivo se e somente se $A_{uv}(P) = \emptyset$.*

Agora queremos caracterizar as mudanças ocorridas no conjunto de pares passivos de uma ordem P ao adicionarmos um par uv qualquer. Ocorre que este conjunto tanto pode perder quanto ganhar elementos devido à inclusão de um único par xy em P. Pelo lema anterior percebemos que basta mantermos, para cada par ordenado $uv \in I(P)$, o conjunto $A_{uv}(P)$ e observarmos as mudanças ocorridas nele. Caso tal conjunto torne-se vazio, teremos que uv passa a ser passivo. Caso $A_{uv}(P)$ seja vazio e ganhe algum elemento, uv deixa de ser passivo.

O lema a seguir identifica, para um dado par incomparável uv, os elementos que deixam de ser ativos com relação a uv após a inclusão de um par passivo qualquer.

Lema 7.1.2. *Seja xy um par passivo em P, uv um par incomparável em $P + xy$ e $w \in C$. Então $w \in A_{uv}(P) \setminus A_{uv}(P + xy)$ se e somente se*

$$x = u,\ y = w \text{ e } vw \in P, \text{ ou}$$

$$x = w,\ y = v \text{ e } wu \in P$$

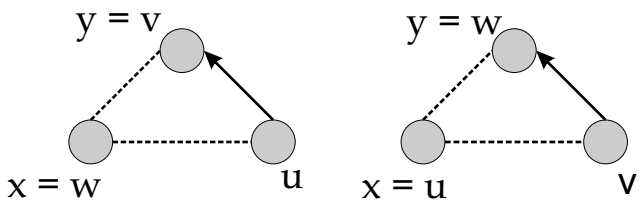

Figura 7.5. *Ilustração das condições do Lema 7.1.2.*

O lema a seguir identifica, para um dado par incomparável uv, os elementos que passam a ser ativos com relação a uv após a inclusão de um par passivo qualquer.

Lema 7.1.3. *Seja xy um par incomparável em P, uv um par passivo em $P+xy$ e $w \in C$. Então $w \in A_{uv}(P + xy) \setminus A_{uv}(P)$ se e somente se*

$$x = v, y = weuw \in I(P), ou$$

$$x = w, y = uewv \in I(P)$$

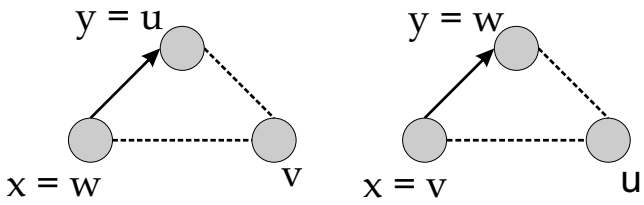

Figura 7.6. *Ilustração das condições do Lema 7.1.3.*

Na verdade, é suficiente manter, para cada par incomparável uv, o parâmetro a_{uv} que indica $|A_{uv}|$ (pares maiores detalhes, consulte [23]). Caso a_{uv} seja 0 no momento em que a função $ParesPassivos(P)$ é chamada no Algoritmo 9, então o par uv é retornado juntamente com todos os outros cujo valor associado também seja 0.

Observamos que as provas dos lemas acima não necessariamente exigem que o par adicionado seja passivo. Logo, as mesmas modificações podem ser aplicadas no nosso

caso, onde algumas vezes adicionaremos pares não passivos juntamente com todo o seu fecho transitivo. Para maiores detalhes, recomendamos a leitura do artigo [23].

7.2. Limitando o espaço de busca: Ordens Parciais

Como já foi dito anteriormente, uma alternativa para a utilização da enumeração proposta seria restringir o espaço de busca inicial para o conjunto de extensões totais de uma ordem parcial que contivesse outros pares além dos reflexivos. Tendo isto em vista, apresentamos a seguir alguns métodos para o cálculo de uma ordem parcial inicial.

7.2.1. Rotulação

Uma maneira de calcular ordens parciais é utilizando a mesma idéia das heurísticas baseadas em rotulação. Sob o ponto de vista de ordens parciais, estas heurísticas funcionam basicamente da seguinte forma: inicializa-se os rótulos dos vértices; escolhe-se um vértice v de melhor rótulo; relaciona-se este vértice com todos os outros ainda não escolhidos (por exemplo, colocar v no início da ordem de eliminação até então obtida equivale a adicionar uv na ordem parcial, para todo u ainda não ordenado); atualiza-se os rótulos; volta-se a escolher um vértice até que todos tenham sido escolhidos (ou "ordenados").

Note que, quando da escolha do vértice, não existirá necessariamente apenas um vértice de melhor rótulo. Para adequar o algoritmo de forma a obter uma ordem parcial, ao escolher v ainda não ordenado, relacionaremos v com todo vértice ainda não ordenado, exceto aqueles cujo valor de rótulo é igual ao de v.

Observe o exemplo da Figura 7.7, onde foi utilizado o método de rotulação Cardinalidade Máxima. Como nos exemplos vistos no capítulo 5, os vértices pretos indicam os vértices já escolhidos e o cinza indica o próximo a ser escolhido. A ordem parcial é indicada pelas setas. Ao final, as setas pontilhadas foram incluídas para manter a transitividade e a aresta destacada foi incluída de forma a tornar a pós-vizinhança do vértice e em uma clique.

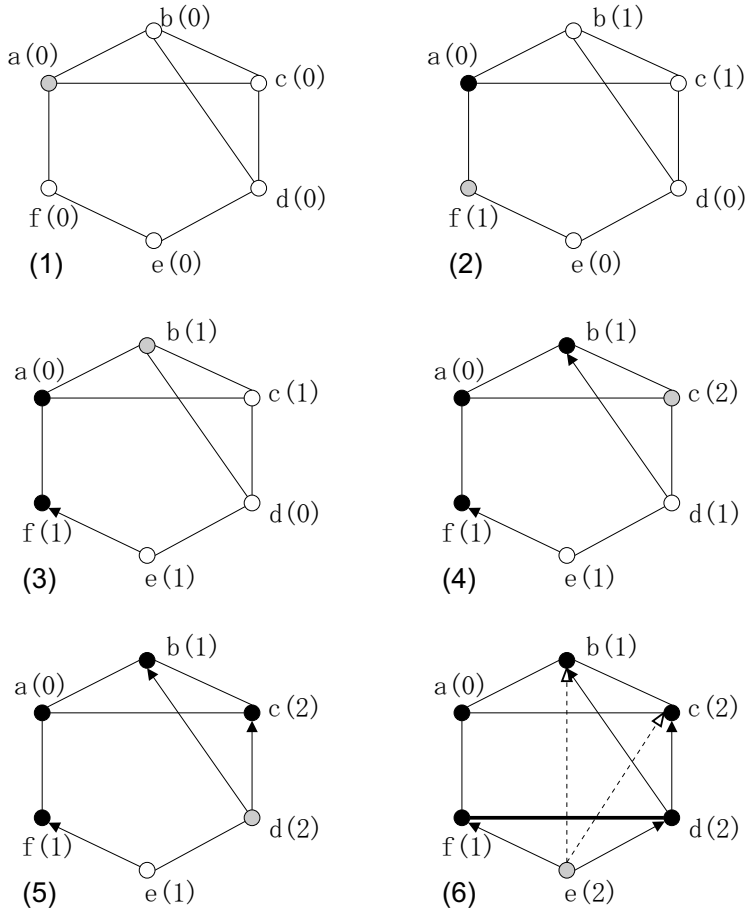

Figura 7.7. *Os números 1 a 6 indicam a ordem em que foram feitas as operações. Sendo P a ordem representada pelas setas em (6), o grafo (6) ignorando-se as setas pontilhadas equivale a G_P^*.*

Desta forma, a obtenção de uma ordem parcial seguirá o Algoritmo 10, que é apenas uma modificação do Algoritmo 4.

