PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

DISSERTAÇÃO DE MESTRADO

UÉVERTON DOS SANTOS SOUZA

Uma Abordagem Parametrizada para Grafos E/Ou e Grafos X-*de*-Y

Orientador: Prof[a]. D.Sc. Lilian Markenzon
Co-orientador: Prof[a]. D.Sc. Maise Dantas da Silva

Rio de Janeiro
2010

Instituto de Matemática

NCE UFRJ NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

S725 Souza, Uéverton dos Santos

Uma Abordagem Parametrizada para Grafos E/Ou e Grafos X-*de*-Y / Uéverton dos Santos Souza. – 2010.

84 f.: il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica, 2010.

Orientador: Lilian Markenzon

Co-orientador: Maise Dantas da Silva.

1. Grafos E/Ou - Teses. 2. Grafos X-de-Y - Teses. 3. Complexidade Parametrizada - Teses. I. Lílian Markenzon (Orient.). II. Maise Dantas da Silva (Co-orient.) III. Universidade Federal do Rio de Janeiro. Instituto de Matemática, Núcleo de Computação Eletrônica. IV. Título.

CDD

Uma Abordagem Parametrizada para Grafos E/Ou e Grafos X-*de*-Y

Uéverton dos Santos Souza

Dissertação de Mestrado submetida ao Corpo Docente do Departamento de Ciência da Computação do Instituto de Matemática, e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários para obtenção do título de Mestre em Informática.

Aprovado em: 26 de Outubro de 2010

Profª. D.Sc. Lilian Markenzon (Orientador)

Profª. D.Sc. Maise Dantas da Silva (Co-orientador)

Prof. D.Sc. Fábio Protti

Prof. D.Sc. Mitre Costa Dourado

Prof. D.Sc. André Luiz Pires Guedes

Dedico esse trabalho a minha mãe e a todos que acreditaram em mim.

AGRADECIMENTOS

A Deus, que sempre iluminou a minha caminhada.

A minha família, por sempre estar do meu lado apoiando as minhas decisões.

Ao prof. Fábio Protti, pela orientação, amizade e por ter acreditado em mim desde o início.

Às minhas orientadoras, Lilian Markenzon e Maise Dantas da Silva, pelo carinho, atenção e confiança.

Aos meus amigos, pela força, motivação e apoio que sempre me deram.

RESUMO

Este trabalho apresenta um estudo sobre grafos E/Ou e grafos X-*de*-Y, suas características e exemplos de aplicações. Grafos E/Ou são digrafos acíclicos, tais que os vértices possuem rotulações E ou Ou, que determinam se a dependência desses vértices com seus vizinhos de saída será conjuntiva ou disjuntiva. Grafos X-*de*-Y, por sua vez, são uma generalização de grafos E/Ou, que definimos com motivação nos modelos para detecção de *deadlocks* de sistemas distribuídos. São abordados também os problemas MIN–E/Ou e MIN–X-DE-Y, problemas que se baseiam em encontrar um subgrafo-solução de custo mínimo. São estudados seus casos polinomiais e de NP-completude. Por fim são apresentadas versões parametrizadas destes problemas: MIN–E/Ou(k); MIN–E/Ou(k, r); MIN–X-DE-Y(k).

Palavras-chave: Grafos E/Ou, Grafos X-*de*-Y, Complexidade Parametrizada.

ABSTRACT

This work presents a study on *And/Or* graphs and *X-of-Y* graphs, their characteristics and examples of applications. *And/Or* graphs are acyclic digraphs, where vertices have labels *And* or *OR*, which determine whether the dependence of these vertices with respect to their out-neighbours is conjunctive or disjunctive. *X-of-Y* graphs are a generalization of *And/Or* graphs, whose definition is based on models for deadlock detection in distributed systems. Also, we will dealt with the problems MIN-AND/OR and MIN-X-OF-Y, which are based on finding a minimum cost solution-subgraph. Polynomial and NP-Completeness cases of these problems are studied. Finally, parametrized versions of such problems are considered: MIN-AND/OR(k); MIN-AND/OR(k, r); MIN-X-OF-Y(k).

keywords: *And/Or* Graphs, *X-of-Y* Graphs, Parameterized Complexity.

SUMÁRIO

LISTA DE FIGURAS 10

LISTA DE ALGORITMOS 11

LISTA DE ABREVIATURAS E SIGLAS 12

1 INTRODUÇÃO 13
1.1 Definições básicas e notações 15
1.2 Complexidade de algoritmos 17

2 COMPLEXIDADE PARAMETRIZADA 19
2.1 A classe FPT - Problemas tratáveis por parâmetro fixo 21
2.1.1 O método de redução a um núcleo do problema 21
2.2 Intratabilidade Parametrizada 22
2.3 A classe $W[1]$ 24
2.4 Transformação Paramétrica 25

3 GRAFOS E/OU E SUAS APLICAÇÕES 26
3.1 Decomposição de Problemas 27
3.1.1 Representação do problema utilizando grafos E/Ou 28
3.2 Geração de Padrões de Cortes Bidimensionais em Placas 29
3.3 Redes de Atividades 33
3.3.1 Representação de Redes de Atividades como grafos E/Ou. 34
3.3.2 Escalonamento de grafos E/Ou 35
3.4 Fórmulas Booleanas 37
3.4.1 Construção de Árvores E/Ou 38
3.5 Hipergrafos direcionados 41
3.5.1 Transformação de F-grafos em grafos E/Ou 42
3.5.2 Transformação de grafos E/Ou em F-grafos 44

4 OS PROBLEMAS MIN–E/OU **E** MIN–X-DE-Y 45
4.1 NP-Completude 47
4.2 Casos Polinomiais para MIN–E/Ou 52
4.3 Limite Inferior 54
4.4 Limite Superior e Aproximação 57
4.5 Grafos *X-de-Y* 64
4.5.1 Algoritmo de tempo polinomial para árvores *X-de-Y* 66

5 MIN–E/OU **E** MIN–X-DE-Y **PARAMETRIZADOS** 69
5.1 O Problema MIN–E/Ou(k, r) 71
5.2 O Problema MIN–X-de-Y(k) 74

6 CONSIDERAÇÕES FINAIS 79
6.1 Trabalhos Futuros 80

REFERÊNCIAS 82

LISTA DE FIGURAS

Figura 3.1: Exemplo de Grafo E/Ou. 26
Figura 3.2: Árvore E/Ou para multiplicação de cadeias de matrizes. 29
Figura 3.3: Ilustrações de padrões de cortes. 30
Figura 3.4: (a) Placa P. (b) Conjunto das m peças. (c) Padrão de corte de P. 32
Figura 3.5: Árvore E/Ou de padrões de corte da Figura 3.4. 32
Figura 3.6: Representação da rede de atividades da Tabela 3.1. 34
Figura 3.7: (a) Um grafo E/Ou G. (b) Grafo G escalonado. 37
Figura 3.8: Árvore E/Ou da fórmula $F = (x_1 + x_2 + x_3)(x_4.x_5)$. 38
Figura 3.9: (a) Representação de conjunção. (b) Representação de disjunção. 39
Figura 3.10: Árvore E/Ou de uma fórmula booleana. 41
Figura 3.11: Exemplo de um hipergrafo direcionado. 42
Figura 3.12: (a) Exemplo de B-arco. (b) Exemplo de F-arco. 42
Figura 3.13: Transformação de um $F\text{-}grafo$ em um grafo E/Ou. 44
Figura 3.14: Transformação do grafo E/Ou em (a) para o F-grafo em (b). . . 44

Figura 4.1: Grafo E/Ou da fómula $F = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_1} + x_2)$. 49
Figura 4.2: Subgrafo-solução para $F = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_1} + x_2)$. 51
Figura 4.3: (a) Grafo G. (b) Subgrafo-solução $H_{opt}(G)$. (c) Subgrafo C de G. 56
Figura 4.4: (a) Árvore E/Ou. (b) Custos dos vértices e solução ótima de (a). 58
Figura 4.5: Grafo E/Ou onde $c(v_5, v_6) = 3$. 58
Figura 4.6: (a) Grafo E/Ou. (b) Solução ótima H. (c) Solução viável G'. . . 60
Figura 4.7: Exemplo de escalonamento de um grafo E/Ou. 61
Figura 4.8: Caso onde $c(v_{10}, v_{11}) = \alpha$. 63
Figura 4.9: Pior caso para a aproximação do Algoritmo 4.3. 63
Figura 4.10: (a) Grafo $X\text{-}de\text{-}Y$. (b) Subgrafo-solução do grafo em (a). . . . 65

Figura 5.1: Transformação paramétrica de G em G'. 77
Figura 5.2: (a) Clique de tamanho 3. (b) Subgrafo-solução de custo 18 em G'. 78

LISTA DE ALGORITMOS

3.1 Escalonamento de grafos E/Ou acíclicos 36

4.1 MIN–E/Ou para árvores 53
4.2 Limite inferior para MIN–E/Ou 55
4.3 Limite superior e aproximação para MIN–E/Ou 59
4.4 MIN–X-de-Y para árvores 67

LISTA DE ABREVIATURAS E SIGLAS

I_v	Conjunto dos vizinhos de entrada de v.
O_v	Conjunto dos vizinhos de saída de v.
Z^+	Conjunto dos números inteiros não negativos.
CNF	Forma normal conjuntiva.
APX	Classe de problemas que admitem algoritmos aproximativos.
FPT	Classe de problemas tratáveis por parâmetro fixo.

1 INTRODUÇÃO

Nesta dissertação, são estudadas inicialmente as estruturas de dados denominadas Grafos E/Ou e Grafos X-*de*-Y, suas características e aplicações, além de alguns problemas relacionados a elas. Em seguida, em razão de sua aplicação em diversas áreas, é dada uma atenção especial aos problemas MIN–E/OU e MIN–X-DE-Y. Finalmente, como foco principal deste trabalho, são apresentadas as versões parametrizadas para esses problemas: MIN–E/OU(k), MIN–E/OU(k, r) e MIN–X-DE-Y(k), sendo estudadas suas classificações na teoria de complexidade parametrizada.

Os estudos desta dissertação partiram do problema MIN–E/OU, descrito a seguir. Dado um digrafo acíclico e ponderado $G = (V, E)$ com um vértice fonte $s \in V$, e tal que cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$ e cada aresta e possui um peso $\tau(e) \in Z^+$, determinar o peso mínimo de um subdigrafo $H = (V', E')$ de G que satisfaça às seguintes propriedades:

1. $s \in V'$;

2. se $v \in V'$ e $f(v) = E$, então todos as arestas de saída de v devem pertencer a E';

3. se $v \in V'$ e $f(v) = Ou$, então exatamente uma das arestas de saída de v deve pertencer a E'.

Motivou-nos o fato de que o problema MIN–E/Ou, apesar de ser um problema natural e possuir várias e interessantes aplicações, não vinha sendo muito estudado na literatura especializada. As referências encontradas eram poucas e, de forma geral, pouco recentes. Sendo assim, procuramos investigar variações do problema, e em particular abordá-lo do ponto de vista da complexidade parametrizada, algo que, tanto quanto sabemos, ainda não havia sido iniciado.

Este trabalho está organizado como segue. No Capítulo 1, são introduzidas algumas definições básicas e notações necessárias para o desenvolvimento do trabalho. Definições específicas serão introduzidas conforme a necessidade nos capítulos correspondentes.

No Capítulo 2, são introduzidos os conceitos de complexidade parametrizada, com foco nas classes FPT e $W[1]$.

O Capítulo 3 traz uma caracterização de grafos E/Ou, além de exemplificar algumas de suas aplicações, mostrando que estes grafos podem ser utilizados nas mais diversas áreas. Além disso, são apresentadas uma modelagem de redes de atividades e de F-grafos como grafos E/Ou.

O Capítulo 4 apresenta o problema MIN–E/Ou, sua prova de NP-dificuldade para grafos em geral, e um algoritmo polinomial para o caso em que a entrada é restrita a árvores. Também serão apresentados um algoritmo que retorna um limite inferior e um algoritmo aproximativo para o problema MIN–E/Ou. Por fim, será definida uma generalização desses grafos, os grafos X-*de*-Y, e, de forma análoga a grafos E/Ou, é formulado o problema MIN–X-de-Y, e apresentado um algoritmo

polinomial para o custo mínimo de árvores X-*de*-Y.

No Capítulo 5, são definidos os problemas MIN–E/OU(k), MIN–E/OU(k, r) e MIN–X-DE-Y(k). Em seguida, através do método de redução a um núcleo, é provado que o problema MIN–E/OU(k, r) pertence à classe FPT. Através de uma transformação paramétrica do problema parametrizado CLIQUE(k), é provado também que o problema MIN–X-DE-Y(k) é W[1]-difícil. Por fim, é apresentado um conjunto de heurísticas que podem ser aplicadas ao problema MIN–E/OU(k).

Finalmente, o Capítulo 6 apresenta uma conclusão do que foi apresentado neste trabalho, as considerações finais e uma relação dos trabalhos futuros.

Nesta dissertação foram obtidos alguns resultados, dentre eles apresentamos um algoritmo de tempo polinomial para o problema MIN–X-DE-Y quando o grafo é uma árvore, provamos que o problema MIN–E/OU(k, r) pertence a classe FPT, e que o problema MIN–X-DE-Y(k) é W[1]-difícil.

Em paralelo às pesquisas e estudos para desenvolvimento desta dissertação, também foi desenvolvido durante o mestrado um trabalho sobre roteamento de ônibus [21], que ilustra um exemplo prático do uso de grafos na busca de soluções para problemas reais.

1.1 Definições básicas e notações

Um algoritmo é um conjunto de instruções bem definidas e ordenadas que descrevem uma sequência de passos destinada à solução de um problema. Muitas vezes, para auxiliar a descrição desses passos, os dados são estruturados de forma homogênea, possibilitando sua modelagem matemática através de uma estrutura de

dados. Dentre os diversos tipos de estruturas existentes, uma das mais abrangentes e capaz de modelar diversos problemas é o *digrafo*.

Um *digrafo* G é uma estrutura composta por um par $(V(G), E(G))$, onde $V(G)$ é um conjunto finito de $n = |V(G)|$ elementos chamados *vértices* e $E(G)$ é um conjunto de $m = |E(G)|$ *arestas*, tal que cada aresta e é um par ordenado de vértices de G.

Um *passeio* é uma sequência de vértices de um digrafo G tal que, entre cada par de vértices consecutivos na sequência, existe uma aresta. Caso esta sequência se inicie e termine no mesmo vértice, este passeio denomina-se *passeio fechado*. Um passeio fechado onde todos os vértices são distintos, com exceção dos extremos, é denominado *ciclo*. Se todos os vértices do passeio forem distintos, este passeio é chamado de *caminho*.

Um digrafo G é um *digrafo simples* quando possui no máximo uma aresta entre quaisquer dois vértices (isto é, sem *arestas paralelas*) e não possui arestas cujas terminações estão no mesmo vértice (isto é, sem *laços*). G é *conexo* se existe um caminho entre cada par de vértices distintos $v, w \in V(G)$. Quando o digrafo G é um digrafo simples, conexo e acíclico (sem ciclos), dizemos que G é uma *árvore*. Se, para todo par de vértices $v, w \in V(G)$, existir uma aresta, dizemos que G é um *grafo completo*.

Em um digrafo, suas arestas e/ou vértices podem ser ponderados por *pesos* (custos) de acordo com o problema que modelam. Neste trabalho, os pesos dos vértices v e das arestas e serão representados, respectivamente, por $\tau(v)$ e $\tau(e)$.

Cada aresta $e = (u, v) \in E(G)$, possuirá uma origem u e um destino v. Para a origem u, e é denotada *aresta de saída* e v seu *vizinho de saída*. Analogamente, para o destino v, e é denotada *aresta de entrada* e u seu *vizinho de entrada*. Neste

trabalho, O_v e I_v denotam, respectivamente, o conjunto dos vizinhos de saída e de entrada de v, sendo $|O_v|$ o *grau de saída* de v e $|I_v|$ o seu *grau de entrada*.

Em um digrafo, um vértice que possui somente arestas de saída é denominado *fonte*, enquanto um vértice que possui somente arestas de entrada é denominado *sumidouro*. Um digrafo que não possui ciclos dirigidos, ou seja, ciclos que seguem a orientação das arestas, é denominado *digrafo acíclico*. Se o digrafo não possuir mais de uma aresta direcionada com a mesma origem e mesmo destino, então este digrafo é um *digrafo simples*.

Em um digrafo acíclico G, dados dois vértices $v_j, v_k \in V(G)$, escrevemos $v_j \prec v_k$ se existir um caminho dirigido de v_j a v_k em G. Uma *ordenação topológica inversa* de um digrafo acíclico G é uma disposição dos vértices de G em uma sequência $v_1, v_2, ..., v_n$ tal que, sempre que $v_j \prec v_k$, tem-se $j > k$. Tal ordenação é sempre possível em um digrafo acíclico.

Dados dois grafos H e G, dizemos que H é um *subgrafo* de G se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$. Se H é um subgrafo de G e H for um grafo completo, então dizemos que H é uma *clique* de G.

Neste trabalho, são abordados apenas digrafos simples. Assim, ao longo do texto, denominamos estes apenas por grafos e as arestas dirigidas simplesmente por arestas. Logo, dados dois vértices u e v, a aresta (u, v) é diferente da aresta (v, u).

