PROPRIEDADES E ALGORITMOS PARA ESPECIALIZAÇÕES DE HIPERGRAFOS ORIENTADOS

JOÍSA DE SOUZA OLIVEIRA

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Lilian Mar

Prof.ª Lilian Markenzon, D.Sc.
(Presidente)

Sheila Regina Murgel Veloso

Prof.ª Sheila Regina Murgel Veloso, D.Sc.

Nair Maria Maia de Abreu

Prof.ª Nair Maria Maia de Abreu, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1994

OLIVEIRA, JOÍSA DE SOUZA

Propriedades e Algoritmos para Especializações de Hipergrafos Orientados
[Rio de Janeiro], 1994
vi, 113 p., 29,7cm. (COPPE/UFRJ, M.Sc.,
Engenharia de Sistemas e Computação, 1994)
Tese - Universidade Federal do Rio de Janeiro
1. Hipergrafos Orientados
I. COPPE/UFRJ II. Título (série).

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

Propriedades e Algoritmos para Especializações de Hipergrafos Orientados

Joísa de Souza Oliveira
abril de 1994

Orientadora: Lilian Markenzon
Programa: Engenharia de Sistemas e Computação

O objeto de estudo desse trabalho é uma classe particular de hipergrafos orientados, a 2-grafos orientados. Dá-se destaque a uma subclasse da mesma, a BF-grafos. As classes 2-grafos e BF-grafos são utilizadas na modelagem de problemas que aparecem em Ciência da Computação e Pesquisa Operacional, como os grafos E/OU em Inteligência Artificial e os FD-grafos em Banco de Dados Relacional.

São estudados e analisados esquemas de representação para 2-grafos, a partir das representações mais utilizadas para grafos. Será descrito um conjunto de operações mais freqüentes, a ser utilizado na avaliação comparativa das representações. Objetiva-se a escolha de uma estrutura que permita que essas operações sejam feitas com a maior eficiência possível.

Definições e resultados para casos particulares encontrados no estudo de algumas aplicações estão relacionados.

Ao final, é feita uma análise sobre a conceituação e obtenção de caminhos (hipercaminhos) e árvores em BF-grafos.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

Properties and Algorithms for Restrictions of Directed Hypergraphs

Joísa de Souza Oliveira
April, 1994

Thesis Supervisor: Lilian Markenzon
Department : Systems Engineering and Computing

The object of study of this work is a particular class of directed hypergraphs, the directed 2-graphs. Special emphasis is given to one of its subclasses, the BF-graphs. The 2-graphs and BF-graphs are used in the modelling of problems that appear in Computer Science and Operational Research, such as the AND/OR graphs in Artificial Intelligence and the FD-graphs in relational data bases.

Schemes of representation for the 2-graphs are studied and analyzed from the most used representations for graphs. A basic set of the most frequent operations is presented to be used in the comparative evaluation of the representations. The purpose of this is to choose a structure that allows those operations to have the most efficient performance possible.

Definitions and results for particular cases found in the study of some of the aplications are presented.

Finally, an analysis of the conception and the obtention of paths(hyperpaths) and trees in BF-graphs is done.

Índice

I Introdução 1

I.1 Apresentação 1

I.2 Noções Básicas Sobre Grafos 2

I.3 Conceitos Fundamentais de Hipergrafos Orientados 3

I.3.1 As classes 2-grafos e BF-grafos 3

I.3.2 Caminho e Ciclo em BF-Grafos 5

I.3.3 Hipercaminhos em BF-grafos 6

II Representações para 2-grafos 9

II.1 Um conjunto básico de operações 10

II.2 Representações tradicionais para grafos 11

II.2.1 Matriz de Adjacência 11

II.2.2 Listas de Adjacência 12

II.2.3 Alternativas 13

II.2.4 A Matriz de Incidência 16

II.2.5 A Matriz de Incidência Modificada 22

II.3 Conclusões 29

III Representações com as arestas explícitas 31

III.1 Estruturas para Hiper-Arestas 31

III.2 Listas de Incidência BS(v) e FS(v) 41

III.3 As Quatro Listas: Cabeça(e), Rabo(e), BS(v) e FS(v) 49

III.4 O Digrafo D(H) 55

III.5 Conclusões 62

IV Casos Particulares de 2-Grafos 68

IV.1 Introdução 68

IV.2 Hipergrafos Orientados Segundo Ausiello 69
IV.2.1 Principais Definições 70
IV.2.2 FD-grafos 73
IV.2.3 Minimalidade e Equivalência Forte 77
IV.2.4 Correspondência de Terminologias e Extensões 80
IV.3 Manutenção dinâmica de hipergrafos orientados 81
IV.3.1 Fechamento de Nós simples 82
IV.3.2 Nós simples: manutenção dinâmica do fechamento 84
IV.4 Representação utilizando nós compostos para casos particulares de 2-grafos 87
IV.5 Grafos E/Ou e Grafos E/Ou Generalizados 89
IV.5.1 Grafos E/Ou 90
IV.5.2 Grafos E/Ou Generalizados 90
IV.5.3 GEOs e GEOGs como Hipergrafos 92
IV.6 Arborescência de Woeginger 93
IV.6.1 Arborescência 93

V Caminhos, Árvores e Visitas em 2-Grafos Orientados **99**
V.1 Introdução 99
V.2 B-Caminho e B-visita 99
V.3 B-árvore 106
V.4 BF-caminho 108

VI Conclusões **111**

Capítulo I

Introdução

I.1 Apresentação

Hipergrafos são uma generalização do conceito de grafos e possuem um numeroso conjunto de aplicações nas mais variadas áreas de Ciência da Computação e Otimização Combinatória. O objeto de estudo desse trabalho é uma classe particular de hipergrafos orientados, a classe dos 2-grafos, com enfoque sendo dado a uma subclasse da mesma, a BF-grafos. Para esta classe, serão considerados e analisados problemas como representação interna, buscas e definição e obtenção de caminhos e árvores.

A motivação para o estudo da 2-grafos e da BF-grafos é a sua utilização na modelagem de diversos problemas em Ciência da Computação e em Pesquisa Operacional, às vezes com nomes diferentes. Eles aparecem na representação de dependências funcionais (FD-grafos) em Banco de Dados Relacional ([Ausiello 86], [Ausiello 90], [Date 91]) , como os grafos E/Ou em Inteligência Artificial ([Nilsson 80], [Rich88], [Levi 76]) ou na descrição do comportamento de sistemas concorrentes ([Markenzon 91]), por exemplo.

Nesse primeiro capítulo são apresentados os conceitos fundamentais de hipergrafos orientados, precedidos de algumas noções básicas de grafos necessárias ao estabelecimento da terminologia adotada.

Nos dois capítulos seguintes são estudados e analisados esquemas de representação para a classe 2-grafos. No capítulo II é apresentado o conceito de representação de um hipergrafo, após o que é descrito um conjunto básico de operações a ser utilizado na avaliação comparativa dos diferentes esquemas. Nesse capítulo são consideradas extensões das mais tradicionais representações para grafos.

No capítulo III são analisados esquemas de representação em que existem estruturas especiais para as arestas. Em ambos os capítulos, para cada representação são

apresentados definição, exemplos e os algoritmos para o conjunto de operações, com as respectivas complexidades.

No capítulo IV é feito um estudo de alguns casos particulares encontrados na literatura, como os hipergrafos dos trabalhos de Ausiello ([Ausiello 83], [Ausiello 85], [Ausiello86], [Ausiello 90]) e de Woeginger ([Woeginger 92]) e os grafos E/OU em Inteligência Artificial ([Nilsson 80], [Rich88], [Levi 76]).

No capítulo V é feita uma análise sobre conceituação e obtenção de caminhos, ciclos e árvores, com base nos trabalhos de Gallo ([Gallo 90][Gallo 93]) e Markenzon ([Markenzon 91]).

I.2 Noções Básicas Sobre Grafos

Um **grafo** é um par $G(V, E)$, onde V é um conjunto finito não vazio de elementos denominados **vértices** ou **nós** e E é um conjunto de pares não ordenados de elementos distintos de V, chamados **arestas**.

Um **grafo direcionado** (ou **digrafo**) é um par $D(V, E)$, onde V é um conjunto finito não vazio (os vértices) e E um conjunto de pares ordenados de vértices distintos (as arestas).

Seja $D(V, E)$ um digrafo. Dada uma aresta $(v, w) \in E$, o vértice v é dito **predecessor** de w e w é o **sucessor** de v.

Seja $D(V, E)$ um digrafo. Um **caminho** de um vértice v_1 a um vértice v_k é uma sequência de vértices $v_1, ..., v_k$ tal que $(v_j, v_{j+1}) \in E$, $1 \leq j \leq k-1$. O valor $k-1$ é o comprimento do caminho. Diz-se nesse caso que v_1 **alcança** v_k. Um caminho é dito simples se todas as suas arestas forem distintas. Se todos os vértices de um caminho simples forem distintos, ele é dito elementar. Um **ciclo** é um caminho $v_1, ..., v_k$ tal que $v_1 = v_k$. Um digrafo sem ciclos é chamado acíclico.

Seja S um conjunto e $S' \subseteq S$. Diz-se que S' é **minimal** (maximal) em relação a uma certa propriedade P quando S' satisfaz à propriedade P e não existe subconjunto $S'' \subseteq S'$ ($S'' \supseteq S'$) que também satisfaz P.

Seja $D(V, E)$ um digrafo acíclico. Denomina-se **fechamento transitivo** de D ao maior digrafo $D_f(V, E_f)$ que preserva a alcançabilidade de D. Isto é, para todo $v, w \in V$, se v alcança w em D então $(v, w) \in E_f$. De forma análoga, a **redução transitiva** de D é o menor digrafo $D_r(V, E_r)$ que preserva a alcançabilidade de D.

Um grafo é denominado **rotulado** em vértices (ou arestas) quando a cada vértice (ou aresta) estiver associado um identificador, denominado rótulo.

I.3 Conceitos Fundamentais de Hipergrafos Orientados

As definições adotadas nesse trabalho são as encontradas em [Gallo 93]. Um **hipergrafo** é um par $H(V, E)$, onde V é o conjunto de vértices e $E = \{e_1, ..., e_m\}$, $e_i \subseteq V$ para $i = 1, ..., m$, é o conjunto das hiper-arestas.

Nesse trabalho somente hipergrafos orientados são considerados, ou seja, hipergrafos em que todas as hiper-arestas sejam orientadas.

Uma **hiper-aresta orientada** ou **hiperarco** $e = (L_1, L_2, ..., L_r)$ é uma sequência ordenada de r subconjuntos de vértices não vazios e disjuntos. Os conjuntos L_i são denominados **camadas** do hiperarco, sendo as camadas extremas do hiperarco denominadas $rabo(e) = L_1$ e $cabeça(e) = L_r$ respectivamente.

Um hiperarco sem camadas é o hiperarco vazio, enquanto o hiperarco com uma única camada é chamado laço.

I.3.1 As classes 2-grafos e BF-grafos

Def.I.1: **2-grafos** são os hipergrafos cujos hiperarcos possuem exatamente duas camadas (2-arcos). O hipergrafo da Fig.I.1 é um 2-grafo.

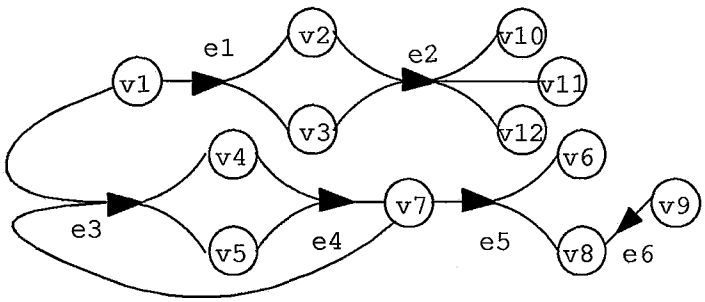


Figura I.1: Um 2-grafo

Essa é a classe de interesse desse trabalho, com enfoque sendo dado a uma sub-classe da 2-grafos, a BF-grafos. Para caracterizar a BF-grafos são necessárias algumas definições, apresentadas em seguida.

Def.I.2: Um 2-arco e tal que $|\ cabeça(e)\ | = 1$ é chamado um B-arco (Fig.I.2(a)).

Def.I.3: Um 2-arco e tal que $|\ rabo(e)\ | = 1$ é um F-arco (Fig.I.2(b)).

Def.I.4: O 2-grafo $H(V, E)$ tal que todo hiperarco $e \in E$ é um B-arco é chamado **B-grafo**.

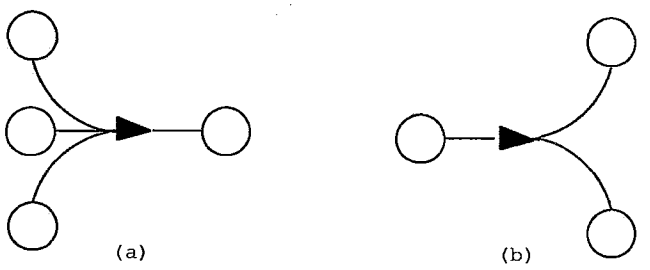


Figura I.2: Um B-arco (Fig.I.2(a)) e um F-arco (Fig.I.2(b))

Def.I.5: De maneira análoga, um **F-grafo** é o 2-grafo $H(V,E)$ tal que todo hiperarco $e \in E$ é um F-arco.

Def.I.6: Um **BF-grafo** é o 2-grafo cujos hiperarcos são ou B-arcos ou F-arcos (Fig.I.3).

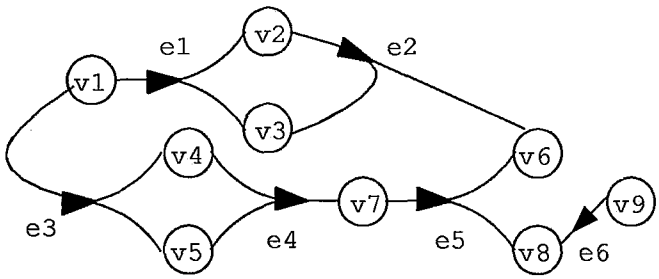


Figura I.3: Um BF-grafo

É interessante observar que todo 2-grafo pode ser transformado em um BF-grafo mediante a introdução de um novo vértice entre as duas camadas de cada um dos arcos do 2-grafo (Fig.I.4).

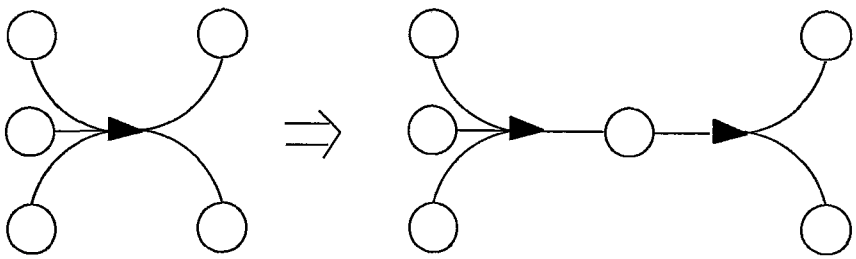


Figura I.4: Transformação de um 2-arco em um B-arco e um F-arco

Def.I.7: Dado um 2-grafo $H(V,E)$, a sua **imagem simétrica** é o hipergrafo $H_s(V,E_s)$ onde (X,Y) está em E_s se $(Y,X) \in E$. A imagem simétrica de um B-grafo é um F-grafo e vice-versa.

Seja $H(V,E)$ um 2-grafo. Com relação a um determinado vértice $v \in V$ podemos distinguir três subconjuntos de E. São eles:

$BS(v) = \ e \in E \ tq \ v \in cabeça(e)$;

$FS(v) = \ e \in E \ tq \ v \in rabo(e)$;

$ST(v) = \ e \in E \ tq \ v \in cabeça(e)$ ou $v \in rabo(e)$.

Ou seja $ST(v) = BS(v) \cup FS(v)$.

Considere o 2-grafo $H(V, E)$. Define-se:

$$size(E) = \sum_{e \in E} (\mid cabeça(e) \mid + \mid rabo(e) \mid)$$

Observa-se que :

$$\sum_{e \in E} (\mid cabeça(e) \mid + \mid rabo(e) \mid) = \sum_{v \in V} (\mid BS(v) \mid + \mid FS(v) \mid)$$

Verifica-se informalmente a validade dessa igualdade observando-se que nos dois somatórios é considerada uma única vez a pertinência de um determinado vértice v a uma determinada aresta e ($v \in cabeça(e) \cup rabo(e)$), para todos os vértices e arestas do 2-grafo. Mais adiante, a apresentação de uma das propostas de representação, a matriz de incidências, torna esse fato de fácil visualização, já que nos dois somatórios contabiliza-se a ocorrências dos elementos da matriz diferentes de 0, o primeiro sendo feito nas colunas e o segundo nas linhas da matriz.

I.3.2 Caminho e Ciclo em BF-Grafos

Generalizando a noção existente para grafos, define-se o que é caminho em um BF-grafo $H(V, E)$, identificando-se tipos de caminhos com características especiais:

Def.I.8: Um **caminho** de comprimento q é uma sequência alternada de vértices v_i e de arestas e_i. O caminho P que vai do vértice origem s ao vértice destino t é então:

$$P_{st} = (v_1 = s, e_1, v_2, e_2, ..., e_q, v_{q+1} = t)$$

onde

$$s \in rabo(e_1), t \in cabeca(e_q), v_j \in cabeca(e_{j-1}) \cap rabo(e_j), j = 2...q$$

Um possível caminho de v_1 a v_8 no BF-grafo H da Fig.I.3 aparece em destaque na Fig.I.5.

O vértice s é dito a **origem** de P_{st}, enquanto t é dito o **destino** de P_{st}.

Dados dois vértices $x, y \in V$, dizemos que y **é conectado** a x se existe um caminho P_{xy} em H. Deve-se observar que, em um BF-grafo orientado, y ser conectado a x não implica necessariamente x conectado a y.

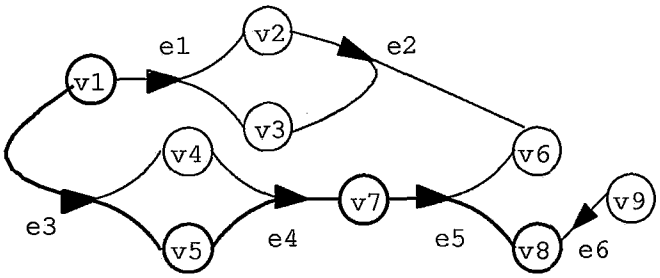


Figura I.5: Um caminho de v_1 a v_8 no BF-grafo H

$$x \text{ conectado a } y \Longleftrightarrow \exists\ P_{y,x} \text{ em } H$$

Def.I.9: Um **ciclo** em um BF-grafo é um caminho $P_{st} = (v_1 = s, e_1, v_2, e_2, ..., e_q, v_{q+1} = t)$, tal que $t \in rabo(e_1)$.

Exemplo: O ciclo $v_4, e_4, v_7, e_5, v_8, e_7, v_5$ na Fig.I.6

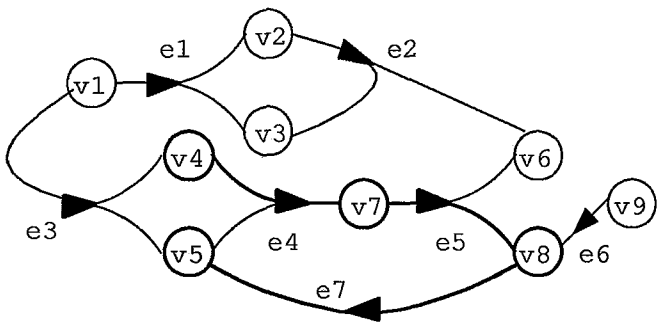


Figura I.6: Ciclo num BF-grafo

Seja C um ciclo de H. Então para qualquer par de vértices $(x, y) \in C$, $x, y \neq s$, x está conectado a y e y está conectado a x.

Um caminho é dito simples se todos os seus hiperarcos são distintos, e um caminho simples é dito elementar se todos os seus vértices são distintos. Analogamente, define-se ciclo simples e ciclo elementar. Um caminho é dito acíclico se ele não contém qualquer subcaminho que seja um ciclo.

I.3.3 Hipercaminhos em BF-grafos

Nos BF-grafos podem ser identificados caminhos especiais que se destacam pela seguinte característica: se um determinado hiperarco e pertence ao caminho, então todos os vértices de $cabeça(e) \cup rabo(e)$ pertencem ao caminho. Esses caminhos são chamados **hipercaminhos.**

Def.I.10: Um **hipercaminho** de origem s e destino t é o BF-grafo minimal $H_p = (V_p, E_p)$ tal que:

(i) $V_p \subseteq V$ e $E_p \subseteq E$;

(ii) $s, t \in V_p = \bigcup_{e \in E_p} e$;

(iii) t é conectado a s mediante um caminho simples;

Existem três tipos distintos de hipercaminhos: B-caminhos, F-caminhos e BF-caminhos. As respectivas definições são apresentadas em seguida, sendo conveniente esclarecer à priori a não obviedade das mesmas, ou seja, um B-caminho não é simplesmente um caminho formado por B-arcos.

Def.I.11: **B-caminho** de origem s e destino t :

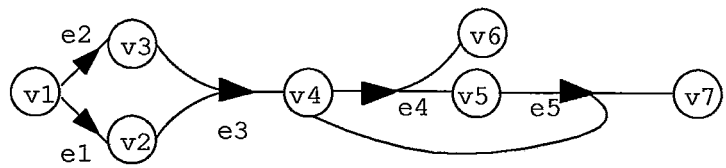
Um B-caminho de origem s e destino t é o BF-grafo minimal $H_b = (V_b, E_b)$ tal que:

(i) $E_b \subseteq E$;

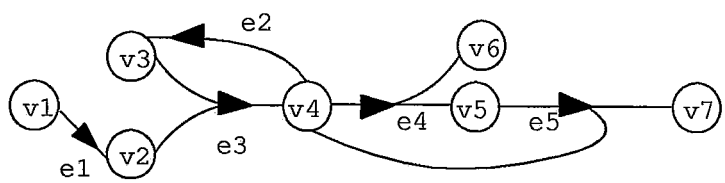
(ii) $s, t \in V_b = \bigcup_{e \in E_b} e$;

(iii) se $x \in V_b$ então x é conectado a s em H_b mediante um caminho simples acíclico;

Observe que a definição não implica aciclicidade do B-caminho; para todo vértice v do B-caminho existe um caminho sem ciclos da origem s até v. O BF-grafo da Fig.I.7(a) é um B-caminho; existem caminhos simples de v_1 a todos os nós do B-caminho, que não contêm o ciclo $(v_4, e_4, v_5, e_5, v_4)$ como subcaminho. No caso do BF-grafo da Fig.I.7(b), não existe caminho simples de v_1 a v_3 que não contenha o ciclo $(v_2, e_3, v_4, e_2, v_3)$. Logo, ele não é um B-caminho.



(a)



(b)

Figura I.7: Um B-caminho(a) e um BF-grafo que não é B-caminho(b)

A definição de **F-caminho** apresentada a seguir é obtida utilizando-se a noção de imagem simétrica introduzida na definição I.7.

Def.I.12: Um BF-grafo é um **F-caminho** de s a t se a sua imagem simétrica é um B-caminho de t a s.

É importante observar que um vértice x ser conectado a t por um caminho simples acíclico na imagem inversa de H é diferente de t ser conectado a x por um caminho simples acíclico em H.

O BF-grafo da Fig.I.8 é um F-caminho de v_1 a v_6.

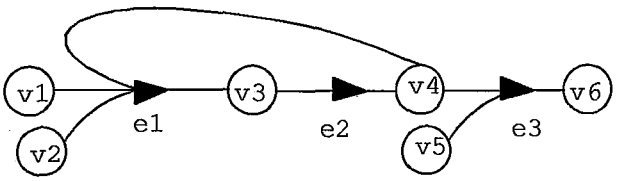


Figura I.8: Um F-caminho

Def.I.13: Um BF-caminho de s a t é o BF-grafo que é ao mesmo tempo um B-caminho e um F-caminho de s a t.

O BF-grafo da Fig.I.9 é um BF-caminho de v_1 a v_9.

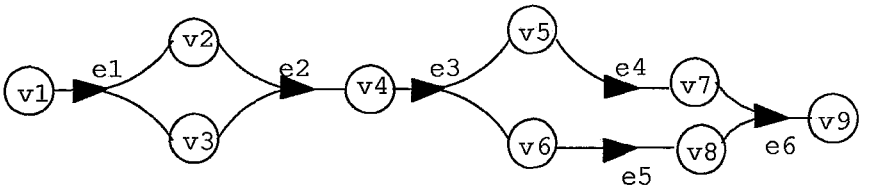


Figura I.9: Um BF-caminho

Dados um BF-grafo $H(V,E)$ e dois vértices $x, y \in V$, dizemos que o vértice x é B { F, BF }-conectado a y se existe um B {F, BF}-caminho P_{yx} em H.

Def.I.14: Dados um 2-grafo $H(V,E)$ e dois vértices $x, y \in V$, uma aresta $e \in E$ é chamada **aresta de ligação** entre x e y se $x \in rabo(e)$ e $y \in cabeça(e)$.

Capítulo II

Representações para 2-grafos

A representação de hipergrafos orientados em computador constitui o objeto de estudo desse capítulo. Mais especificamente, estaremos interessados em esquemas de representação para a classe 2-grafos.

O conceito de representação de um hipergrafo (orientado ou não) aqui utilizado é estabelecido de modo análogo ao conceito de representação de um grafo descrito em [Pombo 79]: considere um hipergrafo (orientado) H e a estrutura R_H que descreve esse hipergrafo (orientado). R_H é dita uma representação de H se todas as informações de H são unicamente determinadas a partir exclusivamente de R_H. Dessa forma, dados dois hipergrafos orientados distintos H_1 e H_2, suas respectivas representações R_1 e R_2 são necessariamente distintas.

Serão estudados e analisados diferentes esquemas de representação interna para a classe 2-grafo. É importante observar que qualquer esquema de representação válido deverá atender à condição fundamental do conceito de representação estabelecido acima: a biunivocidade entre um 2-grafo e sua representação. A avaliação da adequação de uma representação é feita objetivando-se a escolha de uma estrutura que permita que as operações mais freqüentes, como determinar quais os vértices do *rabo* de uma determinada aresta, sejam feitas com a maior eficiência possível.

Na próxima seção será descrito um conjunto básico de operações a ser utilizado na comparação dos diferentes esquemas de representação interna.

Um estudo inicial é feito a partir das principais e mais comuns representações de grafos, como matriz e lista de adjacência, quando da sua utilização para representar 2-grafos, verificando que nem sempre isso é possível e apresentando modificações que possam viabilizar essa utilização.

Para cada representação será apresentada sua definição, exemplos e os algoritmos que implementam cada uma das operações do conjunto básico, juntamente com a complexidade de cada um deles.

II.1 Um conjunto básico de operações

A partir do conjunto básico de operações abaixo será feito um estudo comparativo de eficiência para as representações apresentadas em seguida. As operações desse conjunto são as mais comumente utilizadas em algoritmos para problemas clássicos, como, por exemplo, buscas e obtenção de caminhos em 2-grafos.

Podemos identificar dois tipos de operações: as estruturais e as de manutenção dinâmica do 2-grafo.

No primeiro grupo se encontram as operações de pertinência de vértices e arestas, determinação de *cabeça* e *rabo* de uma aresta, determinação de BS e FS de um vértice.

O grupo de manutenção dinâmica consiste das operações de inserção e remoção de vértices e arestas.

Seja um 2-grafo $H = (V, E)$ cujos n vértices e m arestas são rotulados. Dados $v \in V$ e $e \in E$, as operações mais comuns seriam:

1) Identificar se $v \in$ *cabeça* (e).

Observe que $e \in BS(v)$ sse $v \in$ *cabeça(e)*. Assim sendo, determinar se $e \in BS(v)$ é equivalente a saber se $v \in$ *cabeça* (e), não havendo a necessidade de considerarmos uma operação distinta para cada caso.

2) Identificar se $v \in rabo(e)$.

Nesse caso $e \in FS(v)$ sse $v \in rabo(e)$, ou seja, determinar se $e \in FS(v)$ é equivalente a saber se $v \in rabo(e)$.

3) Determinar *cabeça(e)*.

Obter todos os vértices $v \in V$ *tq* $v \in$ *cabeça(e)*.

4) Determinar $rabo(e)$.

Obter todos os vértices $v \in V$ *tq* $v \in rabo(e)$.

5) Determinar $BS(v)$, dado um vértice $v \in V$.

Obter todos as arestas $e \in E$ *tq* $v \in$ *cabeça(e)*.

6) Determinar $FS(v)$, dado um vértice $v \in V$.

