ESTUDO DE ALGUNS ALGORITMOS PARA

ANÁLISE GLOBAL DE FLUXO DE DADOS

Este exemplar corresponde à redação da tese defendida pela Srta. KATIA LUCKWÜ DE SANTANA SILVA e aprovada pela comissão julgadora.

Campinas, de de 1984.

Prof. Dr. TOMASZ KOWALTOWSKI
Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

AGOSTO/1984.

UNICAMP

Aos meus pais,
com muito carinho.

SUMÁRIO

São analisados neste trabalho três métodos para a solução dos problemas de análise global de fluxo de dados, quando as equações têm como coeficientes subconjuntos de um universo finito (vetores de bits): método iterativo de Hecht e Ullman, método dos intervalos de Cocke e Allen, e o método das regiões fortemente conexas de Graham e Wegman. A comparação dos métodos é realizada através de uma microanálise das suas implementações, aplicada a algumas famílias de grafos de fluxo que têm forma padronizada. Os resultados indicam que, neste caso, o método das regiões é mais eficiente em termos de operações com vetores de bits, enquanto que o método iterativo é mais eficiente em termos de operações de controle e manipulação de estruturas de dados auxiliares.

AGRADECIMENTOS

Gostaria de agradecer de maneira muito especial ao Tomasz, pelo excelente trabalho de orientação, e principalmente pelo voto de confiança num trabalho que ficou tanto tempo parado.

Gostaria de agradecer ainda ao pessoal da biblioteca do IMECC, a Madalena, a Lina, a Lucy, e mais particularmente à Marilda, pelo apoio durante todo o desenvolvimento desse trabalho.

À minha amiga Ana Cristina pela realização dos desenhos.

À Lourdes pelo ótimo trabalho de datilografia (em tempo real).

Ao Departamento de Informática da UFPE por ter-me liberado para a conclusão desse trabalho.

Aos meus pais por saberem fazer-se sentir presentes durante todo o tempo, apesar da enorme distância.

Aos meus amigos queridos, aos colegas, a todos que me dão sorrisos, e por fim a Deus, pois sem eles todos eu não existiria.

ABSTRACT

We analyse three methods for solving the problem of global data flow analysis, when the equations have coefficients which are subsets of a finite universe represented by bit vectors. These methods are: iterative analysis by Hecht and Ullman, interval analysis by Cocke and Allen, and strongly connected region analysis by Graham and Wegman. We present a microanalysis of the implementation of these methods, when applied to some families of flow graphs with a standard form. The results show that in this case the region analysis method is more efficient in terms of bit vector operations, whereas the iteractive method is more efficient in terms of the control and auxiliary data structures manipulation operations.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

1.1. GENERALIDADES SOBRE ANÁLISE GLOBAL DE FLUXO DE DADOS.

A análise global de fluxo de dados (AGFD) consiste em levantar informações úteis distribuídas em todo o corpo de um procedimento, veiculando-as até pontos em que elas possam ser utilizadas. Em contraposição às análises de fluxo de dados local (que considera apenas pequenos trechos do procedimento onde não ocorrem desvios), e "interprocedural" (que inclui também as chamadas a procedimentos), a análise global abrange apenas o corpo do procedimento em questão, supondo o pior caso possível a cada chamada de procedimento. (Algumas vezes, ao invés de supor o pior caso, pode-se utilizar os resultados da análise de fluxo de dados do procedimento, feita em separado, mas nem sempre).

Inicialmente, a grande aplicação de informações obtidas pela AGFD era na otimização de código em compiladores. Mais tarde, a utilização desse tipo de informação foi proposta na detecção de erros de programação (variáveis cujo valor é utilizado sem que tenha recebido uma atribuição anteriormente, verificação de limites de vetores, etc) (FOSDICK [1976]).

Essa análise pode determinar, por exemplo:

a) Em que pontos do procedimento poderia o valor de uma variável A usada num ponto p ter sido definido, ocasionando a detecção de computações invariantes em malhas, de variáveis de indução múltiplas, ou de variáveis sem um valor inicial;

b) Se uma expressão computada duas ou mais vezes em diferentes pontos do programa já não teria seu resultado disponível em alguns desses pontos, possibilitando eliminar o cálculo de subexpressões comuns;

c) Se a única definição da variável A que alcança um comando c onde o valor de A é usado, é A ← B, e em todo comando executado após aquela definição até o comando c não há atribuição de valor a B, o que permitiria eliminar a atribuição A ← B, substituindo A por B em todos os comandos em que essa definição de A era usada;

d) Se uma dada variável A não será mais utilizada após um ponto do programa, o que torna desnecessário transferir o seu valor para a memória, se este estiver em um registrador, e permite a reutilização desse registrador;

e) Se uma expressão será calculada em diferentes comandos, porém com o mesmo resultado, qualquer que seja a seqüência de execução a partir de um ponto p, de forma que aqueles diversos comandos podem ser substituídos por um único no ponto p, o que diminui o tamanho do código.

O objetivo desse trabalho é estudar e comparar a eficiência de alguns algoritmos para AGFD, a partir de informações locais. Na seqüência desse capítulo, serão introduzidas a representação dos algoritmos, definições e propriedades básicas de grafos, e as equações usadas na AGFD. Os três capítulos seguintes apresentam métodos de solução para essas equações, assim como a análise de seus custos. No capítulo V são avaliados os custos dos algoritmos em situações particulares, visando obter parâmetros comuns aos métodos apresentados, de forma a possibilitar um confronto de seus desempenhos.