1.2 Complexidade de algoritmos

Seja um problema $\Pi = (D, Q)$, onde D é um conjunto de dados e Q é a questão do problema. Os dados específicos que constituem uma entrada para Π formam uma

instância do problema. Dizemos que Π é um *problema de decisão* se seu objetivo for responder *sim* ou *não* à questão Q. Se for possível desenvolver um algoritmo com tempo de execução polinomial no tamanho da entrada para resolver este problema, então Π pertence à classe denominada P. Se Π admitir um algoritmo que verifique uma solução para o sim em tempo polinomial no tamanho da entrada, dizemos que Π pertence à classe NP. É importante observarmos que o fato $\Pi \in NP$ não implica que uma resposta para Π possa ser encontrada em tempo polinomial. Assim, temos que $P \subseteq NP$, mas não se sabe se $NP \subseteq P$.

Sejam $\Pi_1 = (D_1, Q_1)$ e $\Pi_2 = (D_2, Q_2)$ dois problemas de decisão. Uma *transformação polinomial* de Π_1 para Π_2 é uma função $f : D_1 \to D_2$, computada em tempo polinomial no tamanho de D_1, tal que qualquer instância $I \in D_1$ produz resposta sim para Π_1 se e somente se $f(I)$ também produzir resposta sim para Π_2.

Um problema Π é *NP-difícil* se, para todo problema de decisão $\Pi' \in NP$, existir uma transformação polinomial de Π' para Π. Se, além disso, $\Pi \in NP$, dizemos que Π é *NP-completo*.

2 COMPLEXIDADE PARAMETRIZADA

A teoria de complexidade parametrizada introduzida por Downey e Fellows [8, 9], incorpora a problemas NP-completos parâmetros adicionais na sua formulação. Esses parâmetros se devem ao fato de que, em muitos casos, somente uma pequena faixa de valores para a solução dos problemas é realmente importante na prática. Uma vez fixados os valores desses parâmetros, busca-se desenvolver novos algoritmos que transferem a exponencialidade do problema para esse conjunto K de parâmetros adicionais, tornando assim o problema tratável.

Segundo essa teoria, dado um conjunto K de parâmetros, dizemos que um problema Π é *tratável por parâmetro fixo* (FPT) para este conjunto de parâmetros K se, uma vez adicionado o conjunto K à definição do problema, for possível desenvolver um algoritmo para solucioná-lo em tempo $f(K).n^{\alpha}$, onde α é uma constante independente de n e K.

Na teoria de complexidade parametrizada, estamos interessados em problemas especificados por três itens:

1. A parte principal da entrada do problema.
2. Alguns aspectos da definição do problema, que constituem o(s) parâmetro(s).
3. A questão a ser respondida.

Para exemplificar problemas desse tipo, são apresentados a seguir as versões parametrizadas dos problemas NP-completos Cobertura de vértices e Clique [5]:

Clique(k)

Parâmetro: Um inteiro positivo k.

Instância: Um grafo $G = (V, E)$.

Questão: G possui uma clique de tamanho k?

Cobertura de Vértices(k)

Parâmetro: Um inteiro positivo k.

Instância: Um grafo $G = (V, E)$.

Questão: G possui um conjunto de vértices $V' \subseteq V(G)$, tal que $|V'| \leq k$ e, para toda aresta (u, v) de G, $u \in V'$ e/ou $v \in V'$?

Uma vez fixado um parâmetro para um problema NP-completo, este pode ser classificado quanto à sua tratabilidade (FPT) ou nível de intratabilidade ($W-hierarquia$). Alguns problemas possuem como parte de sua definição mais de um parâmetro, podendo receber diferentes classificações de acordo com o parâmetro que passa a ser fixo.

2.1 A classe FPT - Problemas tratáveis por parâmetro fixo

Definição 1 [8] *Uma linguagem parametrizada é um conjunto* $L \subseteq \sum^* \text{x} \sum^*$, *onde* $\sum$ *é um alfabeto parametrizado.*

Se L é uma linguagem parametrizada e $(x, k) \in L$, então x é referente à parte principal, e k é referente ao parâmetro.

Definição 2 [8] *Uma linguagem parametrizada* L *é tratável por parâmetro fixo se for possível determinar em tempo* $f(k).n^\alpha$ *se* $(x, k) \in L$, *onde* $n = |x|$, α *é uma constante independente de* n *e de* k, *e* f *é uma função arbitrária (possivelmente exponencial).*

Em outras palavras, um problema parametrizado é tratável por parâmetro fixo se admite um algoritmo que o resolva em tempo $f(k).n^\alpha$. A clase de problemas tratáveis por parâmetro fixo é denotada por FPT (*Fixed Parameter Tractable*).

Podemos notar que a definição de FPT permanece inalterada se substituirmos $f(k).n^\alpha$ por $f(k) + n^\alpha$.

Um exemplo de problema NP-completo que pertence à classe FPT é o problema Cobertura por Vértices(k).

2.1.1 O método de redução a um núcleo do problema

Para provar que um problema Π é FPT, devemos exibir um algoritmo de tempo $f(k).n^\alpha$ que solucione Π. Muitas vezes, a construção desse algoritmo não ocorre de forma imediata, sendo necessário o uso de técnicas que auxiliem a sua construção.

Dentre as técnicas conhecidas para classificação de um problema como FPT, uma das mais utilizadas é o método de *redução a um núcleo do problema.*

A idéia principal deste método é reduzir, em tempo polinomial, uma instância I do problema Π a uma instância I', tal que o tamanho de I' seja limitado por alguma função do parâmetro k independente da cardinalidade de I. Dessa forma, a instância I' constitui um núcleo do problema com cardinalidade constante. Logo, I' pode ser exaustivamente analisada, e sua solução pode ser apresentada como uma solução para I, caso exista.

Teorema 3 [8] *Um problema parametrizado pertence a classe FPT se e somente se for possível reduzi-lo a um núcleo.*

2.2 Intratabilidade Parametrizada

Além da classe FPT, Downey e Fellows definiram classes apropriadas de problemas parametrizados de acordo com seu nível de intratabilidade. Essas classes são organizadas em uma hierarquia (*W-hierarquia*) e são baseadas intuitivamente na complexidade dos circuitos necessários para se verificar a validade de uma solução, ou, alternativamente, na profundidade lógica natural do problema.

Definição 4 [9] *Um circuito booleano é de tipo misto se ele consistir de circuitos que possuem portas lógicas dos seguintes tipos:*

1. *Portas lógicas estreitas: portas* não, e *e* ou, *com tamanho da entrada limitado. Usualmente, assume-se que o tamanho limite da entrada é 2 para portas* e *e* ou, *e 1 para portas* não.
2. *Portas lógicas largas: portas* E *e* OU, *com tamanho da entrada irrestrito.*

Definição 5 [9] *A profundidade de um circuito C é definida pelo número máximo de portas lógicas (estreitas ou largas), não considerando as portas lógicas* não*, em um caminho da entrada à saída de C. O entrelaçamento de um circuito C é o número máximo de portas lógicas largas em um caminho da entrada à saída de C.*

Dizemos que uma família de circuitos F possui profundidade limitada se existir uma constante h tal que todo circuito pertencente à família F possui profundidade máxima h. Dizemos que F possui entrelaçamento limitado se existir uma constante t tal que todo circuito em F possui entrelaçamento máximo t. F é uma família de circuitos de decisão se cada circuito possui uma única saída. Dizemos que um circuito de decisão C aceita um vetor de entrada x se a saída do circuito possui valor 1 para x. O peso do vetor booleano x é o número de valores 1 no vetor.

Definição 6 [9] *Seja F uma família de circuitos de decisão. Sabe-se que F pode conter muitos circuitos diferentes com um dado número de entradas. É associado a F o problema de circuito parametrizado $L_F = \{(C, k) : \text{Um circuito } C \text{ aceita um vetor de entrada de peso } k, \text{ onde } k \text{ é uma constante}\}$.*

Definição 7 [9] *Um problema parametrizado L pertence a classe $W[t]$ se existir uma redução uniforme de L ao problema de circuito parametrizado L_F, para alguma família F, formada por circuitos de decisão do tipo misto de entrelaçamento máximo t e profundidade máxima h, para alguma constante h.*

Assim, temos a relação de continência

$$FPT \subseteq W[1] \subseteq W[2] \subseteq ...$$

e existe a conjectura de que cada uma dessas relações é própria. A união das classes da hierarquia W é denotada por WH (W-hierarquia). Se $P = NP$, temos a seguinte implicação:

Lema 8 [18] *Se $P = NP$, então $WH \subseteq FPT$.*

2.3 A classe $W[1]$

A classe $W[1]$ é atualmente a classe mais estudada de problemas parametrizados que acredita-se serem intratáveis, pois diversos problemas já foram provados W[1]-completos, e o seguinte problema genérico é W[1]-completo:

Problema: Short Turing Machine Acceptance(k)

Instância: Uma máquina de Turing não determinística M e uma string x.

Parâmetro: Um inteiro positivo k.

Questão: M tem um caminho computacional que aceita x em no máximo k passos?

Este problema é o análogo parametrizado do problema Turing Machine Acceptance (onde o limite no número de passos é polinomial no tamanho da entrada $|x|$, em vez de k), que é o problema NP-completo básico genérico na teoria de complexidade clássica. Short Turing Machine Acceptance(k) pode ser trivialmente resolvido em tempo $O(n^{k+1})$, onde n denota o tamanho total da entrada. Isso é feito explorando-se exaustivamente todos os caminhos computacionais de k passos. Acredita-se que este resultado não possa ser significativamente melhorado.

Enunciamos também o seguinte resultado:

Lema 9 [9] Clique(k) *é W[1]-completo.*

2.4 Transformação Paramétrica

Assim como é feito com a complexidade clássica, podemos encontrar evidências para a intratabilidade parametrizada estudando as noções apropriadas de transformações de problemas.

Definição 10 [8] *Uma transformação paramétrica (ou redução uniforme) de uma linguagem parametrizada L em uma linguagem parametrizada L' é um algoritmo que transforma uma entrada (x, k) em uma entrada (x', k') tal que:*

1. *$(x, k) \in L$ se e somente se $(x', k') \in L'$.*
2. *$k' = g(k)$ é uma função somente em k.*
3. *A computação da transformação é executada em tempo $f(k).n^{\alpha}$, onde $n = |x|$, α é uma constante independente de n e de k, e f é uma função arbitrária.*

A propriedade essencial das transformações paramétricas é que, se L pode ser transformada em L' e $L' \in FPT$, então $L \in FPT$. Caso L seja W[t]-completa, então L' é W[t]-difícil, sendo completa para a classe $W[t]$ se $L' \in W[t]$.

3 GRAFOS E/OU E SUAS APLICAÇÕES

Um grafo E/Ou é um digrafo G, tal que todo vértice $v \in V(G)$ possui um rótulo $f(v) \in \{E, Ou\}$. Neste grafo, as arestas representam relações de dependência entre os vértices: vértices do tipo E dependem estritamente de todos os seus vizinhos de saída (dependência conjuntiva), enquanto vértices do tipo Ou dependem apenas de um dos seus vizinhos de saída (dependência optativa).

Na representação de grafos E/Ou, os vértices do tipo E são ilustrados com um arco entre suas arestas de saída. A Figura 3.1 ilustra um grafo E/Ou, onde s é um vértice do tipo E e t um vértice do tipo Ou. Na literatura, também são encontradas definições de grafos E/Ou onde os rótulos $\{E, Ou\}$ se encontram nas arestas [11].

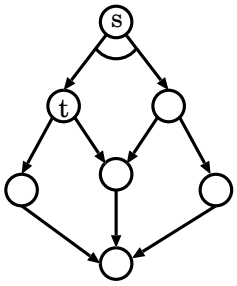

Figura 3.1: Exemplo de Grafo E/Ou.

Grafos E/Ou são utilizados em vários ramos da ciência. Neste capítulo, são descritas algumas de suas aplicações.

3.1 Decomposição de Problemas

Decomposição de Problemas (*problem reduction*) é uma estratégia utilizada para soluções de problemas complexos, que consiste em decompor problemas em subproblemas com complexidade menor que a do original. A idéia desta estratégia é seguir esta decomposição recursivamente, até alcançar um conjunto de problemas de solução imediata, de forma que a solução desses problemas gere uma solução para o problema original.

Em Inteligência Artificial, grafos E/Ou são amplamente utilizados para representação de todas as possíveis decomposições de um problema, ou seja, para representar o espaço de busca de decomposição de um problema [15]. Nesta estratégia, uma solução para o problema original pode ser encontrada a partir de buscas no espaço das possíveis decomposições para o problema. Esse espaço de busca é geralmente representado graficamente por um grafo E/Ou. Neste grafo, os vértices contêm a descrição dos problemas (subproblemas) e as arestas indicam como os problemas são decompostos. O vértice fonte s representa o problema original, os sumidouros subproblemas triviais, e os vértices intermediários subproblemas não triviais. Os vértices do tipo Ou indicam que os vizinhos de saída são subproblemas optativos, enquanto os vértices do tipo E indicam que os vizinhos de saída são subproblemas conjuntivos.

A seguir, será apresentado o problema de multiplicação de cadeias de matrizes e a representação do seu espaço de busca como grafo E/Ou.

Problema: Multiplicação de Cadeias de Matrizes

Instância: Uma cadeia de n matrizes (M_1, M_2, ... , M_n), onde $n \geq 2$ e M_i possui r_i linhas e r_{i+1} colunas, $1 \leq i \leq n$.

Questão: Determinar a ordem em que M_1, ... , M_n devem ser multiplicadas, produzindo a matriz $M_{1,n}$, de modo a minimizar o número de operações, considerando que o produto de uma matriz p x q por outra matriz q x r requer $O(pqr)$ operações.

Exemplo: Seja $M_{1,4} = M_1[10, 20]$ x $M_2[20, 50]$ x $M_3[50, 1]$ x $M_4[1, 100]$.

A obtenção de $M_{1,4}$ pela multiplicação M_1 x (M_2 x (M_3 x M_4)) requer 125.000 operações. Já a parentização (M_1 x (M_2 x M_3)) x M_4, que é ótima, requer apenas 2.200 operações.

3.1.1 Representação do problema utilizando grafos E/Ou

Considere $M_{i,j} = (M_i$ x ... x $M_j)$. O problema de multiplicação de cadeias de matrizes pode ser modelado como uma árvore E/Ou da seguinte forma:

$M_{i,j}$ será um nó Ou com $(j - i)$ filhos, onde cada filho k $(i \leq k \leq j - 1)$ será um nó E representando a multiplicação ($M_{i,k}$ x $M_{k+1,j}$). Cada um destes nós E possui dois filhos, representando, respectivamente, $M_{i,k}$ e $M_{k+1,j}$, onde $M_{l,l}$ equivale a M_l. A construção da árvore segue recursivamente até que cada folha represente exatamente uma matriz da instância original do problema ou a multiplicação de duas dessas matrizes. A Figura 3.2 ilustra a representação das possíveis soluções para a matriz $M_{1,4}$, exemplificada acima.

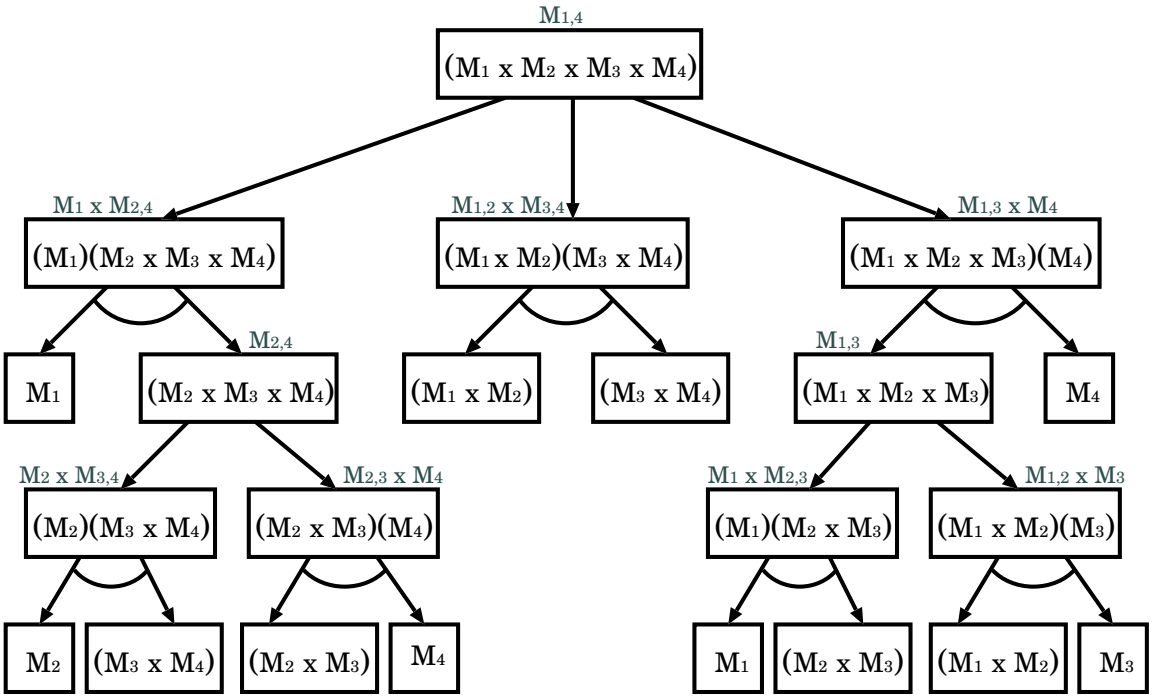

Figura 3.2: Árvore E/Ou para multiplicação de cadeias de matrizes.