Obter todos as arestas $e \in E$ *tq* $v \in rabo(e)$.

7) Inserir em H um vértice v.

8) Inserir em H uma aresta ($rabo(e)$,*cabeça(e)*), $rabo(e)$, *cabeça(e)*$\subseteq V$.

9) Remover de H um vértice v, $v \in V$.

Deve-se notar que essa operação inclui a remoção de todas as arestas que tenham v em seus conjuntos *rabo* ou *cabeça*.

10) Remover de H uma aresta $e, e \in E$.

Existe ainda um terceiro grupo, com duas outras operações, que poderiam ser classificadas como compostas. Essas operações são as que estabelecem a existência de ligação entre vértices (ou conjuntos de vértices) no 2-grafo. Podem servir como critérito de desempate em alguns casos em que duas representações distintas equivalem em termos de eficiência com relação as operações dos dois outros grupos. São importantes para várias aplicações.

11) Dados dois vértices v e w, relacionar quais são as arestas de ligação entre v e w em H.

Pela definição I.14 a aresta $e \in E$ é chamada aresta de ligação entre v e w se $v \in rabo(e)$ e $w \in cabeça(e)$.

12) Dados dois conjuntos de vértices V_1 e V_2, determinar se $(V_1,V_2) \in E$.

II.2 Representações tradicionais para grafos

II.2.1 Matriz de Adjacência

Numa primeira tentativa de se estabelecer um esquema de representação para um 2-grafo orientado é bastante razoável levar-se em consideração a mais usual representação de grafos (digrafos) encontrada na literatura ([Pombo 79], [Szwarcfiter 84]), que é a matriz de adjacência. Dessa forma, estendendo a noção de matriz de adjacência para um 2-grafo orientado $H = (V, E)$ cujos n vértices estão rotulados, define-se a matriz $A_{n \times n}$, onde o elemento $a_{i,j}$, para i, j = 1...n, é:

$$a_{i,j} = \begin{cases} 1, & \text{se existe aresta } e \in E \text{ } tq \text{ } v_i \in rabo(e) \text{ e } v_j \in cabeça(e); \\ 0, & \text{caso contrário.} \end{cases}$$

Considere o 2-grafo H da figura II.1.

A matriz de adjacência para H seria:

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	1	1	1	0	0
v_2	0	0	0	0	1	0
v_3	0	0	0	0	1	1
v_4	0	0	0	0	0	1
v_5	0	0	0	0	0	0
v_6	0	0	0	0	0	0

Observa-se que a matriz de adjacência não atende à propriedade fundamental de representação (correspondência um-pra-um). Considere o 2-grafo H' da figura II.2.

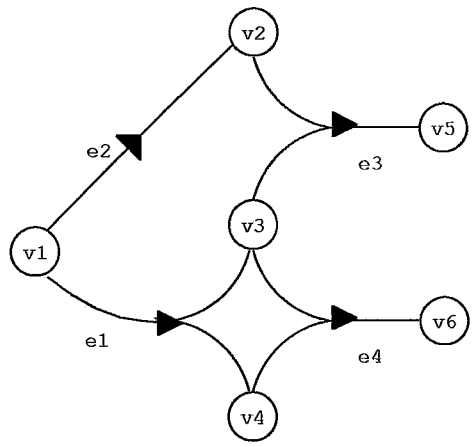


Figura II.1: O hipergrafo H

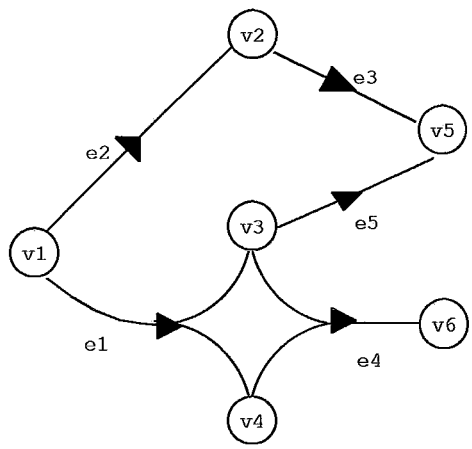


Figura II.2: O hipergrafo H'

É imediato verificar que a matriz de adjacência de H' é a mesma de H, não havendo forma de diferenciar a existência de uma única aresta de $(\{v_2, v_3\}, \{v_5\})$ em H e duas arestas distintas $(\{v_2\}, \{v_5\})$ e $(\{v_3\}, \{v_5\})$ em H'.

II.2.2 Listas de Adjacência

Analogamente ao que foi feito para matriz de adjacência, estende-se aqui a noção de listas de adjacência para hipergrafos. Nessa representação cada vértice v aponta para uma lista encadeada composta pelos vértices w tal que exista uma aresta de ligação entre v e w.

Lista de v: $\{ w \in V,\ \exists e \in E\ tq\ v \in rabo(e)$ e $w \in cabeça(e) \}$.

É natural esperar que o problema anterior ocorra também para listas de adjacência, já que essas são uma representação compacta (evitando espaços desper-

diçados com tantos 0's) da matriz de adjacência. Na verdade, as listas de adjacência pdem ser obtidas a partir da matriz de adjacência por uma transformação bem definida, onde para cada linha da matriz ligamos todos os vértices cujos elementos correspondentes nessa linha têm valor 1. Assim sendo, podemos comprovar a ocorrência do mesmo problema simplesmente verificando que as listas da figura II.3 são representações tanto de H como de H'(figuras II.1 e II.2).

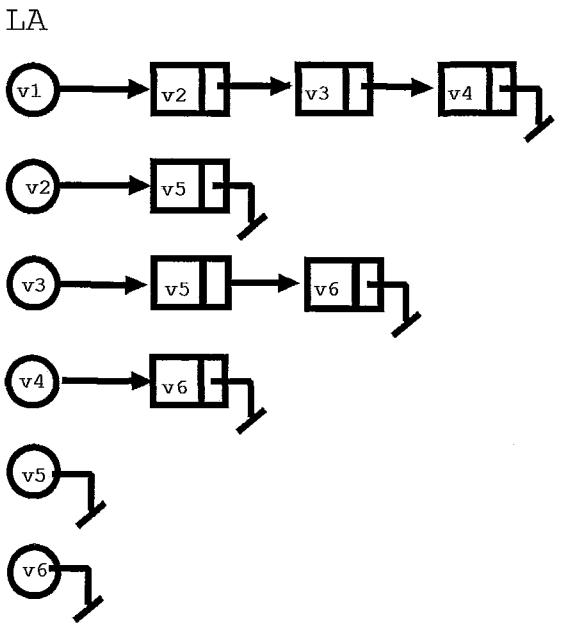


Figura II.3: Listas de Adjacência de H e H'

II.2.3 Alternativas

II.2.3.1 Matriz de Adjacência Modificada

Modificações na matriz de adjacência que possibilitem a sua utilização devem ser consideradas. Uma primeira idéia seria representar o 2-grafo H, cujos n vértices e m arestas estão rotulados, por uma matriz $A_{n \times n}$, onde o elemento $a_{i,j}$ contém, ao invés do valor 1, a aresta que liga v_i a v_j.

Para o 2-grafo H teríamos então:

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	e_2	e_1	e_1	0	0
v_2	0	0	0	0	e_3	0
v_3	0	0	0	0	e_3	e_4
v_4	0	0	0	0	0	e_4
v_5	0	0	0	0	0	0
v_6	0	0	0	0	0	0

Enquanto para H' a nova matriz seria:

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	e_2	e_1	e_1	0	0
v_2	0	0	0	0	e_3	0
v_3	0	0	0	0	e_5	e_4
v_4	0	0	0	0	0	e_4
v_5	0	0	0	0	0	0
v_6	0	0	0	0	0	0

Dessa forma teríamos representações distintas para os dois hipergrafos.

O problema que se coloca agora é como representar duas arestas distintas que fazem a ligação entre o mesmo par de vértices. Sejam $v_i, v_j \in V$, $e_p, e_q \in E$, tais que $v_i \in rabo(e_p) \cap rabo(e_q)$ e $v_j \in cabeça(e_p) \cap cabeça(e_q)$. A representação apresentada exige a escolha $a_{i,j}$=e_p ou $a_{i,j}$=e_q

Por exemplo, consideremos o 2-grafo H" abaixo.

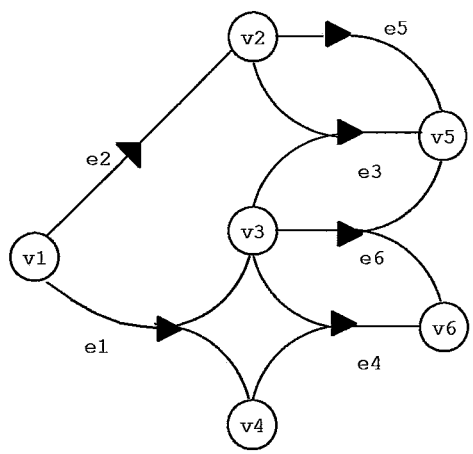


Figura II.4: O hipergrafo H"

Se considerarmos os vértices v_2 e v_5, $a_{2,5}$ só poderá conter ou e_3 ou e_5, caracterizando-se perda de informação.

A solução é fazer a célula da matriz apontar para uma lista das arestas que ligam v_i a v_j, observando que uma mesma aresta pode aparecer em duas ou mais listas diferentes.

A representação do hipergrafo H" da figura II.4 seria então a da figura II.5.

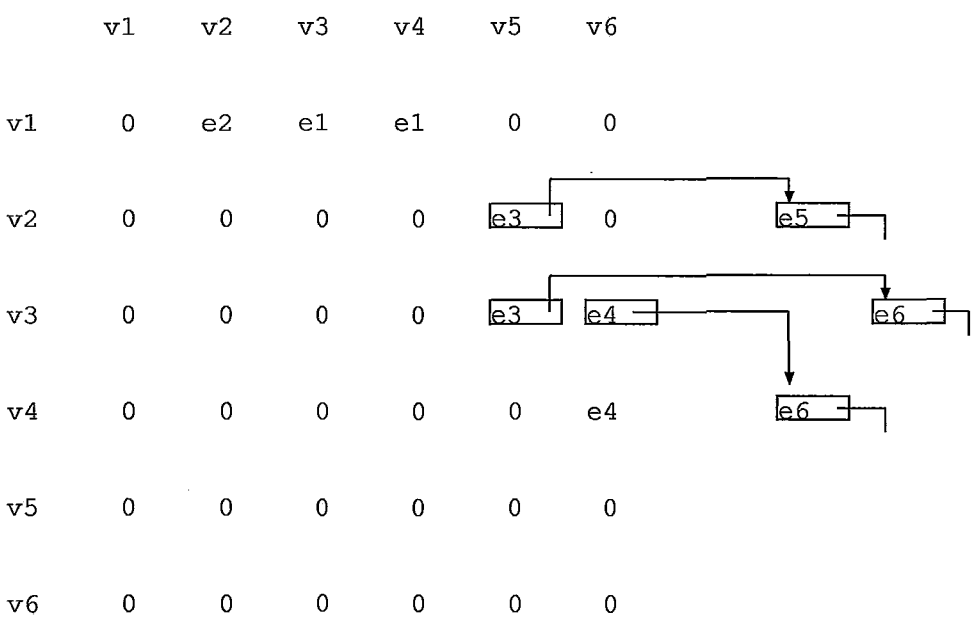


Figura II.5: Matriz de Adjacência com apontadores externos

Se a bidimensionalidade de uma matriz de adjacência já pode tornar proibitiva a sua utilização em alguns casos, dada a quantidade de memória necessária para armazená-la, a tridimensionalidade dessa nossa representação a torna ainda pior. Isso porque, em caso de empate ou proximidade das complexidades das operações para duas representações R_1 e R_2, o critério de escolha é a complexidade de memória necessária para cada representação. Não devemos, entretanto, desconsiderá-la.

II.2.3.2 Listas de Adjacência Modificadas

De maneira análoga a que foi feita para matriz de adjacência, procuraremos mudanças que viabilizem a utilização das listas de adjacência. A idéia é que um vértice v_j esteja na lista de v_i um número de vezes igual ao número de arestas e pertencentes a E tal que $v_i \in rabo(e)$ e $v_j \in cabeça(e)$. Além do campo para o vértice v_j e um ponteiro para o próximo vértice adjacente, uma posição extra deverá ser utilizada para guardar a aresta que foi considerada na colocação de v_j na lista. A essas listas chamaremos listas de adjacência modificadas. Por exemplo, para o hipergrafo H" da figura II.4, teríamos as listas da figura II.6.

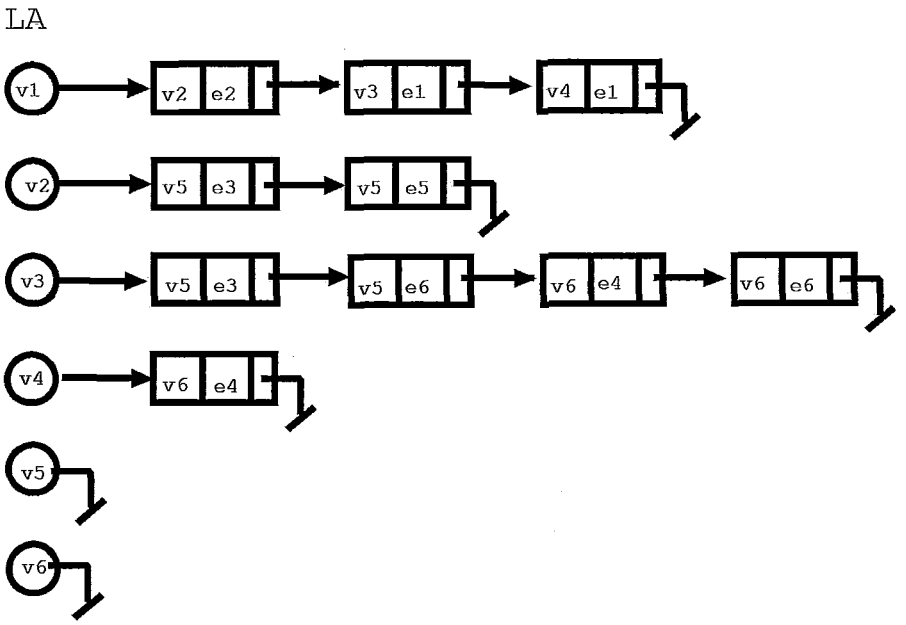


Figura II.6: Listas de Adjacência Modificadas de H"

As complexidades para as operações de obtenção de todos os vértices pertencentes à cabeça (ou rabo) de uma aresta e as arestas do BS de um vértice são muito grandes, pois nesses casos seria necessário percorrer até o fim todas as listas de adjacência de todos os vértices. Isso compromete tambem operações como deleção de vértices/arestas, que envolvem (explicitamente ou não) as operações acima. Isso justifica o não aprofundamento do estudo das complexidades para o nosso conjunto básico de operações no momento. No próximo capítulo, quando trataremos de casos que utilizam estruturas especiais auxiliares para a representação de arestas, voltaremos a falar em listas de adjacência modificadas.

A utilização das duas representações mais tradicionais de grafos na tentativa de representar hipergrafos parece natural e simples. No entanto, conforme anteriormente mostrado, isso pode ser incorreto ou, com algumas modificações estruturais, complexo. Veremos agora como se comporta uma outra proposta de representação, que embora conhecida para grafos, não é uma das representações mais usadas.

II.2.4 A Matriz de Incidência

A matriz de incidência para 2-grafos é a matriz $A_{m \times n}$, onde o elemento $a_{i,j}$ é:

$$a_{i,j} = \begin{cases} -1, & \text{se } v_i \in rabo(e_j), \\ 1, & \text{se } v_i \in cabeça(e_j), \\ 0, & \text{caso contrário.} \end{cases}$$

Exemplo: a matriz de incidência para o hipergrafo H'da figura II.2

	e_1	e_2	e_3	e_4	e_5
v_1	-1	-1	0	0	0
v_2	0	1	-1	0	-1
v_3	1	0	-1	-1	0
v_4	1	0	0	-1	0
v_5	0	0	1	0	1
v_6	0	0	0	1	0

A matriz de incidência é muito provavelmente o esquema de representação mais simples e natural em que se pode pensar para um 2-grafo H, e já aparece como tal nos trabalhos [Berge 70][Gallo 90] e [Markenzon 91]. É bastante fácil a identificação das características estruturais de H na matriz A_H que o representa, havendo basicamente as desvantagens de qualquer esquema matricial, que é a complexidade de espaço em memória possivelmente proibitiva e o fato das operações de inserção de vértices e arestas poder causar o redimensionamento da matriz.

A seguir, apresentaremos as rotinas para o conjunto básico de operações e suas respectivas complexidades.

Operações para a Matriz de Incidência:

1) Identificar se $v \in$ *cabeça*(e).

`rotina` Pertence_Cabeça(v,e)

pertence := `falso`;
`se` $a_{v,e} = 1$ `então` *pertence*:= `verdade`
`senão` *pertence* := `falso`;
`retorna`(*pertence*);

complex: $O(1)$.

A rotina (1) possui complexidade constante, bastando consultar a posição $a_{v,e}$ da matriz e testar o valor (no caso se igual a 1) para resolver de imediato a questão.

2) Identificar se $v \in$ *rabo*(e)

`rotina` Pertence_Rabo(v,e)

pertence := `falso`;
`se` $a_{v,e}$ = -1 `então` *pertence*:= `verdade`
`senão` *pertence* := `falso`;
`retorna`(*pertence*);

complex: $O(1)$.

A rotina (2) é obtida de maneira análoga à anterior, sofrendo modificação somente no teste (agora igual a -1). Logo se mantém a complexidade, também constante.

3) Determinar *cabeça(e)*.

`rotina` Obtem_Cabeça(e)
C_Cabeça := $\emptyset$;
`para todo` $v \in V$ `faça`
`se` $a_{v,e} = 1$ `então`
incluir v em *C_Cabeça* ;
`retorna`(*C_Cabeça*);

complex: $O(n)$.

A rotina consiste em percorrer toda a coluna relativa a e, incluindo no conjunto o vértice cujo elemento correspondente na coluna for igual a 1, ou seja, é feito um teste com complexidade constante para cada vértice.A complexidade no caso é da ordem do número máximo de vértices no 2-grafo (ou número de linhas da matriz): $O(n)$.

4) Determinar *rabo*(e).

`rotina` Obtem_Rabo(e)
C_Rabo:= $\emptyset$;
`para todo` $v \in V$ `faça`
`se` $a_{v,e}$ = -1 `então`
incluir v em *C_Rabo* ;
`retorna`(*C_Rabo*);

complex: $O(n)$.

Procede-se de forma semelhante a rotina anterior (3), considerando para inclusão os vértices com valor igual a -1 na coluna da aresta em questão.

5) Determinar $BS(v)$

```
rotina Obtem_BS(v)
    C_BS := ∅;
    para toda e ∈ E faça
        se a_{v,e} = 1 então
            incluir e em C_BS
    retorna(C_BS);
```

complex: $O(m)$.

A rotina consiste em percorrer a linha relativa a v, incluindo em $BS(v)$ a aresta cujo elemento correspondente na linha for igual a 1. É feito um teste (tempo constante) para cada aresta do 2-grafo. Complexidade é da ordem do número máximo de arestas (ou número de colunas), $O(m)$.

6) Determinar $FS(v)$

```
rotina Obtem_FS(v)
    C_FS := ∅;
    para toda e ∈ E faça
        se a_{v,e} = -1 então
            incluir e em C_FS
    retorna(C_FS);
```

complex: $O(m)$.

Rotina análoga a anterior que, para toda a aresta e do 2-grafo, testa em tempo constante se o valor na posição $a_{v,e}$ é igual a -1. A complexidade é também $O(m)$.

7) Inserir em H um vértice v tq $v \notin V$.

```
rotina Insere_Vertice(v, H)
    nlivre:= nlivre + 1;
    v:= nlivre;
    nn := nn +1
```

complex: $O(1)$.

A rotina para inserção de vértice não prevê situação de erro, como o esgotamento do espaço reservado e o conseqüente redimensionamento da matriz. Assim sendo, consiste simplesmente em associar ao vértice a ser inserido uma linha livre na matriz, e incrementar o número de vértices no 2-grafo, o que é feito em tempo constante. A variável nlivre (mlivre) é utilizada para guardar qual a posição livre para atribuir ao vértice (aresta) que está sendo inserido. O número real de vértices (arestas) no grafo é mantido em nn (mm). Isso porque as posições ocupadas por vértices (arestas) deletados não são liberadas.

8) Inserir em H a aresta ($rabo(e)$,$cabeça(e)$)

```
rotina Insere(rabo(e),cabeça(e), H)
    mlivre := mlivre + 1;
    e := mlivre;
    mm := mm +1;
     tt para todo v ∈ cabeça(e) faça
        a_{v,e} := 1;
    para todo v ∈ rabo(e) faça
        a_{v,e} := -1;
```

complex: $O(size_e)$

A rotina acima associa uma coluna livre da matriz a aresta a ser inserida. Depois disso, atualiza a informação de $|\ rabo(e)\ | + |\ cabeça(e)\ |$ posições da matriz. A complexidade é $O(size_e)$, onde $size_e = |\ rabo(e)\ | + |cabeça(e)\ |$.

9) Remover de H um vértice v, $v \in V$.

```
rotina Remove_Vertice(v,H)
    para toda e ∈ E faça
        se a_{v,e} ≠ 0 então
            Remove_Aresta(e, H)
            a_{v,e} := 0;
    nn:= nn-1;
```

complex: $O(m + st \times n)$

Deve-se observar que essa operação inclui a remoção de todas as arestas que tenham v em seus rabos ou cabeças. A rotina acima consiste em percorrer toda a linha de v ($O(m)$), para determinar $ST(v)$, isto é, as arestas que devem ser removidas. A remoção de uma aresta é feita com complexidade $O(n)$ (ver a rotina 10 a seguir). São feitas no total st remoções. A complexidade total nesse caso é $O(m + (st \times n))$, onde $st =\mid ST(v) \mid$.

10) Remover de H uma aresta $e, e \in E$.

```
rotina Remove_Aresta(e,H)
    para todo  v ∈ V faça
        a_{v,e} := 0;
    mm := mm-1;
```

complex: $O(n)$.

Para remover uma aresta e percorre-se toda a coluna relativa a e, atribuindo-se a todo elemento nessa coluna o valor 0. Cada atribuição é feita em tempo constante e no total são feitas n atribuições. A complexidade da rotina é $O(n)$.

11) Dados dois vértices v e w, relacionar quais as arestas de ligação entre v e w

```
rotina Obtem_Arestas_Ligação(v,w)
    C_Aresta_Ligação := ∅
    para toda e ∈ E faça
        se a_{v,e} = −1 e a_{w,e} = 1 então
            incluir e em C_Aresta_Ligação;
```

retorna(*C_Aresta_Ligação*);

complex: $O(m)$

Percorre-se as duas linhas da matriz relativas a v e w. No total são consultadas 2 posições por aresta ($a_{v,e}$ e $a_{w,e}$), para todas as arestas do 2-grafo. Logo, a complexidade é $O(m)$.

12) Dados dois conjuntos de vértices V_1, V_2, determinar se $(V_1, V_2) \in E$?

```
rotina Existe_Aresta(V1,V2);
    aresta := 0 ;
    para t1 ∈ V1 e h1 ∈ V2 faça
        Obtem_Arestas_Ligação(t1,h1)
    enquanto aresta = 0 e C_Aresta_Ligação ≠ ∅ faça
        retira e ∈ C_Aresta_Ligação
        Obtem_Cabeça(e); Obtem_Rabo(e)
        se C_Cabeça = V1 e C_Rabo = V2 então
            aresta := e;
    retorna (aresta);

    complex: O(m + kn)
```

Inicialmente obtém-se um conjunto de ligação para um par de vértices (t_1, h_1), o que é feito para qualquer par de vértices em tempo $O(m)$. O pior caso é considerarmos a seleção de um par (t_1, h_1), tal que o conjunto de arestas de ligação para esse par tenha a maior cardinalidade possível, dentre os conjuntos de ligação para qualquer par (t, h) tq $t \in V_1$ e $h \in V_2$. Seja k essa cardinalidade máxima, $k \leq m$. Para cada uma das k arestas desse conjunto de ligação, tem que se obter o conjunto *rabo* e o conjunto *cabeça*, o que é feito em ambos os casos com complexidade $O(n)$ (rotinas 4 e 3). A complexidade no total é $O(m + kn)$.

II.2.5 A Matriz de Incidência Modificada

A matriz de incidência pode ser redefinida de modo a melhorar a complexidade de algumas das operações. Para isso os elementos $a_{i,j}$ diferentes de 0 conterão, além da

informação de pertinência do vértice v_i à cabeça (sinal +) ou rabo (sinal -) da aresta e_j, qual o próximo vértice pertencente a esse mesmo conjunto. Ou seja, desse modo cada conjunto cabeça ou rabo de uma aresta e fica representado explicitamente como uma lista circular cujos elementos pertencem obrigatoriamente a mesma coluna na matriz (relativa a e). Além disso serão utilizados dois vetores auxiliares $PCab$ e $PRab$,de tamanho m, tais que $PCab[i]$ e $PRab[i]$ contêm os primeiros elementos de cabeça e rabo da aresta e_i respectivamente.

Com essa modificação, a nova matriz de incidência $A_{m\times n}$ é assim definida:

$$a_{i,j} = \begin{cases} -k, & /v_k \text{ é o próximo vértice do rabo}(e_j), \text{ se } v_i \in \text{rabo}(e_j) \\ k, & /v_k \text{ é o próximo vértice de cabeça}(e_j), \text{ se } v_i \in \text{cabeça}(e_j), \\ 0, & \text{caso contrário} \end{cases}$$

A matriz modificada para o grafo do exemplo anterior é:

	e_1	e_2	e_3	e_4	e_5
v_1	-1	-1	0	0	0
v_2	0	2	-3	0	-2
v_3	4	0	-2	-4	0
v_4	3	0	0	-3	0
v_5	0	0	5	0	5
v_6	0	0	0	6	0

Como é previsível, os algoritmos pertinentes ao conjunto de operações básico para a matriz de incidência modificada são semelhantes ao da matriz de incidência original. As diferenças ocorrerão nas rotinas que envolvam a obtenção do conjunto *cabeça(e)* ou *rabo*(e).

As operações para a Matriz de incidência Modificada:

1) Identificar se $v \in$ *cabeça*(e)

```
rotina Pertence_Cabeça(v,e)

    pertence := falso;
    se a_{v,e} > 0 então pertence:= verdade
    senão pertence := falso;
    retorna(pertence);
```

complex: $O(1)$.

A rotina tem complexidade constante, bastando fazer uma consulta à posição $a_{v,e}$.

2) Identificar se $v \in rabo(e)$

```
rotina Pertence_Rabo(v,e)

    pertence := falso;
    se a_{v,e} < 0 então pertence:= verdade
    senão pertence := falso;
    retorna(pertence);
```

complex: $O(1)$.

Rotina com complexidade constante, caso análogo a da rotina anterior.

3) Determinar *cabeça(e)*

```
rotina Obtem_Cabeça(e)
    C_Cabeça := ∅;
    primeiro := PCab[e];
    incluir primeiro em C_Cabeça;
    próximo := a_{primeiro,e}
    enquanto próximo ≠ primeiro faça
        incluir próximo em C_Cabeça;
        próximo := a_{próximo,e};
    retorna(C_Cabeça);
```

complex: $O(|\ cabeça(e)\ |)$.

Como os vértices de *cabeça(e)* estão agora encadeados circularmente na própria coluna, não é mais necessário consultar todos os elementos da coluna. É suficiente determinar em $PCab[e]$ o primeiro vértice do conjunto e a partir dai, cada vértice da *cabeça(e)* terá no seu elemento correspondente na coluna de e o próximo vértice que também está no conjunto. São consultadas somente as posições relativas a vértices pertencentes à *cabeça(e)*, logo a complexidade é de ordem igual a cardinalidade desse conjunto.

4) Determinar $rabo(e)$

```
rotina Obtem_Rabo(e)
    C_Rabo := ∅;
    primeiro := PRab[e];
    incluir primeiro em C_Rabo;
    próximo := - a_{primeiro,e}
    enquanto próximo ≠ primeiro faça
        incluir próximo em C_Rabo;
        próximo := - a_{próximo,e};
    retorna(C_Rabo)
```

complex: $O(|\ rabo(e)\ |)$

De modo semelhante ao da rotina anterior, só serão consultadas as posições relativas a vértices que estão no conjunto $rabo(e)$, a partir de um primeiro vértice guardado em $PRab[e]$. A complexidade é de tamanho igual a cardinalidade do conjunto.