O algoritmo também recebe como entrada um método de rotulação L que deve implementar exatamente as mesmas funções e procedimentos indicados na Seção 5.1. Recebe também um grafo $G = (V, E)$ e fornece como saída uma ordem parcial P sobre V. Além disso, $r(v)$ denota o rótulo do vértice v e $Insere(P, uv)$ acrescenta o par ordenado uv à ordem P, além dos pares necessários para manter transitividade (ou seja, $fecho(P.uv)$).

Algoritmo 10. *OrdemParcialRotul*

Entrada: Grafo $G = (V, E)$, Método de rotulação L

Saída: Ordem parcial P sobre V

1: $L.InicializaRotulos(G)$
2: $S \leftarrow V$
3: $P \leftarrow \{(v, v),\ para\ todo\ v \in V\}$
4: **enquanto** $S \neq \emptyset$ **faça**
5: $\quad u \leftarrow L.ProxVertice(S)$
6: $\quad$ **para todo** $v \in S$ tal que $r(u)$ é melhor do que $r(v)$ **faça**
7: $\quad\quad$ **se** $L.Crescente()$ **então**
8: $\quad\quad\quad Insere(P, uv)$
9: $\quad\quad$ **senão**
10: $\quad\quad\quad Insere(P, vu)$
11: $\quad S \leftarrow S \setminus \{u\}$
12: $\quad L.AtualizaRotulos(S, u)$
13: **retorne** P

Precisamos do resultado seguinte na prova do Lema 7.2.2, que mostra a corretude do Algoritmo 10, ou seja, mostra que a relação retornada é realmente uma ordem parcial sobre os vértices do grafo fornecido. Note que o algoritmo tem exatamente n iterações.

Lema 7.2.1. *Seja P a ordem parcial obtida até a i-ésima iteração, $i \leq n$, e S o conjunto dos vértices ainda não escolhidos. Então, um dos dois casos ocorre:*

- *Se $L.Crescente()$ retorna verdadeiro, então não existe $u \in S$ tal que $uv \in P$, para qualquer $v \in V$, $v \neq u$;*
- *Caso contrário, não existe $u \in S$ tal que $vu \in P$, para qualquer $v \in V$, $v \neq u$.*

Prova: Por indução no número de iterações i. Obviamente, para $i = 0$ é válido, pois inicialmente a ordem contém apenas pares reflexivos. Suponha válido para todo $i < k$, onde $k > 0$. Suponha que $L.Crescente()$ retorna verdadeiro. Seja v_k o vértice escolhido nesta iteração. Temos que mostrar que nenhum par do tipo uv tal que $u \in S \setminus \{v_k\}$ e $v \neq u$ é adicionado. Sabemos que todo par adicionado pertence a $fecho(P, v_k v)$, para algum $v \in S \setminus \{v_k\}$ e que

$$fecho(P, v_k v) = \{v_k y : y \in Y\} \cup \{xy : x \in X \text{ } e \text{ } y \in Y\}$$

onde $Y = \{y : vy \in P \text{ } e \text{ } v_k y \in I(P)\}$ e $X = \{x : xv_k \in P \text{ } e \text{ } xv \in I(P)\}$. É fácil ver, então, que basta mostrar que $X \cap (S \setminus \{v_k\}) = \emptyset$. Trivial, já que todo $x \in X$ é tal que $xv_k \in P$ e pela hipótese de indução.

Um raciocínio análogo pode ser feito no caso em que $L.Crescente()$ retorna falso. □

Lema 7.2.2. *A relação binária retornada pelo Algoritmo 10 trata-se de uma ordem parcial sobre os vértices do grafo G dado como entrada.*

Prova: Considere a i-ésima iteração do algoritmo e sejam P a ordem obtida e S o conjunto de vértices ainda não escolhidos até o momento imediatamente antes do início da iteração. Seja v_i o vértice escolhido. Temos que provar que nenhum par adicionado viola anti-simetria ou transitividade (a reflexividade jamais será violada já que todos os pares reflexivos são adicionados antes do loop e nunca são retirados). Na verdade, é suficiente mostrarmos que os pares adicionados não violam anti-simetria, pois a função $Insere$

encarrega-se de adicionar os pares necessários para a que a transitividade não seja violada. Suponha que $L.Crescente()$ retorna verdadeiro. Sabemos que todo par adicionado xy pertence a $fecho(P, v_iv)$, para algum $v \in S \setminus \{v_i\}$. Logo, dado algum $v \in S \setminus \{v_i\}$ tal que a função $Insere(P, v_iv)$ foi chamada, temos que mostrar que se $xy \in fecho(P, v_iv)$ então $yx \notin P$. Escrevemos $fecho(P, v_iv)$:

$$fecho(P, v_iv) = \{v_iy : y \in Y\} \cup \{xy : x \in X \ e \ y \in Y\}$$

onde $Y = \{y : vy \in P \ e \ v_iy \in I(P)\}$ e $X = \{x : xv_i \in P \ e \ xv \in I(P)\}$.

Como $v \in S \setminus \{v_i\}$, pelo lema 7.2.1 vemos que $Y = \{v\}$ (não é vazio, pois certamente $v_iv \in I(P)$, já que $Insere$ foi chamada e pelo lema 7.2.1). Logo, $fecho(P, v_iv) = \{v_iv\} \cup \{xv : xv_i \in P \ e \ xv \in I(P)\}$. Ou seja, todo par adicionado é do tipo xv, para algum $v \in S \setminus v_i$, $x \neq v$. Neste caso, pelo lema 7.2.1, sabemos que $vx \notin P$, para todo $x \neq v$, como queríamos demonstrar.

É possível fazer uma prova análoga, caso $L.IsCrescent()$ retorne falso. □

Uma propriedade interessante de algumas das heurísticas baseadas em rotulação vistas no capítulo anterior é que elas garantem que, caso o grafo dado seja triangularizado, a ordem de eliminação encontrada será perfeita. Desta forma, têm-se ao menos a garantia de que um grafo cordal não será "piorado". Quando passamos a utilizar ordens parciais, não há esta garantia, pois afinal mesmo um grafo cordal pode deixar dúvidas quanto ao próximo vértice a ser posto na ordem. Porém, é fácil ver que existirá pelo menos uma extensão total da ordem obtida equivalente à uma ordem de eliminação perfeita. Basta notar que existirá uma execução do Algoritmo 4 que equivale a uma extensão total da ordem parcial obtida.

7.2.2. Subestruturas

Um espaço de soluções interessante de explorar seria o conjunto de extensões totais de uma ordem parcial obtida levando-se em consideração determinadas estruturas do grafo.

Como uma subestrutura possível, definimos a subestrutura de semi-garra (a figura 7.8 mostra os dois tipos de semi-garras existentes):

Definição 12. *Uma **semi-garra** é um grafo C composto de exatamente quatro vértices ligados entre si por no máximo quatro arestas e contendo um vértice universal, i.e., C é ele mesmo uma garra ou uma garra a menos de uma aresta.*

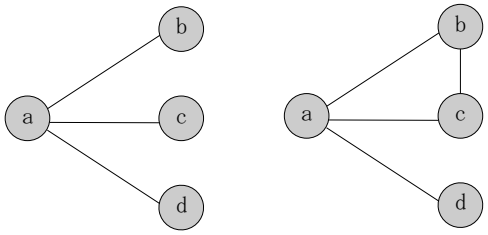

Figura 7.8. *Tipos de semi-garras*

Chamamos o vértice universal de **vértice central** e de **vértices externos**, aos demais vértices. A idéia intuitiva é construir uma ordem parcial sobre V tal que os vértices externos tenham sua vizinhança aumentada em no máximo um vértice devido à eliminação do vértice central, considerando-se apenas os vértices da semi-garra. Ou seja, queremos que após a eliminação do vértice central de acordo com a ordem, qualquer dos vértices externos seja vizinho de no máximo um dos outros vértices externos. Note que isso só não ocorre caso a ordem contenha as relações representadas na Figura 7.9.