1.2. REPRESENTAÇÃO E AVALIAÇÃO DOS ALGORITMOS

Os algoritmos constantes desse trabalho serão apresentados por fases: inicialmente são discutidos as idéias fundamentais,em seguida são esboçados em linhas gerais os passos necessários para implementar essas idéias, e por fim são apresentados procedimentos numa linguagem algorítmica, do tipo "pascalês", que forne cem os detalhes de programação, incluindo as estruturas de dados usadas. Alguns procedimentos auxiliares simples, como para incluir ou remover elementos de uma lista, são omitidos.

Na avaliação dos custos dos procedimentos é usualmente empregada a técnica de macroanálise, que consiste em escolher uma operação dominante e expressar o tempo de execução como uma função

do número de vezes que essa operação é executada, em geral empregado apenas a ordem de grandeza (O) desse número. Nesse trabalho, entretanto, utilizaremos a microanálise (COHEN [1982]), na qual o tempo de execução é expresso como uma função do tamanho da entrada e de variáveis simbólicas, que representam o tempo necessário para executar cada uma das operações básicas envolvidas no procedimento, onde por operações básicas entende-se um conjunto de operações elementares que existem na maioria dos computadores, tais como atribuição, comparação, desvio, subscrição ou chamada de sub-rotina. Conhecendo-se o custo associado a cada operação básica num determinado computador, pode-se avaliar o custo do procedimento. KNUTH [1973] usa essa estratégia, avaliando o custo de implementação de algoritmos em função de um modelo de computador (MIX).

No nosso caso, o tamanho da entrada será dado por parâmetros como número de vértices (N_v), o número de arestas (N_a), ou número de laços (N_L) de um grafo. Uma relação completa dos parâmetros utilizados está no apêndice 1. O apêndice 2 é uma apresentação detalhada de todas as operações básicas. Estas foram escolhidas arbitrariamente, entretanto outras poderiam ser facilmente adotadas e suas quantidades computadas.

O porque da escolha a micro-análise reside na avaliação mais completa do custo dos procedimentos, uma vez que através de operações como subscrição, seleção de campo, inclusão ou remoção em lista, é refletido mais claramente o "overhead"

necessário na implementação de cada um dos algoritmos. Além do que, as operações lógicas sobre vetores de bits, em geral usadas na comparação dos procedimentos de AGFD, não são as operações dominantes.

No cálculo do custo de um procedimento, são avaliados inicialmente os custos associados a cada linha ou grupo de linhas, indicando, se possível, o número preciso de operações básicas de cada tipo em função do tamanho da entrada, ou então considerando o pior caso. A seguir, todas as operações básicas de um tipo executadas ao longo de um procedimento são agregadas, e por fim é calculada a soma dessas quantidades para todos os procedimentos envolvidos.

Alguns trechos dos procedimentos merecem consideração especial no momento da avaliação:

1. Nas operações de comparação explícita (instrução *se*), é contada a ocorrência de um desvio a cada teste (caso de *se-então-senão*), ou a ocorrência de um desvio, apenas quando o resultado da expressão booleana é falso (caso do *se-então*).

2. O número de operações no controle de saída de uma malha depende sempre do número de vezes que o seu corpo é executado (NEXEC) e do número de entradas na malha durante a execução (NENTR), porém não é uma função constante.

Se o corpo da malha é executado pelo menos uma vez a cada entrada (instrução do tipo *repita*), o teste de fim de itera - ção está localizado após o corpo da malha, logo são efetuadas NEXEC comparações e (NEXEC-NENTR) desvios. Se não é possível garantir uma primeira execução do corpo da malha (instrução do tipo *enquanto*), o teste de fim de iteração fica após a atribuição inicial à variável de controle, donde ocorrem (NEXEC + + NENTR) comparações e (NEXEC+NENTR) desvios.

Nas malhas do tipo "*para todo*" que visitam os nós de uma lista, são ainda necessárias NENTR atribuições, NEXEC avanços em lista (tomar o próximo nó), e (NENTR + número de comparações) acessos ao valor de uma variável. Se, por outro lado, a malha é do tipo "*para todo*" e percorre uma sequência de inteiros consecutivos, são usadas ainda (NENTR + NEXEC) atribuições, NEXEC operações aritméticas, e, dependendo do caso da sequência começar com uma constante e terminar com uma variável ou vice-versa, serão necessárias (NEXEC + 2 × número de comparações) ou (NENTR + NEXEC + número de comparações) acessos ao valor de uma variável, respectivamente.

3. Em algumas malhas, a atribuição de valor à variável de controle a cada entrada envolve operações diferentes das utilizadas nas demais atribuições a essa variável. Conta-se, então, separadamente essas operações. (Uma ilustração dessa situação é uma visita aos elementos de uma lista cuja cabeça está num dos

componentes de um vetor de referências, quando a atribuição na entrada envolve uma subscrição e as demais envolvem seleção de campo).

4. Nos laços iterativos cujo objetivo é visitar cada um dos m vértices de um grafo, onde a ordem é decrescente ou irrelevante, supõe-se uma variação de m-1 a 0 no valor da variável de controle, o que diminui o número de acessos ao valor de m no teste de fim de iteração. Se a ordem for necessariamente crescente, esses acessos são inevitáveis.

5. Algumas vezes é difícil avaliar com exatidão o número de iterações executadas a cada entrada numa malha ou o número de chamadas a um procedimento num dado ponto, entretanto é fácil precisar o número de vezes que o corpo da malha é executado ou quantas vezes um procedimento é ativado em todo o decorrer do programa. Nessas situações, portanto, "subimos a montanha" e contamos esses números totais.