3.2 Geração de Padrões de Cortes Bidimensionais em Placas

Na área de Pesquisa Operacional, grafos E/Ou são utilizados para auxiliar na tomada de decisões e na análise de sistemas complexos do mundo real, tipicamente com o objetivo de melhorar ou otimizar a sua performance. Uma das aplicações de grafos E/Ou em Pesquisa Operacional é a sua utilização na geração de padrões de cortes bidimensionais em placas.

Este problema, que é NP-Difícil [14], aparece em diversos processos industriais, tais como: corte de bobinas de papel e alumínio, barras de aço, chapas metálicas e de madeira, placas de circuito impresso, caixas de papelão, rolos de tecido, entre outros. A seguir, uma descrição do problema de geração de padrões de cortes guilhotinados em placas é apresentada.

O modo como as peças são arranjadas na placa é chamado *padrão de corte*.

A Figura 3.3 ilustra dois exemplos de padrões de corte.

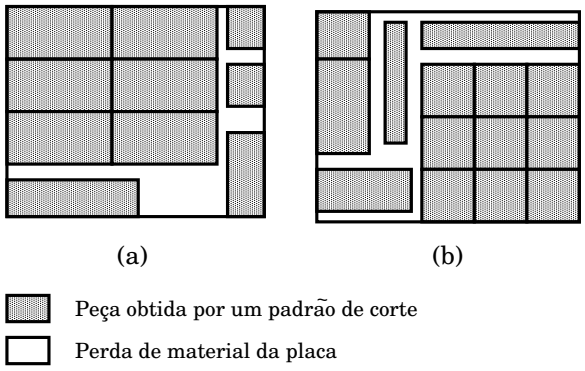

Figura 3.3: Ilustrações de padrões de cortes.

Problema: Geração de Padrões de Cortes Bidimensionais em Placas

Instância: Uma placa retangular P de dimensões (L, W), onde L é o comprimento e W a largura, um conjunto de m peças retangulares p_i de dimensões (l_i, w_i), tais que $l_i \leq L$ e $w_i \leq W$, $1 \leq i \leq m$, e um valor utilidade v_i associado a cada peça p_i.

Questão: Encontrar um padrão de corte da placa P que produza um conjunto de peças que maximize o valor de utilidade total, ou seja, maximize o somatório dos valores utilidade v_i das peças obtidas pelo corte. (Se o valor utilidade v_i for a área da peça p_i, então o objetivo é equivalente a minimizar a perda de material da peça P.)

Dada uma placa P, uma árvore E/Ou T pode ser definida para representar todos os possíveis padrões de corte de P. Os vértices do tipo Ou representam partes da placa, e os vértices do tipo E possíveis cortes na placa. Nesta representação, cada filho de um vértice v do tipo Ou é um vértice do tipo E, que representa uma possível alternativa de corte na placa representada por v, e cada vértice u do tipo E possui

dois filhos (placas obtidas após o corte representado por u). A raiz da árvore é um vértice do tipo Ou que representa a placa P.

Na árvore T assim obtida, a busca por um padrão de corte pode ser feita examinando-se todas as possíveis alternativas de cortes a partir de um nó Ou e, para cada vértice E, as peças obtidas após o corte.

A partir desta representação por grafos E/Ou, algoritmos de busca e programação dinâmica podem ser aplicados para encontrar uma solução ótima. Em 2006, Arenales [2] propôs uma abordagem utilizando grafos E/Ou para geração de padrões de corte em placas defeituosas. Em seguida, em 2007, Morabito [14] desenvolveu um método para geração de padrões de cortes restritos utilizando esses grafos.

Nesta representação, um padrão de corte pode ser obtido a partir de uma subárvore (subárvore-solução) com a mesma raiz de T, onde cada vértice Ou desta subárvore possui um filho (corte escolhido) e cada vértice E possui 2 filhos (placas obtidas após o corte), sendo as peças deste padrão de corte, as folhas dessa subárvore-solução.

A Figura 3.4 ilustra em (a) uma placa P, em (b) um conjunto de m peças que pode ser obtido a partir de P, e em (c) um possível padrão de corte para P.

A Figura 3.5 ilustra uma árvore E/Ou T que modela possíveis cortes na placa P, onde as folhas são possíveis peças obtidas pelos cortes. Nesta árvore, os vértices do tipo Ou indicam que cada filho é um possível corte e os vértices E indicam que seus filhos são placas obtidas a partir de um corte. As arestas em destaque formam a subárvore-solução do padrão de corte representado na Figura 3.4 (c).

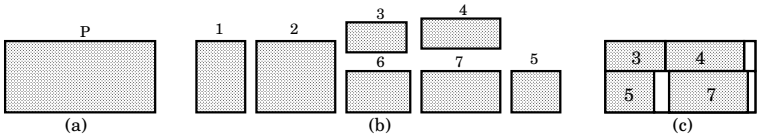

Figura 3.4: (a) Placa P. (b) Conjunto das m peças. (c) Padrão de corte de P.

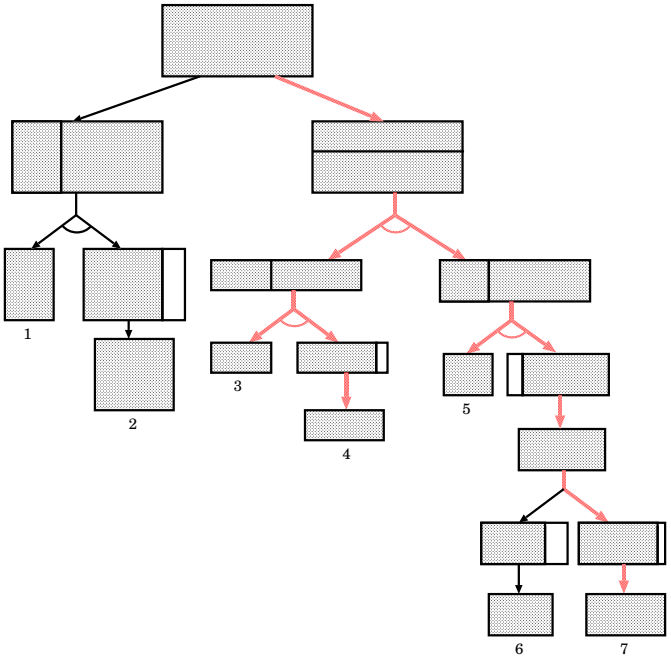

Figura 3.5: Árvore E/Ou de padrões de corte da Figura 3.4.

Muitos trabalhos limitam seus estudos a padrões guilhotinados, isto é, padrões obtidos por uma sequência de cortes em placas retangulares que produzem sempre dois novos retângulos. Alguns trabalhos adicionam ainda uma restrição para o número máximo de vezes que um tipo de peça poderá ser cortado a partir de uma chapa. A existência de possíveis defeitos nas placas também é considerada por alguns autores. Para mais informações sobre o problema, as referências [14, 2] podem ser consultadas.

3.3 Redes de Atividades

Grafos E/Ou também são muito utilizados na área de Engenharia de Software. Em 1998, Corandi e Westlechtel [6] apresentaram um modelo para configurar e administrar versões de software, que se baseia nestes grafos. Uma outra aplicação desses grafos nesta área é a sua utilização na representação de redes de atividades.

Redes de Atividades são basicamente um conjunto de atividades e de relações de dependência entre elas, onde a execução de uma atividade pode ser iniciada somente após a execução de um subconjunto de atividades da qual esta depende. Redes de atividades são muito utilizadas para modelar projetos de software, onde, a partir dessa rede, analisa-se a melhor forma de concluir o projeto.

Em uma rede de atividades, para cada atividade p_i, existe um subconjunto de atividades O_i, formado pela união de r_i subconjuntos não necessariamente disjuntos ($O_i = O_i^1 \cup O_i^2 \cup ... \cup O_i^{r_i}$), de forma que a atividade p_i pode ser executada somente após a execução de todas as atividades de pelo menos um subconjunto O_i^k, $1 \leq k \leq r_i$.

A Tabela 3.1 exemplifica uma rede de atividades.

Atividade	O_i	**Depedência**
p_1	$\{p_2, p_3, p_4, p_5\}$	$\{p_2, p_3\}$ ou $\{p_3, p_4\}$ ou $\{p_4, p_5\}$
p_2	$\{p_6\}$	$\{p_6\}$
p_3	$\{p_6, p_7, p_8\}$	$\{p_6, p_7, p_8\}$
p_4	$\{p_6, p_7, p_8\}$	$\{p_6\}$ ou $\{p_7\}$ ou $\{p_8\}$
p_5	$\{p_8\}$	$\{p_8\}$
p_6	$\{\ \}$	$Nenhuma$
p_7	$\{\ \}$	$Nenhuma$
p_8	$\{\ \}$	$Nenhuma$

Tabela 3.1: Descrição de uma rede de atividades.

3.3.1 Representação de Redes de Atividades como grafos E/Ou.

Uma rede de atividades pode ser representada como um grafo E/Ou da seguinte forma:

- Cada atividade p_i é representada por um vértice.
- Se $r_i = 1$, então o vértice p_i é rotulado como E e uma aresta (p_i, p_j) é criada para toda atividade $p_j \in O_i$.
- Se $r_i \neq 1$ e $|O_i^k| = 1$ para todo $1 \leq k \leq r_i$, então o vértice p_i é rotulado como Ou e uma aresta (p_i, p_j) é criada para toda atividade $p_j \in O_i$.
- Se $r_i \neq 1$ e $|O_i^k| \neq 1$ para algum $1 \leq k \leq r_i$, o vértice p_i é rotulado como Ou e, para todo subconjunto O_i^k $(1 \leq k \leq r_i)$, cria-se um vértice auxiliar p_i^k do tipo E, uma aresta (p_i, p_i^k) é criada e uma aresta (p_i^k, p_j) é adicionada para toda atividade $p_j \in O_i^k$.

A Figura 3.6 ilustra a representação da rede de atividades correspondente às atividades e relações de depêndencia da Tabela 3.1.

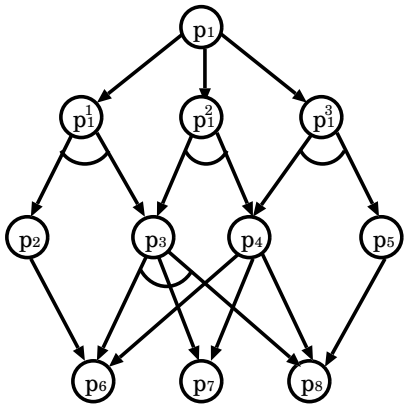

Figura 3.6: Representação da rede de atividades da Tabela 3.1.

3.3.2 Escalonamento de grafos E/Ou

A interpretação de grafos E/Ou como redes de atividades muitas vezes possibilita uma melhor compreensão de diversos problemas relativos a esses grafos. Um exemplo de problema onde a interpretação como redes de atividades se torna apropriada é o problema de escalonamento de grafos E/Ou.

Problema: Escalonamento de grafos E/Ou

Instância: Um grafo E/Ou conexo e ponderado G com um vértice fonte $s \in V(G)$, onde cada aresta tem um peso $\tau(e) \in Z^+$.

Questão: Encontrar o menor valor $t(v_j)$, $\forall\ v_j \in V(G)$, satisfazendo às seguintes condições:

$$\begin{cases} t(v_j) = 0 \text{ se } v_j \text{ for sumidouro,} \\ t(v_j) \geq \max\limits_{v_i \in O(v_j)} \{t(v_i) + \tau(v_j, v_i)\} \text{ se } f(v_j) = E, \\ t(v_j) \geq \min\limits_{v_i \in O(v_j)} \{t(v_i) + \tau(v_j, v_i)\} \text{ se } f(v_j) = Ou, \\ t(v_j) \geq 0, \forall\ v_j \end{cases}$$

O problema de escalonamento de grafos E/Ou pode ser facilmente interpretado como uma rede de atividades, onde as arestas (u, v), $\forall\ u, v \in V(G)$, representam o tempo necessário para que a atividade u seja informada do término da execução da atividade v. O objetivo do problema é encontrar o menor instante de tempo em que cada atividade pode iniciar sua execução.

Na década de 90, Dinic, ao estudar o problema de escalonamento de grafos E/Ou, exibiu um algoritmo de tempo polinomial para solucioná-lo. Posteriormente, seus resultados foram melhorados, sendo o mais recente um algoritmo de complexidade $O(nm)$ proposto por Velsky em 2002 [1], para grafos E/Ou possivelmente cíclicos.

Sendo assim, apresentamos um algoritmo desenvolvido nesta dissertação para grafos E/Ou acíclicos, de complexidade $O(m)$.

Algoritmo 3.1: Escalonamento de grafos E/Ou acíclicos

Dados: Um grafo E/Ou $G = (V, E)$.
Entrada: Uma lista inicialmente vazia $t[\,]$.
Saída: $t[v]$ correspondendo ao valor $t(v)$ dado pela recorrência da questão do problema, $\forall\ v \in V(G)$.

início

Seja $v_1, v_2, v_3, ..., v_n$ uma disposição dos vértices de G, dada por uma ordenação topológica inversa;

para cada $v_i \in V(G)$ **faça**
  se $(f(v_i) = E)$ *ou* $(|O_{v_i}| = 0)$ **então**
    $t[v_i] := 0$;
  senão
    $t[v_i] := \infty$;

para *i:=2 até n* **faça**
  para cada *aresta* $e = (v_i, w) \in E(G)$ **faça**
    se $(f(v_i) = E)$ *e* $(t[v_i] < \tau(e) + t[w])$ **então**
      $t[v_i] := \tau(e) + t[w]$;
    se $(f(v_i) = Ou)$ *e* $(t[v_i] > \tau(e) + t[w])$ **então**
      $t[v_i] := \tau(e) + t[w]$;

No Algoritmo 3.1, o valor $t[v]$ é calculado conforme v é visitado, $\forall\ v \in V(G)$. Neste algoritmo, a ordenação topológica inversa torna-se necessária para garantir que, ao visitar um vértice v, todos os seus vizinhos de saída já tenham sido visitados.

Ao término deste algoritmo, o valor de v_i no escalonamento encontra-se armazenado em $t[v_i]$. Como (i) a ordenação topológica pode ser obtida em tempo $O(m)$ [7], (ii) o grafo é conexo e (iii) para cada vértice são analisadas todas as suas arestas de saída, a complexidade deste algoritmo é $O(m)$.

A Figura 3.7 ilustra um exemplo de escalonamento de um grafo E/Ou.

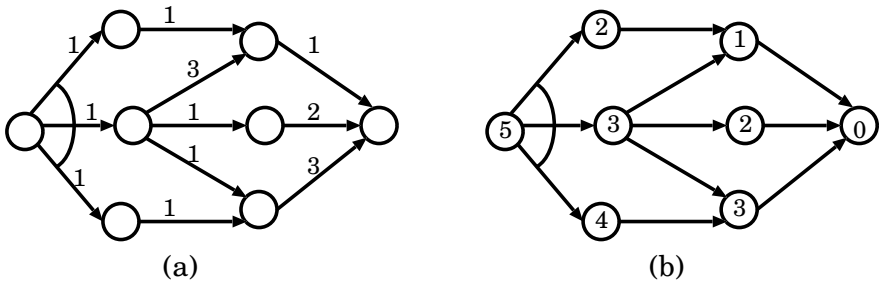

Figura 3.7: (a) Um grafo E/Ou G. (b) Grafo G escalonado.

No problema de escalonamento para grafos E/Ou, os vértices E equivalem a um vértice do tipo Max, ou seja, vértices que dependem apenas do vizinho de saída de maior peso, enquanto os vértices Ou equivalem a vértices do tipo Min, ou seja, vértices que dependem apenas do vizinho de saída de menor peso. Uma variação de árvores com esta propriedade são as árvores de jogos (*game trees*). Estas árvores são muito utilizadas em teoria combinatória dos jogos. Mais informações sobre árvores de jogos podem ser consultadas em [16].

3.4 Fórmulas Booleanas

Outra utilização de grafos E/Ou consiste na modelagem de fórmulas boolenas. Estas são basicamente um conjunto U de variáveis e um conjunto F de operações (conjunções/disjunções) entre cláusulas. Cada cláusula é composta por operações (conjunções/disjunções) entre literais das variáveis de U.

Em 2008, Laber [13] desenvolveu um algoritmo probabilístico competitivo que, dada uma determinada atribuição de valores às variáveis de uma fórmula F, e considerando que cada variável possui um custo de leitura, utiliza árvores E/Ou para determinar o custo mínimo de se calcular o valor da fórmula F.

Exemplo: Considere a fórmula booleana $F = (x_1+x_2+x_3)(x_4.x_5)$, onde as variáveis x_1, x_2, x_3, x_4 e x_5 possuem respectivamente valores $0, 1, 1, 1, 1$ e custos de leitura $3, 5, 2, 6, 4$. Temos que $F = (0+1+1)(1.1) = 1$. Para sabermos o valor de F, basta conhecermos os valores de x_5, x_4 e (x_3 ou x_2), sendo o custo de leitura de x_5, x_4 e x_2 igual a $4+6+5 = 15$, enquanto o custo de leitura de x_5, x_4 e x_3 igual a $4+6+2 = 12$. Caso a variável x_1 fosse lida, como esta possui valor 0, obrigatoriamente x_3 ou x_2 deverão ser lidas. A Figura 3.8 ilustra uma árvore E/Ou para F, no modelo utilizado por Laber para o desenvolvimento do algoritmo probabilístico.