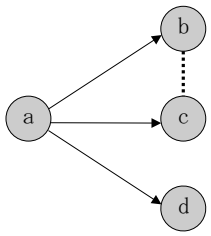

Figura 7.9. *A aresta (b, c) é irrelevante neste caso, pois d tornar-se-ia vizinho de b e c, o que gostaríamos de evitar.*

Obviamente, a ordem formada apenas pelos pares reflexivos não contém as relações da Figura 7.9. Porém, essa ordem é desinteressante uma vez que o intuito é limitar o espaço de busca. Gostaríamos, então, de construir uma ordem que não contenha 7.9 e que relacione o maior número de vértices possível. Nem sempre pode-se construir uma ordem total desta forma, veja por exemplo a Figura 7.10. Na verdade, calcular uma ordem maximal que não contenha as relações de 7.9 pode ser muito custoso e ainda pode nos fornecer ainda uma ordem parcial com poucos pares comparáveis. Uma solução seria permitir que algumas semi-garras sejam orientadas como em 7.9, porém obedecendo ainda alguns critérios (como uma quantidade máxima de vezes que isso seria permitido, ou permitir apenas para as semi-garras cujo vértice central tivesse grau no máximo um determinado valor). Infelizmente, como veremos no Capítulo 8, não pudemos estudar a melhor forma de orientar as semi-garras, dado que a enumeração das extensões totais não nos permitiu concluir qualquer coisa a respeito da ordem parcial inicial.

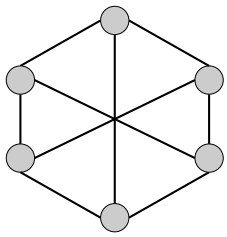

Figura 7.10. *Exemplo de grafo que não possui uma ordem total tal que nenhuma semi-garra esteja orientada como em 7.9.*

Apresentamos a estrutura semi-garra apenas como um exemplo de subestrutura que poderia ser explorada na obtenção de uma ordem parcial inicial. Além desta subestrutura, poderia-se pensar em analisar conjuntos independentes na vizinhança de um vértice ou ainda pensar em como relacionar os vértices de uma clique. Porém, vale ressaltar que a semi-garra é uma estrutura que pode ser facilmente encontrada em um grafo, ao contrário de conjuntos independentes e cliques.

8

Resultados Experimentais e Conclusões

Neste capítulo apresentamos os resultados e a análise dos resultados obtidos para os limites superiores e inferiores. No que diz respeito ao limite superior, foram implementadas todas as heurísticas que utilizam rotulação, com exceção da de preenchimento mínimo, além da meta-heurística *GRASP*. Para o limite inferior, implementou-se o método de geração de ordens totais restringido ao subespaço definido por uma ordem parcial dada como entrada, como visto na Seção 7.2.

Em todas as implementações, utilizou-se a linguagem *Java2SE v1.4.2*. Para as estruturas de grafos e ordens parciais, utilizou-se o pacote *pargoworks.graphs*, desenvolvido pelo grupo **ParGO**, que também utiliza a linguagem citada. Os testes foram executados em um *Pentium IV 1.4GHz* com 1024MB de memória RAM. As instâncias utilizadas para teste constituem-se do *benchmark DIMACS* para o problema de coloração [39, 40]. A escolha deste conjunto de instâncias deve-se ao fato dessas terem sido utilizadas nos estudos computacionais mais conhecidos para o problema.

O intuito principal destes testes é o de analisar a qualidade das soluções fornecidas pelas heurísticas de limite superior e inferior, não importando a eficiência neste caso. Devido a isto, não é fornecido o tempo de execução para nenhuma das heurísticas.

O capítulo está organizado da seguinte maneira: a Seção 8.1 expõe as peculiaridades de cada classe de instâncias testadas; a Seção 8.2 mostra os resultados experimentais

obtidos para o limite superior, enquanto que a Seção 8.3, os resultados obtidos para o limite inferior do espaço de soluções definido por uma ordem parcial encontrada utilizando o método de rotulação MCSM; finalmente, a Seção 8.4 apresenta as conclusões extraídas dos testes.

8.1. Classes de Instâncias

Nesta seção, descrevemos as classes das instâncias utilizadas em nossos testes.

Grafos de Registros Dado o código de um program seqüencial, definimos um grafo relacionado onde: os vértices representam as variáveis e dois vértices são adjacentes se as duas variáveis correspondentes utilizam o mesmo registro em um segmento de código simultaneamente. As instâncias `fpsol2.i.`x, `inithx.i.`x, `mulsol.i.`x e `zeroin.i.`x são grafos de registros. Condon [22] desenvolveu um programa que pega fragmentos de códigos e gera o grafo correspondente.

Grafos Rainha Um grafo rainha `queen`$q.r$ é um grafo com qr vértices, cada um correspondendo a um quadrado de um tabuleiro com q linhas e r colunas. Dois vértices são adjacentes se os quadrados correspondentes vértices estão em uma mesma linha, coluna ou diagonal.

Grafos de Mycielski Dado um grafo $G = (V, E)$ qualquer, a transformação de Mycielski [1] é o seguinte processo de obtenção do grafo G^{my}:

1. Adicione o vértice u^*;
2. Para cada $u \in V$, adicione os vértices u_1 e u_2 e a aresta (u_2, u^*);
3. Para cada $(u, v) \in E$, adicione as arestas (u_1, v_1), (u_1, v_2) e (u_2, v_1)

Sabemos que esta transformação não afeta o tamanho da maior clique do grafo e incrementa o número cromático em uma unidade [49]. O grafo de Mycielski x, denotado por `myciel`x, pode ser obtido aplicando a transformação de Mycielski

sobre o grafo de Mycielski $x-1$, onde a base para a construção é o K_2. Esta classe de grafos parece ser difícil de resolver pois possui muitos ciclos sem cordas (a clique máxima é de cardinalidade 2).

Grafos de Livro Dado um trabalho de literatura, um grafo é criado onde cada vértice representa um personagem e dois vértices são adjacentes se os personagens correspondentes se encontram no texto. Knuth [43] criou os grafos para alguns títulos clássicos da literatura: *Anna Karenina* de Tolstoy (`anna`), *David Copperfield* de Dickens (`david`), *A Ilíada* de Homero (`homer`), *Huckleberry Finn* de Twain (`huck`) e *Les Misérables* de Hugo (`jean`).

Grafos de Campeonato Um grafo dos jogos de um campeonato é um grafo onde os vértices representam os participantes e dois deles são adjacentes se os participantes correspondentes foram adversários em algum jogo. Knuth [43] fornece o grafo do campeonato de futebol americano entre universidades americanas de 1990, denotado por `games120`.

Grafos de Milhas Dado um conjunto de cidades, um grafo de x milhas destas cidades é um grafo com um vértice para cada cidade, onde dois são adjacentes se as cidades correspondentes estão a uma distância de no máximo x milhas uma da outra. `miles`x são grafos de milhas baseados em um conjunto de cidades americanas com distância de no máximo x milhas, dadas por milhas de estradas em 1947. Estes grafos também foram fornecidos por Knuth [43].

Grafos Aleatórios $\mathcal{G}_{n,p}$ são grafos com n vértices onde há uma probabilidade p de existir uma aresta entre quaisquer par de vértices, independente da existência ou não de outra aresta. `DSJC`$n.p$ são grafos aleatórios usados em um trabalho de Johnson, Aragon, McGeoch e Schevon [41].

8.2. Limites Superiores

Nesta seção apresentamos os resultados obtidos para o limite superior. Como já foi dito anteriormente, as heurísticas implementadas incluem as de rotulação, com exceção da de preenchimento mínimo, e a heurística $GRASP$. Além dos resultados obtidos pelas heurísticas implementadas, apresentamos também os resultados fornecidos pela heurística de cortes mínimos. Esses últimos podem ser encontrados em [21].