1.3. DEFINIÇÕES E PROPRIEDADES BÁSICAS

Um *grafo* G é um par de conjuntos (VG, aG), tais que VG é um conjunto finito não vazio cujos elementos são chamados *vértices*, e aG é um conjunto de pares ordenados (u,v), chamados *arestas*, tais que u,v $\in$ VG (isto é, aG $\subseteq$ VG^2). Se α = (u,v) $\in$ $\in$ aG, dizemos que u é *predecessor* de v, v é *sucessor* de u,

e que u é a *origem* e v é o *destino* da aresta α. Se $u = v$, então (u,v) é um *laço*.

O *tamanho* de um grafo G é dado pela soma $|VG| + |aG|$. Um grafo G é *trivial* se G tem tamanho 1 ($|VG| = 1$ e $|aG| = 0$).

Para representar grafos nesse trabalho, vamos designar seus vértices por $0,1,2,\ldots,|VG|-1$ (dessa forma, a única informação necessária para representar VG é o inteiro $|VG|$), e usar um vetor com $|VG|$ componentes, no qual o u-ésimo componente é uma lista dos vértices v tais que $(u,v) \in aG$. Esse vetor será chamado *estrutura de adjacência* de G, e será designado por Δ. Alternativamente podemos utilizar a *estrutura de incidência* ∇ de G, um vetor com $|VG|$ componentes, no qual o u-ésimo componente é uma lista dos vértices w tais que $(w,u) \in aG$. Eventualmente as listas em Δ e em ∇ podem conter também dados associados às arestas do grafo.

Um grafo H é um *subgrafo* de outro G se $VH \subseteq VG$ e $aH \subseteq aG$. Dado um conjunto C de grafos, um grafo H é maximal em C se H não é subgrafo de nenhum grafo em C, exceto de si próprio. Dado um grafo G e um conjunto $W \subseteq VG$, o *subgrafo de G gerado por* W, G[W], é o subgrafo H de G tal que $VH = W$ e $aH = aG \cap W^2$ (ou seja, aH é o conjunto das arestas de G que têm origem e término em W).

Um *passeio* P em G é uma seqüência finita e não vazia de vértices $(v_0,v_1,\ldots,v_n)$ tal que para todo i, $1 \leq i \leq n$,

(v_{i-1},v_i) é uma aresta de G. Diremos que P é um passeio com *origem* v_o e *término* v_n, ou que P é um passeio de v_o a v_n. O inteiro n é o *comprimento* de P. Se n = 0, então P é dito *degenerado*. Um passeio $P = (v_o,v_1,\ldots,v_n)$ não degenerado pode ser alternativamente especificado pela seqüência de arestas $(\alpha_1, \alpha_2,\ldots,\alpha_n)$, na qual para todo i, $1 \leq i \leq n$, $\alpha_i = (v_{i-1},v_i)$. Os conjuntos $\{v_o,v_1,\ldots,v_n\}$ e $\{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ serão designados por VP e aP respectivamente. Diz-se que o passeio P *passa por* v_i se $v_i \in VP$, e que P *passa por* α_j se $\alpha_j \in aP$. Uma *seção* de P é um passeio que é uma subseqüência de termos consecutivos de P. Se as arestas de P forem duas a duas distintas, então P é uma *trilha*. Se os vértices de P forem dois a dois distintos, então P é um *caminho*. Se P tiver comprimento no mínimo 2, com $v_1,v_2,\ldots,v_n$ distintos dois a dois, e com $v_o = v_n$, então P é um *circuito*. Se um grafo G não contém circuitos, então G é dito *acíclico*.

Se P é o passeio $(v_o,v_1,\ldots,v_n)$ e Q é o passeio $(u_o, u_1,\ldots,u_m)$ em G, com $v_n = u_o$, então o *produto* PQ de P e Q é o passeio $(v_o,v_1,\ldots,v_n, u_1,\ldots,u_m)$.

Se num grafo G existe um caminho de um vértice u a um vértice v, então dizemos que u é *ligado* a v, e que v é *alcançável* (a partir) de u. Se além do caminho de u a v também existir um caminho de v a u no grafo G, dizemos que u é *fortemente ligado* a v, ou, por simetria dessa relação, dizemos que u e v são *fortemente ligados*. Algumas vezes as relações

de ligação, de alcance e de ligação forte poderão ser restritas a caminhos que satisfaçam determinadas condições. Um grafo que tem todos os seus vértices fortemente ligados dois a dois é dito *fortemente conexo*.

• GRAFO DE FLUXO DE CONTROLE

Em problemas de AGFD é necessário observar todas as possíveis seqüências de execução das instruções que compõem um procedimento. É possível traduzir todas essas seqüências de maneira finita, através de um grafo no qual os vértices representam seqüências maximais de instruções que são executadas sempre e somente quando todas as instruções anteriores representadas por esse vértice já o foram, e as arestas representam uma possível transferência de controle da última instrução representada pela origem para a primeira instrução representada pelo vértice destino. A esse grafo dá-se o nome de *grafo de fluxo de controle*, e às seqüências maximais de instruções que constituem cada vértice, dá-se o nome de *blocos* ou *blocos básicos*. Quando conveniente, identificaremos a noção do vértice e do bloco correspondente.