5) Determinar $BS(v)$

```
rotina Obtem_BS(v)
    C_BS := ∅;
    para toda e ∈ E faça
        se a_{v,e} > 0 então
            incluir e em C_BS
    retorna(C_BS);
```

complex: $O(m)$.

As modificações na matriz de incidência não afetam a complexidade dessa operação, que continua sendo feita como na matriz de incidência original, a menos da mudança no teste da posição. Continua sendo necessário percorrer toda a linha de v. Assim sendo é consultada uma posição por aresta, para todas as arestas do 2-grafo. A complexidade é $O(m)$.

6) Determinar $FS(v)$

```
rotina Obtem_FS(v)
    Cn_FS := ∅;
    para toda e ∈ E faça
        se a_{v,e} < 0 então
            incluir e em C_FS
    retorna(C_FS);
```

complex: $O(m)$.

Com comportamento idêntico ao da rotina anterior, para obtenção de $FS(v)$, essa também não sofre mudanças na sua complexidade com relação à matriz original. Continua sendo necessário percorrer toda a linha de v. Mantém-se a complexidade, $O(m)$.

7) Inserir em H um vértice v *tq* $v \notin V$.

```
rotina Insere(v,H)
    nlivre:= nlivre + 1;
    nn := nn + 1;
    v:= nlivre
```

complex: $O(1)$.

Rotina idêntica à rotina de inserção de um vértice na matriz de incidência original.

8) Inserir em H a aresta $(rabo(e), cabeça(e))$

```
rotina Insere(rabo(e),cabeça(e),H)
    mlivre := mlivre + 1;
    e := mlivre;
    mm := mm + 1;
    PCab[e] := h_1;
    para i := 1 até |cabeça(e)| −1 faça
```

corrente := h_i;
$a_{corrente,e}$:= h_{i+1};
corrente := h_i;
$a_{corrente,e}$:= $PCab[e]$;
$PRab[e]$:= t_1;
para i := 1 até | $rabo(e)$ | −1 faça
corrente := t_i;
$a_{corrente,e}$:= - t_{i+1};
corrente := t_i;
$a_{corrente,e}$:= - $PRab[e]$;

complex: $O(size_e)$

Para cada vértice v pertencente ao conjunto *cabeça(e)*, a posição $a_{v,e}$ é modificada, passando a armazenar o próximo vértice desse mesmo conjunto. Procede-se da mesma maneira para os vértices do conjunto $rabo(e)$. É feito um acesso por vértice pertencente a $rabo(e) \cup$ *cabeça(e)*. A complexidade é igual a cardinalidade da união dos conjuntos, ou seja, $O(size_e)$.

9)Remover de H um vértice v, $v \in V$.

rotina Remove_Vertice(v,H)
para toda e em E faça
se $a_{v,e} \neq 0$ então
Remove_Aresta(e,H)
$a_{v,e}$:= 0;
nn := nn - 1;

complex:$O(m + st \times size_e)$

Na remoção de um vértice continua sendo necessário percorrer toda a linha de v na matriz, para obter $ST(v)$, o que requer tempo $O(m)$. São removidas st arestas, cada remoção sendo feita em $O(size_e)$ (rotina 10 abaixo). Assim sendo, a complexidade total é $O(m + st \times size_e)$.

10) Remover de H uma aresta $e, e \in E$.

```
rotina Remove_Aresta(e,H)
    Obtem_Cabeça(e); Obtem_Rabo(e);
    para todo v ∈ C_Cabeça ∪ C_Rabo faça
        a_{v,e} := 0;
    mm := mm-1;
```

complex: $O(size_e)$

A obtenção dos conjuntos *cabeça(e)* e *rabo*(e) é feita em $|$ *rabo*(e) $|$ + $|$*cabeça(e)* $|$, que é igual a $size_e$. Depois, para cada vértice v na união desses conjuntos, acessa-se, em tempo constante, a posição $a_{v,e}$ para atribuição do valor 0. Logo, a complexidade é da ordem da cardinalidade da união desses conjuntos, e igual a $size_e$.

11) Dados dois vértices v e w, relacionar quais as arestas de ligação e tais que $v \in$ *rabo*(e) e $w \in$ *cabeça*(e) .

```
rotina Obtem_Arestas_Ligação(v,w)
    C_Aresta_Ligação := ∅
    para toda e ∈ E faça
        se a_{v,e} > 0 ∧ a_{w,e} < 0 então
            incluir e em C_Aresta_Ligação;
    retorna(C_Aresta_Ligação);
```

complex: $O(m)$

À exceção dos testes para a inclusão da aresta no conjunto, a rotina é idêntica ao que é feito para o caso da matriz de incidência original, sendo mantida a complexidade.

12) Dados dois conjuntos de vértices V_1, V_2,determinar se $(V_1, V_2) \in E$.

```
rotina Existe_Aresta(V1,V2);
    aresta := 0 ;
    para t1 ∈ V1 e h1 ∈ V2 faça
        Obtem_Arestas_Ligação(t1,h1)
```

```
enquanto aresta = 0 e C_Aresta_Ligação ≠ ∅ faça
    retira e ∈ C_Aresta_Ligação
    Obtem_Cabeça(e); Obtem_Rabo(e)
    se C_Cabeça = V1 e C_Rabo = V2 então
        aresta := e;
retorna (aresta);
```

complex: $O(m + k \times size_e)$

Obtém-se, em tempo $O(m)$, um conjunto de ligação para um par de vértices (t_1, h_1). Considera-se k como definido para a mesma rotina (12) para a matriz de incidência original, ou seja, a cardinalidade máxima de um conjunto de ligação para qualquer dentre os possíveis pares (t, h), tq $t \in V_1$ e $h \in V_2$. Para cada uma das k arestas desse conjunto de ligação, deve-se obter o conjunto *rabo* e o *cabeça*, o que, diferente do caso da matriz de incidência original, é feito com complexidade $O(|\ rabo(e)\ | + |\ cabeça(e)\ )$. A complexidade no total é $O(m + k \times size_e)$.

II.3 Conclusões

As mais conhecidas representações de grafos, a matriz e as listas de adjacência, se mostram pouco adequadas à representação de 2-grafos orientados. As modificações que viabilizam sua utilização para representar 2-grafos, tornam-as de difícil manipulação e, no caso da matriz de adjacência, acarretam uma complexidade de espaço indesejável.

Em contrapartida, a matriz de incidência que, apesar de conhecida para grafos, era pouco utilizada, se revela bastante apropriada para a representação de 2-grafos. Apresenta, como qualquer esquema matricial, limitações quanto a inserção de vértices e arestas. Outro problema é a complexidade de espaço, eventualmente, proibitiva.

Algumas mudanças podem ser consideradas, objetivando-se maior eficiência em algumas das operações. É esse o caso da matriz de incidência modificada.

É importante levar em consideração que a matriz de incidência tem uma característica que a diferencia da matriz e listas de adjacência: é um esquema de representação orientado a arestas.

Pensar em esquemas em que existam estruturas para as arestas é um caminho a ser considerado. Ele será o ponto de partida para as representações do próximo

capítulo.

O quadro a seguir apresenta os resultados, em termos das complexidades dos algoritmos, para a matriz de incidência e a matriz de incidência modificada.

	Estruturas	
Operações	Matriz de Incidência	Matriz de Incidência Modificada
1) Identificar se $v \in cabeça(e)$	$O(1)$	$O(1)$
2) Identificar se $v \in rabo(e)$	$O(1)$	$O(1)$
3) Determinar $cabeça(e)$	$O(n)$	$O(\mid cabeça(e) \mid)$
4) Determinar $rabo(e)$	$O(n)$	$O(\mid rabo(e) \mid)$
5) Determinar $BS(v)$	$O(m)$	$O(m)$
6) Determinar $FS(v)$	$O(m)$	$O(m)$
7) Inserção de vértice	$O(1)$	$O(1)$
8) Inserção de aresta	$O(size_e)$	$O(size_e)$
9) Deleção de vértice	$O(m + st \times n)$	$O(m + st \times size_e)$
10) Deleção de aresta	$O(n)$	$O(size_e)$
11) Obtenção das arestas de ligação entre v e w	$O(m)$	$O(m)$
12) Determinar se $(V_1, V_2) \in E$	$O(m + k \times n)$	$O(m + k \times size_e)$

Capítulo III

Representações com as arestas explícitas

III.1 Estruturas para Hiper-Arestas

No capítulo anterior verificou-se que a matriz de adjacência e as listas de adjacência, as mais utilizadas representações de grafos, não são adequadas à representação de 2-grafos, enquanto a matriz de incidência se mostrou uma alternativa a ser considerada.

É interessante observar que a entidade aresta (hiper-aresta) se mostra mais complexa quando tratamos de hipergrafos do que quando tratávamos de grafos. Em grafos a presença de uma aresta entre v e w só caracterizava a existência de uma determinada relação entre eles. No caso da hiper-aresta ficam estabelecidas múltiplas relações, uma para cada par de vértices da aresta, podendo ser identificada a ocorrência de dois tipos de relação: uma quando v e w pertencem ao mesmo conjunto (*cabeça* ou *rabo*) e a outra quando v e w estão em conjuntos diferentes.

Dessa forma, é interessante pensar em esquemas de representação nos quais a aresta esteja explicitamente representada. A matriz de incidência já apresentava esta característica. Uma outra alternativa é a utilização de estruturas especiais para a representação das hiper-arestas como as estruturas propostas em [Markenzon 90] e apresentadas em seguida:

Um vetor *Arestas* contendo, para cada aresta e_i, o conjunto de vértices que forma rabo(e_i) e cabeça(e_i);

Um vetor $PontCab$, onde a posição $PontCab[i]$ contém o índice para obter o primeiro vértice de cabeça(e_i) em *Arestas*;

Um vetor $PontRab$, onde a posição $PontRab[i]$ contém o índice para obter o primeiro vértice de rabo(e_i) em *Arestas*.

Para o 2-grafo H" da figura III.1, as estruturas para as arestas seriam as da figura III.2.

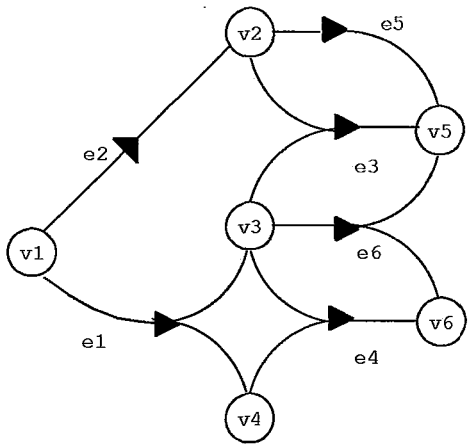


Figura III.1: O hipergrafo H"

hiper-arestas do 2-grafo H" .(III.2)

Arestas	1	3	4	1	2	2	3	5	3	4	6	2	5	3	5	6
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

PontRab	1	4	6	9	12	14

PontCab	2	5	8	11	13	15

Figura III.2: teste da figura

Essas estruturas são por si só uma representação (faz-se necessário somente o acréscimo da relação de vértices, por exemplo, num vetor, que permita a identificação dos vértices isolados do 2-grafo). Entretanto, quase toda operação que envolva a obtenção de informação sobre um vértice v, à exceção de pertinência a uma determinada aresta e, terá custo muito grande. Na maioria desses casos, como, por exemplo, identificar se existe aresta de ligação entre v e w ou obter os conjuntos $BS(v)$ e $FS(v)$, faz-se necessário obter as cabeças (os rabos) de todas as arestas do 2-grafo, consultando-se um grande número (ou a totalidade) de posições do vetor *Arestas*.

Dessa forma, a idéia é associar estruturas especificamente dedicadas às hiperarestas a um outro esquema de representação que se baseie em informações dos vértices.

Consideraremos inicialmente a associação às listas de adjacência modificadas do capítulo anterior. Um nó na lista de um vértice v contém dois tipos de informação: adjacência (w tal que existe aresta de ligação entre v e w), e incidência (a aresta e tal que $v \in rabo(e)$ e $w \in cabeça(e)$).

A análise das rotinas para as operações e as complexidades envolvidas nos mostrarão ser incidência a informação realmente relevante, que não é mantida eficientemente nas listas de adjacência modificadas, já que a mesma aresta e aparece um número de vezes igual a $|cabeça(e)|$ na lista de v.

As listas para o hipergrafo H" da figura III.1, com os vetores *Arestas*, *PontCab* e *PontRab*, são:

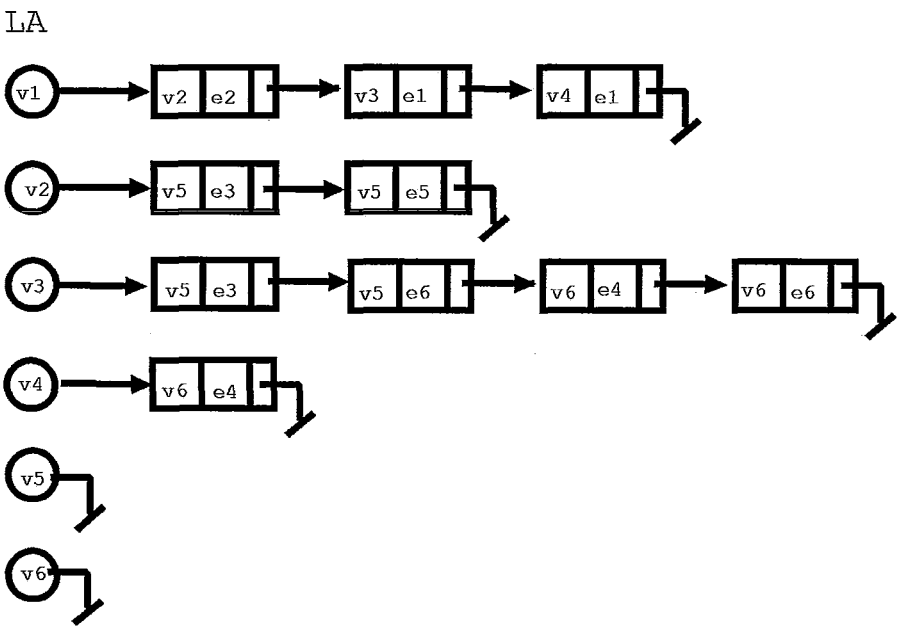


Figura III.3: Listas de Adjacência Modificadas de H"

A seguinte notação é utilizada na análise das complexidades das rotinas para as operações do conjunto básico.

$size_e = |\ rabo(e) + |\ cabeça(e)|$, $e \in E$.

$st(v) = |\ ST(v)\ |$, onde $ST(v) = FS(v) \cup BS(v)$, $v \in V$.

$n\_lig(v) = \sum_{e \in FS(v)} |cabeça(e)|$.

1) Identificar se $v \in cabeça(e)$.

```
rotina Pertence_Cabeça(v,e)
    pertence := falso;
    se PontCab[e] ≠ PontRab[e] então
        aux:= PontCab[e];
        enquanto aux ≠ PontRab[e + 1] e não pertence faça
            se v = Arestas[aux] então
                pertence := verdade ;
            aux:=aux + 1;
    retorna(pertence);
```

complex: $O(|\ cabeça(e)\ |)$.

Utiliza as informações das estruturas para arestas. Consiste em testar se v é igual a algum dos vértices no vetor *Arestas* que estão nas posições ocupadas pelos elementos de $cabeça(e)$. Tais posições são as que têm índices compreendidos entre $PontCab[e]$ e $PontRab[e+1]-1$. Serão testados, no máximo, $|\ cabeça(e)\ |$ vértices. Será feito, em tempo constante, um acesso e um teste por vértice. A complexidade total é $O(|\ cabeça(e)\ |)$

2) Identificar se $v \in rabo(e)$.

```
rotina Pertence_Rabo(v,e)
    pertence := falso;
    se PontCab[e] ≠ PontRab[e] então
        aux:= PontRab[e];
        enquanto aux ≠ PontCab[e] e não pertence faça
            se v = Arestas[aux] então
                pertence := verdade ;
            aux:=aux + 1;
    retorna(pertence);
```

complex: $O(|\ rabo(e)\ |)$.

Semelhante à rotina anterior, consiste em testar os elementos, do vetor *Arestas*, nas posições com índices compreendidos entre $PontRab[e]$ e $PontCab[e]-1$. São feitos, no máximo, $|\ rabo(e)\ |$ acessos.

3) Determinar *cabeça(e)*.

```
rotina Obtem_Cabeça(e)
    C_Cabeça := ∅;
    se PontCab[e] ≠ PontRab[e] então
        aux:= PontCab[e];
        enquanto aux ≠ PontRab[e + 1] faça
            incluir Arestas[aux] em C_Cabeça;
            aux:=aux + 1;
    retorna(C_Cabeça);
```

complex: $O(|\ cabeça(e)\ |)$.

O vértices de *cabeça*(e) são obtidos nas posições consecutivas no vetor *Arestas*, cujos índices estão compreendidos entre $PontCab[e]$ e $PontRab[e+1]-1$. São acessadas, no total, $|\ cabeça(e)\ |$ posições, com tempo constante por acesso.

4) Determinar *rabo*(e)

```
rotina Obtem_Rabo(e)
    C_rabo := ∅ ;
    se PontCab[e] ≠ PontRab[e] então
        aux:= PontRab[e];
        enquanto aux ≠ PontCab[e] faça
            incluir Arestas[aux] em C_Rabo;
            aux:=aux + 1;
    retorna(C_Rabo);
```

complex: $O(|\ rabo(e)\ |)$.

Analogamente à rotina anterior, são incluidos no conjunto os vértices obtidos nas posições consecutivas no vetor *Arestas*, cujos índices estão compreendidos entre $PontRab[e]$ e $PontCab[e]-1$. São acessadas, no total, $|\ rabo(e)\ |$ posições.

5) Determinar $BS(v)$.

```
rotina Obtem_BS(v)
    C_BS := ∅;
    para e= 1, .., m faça
        aux := PontCab[e];
        enquanto aux ≠ PontRab[e + 1] faça
            se Arestas[aux] = v então
                incluir e em C_BS;
            aux := aux +1;
    retorna(C_BS);
```

complex: $O(\sum_{e \in E} \mid cabeça(e) \mid)$.

Para cada aresta e do 2-grafo, consulta-se no vetor *Arestas* todas posições com os vértices de *cabeça(e)*. Para cada uma das m arestas, são feitos $\sum_{e \in E} \mid cabeca(e) \mid$ acessos.

6) Determinar $FS(v)$

```
rotina Obtem_FS(v)
    C_FS := ∅;
    aux := LA[v];
    enquanto aux ≠ NIL faça
        se aux↑.aresta não está em C_FS então
            incluir aux↑.aresta em C_FS;
        aux := aux↑.prox;
    retorna(C_FS);
```

complex : $O(n\_lig(v))$

A rotina consiste em percorrer a lista de v e incluir a aresta e no conjunto quando encontrar sua primeira ocorrência de e na lista. Um percurso numa lista é feito com complexidade da ordem do tamanho da lista. No caso a lista de adjacência modificada de v, $O(\sum_{e \in FS(v)} \mid cabeça(e) \mid)$, ou seja, $O(n\_lig(v))$.

Deve-se considerar que a informação utilizada na lista é a aresta do nó, havendo aumento da complexidade causada pelas $\mid cabeça(e) \mid$ repetições da aresta e na lista.

7) Inserir em H um vértice v *tq* $v \notin V$.

```
rotina Insere(v, H)
    n:= n + 1;
    VLivre := VLivre + 1;
    v:= VLivre;
    LA[v] := NIL;

complex: O(1).
```

Além da atribuição de um identificador ao vértice, consiste na criação de uma lista vazia, o que é feito em tempo constante. O número de vértices no 2-grafo é mantido em n. Entretanto, como as posições ocupadas por vértices deletados continuam indisponíveis, é necessária uma variável *VLivre* que conterá qual a primeira posição realmente livre. Na deleção de um vértice, apenas n será decrementada.

8) Inserir em H a aresta (*rabo(e)*,*cabeça(e)*)

```
rotina Insere(rabo(e),cabeça(e), H)
    m := m + 1;
    PLivre := PLivre + 1;
    e := PLivre;
    PontRab[PLivre] := ALivre;
    enquanto rabo(e) ≠ ∅ faça
        retira t de rabo(e);
        para todo h ∈ cabeça(e) faça
            aloca(aux1);
            aux1↑.prox := LA[t];
            aux1↑.vertice := h
            aux1↑.aresta := e;
            FS[t]:= aux1;
        Arestas[ALivre] := t;
        ALivre := ALivre + 1;
    PontCab[PLivre] := ALivre;
    enquanto cabeça(e)≠ Vazio faça
        retira h de cabeça(e);
        aloca(aux1);
        Arestas[ALivre] := h;
```

ALivre := ALivre + 1;

complex: $O(|\ rabo(e)\ | \times |\ cabeça(e)|)$.

Para um vértice v em $rabo(e)$, será inserido na lista de adjacência de v um nó com informação vert= w e aresta= e, para cada $w \in cabeça(e)$. Serão feitas inserções de $|\ cabeça(e)\ |$ nós nas listas de $|\ rabo(e)\ |$ vértices. Como o tempo de uma inserção é constante, já que ela poderá ser feita sempre no início da lista, a complexidade total é da ordem do número de inserções, ou seja, $O(|\ rabo(e)\ | \times |\ cabeça(e)|)$.

Na variável *ALivre* é guardada a primeira posição livre no vetor *Arestas*. *PLivre* guarda a primeira posição realmente livre nos vetores *PontCab* e *PontRab*. Como no caso dos vértices, também as posições ocupadas por arestas deletadas não ficam disponíveis.

9)Remover de H um vértice v, $v \in V$.

```
rotina Remove_Vertice(v, H)
    Obtem_FS(v);
    Obtem_BS(v);
    aux := LA[v];
    enquanto aux ≠ NIL faça
        LA[v] := aux↑.prox;
        libera(aux);
        aux := LA[v];

    para toda e ∈ C_BS ∪ C_FS faça
        Remove_Aresta(e, H)
    n := n-1;
```

complex: $O(st(v) \times \sum_{w \in rabo(e)} n\_lig(w))$, $e \in ST(v)$.

A obtenção de $BS(v)$ é feita em $O(\sum_{e \in E} |\ cabeça(e)\ |)$. A obtenção de $FS(v)$ e a remoção da lista LA[v] é feita em ordem do tamanho da lista, isso é, $O(n\_lig(v))$. A complexidade dominante será, no entanto, a da remoção de todas as arestas pertencentes à união dos dois conjuntos, São removidas st arestas, utilizando-se a rotina abaixo (10) na remoção de cada uma delas. A complexidade total é $O(st(v) \times \sum_{w \in rabo(e)} n\_lig(w))$.

10) Remover de H uma aresta $e, e \in E$.

```
rotina Remove(e,H)
    se PontCab[e] ≠ PontRab[e] então
        Obtem_Rabo(e);
        para todo v ∈ C_Rabo faça
            aux := LA[v]; ant := NIL;
            enquanto aux ≠ NIL faça
                se aux↑.aresta = e então
                    se LA[v] = aux então
                        LA[v] := aux↑.prox;
                        libera(aux) ;
                        aux := LA[v];
                    senão
                        ant↑.prox := aux↑.prox;
                        libera(aux);
                        aux := ant↑.prox
                senão
                    ant := aux;
                    aux := aux↑.prox;
        PontCab[e] := PontRab[e];
        m := m - 1;
```

complex: $O(\sum_{v \in rabo(e)} n\_lig(v))$.

Primeiro obtem-se $rabo(e)$, em $O(\mid rabo(e) \mid)$. Depois, para cada vértice v que pertence a $rabo(e)$, percorre-se a lista de v, removendo-se os nós cuja informação aresta é e. Isso é feito com complexidade da ordem do tamanho da lista, ou seja, $O(n\_lig(v))$. Como é percorrida uma lista para cada um dos $\mid rabo(e) \mid$ vértices, a complexidade total é $O(\sum_{v \in rabo(e)} n\_lig(v))$.

11) Dados dois vértices v e w, relacionar quais as arestas de ligação e entre v e w.

```
rotina Obtem_Arestas_Ligação(v,w)
    C_Aresta_Ligação := ∅
    aux := LA[v];
```

```
enquanto aux ≠ NIL faça
    se aux↑.vert = w então
        incluir aux↑.aresta em C_Aresta_Ligação;
        aux := aux↑.prox;
retorna(C_Aresta_Ligação);
```

complex: $O(n\_lig(v))$.

A rotina consiste em percorrer a lista encadeada LA[v], incluindo no conjunto a informação aresta do nó cuja informação vértice for w. Isso é feito em tempo da ordem do tamanho da lista. Como para cada aresta e em $FS(v)$ todos os vértices w no seu conjunto *cabeça* estão na lista, a complexidade é $O(\sum_{e \in FS(v)} \mid cabeça(e) \mid)$, ou seja, $O(n\_lig(v))$.

12) Dados dois conjuntos de vértices V_1, V_2, determinar se $(V_1, V_2) \in E$.

```
rotina Existe_Aresta(V1,V2);
    aresta := 0 ;
    para t1 ∈ V1 e h1 ∈ V2 faça
        Obtem_Arestas_Ligação(t1,h1)
    enquanto aresta = 0 e C_Aresta_Ligação ≠ ∅ faça
        retira e ∈ C_Aresta_Ligação
        Obtem_Cabeça(e); Obtem_Rabo(e)
        se C_Cabeça = V1 e C_Rabo = V2 então
            aresta := e;
    retorna (aresta);
```

complex: $O(max_{t \in V1}(n\_lig(t)) + (k \times size_e))$.

Se existir a aresta, ela obrigatoriamente pertence ao conjunto de ligação entre t_i e h_i, para qualquer par (t_i,h_i), tal que $t_i \in V_1$ e $h_i \in V_2$. Obtemos o conjunto de ligação para (t_1,h_1), em tempo $O(n\_lig(t_1))$. Considerando-se tal conjunto com cardinalidade máxima, k, resultante da pior escolha do par (t_1,h_1). Para cada aresta e no conjunto, obtém-se *cabeça*(e) e *rabo*(e), o que é feito em $O(size_e)$. A complexidade total é $O(max_{t \in V1}(n\_lig(t)) + (k \times size_e))$.

Comentários: Dois fatos devem ser notados. O primeiro é que a maioria das operações, particularmente as com melhores complexidades, são realizadas em cima das

estruturas das arestas, ou seja, as listas de adjacência trazem pouca contribuição na melhoria da eficiência. O segundo é serem justamente as operações envolvendo direta ou indiretamente a obtenção dos conjuntos BS e FS as mais comprometidas. No caso, a obtenção de $FS(v)$ é feita utilizando as informações das listas de adjacência modificadas, com complexidade menor do que a obtenção de $BS(v)$, que utiliza os vetores para arestas. Mesmo assim , a existência de $|$ *cabeça(e)* $|$ nós na lista para uma mesma aresta e implica uma complexidade maior do que se a lista contivesse apenas a informação de incidência. Isso nos leva a pensar numa representação em que os dois conjuntos $BS(v)$ e $FS(v)$ sejam de fácil obtenção, ou até mesmo, sirvam de base da representação. É esse o caso das Listas de Incidência apresentadas a seguir.

III.2 Listas de Incidência BS(v) e FS(v)

A idéia é representar o hipergrafo através da representação explícita dos conjuntos $BS(v)$ e $FS(v)$, para todo $v \in V$. Desse modo, haverá duas listas encadeadas para cada vértice v, uma com as arestas de $BS(v)$ e outra com as arestas de $FS(v)$. Além disso, também aqui as arestas estarão representadas explicitamente, mediante os vetores *Aresta*, *PontCab* e *PontRab* introduzidos na seção II.1.