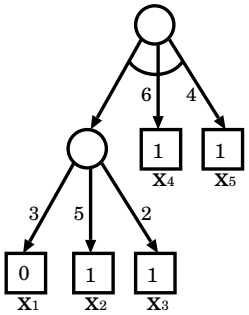

Figura 3.8: Árvore E/Ou da fórmula $F = (x_1 + x_2 + x_3)(x_4.x_5)$.

3.4.1 Construção de Árvores E/Ou

Em um grafo E/Ou, os vértices possuem características que se assemelham a conjunções e disjunções de uma fórmula booleana. Para uma conjunção ser satisfatível, esta depende da satisfabilidade de todos os termos que a compõem, enquanto uma disjunção depende da satisfabilidade de pelo menos um dos termos que a compõem. Assim, conjunções de cláusulas podem ser facilmente modeladas por vértices E, e disjunções por vértices Ou. A Figura 3.9 ilustra em (a) um vértice E representando a conjunção $(C_1)(C_2)(C_3)$ e em (b) um vértice Ou representando a disjunção $(C_1) + (C_2) + (C_3)$.

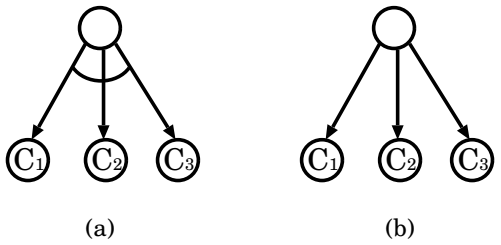

Figura 3.9: (a) Representação de conjunção. (b) Representação de disjunção.

Desta forma, a partir das características dos vértices E/Ou, é possível construir uma árvore E/Ou modelando uma fórmula booleana F (não necessariamente CNF).

A seguir é apresentado o passo-a-passo de uma construção recursiva destas árvores. Nesta construção, consideramos que as cláusulas são indentificadas por parênteses e que a fórmula não possui parentização redundante. Para cada iteração, serão examinadas apenas as cláusulas mais externas da fórmula corrente sendo analisada:

Passo 1 Criar um vértice f, que será a raiz da árvore, representando a fórmula F.

Passo 2 Sejam $C_i, C_{i+1}, ..., C_k$ as expressões booleanas mais externas da fórmula a ser analisada pela iteração corrente, e v o vértice que representa esta fórmula.

2.1 Se a fórmula corrente é uma conjunção $(C_i \wedge C_{i+1} \wedge ... \wedge C_k)$ então $f(v) = E$. Caso contrário, $f(v) = Ou$.

2.3 Se $f(v) = E$ faça:

- Criar um vértice v_j representando a cláusula C_j $(i \leq j \leq k)$.
- Adicionar uma aresta (v, v_j), para $i \leq j \leq k$.
- Repetir o Passo 2 para cada cláusula C_j $(i \leq j \leq k)$, que não representa um único literal.

2.4 Caso contrário, se $f(v) = Ou$

- Criar um vértice v_j vizinho de saída de v, para cada cláusula C_j ($i \leq j \leq k$) entre disjunções ($... \vee C_j \vee ...$).
- Repetir o Passo 2 para cada uma das cláusulas entre disjunções, que não representam um único literal.
- Criar um vértice w vizinho de saída de v, para cada conjunção maximal de cláusulas $S \subseteq \{C_i, ..., C_k\}$. Exemplo: $(... \vee \underline{C_j \wedge C_{j+1} \wedge ... C_{|S|+j-1}} \vee ...)$.
- Repetir o Passo 2 para cada conjunção maximal S analisada anteriormente.

Na construção descrita, as folhas da árvore representam literais de F, e cada cláusula é representada por um nó interno. Sendo m o número de cláusulas de F e k o número de literais de F (considerando repetições), esta árvore terá no máximo $k + 2m - 1$ vértices: k folhas, m vértices representando cláusulas, e no máximo $m - 1$ vértices internos[1] que não representam cláusulas. Além disso, cada vértice não folha terá no mínimo 2 filhos (caso contrário F teria parentização redundante).

A construção de árvores E/Ou modelando fórmulas booleanas pode ser utilizada por exemplo para o desenvolvimento de novos algoritmos probabilísticos[2] para o problema de satisfabilidade de fórmulas booleanas.

A Figura 3.10 ilustra uma árvore E/Ou construída a partir da fórmula $F = (\bar{x}y\bar{z})(\bar{x}+ y + z) + (xyz) + ((x + y + z)(\bar{x}\bar{y}\bar{z}))$.

[1] Valor que decorre do resultado de que, em uma árvore com m folhas, onde cada nó interno tem pelo menos 2 filhos, existem no máximo $m - 1$ nós internos.

[2] Algoritmos probabilísticos são algoritmos que utilizam escolhas aleatórias, em uma ou mais tomadas de decisão da sua execução [4].

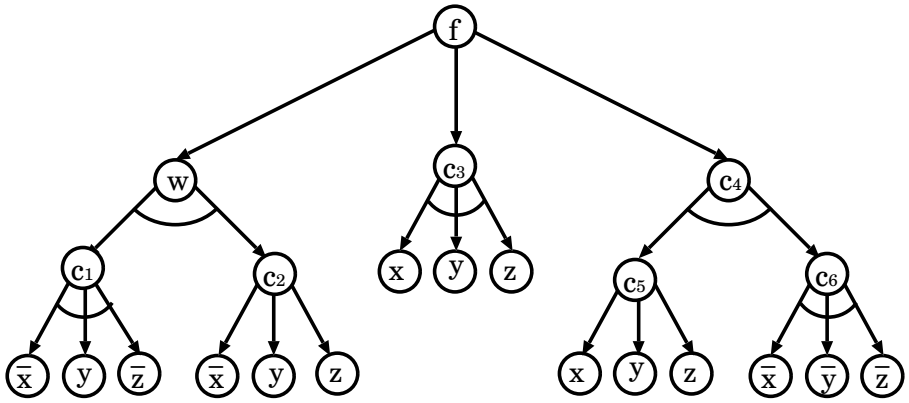

Figura 3.10: Árvore E/Ou de uma fórmula booleana.

3.5 Hipergrafos direcionados

Um hipergrafo direcionado $H = (V, E)$ é uma generalização de digrafos, onde $V(H) = \{v_1, v_2, ..., v_n\}$ é um conjunto de vértices e $E(H) = \{E_1, E_2, ..., E_m\}$ é um conjunto de hiperarcos. Todo hiperarco $E_i \in E(H)$ é um par ordenado de subconjuntos disjuntos de vértices (S, T), onde $S(E_i)$ é o conjunto de saída de E_i e $T(E_i)$ o seu conjunto de entrada. Neste trabalho, hipergrafos direcionados serão denominados simplesmente como hipergrafos. A Figura 3.11 ilustra um hipergrafo direcionado.

A dimensão de um hipergrafo H não é medida simplesmente pelo seu número de vértices $|V(H)| = n$ e pelo seu número de hiperarcos $|E(H)| = m$. Em um hipergrafo, a sua dimensão depende da cardinalidade de suas hiperarestas, sendo o tamanho de H definido como o somatório da cardinalidade das hiperarestas:

$$tamanho(H) = \sum_{E_i \in E(H)} |E_i|, \text{ onde } |E_i| = |S(E_i)| + |T(E_i)|$$

Um *B-arco* (*backward hyperarc*) é um hiperarco $E_i = (S, T)$ onde $|T(E_i)| = 1$, já um *F-arco* (*forward hyperarc*) é um hiperarco $E_i = (S, T)$ onde $|S(E_i)| = 1$ [11]. A

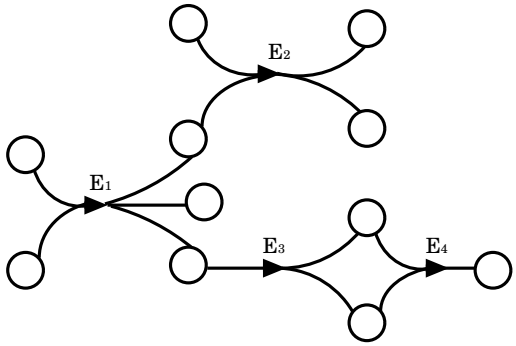

Figura 3.11: Exemplo de um hipergrafo direcionado.

Figura 3.12 ilustra em (a) um exemplo de B-arco e em (b) um exemplo de F-arco.

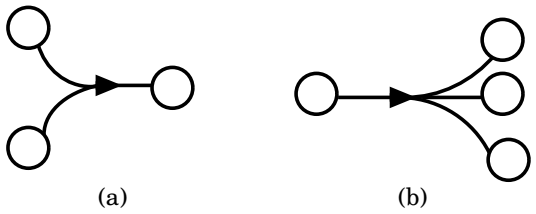

Figura 3.12: (a) Exemplo de B-arco. (b) Exemplo de F-arco.

Um *B-grafo* (ou B-hipergrafo) é um hipergrafo onde todos os hiperarcos são B-arcos. Um *F-grafo* (ou F-hipergrafo) é um hipergrafo onde todos os hiperarcos são F-arcos.

F-grafos têm sido muito estudados no contexto de problemas de transporte urbano [11].

3.5.1 Transformação de F-grafos em grafos E/Ou

Dado um F-grafo H, este pode ser facilmente transformado em um grafo E/Ou G equivalente da seguinte forma:

1. Para todo vértice $v \in V(H)$ atribua valor Ou para o rótulo $f(v)$.

2. Para cada *F-arco* E_i, onde $T(E_i) \geq 2$, faça:

   - Criar um vértice v_i, onde $f(v_i) = E$.
   - Adicionar uma aresta (u, v_i), onde $u \in S(E_i)$.
   - Para cada vértice $w_j \in T(E_i)$ adicionar uma aresta (v_i, w_j).
   - Remover o *F-arco* E_i.

3. Transforme todo *F-arco* $E_i = (S, T)$, tal que $|S(E_i)| = |T(E_i)| = 1$ em uma aresta $e_i = (u, v)$, onde $S(E_i) = \{u\}$ e $T(E_i) = \{v\}$.

Considerando que cada operação de inserção e remoção de vértices e arestas é feita em tempo constante, a transformação acima será executada em tempo $O(tamanho(H))$.

Como podemos observar, dado o hipergrafo H, onde

$$|V(H)| = n \ , \ |E(H)| = m \text{ e } tamanho(H) = \sum_{E_i \in E(H)} |E_i|,$$

H será transformado em um grafo E/Ou G, onde

$$|V(G)| \leq (|V(H)| + |E(H)|) \text{ e } |E(G)| \leq tamanho(H).$$

A primeira desigualdade se deve ao fato de hiperarestas com apenas um destino não adicionarem vértices na construção do grafo E/Ou.

A Figura 3.13 ilustra a transformação de um F-grafo em um grafo E/Ou.

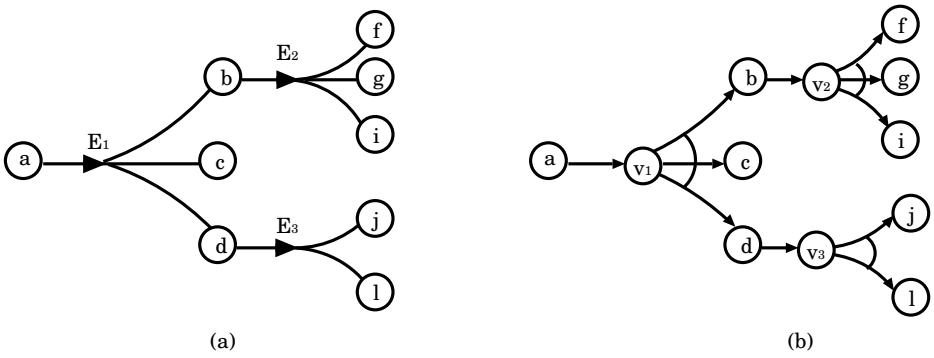

Figura 3.13: Transformação de um F-$grafo$ em um grafo E/Ou.

3.5.2 Transformação de grafos E/Ou em F-grafos

Conforme vimos na subseção anterior, F-grafos podem ser facilmente transformados em grafos E/Ou. De forma semelhante, podemos transformar grafos E/Ou em F-grafos da seguinte forma:

- Substituir as arestas de saída de todo vértice v do tipo E por um F-$arco$ $(\{v\}, O_v)$.

A Figura 3.14 ilustra a transformação de um grafo E/Ou em um F-$grafo$.

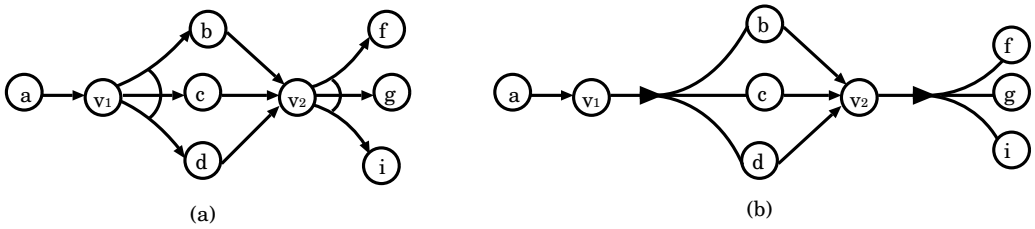

Figura 3.14: Transformação do grafo E/Ou em (a) para o F-grafo em (b).

4 OS PROBLEMAS MIN–E/OU E MIN–X-DE-Y

O problema MIN–E/OU consiste basicamente em, dado um grafo E/Ou ponderado, com um nó fonte s, encontrar um subgrafo (subgrafo-solução) que satisfaça a um determinado conjunto de restrições. A seguir, são descritas suas formulações como problema de otimização e como problema de decisão:

Versão de Otimização: MIN–E/OU

Instância: Um grafo E/Ou acíclico, conexo e ponderado $G = (V, E)$ com um vértice fonte $s \in V$, e tal que cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$ e cada aresta e possui um peso $\tau(e) \in Z^+$.

Questão: Determinar o peso mínimo de um subgrafo $H = (V', E')$ de G, que satisfaça a:

1. $s \in V'$;
2. se $v \in V'$ e $f(v) = E$, então todos as arestas de saída de v devem pertencer a E';
3. se $v \in V'$ e $f(v) = Ou$, então exatamente uma das arestas de saída de v deve pertencer a E'.

Versão de Decisão: MIN–E/Ou

Instância: Um inteiro positivo k e um grafo E/Ou acíclico, conexo e ponderado $G = (V, E)$ com um vértice fonte $s \in V$, onde cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$ e cada aresta possui um peso $\tau(e) \in Z^+$.

Questão: Determinar se existe um subgrafo-solução $H = (V', E')$ de G tal que o somatório do peso das arestas de H não exceda k, atendendo a:

1. $s \in V'$;
2. se $v \in V'$ e $f(v) = E$, então todas as arestas de saída de v devem pertencer a E';
3. se $v \in V'$ e $f(v) = Ou$, então exatamente uma das arestas de saída de v deve pertencer a E'.

É importante observar que embora a definição de grafos E/Ou permita vértices do tipo Ou com mais de uma aresta de saída em subgrafos-solução, no problema MIN–E/Ou estes vértices possuem exatamente uma aresta de saída, referindo-se apenas a subgrafos-solução minimais.

Neste capítulo, serão utilizadas as seguintes notações auxiliares:

$V_E(G)$:	Conjunto dos vértices do tipo E de G.
$V_{Ou}(G)$:	Conjunto dos vértices do tipo Ou de G.
$\tau(G)$:	Somatório dos pesos das arestas de G.
$s(G)$:	Vértice s do grafo G.
$H_{opt}(G)$:	Subgrafo-solução ótimo de G

4.1 NP-Completude

Nesta seção, é demonstrada a prova de NP-completude do problema MIN–E/Ou para grafos em geral, obtida por Sahni em 1974 [17].

Lema 11 MIN–E/Ou $\in NP$.

Prova: Seja G o grafo E/Ou de entrada para o problema MIN–E/Ou. Um certificado C para uma resposta *sim* do problema é um subgrafo H de G tal que: (i) $s \in V(H)$; (ii) para todo $v \in V(H)$, se $f(v) = E$ então todas as arestas de saída de v pertencem a $E(H)$, e se $f(v) = Ou$ então apenas uma das arestas de saída de v pertence a $E(H)$; (iii) o custo de H não excede k. Claramente, a corretude desse certificado C pode ser verificada em tempo polinomial, pois o custo de H é obtido através do somatório dos pesos de suas arestas. Logo, existe um certificado para MIN–E/Ou que pode ser verificado em tempo polinomial. ■

Teorema 12 3-Sat $\propto$ MIN–E/Ou

Problema: 3-Sat

Instância: Uma fórmula booleana F na forma normal conjuntiva, com m cláusulas e n variáveis, onde cada cláusula possui no máximo 3 literais.

Questão: Existe alguma atribuição de valores às variáveis de F tal que F seja satisfeita?

Prova: Para mostrar que MIN–E/Ou é NP-completo, Sahni apresentou uma redução que transforma em tempo polinomial uma instância qualquer F do problema 3-Sat em uma instância I do problema MIN–E/Ou, tal que F é satisfatível se e somente se existir um subgrafo-solução de custo menor ou igual a k para I.