Chamaremos de *ponto* em um procedimento especificado numa linguagem qualquer, uma posição imediatamente anterior ou imediatamente após uma instrução. Chamaremos de *ponto de entrada* de um um bloco ao ponto imediatamente anterior à sua primeira instrução, e *ponto de saída* ao ponto imediatamente após a sua última

instrução.

Um procedimento no qual todas as instruções são, sintaticamente, passíveis de execução em alguma ativação, pode ser particionado em um conjunto de blocos facilmente identificáveis. Um bloco inicia-se com a primeira instrução executável numa ativação do procedimento, com uma instrução que é referenciada por uma instrução de desvio (condicional, incondicional, chamada e retorno de sub-rotina, saída de malha, etc), ou ainda com uma instrução que segue imediatamente um comando de desvio condicional, e encerra-se imediatamente após cada instrução de desvio ou após uma instrução que encerre a execução do procedimento.

Um grafo de fluxo de controle será denotado por $G = (VG, aG, n_o)$, onde $n_o \in VG$ é o vértice que contêm a primeira instrução executável numa ativação do procedimento, chamado *vértice inicial* do grafo. Pelo fato de todas as instruções serem passíveis de execução, todo vértice do grafo é alcançável a partir de n_o. Lembrando que os vértices serão denotados pelos inteiros de 0 a $|VG|-1$, convencionaremos que $n_o = 0$.

A partir desse ponto, utilizaremos simplesmente o termo *grafo* para designar um grafo de fluxo de controle. Um grafo G é uma *estrela* se toda aresta de G tem como origem o vértice inicial (0). Dado um grafo G, uma *árvore geradora* de G é um subgrafo acíclico e conexo de G tal que $VH = VG$, $|aH| = |VG|-1$, e todo vértice de $VH \setminus \{0\}$ tem exatamente um predecessor.

• DOMINÂNCIA

Se $G = (VG, aG, n_o)$ é um grafo, e $u,v \in VG$, diz-se que u *domina* v se todo passeio de n_o a v em G passa por u. Note se que a relação de dominância é de ordem parcial, com n_o dominando todos os vértices de VG.

Seja $(v,u) \in aG$. Se $u \neq v$ e u domina v em G, então (v,u) é dita uma *aresta de retorno*.

• GRAFOS REDUTÍVEIS

Um grafo G é dito *redutível* se não existe subgrafo H de G, tal que VH contém vértices u,v,w, distintos entre si, com $v,w \neq n_o$, e existem em H caminhos de n_o a u, de u a v, de de v a w, e de w a v, cujos vértices são todos distintos entre si, a menos das origens ou términos. A figura 1.1 apresenta o esquema de um grafo H como descrito, onde o caminho de n_o a u pode ser degenerado. Se um grafo não é redutível então ele é dito *irredutível*.

Outras caracterizações equivalentes de um grafo redutível G são apresentadas em HECHT [1974], dentre elas:

a) Existe um único subgrafo maximal acíclico de G;

b) Todo circuito C de G tem um vértice que domina todos os outros em VC;

c) Se G' é um subgrafo maximal acíclico de G, então toda aresta em $aG \setminus aG'$ é de retorno.

Note-se que os grafos redutíveis correspondem sempre a procedimentos nos quais cada malha tem um único ponto de entrada a partir do vértice inicial, e vice-versa. Os métodos de AGFD baseados em intervalos e em regiões, que serão apresentados nesse trabalho, só são aplicáveis a grafos redutíveis. Felizmente eles ocorrem com muita freqüência na prática, incluindo os procedimentos chamados de estruturados (HENDERSON [1976]).

• NUMERAÇÃO EM PROFUNDIDADE

Os algoritmos de AGFD aqui apresentados baseiam-se fortemente na ordem de visita aos vértices do grafo, sendo importante, também, identificar rapidamente as arestas de retorno. Essa ordenação dos vértices e identificação das arestas de retorno são facilitadas pelo estabelecimento de uma bijeção de VG nos naturais de 0 a |VG|-1, chamada *numeração em profundidade*. Essa numeração é obtida determinando-se uma árvore geradora do grafo G, e atribuindo aos vértices números consecutivos a partir de zero, de tal maneira que todo vértice tenha numeração menor que todos os seus sucessores na árvore. É fácil ver que, em geral, não existe uma única numeração em profundidade dos vértices de um grafo, como no exemplo da figura 1.2, entretanto note-se que nessa numeração, a imagem de n_o é sempre zero.

A figura 1.3. apresenta um algoritmo que obtém uma numeração em profundidade para um grafo G dado.

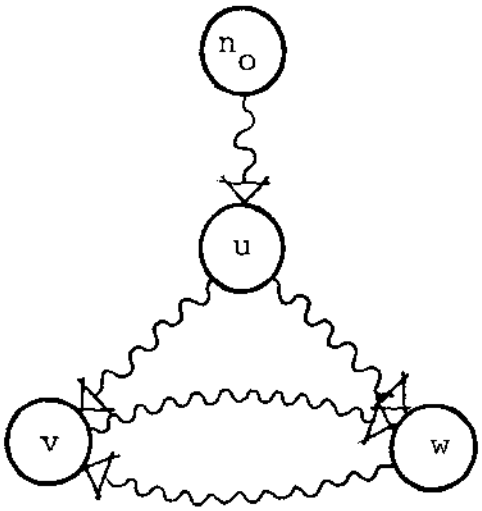
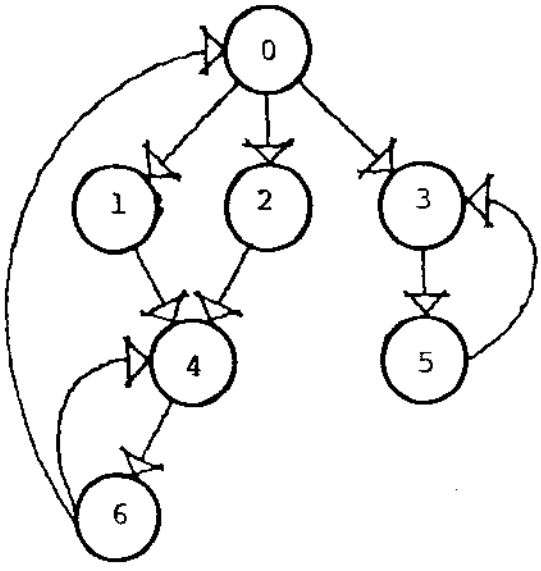