As listas de incidência para nosso hipergrafo H" são:

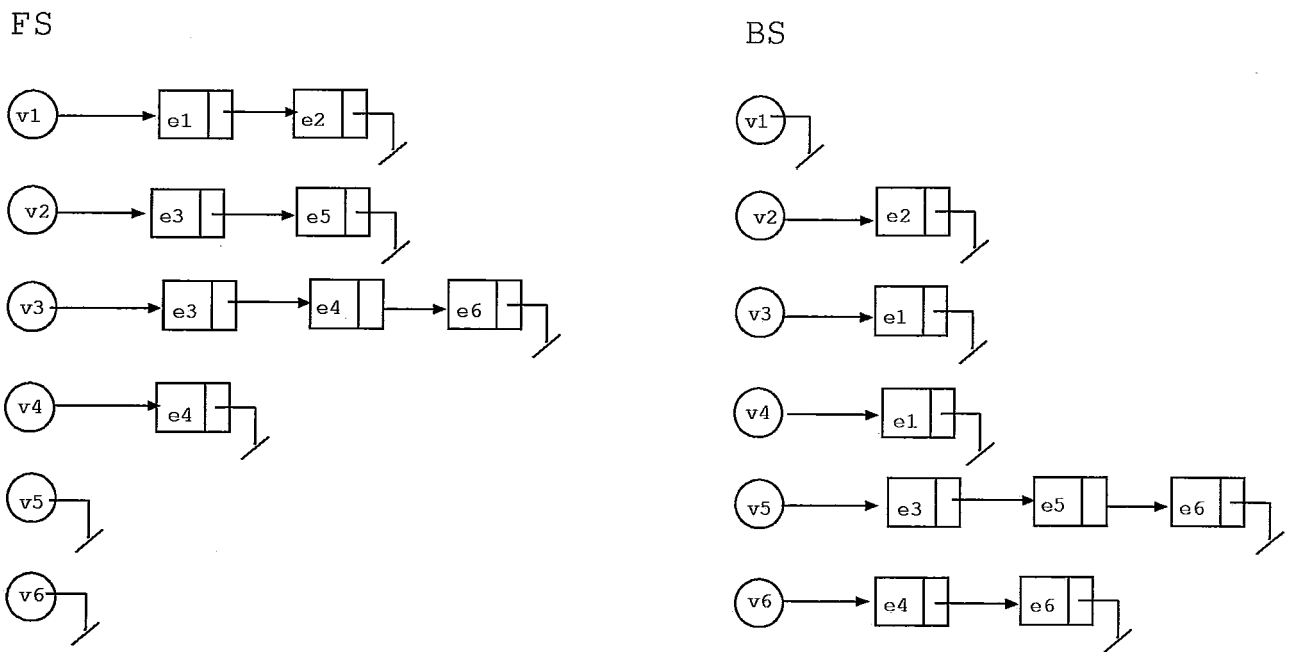


Figura III.4: Listas de Incidência de H"

Complexidade das operações para as Listas de Incidência:

1) Identificar se $v \in$ *cabeça*(e).

```
rotina Pertence_Cabeça(v,e)
    pertence := falso ;
    aux := BS[v];
        enquanto (aux ≠ NIL) e (não pertence) faça
            se aux↑.aresta = e então pertence := verdade ;
            aux:= aux↑.prox;
        retorna(pertence);

    complex: O(| BS(v) |).
```

Um vértice v está em *cabeça*(e) se e está em $BS(v)$. A rotina consiste então em fazer uma busca numa lista encadeada, no caso a $BS[v]$, o que é feito com complexidade da ordem do tamanho da lista, $O(\mid BS(v) \mid)$.

Obs.: Uma alternativa na implementação da rotina seria consultar, no vetor *Arestas*, as posições cujos índices estão compreendidos entre $PontCab[e]$ e $PontRab[e+1]-1$, que contêm os vértices de *cabeça*(e). A complexidade nesse caso seria $O(|$*cabeça*(e) $|)$.

2) Identificar se $v \in$ *rabo*(e).

```
rotina Pertence_Rabo(v,e)
    pertence := falso;
    aux := FS[v];
    enquanto (aux ≠ NIL) e (não pertence) faça
        se aux↑.aresta = e então pertence := verdade ;
        aux:= aux↑.prox;
    retorna(pertence);

complex: O(| FS(v) |).
```

Rotina com complexidade $O(\mid FS(v) \mid)$, pois trata-se de uma busca na lista $FS[v]$. Também aqui haveria a alternativa de consultar no vetor aresta as posições que guardam os vértices de *rabo*(e), cujos índices estão entre $PontRab[e]$ e $PontCab[e]-1$. A complexidade seria $O(\mid$ *rabo*(e) $\mid)$.

3) Determinar *cabeça(e)*.

```
rotina Obtem_Cabeça(e)
    C_Cabeça := ∅;
    se PontCab[e] ≠ PontRab[e] então
        aux:= PontCab[e];
        enquanto aux ≠ PontRab[e + 1] faça
            incluir Arestas[aux] em C_Cabeça;
            aux:=aux + 1;
    retorna(C_Cabeça);
```

complex: $O(|cabeça(e)|)$.

A rotina faz uso das informações nas estruturas para arestas. No caso, basta obter, incluindo no conjunto, os vértices em posições consecutivas do vetor *Arestas*, cujos índices estão entre $PontCab[e]$ e $PontRab[e+1]-1$. A complexidade é da ordem do número de posições consultadas, $O(|cabeça(e)|)$.

4) Determinar *rabo(e)*

```
rotina Obtem_Rabo(e)
    C_rabo := ∅ ;
    aux:= PontRab[e];
    enquanto aux ≠ PontCab[e] faça
        incluir Arestas[aux] em C_Rabo;
        aux:=aux + 1;
retorna(C_Rabo);
```

complex: $O(|\ rabo(e)\ |)$.

Procede-se de modo análogo ao da rotina anterior (3), obtendo-se os vértices de *rabo(e)* nas posições com índices entre $PontRab[e]$ e $PontCab[e+1]-1$. São consultadas, no total, $|\ rabo(e)\ |$ posições.

5) Determinar $BS(v)$.

```
rotina Obtem_BS(v)
    C_BS := ∅;
    aux := BS[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.aresta em C_BS;
        aux := aux↑.prox;
    retorna(C_BS);
```

complex: $O(\mid BS(v) \mid)$.

A rotina consiste em percorrer uma lista encadeada, no caso a $BS[v]$, incluindo no conjunto a aresta que é a informação de cada nó da lista. Isso é feito com complexidade da ordem do tamanho da lista, ou seja, $O(\mid BS(v) \mid)$.

6) Determinar $FS(v)$.

```
rotina Obtem_FS(v)
    C_FS := ∅;
    aux := FS[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.aresta em C_FS;
        aux := aux↑.prox;
    retorna(C_FS);
```

complex: $O(\mid FS(v) \mid)$.

Semelhante à rotina anterior (6), consiste em percorrer a lista $FS[v]$, o que implica complexidade $O(\mid FS(v) \mid)$.

7) Inserir em H um vértice v tq $v \notin V$.

```
rotina Insere(v, H)
    n:= n + 1;
    VLivre := VLivre + 1;
    v:= VLivre;
    BS[v] := NIL;
```

$FS[v]$:= NIL.

complex: O(1).

Requer a criação de duas listas vazias $BS[v]$ e $FS[v]$, o que é feito em tempo constante.

8) Inserir em H a aresta ($rabo(e)$,$cabeça(e)$)

```
rotina Insere(rabo(e),cabeça(e),H)
    m := m + 1;
    PLivre := PLivre + 1;
    e := PLivre;
    PontRab[PLivre] := ALivre;
    enquanto rabo(e) ≠ Vazio faça
        retira t de rabo(e);
        aloca(aux1);
        aux1↑.prox := FS[t];
        aux1↑.aresta := e;
        FS[t]:= aux1;
        Arestas[ALivre] := t;
        ALivre := ALivre + 1;
    PontCab[PLivre] := ALivre;
    enquanto cabeça(e)≠ Vazio faça
        retira h de cabeça(e);
        aloca(aux1);
        aux1↑.prox := BS[h];
        aux1↑.aresta := e;
        BS[h]:= aux1;
        Arestas[ALivre] := h;
        ALivre := ALivre + 1;
```

complex: $O(size_e)$.

A rotina consiste em, ao retirar um vértice v do conjunto $rabo(e)$, colocar v na primeira posição livre do vetor *Arestas* e inserir a aresta e no início da lista

$FS[v]$. Ambas as ações são feitas em tempo constante. Faz-se a mesma coisa para os vértices de *cabeça(e)*, fazendo dessa vez a inserção de e na lista $BS[v]$. No total, são considerados $size_e$ vértices, onde $size_e = |cabeça(e)| + |rabo(e)|$.

9)Remover de H um vértice v, $v \in V$.

```
rotina Remove_Vertice(v,H)
    Obtem_BS(v); Obtem_FS(v);
    para toda e ∈ C_BS ∪ C_FS faça
        Remove_Aresta(e,H);
    libera_lista(BS[v]);
    libera_lista(FS[v]);
    FS[v]:= NIL;
    BS[v]:= NIL;
    n:= n -1
```

complex: $O(st(v) \times (\sum_{r \in rabo(e)} | FS(r) | + \sum_{c \in cabeça(e)} | BS(c) |))$, $e \in ST(v)$.

Obs: Consideraremos a existência de uma rotina libera_lista(p), que percorre a lista apontada por p removendo cada nó da lista, com complexidade da ordem do tamanho da lista.

A remoção de um vértice implica a remoção de todas as arestas que tenham v em seus conjuntos *rabo* ou *cabeça*. Inicialmente são obtidos os conjuntos $BS(v)$ e $FS(v)$, o que é feito em tempo $O(|BS(v)| + |FS(v)|)$, utilizando-se as rotinas Obtem_BS e Obtem_FS. Depois disso, cada aresta em $ST(v) = BS(v) \cup FS(v)$ é removida, ou seja, são removidas st ($st = |ST|$) arestas. Na remoção de uma aresta utiliza-se a rotina Remove_Aresta, apresentada em seguida, que tem complexidade $O(\sum_{r \in rabo(e)} | FS(r) | + \sum_{c \in cabeça(e)} | BS(c) |)$.

10) Remover de H uma aresta $e, e \in E$.

```
rotina Remove_Aresta(e,H)
    se PontRab[e] ≠ PontCab[e] faça
        Obtem_Rabo(e);
        para todo v ∈ C_Rabo faça
            aux := FS[v];
            se aux = NIL então FS[v] := aux↑.prox
```

```
            senão
                ant := NIL;
                enquanto aux↑.aresta ≠ e faça
                    ant := aux;
                    aux := aux↑.prox;
                ant↑.prox := aux↑.prox;
            libera(aux);
        Obtem_Cabeça(e);
        para todo v ∈ C_Cabeça faça
            aux := BS[v];
            se aux = NIL então BS[v] := aux↑.prox
            senão
                ant := NIL;
                enquanto aux↑.aresta ≠ e faça
                    ant := aux;
                    aux := aux↑.prox;
                ant↑.prox := aux↑.prox;
            libera(aux);
        PontCab[e] := PontRab[e];
        m := m - 1;
```

complex: $O(\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid)$.

Primeiro obtém-se $rabo(e)$ ($O(\mid rabo(e) \mid)$) e $cabeça(e)$ ($O(\mid cabeça(e) \mid)$). Para cada $v \in rabo(e)$, faz-se uma busca na lista $FS[v]$, localizando e removendo o nó que contém e. Isso é feito em $O(\mid FS(v) \mid)$, para cada v. Do mesmo modo, para cada $v \in cabeça(e)$, faz-se a remoção do nó com e da lista $BS[v]$ ($O(\mid BS(v) \mid)$). Nas estruturas para arestas, uma aresta é identificada deletada se $PontCab[e] = PontRab[e]$, o que é feito com mediante uma simples atribuição, em tempo constante.

11) Dados dois vértices v e w, relacionar quais as arestas de ligação e entre v e w

```
rotina Obtem_Arestas_Ligação(v,w)
    C_Aresta_Ligação := ∅
    Obtem_FS(v); Obtem_BS(w);
    para e ∈C_FS(v)∩C_BS(w) faça
```

incluir e em *C_Aresta_Ligação*;
retorna(*C_Aresta_Ligação*);
complex: $O(max(\mid BS(w) \mid, \mid FS(v) \mid))$

Primeiro obtemos $FS(v)$, que é o conjunto de arestas tais que $v \in rabo(e)$, o que tem complexidade $O(\mid FS(v) \mid)$. Depois obtemos $BS(w)$, com as arestas e tais que $w \in cabeça(e)$, em $O(\mid BS(w) \mid)$. As arestas do conjunto de ligação são os elementos pertencentes à interseção desses dois conjuntos, que é obtido em tempo da ordem da cardinalidade do maior conjunto. A complexidade no total é da ordem da maior entre as cardinalidades de $BS(w)$ e $FS(v)$.

12) Dados dois conjuntos de vértices V_1, V_2, determinar se $(V_1, V_2) \in E$.

```
rotina Existe_Aresta(V1,V2);
    aresta := 0 ;
    para t1 ∈ V1 e h1 ∈ V2 faça
        Obtem_Arestas_Ligação(t1,h1)
    enquanto aresta = 0 e C_Aresta_Ligação ≠ ∅ faça
        retira e ∈ C_Aresta_Ligação
        Obtem_Cabeça(e); Obtem_Rabo(e)
        se C_Cabeça = V1 e C_Rabo = V2 então
            aresta := e;
    retorna (aresta);
```

complex: $O(max(\mid BS(h) \mid, \mid FS(t) \mid) + k \times size_e))$, $t \in V_1, h \in V_2$.

Se existir a aresta, ela obrigatoriamente pertence ao conjunto de ligação entre t_i e h_i, para qualquer par (t_i,h_i), tal que $t_i \in rabo(e)$ e $h_i \in cabeça(e)$. Obtemos o conjunto de ligação para (t_1,h_1), em tempo $O(max(\mid BS(h) \mid, \mid FS(t) \mid))$. Considerando-se tal conjunto com cardinalidade máxima, k, resultante da pior escolha do par (t_1,h_1). Para cada aresta e no conjunto, obtém-se $cabeça(e)$ e $rabo(e)$, o que é feito em $O(\mid cabeça(e) \mid + \mid rabo(e) \mid)$, ou seja, $O(size_e)$.

III.3 As Quatro Listas: Cabeça(e), Rabo(e), BS(v) e FS(v)

Diferencia-se da representação anterior pela estrutura utilizada para representar as arestas. Ao invés de utilizarmos os vetores *Arestas*, *PontRab* e *PontCab*, serão utilizadas as listas encadeadas $Cab(e)$ e $Rab(e)$. Nesse caso particular, a intenção é mostrar que não há diferença nas complexidades das operações entre ambas representações, ou seja, os vetores são equivalentes às listas Cab e Rab, em termos de complexidade de tempo. Entretanto, em termos de espaço, a remoção de arestas não deixará posições de memória ocupadas com arestas deletadas, o que acontecia com a utilização dos vetores. Evita-se também um erro causado pela má estimativa do tamanho máximo do vetor *Arestas*, que dependerá não somente de número de arestas total no 2-grafo, incluindo as deletadas, mas também das cardinalidades dos conjuntos *rabo* e *cabeça* dessas arestas.

Assim sendo, para o H" da Figura III.1, a representação é composta das quatro listas encadeadas da Figura III.5.

Complexidade das operações para as 4-Listas :

1) Identificar se $v \in cabeça(e)$.

```
rotina Pertence_Cabeça(v,e)

    pertence := falso;
    aux := Cab[e];
    enquanto (aux ≠ NIL) e (não pertence) faça
        se aux↑.vértice = v então pertence := verdade ;
        aux:= aux↑.prox;
    retorna(pertence);
```

complex: $O(|\ cabeça(e)\ |)$.

A rotina consiste em fazer uma busca em uma lista encadeada, a lista $Cab[e]$, o que é feito em na ordem do tamanho da lista, $O(|\ cabeça(e)\ |)$.

Obs: Uma alternativa para o algoritmo anterior seria percorrer a lista $BS[v]$ verificando se alguma das arestas desta é e. Com isso, a complexidade seria então $O(|\ BS(v)\ |)$.

2) Identificar se $v \in rabo(e)$.

```
rotina Pertence_Rabo(v,e)

    pertence := falso;
    aux := Rab[e];
    enquanto (aux ≠ NIL) e (não pertence) faça
        se aux↑.vértice = v então pertence := verdade ;
        aux:= aux↑.prox;
    retorna(pertence);
```

complex: $O(|\ rabo(e)\ |)$.

É feita uma busca na lista encadeada $Rab[e]$, a complexidade é $O(|\ rabo(e)\ |)$. Obs: Também nesse caso haveria uma alternativa, que consistiria em percorrer a lista $FS(v)$ verificando se e está nela. Nesse caso a complexidade seria $O(|\ FS(v)\ |)$.

3) Determinar *cabeça(e)*.

```
rotina Obtem_Cabeça(e)
    C_Cabeça := ∅;
    aux := Cab[e];
    enquanto (aux ≠ NIL) faça
        incluir aux↑.vertice em C_Cabeça;
        aux:= aux↑.prox;
    retorna (C_Cabeça) ;
```

complex: $O(|\ cabeça(e)\ |)$.

A rotina consiste em percorrer a lista encadeada $Cab[e]$, incluindo no conjunto o vértice contido em cada nó da lista. Isso é feito em $O(|\ cabeça(e)\ |)$.

4) Determinar $rabo(e)$

```
rotina Obtem_Rabo(e)
    C_Rabo := ∅
```

```
aux := Rab[e];
enquanto (aux ≠ NIL) faça
    incluir aux↑.vertice em C_Rabo;
    aux:= aux↑.prox;
retorna (C_Rabo);
```

complex: $O(|\ rabo(e)\ |)$.

Analogamente à rotina anterior (3), percorre-se a lista $Rab[e]$.

5) Determinar $BS(v)$.

```
rotina Obtem_BS(v)
    C_BS := ∅;
    aux := BS[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.aresta em C_BS;
        aux := aux↑.prox;
    retorna (C_BS);
```

complex: $O(|\ BS(v)\ |\ )$.

A rotina consiste em percorrer a lista encadeada $BS[v]$, incluindo no conjunto a aresta contida em cada nó da lista. Isso é feito em $O(|\ BS(v)\ |\ )$.

6) Determinar $FS(v)$.

```
rotina Obtem_FS(v)
    C_FS := ∅;
    aux := FS[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.aresta em C_FS;
        aux := aux↑.prox;
    retorna (C_FS);
```

complex: $O(|\ FS(v)\ |\ )$.

Analogamente à rotina anterior (5), percorre-se a lista $FS[v]$.

7) Inserir em H um vértice v tq $v \notin V(H)$.

```
rotina Insere(v,H)
    n:= n + 1; VLivre := VLivre + 1;
    v := VLivre;
    BS[v] := NIL;
    FS[v] := NIL.
```

complex: O(1).

Criação de duas listas vazias, $BS[v]$ e $FS[v]$. Requer tempo constante.

8) Inserir em H a aresta $(rabo(e),cabeça(e))$

```
rotina Insere(rabo(e),cabeça(e),H)
    m:= m + 1; ALivre=: ALivre + 1
    e:= ALivre;
    Rab[e] := NIL;
    Cab[e] := NIL;
    enquanto rabo(e) ≠ Vazio faça
        retira t de rabo(e);
        aloca(aux1);
        aux1↑.prox := FS[t];
        aux1↑.aresta := e;
        FS[t]:= aux1;
        aloca(aux2);
        aux2↑.prox := Rab[e];
        aux↑.vertice := t;
        Rab[e] := aux2;
    enquanto cabeça(e)≠ Vazio faça
        retira h de cabeça(e);
        aloca(aux1);
        aux1↑.prox := BS[h];
        aux1↑.aresta := e;
        BS[h]:= aux1;
```

aloca($aux2$);
$aux2\uparrow$.prox := $Cab[e]$;
$aux\uparrow$.vertice := h;
$Cab[e]$:= $aux2$;

complex: $O(size_e)$.

Para cada vértice $v \in rabo(e)$, insere-se v no início da lista $Rab[e]$ e insere-se e no início da lista $FS[v]$, o que, em ambos os casos, é feito em tempo constante. Analogamente, para os vértices $w \in cabeça(e)$, faz-se a inserção de v na lista $Cab[e]$ e e na lista $BS[v]$, totalizando $size_e$ inserções, cada uma realizada em tempo constante.

9)Remover de H um vértice v, $v \in V(H)$.

rotina Remove_Vertice(v,H)
Obtem_BS(v); Obtem_FS(v);
para toda $e \in C\_BS \cup C\_FS$ faça
Remove_Aresta(e,H);
libera_lista($BS[v]$);
libera_lista($FS[v]$);
$FS[v]$:= NIL;
$BS[v]$:= NIL;
n:= n -1

complex: $O(st(v) \times (\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid))$, $e \in ST(v)$.

A obtenção de $BS(v)$ e $FS(v)$ é feita em $O(st)$. Para cada aresta e na união dos conjuntos, faz-se a remoção de e utilizando a rotina abaixo (10), que tem complexidade $O(\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid)$.

10) Remover de H uma aresta $e, e \in E(H)$.

rotina Remove_Aresta(e,H)
Obtem_Rabo(e);
para todo $v \in C\_Rabo$ faça
aux := $FS[v]$;

```
se aux↑.elemento = e então FS[v] := aux↑.prox
senão
    ant := NIL;
    enquanto aux↑.aresta ≠ e faça
        ant := aux;
        aux := aux↑.prox;
    ant↑.prox := aux↑.prox;
libera(aux);
Obtem_Cabeça(e);
para todo v ∈ C_Cabeça faça
    aux := BS[v];
    se aux↑.elemento = e então BS[v] := aux↑.prox
    senão
        ant := NIL;
        enquanto aux↑.aresta ≠ e faça
            ant := aux;
            aux := aux↑.prox;
        ant↑.prox := aux↑.prox;
    libera(aux);
libera_lista(Cab[e]);
libera_lista(Rab[e]);
Cab[e] := NIL;
Rab[e] := NIL;
m := m - 1;
```

complex: $O(\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid)$.

Obtem-se $rabo(e)$ e $cabeça(e)$, com a posterior liberação das respectivas listas. ($O(\mid rabo(e) \mid + \mid cabeça(e) \mid)$). Para cada $v \in rabo(e)$, faz-se uma busca na lista $FS[v]$, localizando e removendo o nó que contém e. Isso é feito em $O(\mid FS(v) \mid)$, para cada v. Do mesmo modo, para cada $v \in cabeça(e)$, faz-se a remoção do nó com e da lista $BS[v]$ ($O(\mid BS(v) \mid)$).

11) Dados dois vértices v e w, relacionar quais as arestas de ligação e entre v e w

```
rotina Obtem_Arestas_Ligação(v,w)
```

C_Aresta_Ligação := ∅
Obtem_$FS(v)$; Obtem_$BS(w)$;
para e ∈C_FS(v)∩C_BS(w) faça
incluir e em *C_Aresta_Ligação*;
retorna(*C_Aresta_Ligação*);
complex: $O(max(\mid BS(w) \mid, \mid FS(v) \mid))$

Rotina análoga à rotina (11) da representação anterior.

12) Dados dois conjuntos de vértices V_1, V_2, determinar se $(V_1, V_2) \in E(H)$.

rotina Existe_Aresta(V_1,V_2);
aresta := 0 ;
para $t_1 \in V_1$ e $h_1 \in V_2$ faça
Obtem_Arestas_Ligação(t_1,h_1)
enquanto aresta = 0 e *C_Aresta_Ligação* $\neq \emptyset$ faça
retira $e \in$ *C_Aresta_Ligação*
Obtem_Cabeça(e); Obtem_Rabo(e)
se *C_Cabeça* = V_1 e *C_Rabo* = V_2 então
aresta := e;
retorna (aresta);

complex: $O(max(\mid BS(t) \mid, \mid FS(h) \mid) + k \times size_e))$.

Rotina análoga à rotina (12) da representação anterior.

III.4 O Digrafo D(H)

A idéia é representar um 2-grafo através de um digrafo, a partir da observação de equivalência entre o 2-grafo $H(V_h, E_h)$ e um digrafo $D(V_d, E_d)$, onde o conjunto de vértices V_d é igual a $V_h \cup V_a$, sendo V_a obtido pela criação de um novo vértice e' para cada aresta e do 2-grafo H. O conjunto de arestas E_d é obtido da seguinte forma:

para todo vértice criado e'(e) faça
para todo vértice $v \in$ *cabeça(e)* faça
crie a aresta orientada (e', v)

para todo vértice $v \in rabo(e)$ faça
crie a aresta orientada (v, e')

Exemplo : na figura III.6 temos o 2-grafo H e o digrafo correspondente.

Para verificar se há a preservação de todas as informações de H em D deverão ser examinados os casos problemas observados no estudo das representações anteriores.

Problema 1: A ambiguidade da representação. Por exemplo, em H uma única aresta e_3=({ v_2, v_3 } , { v_5 }) faz as ligações $v_2 - v_5$, enquanto em H' há duas arestas distintas estabelecendo as ligações, e_3=({ v_2 }, { v_5 }) e e_5=({ v_3 }, { v_5 }). É necessário que existam representações distintas para os dois casos, o que é garantido pela criação de um vértice para cada aresta. Isso pode ser verificado nas figuras III.6 e III.7.

Problema 2: Surge ao considerarmos que em um 2-grafo H pode haver duas ou mais arestas de ligação distintas entre o mesmo par de vértices (v, w). É o caso das arestas e_3 e e_5 com relação a v_3 e v_5 em H". No digrafo $D(H)$ elas estarão representadas por vértices distintos, o que resolve o problema (Figura III.8).

Garantida a preservação das informações, o problema torna-se um problema de representação de digrafos, que têm nas listas de adjacência sua mais tradicional representação. Além dessas, é comum por questões de eficiência, manter-se as listas com os antecessores imediatos de v, já que a remoção de um vértice v implica a remoção de todas as arestas em que v é predecessor ou sucessor.

É necessário manter para os vértices em D qual a sua origem em H (vértice ou aresta).

Na verdade, uma representação eficiente para um 2-grafo H requer a utilização de estruturas que contêm as mesmas informações de estruturas correspondentes numa representação eficiente do seu digrafo associado $D(H)$. No caso, as listas $FS[v]$ e $Cab[e]$ do 2-grafo H terão seus correspondentes nas listas de sucessores (listas de adjacência) do digrafo $D(H)$, enquanto as listas $BS[v]$ e $Rab[e]$ corresponderão às listas de antecessores.

A igualdade das complexidades das rotinas para ambas representações vêm ratificar esse fato.

LS = lista dos sucessores imediatos de v

LA = lista dos antecessores imediatos de v.

Complexidade das operações para digrafos:

1) Identificar se $v \in$ *cabeça*(e).

`rotina` Pertence_Cabeça(v,e)

pertence := `falso`;
aux := LS[e];
`enquanto` (*aux* $\neq$ NIL) e (não pertence) `faça`
`se` *aux*$\uparrow$.elemento = e `então` *pertence* := `verdade` ;
aux:= *aux*$\uparrow$.prox;
`retorna`(*pertence*);

complex: $O(|$ *cabeça(e)* $|)$.

Pode-se obter a mesma informação na lista de antecessores de v, $LA[v]$, com complexidade no caso, igual a $O(|\ BS(v)\ |)$

2) Identificar se $v \in$ *rabo*(e).

`rotina` Pertence_Rabo(v,e)

pertence := `falso`;
aux := LA[e];
`enquanto` (*aux* $\neq$ NIL) e (não pertence) `faça`
`se` *aux*$\uparrow$.elemento = e `então` *pertence* := `verdade` ;
aux:= *aux*$\uparrow$.prox;
`retorna`(*pertence*);

complex: $O(|$ *rabo*(e) $|)$.

Pode-se obter a mesma informação na lista de sucessores de v, $LS[v]$, com complexidade no caso, igual a $O(|\ FS(v)\ |)$

3) Determinar *cabeça(e)*.

`rotina` Obtem_Cabeça(e)
C_Cabeça := $\emptyset$;

```
    aux := LS[e];
    enquanto (aux ≠ NIL) faça
        incluir aux↑.elemento em C_Cabeça;
        aux:= aux↑.prox;
    retorna (C_Cabeça) ;
```

complex: $O(|\ cabeça(e)\ |)$.

4) Determinar $rabo(e)$

```
rotina Obtem_Rabo(e)
    C_Rabo := ∅
    aux := LA[e];
    enquanto (aux ≠ NIL) faça
        incluir aux↑.elemento em C_Rabo;
        aux:= aux↑.prox;
    retorna (C_Rabo);
```

complex: $O(|\ rabo(e)\ |)$.

5) Determinar $BS(v)$.

```
rotina Obtem_BS(v)
    C_BS := ∅;
    aux := LA[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.elemento em C_BS;
        aux := aux↑.prox;
    retorna (C_BS);
```

complex: $O(|\ BS(v)\ |\ )$.

6) Determinar $FS(v)$

```
rotina Obtem_FS(v)
```

```
    C_FS := ∅;
    aux := LS[v];
    enquanto aux ≠ NIL faça
        incluir aux↑.elemento em C_FS;
        aux := aux↑.prox;
    retorna (C_FS);
```

complex: $O(|\ FS(v)\ |\ )$.

7) Inserir em H um vértice v tq $v \notin V(H)$.

```
rotina Insere(v,H)
    n:= n + 1;
    nd := nd + 1;
    v := nd;
    origem[v] := VERTICE;
    LS[v] := NIL;
    LA[v] := NIL.
```

complex: O(1).