Nas tabelas a seguir, representamos cada heurística por uma sigla. As siglas e o que elas representam são:

- **LP**: Busca lexicográfica;
- **LM**: Busca lexicográfica utilizando o método dos caminhos de menores rótulos para a atualização dos rótulos;
- **MP**: Cardinalidade máxima;
- **MM**: Cardinalidade máxima utilizando o método dos caminhos de menores rótulos para a atualização dos rótulos;
- **MD**: Grau mínimo;
- **GR**: heurística $GRASP$;
- **MS**: Cortes de vértices mínimos.

As Tabelas 8.1 a 8.4 contêm o tamanho de cada instância, os limites superiores encontrados por cada uma das heurísticas, além do melhor limite inferior conhecido. Estes limites inferiores podem ser encontrados utilizando o método MMD visto em 6.2 juntamente com a técnica apresentada em 6.3 para melhorar um limite inferior qualquer. Ressaltamos que para as heurísticas de rotulação, com exceção da de grau mínimo, foram executadas $|V|$ iterações, cada uma iniciando a construção da ordem por um vértice diferente (lembre-se que inicialmente todos os rótulos são iguais). Para a heurística de grau

mínimo, foram executadas d iterações, onde d é igual à quantidade de vértices de G cujo grau é igual a $\delta(G)$. O resultado fornecido para cada uma das heurísticas de rotulação é o melhor dentre todos os resultados encontrados.

A Tabela 8.1 contém as instâncias cujas larguras são conhecidas (devido a isto suprimimos a coluna contendo o melhor limite inferior).

O restante das instâncias estão divididas nas Tabelas 8.2 a 8.4 de acordo com a razão r^* dada por:

$$r^* = \frac{LS^* - LI^*}{LS^*},$$

onde LS^* representa o menor limite superior e LI^* o maior limite inferior conhecidos. Note que quanto maior o valor r^*, maior a distância relativa entre o melhor limite superior e o melhor limite inferior conhecidos para a instância em questão. O valor desta distância é dividido pelo melhor limite superior conhecido para parametrizar este valor e tornar mais fácil a comparação entre as diferentes heurísticas.

As Tabelas 8.2, 8.3 e 8.4 contêm as instâncias cujo valor de r^* é maior do que 0 e menor ou igual a 0.1, maior do que 0.1 e menor ou igual a 0.5 e maior do que 0.5, respectivamente.

O parâmetro r definido a seguir é usado para comparar as heurísticas na Tabela 8.5 e reflete a distância relativa entre o limite superior encontrado pela heurística e o melhor limite inferior conhecido.

$$r = \frac{LS - LI^*}{LS^*},$$

onde LS^* representa o menor limite superior e LI^* o maior limite inferior conhecidos e LS o limite superior encontrado pela heurística em questão.

A Tabela 8.5 fornece, para cada heurística, a média dos valores de r para cada partição das instâncias, assim como também a média considerando todas as intâncias. Nesta tabela incluímos também as instâncias cuja largura é conhecida, pois nenhuma heurística consegue chegar ao ótimo de todas estas instâncias. Além disso, para a heurística MM fornecemos dois valores, o primeiro considerando todas as instâncias e o segundo deixando de fora as instâncias do tipo `inithx.i` e `mulsol.i` (com exceção da `mulsol.i.1`). Fizemos

isso pelo fato de que os valores gerais para esta heurística são bastante influenciados pelos resultados obtidos para as instâncias mencionadas. Logo, seria interessante também perceber o quanto o resultado é influenciado.

Finalmente, a Tabela 8.6 nos diz a percentagem de instâncias em que cada heurística obteve o melhor valor de limite superior conhecido.

Instância	$\|V\|$	$\|E\|$	**LP**	**LM**	**MP**	**MM**	**MD**	**GR**	**MS**
`fpsol2.i.1`	269	11654	66	66	66	66	66	66	66
`fpsol2.i.2`	363	8691	41	36	41	52	31	44	31
`fpsol2.i.3`	363	8688	41	36	41	52	31	44	31
`inithx.i.1`	519	18707	56	56	56	223	56	56	56
`jean`	80	508	9	9	9	9	9	9	9
`huck`	74	602	10	10	10	10	10	10	10
`miles500`	128	1170	25	22	26	22	29	22	28
`miles1500`	128	5198	80	78	80	77	83	77	83
`mulsol.i.1`	138	3925	50	50	66	50	66	50	50
`mulsol.i.2`	173	3885	38	32	38	69	32	38	32
`mulsol.i.3`	174	3916	38	33	38	69	32	38	32
`mulsol.i.4`	175	3946	38	33	38	69	32	38	32
`mulsol.i.5`	176	3973	38	31	38	69	31	38	31
`zeroin.i.1`	126	4100	50	50	50	50	50	50	50

Tabela 8.1. *Resultados das Heurísticas de Limite Superior- Instâncias com largura em árvore conhecida*

Instância	$\|V\|$	$\|E\|$	LI	LP	LM	MP	MM	MD	GR	MS
anna	138	986	11	13	12	14	12	12	12	12
DSJC125.9	125	6961	108	120	119	121	119	119	119	120
DSJC250.9	250	27897	218	245	243	245	243	244	246	244
miles1000	128	3216	48	55	49	55	49	54	51	53
zeroin.i.2	157	3541	31	42	37	42	40	33	43	33
zeroin.i.3	157	3540	31	42	37	42	40	33	43	33

Tabela 8.2. *Resultados das Heurísticas de Limite Superior - Instâncias cuja razão r^* é menor do que* 0.1

8.3. Limites Inferiores

Implementou-se a modificação sobre o algoritmo de Corrêa e Szwarcfiter [23] proposta nesta dissertação. Como fora dito anteriormente, gerar todas as ordens parciais é algo muito custoso computacionalmente. Por isso, optamos por restringir o espaço de busca para um subconjunto destas ordens totais, gerando apenas as extensões totais de uma ordem parcial dada como entrada.

Propomos na Seção 7.2 alguns métodos para calcular tal ordem parcial. Dentre eles, geramos as extensões totais da ordem parcial obtida com o método de rotulação MCSM. Devido aos resultados obtidos, que serão discutidos no próximo capítulo, optou-se por não testar o espaço definido pelas extensões totais das ordens parciais obtidas com os demais métodos. O principal motivo para isso é que os testes não nos permitiu tirar conclusões a respeito do espaço definido pela ordem parcial dada propriamente dito. Logo, as conclusões seriam as mesmas independentemente da ordem parcial tomada como entrada.

As informações dadas nas Tabelas 8.7, 8.8 e 8.9 dizem respeito à execução do algoritmo de geração de extensões totais recebendo como entrada uma ordem parcial obtida com o método de rotulação MCSM. Ressaltamos que houve estouro de memória para a maioria das instâncias testadas, com exceção das marcadas com $*$ e das instâncias cujo

Instância	$\|V\|$	$\|E\|$	**LI**	**LP**	**LM**	**MP**	**MM**	**MD**	**GR**	**MS**
david	87	812	10	14	13	14	13	13	13	13
DSJC125.5	125	6961	62	113	110	114	110	110	111	110
DSJC250.5	250	15668	125	237	233	238	232	233	240	233
homer	561	3258	21	38	37	37	36	33	34	31
inithx.i.2	558	13979	31	42	37	43	227	35	47	35
inithx.i.3	559	13969	31	42	37	43	227	35	48	35
miles250	125	387	8	10	10	10	10	9	10	9
miles750	128	2113	33	41	37	41	36	40	38	38
myciel3	11	20	4	5	5	7	5	5	5	5
myciel4	23	71	6	11	10	12	10	11	10	11
myciel5	47	236	12	21	20	23	20	20	20	20
myciel6	95	755	20	37	40	41	39	39	40	35
queen5_5	25	160	12	18	18	18	18	18	18	18
queen6_6	36	290	15	29	27	28	25	26	26	28
queen7_7	49	476	20	38	36	37	36	36	36	38

Tabela 8.3. *Resultados das Heurísticas de Limite Superior - Instâncias cuja razão r^* é maior do que* 0.1 *e menor ou igual a* 0.5