A construção de um grafo E/Ou G a partir de F é dada da seguinte forma:

- um vértice fonte s de rótulo E é criado;
- para toda variável x_i da fórmula F, cria-se um vértice x_i do tipo Ou;
- um vértice f vizinho de saída de s é criado (f será um vértice do tipo E);
- para cada cláusula C_i de F, cria-se um vértice c_i do tipo Ou;
- para cada vértice x_i são criados dois vizinhos de saída: t_{x_i} e f_{x_i};
- para cada vértice t_{x_i} ou f_{x_i} é criado um vizinho de saída v_j que é um vértice terminal (sumidouro);
- uma aresta (s, x_i) é criada para todo vértice x_i;
- uma aresta (f, c_i) é adicionada para todo vértice c_i;
- uma aresta (c_i, t_{x_j}) é adicionada se o literal x_j ocorre na cláusula C_i de F;
- uma aresta (c_i, f_{x_j}) é adicionada se o literal $\bar{x}_j$ ocorre na cláusula C_i de F;
- as arestas de saída dos vértices t_{x_i} e f_{x_i} possuem peso 1, as demais arestas peso 0.

Considerando que a criação de cada vértice e aresta é feita em tempo constante, esta construção pode ser executada em tempo $O(n+m)$, onde n é o número de variáveis e m o número de cláusulas.

A Figura 4.1 ilustra o grafo E/Ou obtido a partir da transformação da fómula $F = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_1} + x_2)$.

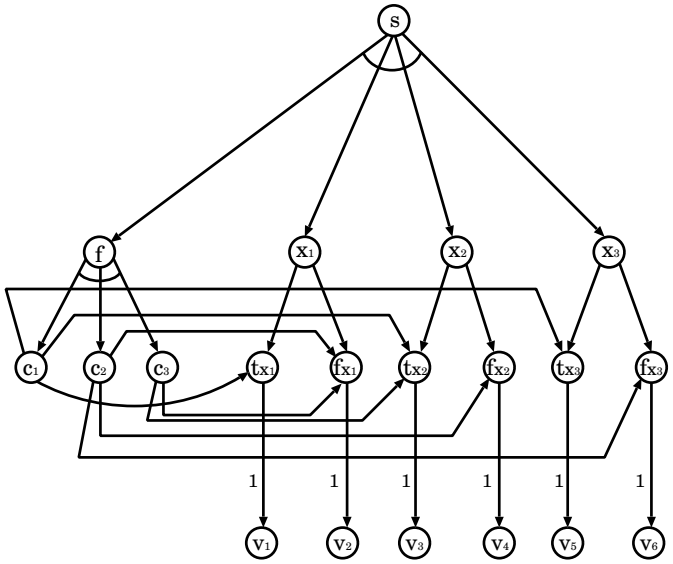

Figura 4.1: Grafo E/Ou da fórmula $F = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_1} + x_2)$.

A partir do grafo G obtido pela construção acima, temos que todo subgrafo-solução H de G possui as seguintes características:

- $s \in V(H)$;
- como s é um vértice do tipo E, todos os seus vizinhos de saída $(f, x_1, x_2, ..., x_n)$ também pertencem a $V(H)$;
- como todo vértice x_i é do tipo Ou, apenas uma aresta de saída de x_i pertence a $E(H)$;
- para todo vértice t_{x_i} ou $f_{x_i} \in V(H)$, seu respectivo vizinho de saída (vértice terminal) também pertence a $V(H)$;
- como f é um vértice do tipo E, seus vizinhos de saída $(c_1, c_2, c_3, ..., c_m)$ também pertencem a $V(H)$;
- como todo vértice c_i é do tipo Ou, então apenas uma de suas arestas de saída deve pertencer a $V(H)$.

Em um subgrafo-solução H de G, todo vértice x_i possui apenas um vizinho de saída (t_{x_i} ou f_{x_i}). Desta forma, podemos assumir que esta vizinhança induz uma atribuição de valores às variáveis de F: se t_{x_i} é vizinho de saída de x_i em H, assumimos que o valor da variável x_i em F é "verdadeiro"; caso contrário, assumimos que o valor é "falso". Os vizinhos de saída dos vértices x_i em H formam um conjunto de vértices A que induz uma atribuição de valores às variáveis de F.

Sabemos que $A \subseteq V(H)$, $|A| = n$ (número de variáveis), e que todo vértice t_{x_i} ou f_{x_i} possui uma aresta de saída de peso 1. Logo, qualquer subgrafo-solução possui custo maior ou igual a n (número de variáveis).

($\Rightarrow$) Supondo que F é satisfatível, então existe um conjunto A contido em um subgrafo-solução H que induz uma atribuição de valores que satisfaz F. Neste subgrafo H, os vértices c_i ($1 \leq i \leq m$) precisam ter apenas um vizinho de saída, pois são vértices do tipo Ou. Na construção de H, é possível escolher para c_i um vizinho de saída que corresponde a um elemento de A, pois caso contrário a cláusula C_i não é satisfeita pela atribuição induzida por A. Escolhendo para o vértice c_i um vizinho de saída pertencente a A, c_i não adiciona custo a H, pois o vértice escolhido já irá pertencer ao subgrafo-solução. Como para todos os vértices c_i é possível escolher um vizinho de saída que corresponda a um elemento de A, pois a fórmula F é satisfatível, então o subgrafo-solução H possui custo $k = n$.

($\Leftarrow$) Se existe um subgrafo-solução H de custo $k = n$ então, para cada vértice c_i $\in V(H)$, o seu vizinho de saída pertence ao conjunto $A \subseteq V(H)$, pois caso contrário c_i teria incrementado o custo de H, pois mais uma aresta de peso 1 teria sido adicionada a H, e consequentemente o peso de H seria superior a n. Se um vértice c_i tem um vizinho de saída contido em A, então a cláusula representada por c_i é satisfeita pela atribuição induzida por A (por construção). Logo como H tem custo n então todas as cláusulas de F são satisfeitas pela atribuição induzida por A, e

portanto F é satisfatível.

Logo, F é satisfatível se e somente se no grafo E/Ou correspondente for possível obter um subgrafo-solução H de custo $k = n$. ■

Corolário 13 MIN–E/Ou *é NP-completo.*

Segue diretamente do Lema 11 e do Teorema 12. ■

A Figura 4.2 ilustra o subgrafo-solução de custo $k = n = 3$, do grafo E/Ou ilustrado na Figura 4.1.

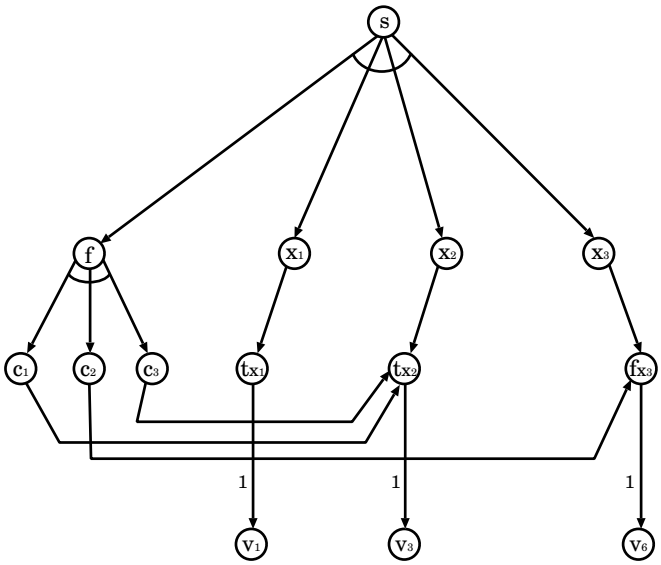

Figura 4.2: Subgrafo-solução para $F = (x_1 + x_2 + x_3)(\bar{x_1} + \bar{x_2} + \bar{x_3})(\bar{x_1} + x_2)$.

Posteriormente à prova apresentada por Sahni, Garey e Jhonson [12] mostraram que o problema MIN–E/Ou continua NP-completo para o caso onde todas as arestas possuem peso igual a 1. Para tal prova, podemos utilizar a construção anterior, porém considerando que todas as arestas possuem custo 1. Neste caso, a fórmula F é satisfatível se e somente se, no grafo E/Ou correspondente, for possível obter um subgrafo-solução H de custo $3n + 2m + 1$.

4.2 Casos Polinomiais para MIN–E/Ou

Seja G o grafo E/Ou de entrada do problema MIN–E/Ou. Apresentamos três casos em que o problema MIN–E/Ou pode ser solucionado em tempo polinomial:

1. G é um grafo E/Ou e todos os vértices de G são do tipo E (Grafo E) — Neste caso, G é o próprio subgrafo-solução;

2. G é um grafo E/Ou e todos os vértices de G são do tipo Ou (Grafo Ou) — Neste caso, o subgrafo-solução ótimo de G equivale ao caminho mínimo de s até um dentre todos sumidouros;

3. G é uma árvore. — Neste caso, o custo do subgrafo-solução ótimo de G pode ser obtido a partir da seguinte relação de recorrência [20]:

$$custo(s) = \begin{cases} 0, \text{ se } s \text{ é um sumidouro,} \\ \sum\limits_{v_j \in O_s} [\tau(s, v_j) + custo(v_j)], \text{ se } s \text{ é um vértice } E \\ \min\limits_{v_j \in O_s} [\tau(s, v_j) + custo(v_j)], \text{ se } s \text{ é um vértice } Ou \end{cases}$$

onde $custo(s)$ é o custo da subárvore-solução ótima enraizada por s.

Dada esta recorrência, para o caso 3, desenvolvemos o Algoritmo 4.1 para resolvê-lo.

Neste algoritmo, são utilizadas duas listas sequenciais C e F com n posições (cada posição se refere a um vértice da árvore). No início da execução do algoritmo, todas as posições de C e F estão vazias. Ao final da execução, o valor de $C[v]$ representa o custo da subárvore-solução ótima enraizada por v, e $F[v]$ armazena os filhos de v nesta árvore.

A complexidade deste algoritmo é $O(n)$, pois todo vértice será percorrido apenas uma vez, e terá o seu custo calculado.

Algoritmo 4.1: MIN–E/Ou para árvores

Dados: Uma árvore E/Ou $T = (V, E)$ enraizada por s.
Entrada: Duas listas $C[\,]$ e $F[\,]$ inicialmente vazias.
Saída: $C[v]$ com o custo da subárvore-solução ótima enraizada por v, e $F[v]$ os filhos de v nesta subárvore, $\forall\ v \in V(T)$.

```
início
    para cada vértice v_j ∈ V(G) faça
        se f(v_j) = E então
            C[v_j] := 0;
        senão
            C[v_j] := ∞;
    Custo(s);
procedimento Custo(v_i: vértice v_i ∈ V(T))
    se O_{v_i} ≠ {} então
        para cada vértice w ∈ O_i faça
            Custo(w);
            se f(w) = E então
                C[v_i] := C[v_i] + C[w] + τ(v_i, w);
                F[v_i] := F[v_i] ∪ {w};
            se (f(v_i) = Ou) e (C[v_i] > C[w] + τ(v_i, w)) então
                C[v_i] := C[w] + τ(v_i, w);
                F[v_i] := {w};
    senão
        C[v_i] := 0;
```

É importante ressaltar que embora sabe-se na literatura que o problema MIN–E/Ou é polinomial para árvores, durante as pesquisas dessa dissertação não encontramos na literatura um algoritmo para este caso. Sendo o Algoritmo 4.1, uma das contribuições desta dissertação.

4.3 Limite Inferior

Conforme visto na seção 4.1, o problema MIN–E/Ou é NP-completo. No entanto, na Seção 3.3.2, foi demonstrado que o problema escalonamento para grafos E/Ou [1] é polinomial. Embora estes problemas sejam distintos, podemos observar que a entrada de ambos é um grafo E/Ou.

Considerando que o mesmo grafo G seja entrada para ambos os problemas, claramente o problema de escalonamento produz um inteiro $t(s)$ que é um limite inferior para a solução ótima do problema MIN–E/Ou. Assim, embora MIN–E/Ou seja NP-completo, para os casos onde $t(s) > k$ é possível resolvê-lo em tempo polinomial.

Podemos ainda tornar o valor desse limite inferior mais justo, alterando o Algoritmo 3.1 para incluir no peso de um vértice E o valor de todas as suas arestas de saída. Neste caso, o algoritmo não será mais um algoritmo de escalonamento, mas um algoritmo que retorna um limite inferior dado pela recorrência apresentada nesta seção.

$$\begin{cases} t(v_i) = 0 \text{ se } v_i \text{ for sumidouro,} \\ t(v_i) \geq \sum\limits_{v_j \in O(v_i)} \{\tau(v_i, v_j)\} + \max\limits_{v_j \in O(v_i)} t(v_i) \text{ se } f(v_i) = E, \\ t(v_i) \geq \min\limits_{v_j \in O(v_i)} \{t(v_j) + \tau(v_i, v_j)\} \text{ se } f(v_i) = Ou \end{cases}$$

Algoritmo 4.2: Limite inferior para MIN–E/Ou

Dados: Um grafo E/Ou $G = (V, E)$.
Entrada: Duas listas $t[\,]$ e $F[\,]$ inicialmente vazias.
Saída: $t[v_i]$ correspondendo ao valor $t(v_i)$ dado pela recorrência, $F[v_i]$ o vizinho de saída v_j da recorrência, $\forall v_i \in V(G)$.

```
início
    Seja v_1, v_2, v_3, ..., v_n uma disposição dos vértices de G, dada por uma
    ordenação topológica inversa;
    para cada v_i ∈ V(G) faça
        se (f(v_i) = E) ou (|O_{v_i}| = 0) então
            t[v_i] := 0;
        senão
            t[v_i] := ∞;
    para i:=2 até n faça
        aux := 0;
        para cada aresta e = (v_i, w) ∈ E(G) faça
            se (f(v_i) = E) então
                t[v_i] := t[v_i] + τ(e);
                se t[w] > aux então
                    aux := t[w];
                    F[v_i] := w;
            se (f(v_i) = Ou) e (t[v_i] > τ(e) + t[w]) então
                t[v_i] := τ(e) + t[w];
                F[v_i] := w;
        se (f(v_i) = E) então
            t[v_i] := t[v_i] + aux;
```

O Algoritmo 4.2 gera um inteiro $t[v]$ correspondendo ao valor $t(v)$ da recorrência, para todo vértice $v \in V(G)$.

Sendo assim, o valor $t[s]$ é o peso de um subgrafo C de G, onde C corresponde a um caminho de s até um sumidouro, que satisfaça à seguinte propriedade: para todo vértice v_i contido neste caminho, o vértice v_j subsequente é o vértice que

corresponde ao valor $\max_{v_j \in O_i} t[v_j]$ se $f(v_i) = E$, caso contrário o vértice v_j corresponde ao valor $\min_{v_j \in O_i} (\tau(v_i, v_j) + t(v_j))$. A C também são adicionadas todas as arestas de saída dos vértices E contidos no caminho anteriormente descrito.

Seja $Alg(G)$ o valor de $t[s(G)]$ obtido pelo Algoritmo 4.2. Provamos que este Algoritmo gera um limite inferior para o MIN–E/Ou da seguinte forma:

1. $\tau(C) = Alg(G)$, conforme descrito anteriormente.
2. $Alg(G) \leq Alg(H_{opt}(G))$ pois, caso contrário, o subgrafo C correspondente a $Alg(G)$ não seria o que minimiza a recorrência.
3. $Alg(H_{opt}(G)) \leq \tau(H_{opt}(G))$, pois $\tau(H_{opt}(G))$ corresponde ao somatório dos pesos de todas as arestas de $H_{opt}(G)$, e o subgrafo C' de $H_{opt}(G)$ não necessariamente possui todas as arestas de H_G.

Logo $\tau(C) = Alg(G) \leq Alg(H_{opt}(G)) \leq \tau(H_{opt}(G))$. Portanto, o Algoritmo 4.2 gera um limite inferior para MIN–E/Ou. ■

A Figura 4.3 ilustra um grafo E/Ou G, e seus subgrafos $H_{opt}(G)$ e C.

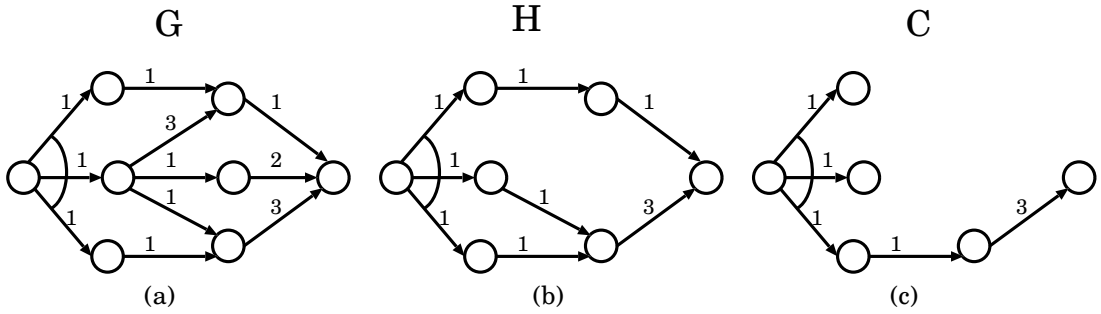

Figura 4.3: (a) Grafo G. (b) Subgrafo-solução $H_{opt}(G)$. (c) Subgrafo C de G.