8) Inserir em H a aresta ($rabo(e)$,$cabeça(e)$)

```
rotina Insere(rabo(e),cabeça(e),H)
    alg: m:= m + 1;
         nd := nd + 1;
         e := nd;
         origem[e] := ARESTA;
         LS[e] := NIL;
         LA[e] := NIL.
         enquanto rabo(e) ≠ Vazio faça
             retira t de rabo(e);
             aloca(aux1);
             aux1↑.prox := LS[t];
             aux1↑.aresta := e;
             LS[t]:= aux1;
```

```
            aloca(aux2);
            aux2↑.prox := LA[e];
            aux↑.vertice := t;
            LA[e] := aux2;
        enquanto cabeça(e) ≠ Vazio faça
            retira h de cabeça(e);
            aloca(aux1);
            aux1↑.prox := LA[h];
            aux1↑.aresta := e;
            LA[h]:= aux1;
            aloca(aux2);
            aux2↑.prox := LS[e];
            aux↑.vertice := h;
            LS[e] := aux2;
```

complex: $O(size_e)$.

9)Remover de H um vértice v, $v \in V(H)$.

```
rotina Remove_Vertice(v,H)
    Obtem_BS(v); Obtem_FS(v);
    para toda e ∈ C_BS ∪ C_FS faça
        Remove_Aresta(e,H);
    libera_lista(LS[v]);
    libera_lista(LA[v]);
    LS[v]:= NIL;
    LA[v]:= NIL;
    n:= n -1
```

complex: $O(st(v) \times (\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid))$, $e \in ST(v)$.

10) Remover de H uma aresta $e, e \in E(H)$.

```
rotina Remove_Aresta(e,H)
    Obtem_Rabo(e);
    para todo v ∈ C_Rabo faça
```

```
        aux := LS[v];
        se aux↑.elemento = e então LS[v] := aux↑.prox
        senão
            ant := NIL;
            enquanto aux↑.elemento ≠ e faça
                ant := aux;
                aux := aux↑.prox;
            ant↑.prox := aux↑.prox;
        libera(aux);
        Obtem_Cabeça(e);
        para todo v ∈ C_Cabeça faça
            aux := LA[v];
            se aux↑.elemento = e então LA[v] := aux↑.prox
            senão
                ant := NIL;
                enquanto aux↑.aresta ≠ e faça
                    ant := aux;
                    aux := aux↑.prox;
                ant↑.prox := aux↑.prox;
            libera(aux);
        libera_lista(LS[e]);
        libera_lista(LA[e]);
        LS[e] := NIL;
        LA[e] := NIL;
        m := m - 1;
```

complex: $O(\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid)$

11) Dados dois vértices v e w, relacionar quais as arestas de ligação e entre v e w

```
rotina Obtem_Arestas_Ligação(v,w)
    C_Aresta_Ligação := ∅
    Obtem_FS(v); Obtem_BS(w);
    para e ∈C_FS(v)∩C_BS(w) faça
        incluir e em C_Aresta_Ligação;
    retorna(C_Aresta_Ligação);
```

complex: $O(max(\mid BS(w) \mid, \mid FS(v) \mid))$

12) Dados dois conjuntos de vértices V_1, V_2, determinar se $(V_1, V_2) \in E(H)$.

```
rotina Existe_Aresta(V1,V2);
    aresta := 0 ;
    para t1 ∈ V1 e h1 ∈ V2 faça
        Obtem_Arestas_Ligação(t1,h1)
    enquanto aresta = 0 e C_Aresta_Ligação ≠ ∅ faça
        retira e ∈ C_Aresta_Ligação
        Obtem_Cabeça(e); Obtem_Rabo(e)
        se C_Cabeça = V1 e C_Rabo = V2 então
            aresta := e;
    retorna (aresta);
```

complex: $O(max(\mid BS(t) \mid, \mid FS(h) \mid) + k \times size_e))$, $t \in V_1$.

III.5 Conclusões

No capítulo anterior, as representações mais comuns de grafos, a matriz de adjacência e as listas de adjacência, mostraram-se inadequadas à representação de 2-grafos.

Observou-se a necessidade de uma representação em que as arestas estivessem explicitamente representadas, não sendo decorrentes apenas das informações sobre os vértices.

A utilização de um vetor *Arestas*, para guardar os vértices dos conjuntos *cabeça* e *rabo* das arestas, solucionou parcialmente a questão, havendo restrições à sua utilização no caso dinâmico. Nesse caso, a utilização das listas encadeadas $Cab[e]$ e $Rab[e]$ é mais apropriada.

Para os vértices,a informação relevante não é a adjacência a outros vértices, mas a incidência às arestas. A opção, no caso, é a utilização das listas encadeadas $BS[v]$ e $FS[v]$.

Pode-se enxergar um 2-grafo como um digrafo, havendo um vértice extra correspondente a cada aresta do 2-grafo, sendo necessária a existência de estruturas com as mesmas informações para representar eficientemente o 2-grafo ou o digrafo associado.

Não foram considerados casos de 2-grafos com características especiais. No capítulo IV, a partir de uma estrutura para a representação não redundante dos conjuntos *rabo* em B-grafos, apresentada em [Ausiello 90], é apresentado um esquema de representação para um caso particular de 2-grafos.

	Estruturas	
Operações	Listas de Adjacência Modificadas + Vetores para Arestas	Listas $BS(v)$ e $FS(v)$ + Vetores para Arestas (**)
1) Identificar se $v \in cabeça(e)$	$O(\mid cabeça(e) \mid)$	$O(\mid cabeça(e) \mid)$
2) Identificar se $v \in rabo(e)$	$O(\mid rabo(e) \mid)$	$O(\mid rabo(e) \mid)$
3) Determinar $cabeça(e)$	$O(\mid cabeça(e) \mid)$	$O(\mid cabeça(e) \mid)$
4) Determinar $rabo(e)$	$O(\mid rabo(e) \mid)$	$O(\mid rabo(e) \mid)$
5) Determinar $BS(v)$	$O(\sum_{e \in E} cabeça(e))$	$O(\mid BS(v) \mid)$
6) Determinar $FS(v)$	$O(n\_lig(v))$	$O(\mid FS(v) \mid)$
7) Inserção de vértice	$O(1)$	$O(1)$
8) Inserção de aresta	$O(\mid rabo(e) \mid \times \mid cabeça(e) \mid)$	$O(size_e)$
9) Deleção de vértice	$O(st(v) \times \sum_{w \in rabo(e)} n\_lig(w))$, $e \in ST(v)$	$O(st(v) \times (\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid))$, $e \in ST(v)$
10) Deleção de aresta	$O(\sum_{v \in rabo(e)} n\_lig(v)$	$O(\sum_{r \in rabo(e)} \mid FS(r) \mid + \sum_{c \in cabeça(e)} \mid BS(c) \mid)$
11) Obtenção das arestas de ligação entre v e w	$O(\_lig(v))$	$O(max(\mid BS(w) \mid, \mid FS(v) \mid))$
12) Determinar se $(V_1, V_2) \in E$	$O(max_{t \in V1}(n\_lig(t))+ k \times size_e$	$O(max(\mid BS(t) \mid, \mid FS(h) \mid)+ k \times size_e))$

Obs.: As complexidades para as Quatro Listas e para Digrafo são as mesmas de (**).

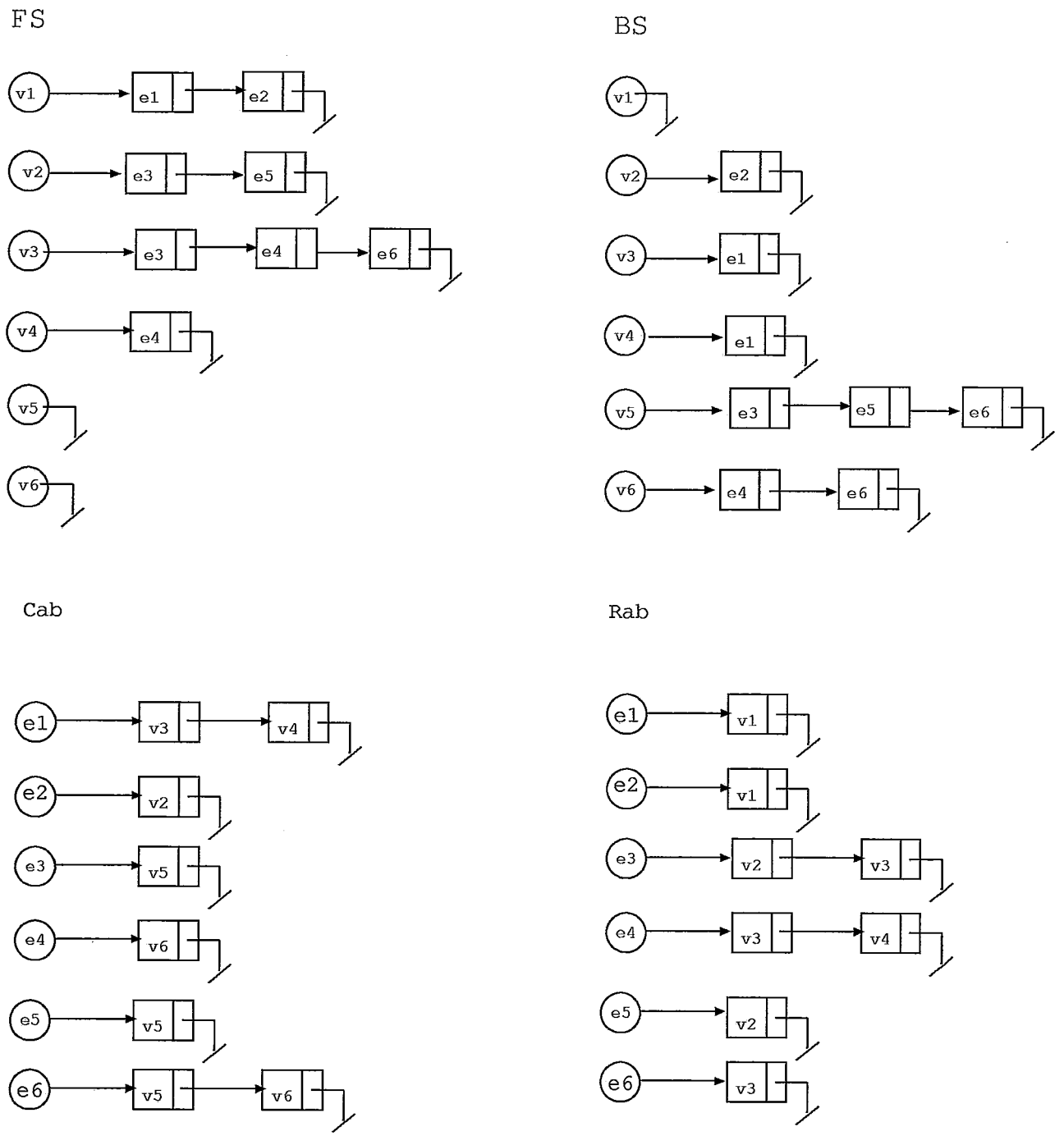


Figura III.5: As quatro listas para H"

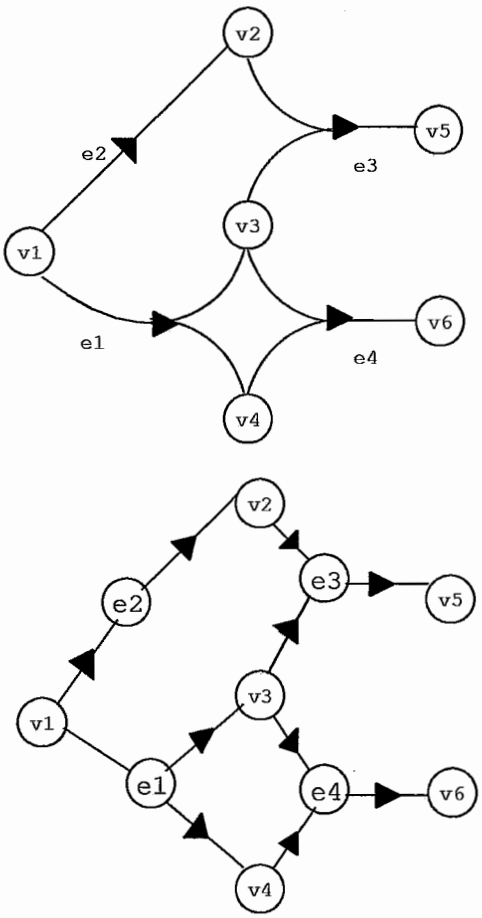


Figura III.6: O hipergrafo H e seu digrafo correspondente

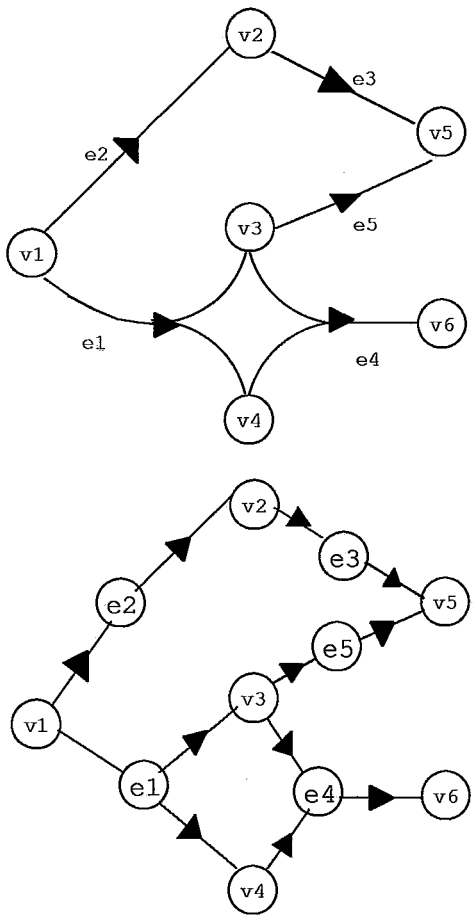


Figura III.7: O hipergrafo H' e seu digrafo

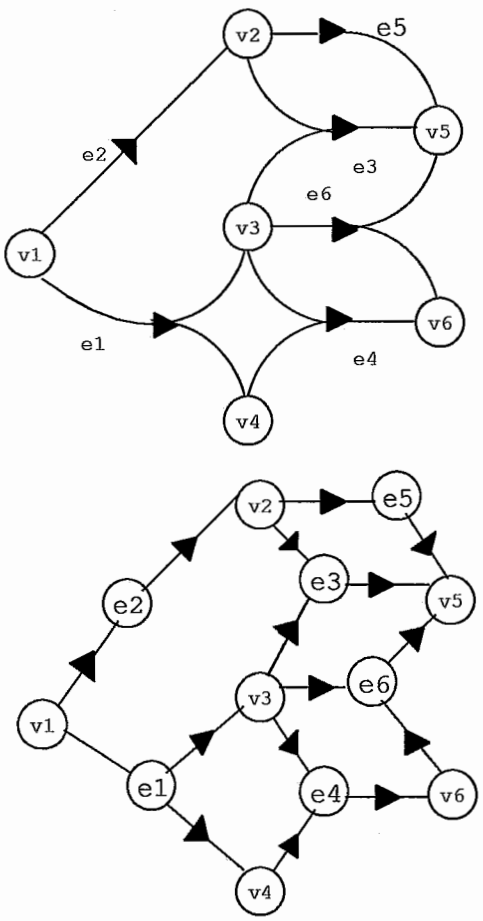


Figura III.8: O hipergrafo H"

Capítulo IV

Casos Particulares de 2-Grafos

IV.1 Introdução

Nesse capítulo serão apresentados definições e resultados interessantes para alguns casos particulares de hipergrafos orientados, encontrados na literatura. Além disso, procuraremos identificar corretamente a classe tratada em cada um deles, de acordo com a nossa notação, estabelecendo a correspondência entre os termos. É bastante comum encontrarmos trabalhos que apresentam determinada classe como sendo a hipergrafos orientados, quando, na verdade, trata-se de uma classe muito particular desta. É esse o caso dos hipergrafos orientados dos trabalhos de Ausiello ([Ausiello 83], [Ausiello 85], [Ausiello 86], [Ausiello 90]) e Woeginger ([Woeginger 92]), que veremos, mais adiante, serem B-grafos. Outro fato bastante comum é uma determinada estrutura ser referenciada em diversos trabalhos como uma aplicação de hipergrafos orientados, sem que esta relação esteja explícita em lugar algum, como acontece com os grafos E/Ou.

Na realidade os casos mais freqüentes na literatura correspondem aos B-grafos e F-grafos. Isso tem duas principais razões. A primeira é o fato de modelarem uma relação entre um conjunto de elementos (vértices) e um único elemento (vértice), relação esta freqüentemente encontrada em Ciência da Computação e que pode ser do tipo vários-em-um (B-arco) ou um-em-vários (F-arco). A segunda é que justamente estes casos particulares são, devido às suas muitas restrições, mais fáceis de se trabalhar e obter resultados relevantes.

As duas primeiras seções são dedicadas aos trabalhos de Ausiello. A primeira apresenta os principais conceitos relacionados nesses trabalhos e alguns dos resultados em [Ausiello 83], [Ausiello 85] e [Ausiello 86], estabelecendo-se a correspondência de terminologias. Estruturas e algoritmos para manutenção dinâmica do fechamento de hipergrafos orientados (B-grafos), encontrados em [Ausiello 90], são apresentados

em separado na segunda seção. A representação compacta para B-arcos, utilizada em [Ausiello 90], é estendida para a representação de BF-grafos e 2-grafos na seção IV.3.

Na seção IV.4 os grafos E/Ou e E/Ou generalizados são caracterizados como aplicações de BF-grafos.

O objeto de estudo da última seção é o trabalho de Woeginger ([Woeginger 92]), que trata da conceituação e obtenção de arborescência para a classe definida e referida pelo autor como hipergrafos orientados.

IV.2 Hipergrafos Orientados Segundo Ausiello

Esta seção será destinada aos hipergrafos de Ausiello, estudados em [Ausiello 83], [Ausiello 85], [Ausiello 86] e [Ausiello 90].

A denominação hipergrafos orientados é dada a uma classe de hipergrafos definida e tratada por Ausiello nesses quatro trabalhos. Essa classe pode ser informalmente definida como aquela em que cada hiperaresta (orientada) parte de um conjunto de vértices e chega a um único vértice. Essa relação de vários-em-um é frequentemente encontrada em Ciência da Computação, conforme as aplicações referidas em [Ausiello 86]. De acordo com as nossas definições, essa classe é na verdade a B-grafos, uma subclasse de hipergrafos orientados.

Nesses trabalhos podem ser encontrados diversas definições, problemas e resultados interessantes para a referida classe. Também essas definições, problemas e resultados podem equivaler a conceitos ou casos específicos na nossa notação. Por enquanto, utilizaremos a terminologia adotada pelos autores, deixando para explicitar as correspondências mais adiante.

Uma classe especial de grafos, a FD-grafos, é introduzida em [Ausiello 83], como forma de representação de dependências funcionais em banco de dados relacional. Essa classe é utilizada em trabalhos posteriores ([Ausiello 86] e [Ausiello 90]) como forma de representação de hipergrafos orientados.

Equivalência e representações minimais equivalentes são o assunto de [Ausiello 86], que relaciona os diferentes critérios, de acordo com os parâmetros considerados, para a conceituação e obtenção de minimalidade. Uma noção de equivalência mais forte do que essa havia sido introduzida em [Ausiello 85].

O tratamento dinâmico de hipergrafos é apresentado, pela primeira vez, em [Ausiello 90], que apresenta estruturas e algoritmos para a manutenção eficiente do fechamento de hipergrafos orientados no caso dinâmico. Uma forma compacta de

representar B-arcos é introduzida e, será utilizada em um esquema de representação para 2-grafos com restrições, proposto na seção IV.3.

São apresentadas inicialmente a notação e as principais definições contidas nesses trabalhos. É, então, apresentada a classe FD-grafos. Em seguida, são apresentados critérios para minimalidade e o conceito de equivalência forte. Finalmente, é estabelecida a correspondência entre notações, buscando-se, quando possível, estender as definições e resultados vistos anteriormente para classes mais genéricas, como a BF-grafos e a 2-grafos. O caso dinâmico será tratado no próximo seção.

IV.2.1 Principais Definições

Abaixo estão relacionados, além das definições básicas da classe, conceitos que se destacam no estudo das aplicações de hipergrafos orientados, análogos aos existentes para grafos. São conceitos como caminho, fechamento, equivalência e representação minimal, comuns aos quatro trabalhos, com algumas diferenças na formalidade da apresentação, que procuraremos padronizar.

Def.IV.1: Um **hipergrafo orientado** H é o par (N, E) onde N é o conjunto de nós (nós simples) e E é o conjunto de hiperarcos. Exemplo: Figura IV.1

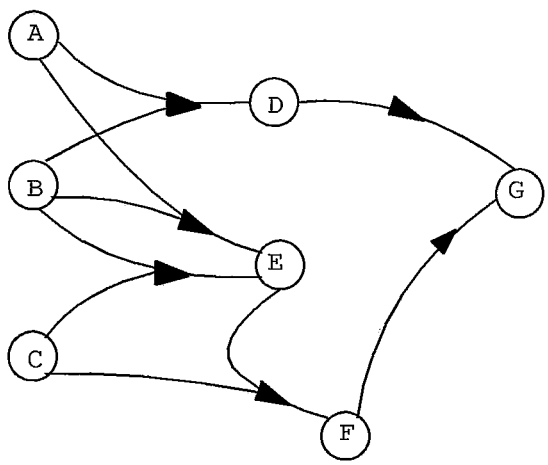


Figura IV.1: O hipergrafo H

Def.IV.2: Um **hiperarco** é um par ordenado (X, i), onde $X \subset N$ é um conjunto não vazio e $i \in N$. Exemplo: na figura IV.2 estão alguns hiperarcos do hipergrafo da figura IV.1.

Def.IV.3: Um **conjunto origem** é um conjunto $X \subset N$ tal que existe ao menos um hiperarco $(X, i) \in E$, $i \in N$. Exemplo: alguns dos conjuntos origem do hipergrafo da figura IV.1 seriam {A,B }, {B,C } e {C,E }.

Simbologia básica relacionada:

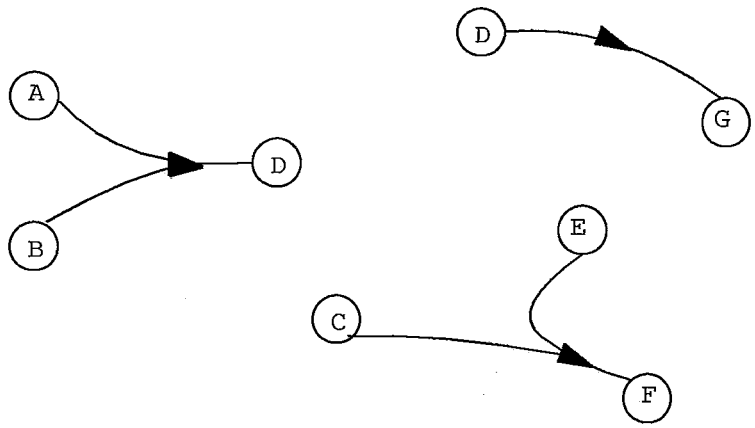


Figura IV.2: Alguns hiperarcos no hipergrafo H

$P(N)$ = conjunto das partes de N (todos os subconjuntos de N)

n_s = número de nós simples

h = número de hiperarcos

s = número de conjuntos origem

a = somatório das cardinalidades de todos os conjuntos origem distintos de H, denominado área da origem.

m = tamanho de H (tamanho da descrição total de H), m = a + h

Def.IV.4: Dados um hipergrafo orientado $H=(N, E)$, um subconjunto de nós $X \subseteq N$ e um nó $i \in N$, existe um **hipercaminho** C de X a i se uma das seguintes condições é satisfeita:

(i) $i \in X$ (reflexividade estendida);

(ii) existe um hiperarco $(Y, i) \in E$ e para cada $j \in Y$ existe um hipercaminho de X a j (transitividade estendida).

Exemplo: um hipercaminho de BC a E no hipergrafo H da figura IV.1 é mostrado na figura IV.3.

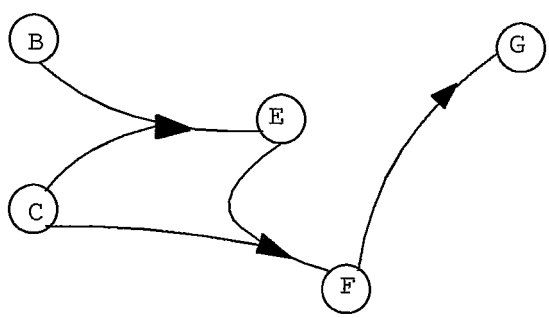


Figura IV.3: Hipercaminho de BC a E

Def.IV.5: Dado o hipergrafo orientado $H=(N,E)$, o **fechamento** de H, denotado por H^+, é o hipergrafo orientado (N,E^+), tal que $(X,i) \in E^+$ se uma das seguintes condições é satisfeita:

(i) (X,i) é um hiperarco em E;

(ii) i é um elemento de X;

(iii) existe conjunto de nós $Y = \{n_1,...,n_q\}$ tal que existem hiperarcos (X,n_j), para $j = 1...q$, em E^+ e (Y,i) é um hiperarco em E.

O que é equivalente a dizer que existe em H um hipercaminho de X a i.

Alguns hiperarcos no fechamento do hipergrafo da figura IV.1 são: { (AB,A), (ABC,A), (AB,D) }.

Deve-se observar que o número de arcos no fechamento de um hipergrafo orientado pode ser exponencial no número de nós.

Dado um conjunto X, denominamos fechamento de X ao conjunto de nós $i \in N$, tal que $(X,i) \in H^+$.

Def.IV.6: Dois hipergrafos orientados H e H' são ditos **equivalentes** se e somente se eles têm o mesmo fechamento, ou seja, $H^+=H'^+$. Por exemplo, o hipergrafo da figura IV.4 é equivalente ao hipergrafo H da figura IV.1.

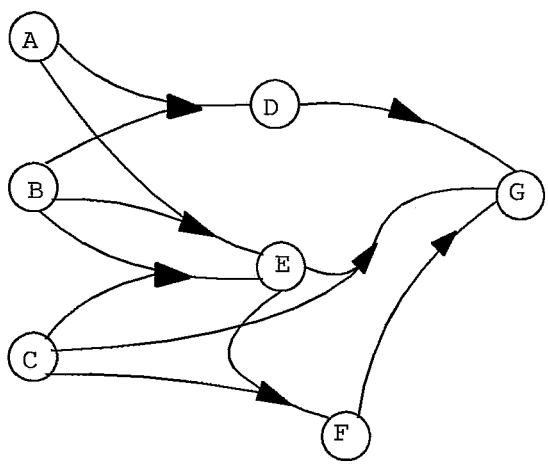


Figura IV.4: Hipergrafo equivalente a H

Ao trabalhar com os seus hipergrafos orientados, Ausiello muitas vezes se utiliza de um forma de representação definida por ele, os FD-grafos, que serão apresentados a seguir.

IV.2.2 FD-grafos

Os FD-grafos são introduzidos em [Ausiello 83], como uma proposta de representação das dependências funcionais em banco de dados relacional, de modo que as propriedades das dependências funcionais possam ser caracterizadas em termos das propriedades dos grafos.

É em [Ausiello 86] que os FD-grafos são apresentados como uma estrutura para a representação de hipergrafos orientados. O FD-grafo é um digrafo rotulado, em que os nós podem ser simples ou compostos, e os arcos cheios ou pontilhados, havendo sempre um arco pontilhado de um nó composto a cada um dos seus componentes.

Essa estrutura será utilizada na obtenção de representações minimais equivalentes, em [Ausiello 86], e também na obtenção de uma estrutura mais complexa, que forneça "aproximadamente" a mesma informação que o fechamento de um hipergrafo, no caso dinâmico de [Ausiello 90].

A seguir são apresentadas as definições de FD-grafo associado a um conjunto de dependências funcionais, e a de FD-grafo associado a um hipergrafo orientado. Após essas definições, são apresentados os conceitos de caminho, fechamento e cobertura para FD-grafos.

- Dependências Funcionais e FD-grafos

Seja um conjunto de atributos $U = \{ A, B, C, ... \}$. Um conjunto de dependências funcionais Σ é uma relação sobre $P(U) \times P(U)$, onde $P(U)$ é o conjunto das partes de U. Sejam X, Y subconjuntos de U. X determina funcionalmente Y ($X \rightarrow Y$) se e somente se para cada valor de X for associado ao mesmo precisamente um valor de Y ([Date 91]).

Dizemos ainda que um conjunto Σ está na sua forma reduzida se

(a) não existem duas dependências funcionais (DF's) $X \rightarrow Y$ e $X' \rightarrow Y'$ tal que $X = X'$, e

(b) para todas DF's $X \rightarrow Y$, $X \cap Y = \emptyset$.