Instância	$\|V\|$	$\|E\|$	LI	LP	LM	MP	MM	MD	GR	MS
DSJC125.1	125	736	16	74	71	74	71	67	70	67
DSJC250.1	250	3218	32	195	186	194	184	179	201	179
games120	120	638	12	38	36	40	35	47	35	51
myciel7	191	2360	34	70	76	72	76	73	87	74
queen8_8	64	728	23	50	48	53	48	49	47	49
queen9_9	81	1056	26	66	60	66	60	62	60	66
queen10_10	100	1470	31	81	76	86	76	77	75	79
queen11_11	121	1980	34	97	92	102	92	98	94	101
queen12_12	144	2596	37	117	111	121	110	116	114	120
queen13_13	169	3328	42	131	130	134	129	135	157	145
queen14_14	196	4186	46	165	149	179	148	161	183	164
queen15_15	225	5180	46	192	172	190	170	181	210	192
queen16_16	256	6320	53	211	198	228	198	216	241	214

Tabela 8.4. *Resultados das Heurísticas de Limite Superior - Instâncias cuja razão r^* é maior do que* 0.5

	LP	LM	MP	MM		MD	GR	MS
Geral	0.38	0.32	0.39	0.72	0.4	0.32	0.38	0.33
$r^* = 0$	0.12	0.05	0.12	0.67	0.19	0.03	0.12	0.03
$r^* \leq 0.1$	0.19	0.1	0.2	0.13		0.09	0.17	0.09
$r^* \leq 0.5$	0.39	0.34	0.44	1.05	0.36	0.33	0.38	0.33
$r^* > 0.5$	0.73	0.68	0.77	0.67		0.66	0.76	0.76

Tabela 8.5. *Média dos valores de r obtidos para cada heurística.*

	LP	**LM**	**MP**	**MM**	**MD**	**GR**	**MS**
Geral	15%	46%	15%	58%	54%	42%	58%
$r^* = 0$	43%	57%	43%	43%	79%	57%	86%
$r^* <= 0.1$	0	67%	0	67%	67%	33%	67%
$r^* <= 0.5$	2%	47%	7%	67%	60%	40%	73%
$r^* > 0.5$	8%	23%	0	62%	15%	31%	15%

Tabela 8.6. *Percentagem de instâncias em que obteve-se o melhor valor de limite superior.*

limite inferior inicial igualou o limite superior inicial, obtido aplicando-se a heurística de rotulação MCSM para limite superior. As instâncias em que o limite inferior inicial iguala o limite superior inicial são aquelas em que o campo $|Sel|$ é igual a zero. Note que para isso ocorrer, a ordem parcial inicial não precisa necessariamente tratar-se de uma ordem total. As tabelas contêm os seguintes parâmetros:

- LI_0: limite inferior da ordem parcial dada como entrada;
- LI: limite inferior mínimo de um nó qualquer marcado para visita ao final da execução;
- $|Sel|$: quantidade de nós visitados;
- $|L|$: quantidade de nós marcados para serem visitados cujo valor de limite inferior é mínimo.

Atentamos para o fato de que, para as instâncias `queen8_8` e `queen9_9` os valores de limite inferior inicial são maiores ou igual ao melhor limite superior obtido por MCSM apresentado na seção anterior. Isto se deve ao fato de que aquele valor é encontrado executando $|V|$ vezes a heurística de rotulação, enquanto que a ordem parcial tomada é obtida executando a heurística apenas um vez.

Inst	LI_0	LI	$\|Sel\|$	$\|L\|$
anna*	10	12	2276	0
DSJC125.9	119	119	0	0
DSJC250.9	243	243	0	0
miles1000	49	49	0	0
zeroin.i.2	38	38	121	1281
zeroin.i.3	38	38	124	1278

Tabela 8.7. *Resultados da geração de extensões - Instâncias cuja razão r^* é menor do que* 0.1

Inst	LI_0	LI	$\|Sel\|$	$\|L\|$
david	13	13	0	0
DSJC125.5	106	106	102	2699
inithx.i.2	227	227	0	0
miles250	10	10	0	0
miles750	36	36	0	0
myciel4*	7	10	160027	0
myciel5	14	14	3362	1406
queen5_5	17	17	17155	22018
queen6_6	24	24	5175	5319

Tabela 8.8. *Resultados da geração de extensões - Instâncias cuja razão r^* é maior do que* 0.1 *e menor ou igual a* 0.5

Inst	LI_0	LI	$\|Sel\|$	$\|L\|$
DSJC125.1	38	38	1201	345
DSJC250.1	164	164	209	178
games120	22	22	1249	800
myciel7	55	55	203	217
queen8_8	48	48	2306	14284
queen9_9	58	58	1986	1699
queen10_10	77	77	962	7152
queen11_11	90	90	695	4986

Tabela 8.9. *Resultados da geração de extensões - Instâncias cuja razão r^* é maior do que* 0.5.

8.4. Análise dos Resultados

Nesta seção analisamos os resultados vistos anteriormente. A Seção 8.4.1 trata dos limites superiores e a 8.4.2, dos limites inferiores.

8.4.1. Limites Superiores

Primeiramente, notamos que tanto LP como MP são menos competitivas quando comparadas às outras heurísticas. Apenas uma vez a heurística LP é a única a fornecer o melhor limite superior (instância myciel7), enquanto que MP sempre é acompanhada por outras quando encontra o melhor limite superior. No que diz respeito ao desempenho geral (Tabela 8.5), estas duas heurísticas também são superadas pelas restantes, com exceção da MM cujo desempenho geral é bastante influenciado pelos péssimos resultados obtidos para as instâncias inithx.i e mulsol.i.

Analisando os resultados das demais heurísticas, vemos que todas elas têm, por assim dizer, seus "altos e baixos". Por exemplo, apesar da média de r para a heurística MD ser sempre menor ou igual à média das outras heurísticas (significando que em média a

distância entre o limite superior encontrado e o maior limite inferior conhecido é menor do que das outras heurísticas), temos que em geral esta não é a heurística que fornece melhores limites superiores. Na verdade, a diferença entre os limites superiores encontrados por cada heurística é, na maioria das vezes, muito pequena. As exceções constituem-se basicamente das intâncias `inithx.i` e `mulsol.i.2` a `mulsol.i.4`, em que a heurística MM obtém péssimos resultados, e das instâncias `queen13_13` a `queen16_16`, em que a MM obtém resultados bem melhores do que as demais.

Optamos por utilizar a heurística MM no estudo do espaço de soluções restrito pois essa forneceu os melhores resultados para o conjunto de instâncias cujo valor do parâmetro r^* é maior do que $0,5$ (instâncias consideradas mais difíceis).

Finalmente, ressaltamos que a heurística $GRASP$ possui vários parâmetros que não foram explorados já que o foco do presente trabalho não era o estudo da aplicação desta heurística ao problema. Ainda assim, vemos que o desempenho de GR é comparável ao desempenho das demais, inclusive para as instâncias `queen8_8` e `queen10_10` supera todas as outras, definindo um novo melhor limite superior. Tém-se então um bom indício de que a aplicação da heurística $GRASP$ é promissora.

8.4.2. Limites Inferiores

Como falamos anteriormente, os resultados obtidos restringindo o espaço de soluções não são suficientes para podermos inferir sobre a ordem cujas extensões totais estão sendo geradas. Isso porque apenas duas instâncias, `anna` e `myciel4`, tiveram este espaço completamente explorado. As outras instâncias ou são tais que o limite inferior inicial iguala o limite superior encontrado inicialmente (e por isso, não é necessário fazer uma busca, dado que a solução inicial trata-se de um ótimo local), ou são tais que a geração estourou a memória disponível. Neste caso, apenas estas últimas merecem nossa atenção.