4.4 Limite Superior e Aproximação

Nesta seção, faremos uso intenso da relação de recorrência utilizada para árvores E/Ou na Seção 4.2. Para auxiliar a leitura desta seção, abaixo reapresentamos tal recorrência:

$$custo(s) = \begin{cases} 0, \text{ se } s \text{ é um sumidouro,} \\ \sum\limits_{v_j \in O_s} [\tau(s, v_j) + custo(v_j)], \text{ se } s \text{ é um vértice } E \\ \min\limits_{v_j \in O_s} [\tau(s, v_j) + custo(v_j)], \text{ se } s \text{ é um vértice } Ou \end{cases}$$

Para uma árvore E/Ou T, esta recorrência calcula a solução ótima, pois todo vértice em uma árvore tem grau de entrada 1. Consequentemente, para toda aresta $e = (u, v) \in E(T)$ vale: se $e \in H_{opt}(T)$ então $\tau(e)$ foi contado exatamente uma vez no cálculo de $custo(s)$; caso contrário, $\tau(e)$ não foi contado. Esta consequência pode ser observada pelo fato de que $custo(s) = \tau(H_{opt}(T))$, e, para todo $e = (u, v) \in E(H_{opt}(T))$, existe um único caminho de s até u.

A Figura 4.4 ilustra em (a) uma árvore E/Ou, e em (b) o custo de cada vértice dessa árvore. Em (b) é apresentada a subárvore-solução ótima $H_{opt}(T)$ de T (arestas pontilhadas), onde $\tau(H_{opt}(T)) = custo(s)$.

Conforme dito anteriormente, em uma árvore T, $\tau(e)$ será contabilizado no máximo uma vez no cálculo de $custo(s)$. No entanto, se o grafo não for uma árvore, tal afirmação não necessariamente será verdade. Denotamos por $c(e)$ o número de vezes em que $\tau(e)$ foi contabilizado em $custo(s)$, através da aplicação da recorrência acima em um grafo E/Ou G, onde $e \in V(G)$.

A Figura 4.5 ilustra um grafo E/Ou onde $c(v_5, v_6) = 3$.

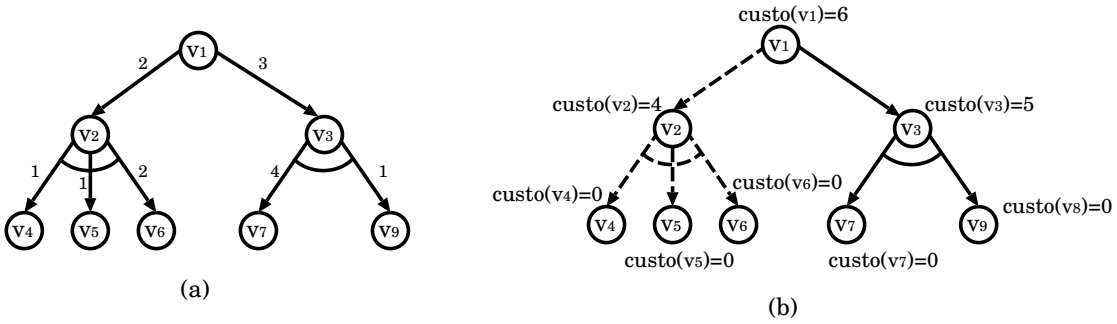

Figura 4.4: (a) Árvore E/Ou. (b) Custos dos vértices e solução ótima de (a).

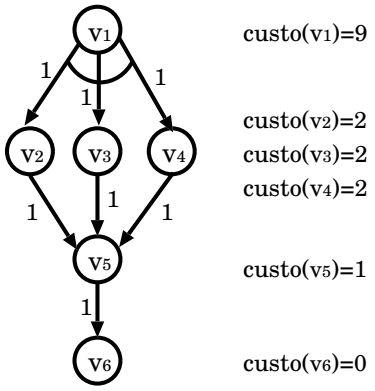

Figura 4.5: Grafo E/Ou onde $c(v_5, v_6) = 3$.

Para o caso em que o grafo não é uma árvore, não é possível utilizar esta recorrência para encontrar o subgrafo-solução ótimo. Como alguns vértices possuem grau de entrada maior que 1, possivelmente s contabilizará algumas arestas mais de uma vez e, portanto, o subgrafo-solução que minimiza o peso de s não necessariamente será o subgrafo-solução ótimo. No entanto, podemos utilizar esta mesma recorrência para encontrar um subgrafo-solução viável (não necessariamente ótimo).

O Algoritmo 4.3 encontra uma solução viável para o problema MIN–E/Ou, baseando-se na recorrência para árvores E/Ou.

Algoritmo 4.3: Limite superior e aproximação para MIN–E/Ou

Dados: Um grafo E/Ou $G = (V, E)$.
Entrada: Duas listas $t[\,]$ e $F[\,]$ inicialmente vazias.
Saída: $t[v_i]$ correspondendo ao valor $custo(v_i)$ dado pela aplicação da recorrência de árvores E/Ou em G, e $F[v_i]$ contendo os vizinhos de saída utilizados no cálculo de $t[v_i]$, $\forall\ v_i \in V(G)$.

início
  Seja $v_1, v_2, v_3, ..., v_n$ uma disposição dos vértices de G, dada por uma ordenação topológica inversa;
  para cada $v_i \in V(G)$ **faça**
    se $(f(v_i) = E)$ *ou* $(|O_{v_i}| = 0)$ **então**
      $t[v_i] := 0$;
    senão
      $t[v_i] := \infty$;
  para *i:=2 até n* **faça**
    para cada *aresta* $e = (v_i, w) \in E(G)$ **faça**
      se $(f(v_i) = E)$ **então**
        $t[v_i] := t[v_i] + \tau(e) + t[w]$;
        $F[v_i] := F[v_i] \cup \{w\}$;
      senão
        se $(t[v_i] > \tau(e) + t[w])$ **então**
          $t[v_i] := \tau(e) + t[w]$;
          $F[v_i] := w$;

Neste algoritmo, são utilizadas duas listas sequenciais t e F com n posições cada (cada posição se refere a um vértice do grafo). No início da execução do algoritmo, todas as posições de t e F estão vazias. Ao final da execução, o valor de $t[v]$ representa o peso de v obtido pela recorrência de árvores E/Ou, e $F[v]$ armazena os vizinhos de saída de v cujos pesos foram contabilizados no cálculo de $t[v]$.

Após o término do Algoritmo 4.3, temos que, se um vértice v é do tipo Ou, então $|F[v]| = 1$, e se v é do tipo E, então $|F[v]| = |O_v|$. Assim, dada a lista F obtida

após o término do algoritmo, podemos construir o subgrafo-solução que minimiza $t[s]$. Para tal, basta percorrer a lista F a partir de s. Após visitar um vértice v, todos os vértices contidos em $F[v]$ deverão ser visitados. No final do percurso, os vértices visitados serão os vértices do subgrafo-solução G', onde uma aresta $(v, w) \in E(G')$ se e somente se v foi visitado e $w \in F[v]$.

A Figura 4.6 ilustra em (a) um grafo E/Ou G, em (b) o subgrafo-solução ótimo de G, e em (c) o subgrafo-solução G' de G obtido pelo Algoritmo 4.3.

A Figura 4.7 ilustra em (a) as listas F e t após o término do Algoritmo 4.3, onde a entrada é o grafo G da Figura 4.6, e em (b) como o subgrafo G' é construído.

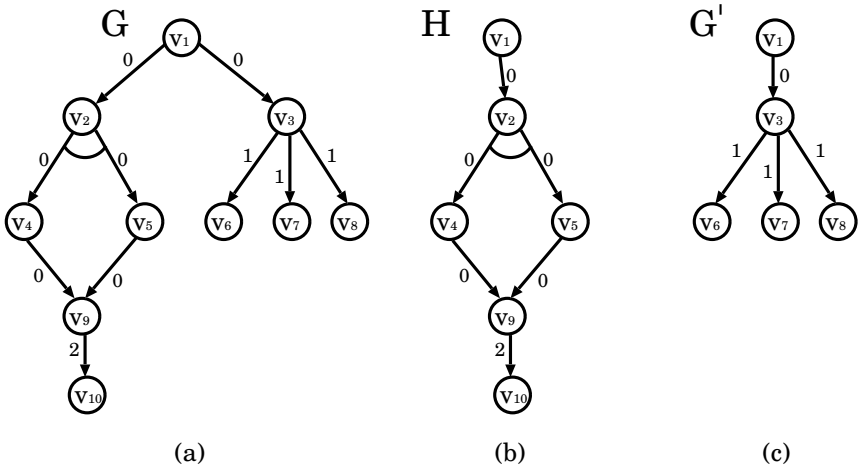

Figura 4.6: (a) Grafo E/Ou. (b) Solução ótima H. (c) Solução viável G'.

Como o Algoritmo 4.3 gera um subgrafo-solução viável para MIN–E/Ou, obtemos, em tempo polinomial, um limite superior para o problema, pois o custo do subgrafo-solução ótimo será menor ou igual ao custo do subgrafo obtido pelo algoritmo.

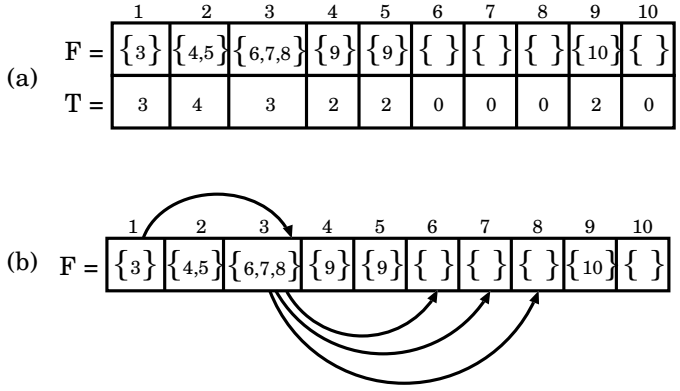

Figura 4.7: Exemplo de escalonamento de um grafo E/Ou.

Após o término deste algoritmo, se G' não é o subgrafo-solução ótimo H de G, então temos o seguinte cenário:

1. $\tau(H) < \tau(G')$, pois, caso contrário, o subgrafo H não seria ótimo.
2. $t[s(G')] < t[s(H)]$, pois, caso contrário, o subgrafo G' não minimizaria $t[s]$.
3. $\tau(G') < t[s(G')]$, pois $\tau(G')$ é obtido pelo somatório dos pesos de G'.

Logo:

$$\tau(H_{opt}(G)) \leq \tau(G') \leq t[s(G')] \leq t[s(H_{opt}(G))].$$

As desigualdades acima listadas ocorrem quando existe um subconjunto S tal que

$$S = \{e \in E(H_{opt}(G)) \mid c(e) \geq 1\},$$

de forma que $t[s(H_{opt}(G))]$ seja maior que $t[s(G')]$.

O Algoritmo 4.3 aplicado ao subgrafo-solução $H_{opt}(G)$ contabiliza mais de um vez uma aresta $e = (u, v)$ $(c(e) > 1)$ se e somente se existir mais de um caminho de s até v passando por e em $H_{opt}(G)$, pois cada um desses caminhos contabiliza $\tau(e)$ apenas uma vez.

Como $H_{opt}(G)$ é um subgrafo-solução, todo caminho distinto entre s e v é contabilizado em $t[s(H_{opt}(G))]$, pois todo vértice do tipo Ou em $H_{opt}(G)$ (não sumidouro) tem apenas um vizinho de saída, e os vértices do tipo E acumulam o peso de todos os seus vizinhos de saída. Logo, para cada aresta $e = (u, v) \in E(H_{opt}(G))$, $c(e)$ será exatamente o número de caminhos distintos entre s e v que passam por e, valor este que será não mais que o produtório do grau de saída dos vértices do tipo E existentes entre s e u (o número de vértices E entre s e u pode ser obtido aplicando-se uma busca em profundidade[7] inversa, partindo de u até a fonte s). No entanto, como o grafo H não é conhecido a priori, sabemos apenas que o número de vértices do tipo E de $H_{opt}(G)$ é menor ou igual ao número de vértices do tipo E de G. Portanto, $c(e) \leq \prod_{v \in V_E(G)} |O_v|$, $\forall\ e \in E(G)$.

Sendo assim temos:

$$\tau(H_{opt}(G)) \leq \tau(G') \leq t[s(G')] \leq t[s(H_{opt}(G))] \leq \prod_{v \in V_E} |O_v|.\tau(H_{opt}(G)).$$

Logo o algoritmo acima retorna um subgrafo-solução cujo custo é uma α-aproximação do custo do subgrafo-solução ótimo, onde $\alpha = \prod_{v \in V_E(G)} |O_v|$.

A Figura 4.8 ilustra o exemplo de um grafo E/Ou onde $c(v_{10}, v_{11}) = \alpha = \prod_{v \in V_E(G)} |O_v|$.

A Figura 4.9 ilustra um exemplo, onde $\tau(G') = \prod_{v \in V_E(G)} |O_v|.\tau(H_{opt}(G))$, sendo este

o pior caso para a aproximação.

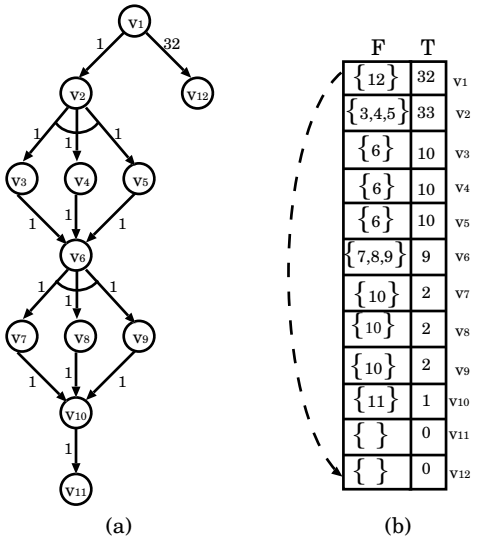

Figura 4.8: Caso onde $c(v_{10}, v_{11}) = \alpha$.

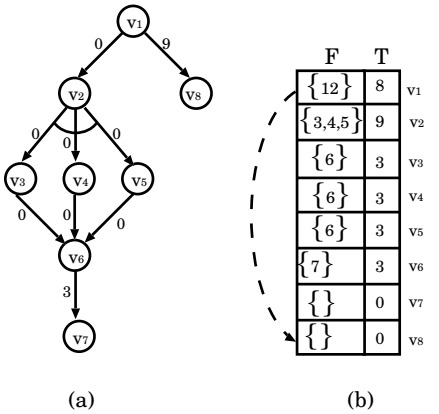

Figura 4.9: Pior caso para a aproximação do Algoritmo 4.3.

Definição 14 [10] *Um algoritmo A é dito aproximativo para um problema Π se, para toda instância viável I de Π, A devolve uma solução viável $A(I)$ de I, tal que o valor de $A(I)$ é no máximo/mínimo α vezes o valor ótimo do problema de minimização/maximização, onde α é um valor conhecido que pode depender de I.*

Definição 15 [10] *Se um problema Π admite um algoritmo aproximativo para resolvê-lo, então Π pertence à classe APX.*

Teorema 16 MIN–E/Ou $\in APX$.

Prova: Conforme visto anteriormente, o Algoritmo 4.3 devolve uma solução viável para MIN–E/Ou, cujo custo é menor ou igual a α vezes o custo mínimo, onde $\alpha = \prod_{v \in V_E(G)} |O_v|$. Como α é um valor conhecido que depende de I, por definição, o Algoritmo 4.3 é um algoritmo aproximativo, e o problema de custo mínimo para grafos E/Ou pertence à classe APX. ■

4.5 Grafos X-*de*-Y

Grafos X-*de*-Y são uma generalização de grafos E/Ou. Nestes grafos, todo vértice v_i possui um rótulo x_i-y_i, tal que v_i depende de x_i dentre os seus y_i vizinhos de saída, $0 \leq x_i \leq y_i$ e $y_i = |O_{v_i}|$. Grafos X-*de*-Y foram definidos inspirados nos modelos para detecção de *deadlocks* em sistemas distribuídos [3].

Dado um grafo E/Ou, podemos construir um grafo X-*de*-Y da seguinte forma:

- para cada vértice $v_i \in V(G)$ do tipo E, criamos em G' um vértice x_i-y_i onde $x_i = y_i$, pois um vértice do tipo E depende de todos os seus y_i vizinhos de saída;
- para cada vértice $v_i \in V(G)$ do tipo Ou, criamos em G' um vértice x_i-y_i onde $x_i = 1$, pois um vértice do tipo Ou depende de apenas 1 dentre seus y_i vizinhos de saída.

De forma análoga a grafos E/Ou, podemos formular o problema de custo para grafos X-*de*-Y. Suas formulações como problema de otimização e decisão, respectivamente, são definidas a seguir:

Versão de Otimização: MIN–X-DE-Y

Instância: Um grafo X-*de*-Y acíclico e ponderado $G = (V, E)$ com um vértice fonte $s \in V(G)$, onde cada vértice v_i possui um rótulo x_i-y_i e cada aresta e possui um peso $\tau(e) \in Z^+$.

Questão: Encontrar um subgrafo $H = (V', E')$ de G, tal que o somatório dos pesos das arestas seja mínimo, atendendo a:

1. $s \in V'$;
2. para todo $v_i \in V'$, x_i dentre suas y_i arestas de saída devem pertencer a E'.

Versão de Decisão: MIN–X-DE-Y

Instância: Um inteiro positivo k e um grafo X-*de*-Y acíclico e ponderado $G = (V, E)$ com um vértice fonte $s \in V(G)$, onde cada vértice v_i possui um rótulo x_i-y_i e cada aresta possui um $\tau(e) \in Z^+$.

Questão: Determinar se existe um subgrafo-solução $H = (V', E')$ de G, tal que o somatório dos pesos das arestas de H não exceda k, atendendo a:

1. $s \in V'$;
2. para todo $v_i \in V'$, x_i dentre suas y_i arestas de saída devem pertencer a E'.