FIGURA 1.1



Vértice	Numeração 1	Numeração 2	Numeração 3
0	0	0	0
1	4	2	1
2	3	1	4
3	1	5	2
4	5	3	5
5	2	6	3
6	6	4	6

FIGURA 1.2

• PROPRIEDADE NP

Sejam G = (VG, aG, n_o) um grafo redutível, e *num* uma numeração em profundidade dos vértices de G. Se (u,v) ∈ aG, então v domina u sse *num*(v) ≤ *num*(u). Conseqüentemente, num grafo redutível toda aresta cuja origem tem numeração maior que o destino, é de retorno.

```
        Algoritmo Numera-em-Profundidade (G)

1. Para cada vértice u de VG
        marque u como não visitado

2. i ⟵ |VG| - 1

3. Observa-Sucessores (n_o)

        Algoritmo Observa-Sucessores (u)

1. Marque u como visitado

2. Para cada sucessor v de u
        se v não foi visitado
                então Observa-Sucessores (v)

3. Numeração (u) ⟵ i

4. i ⟵ i-1
```

FIGURA 1.3

1.4. DESCRIÇÃO GERAL DO PROBLEMA DE ANÁLISE DE FLUXO

Uma *expressão* é dita *disponível* num ponto p de um procedimento se, qualquer que seja a seqüência de execução considerada desde o início do procedimento até esse ponto (podendo incluir diversas passagens pelo ponto p), a expressão foi calculada antes do controle atingir o ponto p e após o último cálculo não houve alteração no valor de nenhuma das variáveis envolvidas.

Tomemos o problema de identificar as expressões disponíveis num ponto qualquer de um procedimento. Uma vez conhecidas as expressões disponíveis no ponto de entrada do bloco onde o ponto se encontra, a solução desse problema torna-se trivial. Trataremos então de determinar as expressões disponíveis no ponto de entrada de cada bloco do procedimento.

Dizemos que uma expressão é *gerada* em um vértice v do grafo, se ela é calculada em algum comando de v e nenhuma das variáveis envolvidas no cálculo é modificada subseqüentemente em v. Dizemos que uma expressão é *preservada* em um vértice v, se nenhuma das variáveis envolvidas no seu cálculo é modificada e a expressão não é calculada em v.

Note-se que, para um dado procedimento, podemos determinar o conjunto finito de expressões utilizadas, que constituirá o universo para esse problema. Denotá-lo-emos por U.

Se representarmos por Y_v e X_v os conjuntos das expressões

disponíveis na entrada e na saída do bloco v, respectivamente , e por k_v e c_v os conjuntos das expressões preservadas e geradas no bloco v, respectivamente, teremos pela definição de expressões disponíveis que:

$$\begin{cases} Y_v = \begin{cases} \phi & \text{se} \quad v = 0 \\ \bigcap\limits_{u \in \nabla v} X_u & \text{se} \quad v \in VG \setminus \{0\} \end{cases} \\ X_v = (Y_v \cap k_v) \cup c_v \end{cases}$$

Note-se que, no caso particular desse problema, a condição $Y_0 = \phi$ é imprescindível para programas. Para procedimentos em geral, no entanto, pode ser introduzida uma condição de contorno que representa as expressões globais disponíveis na entrada do vértice inicial, imediatamente antes de qualquer execução do procedimento (ou seja, aquelas disponíveis em todos os pontos de ativação). Essa condição de contorno pode ser expressa por um conjunto y, e a equação para Y_0 no sistema pode ser substituída por $Y_0 = y \cap \bigcap\limits_{u \in \nabla 0} X_u$. Por questões de simplicidade, porém, iremos adotar o sistema de equações apresentado originalmente.

Sistemas de equações desse tipo podem ter diversas soluções, que nesse caso correspondem a considerar não disponíveis expressões

que de fato o são. Do ponto de vista de utilização desses resultados, entretanto, a solução desejada é aquela que apresenta todas as expressões disponíveis. Pode-se provar que existe uma única solução maximal para sistemas desse tipo, e que nesse caso fornece a solução procurada.

Na prática, os conjuntos envolvidos nesse sistema são representados por vetores característicos (isto é, de "bits"), e as operações entre eles são efetuadas através da conjunção ($\wedge$), disjunção ($\vee$) e negação ($\neg$) lógicas, preservando as prioridades usuais. Continuaremos utilizando os símbolos $\in$, $\notin$, $\subseteq$, $\subset$, $\supset$, $\supseteq$, comuns no contexto de conjuntos.