Observando as instâncias em que houve estouro de memória, vemos que todas elas são tais que os valores LI_0 e LI são iguais, ou seja, ao final da execução ainda possuimos nós cujo valor de limite inferior é igual ao valor inicial. Além disso, vemos que a quantidade de

nós nesta situação é geralmente muito grande, algumas vezes até ultrapassa em muito a quantidade de nós visitados. Ou seja, mesmo que não tivesse ocorrido estouro de memória, a menos que se encontrasse rapidamente uma solução cujo valor igualasse LI_0 de forma que todos estes nós fossem podados, a quantidade de nós a serem percorridos seria ainda muito grande. Isso ocorre mesmo em instâncias cujo limite superior inicial é bastante próximo do limite inferior inicial (como por exemplo, `zeroin.i.2` e `zeroin.i.3`, cujos valores de limite inferior e superior iniciais eram 38 e 40, respectivamente). Além disso, ressaltamos que nenhuma execução chegou a gerar uma folha, ou seja, chegou a uma solução diferente da inicial.

Com tudo isso e sabendo também da existência de instâncias cuja diferença entre o melhor limite inferior e melhor limite superior conhecidos é muito grande, concluímos que é necessário encontrar um método para calcular um limite inferior mais eficaz para cada nó, de forma a intensificar as podas na árvore.

9

Conclusões e Trabalhos Futuros

Como já foi dito anteriormente, o problema de encontrar uma decomposição em árvore ótima para um grafo qualquer é $\mathcal{NP}$-difícil. Vários estudos têm sido feitos na última década no intuito de melhorar os limites superiores [10, 44], de melhorar os limites inferiores [16, 21, 50], de propor métodos enumerativos [14] e aproximativos [3, 19, 13]. Porém, nenhum bom resultado prático foi até agora alcançado [45, 56], pois além de haver uma carência de um bom parâmetro de comparação, ocorre também que o único método enumerativo conhecido na literatura se mostra ineficiente mesmo para o problema com k fixo em valores pequenos (por exemplo, $k = 4$) [56].

As principais contribuições desta dissertação são: 1) uma representação para as soluções do problema que utiliza ordens parciais (esta mesma representação é também utilizada para definir espaços de soluções); 2) um método de enumeração do espaço de soluções através da adaptação do algoritmo proposto por Corrêa e Szwarcfiter [23] para enumeração de todas as extensões de uma ordem parcial; e 3) uma forma de utilizar a enumeração para calcular um limite inferior e para explorar sub-espaços de solução.

O método enumerativo proposto pode ser utilizado em conjunto com o método "branch and bound". Foi testada a eficiência deste método para enumerar um subespaço de soluções definido por uma dada ordem parcial. Além disso, contribuímos para o estado atual do estudo dos limites superiores para o problema propondo a aplicação da heurística

GRASP. Apesar de não termos explorado completamente esta heurística, damos fortes indícios de que essa pode ser um bom investimento para o cálculo de um limite superior.

Note, a princípio, que todas as instâncias testadas são de tamanho médio ou pequeno. Mesmo assim, existe um grande número delas cuja distância relativa entre o melhor limite inferior e o melhor limite superior conhecidos é alta (instâncias cujo valor de r^* é maior do que $0,5$). Isso, somado aos resultados obtidos com a enumeração, nos leva a concluir que é de fundamental importância um método para calcular um limite inferior do problema mais eficaz do que os já conhecidos. Caso tivéssemos um método que fornecesse um melhor limite inferior, além de possibilitar uma avaliação mais conclusiva das diferentes heurísticas para limite superior, também poderia viabilizar a enumeração proposta como um método de resolução exato para o problema.

Sabendo da necessidade de um melhor limite inferior, pode-se pensar como trabalho futuro em elaborar uma formulação inteira para o problema. A própria idéia de ordens parciais poderia ser utilizada neste sentido, permitindo que fosse aproveitado o método de "branch" do algoritmo de geração de extensões de Corrêa e Szwarcfiter. A vantagem de trabalhar com uma formulação inteira seria que, mesmo que o limite inferior fornecido pela relaxação fosse a priori muito ruim, tornando o espaço de soluções a ser percorrido muito grande, uma solução para melhorar este limite poderia ser adicionar planos de cortes.

Por outro lado, dados um grafo $G = (V, E)$ e uma ordem parcial $P \subseteq V^2$, note que o grafo G_P pode ser triangularizado. Desta forma, o algoritmo proposto não precisaria enumerar toda a subárvore enraizada em P, já que a solução do problema é, na verdade, a triangularização e não a ordem total que leva àquela. Logo, uma forma de diminuir o número de nós percorridos seria testar a cada "branch" se G_P é triangularizado. Porém, fazer isso pode ser muito custoso. Algumas questões que podem ser levantadas são, então: 1) como reconhecer que G_P é triangularizado de uma forma simples (de preferência, levando em consideração apenas a ordem parcial); 2) qual o conjunto de ordens parciais definido por uma determinada triangularização; e 3) como particionar as ordens parciais

de acordo com as triangularizações que estas definem e, com isso, enumerar apenas um elemento de cada conjunto da partição.

Podemos citar também como possível trabalho futuro um estudo mais detalhado da heurística GRASP. Quanto a isto, tanto pode-se pensar em explorar melhor os vários parâmetros existentes na heurística, quanto numa diferente abordagem na representação das soluções, passando-se a trabalhar também com ordens parciais ao invés de esquemas de eliminação.

Referências Bibliográficas

[1] Sur le coloriage des graphs. *Colloquium Mathematicum*, 3:161–162, 1955.

[2] K.R. Abrahamson and M.R. Fellows. Finite automata, bounded treewidth and well-quasiordering. In *Graph Structure Theory, Contemporary Mathematics*, volume 147, pages 539–564. AMS, 1993.

[3] E. Amir. Efficient approximation for triangulation of minimum treewidth. In *Proceedings UAI'01*, pages 7–15, 2001.

[4] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Math*, 7:277–284, 1986.

[5] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

[6] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7(2):305–314, 1986.

[7] S. Arnborg, A. Proskurowski, and D. Seese. Monadic second order logic, tree automata and forbidden minors. In *Proc. 4th Worksh. Computer Science Logic*, pages 1–16. Springer-Verlag, 1990.

[8] E. Balas and J. Xue. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Algorithmica*, 15:397–412, 1996.

[9] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal clique trees. *Artif. Intell.*, 125(1-2):3–17, 2001.

[10] A. Berry, J. Blair, P. Heggernes, and B. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39:287–298, 2004.

[11] H. L. Bodlaender. Improved self-reduction algorithms for graphs with bounded treewidth. In *WG '89: Proceedings of the fifteenth international workshop on Graph-theoretic concepts in computer science*, pages 232–244, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[12] H.L. Bodlaender. Necessary edges in k-chordalization of graphs. Technical report, Institute of Information and Computing Science, Utrecht University, 2000.

[13] H.L. Bodlaender, J.R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995.

[14] H.L. Bodlaender and T. Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21:358–402, 1996.

[15] H.L. Bodlaender and A. Koster. Safe separators for treewidth. Technical report, Institute of Information and Computing Science, Utrecht University, 2003.

[16] H.L. Bodlaender, A. Koster, and T. Wolle. Contraction and treewidth lower bounds. Technical report, Institute of Information and Computing Science, Utrecht University, 2004.

[17] R. B. Borie, R. G. Parker, and C. A. Tovey. Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families. *Algorithmica*, 7(5&6):555–581, 1992.

[18] C. F. Bornstein. *Parallelizing and De-parallelizing Elimination Orders*. PhD thesis, Carnegie Mellon University, 1998.

[19] V. Bouchitté, D. Kratsch, H. Müller, and I. Todinca. On treewidth approximations. *Discrete Appl. Math.*, 136(2-3):183–196, 2004.

[20] G. Brightwell and P. Winkler. Counting linear extensions is #p-complete. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 175–181, New York, NY, USA, 1991. ACM Press.

[21] F. Clautiaux, J. Carlier, A. Moukrim, and S. Négre. New lower and upper bounds for graph treewidth. In J.D.P. Rolim, editor, *Proceedings International Workshop on Experimental and Efficient Algorithms, WEA 2003*, volume 2647, pages 70–80. Springer Lecture Notes in Computer Science, 2003.