A Figura 4.10 ilustra em (a) um grafo X-*de*-Y, e em (b) seu subgrafo-solução.

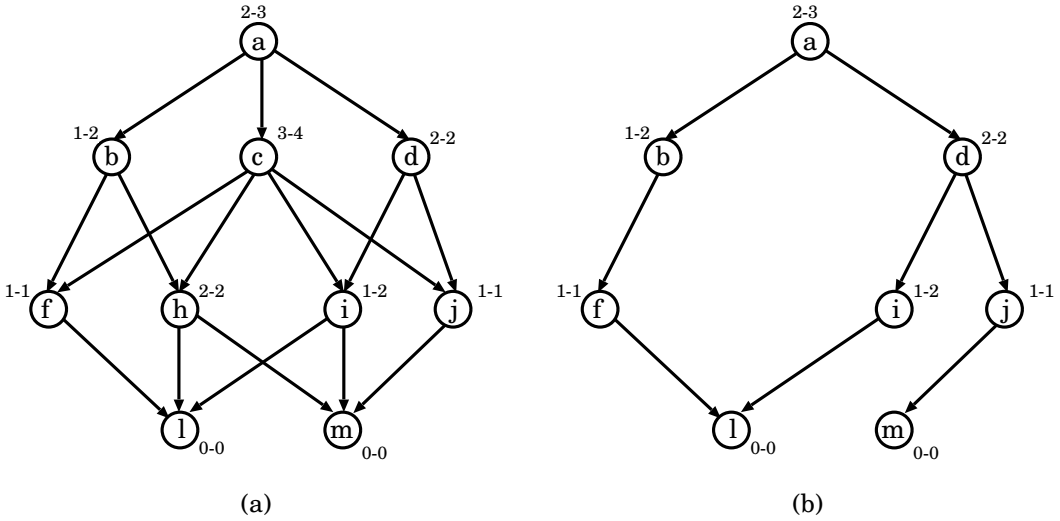

Figura 4.10: (a) Grafo X-*de*-Y. (b) Subgrafo-solução do grafo em (a).

Teorema 17 MIN–X-DE-Y *é NP-completo.*

Corolário 18 MIN–X-DE-Y $\in P$ *se* $x_i = y_i$ *para todo* $v_i \in V(G')$.

Prova: Se $x_i = y_i$ para todo $v_i \in V(G')$, então G' equivale a um grafo E (grafo onde todos os vértices são do tipo E). Logo, o próprio G' será o subgrafo-solução de custo mínimo. ■

Corolário 19 MIN–X-DE-Y $\in P$ *se* $x_i = 1$ *para todo* $v_i \in V(G')$.

Prova: Se $x_i = 1$ para todo $v_i \in V(G')$, então G' equivale a um grafo Ou (grafo onde todos os vértices são do tipo Ou). Logo, o subgrafo-solução de custo mínimo de MIN–X-DE-Y equivale ao caminho mínimo de s até um sumidouro. ■

4.5.1 Algoritmo de tempo polinomial para árvores *X-de-Y*

Quando o grafo G' é uma árvore, o custo do subgrafo-solução ótimo de G' pode ser obtido a partir da seguinte relação de recorrência:

$$custo(v_i) = \begin{cases} 0, \text{ se } v_i \text{ é um sumidouro} \\ \sum_{j=1}^{x_i} [\tau(v_i, v_{ij}) + custo(v_{ij})], \text{caso contrário} \end{cases}$$

Nas equações acima, v_{ij} é o j-ésimo vizinho de saída de v_i de menor peso ($\tau(v_i, v_{ij}) + custo(v_{ij})$).

Utilizando a recorrência acima para o problema MIN–X-DE-Y, construímos o Algoritmo 4.4 descrito a seguir, cuja complexidade é $O(n \log x)$, onde x é o tamanho do maior *heap* utilizado, como veremos na explicação da implementação.

O Algoritmo, por razões de formatação, encontra-se na próxima página.

Algoritmo 4.4: MIN–X-DE-Y para árvores

Dados: Uma árvore E/Ou $T = (V, E)$ enraizada por s.

Entrada: Um heap H_i de x_i posições inicialmente vazias, onde cada posição possui dois campos (nó e custo), para todo $v_i \in V(T)$, e uma lista $C[\,]$ de n posições inicialmente vazias.

Saída: H_i contendo os x_i vizinhos de saída de v_i que minimizam o seu peso, $C[v_i]$ contendo o custo da subárvore-solução ótima enraizada por v_i, para todo $v_i \in V(T)$.

```
início
    para cada vértice v_i ∈ V(T) faça
        C[v_j] := 0;
    Custo(s);
procedimento Custo(v_i: vértice v_i ∈ V(T))
    se O_i ≠ { } então
        para cada vértice w ∈ O_i faça
            Custo(w);
            Aux.no:=w;
            Aux.custo:=C[w] + τ(v_i, w);
            se H_i[x_i] = { } então
                C[v_i] := C[v_i] + Aux.custo;
                Inserir Aux no heap H_i;
            senão
                se H_i[1].custo > Aux.custo então
                    C[v_i] := C[v_i] − H_i[1].custo + Aux.custo;
                    H_i[1] := Aux;
                    Rearranjar o heap H_i;
    senão
        C[v_i] := 0;
```

O Algoritmo 4.4 possui uma lista sequencial C de n posições e, para todo vértice v_i, possui um heap H_i de x_i posições, inicialmente vazio. Cada heap H_i refere-se aos x_i vizinhos de saída de v_i de menor custo, sendo o primeiro elemento o de maior custo entre estes. Ao término do algoritmo, o valor de $C[v_i]$ será o custo da subárvore-solução ótima enraizada por v_i, e H_i conterá os filhos de v_i nesta árvore.

Neste algoritmo, o uso do heap H_i para armazenar os x_i vizinhos de saída de menor peso de v_i garante que, durante a execução do algoritmo, seja possível localizar e substituir em tempo $O(1)$ o vizinho armazenado de maior peso.

Para um vértice v_i obter os seus x_i vizinhos de saída de menor peso, será necessário no pior caso $O(x_i\ log\ x_i)$ passos. Este caso ococorre quando o vértice v_i analisa os seus vizinhos de saída em ordem decrescente ao peso, implicando assim para cada vizinho de saída analisado uma operação de inserção e rearranjo no heap H_i.

Como para cada vértice v_i será determinado os seus x_i vizinhos de saída, a complexidade deste algoritmo será dada por:

$$\sum_{v_i \in V(T)} (x_i\ log\ x_i) \leq \sum_{v_i \in V(T)} (x_i\ log\ x) = \sum_{v_i \in V(T)} (x_i) log\ x = (n-1)\ log\ x$$

onde $x = \max_{v_i \in V(T)} x_i$.

Portanto a complexidade de pior caso do Algoritmo 4.4 é $O(n\ log\ x)$.

5 MIN–E/OU E MIN–X-DE-Y PARAMETRIZADOS

Neste capítulo, são apresentados alguns resultados obtidos sobre as versões parametrizadas (MIN–E/Ou(k), MIN–E/Ou(k, r) e MIN–X-DE-Y(k)) dos problemas NP-completos: MIN–E/Ou e MIN–X-DE-Y.

Problema: MIN–E/Ou(k)

Parâmetro: Um inteiro positivo k.

Instância: Um grafo E/Ou acíclico, conexo e ponderado $G = (V, E)$ com um vértice fonte $s \in V$, e tal que cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$ e cada aresta e possui um peso inteiro $\tau(e) \geq 1$.

Questão: Determinar se existe um subgrafo-solução $H = (V', E')$ de G tal que o somatório dos pesos das arestas de H não exceda k, atendendo a:

1. $s \in V'$;
2. se $v \in V'$ e $f(v) = E$, então todas as arestas de saída de v devem pertencer a E';
3. se $v \in V'$ e $f(v) = Ou$, então exatamente uma das arestas de saída de v deve pertencer a E'.

Problema: MIN–E/Ou(k, r)

Parâmetros: Dois inteiros positivos k e r.

Instância: Um grafo E/Ou acíclico, conexo e ponderado $G = (V, E)$ com um vértice fonte $s \in V(G)$, e tal que cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$, cada aresta e possui um peso inteiro $\tau(e) \geq 1$, e cada vértice com rótulo Ou possui no máximo r arestas de saída de mesmo peso.

Questão: Determinar se existe um subgrafo-solução $H = (V', E')$ de G tal que o somatório dos pesos das arestas de H não exceda k, atendendo a:

1. $s \in V'$;
2. se $v \in V'$ e $f(v) = E$, então todas as arestas de saída de v devem pertencer a E';
3. se $v \in V'$ e $f(v) = Ou$, então exatamente uma das arestas de saída de v deve pertencer a E'.

Problema: MIN–X-de-Y(k)

Parâmetro: Um inteiro positivo k.

Instância: Um grafo X-*de*-Y acíclico e ponderado $G = (V, E)$ com um vértice fonte $s \in V(G)$, e tal que cada vértice v possui um rótulo $f(v) \in \{E, Ou\}$ e cada aresta possui um peso inteiro $\tau(e) \geq 1$.

Questão: Determinar se existe um subgrafo-solução $H = (V', E')$ de G tal que o somatório dos pesos das arestas de H não exceda k, atendendo a:

1. $s \in V'$;
2. para todo $v_i \in V'$, x_i dentre suas y_i arestas de saída devem pertencer a E'.

5.1 O Problema MIN–E/Ou(k, r)

Neste trabalho, formulamos a variação MIN–E/Ou(k, r) do problema MIN–E/Ou(k). Para MIN–E/Ou(k, r), o parâmetro adicional r representa o número máximo de arestas de saída de pesos idênticos em um vértice do tipo *Ou*.

É importante observar que o problema MIN–E/Ou(k, r) equivale ao problema MIN–E/Ou(k), quando r é suficientemente grande, por exemplo, $r = \Delta(G)$ (grau máximo).

Lema 20 MIN–E/Ou *é NP-completo, mesmo fixando* $r = 3$ *para o número máximo de arestas de saída de um vértice do tipo Ou.*

Prova: A corretude deste lema pode ser verificada atráves da prova de NP-completude da Seção 4.1, onde toda instância do problema 3-Sat é transformada em um grafo E/Ou, e todo vértice *Ou* tem grau de saída menor ou igual a 3. ■

Teorema 21 *Seja* G *uma instância do problema* MIN–E/Ou(k, r). *É possível obter um núcleo de tamanho* $O((kr)^{k+1})$ *em tempo* $O(n + m)$.

Prova: Como prova, é apresentado um conjunto de regras de redução corretas e suficientes que podem ser aplicadas à instância de entrada, em tempo $O(n + m)$, construindo um núcleo de tamanho menor ou igual a $(kr)^{k+1}$:

Regra 1 Para todo vértice v_i do tipo E, se $\sum_{v_j \in O_i} \tau(v_i, v_j) > k$, então remova v_i.

Regra 2 Para toda aresta $e \in E(G)$, se $\tau(e) > k$, então remova-a.

Regra 3 Para todo vértice $v_i \neq s$, se o peso do caminho mínimo de s até v_i for maior que k, então remova-o.

Regra 4 Se algum vértice tornou-se desconexo de s, então remova-o.

Regra 5 Se algum vértice inicialmente não sumidouro tornou-se sumidouro, então atribua peso $k+1$ a todas suas arestas de entrada.

Regra 6 Se um nó E teve um filho removido, então atribua peso $k+1$ a uma de suas arestas de saída.

Regra 7 Todo vértice com grau de saída menor ou igual a 1 é rotulado como Ou.

Seja H o grafo resultante da aplicação das regras de redução.

A Regra 1 retira do grafo todos os vértices E tais que o somatório dos pesos de suas arestas de saída é maior do que k. Isto se deve ao fato de estes vértices não poderem fazer parte de um subgrafo-solução de custo máximo k. Assim, todo vértice E de H, após a aplicação da Regra 1, terá no máximo k arestas de saída. Em seguida, a Regra 2 é aplicada, retirando do grafo todas as arestas de saída de peso maior que k, pois estas arestas não poderão fazer parte de um subgrafo-solução de custo máximo k. Devido à aplicação da Regra 1, é fácil ver que a Regra 2 é aplicada apenas a arestas de saída de vértices do tipo Ou. Após a aplicação das Regras 1 e 2, H terá k pesos distintos de arestas. Logo, todo vértice Ou terá no máximo kr arestas de saída. A Regra 3, por sua vez, retira do grafo todo vértice v_i tal que o custo de um caminho mínimo de s até v_i seja maior do que k, pois se v_i pertencesse a um subgrafo-solução, então um caminho de s até v_i deveria existir neste subgrafo e, consequentemente, este subgrafo teria peso maior do que k.

Após a aplicação das Regras 1, 2 e 3, se algum vértice v_i tornou-se desconexo de s, isto significa que todo caminho de s a v_i passava por um vértice ou aresta que não poderia fazer parte de um subgrafo-solução de custo máximo k, logo v_i deverá ser removido, o que é feito pela Regra 4. Se algum vértice v_i inicialmente não

sumidouro tornou-se sumidouro após a aplicação das Regras 1, 2, 3 e 4, isto significa que nenhuma dependência de v_i pode fazer parte de um subgrafo-solução de custo máximo k. Assim, v_i também não poderá fazer parte de um subgrafo-solução de G de custo máximo k. A Regra 5 atribui peso $k+1$ a toda aresta de entrada de v_i, impedindo assim que v_i faça parte de um subgrafo-solução de custo máximo k em H (este vértice não é removido, para evitar a criação de novos sumidouros). A Regra 6 efetua processo semelhante ao da Regra 5, para vértices do tipo E, e a Regra 7 transforma todo vértice E com grau de saída menor ou igual a 1 em um vértice do tipo Ou, pois neste caso a alteração do rótulo não altera a relação de dependência dos vértices.

Conforme podemos observar, as regras de redução alteraram e/ou removeram somente vértices e arestas que não poderiam fazer parte de um subgrafo-solução de custo máximo k. Desta forma, se o grafo H possui um subgrafo-solução de custo máximo k, este também será um subgrafo-solução de custo máximo k para G.

A aplicação das regras de redução demanda tempo $O(n+m)$, o que pode ser verificado observando-se cada regra isoladamente. A Regra 1 pode ser executada em tempo $O(m)$, pois será necessário percorrer todos os vértices com rótulo E e suas arestas. A Regra 2 demandará tempo $O(m)$, uma vez que todas as arestas devem ser analisadas. A Regra 3 poderá ser executada em tempo $O(n+m)$, a partir do algoritmo *DAG-Shortest-Paths* [7], pois G é um grafo orientado acíclico. A Regra 4 também poderá ser executada em tempo $O(n+m)$ através de um percurso genérico. A Regra 5 demandará tempo $O(n+m)$, pois no pior caso, para cada sumidouro, será necessário alterar o peso de todas as suas arestas de entrada. A Regra 6 demandará tempo $O(n)$. Como nenhuma regra tem complexidade superior a $O(n+m)$ e o grafo é conexo, as regras de redução podem ser aplicadas em tempo $O(n+m)$. Por fim, a Regra 7, pode ser executada em tempo $O(n)$.

No grafo resultante H, o maior caminho mínimo de s até um sumidouro possui peso máximo k, e cada vértice possui no máximo kr vizinhos de saída. Desta forma, H terá número máximo de vértices se: (i) todos os vértices não sumidouros possuirem grau de saída igual a kr ; (ii) não houver vértices com vizinhos de saída em comum; (iii) o custo de um caminho mínimo de s a qualquer sumidouro for k. Em outras palavras, H terá número máximo de vértices quando for uma árvore (kr)-ária cheia de altura $k+1$. Portanto, o número de vértices de H é $O((kr)^{k+1})$.

Como (i) as regras de redução podem ser aplicadas em tempo polinomial, (ii) o tamanho do grafo resultante H está em função dos parâmetros k e r, e (iii) um subgrafo-solução de custo máximo k' em H equivale a um subgrafo-solução de custo máximo k' em G, concluímos que o subgrafo H é um núcleo para o problema MIN–E/OU(k,r). ■

Corolário 22 MIN–E/OU$(k,r) \in FPT$.

Prova: O resultado segue diretamente do Teorema 3 e do Teorema 21. ■

5.2 O Problema MIN–X-DE-Y(k)

Nesta seção, é demonstrado que o problema MIN–X-DE-Y(k) é W[1]-difícil, sendo portanto, aparentemente intratável.

Para mostrarmos a W[1]-dificuldade do problema MIN–X-DE-Y(k), iremos exibir uma transformação paramétrica do problema W[1]-completo CLIQUE(k) para o problema MIN–X-DE-Y(k).

Teorema 23 CLIQUE(k) *é transformado parametricamente em* MIN–X-DE-Y(k).

Prova: Dada uma instância G do problema CLIQUE(k), podemos transformá-la em uma instância G' de MIN–X-DE-Y(k') da seguinte forma:

Seja G' um grafo inicialmente vazio.

1. Criar um vértice fonte s;
2. Criar um conjunto de vértices $\{u_1, u_2, ..., u_n\}$, vizinhos de saída de s (o vértice u_i de G' está associado ao vértice v_i de G);
3. Para cada vértice u_i, são criados dois filhos: z_i e w_i;
4. Para cada vértice z_i, é criada uma aresta (z_i, w_j) se e somente se v_j for vizinho de v_i em G;
5. Para cada vértice w_i, é criado um vizinho de saída sumidouro t_i.