Os conjuntos de dependências funcionais serão considerados na sua forma reduzida.

Def.IV.7: Dado um conjunto de dependências funcionais Σ sobre U, o FD-grafo $G(\Sigma)=(N_G, A_f, A_d)$ associado a Σ é o digrafo com a função de rotulamento de nós w: $N_G \rightarrow P(U)$, tal que:

(i) para cada atributo $A \in U$ existe em N_G um nó rotulado A (nó simples);

(ii) para cada dependência $X \rightarrow Y$ em Σ onde $| X | > 1$, existe em N_G um nó rotulado X (nó composto);

(iii) para cada dependência $X \rightarrow Y$ em Σ onde $Y = A_1...A_k$, existem em A_f arcos (cheios) do nó X a cada um dos nós $A_1, ..., A_k$;

(iv) para cada nó composto $i \in N_G$ $A_1...A_k$ existem em A_d arcos (pontilhados) do nó i a cada um dos nós simples $A_1, ..., A_k$ (componentes de i).

Por exemplo, para o conjunto de dependências funcionais $\Sigma = \{ A \rightarrow BC, C \rightarrow D, BD \rightarrow E \}$, o FD-grafo associado seria o da figura IV.5.

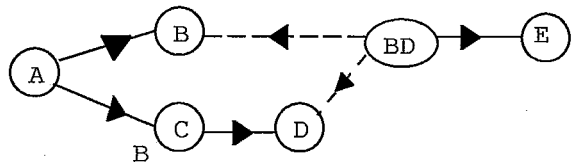


Figura IV.5: O FD-grafo associado a Σ

- Hipergrafos Orientados e FD-grafos

Apresentaremos agora a definição de FD-grafo associado a um hipergrafo orientado.

Def.IV.8:

Dado um hipergrafo $H(N, E)$, o FD-grafo associado a H é o digrafo rotulado G(H)(N_G, A_f, A_d), onde:

$N_G = N \cup N_c$ é o conjunto de nós, onde N é o conjunto de **nós simples** e $N_c = \{ X \subseteq N$ tq X é um conjunto origem em H com $\mid X \mid > 1 \}$ é chamado conjunto de **nós compostos**. Cada nó em X é um **componente** do nó composto X;

$A_f \subseteq N_G \times N = \{ (X, i), (X, i) \in E \}$ é o conjunto de arcos cheios; e

$A_d \subseteq N_G \times N = \{ (X, j)$ tq $X \in N_c$ e $j \in X \}$ é o conjunto de arcos pontilhados.

Exemplo: na figura IV.6 temos o FD-grafo associado ao hipergrafo H da figura IV.1, onde { AB, BC, EC } é o conjunto de nós compostos, A_d= { (AB,A), (AB,B), (BC,B), (BC,C), (CE,C), (CE,E) }, e A_f= { (AB,D), (AB,E), (BC,E), (CE,F), (D,G), (F,G) }.

- FD-caminho

Def.IV.9:

Dado um FD-grafo $G(H)$ =(N_G, A_f, A_d) e dois nós $i, j \in N_G$, um FD-caminho de i a j é o subgrafo minimal G'= (N_G', A_f', A_d') de G tal que $i, j \in N_G$' e uma das seguintes condições é atendida:

(*i*) $(i, j) = A_f' \cup A_d'$, ou

(*ii*) j é um nó simples, e existe um nó k tal que $(k, j) \in A_G' \cup A_d'$ e existe em $G'(H)$ um FD-caminho de de i a k, ou

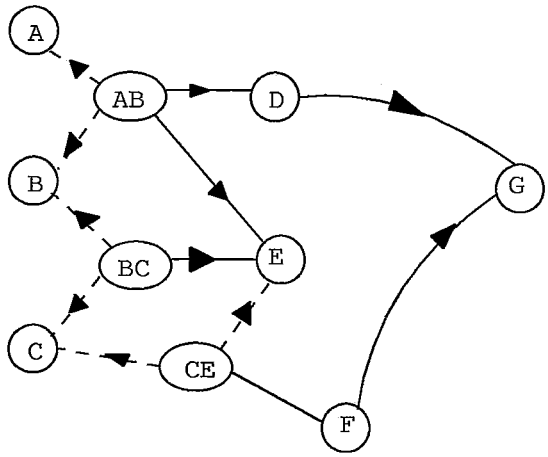


Figura IV.6: O FD-grafo associado ao hipergrafo H na fig.IV.1

(iii)j é um nó composto com componentes $m_1, ..., m_r$ e os arcos (pontilhados) $(j, m_1), ..., (j, m_r) \in A_d$' e existem FD-caminhos de i a m_q em $G(H)$', para $q = 1, ..., r$ e $m_q \neq i$.

Exemplo: um FD-caminho BC a G no FD-grafo $G(H)$ da fig.IV.6 pode ser visto na figura IV.7.

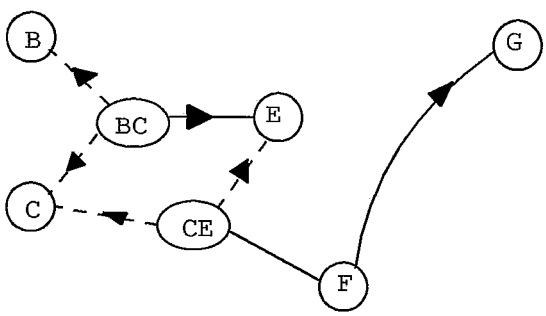


Figura IV.7: Um FD-caminho de BC a G

Um FD-caminho de i a j é chamado **pontilhado** se todos os arcos com origem em i no FD-caminho são pontilhados. Caso contrário ele é dito **cheio**.

- FD-fechamento

Def.IV.10:

Dado um FD-grafo G=(N_G, A_f, A_d) define-se o FD-fechamento de G como o digrafo rotulado G^+=(N_G, A_f^+, A_d^+), onde:

A_d^+ = { (i,j) tq $i, j \in V$ e existe um FD-caminho pontilhado entre i e j em G};

A_f^+ = { (i,j) tq $i, j \in V$, (i,j) não $\in A_d^+$, e existe um FD-caminho cheio entre i e j em G }.

O FD-fechamento do FD-grafo da figura IV.6 pode ser visto na figura IV.8.

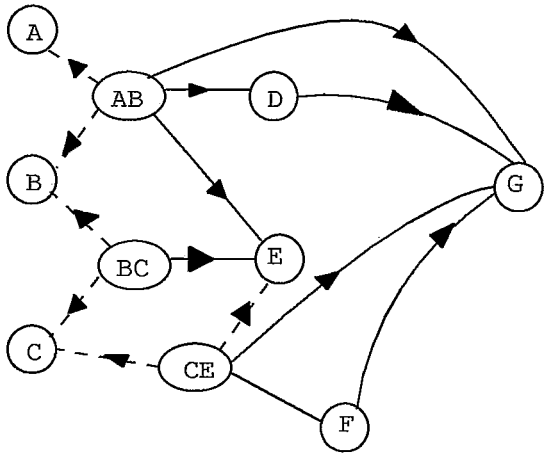


Figura IV.8: O FD-fechamento do FD-grafo da fig. IV.6

- FD-fechamento e Dependências Funcionais

Para estabelecer a relação entre o FD-fechamento e dependências funcionais (DF's), necessitamos do conceito de fechamento de um conjunto de dependências funcionais.

Dado um conjunto Σ de DF's, denota-se por Σ^+ o fechamento de Σ com relação às regras de inferência de Armstrong:

Reflexividade: Se $Y \subseteq X$, então $X \rightarrow Y$.

Transitividade: Se $X \rightarrow Y$ e $Y \rightarrow Z$, então $X \rightarrow Z$.

União: Se $X \rightarrow Y$ e $Y \rightarrow Z$, então $X \rightarrow YZ$.

Teorema IV.1: Seja $G(\Sigma) = (N_G, A_f, A_d)$ o FD-grafo associado ao conjunto Σ de DF's, e seja $G^+(\Sigma) = (N_G^+, A_f^+, A_d^+)$ o seu fechamento. Um arco (i, j) só está em $A_f^+ \cup A_d^+$ se e somente se $w(i) \rightarrow w(j)$ está em Σ^+.

A prova do teorema anterior está em [Ausiello 83].

O teorema IV.1 é relevante na medida que distingue no fechamento Σ^+ de Σ quais seriam as dependências funcionais realmente significativas. No caso, as DF's em Σ^+ que têm arcos correspondentes em $G^+(\Sigma)$, têm como característica o fato de envolverem somente os conjuntos de atributos que estão no conjunto de DF's original Σ. São desconsideradas DF's menos importantes que eram obtidas através da regra de reflexividade. Isso evitaria o crescimento exponencial que ocorre no fechamento.

- FD-fechamento e Hipergrafos Orientados

Pela definição de fechamento para um hipergrafo orientado, def.IV.5, também nesse caso o número de arcos no fechamento é exponencial no número de nós.

O FD-fechamento é uma representação sucinta do fechamento de um hipergrafo. Consideremos o teorema a seguir.

Teorema IV.2:

Seja H um hipergrafo e $G(H)$=(N_G, A_f, A_d) o FD-grafo associado a H. Dados um par de nós $i, j \in N_G$, onde j é um nó simples e $j \neq i$, o arco (i, j) está em $G^+(H)$ se e somente se existe um hiperarco correspondente em H^+.

A prova desse teorema está em [Ausiello 86].

Dessa forma, dado um hipergrafo H, o FD-fechamento fornece o fechamento de qualquer conjunto de nós X tal que $\mid X \mid = 1$, que correspondem aos nós simples, ou X é o conjunto origem de algum hiperarco em H, que correspondem aos nós compostos.

- FD-cobertura

Def.IV.11:

Dado um FD-grafo $G(H)$, o FD-grafo $G(H')$ é dito uma FD-cobertura de $G(H)$ se H' é equivalente a H, onde H' é o hipergrafo representado por $G(H')$.

A noção de FD-cobertura é utilizada na próxima seção ao se falar de representações equivalentes minimais.

IV.2.3 Minimalidade e Equivalência Forte

Como foi visto anteriormente, a definição de equivalência para hipergrafos orientados depende diretamente do conceito fechamento, já que dois hipergrafos são ditos equivalentes se eles possuem o mesmo fechamento. Isso faz com que o problema de obtenção de representação minimal no caso de um hipergrafo H seja mais complexo do que para grafos, já que o número de hiperarcos no fechamento de H é sempre exponencial no número de nós.

A utilização dos FD-grafos como forma de representação e do FD-fechamento permite, em alguns casos, determinar hipergrafos equivalentes minimais sem que ocorra a explosão exponencial do fechamento de hipergrafos. Isso porque no FD-fechamento o crescimento é no máximo quadrático.

As etapas para a obtenção de uma representação minimal H' para o hipergrafo orientado H são:

1) Obter a FD-grafo representação G_H de H.

2) Determinar o FD-fechamento G_H^+ de G_H, ao invés de H^+.

3) Utilizar esse FD-fechamento na obtenção de uma representação reduzida $G_{H'}$ de G_H (que é uma cobertura minimal de G_H) correspondente a H'.

Essas etapas estão detalhadas em [Ausiello 86].

Os conceitos de minimalidade são os mais diversos, já que vão depender basi-

camente do(s) parâmetro(s) do hipergrafo que se deseja minimizar. Antes disso, é necessário relacionar os elementos que caracterizam **redundância** num dado hipergrafo $H = (N, E)$.

(1) um nó é dito redundante num conjunto origem X de H se $j \in X$ e $(X - j, j) \in H^+$;

(2) um hiperarco (X, j) é dito redundante se (X, j) está no fechamento obtido de H mediante a eliminação de (X, j).

Assim sendo, um hipergrafo é dito não redundante se não tiver nenhuma ocorrência de nós ou hiperarcos redundantes.

Seja H um hipergrafo não redundante. De acordo com as principais definições de minimalidade, H é:

Mínimo-Origem (MO) se não existir outro hipergrafo equivalente a H com menor número de conjuntos origem;

Mínimo-Hiperarco (MH) se não existir outro hipergrafo equivalente a H com menor número de hiperarcos;

Mínimo-Origem-Hiperarco (MOH) se não existir outro hipergrafo H' equivalente tal que $s' + h \text{ '} < s + m$;

Mínimo-Área-Origem (MAO) se não existe outro hipergrafo equivalente com menor área de origem;

Ótimo (O) se não existe outro hipergrafo com tamanho menor $(a + h)$.

As relações entre os diferentes critérios de minimalidade estão na figura IV.9.

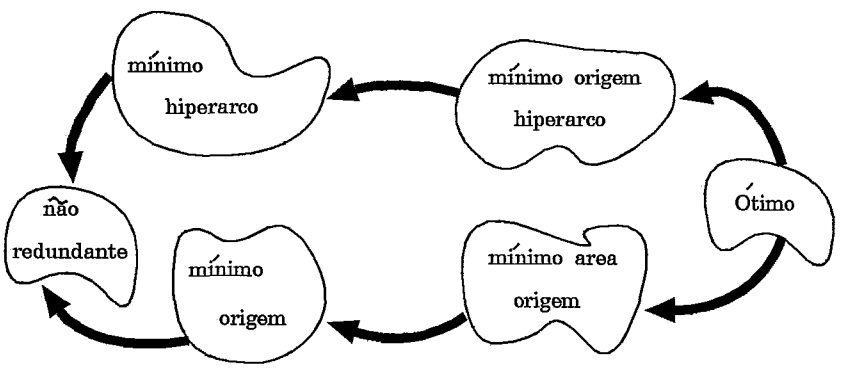


Figura IV.9: Relações entre critérios de minimalidade

Uma noção de equivalência mais forte do que a considerada até aqui é encontrada em [Ausiello 85], a qual necessita das seguintes definições:

Def.IV.12: **Núcleo (Kernel)**

Um núcleo de um hipergrafo orientado $H = (N, E)$ é uma família minimal de

conjuntos origem $K = S_1, ..., S_k$, tal que para cada conjunto origem X em H existe um conjunto origem S_i em K para o qual X está em S_i^+.

Além disso, dados dois hipergrafos $H = (N, H)$ e $H' = (N, H')$ o núcleo K de H é dito **equivalente** ao núcleo K' de H', se para cada conjunto origem S_i em K existe um conjunto origem S'_j em K' tal que $S_i^+ = (S'_j)^+$ e vice-versa.

Teorema V.3: Todos os núcleos de um hipergrafo são equivalentes dois a dois e possuem o mesmo número de conjuntos origem. A prova deste teorema (como dos demais que se seguem) será omitida aqui.

Def.IV.13: Equivalência Forte

Dois hipergrafos orientados H e H' são ditos **fortemente equivalentes** se e somente se eles são equivalentes e seus núcleos são equivalentes.

Na figura IV.10, todos os hipergrafos são equivalentes, mas só os de (a) e (b) são fortemente equivalentes.

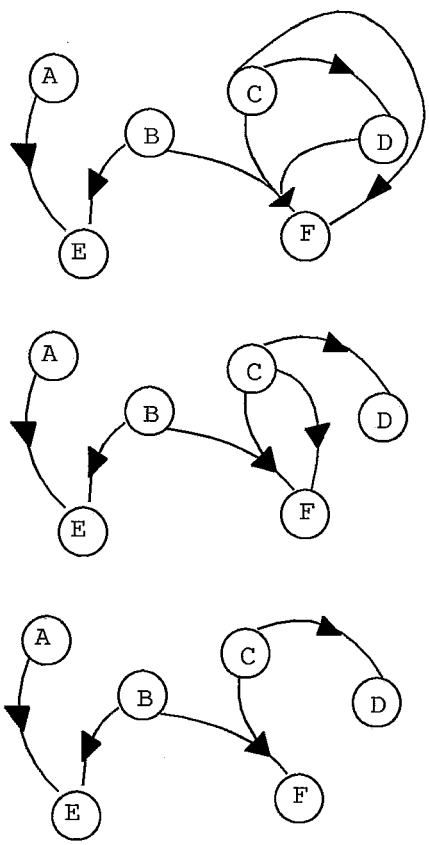


Figura IV.10: Hipergrafos equivalentes e fortemente equivalentes

IV.2.4 Correspondência de Terminologias e Extensões

Aqui será feita a correspondência, sempre que possível, entre as definições apresentadas nas seções anteriores e as definições do capítulo de introdução desse trabalho, que adota a terminologia encontrada em [Gallo 93].

O **hiperarco** (X, i), onde X é um subconjunto de nós (**os vértices**) e i um nó simples, corresponde ao nosso **B-arco**, lembrando que um B-arco é uma hiperaresta orientada de duas camadas (rabo, cabeça), com a cardinalidade do conjunto cabeça igual a 1.

Como por definição os hipergrafos orientados com hiperarestas de 2-camadas são os 2-grafos e, mais especificamente, os 2-grafos formados unicamente por B-arcos são os B-grafos, a classe definida e referida pelos autores é na realidade uma subclasse particular de hipergrafos orientados, a classe **B-grafos.**

O **conjunto origem** X do hiperarco (X, i), como é chamado o subconjunto de nós que se encontra na camada mais à esquerda do hiperarco, corresponde ao **rabo** do B-arco.

Em [Gallo 93] a definição de conectividade é generalizada para um conjunto de vértices da seguinte maneira: em um BF-grafo $H(V, E)$ o nó y é dito B-conectado (F-conectado, BF-conectado) a um conjunto de nós S, $S \subseteq V$, se y é B-conectado (F-conectado, BF-conectado) a s no BF-grafo H_s, obtido de H pela adição de um novo nó origem s e um arco (s, x) para cada $x \in S$. Da mesma forma, define-se a conexão de um conjunto de nós T a um vértice origem x.

O hipergrafo minimal H_C tal que existe em H_C um **hipercaminho** de um subconjunto X a um nó i é um B-caminho de X a i. Pela definição de hipercaminho, um nó v só estará em H_C se v está conectado a X, e para qualquer aresta e em H_C, todos os vértices de *rabo*(e) e *cabeça*(e) estão em H_C. O hipergrafo H tal que o FD-grafo associado a H é um FD-caminho é também um B-caminho, já que a minimalidade é garantida pela definição de FD-caminho.

Não existe uma definição anterior de fechamento para toda a classe dos hipergrafos orientados. A idéia é , considerando a propriedade caracterizada no fechamento dos B-grafos, estender esse conceito para a classe BF-grafos. No fechamento de um hipergrafo de Ausiello, a propriedade considerada para a inclusão de (X, i) no fechamento de H é i ser B-conectado a H.

Uma definição de fechamento estendida para a BF-grafos seria: Dado o BF-grafo H(V,E), o B-fechamento de H é o par $H^+(V, E^+)$, onde: $(X, i) \in E^+$, $X \subset V$, $i \in V$, se i é B-conectado a X em H.

Analogamente, pode ser definido o F-fechamento.

A definição de equivalência pode ser então estendida à classe BF-grafos de forma bastante imediata. Dois BF-grafos são ditos B{F}-equivalentes quando possuem o mesmo B{F}-fechamento.

O FD-grafo é um digrafo rotulado, com nós simples e compostos. É possível enxergar a representação de B-grafos por FD-grafos como um caso especial da representação de hipergrafos por digrafos do capítulo anterior, em que os rótulos dos nós eram nós arestas ou vértices, de acordo com a sua origem no hipergrafo a ser representado. Seria interessante pensar se haveria, no caso da 2-grafos, uma estrutura intermediária entre o FD-grafo e o digrafo mais geral.

IV.3 Manutenção dinâmica de hipergrafos orientados

A seguir serão relacionados as estruturas e os algoritmos desenvolvidos em [Ausiello 90] para o caso de manutenção dinâmica de hipergrafos orientados (os nossos B-grafos). Mais especificamente, o problema considerado é a manutenção do fechamento transitivo de um dado hipergrafo orientado H durante a inserção de novos hiperarcos. Problema análogo já foi estudado para grafos, mas a normalmente esperada extensão dos resultados obtidos para estes é dificultada pela explosão exponencial que ocorre no fechamento de hipergrafos. Desse modo, para que o problema torne-se tratável a representação considerada será a FD-grafo, assim como o fechamento relativo a esta, o FD-fechamento.

As definições de fechamento de hipergrafos bem como a representação FD-grafo e o seu respectivo FD-fechamento foram vistas na seção anterior, à exceção da orientação dada aos arcos pontilhados, anteriormente no sentido nó composto para nó componente e agora considerada no sentido inverso, do nó componente para o nó composto. Isso é feito unicamente com o intuito de unificar a codificação dos algoritmos, sem implicar em qualquer conseqüência no que foi visto anteriormente para FD-grafos.

Após cada inserção de novo hiperarco, as estruturas devem ser atualizadas de forma a manter o FD-grafo correspondente ao novo hipergrafo.

O problema de manter as informações do fechamento de um hipergrafo pode ser particionado em dois subproblemas. O primeiro é a preocupação exclusiva com os nós simples, ou seja, com a existência de caminho entre dois nós simples x e y. O outro problema seria a preservação da alcançabilidade de nós compostos, ou seja, se a partir de um determinado conjunto origem um determinado nó simples é ou não

alcançado. Os problemas serão tratados em separado, com o grau de dificuldade e complexidade aumentando muito do primeiro para o segundo caso. No presente trabalho, relacionaremos os resultados para os nós simples.

IV.3.1 Fechamento de Nós simples

Apresentaremos a estrutura de dados e as rotinas para identificação de existência e recuperação do caminho de i a j, i e j nós simples.

Será utilizada a estrutura de dados *Last*, definida para qualquer par de nós simples, na qual é feita a verificação de existência de um hipercaminho entre i e j. No caso da consulta ser positiva, ou seja, de existir tal caminho, ele pode ser facilmente obtido através das rotinas descritas logo após a estrutura Last. Por enquanto, ainda está sendo considerado o caso estático, mas essas estruturas e rotinas que serão também utilizadas para o caso dinâmico.

Last: *Last* é um vetor de tamanho $n_s \times n_s$, onde $Last[x, y]$ aponta o último nó simples ou composto (excluindo y) de um FD-caminho do nó simples x ao nó simples y. Se não houver tal caminho o conteúdo de $Last[x, y] = \text{nil}$.

As seguintes rotinas são utilizadas na recuperação do caminho, quando houver, de i a j, começando em j e fazendo a reconstrução do caminho de trás pra frente. No caso dos nós compostos todos os antecessores (nós componentes) serão examinados, enquanto para os nós simples apenas um o será.

```
rotina hipercaminho(i, j: no_simples);
var
    hcaminho: set_of arcos; cheios ou pontilhados
inicio
    se Last[i, j] ≠ nil então
    inicio
        hcaminho := ∅;
        desmarcar todos os nós marcados (se houver);
        marcar j;
        FD-caminho(i, j);
        retorna hcaminho;
    fim;
fim;
```

A rotina hipercaminho dá inicio à recuperação de um caminho de i a j, verificando primeiro se existe tal caminho, incluindo j no caminho, e chamando a rotina que obtém um FD-caminho de i a j.

```
rotina FD_caminho(i:no_simples, j:no);
var
    w: no;
inicio
    se i ≠ j então
        para cada nó w em FLast(i,j) faça
        inicio
            inclua (w, j) em hcaminho;
            se w é não marcado então
            inicio
                marcar w;
                FD_caminho(i, w);
            fim;
        fim;
fim;
```

A rotina FD-caminho inclui a(s) aresta(s) que precede(m) j no caminho, que no caso de j ser composto são em número igual ao dos componentes de j. Serão incluidos no caminho os vértices v que estão na origem dessa(s) aresta(s) e que são obtidos pela função FLast. Recupera-se enntão os FD-caminhos de i até cada um desses vértices.

```
função FLast(i:no_simples, j: no) : lista de no_simples;
inicio
    se j é simples então
        retorna lista contendo Last[i, j]
    senão
        retorna lista de componentes de j;
fim;
```

A FLast devolve quais vértices precedem j no caminho de i a j, que no caso de j ser simples é o vértice único em $Last[i,j]$ e, no caso de j ser composto, são os vértices componentes de j.

IV.3.2 Nós simples: manutenção dinâmica do fechamento

Para a preservação da informação de alcançabilidade dos nós simples será necessária a utilização de estruturas auxiliares e novas rotinas que as manipulem, além do vetor *Last* da seção anterior.

Existem várias aplicações que implicam a existência de inúmeras arestas com mesmo conjunto origem. Dessa forma, para manter um único nó composto por conjunto X distinto, é necessário verificar, ao inserir uma nova aresta (X,i), se já existe o nó composto para X.

Os nós compostos serão mantidos numa árvore de busca binária, objetivando-se evitar redundância na representação. A árvore considerada é uma árvore AVL, árvore de busca binária que se caracteriza por ter a diferença entre as alturas das duas subárvores de um nó igual a no máximo 1. (ver [Horowitz 84]). Serão também consideradas as operações de busca e inserção numa árvore AVL, que possuem complexidade, no caso das chaves poderem ser testadas em tempo constante, $O(\log n_T)$, onde n_T é o número de nó na árvore.

Assim sendo, consideraremos:

T_c: árvore AVL contendo os nós compostos do hipergrafo que deverá ser consultada quando uma nova aresta (X,i) for inserida. Serão também consideradas as operações básicas relacionadas:

.Busca_AVL(item, tree): retorna um ponteiro para o ítem requisitado, ou o valor NIL se não for encontrado;

.Insere_AVL(item, tree): insere o ítem na árvore.

Reach_y: para cada nó y em N_s existirá um array Reach_y[1..n_s] tal que:

$$Reach\_y[x] = \begin{cases} 0, & \text{se existe um FD-caminho no FD-grafo de x a y} \\ 1, & \text{caso contrário} \end{cases}$$

Isso deverá ser mantido ao longo de toda a execução. Além disso, apesar de redundante no caso de nós simples, já que Reach_j[i] = 1 sse Last[i,j] = NIL, a estrutura é assim definida e mantida visando facilitar a compreensão dos algoritmos.

Reach_x: para cada nó composto $X(x_1, x_2, ..., x_q)$ existirá um array Reach_x[1..n_s] tal que:

$$Reach\_x[i] = \sum_{k=1..q} \{Reach\_x_k[i]\},$$

para todo nó simples i.

Isso também é conservado ao longo de toda a execução.

Reach_x[i] contabiliza, para o nó composto x, o número de componentes x_k de x para as quais não existe caminho de i a x_k. Uma aresta (X, j) só poderá ser incluída num caminho de i a j quando Reach_x[i] for igual a zero.

L_f(x) e L_d(x): são as listas de adjacências que implementam o FD-grafo, compostas respectivamente pelos arcos cheios e arcos pontilhados que partem de x. Elas são definidas para todos os nós (simples e compostos), sendo que as L_d(x) dos nós compostos serão listas vazias.

As rotinas que atualizam as informações sobre alcançabilidade dos nós simples após a inserção de um novo arco são descritas a seguir.

```
função Composto(X: set_of n_simples) : no_composto;
var
    x: no_composto;
    i, j : no_simples;
inicio
    x:= Busca_AVL(X, T_c);
    se x ≠ nil então
    inicio
        criar um novo no_composto X apontado por x;
        Insere_AVL(x, T_c);
        para cada { nó simples } i em N_s faça
            Reach_x[i] :=∑_{j∈X} Reach_j[i];
        para cada { nó simples } i em X faça
            insere x em L_d(i);
    fim;
    retorna x;
fim;
```

A função Composto retorna, se houver, o nó composto correspondente a X, localizado mediante uma busca na árvore T_c. Se não houver tal nó (a busca retornar x=NIL), então cria-se um novo nó composto, com endereço em x, inserindo-o em T_c. Além disso, contabiliza-se em Reach_x[i], para cada nó simples i, o número de componentes de X não alcançadas de i. Por fim, insere-se x na L_d de cada componente de X.

```
rotina Insere(X: set_of no_simples, y: no_simples);
var
    x: no;
    i: no_simples;
inicio
    se | X | = 1 então
        x:= o elemento de X
    senão
        x:= Composto(X);
    insere y em L_f(x);
    para cada { nó simples }i em N_s faça
        se Reach_x[i]=0 então Fechamento(i, x, y)
fim;
```

É a rotina que atualiza as estruturas quando uma aresta (X, i) é inserida. Se X é composto, obtém em x o endereço retornado pela função Composto, caso contrário x é o único elemento de X. Insere y na L_f de x. Para todos os nós simples i para os quais não havia caminho de i a y, faz-se o fechamento, utilizando a rotina abaixo.

```
rotina Fechamento(i:no_simples; x, y: no);
var
    w: no;
inicio
    se Reach_y[i] ≠ 0 então
    inicio
        Reach_y[i] := Reach_y[i] -1;
        se Reach_y[i] = 0 então
        inicio
```

```
            se y  e um nó simples então
                        Last[i, j] := x;
                    para cada w em L_f(y) ∪ L_d(y) faça
                        Fechamento(i, y, w);
                fim;
        fim;
    fim;
```

A rotina fechamento atualiza as informações dos vetores *Reach* e *Last* após a inserção de uma aresta. A primeira chamada dentro da rotina Insere, Fechamento(i, x, y), feita ao verificar-se existência de um caminho entre i e x, busca identificar quais as novas conexões que surgiram com a inserção de (x, y), a partir do nó x.