[22] A. Condon. *DIMACS Challenge*, 1994.

[23] R. C. Corrêa and J. L. Szwarcfiter. On extensions, linear extensions, upsets and downsets of ordered sets. *Discrete Mathematics*, 295(1-3):13–30, 2005.

[24] B. Courcelle. Graph rewriting: An algebraic and logic approach. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 193–242. 1990.

[25] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[26] B. Courcelle. The monadic second-order logic of graphs iii: Tree-decompositions, minors and complexity issues. *Inform. Thor. et Applications*, 26:257–286, 1992.

[27] B. Courcelle. Graph grammars, monadic second-order logic and the theory of graph minors. In N. Robertson et P. Seymour, editor, *Contemporary Mathematics (AMS)*, volume 147, pages 565–590. 1993.

[28] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1-2):49–82, 1993.

[29] R. Diestel. *Graph Theory - Graduate Texts in Mathematics, Volume 173*. Springer-Verlag, Heidelberg, 2000,1997.

[30] R. Diestel, T. R. Jensen, K. Y. Gorbunov, and C. Thomassen. Highly connected sets and the excluded grid theorem. *J. Comb. Theory Ser. B*, 75(1):61–73, 1999.

[31] G.A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg.*, 25:71–76, 1961.

[32] J.A. Ellis, I.H. Sudborough, and J.S. Turner. The vertex separation and search number of a graph. *Inf. Comput.*, 113(1):50–79, 1994.

[33] M. R. Fellows and M. A. Langston. On search, decision, and the efficiency of polynomial-time algorithms. *J. Comput. Syst. Sci.*, 49(3):769–779, 1994.

[34] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[35] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[36] D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965.

[37] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory B*, 16:47–56, 1974.

[38] F. Glover. Tabu search and adaptive memory programing - advances, applications and challenges. *Interfaces in Computer Science and Operations Research*, pages 1–75, 1996.

[39] D. Johnson, A. Mehrotra, and M. Trick. Computational series: Graph coloring and its generalizations. 2004. http://mat.gsia.cmu.edu/COLORING02/.

[40] D. Johnson and M. Trick. Cliques, coloring and satisfiability: Second dimacs implementation challenge. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26, 1996. http://mat.gsia.cmu.edu/COLOR/instances.html.

[41] D. S. Johnson, C. R. Aragon, L .A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation. part i, graph partitioning. *Oper. Res.*, 37(6):865–892, 1989.

[42] T. Kloks. Treewidth. computations and approximations. *Lecture Notes in Computer Science*, 842, 1994.

[43] D. E. Knuth. *The Stanford GraphBase: a platform for combinatorial computing.* ACM Press, New York, NY, USA, 1993.

[44] Koster. *Frequency Assignment - Models and Algorithms.* PhD thesis, Maastricht University, 1999.

[45] A. Koster, H.L. Bodlaender, and S.P.M. van Hoesel. Treewidth: Computational experiments. *Eletronic Notes in Discrete Mathematics*, 8, 2001. Revised version in preparation.

[46] J. Lagergren. Efficient parallel algorithms for tree-decomposition and related problems. In *Proceedings of the* 31*rd Annual Symposium on Foundations of Computer Science*, pages 173–182, 1990.

[47] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *J. Algorithms*, 20(1):20–44, 1996.

[48] Jens Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proceedings of the 18th international colloquium on Automata, languages*

and programming, pages 532–543, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[49] M. Larsen, J. Propp, and D. Ullman. the fractional chromatic number of a graph and a construction of mycielski. *Journal of Graph Theory*, 19(3):411–416, 1995.

[50] B. Lucena. A new lower bound for tree-width using maximum cardinality search. *SIAM J. Discret. Math.*, 16(3):345–353, 2003.

[51] C. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982. Second edition by Dover, 1998.

[52] P.M. Pardalos and J. Xue. The maximum clique problem. *J. of Global Optimization*, 4:301–328, 1994.

[53] S. Ramachandramurthi. The structure and number of obstructions to treewidth. *SIAM J. Discret. Math.*, 10(1):146–157, 1997.

[54] B.A. Reed. Finding approximate separators and computing treewidth quickly. *STOC*, 24:221–228, 1992.

[55] N. Robertson and P.D. Seymour. Graph minors xiii: the disjoint path problem. *J. Combinatorics Theory*, series B 63:65–110, 1995.

[56] H. Röhrig. Tree decomposition: a feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1998.

[57] D.J. Rose, R.E. Tarjan, and G.S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:266–283, 1976.

[58] Frank Ruskey. Generating linear extensions of posets by transpositions. *J. Comb. Theory Ser. B*, 54(1):77–101, 1992.

[59] G. Szekeres and H.S. Wilf. An inequality for the chromatic number of a graph. *Journal of Combinatorial Theory*, 4:1–3, 1968.

[60] A. Takahashi, S. Ueno, and Y. Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Math.*, 127(1-3):293–304, 1994.

[61] R.E Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 13:566–579, 1984.

A

Generalização do Teorema 5.1.1

Neste apêndice adicionamos um resultado obtido logo após a defesa da presente dissertação. Este resultado diz respeito ao grafo de eliminação de uma dada ordem, o que pode afetar o limite inferior obtido para tal ordem, uma vez que pode aumentar a pós-vizinhança máxima de um vértice.

Pedimos ao leitor que reveja a Definição 7 e o Teorema 5.1.1. Sendo G um grafo qualquer e π um esquema de eliminação dos vértices de G, note que existe uma relação entre a definição e o teorema citados quando aplicados a π. Isto ocorre pois, dados dois vértices $u,\ v$ tais que $(u, v) \in E(G_\pi) \setminus E(G)$, pelo teorema existe um caminho p entre eles tal que $\pi(x) < min\{\pi(u), \pi(v)\}$, para todo $x \in p$, e como os vértices de p também são relacionados entre si em π, de acordo com a definição 7 serão adicionadas arestas entre eles até que eventualmente a aresta (u, v) também seja adicionada.

Agora, observe o exemplo da ordem de eliminação P da Figura A.1. Devido ao Teorema 5.1.1 e à existência do caminho $\langle x, y, z\rangle$ em G entre u e v, sabemos que toda extensão total π de P é tal que $(u, v) \in G_\pi$. Então, poderíamos ter adicionado tal aresta logo quando do cálculo do grafo G_P. Além disso, se $uv \in P$ ou $vu \in P$, isto poderia até mesmo aumentar o tamanho da pós-vizinhança máxima de um vértice em P, o que melhoraria nosso limite inferior. Dado este fato, uma simples modificação seria considerar a existência de caminhos $\langle u, x_1, ..., x_q, v\rangle$ tais que $\{x_i u, x_i v\} \subseteq P$, para todo $i \in \{1, ..., q\}$.

Porém, é possível obter um resultado que nos possibilite considerar uma quantidade maior de caminhos, como vemos no Teorema A.0.1. Antes, porém, precisamos do seguinte lema.

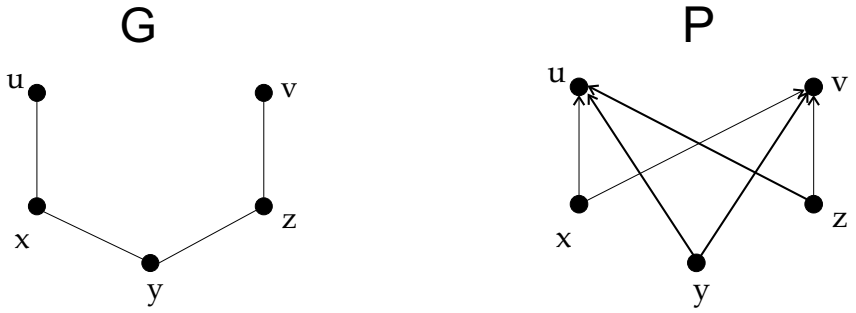

Figura A.1.