Empregando essa notação operacional, o sistema para expressões disponíveis terá a forma:

$$\begin{cases} Y_v = \begin{cases} \phi & \text{se } v = 0 \\ \bigwedge\limits_{u \in \nabla v} X_u & \text{se } v \in VG \setminus \{0\} \end{cases} \\ X_v = Y_v \wedge k_v \vee c_v \end{cases}$$

Muitos outros problemas de AGFD podem ser equacionados em sistemas semelhantes, diferindo possivelmente no sentido *progressivo* ou *regressivo* da propagação da informação entre os vértices (no mesmo sentido ou no sentido contrário ao das arestas do grafo, respectivamente), ou quanto ao caráter *conjuntivo* ou

disjuntivo com que elas são agregadas. No caso de problemas disjuntivos, a solução desejada é a minimal, e no caso de problemas regressivos, a condição de contorno seria adotada para os vértices de saída (vértices cuja última instrução pode encerrar a execução do procedimento). A figura 1.4 indica as quatro possibilidades de combinações. Em todas temos $X_v = Y_v \wedge k_v \vee c_v$, onde os coeficientes k e c são obtidos por simples inspeção dos programas. Note-se que em cada uma delas há mais de um exemplo de aplicação, e que o problema das expressões disponíveis considerado anteriormente é um caso de análise progressiva conjuntiva. AHO [1977] apresenta diversos exemplos de problemas em AGFD.

	C O N J U N T I V O	D I S J U N T I V O
Progressivo	$Y_v = \bigwedge_{u \in \nabla v} X_u$	$Y_v = \bigvee_{u \in \nabla v} X_u$
Regressivo	$Y_v = \bigwedge_{w \in \Delta v} X_w$	$Y_v = \bigvee_{w \in \Delta v} X_w$

FIGURA 1.4.

Utilizando as leis de De Morgan é possível transformar um problema conjuntivo num problema disjuntivo (e vice-versa), cuja solução está relacionada de maneira simples à solução do problema original. Problemas progressivos e regressivos também podem ser transformados um no outro, através de uma mudança de orientação nas arestas do grafo e da criação de um vértice inicial. Nesse caso, a solução maximal (minimal) dos dois sistemas será idêntica, entretanto os algoritmos que só são aplicáveis a grafos redutíveis nem sempre podem ser aplicados ao novo grafo obtido.

Devido à facilidade na transformação de problemas entre as classes indicadas e mesmo à semelhança no processo de solução dos sistemas nas diferentes categorias, vamo-nos ater à análise progressiva conjuntiva (APC) desse ponto em diante.

Para encontrar a solução maximal do sistema de equações, seria possível usar um algoritmo linear que constrói, para cada expressão p do universo considerado, uma sequência de conjuntos S_j tais que:

$$S_0 = \begin{cases} \{v \mid k_v[p] = c_v[p] = 0\} \cup \{0\} & \text{se } c_0[p] = 0 \\ \{v \mid k_v[p] = c_v[p] = 0\} & \text{se } c_0[p] = 1 \end{cases}$$

$$S_{k+1} = S_k \cup (\Delta S_k \setminus T)$$

onde $T = \{v \mid c_v[p] = 1\}$. Existe, então, um $0 \leq m \leq |VG|$ tal que $S_m = S_{m+1}$. Se tomarmos $S = S_m$, teremos que

$$X_v[p] = 0 \quad \text{para todo} \quad v \in S$$

$$X_v[p] = 1 \quad \text{para todo} \quad v \notin S.$$

A construção da seqüência S_j sugere um algoritmo de busca em grafo que, apesar de ser linear, deveria ser repetido $|U|$ vezes. Na prática, interessam algoritmos que executam operações sobre os conjuntos em paralelo.

É fácil verificar, observando a forma das equações que compõem o sistema, que a solução maximal não muda se substituirmos cada par (k,c) por $(k \vee c, c)$. Suporemos, então, por conveniência nos dois últimos métodos apresentados, que para todo para (k,c), temos $k \supseteq c$.

CAPÍTULO II

O ALGORITMO ITERATIVO DE HECHT E ULLMAN

Apresentamos na figura 2.1 um algoritmo iterativo muito simples que manipula dois vetores de vetores característicos X e Y, nos quais os u-ésimos componentes representam X_u e Y_u, respectivamente. Inicialmente é atribuído o conjunto vazio a Y_o e o valor $\langle c_o, U, U, \ldots U \rangle$ ao vetor X, e em seguida são operadas aproximações sucessivas, aplicando as equações do sistema aos novos valores obtidos para X e Y até convergir para a solução.

Verificando que os valores obtidos para os vetores X e Y são sucessivamente menores até convergirem para a igualdade no sistema, e que esses valores são sempre maiores ou iguais à solução maximal do sistema, pode-se provar que o algoritmo iterativo converge para essa solução maximal.

Apesar da simplicidade, esse método pode tomar tempo da ordem de $|VG|^2$, como seria, por exemplo, o caso do grafo da figura 2.2, se $k_u = U$ e $c_u = \phi$ para todo $u = 0,1,\ldots, |VG|-1$, e o percurso escolhido fosse sempre na ordem decrescente de numeração dos vértices. Seriam então necessárias $|VG|/2$ iterações (comando 3) para obter todos os $Y_u = \phi$, e mais uma para verificar a convergência. Como cada iteração toma tempo da ordem de $|VG|$, o algoritmo seria da ordem de $|VG|^2$.