Alguns dos resultados aqui apresentados podem ser estendidos para BF-grafos. Dentre eles, a obtenção de uma representação não redundante dos conjuntos cabeça e *rabo* das arestas, que será apresentado na próxima seção.

Outra possibilidade, é a obtenção de estruturas e rotinas para a manutenção dinâmica do fechamento de nós simples em um BF-grafo, que não consideraremos aqui.

IV.4 Representação utilizando nós compostos para casos particulares de 2-grafos

Em algumas aplicações da classe 2-grafos pode ocorrer que um mesmo conjunto X seja *rabo* ou *cabeça* de várias arestas, para $X \subset V$.

Consideremos a representação de 2-grafos pelas quatro listas FS(v), BS(v), Cab(e) e Rab(e), da seção III.3 do capítulo anterior. Assim sendo, haverá a repetição de toda a lista de componentes de X nas listas Cab(e) (Rab(e)) que tenham X como seu conjunto *cabeça* (*rabo*).

Apresentaremos a seguir um esquema de representação que utiliza a noção de nó composto dos trabalhos de Ausiello.

O que é feito por Ausiello na representação por FD-grafo é criar um nó composto para cada conjunto distinto X, tal que X é o conjunto $rabo(e)$ (conjunto origem) de alguma hiperaresta e com $\mid X \mid > 1$.

A manutenção desses nós compostos numa árvore AVL permite, ao inserir uma

nova hiperaresta (X, i), identificar se já existe o nó composto correspondente a X, localizando-o em caso positivo.

Busca e inserção numa árvore AVL são feitas em tempo $O(temp\_comp \times \log n_c)$, onde n_c é o número de nós na árvore, no nosso caso igual ao número de nós compostos, e $temp\_comp$ é o tempo de comparação de duas chaves, que no nosso caso será igual a $max(\mid X \mid)$. A cardinalidade do conjunto X é, no pior caso, igual a n.

Assim sendo, para que a manutenção seja feita de forma eficiente, é necessário que exista uma constante K, tal que $\mid rabo(e) \leq K$, $\mid cabeça(e) \mid \leq K$, para qualquer aresta e no hipergrafo.

Considerando os 2-grafos com a restrição acima, as estruturas para a representação de um 2-grafo $H(V, E)$, evitando a redundância na representação dos conjuntos *cabeça* e *rabo* das arestas, seriam:

FS(v) e BS(v): são as listas encadeadas contendo respectivamente as arestas de $FS(v)$ e de $BS(v)$, para cada vértice $v \in V$.

VetAresta: vetor de m posições, onde $VetAresta[i]$ é contituído dos seguintes campos:

UCab: contém o elemento de $cabeça(e_i)$ se a $\mid cabeça(e_i) \mid = 1$, ou -1, caso contrário;

URab: contém o elemento de $rabo(e_i)$, se a $\mid rabo(e_i) \mid = 1$, ou -1, caso contrário;

PCompCab: aponta o nó composto correspondente a X, onde X é igual ao conjunto $cabeça(e_i)$, $\mid X \mid > 1$, ou NIL, caso contrário;

PCompRab:aponta o nó composto correspondente a X, onde X é igual ao conjunto $rabo(e_i)$, $\mid X \mid > 1$, ou NIL, caso contrário.

T_c: árvore AVL contendo os nós compostos do 2-grafo. Serão também consideradas as operações básicas relacionadas:

.Busca_AVL(item, tree): retorna um ponteiro para o ítem requisitado, ou o valor NIL se não for encontrado;

.Insere_AVL(item, tree): insere o ítem na árvore.

Cada nó composto na árvore possui dois campos:

l_comp: aponta a lista encadeada com os componentes do nó;

n_oc: contador do número de ocorrências do conjunto X considerado como cabeças ou rabos de arestas.

Para as operações do conjunto básico do capítulo II, a complexidade da maioria das rotinas não será diferente do que as das 4-Listas do capítulo III.

As operações feitas utilizando-se as listas BS(v) e FS(v) não se alteram.

As operações que utilizavam a lista Cab(e) (Rab(e)) utilizarão a lista l_comp do nó x, cujo endereço está em $VetAresta[e].PCompCab$ ($VetAresta[e].PCompRab$), no caso do conjunto ter cardinalidade maior do que 1. No caso do conjunto ter um único vértice, este é obtido em $VetAresta[e].UCab$ ($VetAresta[e].URab$).

As operações que terão as complexidades de suas rotinas alteradas são as de inserção e remoção de aresta, e remoção de vértice.

Ao inserir uma aresta (X, Y), $\mid X \mid > 1$, $\mid Y \mid > 1$ será necessário fazer uma busca na árvore para identificar se o nó X já existe, e inserí-lo no caso dele não existir, ou incrementando o seu número de ocorrências, caso contrário. Faz-se a mesma coisa para o nó Y. Isso é feito em $O(\log n_c)$.

A remoção de uma aresta e poderia implicar a remoção de um (ou dois) nó(s) composto(s), se o número de ocorrências do nó correspondente a *cabeça(e)* (*rabo*(e)), ao ser decrementado na remoção, ficasse igual a 0. A remoção de um nó de uma árvore AVL é feita em $O(\log n_c)$.

Como a remoção de um vértice implica a remoção das arestas em $FS(v)$ e $BS(v)$, esta operação teria também sua complexidade modificada.

Uma alternativa seria na remoção de arestas não fazer a remoção do nó cujo número de ocorrências fosse zerado. Dependendo da aplicação, essa solução seria mais vantajosa, compensando manter alguns poucos nós compostos considerados "deletados" na árvore.

IV.5 Grafos E/Ou e Grafos E/Ou Generalizados

Procuraremos aqui caracterizar Grafos E/Ou e Grafos E/Ou Generalizados como casos particulares da classe BF-Grafos. Serão primeiramente apresentados os principais conceitos em Grafos E/Ou e Grafos E/Ou Generalizados. Em seguida, estabelece-se a correspondência entre esses conceitos e os conceitos já conhecidos em hipergrafos, sendo ao final relacionados os problemas e respectivas soluções considerados.

As definições encontradas nas fontes não obedecem a uma notação padrão. Isso cria a necessidade de alguns passos intermediários que estabeleçam equivalências entre certas representações.

IV.5.1 Grafos E/Ou

É um tipo de estrutura utilizada na obtenção da solução de problemas que podem ser resolvidos mediante a decomposição em um conjunto de problemas menores (subproblemas). A utilização de um grafo E/Ou só é viável quando esses subproblemas forem independentes entre si.

Um hiperarco orientado E é gerado por uma dessas decomposições, ou reduções, e pode ter um número qualquer de nós sucessores como destino. Nesses caso, só haverá uma solução quando **todos** os nós apontados pelo mesmo arco estiverem resolvidos. Um determinado nó pode ser origem de diversos arcos, que representam formas **alternativas** de se tentar resolver o problema, motivo pelo qual o grafo é denominado E/Ou.

A solução em um grafo E/Ou equivale à obtenção de um caminho entre o **nó inicial** e os **nós terminais**, que são os nós correspondentes a estados de solução.

É muito importante observar que em muitos casos nao é suficiente chegar a um dos nós terminais. Isso porque cada ramificaçao de um arco E tem que chegar ao seu nó de solução, que pode ou nao coincidir com os das demais ramificações e arestas.

IV.5.2 Grafos E/Ou Generalizados

Em Grafos E/Ou (GEO) poderíamos ao invés de representar uma decomposição (redução) de problemas por um hiperarco orientado E(problema, subproblemas), representá-la por um nó E, um arco orientado (problema, E) e um arco orientado (E, subproblema) para cada subproblema. Com essa nova representação, para se manter a equivalência entre representações, a seguinte restrição se impõe: um nó E tem sempre um e somente um antecessor. Além disso, um nó E é dito solucionado somente quando todos os seus sucessores estiverem solucionados. Ainda que essa nova representação nada acrescente aos algoritmos de obtenção de solução nos GEO, ela facilitará o entendimento do conceito de generalização de Grafos E/Ou.

Um Grafo E/Ou Generalizado é uma estrutura também utilizada na solução de problemas decomponíveis. Apresenta como principal característica o relaxamento da restrição de independência entre subproblemas. Com relação à estrutura do grafo, isso significa permitir a um nó E ter mais de um antecessor. Desse modo o GEOG se presta a representação de problemas para os quais a utilização de um GEO simples era inadequada.

Define-se um Grafo E/Ou Generalizado (GEOG) como g = (O,T,A,E,S), onde:

O = conjunto de nós Ou não terminais;

T = conjunto de nós Ou terminais;

A = conjunto de nós E;

E = subconjunto de (O ∪ T) × A ∪ A ×(O ∪ T), cujos elementos sao arcos orientados;

S = S ∈ O, S é o nó inicial.

Um nó E a_k é um operador de decomposição (redução) aplicável a um **conjunto** de problemas de entrada a serem decompostos (reduzidos) em um conjunto de problemas denominados problemas de saida. Reforça-se aqui a observação da diferenca com relação ao GEO simples, pois nesse último o operador de decomposição é aplicável a um **único** problema.

Exemplo: O grafo na figura IV.11 é um Grafo E/Ou Generalizado. Nós Ou terminais em letra minúscula, nós Ou não terminais em maiúscula. Os nós com dígitos são nós E.

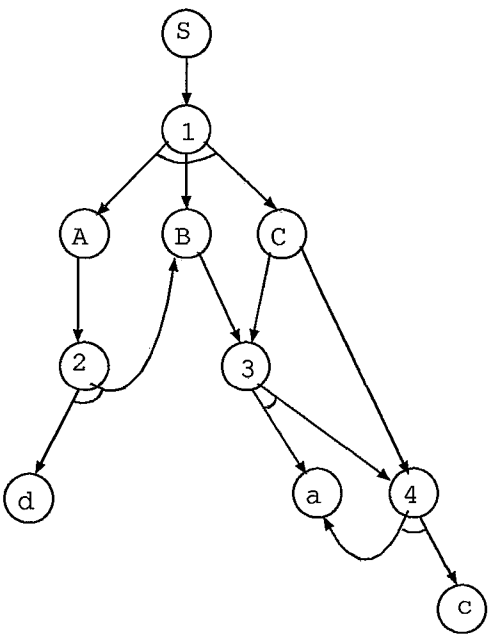


Figura IV.11: Um Grafo E/Ou Generalizado

Solução:

A partir do GEOG induzido por um determinado problema inicial (nó inicial) S deseja-se obter um subgrafo solução que mostre estar S resolvido. Um nó é dito resolvido nos seguintes casos:

(i) Nós terminais são nós resolvidos.

(ii) Um nó E é dito resolvido se e somente se todos os seus sucessores são resolvidos.

(iii) Um nó não terminal Ou é solucionado se e somente se pelo menos um dos seus sucessores é solucionado.

A construção do algoritmo para obtenção de um grafo solução esta baseada na noção de grafo solução potencial (GSP), que será definido em seguida. Seja g' o conjunto de GEOG's acíclicos obtidos descartando-se os nós Ou do GEOG original g. Um grafo solução potencial (GSP) é qualquer subgrafo s de g' tal que:

(i) O nó inicial S esteja em s.

(ii) Se o nó n está em s então:

(a) se n é um nó Ou não terminal entao **exatamente** um sucessor n' de n está em s;

(b) se n é um nó E então todos os sucessores de n estão em s.

Um GSP em que S esteja resolvido é um **grafo solução**. A definição de resolubilidade de um nó implica em todos os nós extremos de um grafo solução serem terminais.

Observação: por conta de diferenças entre referências , nos GEO simples alguns dos nós terminais eram ditos soluções, mas não necessariamente todos, enquanto nos generalizados todos os nós terminais são resolvidos por definição. Isso é explicado pelas diferentes aplicações objetivadas em cada caso.

IV.5.3 GEOs e GEOGs como Hipergrafos

Serão aqui explicitadas as equivalências entre determinados conceitos em hipergrafos e os que aparecem em Grafos E/Ou e Grafos E/Ou Generalizados.

Para o caso de um Grafo E/Ou simples G a correspondência é obtida de forma bastante imediata, baseada na própria definição do hiperarco orientado E, que é na verdade um 2-arco (hiperarco com duas camadas) e, mais do que isso, é um F-arco ($\mid rabo(E) \mid = 1$, $\mid cabeça(E) \mid \geq 1$).

Como todos os hiperarcos de G são F-arcos, G é um F-Grafo.

Reunindo-se os nós terminais de G em um único nó T (ou fazendo com que todos os nós terminais apontem para um único nó terminal T), o grafo solução é um F-caminho de S a T.

O GEOG já é um pouco mais complexo, requerendo uma "visão" do grafo que facilite estabelecer as correspondências. No caso, ao invés de vários arcos partindo do nó E para os subproblemas, considerar um hiperarco único (um F-arco), como era feito para os GEO simples, só que ao invés de ir do problema direto aos subproblemas, ele tem origem no nó E e destino os subproblemas. Da mesma forma, ao

invés de vários arcos dos problemas principais (de entrada) para o nó E, considera-se um hiperarco único (um B-arco) indo dos problemas principais para o nó E.

São então estabelecidas as correspondências de modo análogo ao que foi feito para o caso anterior, verificando-se serem os GEOGs BF-grafos, pois seus arcos componentes são B-arcos ou F-arcos. Nesse caso, a solução é um F-caminho particular, que tem a seguinte restrição: para qualquer vértice v no caminho, $\mid FS(v) \mid$ no caminho é no máximo igual a 1.

IV.6 Arborescência de Woeginger

A seguir é apresentado um resumo contendo os principais conceitos do trabalho de Woeginger ([Woeginger 92]), que trata da conceituação e obtenção de arborescência para a classe definida e referida pelo autor como a hipergrafos orientados, que verificaremos tratar-se na verdade da B-grafos. A idéia é, se possível, tentar estender tal definição para a classe BF-grafos (2-grafos).

IV.6.1 Arborescência

Inicialmente apresentaremos as principais definições no trabalho, identificando a classe tratada a B-grafos. A proposta então é estender a definição de arborescência para a classe BF-grafos (e se possível para a 2-grafos).

Principais definições em [Woeginger 92]:

Def.IV.14: Seja X um conjunto finito de vértices. Um hipergrafo H sobre X é uma família de subconjuntos, chamados arestas, de X.

Def.IV.15: Um hipergrafo orientado é um hipergrafo tal que em todo conjunto $h \in H$ distingue-se um elemento chamado cabeça de h.

Def.IV.16: Um hipergrafo orientado H é uma arborescência se e somente se:

(i) H é vazio, ou

(ii) Existe uma aresta $h_1 \in H$ tal que nenhum outro $h \neq h_1$ contém a cabeça de h e tal que $H - h_1$ é também uma arborescência.

Exemplos de arborescências: na figura IV.12, os hipergrafos de (a) e (b) são arborescências. O hipergrafo de (c) não é.

Def.IV.17: Ciclo em Hipergrafo Orientado

Um ciclo em um hipergrafo orientado é uma sequência de arestas $h_0, h_1, ..., h_{k-1}$ tal que h_i contém a cabeça de h_{i+1}.

Exemplos de ciclos: figura IV.13.

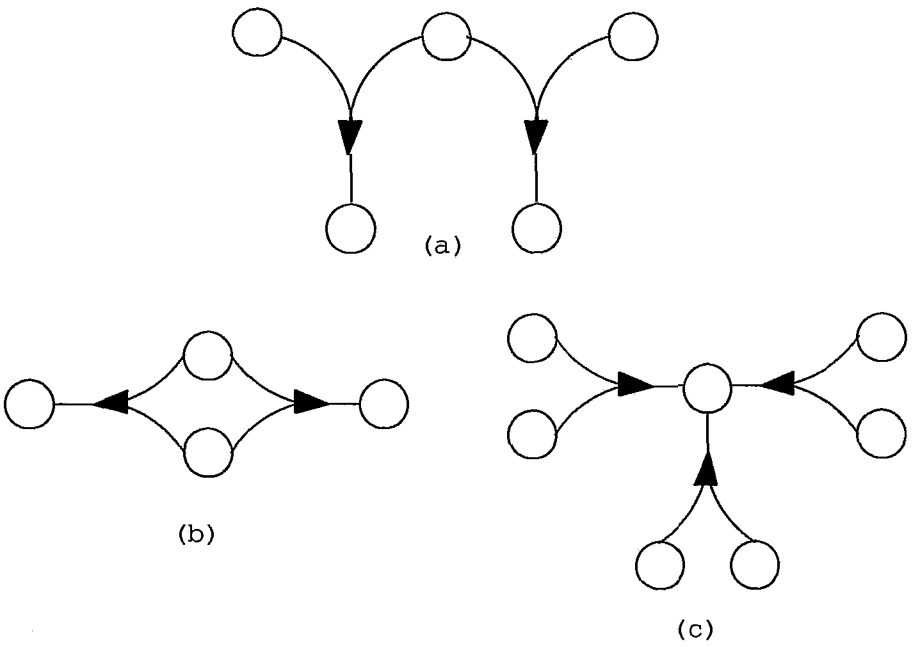


Figura IV.12: (a) e (b) são arborescências,(c) não é

Se um hipergrafo orientado é uma arborescência, ele não tem ciclo. No caso de todas as arestas terem cardinalidade 2 (o hipergrafo é um digrafo) arborescências podem ser chamadas de branchings(desconexo) ou out-trees(conexo), de acordo com as definições abaixo. Exemplo de uma out-tree: figura IV.14

Def.IV.18([Gibbons 89]) : Uma out-tree é uma árvore direcionada onde precisamente um vértice tem grau de entrada igual a 0.

Def.IV.19([Gibbons 89]) :Branching é uma floresta de out-trees.

Na definição de hipergrafo de Woeginger, apesar de destacar no conjunto aresta um elemento como cabeça, não fica clara a orientação das arestas. Somente quando o autor faz a observação anterior, sobre out-trees, é que podemos identificar que a orientação é no sentido dos demais vértices da aresta h ao vértice cabeça(h). Assim sendo, podemos identificar a aresta como sendo um B-arco, o que implica ser a classe de hipergrafos tratada no trabalho a B-grafos.

Deve-se notar também que o ciclo da figura IV.13.(b), como é definido pelo autor, não é intuitivo. Essa definição não tem correspondente no presente trabalho.

Def.IV.20: Ciclo em Hipergrafos Não Orientados [Woeginger 92].

Uma definição anterior se faz necessária: dizemos que uma família de conjuntos $S_1, ..., S_k$ é uma cobertura de um dado conjunto S sse $S \in \bigcup_{i=1,..,k} S_i$. Um hipergrafo H é um ciclo sse para todas as arestas h de H, as arestas remanescentes são uma cobertura de h.

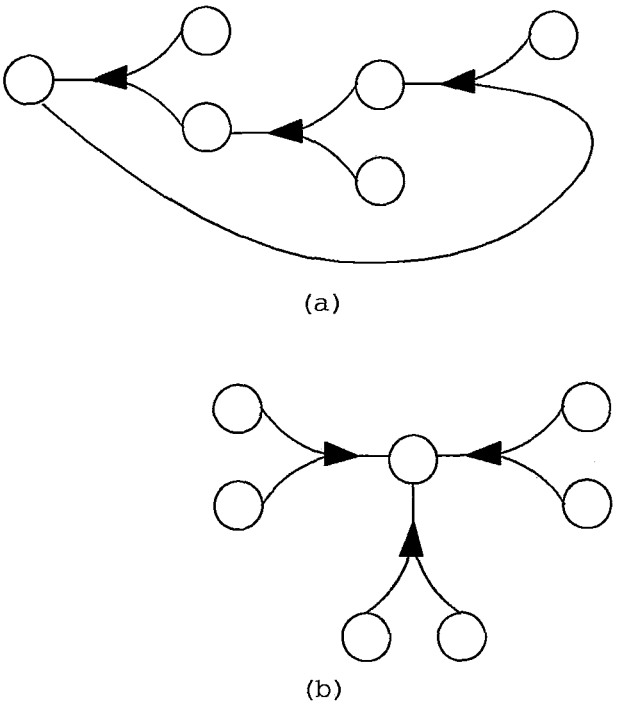


Figura IV.13: ciclos de Woeginger

Seja H uma arborescência. Podemos identificar as seguintes propriedades.

1) H não contém ciclo;

2) Para todo vértice v em H, $\mid BS(v) \mid$ é no máximo 1;

3) O hipergrafo não orientado subjacente H' não contém ciclo, de acordo com a definição anterior.

4) Apesar do nome, não conseguimos identificar um determinado vértice ou mesmo um conjunto de vértices, como raiz da arborescência.

Deve-se notar que não é suficiente para que um hipergrafo orientado seja uma arborescência que o hipergrafo não orientado subjacente seja acíclico.

Estendendo a definição de arborescência para a classe BF-grafos, de modo que fiquem mantidas todas as características, temos:

Def.IV.21: Arborescência em BF-grafos.

Um hipergrafo orientado H é uma arborescência se e somente se

(i) H é vazio, ou

(ii) Existe uma aresta $h_1 \in H$ tal que para todo $v \in$ cabeça de h nenhum outro $h \neq h_1$ contém v e tal que $H - h_1$ é também uma arborescência.

Um exemplo de arborescência em BF-grafos pela def.IV.21 pode ser visto na figura IV.15.

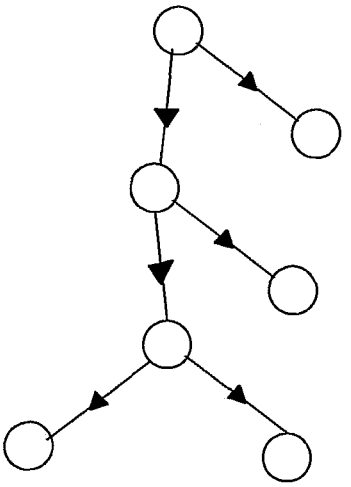


Figura IV.14: Uma out-tree

Deve-se notar que a também a definição de ciclo para hipergrafos não orientados, def.IV.20, apresentada pelo autor em [Woeginger 92], difere da definição de ciclo de Berge [Berge 70], adotada na maioria dos trabalhos e bem mais intuitiva, que será apresentada a seguir.

Def.IV.22: Ciclo em Hipergrafos Não Orientados [Berge 70].

Seja H um hipergrafo não orientado. Um ciclo é a sequência $(x_1, E_1, x_2, E_2, x_3, ..., x_k, E_k, x_1)$ com:

(1) $E_1, E_2, ..., E_k$ arestas distintas de H;

(2) $x_1, x_2, ..., x_k$ vértices distintos de H;

(3) $x_i, x_{i+1} \in E_i (i = 1, 2, ..., k-1)$;

(4) $x_k, x_1 \in E_k$.

Uma alternativa para uma definição de arborescência, que na verdade seria mais apropriado chamarmos uma árvore, em BF-grafo, é fazer uma definição análoga a uma definição clássica de árvore para digrafos, que é a seguinte:

Def.IV.23: Um digrafo é chamado árvore se o seu grafo subjacente é uma árvore (conexo e acíclico).

Primeiramente escolheríamos uma entre as duas definições de ciclo em hipergrafo não orientado, sendo a de Berge a mais usual. A definição de árvore para BF-grafos seria então:

Def.IV.24: Um hipergrafo orientado é uma árvore se o hipergrafo não orientado subjacente não contém ciclo.

Na figura IV.16 temos uma arborescência pela def.IV.24.

As B-árvores (F-árvores) de Gallo que serão vistas no próximo capítulo, são

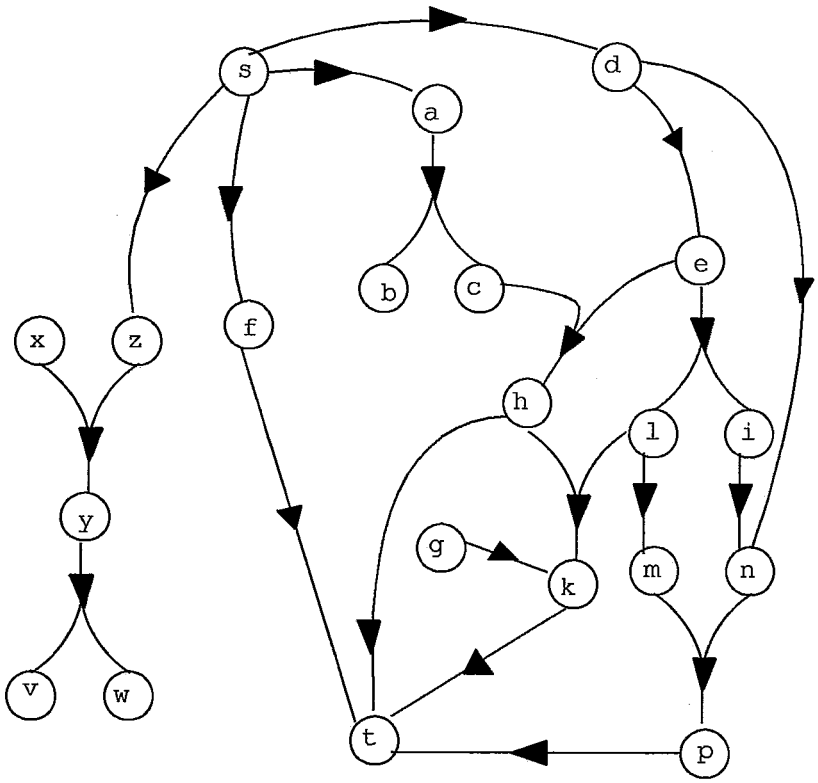


Figura IV.15: Uma arborescência pela def IV.21

definidas por características próprias, não podendo ser identificadas árvores.

Apesar de tratar da definição e obtenção de arborescência em hipegrafos orientados (B-grafos), o trabalho de Woeginger utiliza definições muito particulares, que não correspondem às mais usualmente encontradas na literatura, dificultando a extensão dos resultados obtidos em [Woeginger 92].

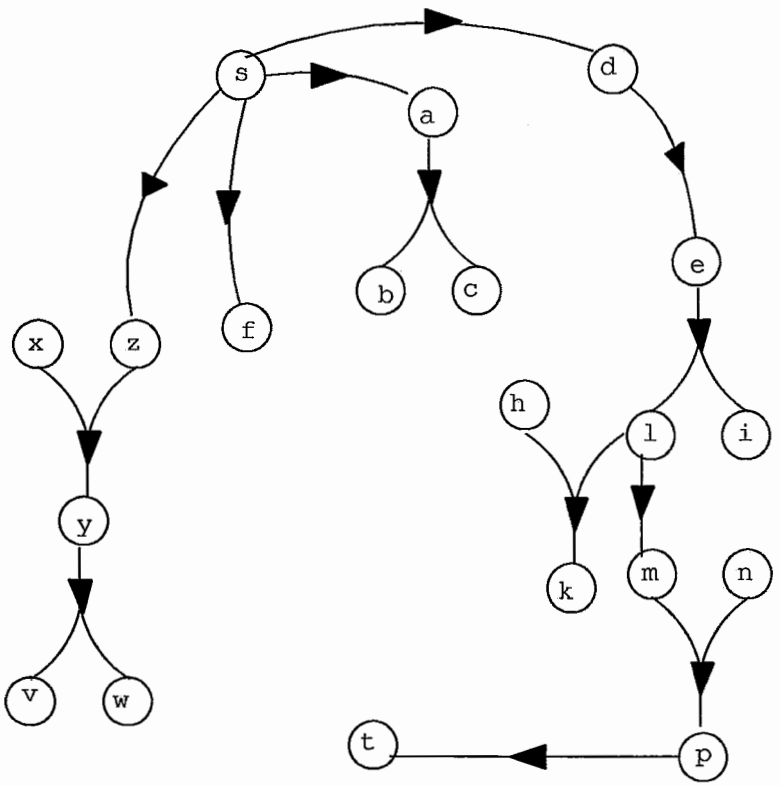


Figura IV.16: Uma arborescência pela def IV.24

Capítulo V

Caminhos, Árvores e Visitas em 2-Grafos Orientados

V.1 Introdução

A proposta desse capítulo é a conceituação e obtenção de caminhos, hipercaminhos e árvores em 2-grafos orientados. Isso será feito com base nos trabalhos de Gallo ([Gallo 90] e [Gallo 93]) e Markenzon ([Markenzon 91]).