Lema A.0.1. *Seja $P \subseteq C^2$ uma ordem parcial. Temos que $P \cup fecho(P, uv)$ é uma ordem parcial, para todo par ordenado $uv \in C^2$.*

Prova: Primeiramente, reescrevemos abaixo a Definição 6 de fecho transitivo:

$$fecho(P, uv) = \begin{cases} \{xy : x \in X, y \in Y\}, & se\ uv \in I(P) \\ \emptyset, & c.c. \end{cases}$$

onde $X = \{x : xu \in P\ e\ xv \in I(P)\}$ e $Y = \{y : vy \in P\ e\ uy \in I(P)\}$.

Sejam $u, v \in C$ quaisquer. Temos que mostrar que $P' = P \cup fecho(P, uv)$ é reflexiva, anti-simétrica e transitiva. Como P é uma ordem parcial sobre os elementos de C, obviamente ao adicionarmos $fecho(P, uv)$ mantemos a reflexividade. Logo, basta mostrar:

1. P' **é anti-simétrica**: Temos que mostrar que não existem $x, y \in C$ tais que $\{xy, yx\} \subset P'$. Como P é uma ordem parcial, basta mostrar que se $xy \in fecho(P, uv)$ então $yx \notin P'$, ou seja, se $xy \in fecho(P, uv)$, então $yx \notin P$ e $yx \notin fecho(P, uv)$. Pela definição de fecho transitivo, se $xy \in fecho(P, uv)$, então $x \in X$ e $y \in Y$. Conseqüentemente, $xv \in I(P)$ e $vy \in P$. Logo, $yx \notin P$, pois caso contrário teríamos que $vx \in P$, absurdo. Por outro lado, se $xv \in I(P)$, temos que $x \notin Y$. Logo, o par yx não pode estar em $fecho(P, uv)$.

2. P' **é transitiva**: Suponha o contrário. Logo, existem elementos $x, y, z \in C$ tais que $xy, yz \in P'$ e $xz \notin P'$. Como P é transitiva, temos que $xy \in fecho(P, uv)$ ou $yz \in fecho(P, uv)$. Observe que se $y \in Y$, então $y \notin X$, e vice-versa. Logo, o ou é exclusivo. Abaixo, analisamos os dois casos:

   - $xy \in fecho(P, uv)$ e $yz \notin fecho(P, uv)$: Temos que $\{yz, xu, vy\} \subseteq P$ e $\{xv, uy\} \in I(P)$. Por transitivade, temos que $vz \in P$. Além disso, $zu \notin P$, pois caso contrário yu estaria em P, e $uz \notin P$, pois caso contrário xz estaria em $P \subseteq P'$. Temos, então, que $uz \in I(P)$ e $vz \in P$, logo $z \in Y$, absurdo, pois neste caso teríamos que $xz \in fecho(P, uv) \subseteq P'$.
   - $yz \in fecho(P, uv)$ e $xy \notin fecho(P, uv)$: Temos que $\{xy, yu, vz\} \subseteq P$ e $\{yv, uz\} \subseteq I(P)$. Por transitividade, $xu \in P$. Além disso, $xv \notin P$, pois caso contrário $xz \in P \subseteq P'$, e $vx \notin P$, pois caso contrário $vy \in P$. Temos, então, que $xv \in I(P)$ e $xu \in P$, logo, $x \in X$, absurdo pois neste caso tem-se que $xz \in fecho(P, uv) \subseteq P'$.

□

Finalmente, o teorema a seguir generaliza o Teorema 5.1.1 permitindo a aplicação para ordens parciais

Teorema A.0.1. *Seja G um grafo qualquer e seja $P \subseteq V^2$ uma ordem parcial. Dado um par $u, v \in V$, temos que $(u, v) \in E(G_\pi)$, para todo $\pi \in Ext_t(P)$, se e somente se $(u, v) \in E(G)$ ou existem caminhos p_1, p_2 entre u e v em G tais que $wu \in P$, para todo $w \in p_1$, e $wv \in P$, para todo $w \in p_2$.*

Prova: ($\Rightarrow$) Iremos mostrar que se $(u, v) \notin E(G)$ e não existem tais caminhos p_1 e p_2, então existe $\pi \in Ext_t(P)$ tal que $(u, v) \notin E(G_\pi)$. Sejam $p_1, ..., p_q$ todos os caminhos entre u e v em G. Iremos tomar um vértice de cada caminho de forma a construir um conjunto I tal que ou

$$wv \notin P, \text{ para todo } w \in I, \text{ ou} \tag{A.1}$$

$$wu \notin P, \text{ para todo } w \in I \tag{A.2}$$

Note que se isso não for possível, então existem caminhos p_1 e p_2 tais que $wu \in P$, para todo $w \in p_1$, e $wv \in P$, para todo $w \in p_2$, absurdo. Construa P^* da seguinte forma:

1. Faça P^* inicialmente igual a P
2. Se I é tal que A.1 então $v^* \leftarrow u$
3. Caso contrário (I é tal que A.2), $v^* \leftarrow v$
4. Para cada $w \in I$ faça $P^* \leftarrow fecho(P^*, v^*w)$.

Pelo lema A.0.1, temos que P^* é uma ordem parcial, logo trata-se de uma extensão de P e, conseqüentemente, $Ext_t(P^*) \subseteq Ext_t(P)$. Além disso, temos que para qualquer caminho p em G entre u e v, existe $w \in p$ tal que $uw \in P^*$ ou $vw \in P^*$. Desta forma, para toda extensão total π de P^*, não existe p entre u e v tal que $\pi(w) < min\{\pi(u), \pi(v)\}$, para todo $w \in p$. Logo, pelo Teorema 5.1.1, temos que $(u, v) \notin E(G_\pi)$, para todo $\pi \in Ext_t(P^*) \subseteq Ext_t(P)$.

($\Leftarrow$) Se $(u, v) \in E(G)$, o resultado segue diretamente. Suponha então que $(u, v) \notin E(G)$. Sejam p_1 e p_2 caminhos entre u e v em P tais que $wu \in P$, para todo $w \in p_1$, e $wv \in P$, para todo $w \in p_2$. É fácil ver que se $p_1 = p_2$, então toda extensão total π de P é tal que $\pi(w) < min\{\pi(u), \pi(v)\}$, para todo $w \in p_1$. Logo, pelo Teorema 5.1.1, temos que $(u, v) \in E(G_\pi)$, para todo $\pi \in Ext_t(P)$. Suponha, então, $p_1 \neq p_2$. Podemos supor que existe $w \in p_1$ tal que $wv \notin P$, pois caso contrário recairíamos no primeiro caso com $p_1 = p_1$. Dada uma extensão $\pi \in Ext_t(P)$ qualquer, temos que $wv \in \pi$ ou $vw \in \pi$, para qualquer $w \in p_1$ tal que $wv \notin P$. Se $wv \in \pi$, para todo $w \in p_1$, novamente pelo Teorema 5.1.1 temos que $(u, v) \in E(G_\pi)$. Suponha que $vw^* \in \pi$, para algum $w^* \in p_1$. Logo, por transitividade, $vu \in \pi$ e, conqüentemente, $w'u \in \pi$, para todo $w' \in p_2$. Daí temos que $wu \in \pi$ e $wv \in \pi$, para todo $w \in p_2$, e pelo Teorema 5.1.1 $(u, v) \in E(G_\pi)$. □

Dado um grafo $G = (V, E)$ qualquer, graças ao Teorema A.0.1 podemos redefinir o

grafo de eliminação relacionado com a ordem parcial $P \subseteq V^2$.

Definição 13. *O* ***grafo de eliminação segundo a ordem*** P *é o grafo* $G_P = (V, E \cup E_P)$*, onde* E_P *contém todo par* $(u, v) \in V^2$ *tal que existem caminhos* p_1 *e* p_2 *entre* u *e* v *em* G *tais que* $wu \in P$*, para todo* $w \in p_1$*, e* $wv \in P$*, para todo* $w \in p_2$*.*