Pode-se melhorar o algoritmo substituindo-se a referência a Xant por X na malha mais interna. Entretanto, o tempo continuará sendo da ordem de $|VG|^2$ no pior caso. O próprio exemplo apresentado na figura 2.2 ilustra esse fato.

Algoritmo Iterativo (G)

1. $Y_0 \longleftarrow \phi$

2. $X \longleftarrow \langle c_0, U, \ldots, U \rangle$

3. Repita

   Xant $\longleftarrow$ X

   Para cada v de $VG \setminus \{0\}$ faça

   $Y_v \longleftarrow \bigwedge_{u \in \nabla v} Xant_u$

   $X_v \longleftarrow Y_v \wedge k_v \vee c_v$

   até que X = Xant

4. Devolva X,Y

FIGURA 2.1.

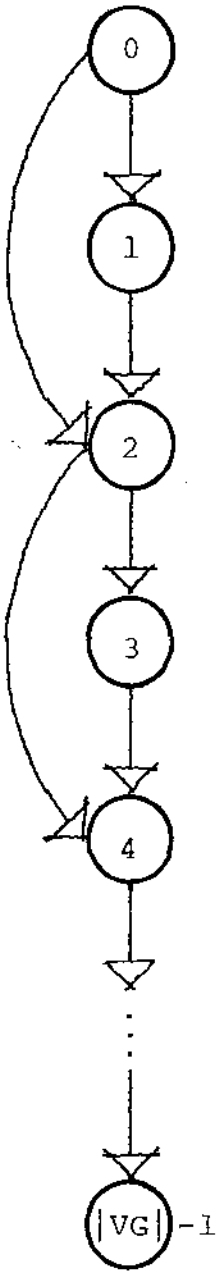


FIGURA 2.2

O algoritmo proposto por Hecht e Ullman (algoritmo HU) é uma variação do algoritmo iterativo, na qual os vértices do grafo são visitados a cada iteração na ordem crescente de uma numeração em profundidade. Um procedimento que implementa o algoritmo HU é apresentado na figura 2.3. Nele, supõe-se que o grafo G é dado pelo número |VG|-1 e pela estrutura de adjacência Δ. Supõe-se também que são parâmetros de entrada dois vetores com |VG| vetores característicos cada, representando os valores de k e c para os vértices de G. A variável Yaux é usada apenas para diminuir o número de subscrições na malha mais interna do procedimento.

O procedimento Visita-Sucessores é usado para gerar a estrutura de incidência ∇ de G e a bijeção Vértice-com-Numeração, inversa de uma numeração em profundidade dos vértices do grafo. O procedimento Inclui, para incluir um nó numa lista simplesmente ligada, é omitido devido à sua simplicidade, porém aparece no apêndice 3.

```
    procedimento  HU(G = (VG,Δ),k,c)
1   ┌ para todo  v  em  VG \{0} faça
2   │     ┌ Por-Visitar [v]  := verdadeiro
3   │     │ ∇[v]             := lista vazia
4   │     └ X[v]             := U
5   │ Por-Visitar [0]  := verdadeiro
6   │ ∇[0]             := lista vazia
7   │ X[0]             := c[0]
8   │ Y[0]             := ϕ
9   │ i                := |VG|-1
10  │ Visita-Sucessores (0)
11  │ M: repita
12  │          estável := verdadeiro
13  │          para i := 1  até  |VG|-1  faça
14  │                ┌ v    := Vértice-com-Numeração [i]
15  │                │ Xant := X[v]
16  │                │ Yaux := U
17  │                │ para toda aresta  a  em  ∇[v]  faça
18  │                │      ┌ u    := a.origem
19  │                │      └ Yaux := Yaux ∧ X[u]
20  │                │ Y[v]  := Yaux
21  │                │ X[v]  := Yaux ∧ k[v] ∨ c[v]
22  │                │ se  X[v] ≠ Xant
23  │                └ então estável := falso
24  │    até estável
25  └ devolva  Y

    procedimento Visita-Sucessores (u)
26  ┌ Por-Visitar [u]  := falso
27  │ para toda aresta  a  em  Δ[u]  faça
28  │     ┌ v := a.destino
29  │     │ Inclui(∇[v],u)
30  │     │ se Por-Visitar [v]
31  │     └ então Visita-Sucessores (v)
32  │ Vértice-com-Numeração [i]  := u
33  └ i := i - 1
```

FIGURA 2.3.

Como a ordem de visita é apenas uma particularização da ordem arbitrária utilizada no algoritmo iterativo simples, concluímos que o procedimento HU está correto.

PROPOSIÇÃO (HECHT [1975]).

Sejam G um grafo redutível e d o maior número de arestas de retorno em qualquer caminho em G. A ordem de visita importa pelo algoritmo HU estabelece um limite superior de $d+2$ no número máximo de vezes que a iteração M é executada.

PROVA

A princípio $X_u = U$ para todo $u \neq 0$. Cada X_u só pode ser alterado se, para alguma expressão p, $c_u[p] = 0$ e além disso $k_u[p] = 0$ ou existe um vértice v tal que $X_v[p] = 0$ e u é alcançável de v por um caminho $C = (v, w_1, w_2, \ldots, w_m, u)$ no qual $c_{w_i}[p] = 0$, $1 \leq i \leq m$. Se tivermos $c_u[p] = k_u[p] = 0$, teremos $X_u[p] = 0$ logo após a primeira iteração. Caso contrário, se o caminho C não passa por arestas de retorno, a numeração dos vértices é sempre crescente, e conseqüentemente $X_u[p] = 0$ também após uma única iteração.