Serão relacionadas as definições de caminhos apresentadas em [Gallo 90] e as modificações posteriores adotadas em [Gallo 93], juntamente com uma análise das diferenças existentes entre estas e no que acarretam, em particular com relação ao comportamento das buscas (visitas). Serão também apresentados alguns problemas identificados e alternativas para solucioná-los.

Em ambos os trabalhos o autor procura caracterizar os resultados ao final da execução dos algoritmos de buscas (visitas) como árvores (classes especiais de árvores), havendo um certo relaxamento quanto a formalidade e completude dessas definições. Assim sendo, a formalização do conceito de árvore com base nas definições anteriores será apresentada, bem como o algoritmo que permita a sua obtenção.

V.2 B-Caminho e B-visita

A definição e obtenção de hipercaminhos em BF-grafos são bem mais elaboradas do que definir e obter caminhos em grafos. Uma definição de B-caminho (F e BF-caminho) é apresentada em [Gallo 90], mas sofre modificações para ser reapresentada em [Gallo 93]. Os conceitos fundamentais de hipergrafos orientados utilizados, como caminho simples, ciclo e conexão num BF-grafo, são os apresentados no capítulo I. Consideremos a definição inicial de B-caminho, onde fica caracterizada a ausência

de ciclos, apresentada em [Gallo 90], e que também é a utilizada por Markenzon em [Markenzon 91].

Def.V.1: B-caminho de origem s e destino t :[Gallo 90]

Um B-caminho de origem s e destino t é o BF-grafo acíclico minimal $H_b = (V_b, E_b)$ tal que:

(i) $E_b \subseteq E$;

(ii) $s, t \in V_b = \bigcup_{e \in E_b} e$;

(iii) se $x \in V_b$ então x é conectado a s em H_b;

Exemplo: O BF-grafo da figura V.1(a) é um B-caminho, enquanto os BF-grafos de V.1(b) e V.1(c) não atendem às condições da def.V.1.

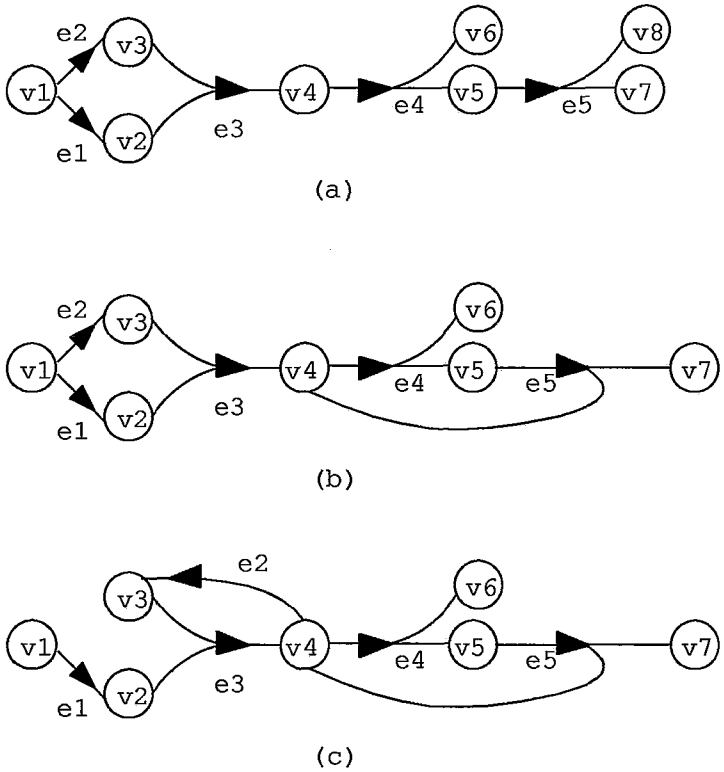


Figura V.1: (a): Um B-caminho por def.V.1 e def.I.11; (b): um B-caminho somente por def.I.11; (c): um BF-grafo que não atende às condições de B-caminho de ambas definições

Também em [Gallo 90], é apresentado, pela primeira vez, o algoritmo que implementa a B-visita. Numa B-visita em um BF-grafo $H(V, E)$, são visitados todos os vértices $v \in V$ tal que exista um B-caminho entre o vértice raiz s e v em H.

Segundo o autor, a idéia é que, além de visitar os vértices identificando-os como B-conectados a s, ao final da B-visita se possa recuperar o B-caminho de s a um

determinado vértice x. Para isso será utilizada a função predecessor $P_v[v]$, definida para todo $v \in V$, tal que:

$$P_v[v] = \begin{cases} k, & e_k \text{ precede } v \text{ no B-caminho de } s \text{ a } v; \\ 0, & \text{caso não exista tal B-caminho.} \end{cases}$$

Toda vez que um vértice v é visitado deve-se, além de se guardar em $P_v[v]$ a aresta através da qual se alcançou v, incluir v num conjunto Q de vértices visitados. Quando um vértice w é retirado de Q, para toda aresta e_j tal que $w \in rabo(e_j)$, um contador K_j é incrementado. Uma aresta e_j só poderá ser "explorada" na B-visita, ou seja, os vértices em $cabeça(e_j)$ só poderão ser visitados, quando todos os vértices no $rabo(e)$ tiverem sido visitados, ou seja, quando $K_j = \mid rabo(e) \mid$.

O algoritmo que implementa a B-visita [Gallo 90] é apresentado em seguida.

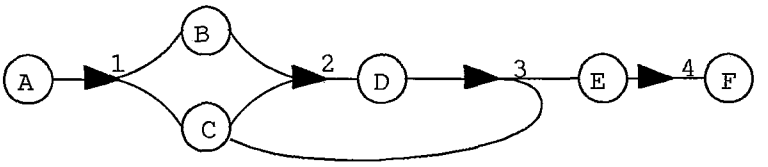
alg.V.1 : B-visita($s, H(V,E)$)

```
inicio
    para cada i ∈ V faça
        P_v[i] := 0;
    para cada e_j ∈ E faça
        K_j := 0;
    P_v[s] := nil; Q:= s;
    repita
        selecione e remova i ∈ Q
        para cada e_j ∈ FS(i) faça
        inicio
            K_j := K_j + 1;
            se K_j = | rabo(e_j) | então
            inicio
                para cada h ∈cabeça(e_j) tq P_v[h] = 0 faça
                inicio
                    P_v[h] := e_j;
                    Q := Q ∪ h
                fim
            fim
        fim
    ate Q =∅
fim
```

A complexidade da B-visita é $O(size(E))$.

Podemos identificar dois casos em que o algoritmo não respeita às exigências da def.V.1.

O primeiro é quanto à aciclicidade do caminho. Considere o BF-grafo H da figura V.2. Nele, após a execução da B-visita com raiz = A, o vértice E aparece como B-conectado ao vértice A, com P_v[E] diferente de NIL; observa-se, entretanto, que existe o ciclo DCD nesse B-caminho.



	P v
A	nil
B	1
C	1
D	2
E	3
F	4

Figura V.2: O BF-grafo H e os valores de P_v após a B-visita

A rotina B-visita, ao examinar uma aresta e_k com $\mid cabeça(e_k) \mid \geq 2$ (um F-arco), $v \in cabeça(e)$, inclui e_k no caminho s a v, se v é não marcado, e não faz nada se v já está marcado. Entretanto, consideremos os vértices $x, w \in cabeça(e)$, tais que x é não marcado e w já estava marcado. Nesse caso, não exista nada que garanta w não estar num caminho de s a x que contenha a seqüencia $(w, e_i, v_i, ..., e_k, x)$, e, conseqüentemente, o B-caminho conter o ciclo $(w, e_i, v_i, ..., e_k, w)$.

É importante observar que a detecção de ciclo na B-visita é, se não impossível, bastante difícil. Para isso seria necessário ao alcançar, mediante uma aresta e, um vértice já marcado, identificar se essa aresta e é de avanço ou de retorno. Essa classificação é sem sentido em uma busca em largura, como é o caso da B-visita. Mesmo no caso de grafos simples a identificação de existência de ciclo, como decorrência imediata, só ocorre na busca em profundidade (ver [Szwarcfiter 84]). A natureza da B-visita, no caso, compromete a detecção de ciclo.

Em [Gallo 93], a definição de B-caminho sofre modificações, eliminando-se a necessidade do B-caminho ser acíclico. Essa definição modificada é a definição de B-caminho que está no capítulo I, a def.I.11, que reapresentaremos a seguir.

Def.I.11: Um B-caminho de origem s e destino t é o BF-grafo minimal $H_b = (V_b, E_b)$ tal que:

(i) $E_b \subseteq E$;

(ii) $s, t \in V_b = \bigcup_{e \in E_b} e$;

(iii) se $x \in V_b$ então x é conectado a s em H_b mediante um caminho simples acíclico;

Exemplo: Os BF-grafos da figura V.1(a) e V.1(b)

É muito importante atentar para que a condição (iii) implica o relaxamento parcial de aciclicidade, já que continua sendo necessária a existência de um caminho caminho simples acíclico, de s a qualquer vértice no B-caminho.

Consideremos o BF-grafo da figura V.1(b), que contém ciclo. Observe que existe caminhos simples de v_1 a todos os outros vértices que não contém o ciclo v_4, e_4, v_5, e_5, v_4. Por exemplo, o caminho de de v_1 a v_7 é $(v_1, e_1, v_2, e_3, v_4, e_4, v_5, e_5, v_7)$.

Seja o ciclo identificado no B-caminho obtido na B-visita. Este ciclo é decorrente da inclusão do F-arco e no caminho a um vértice v não marcado, existindo um w em *cabeça*(e) já marcado tal que w estava em um caminho de s a v. Tal ciclo passa a ser permitido por definição, já que para todos os vértices do B-caminho, existem caminhos simples que não contém tal ciclo.

Na verdade, todo ciclo em um B-caminho é dependente de um F-arco com $|cabeça| \geq 2$.

Consideremos agora a condição minimalidade de um B-caminho. Mostraremos que caminhos não minimais podem ser obtidos na B-visita, com ambas as definições.

O que a rotina faz é, mantendo uma aresta predecessora de um vértice, recuperar, para um determinado vértice x, o B-caminho de s a x, incluindo a aresta e guardada em $P_v[x]$ e recuperando os B-caminhos para todos os vértices no *rabo*(e). Seja r um desses vértices. Nada garante que na união dos B-caminhos de s aos vértices de $\{rabo(e) - r\}$ não contenha um B-caminho de s a r, que não contenha e. Assim sendo, e e tudo o que foi recuperado a partir dela não são necessários, sendo o caminho não minimal.

Consideremos a função predecessor P_v utilizada na recuperação do B-caminho e tomemos como exemplo o BF-grafo da figura abaixo.

Se $P_v[\mathrm{C}] = 1$ e se desejássemos obter o B-caminho de A a H, recuperaríamos a aresta 1 nesse B-caminho, quando existe uma caminho de A a C em que 2 precede A,

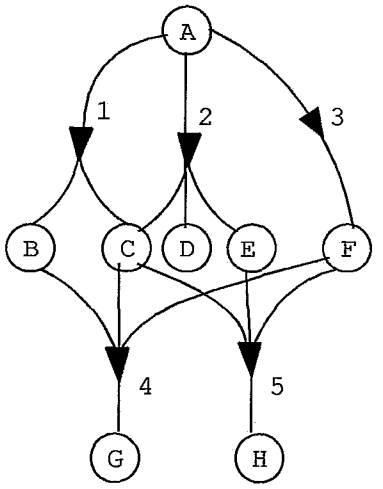


Figura V.3: Um BF-grafo com problemas na B-visita

e 2 é essencial ao B-caminho até H, pois é o único caminho de A a E. Ao contrário, se o B-caminho desejado fosse o de A a G e P_v[C] fosse igual a 2, teríamos problema semelhante. Não existe um único valor para P_v que resolva o problema.

Uma alternativa é recuperarmos primeiramente o provável B-caminho de s a x, ainda não minimal e procedermos de maneira semelhante ao que é feito em [Markenzon 91] para obtenção de minimalidade de um BF-caminho. Consiste basicamente em ir removendo uma a uma todas as arestas, repetir a busca e verificar se o vértice x deixa de ser visitado. Em caso positivo, a aresta é mantida no caminho. A seguir serão apresentadas a rotina que recupera um provável B-caminho de s a x após uma B-visita e a rotina que obtém a minimalidade desse B-caminho.

alg.V.2 : Recupera(s, x, H, H_b)

```
inicio
    V_b := ∅
    E_b := ∅
    se P_v[x] ≠ 0 então
    inicio
        V_b := V_b ∪ x;
        marca_v[x] := verdade;
        E_b := E_b ∪ P_v[x];
        Q := Q ∪P_v[x];
        marca_e[P_v[x]] := verdade;
    enquanto Q ≠ ∅ faça
    inicio
```

```
            selecione e remova e de Q
            para cada v ∈cabeça(e) faça
                se não(marca_v[v]) então
                inicio
                    V_b := V_b ∪ v;
                    marca_v[v] := verdade;
                fim
            para cada v ∈ rabo(e) faça
            se não(marca_v[v]) então
            inicio
                V_b := V_b ∪ v;
                marca_v[v] := verdade;
                se não(marca_e[P_v[v]]) então
                inicio
                    E_b := E_b ∪ P_v[v];
                    Q := Q ∪P_v[v];
                    marca_e[P_v[v]] := verdade;
                fim
            fim
        fim
    para cada v ∈ V_b faça
        marca_v[v] := falso;
    para cada e ∈ E_b faça
        marca_e[e] := falso;
    fim
fim
```

alg. V.3: B-caminho(s, x, H)

```
inicio
    para v ∈ V faça
        marca_v[v] := falso;
    para e ∈ E faça
        marca_e[e] := falso;
    B-visita(s, H);
    se P_v[x] = 0 então
```

```
        'Não há tal B-caminho'
    senão
    inicio
        Recupera(s, x, H, H_b);
        H' := H_b;
        para e ∈ E_b faça
        inicio
            H" := H';
            H" := H"−e;
            B-visita(s, H")
            se P_v[x] ≠ 0 então
                H' := H";
        fim
        Recupera(s, x, H', H_b);
    fim
fim
```

A necessidade de garantir minimalidade implica o aumento da complexidade para a obtenção de um B-caminho.

Na rotina Recupera, são examinadas as arestas retiradas de Q. Uma aresta e só é inserida em Q na primeira vez em que é incluído no caminho algum vértice v, tal que $P_v[v] = e$. No pior caso, os n vértices de H podem estar nesse caminho não minimal. Assim sendo, pode haver, no máximo, n arestas no caminho. Examinar uma aresta é $O(\mid cabeça(e) \mid + \mid rabo(e) \mid$, ou seja, $O(size_e)$. A complexidade total da rotina Recupera é $O(n \times size_e)$.

A complexidade total da rotina B-caminho é $O(size(E) + (n \times size_e))$.

A conceituação e obtenção de árvores são tratadas na próxima seção.

V.3 B-árvore

Em ambos os trabalhos, [Gallo 90] e [Gallo 93], uma B-árvore com raiz r é definida pelo conjunto dos B-caminhos obtidos ao final de uma B-visita, que contém todos os nós B-conectados a r (def.V.3).

Apesar de uma B-árvore ser definida da mesma forma em [Gallo 90] e [Gallo 93], as considerações na seção anterior sobre B-caminho e B-visita, nos levam à formali-

zação do conceito de B-árvore. A redefinição de B-caminho implica a obtenção de B-árvores um pouco diferentes de um trabalho para o outro.

Def.V.3: Uma B-árvore de raiz r é um conjunto de B-caminhos contendo todos os nós que são B-conectados a r, e que são obtidos na B-visita. Ex: figura V.4.

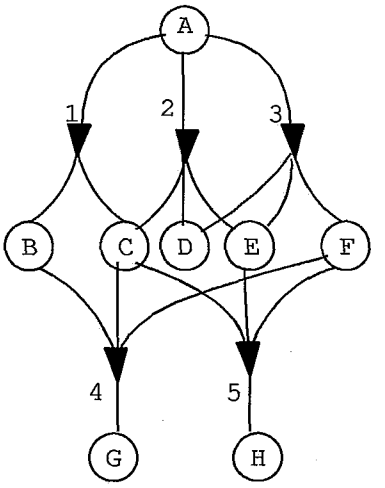


Figura V.4: Uma B-árvore pela Def.V.3

A definição acima não implica minimalidade nem aciclicidade. Como um B-caminho pode ter ciclo, consideremos somente a questão da minimalidade. A nova definição seria então:

Def.V.4: Uma B-árvore com raiz r é um BF-grafo minimal $T(V, E)$ tal que para todo $v \in V$ existe um B-caminho de r a v. Ex: figura V.5.

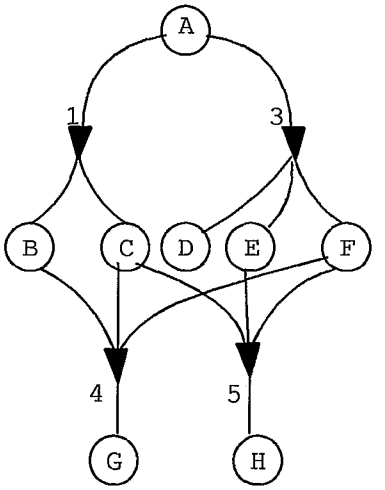


Figura V.5: Uma B-árvore pela Def.V.4

A obtenção de uma B-árvore é feita garantindo-se a minimalidade do conjunto inicialmente obtido na B-visita em um BF-grafo H com raiz s. Isso é feito de modo

análogo ao caso do B-caminho, a partir do BF-grafo $T(V_T, E_T)$, onde V_T é o conjunto de vértices em H visitados na B-visita e E_T é o conjunto de arestas tal que $e \in E_T$, se existe algum $P_v[v] = e$. Para cada $e \in E_T$, remove-se e, refazendo-se a B-visita para T. Se, para todo $v \in V_T$, $P_v[v] \neq 0$, a aresta e não pertence a T.

Deve-se notar que uma B-árvore minimal não garante caminho minimal.

Ex: Na B-árvore minimal da figura V.6, o caminho recuperado de A a G pode ser não minimal (basta que $P_v[C]$=1)

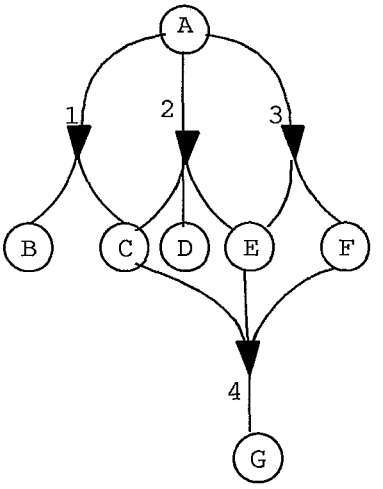


Figura V.6: B-árvore (minimal) e caminho não minimal.

Def. V.5: B-árvore geradora. Seja um BF-grafo orientado $H(V_h, E_h)$. Se existe uma B-árvore $T(V_t, E_t)$ tal que $V_t = V_h$ então essa é chamada B-árvore geradora de H.

Existe uma B-árvore geradora de H se ao final de uma B-visita todos os vértices de H foram visitados.

Podemos definir e obter, de maneira análoga, F-árvore e F-árvore geradora.

V.4 BF-caminho

Também a definição de BF-caminho sofre mudança de [Gallo 90] para [Gallo 93]. Só que, ao contrário do que acontecia para B-caminho, em que a segunda definição implica num grupo de caminhos mais abrangente do que a primeira, incluindo todos os que atendiam a essa e alguns mais, BF-caminho é redefinido de modo mais restrito, como será visto mais adiante. Em seguida serão apresentadas a definição inicial de [Gallo 90] e a versão mais recente [Gallo 93], respectivamente.

Def. V.6: BF-caminho de origem s e destino t :

Um BF-caminho de origem s e destino t é o BF-grafo acíclico minimal $H_p = (V_p, E_p)$ tal que:

(i) $E_p \subseteq E$;

(ii) $s, t \in V_p = \bigcup_{e \in E_p} e$;

(iii) se $x \in V_p$ então x é conectado a s em H_p e t é conectado a x em H_p;

Exemplo: na figura V.7 temos um BF-caminho de s a t pela definição V.6.

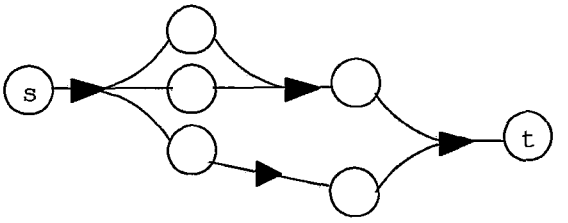


Figura V.7: BF-caminho pela definição V.6.

Para a nova definição de BF-caminho é conveniente rever o conceito de F-caminho [Gallo 93], introduzido no capítulo I (Def.1.11).

Um BF-grafo é um F-caminho de s a t se a sua imagem simétrica é um B-caminho de t a s.

Observe que isto implica em x conectado a t por um caminho simples acíclico na imagem simétrica do BF-grafo H, mas não é equivalente a dizer que t é conectado por um caminho simples acíclico a x no BF-grafo H.

Def. V.7: Um BF-caminho de s a t é o BF-grafo que é ao mesmo tempo um B-caminho e um F-caminho de s a t.

Na figura V.8 podemos ver um BF-caminho de s a t pela definição V.7.

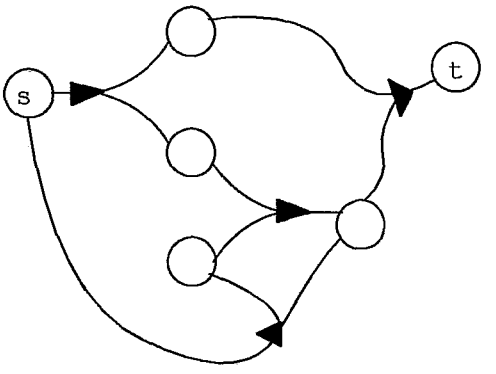


Figura V.8: BF-caminho pela definição V.7.

Ainda que a aciclicidade não seja uma exigência explícita da definição, tal definição implica a ausência de ciclos. Dessa forma, é fácil verificar que todo BF-caminho

pela def.V.7 também o será pela def.V.8. Só que o contrário não se verifica, como é o caso do BF-grafo da figura V.7. O BF-grafo atende às condições de def.V.6, mas não às da def.V.7, pois apesar de ser um F-caminho, não é um B-caminho, por não ser minimal.

Poderíamos considerar na tentativa de definir BF-caminho o aspecto mais importante a idéia de manutenção do fluxo, ou seja, tudo o que sai de um vértice origem s chega a um vértice destino t. Teríamos como alternativa uma extensão da primeira definição, def.V.6, com o mesmo relaxamento da aciclicidade adotado para o B-caminho. Mais formalmente teríamos:

Def.V.8: BF-caminho de origem s e destino t :

Um BF-caminho de origem s e destino t é o BF-grafo minimal $H_p = (V_p, E_p)$ tal que:

(i) $E_p \subseteq E$;

(ii) $s, t \in V_p = \bigcup_{e \in E_p} e$;

(iii) se $x \in V_p$ então x é conectado a s em H_b mediante um caminho simples acíclico e x é conectado a t na imagem simétrica de H_b mediante um caminho simples acíclico ;

BF-grafos em que há a preservação de fluxo entre s e t são denominados grafos fronteira e caracterizados em [Markenzon 91]. Um BF-caminho é um grafo fronteira minimal. Algoritmos para obtenção do grafo fronteira maximal de um BF-grafo podem ser encontrados em [Markenzon 91].

Parecendo ser interessante explorar as duas alternativas, chamaremos ao BF-caminho que acabamos de definir de BouF-caminho, e ao BF-caminho como definido em D7 de BeF-caminho.

Todo ciclo em um B-caminho (F-caminho) é dependente de um F-arco (B-arco) com $\mid cabeça \mid \geq 2$ ($\mid rabo \mid \geq 2$).

Assim sendo, todo B-caminho (F-caminho) num B-grafo (F-grafo) é acíclico. Mais ainda, um BF-grafo que seja ao mesmo tempo um B-caminho e um F-caminho (BeF-caminho) é necessariamente acíclico.

Capítulo VI

Conclusões

Nos dois primeiros capítulos foram estudados diferentes esquemas de representação para a classe 2-grafos. Verificou-se que a matriz de adjacência e as listas de adjacência, as representações mais conhecidas para grafos, não se prestam à representação de 2-grafos. Foram então propostos e estudados esquemas de representação em que houvesse estruturas especiais para a representação das arestas. Dentre esses esquemas, o que se mostrou mais apropriado, com relação à efiência e aproveitamento de espaço em memória, principalmente no caso dinâmico, foi o das listas $BS[v]$, $FS[v]$, $Cab[e]$ e $Rab[e]$.

No estudo de alguns casos particulares na literatura, encontrou-se dificuldade na identificação da classe e problema tratados, causada pelas diferenças das terminologias adotadas nos diferentes trabalhos.

O fato de ser um assunto relativamente novo, que só recentemente vêm sendo mais explorado nas pesquisas, implica a existência de definições que ainda não estão estabelecidas. No presente trabalho, várias das definições inicialmente adotadas, que eram as apresentadas em [Gallo 90], sofreram modificações com a publicação de [Gallo 93]. Essas definições, assim como alguns problemas problemas deixados em aberto no decorrer do trabalho, constituem objeto de pesquisas futuras. Algumas das alternativas seriam o estudo de representação de 2-grafos para casos com restrições, a manutenção dinâmica do fechamento de BF-grafos (2-grafos) e o estudo de equivalência e minimalidade para 2-grafos.

Bibliografia

[Ausiello 83] Ausiello, G., D'Atri, A., Sacca, D., *Graph Algorithms of Functional Dependency Manipulation*, **J. ACM**, vol.30, pp.752-766, 1983.

[Ausiello 85] Ausiello, G., D'Atri, A., Sacca, D., *Strongly Equivalent Directed Hypergraphs*, em Analysis and Design of Algorithms for Combinatorial Problems, **Annals of Discrete Mathematics**, vol.25, pp.1-25, North-Holland, Amsterdam,1985.

[Ausiello 86] Ausiello, G., D'Atri, A., Sacca, D., *Minimal Representation of Directed Hypergraphs*, **Siam J. Comput.** vol.15, pp.418-431, 1986.

[Ausiello 90] Ausiello, G., Nanni, U., Italiano, G.F., *Dynamic Maintenance of Directed Hypergraphs*, **Theoretical Computer Science**, vol.72, pp.97-117, North-Holland, 1990.

[Berge 70] Berge, C., **Graphes et Hipergraphes**, Dunod, Paris, 1970.

[Berge 89] Berge, C., **Hypergraphs**, North-Holland, 1989.

[Date 91] Date, C.J., **Introdução a Sistemas de Banco de Dados**, Campus, Rio de Janeiro, 1991.

[Gallo 90] Gallo, G., Longo, G., Nguyen, S., Pallotino, S., *Directed Hypergraphs and Aplications*, TR-3/90, Dipartimento di Informatica, Univ. di Pisa, 1990.

[Gallo 93] Gallo, G., Longo, G., Nguyen, S., Pallotino, S., *Directed Hypergraphs and Aplications*, **Discrete Applied Mathematics**, vol.42, pp.177-201, North-Holland, 1993.

[Gibbons 89] Gibbons, A.,**Graph Theory**, Cambridge University Press, 1989.

[Horowitz 84] Horowitz, E., Sahni, S., **Fundamentos de Estruturas de Dados**, Campus, Rio de Janeiro, 1984.

[Levi 76] Levi, G., Sirovich, F., *Generalized And/Or Graphs*, **Artificial Intelligence**, vol.7, pp. 243-259, 1976

[Markenzon 87] Markenzon, L., *Propriedades e Algoritmos para Extensões e Especializações de Grafos de Fluxo Redutíveis*, Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, 1987.

[Markenzon 90] Markenzon, L., Nguyen, S., *La Structure 2-Graphe et Quelques Représentations Internes*, manuscrito, Nov. 1990.

[Markenzon 91] Markenzon, L., Nguyen, S., *Quelques Problèmes de Chemins pour les Hypergraphes Orientès*, CRT-763, Centre de Recherche sur les Transports, Université de Montréal, 1991.

[Nilsson 80] Nilsson, N.J., **Principles of Artificial Inteligence**, Morgan-Kaufmann, Los Altos, CA, 1980.

[Pombo 79] Pombo, H. C. R., *Representação de Grafos em Computador*, Tese de Mestrado, COPPE/UFRJ, Rio de Janeiro, 1979.

[Rich 88] Rich, E.; **Inteligência Artificial**, McGraw-Hill, São Paulo, 1988.

[Szwarcfiter 84] Szwarcfiter, J.L., **Grafos e Algoritmos Computacionais**, Campus, Rio de Janeiro, 1984.