Os pesos das arestas serão dados da seguinte forma:

- Se $d(v_i) < k - 1$, então $\tau(s, u_i) = k^2 + 3k + 1$
- Se $d(v_i) \geq k - 1$, então $\tau(s, u_i) = 1$
- Todas as demais arestas terão peso 1.

A rotulação dos vértices será dada da seguinte maneira:

1. s terá rótulo k-n, onde $n = |V(G)|$.
2. Todo vértice u_i terá rótulo 2-2.

3. Todo vértice w_i terá rótulo 1-1.

4. Todo vértice t_i terá rótulo 0-0.

5. Para cada vértice z_i, se $d(v_i) \geq k-1$, então z_i é rotulado com $(k-1)$-$d(v_i)$

6. Para cada vértice z_i, se $d(v_i) < k-1$, então z_i é rotulado com $d(v_i)$-$d(v_i)$

Considerando que a criação de cada vértice e aresta demanda tempo $O(1)$, a construção acima poderá ser executada em tempo $O(n+m)$, uma vez que será necessário apenas para cada vértice de G percorrer todas as suas arestas. Logo, a transformação é feita em tempo $f(k).n^{\alpha}$.

A Figura 5.1 ilustra um grafo G e seu respectivo grafo G' obtido através da transformação acima, onde $k = 3$. Neste grafo, a aresta (s, u_1) possui peso $k^2 + 3k + 1 = 19$ e as demais arestas peso 1.

Como podemos observar, o grafo G foi transformado em um grafo G', onde $n' = 4n + 1$. Será demonstrado a seguir que G possui uma clique de tamanho k se e somente se G' possui um subgrafo-solução de custo máximo $k' = k^2 + 3k$.

($\Rightarrow$) Se G tem um conjunto de vértices $\{v_1, v_2, ..., v_k\}$ que formam uma clique C, então podemos construir um subgrafo-solução H de G' da seguinte forma: como s é um vértice de rótulo k-n, escolhemos $\{u_1, u_2, ..., u_k\}$ como os vizinhos de saída de s em H. Como cada um desses vértices tem rótulo 2-2, os vértices $\{w_1, w_2, ..., w_k\}$ e $\{z_1, z_2, ..., z_k\}$ também farão parte do subgrafo-solução H. Como cada vértice w_i é um vértice 1-1, os vértices $\{t_1, t_2, ..., t_k\}$ também farão parte do subgrafo H. Neste ponto, H já tem custo $4k$, pois cada uma de suas arestas tem peso 1. Como cada vértice z_i depende de $k-1$ vizinhos de saída, escolhemos para z_i um vértice w_j se e somente se v_j pertence à clique C em G (as arestas de saída dos vértices $z_1, z_2, \ldots, z_k$

adicionam custo $k(k-1)$ a H). Como cada vértice z_i escolheu somente vértices w_j que já se encontravam em H, o custo de H será $k(k-1)+4k = k^2+3k$.

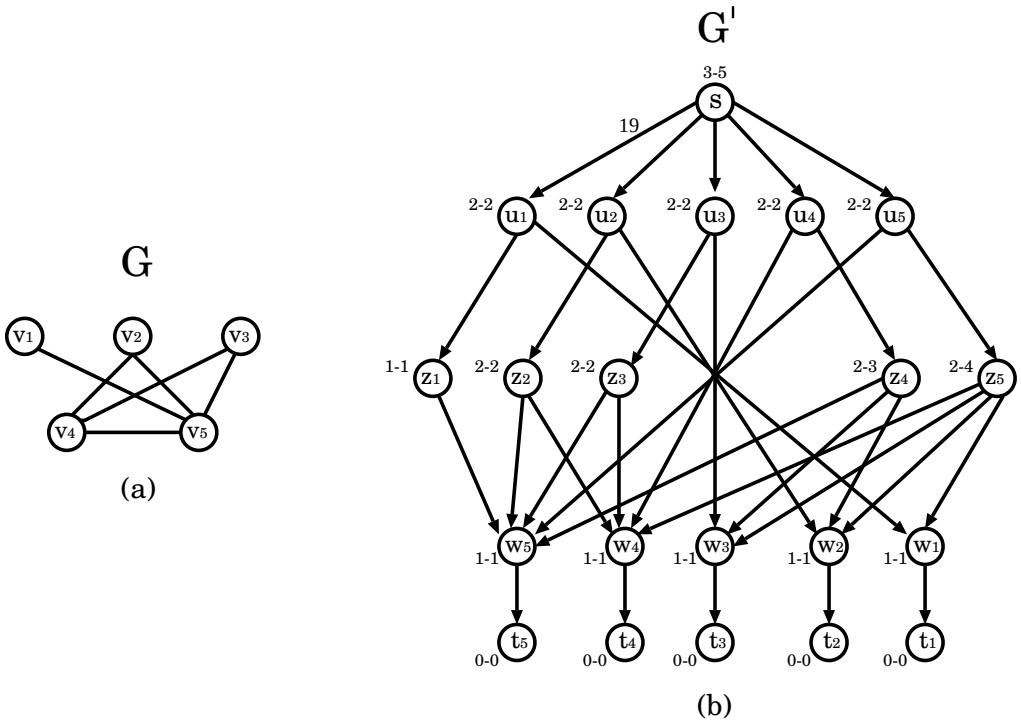

Figura 5.1: Transformação paramétrica de G em G'.

($\Leftarrow$) Suponha, por contradição, que G' tem um subgrafo-solução ótimo H de custo máximo $k' = k^2+3k$, e G não possui uma clique de tamanho k. Então, o subgrafo H possui pelo menos $k+1$ vértices w_j, pois cada um dos k vértices u_i terá um vizinho de saída w_i. Como G não possui uma clique de tamanho k, então pelo menos um vértice z_i escolheu um vértice w_j novo, pois caso contrário, G teria tal clique. No entanto, se G' possui $k+1$ vértices, temos em H que: (i) s possui k vizinhos, e portanto as arestas de saída de s adicionam custo k a H; (ii) cada um dos k vértices u_i possui 2 vizinhos de saída, o que adiciona custo $2k$ a H; cada um dos k vértices z_i possui $k-1$ vizinhos de saída, adicionando portanto custo $k(k-1)$ a H; (iv) cada um dos $k+1$ vértices w_j possui 1 vizinho de saída, o que adiciona custo $k+1$ a H.

Sendo assim, temos no total $\tau(H) = k + 2k + k(k-1) + (k+1) = k^2 + 3k + 1$. Uma contradição, pois foi assumido que H tem custo máximo $k' = k^2 + 3k$.

Logo, G possui uma clique de tamanho k se e somente se G' possui um subgrafo-solução de custo máximo k'. ■

Corolário 24 MIN–X-DE-Y(k) *é W[1]-difícil.*

Segue diretamente do Lema 9 e do Teorema 23. ■

A Figura 5.2 ilustra a clique de tamanho $k = 3$ e o subgrafo-solução de custo $k' = 18$, para os grafos G e G' apresentados na Figura 5.1.

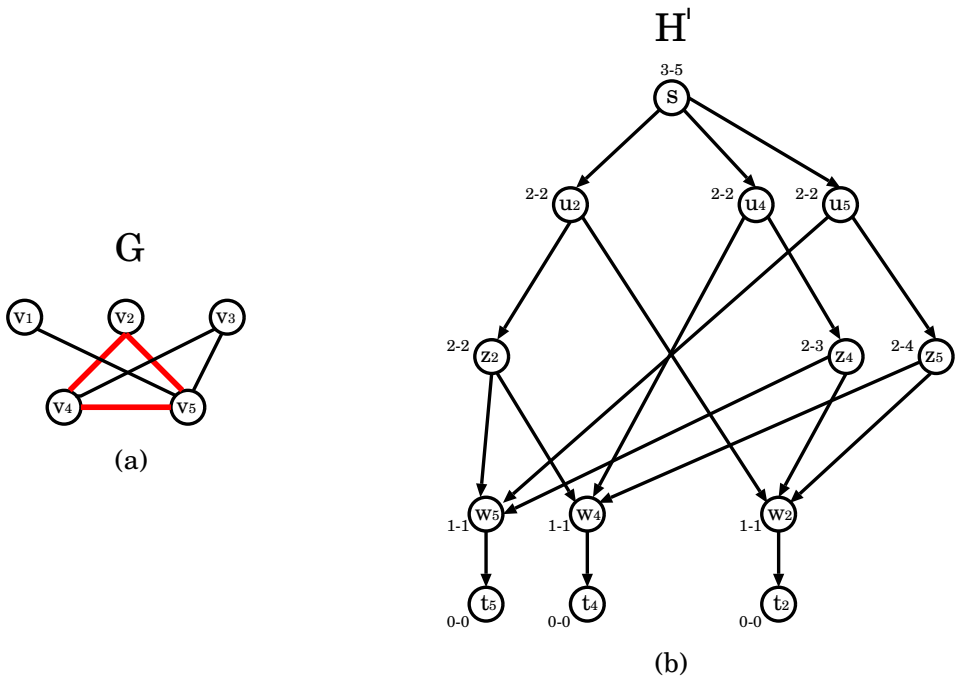

Figura 5.2: (a) Clique de tamanho 3. (b) Subgrafo-solução de custo 18 em G'.

6 CONSIDERAÇÕES FINAIS

Grafos E/Ou são estruturas de dados que possibilitam a modelagem de diversos problemas reais, de forma que soluções para estes problemas possam ser buscadas computacionalmente. Grafos X-*de*-Y, por sua vez, são estruturas que generalizam os grafos E/Ou, capazes de modelar mais facilmente uma gama maior de problemas.

Dentre os diversos problemas relativos a grafos E/Ou e grafos X-*de*-Y abordados nesta dissertação, foi dada uma atenção especial aos problemas MIN–E/Ou e MIN–X-DE-Y, uma vez que um subgrafo-solução pode ser interpretado como um roteiro a ser seguido para a solução de um problema. Apresentamos nesta dissertação, no Capítulo 2, algumas das aplicações de grafos E/Ou, também desenvolvemos um algoritmo de complexidade $O(m)$ para escalonamento de grafos E/Ou.

Estudos sobre a complexidade do problema MIN–E/Ou têm sido feitos desde a década de 70, quando Sahni apresentou uma prova de NP-completude para MIN–E/Ou. No entanto, poucos avanços haviam sido feitos, no que diz respeito a novas abordagens e/ou heurísticas para esses problemas.

Nesta dissertação, no Capítulo 3, foram mostrados casos em que os problemas MIN–

E/Ou e MIN–X-de-Y tornam-se polinomiais (por exemplo, quando os grafos são árvores). Algoritmos para estes casos foram apresentados, além de um algoritmo aproximativo e um algoritmo que retorna um limite inferior para o problema MIN–E/Ou. A prova que o problema MIN–X-de-Y é polinomial para árvores é um dos resultados obtidos nesta dissertação, o algoritmo aproximativo e o algoritmo de limite inferior também são contribuições desta dissertação.

Sob o ponto de vista da complexidade parametrizada, provamos, no Capítulo 4, que o problema MIN–E/Ou(k, r) é FPT, quando fixado um parâmetro k para o custo e um parâmetro r para o número de arestas de saída de mesmo peso para os vértices Ou. Também provamos que o problema MIN–X-de-Y(k) é W[1]-difícil, quando fixado o parâmetro k para o custo. Sendo estes os resultados mais expressivos que foram obtidos nesta dissertação.

Abaixo apresentamos uma tabela de resultados, em preto encontram-se os resultados existentes na literatura e em azul os resultados obtidos nesta dissertação.

	Polinomial	**NP-completo**	**FPT**	**W[1]-difícil**
MIN–E/Ou	árvores	X	—	—
MIN–X-de-Y	**árvores**	X	—	—
MIN–E/Ou(k, r)	—	—	**X**	—
MIN–X-de-Y(k)	—	—	—	**X**

Tabela 6.1: Resultados existentes e obtidos

6.1 Trabalhos Futuros

Em trabalhos futuros, pretendemos provar que o problema MIN–E/Ou(k) é tratável por parâmetro fixo, utilizando as regras de redução definidas na Seção 5.1 e o Algoritmo 4.3 apresentado na Seção 4.4. Quando aplicamos ao grafo de entrada este

conjunto de regras de redução e em seguida o algoritmo aproximativo, obtemos uma solução $k^{k/2}$-aproximada da solução ótima. Aplicando esse raciocínio iterativamente, acreditamos ser possível obter em tempo $f(k).n^{\alpha}$ uma resposta sim ou não para a instância original, o que implicaria na pertinência do problema à classe FPT.

Uma outra proposta de continuidade deste trabalho seria classificar o problema MIN–X-DE-Y(k) dentro da W-hierarquia.

REFERÊNCIAS

[1] ADELSON-VELSKY, G. M.; GELBUKH, A. F.; LEVNER, E. On fast path-finding algorithms in and-or graphs. **Mathematical Problems in Engineering**, v. 8, n. 4, p. 283–293, jun. 2002.

[2] ARENALES, M. N.; VIANNA, A. C. G. O problema de corte de placas defeituosas. **Pesquisa Operacional**, v. 26, n. 2, p. 185–202, ago. 2006.

[3] BARBOSA, V. C. The combinatorics of resource sharing. In: FIALLES, M.; GOMES, F. (Ed.). **Models for Parallel and Distributed Computation: Theory, Algorithmic Techniques and Applications**. The Netherlands: Kluwer Academic Publishers, 2002. p. 27–50.

[4] CERIOLI, M. R.; CARVALHO, M. H.; DAHAB, R.; FEOFILOFF, P.; GUIMARÃES, K. S.; MIYAZAWA, F. K.; JR, J. P.; SOARES, J. A. R.; WAKABAYASHI., Y. Uma introdução sucinta a algoritmos de aproximação. In: **Colóquio Brasileiro de Matemática**, 26, 2007. Rio de Janeiro, Anais... Rio de Janeiro, IMPA: [s.n.], 2007.

[5] CESATI, M. **Compendium of Parameterized Problems**. [S.l.], Set 2006. Disponível em: http://bravo.ce.uniroma2.it/home/cesati/research/compendium. Acesso em Maio 17, 2010.

[6] CORANDI, R.; WESTFECHTEL, B. Version models for software configuration management. **ACM Computing Surveys**, v. 30, n. 2, p. 332–282, jun. 1998.

[7] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos de grafos. In: **Algoritmos**: Teoria e Prática. Rio de Janeiro: Campus, 2002. p. 468–470.

[8] DOWNEY, R.; FELLOWS, M. Fixed-parameter tractability and completeness I: Basic results. **Siam Journal on Computing**, v. 24, n. 4, p. 873–921, ago. 1995.

[9] DOWNEY, R.; FELLOWS, M. Fixed-parameter tractability and completeness II: On completeness for W[1]. **Theoretical Computer Science**, v. 141, p. 109–131, abr. 1995.

[10] FIGUEIREDO, C. M. H.; FONSECA, G. D.; LEMOS, M. J. M. S.; Sá, V. G. P. de. Introdução aos algoritmos randomizados. In: **Colóquio Brasileiro de Matemática**, 23, 2001. Rio de Janeiro, Anais... Rio de Janeiro, IMPA: [s.n.], 2001.

[11] GALLO, G.; LONGO, G.; NGUYEN, S.; PALLOTTINO, S. Directed hypergraphs and applications. **Discrete Applied Mathematics**, v. 42, p. 177–201, abr. 1993.

[12] GAREY, M.; JOHNSON, D. **Computers and Intractability- A Guide to the Theory of NP-Completeness**. United States of America: Bell Telephone Laboratories Incorporated, 1979.

[13] LABER, E. S. A randomized competitive algorithm for evaluating priced. **Theorical Computer Science**, v. 401, n. 1, p. 120–130, jul. 2008.

[14] MORABITO, R.; PUREZA, V. Geração de padrões de cortes bidimensionais guilhotinados restritos via programação dinâmica e busca em grafo-e/ou. **Produção, São Paulo**, v. 17, n. 1, p. 033–051, abr. 2007.

[15] NILSSON, N. J. Problem-reduction representations. In: DOJNY, R. F.; EAKINS, M. (Ed.). **Problem Solving Methods in Artificial Intelligence**. United States of America: McGraw-Hill, 1971. p. 80–112.

[16] RIVEST, R. L. Game tree searching by min/max approximation. **Artificial Intelligence**, v. 34, n. 1, p. 77–96, 1988.

[17] SAHNI, S. Computationally related problems. **Siam Journal on Computing**, v. 3, n. 4, p. 262–279, 1974.

[18] SILVA, M. D. da. **Tratabilidade parametrizada de problemas NP-completos**. Dissertação (Mestrado em Informática) — Instituto de Matemática, NCE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2004.

[19] SILVA, M. D. da. **Complexidade Parametrizada de Problemas de Edição de Grupos**. Tese (Doutorado em Engenharia de Sistemas e Computação) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

[20] SIMON, R.; LEE, R. C. T. On the optimal solution of and/or series parallel graphs. **Journal of Association for Computing Machinery**, v. 18, n. 3, p. 354–372, jul. 1971.

[21] BOAVENTURA-NETTO, P. O.; JURKIEWICZ, S.; MALAJOVICH, B.; SOUZA, C. S. de; SOUZA, U. dos S. Roteamento Ótimo das linhas de Ônibus internas do campus UFRJ-Fundão. In: **Simpósio Brasileiro de Pesquisa Operacional**, 42 SBPO, 2010. Bento Gonçalves, Anais... Bento Gonçalves: [s.n.], 2010.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA
CCMN - Bloco C - Cidade Universitária - Ilha do Fundão
Rio de Janeiro - RJ CEP: 21941-916
www.ppgi.ufrj.br