Agora sejam $(a_1, b_1), (a_2, b_2), \ldots, (a_{d'}, b_{d'})$ as arestas de retorno de C na ordem em que aparecem nesse caminho. Ao final da primeira iteração, $X_{w_i}[p] = 0$ para todo w_i anterior a a_1,

inclusive, em C. Após cada nova iteração n, n = 2, 3 , . . . , d' , $X_{w_i}[p] = 0$ para todo w_i anterior a a_n, inclusive, em C, de forma que ao final da d'-ésima iteração $X_{w_i}[p] = 0$ para todo w_i anterior a $a_{d'}$, inclusive, em C. Após a iteração seguinte, $X_w[p] = 0$ para todo w em C, inclusive u.

(Note-se que agora é importante a utilização dos valores X_u mais recentes, de forma a propagar informações entre vértices ligados por um caminho sem arestas de retorno, numa única iteração. No caso do exemplo apresentado na figura 2.2, temos que d = 0, logo seriam necessárias duas iterações apenas para executar o procedimento HU; entretanto, se ao invés da referência a X_u tivéssemos utilizado $Xant_u$, seriam necessárias ($|VG|/2$) + 1 iterações).

Em todos os casos, uma iteração a mais é necessária para verificar a convergência das equações.

Como d' $\leq$ d, como uma vez igual a zero qualquer coordenada do vetor X_u não se altera, e como esse raciocínio se aplica a qualquer vértice u do grafo e qualquer expressão p, podemos concluir que o número máximo de iterações executadas num grafo redutível qualquer é d+2.

A figura 2.4 mostra o esquema genérico de um caminho com origem num vértice v e término num vértice u, com d' arestas de retorno, num grafo redutível.

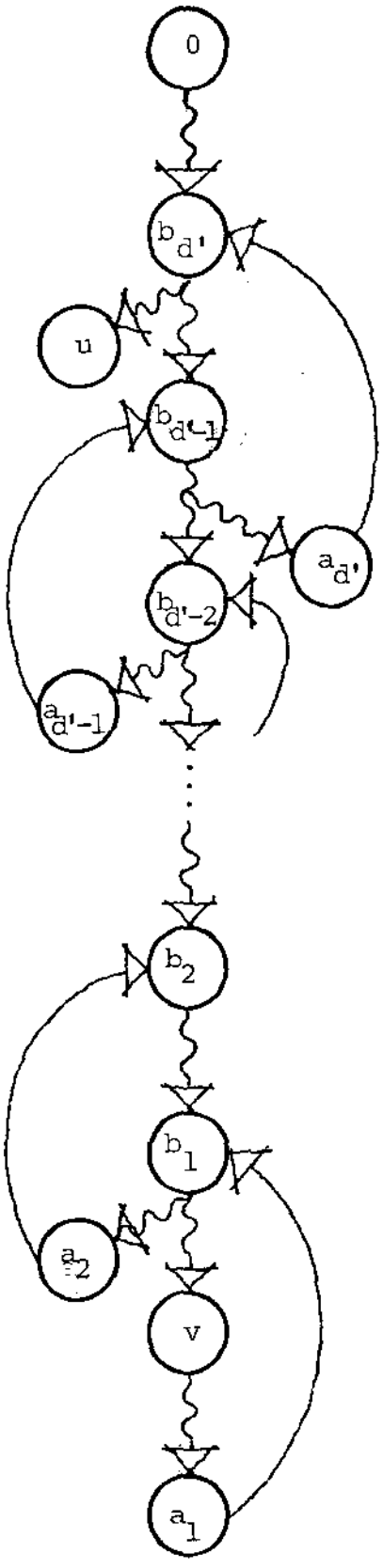


FIGURA 2.4

Se o grafo não for redutível, o número de vezes que a iteração é executada tem limite $D \leq d' + 2$, onde d' é o maior d obtenível eliminando arestas de G até torná-lo redutível. Isso porque se $G' = G \setminus \{\alpha_1, \alpha_2, \ldots, \alpha_j\}$ tem no máximo d' arestas de retorno num caminho qualquer, então no máximo $d' + 1$ iterações seriam necessárias para a convergência do algoritmo HU aplicado a G'. Como o acréscimo de arestas a um grafo nunca retarda a convergência (algumas vezes até acelera), também para G o algoritmo HU converge em no máximo $d' + 1$ iterações.

O corpo da malha na linha 1 é executado (N_v-1) vezes e o teste sobre a variável de controle é efetuado antes, de forma que teremos $N_v(2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$ e $(N_v-1)\mu_{ar}$ no seu controle. Como já foi dito anteriormente, o corpo da malha na linha 11 é executado (considerando o pior caso) $(d+2)$ vezes, sendo utilizados no controle $(d+2)(\mu_{ac} + \mu_c)$ e $(d+1)\mu_d$. Já o corpo da malha na linha 13 é executado (N_v-1) vezes a cada uma das $(d+2)$ entradas. Como o teste deve ser anterior à execução, e o valor limite é dado por uma variável, esse controle demanda $(d + 2)$ $N_v(2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$ mais $(d+2)(N_v-1)(\mu_{ac} + \mu_{ar})$.

Computando todas as $(d+2)(N_v-1)$ entradas na malha da linha 17, o seu corpo é executado $(d+2)(N_a - N_1)$ vezes. Como todo vértice exceto a raiz tem pelo menos um predecessor, o teste é efetuado depois, e serão usadas no controle $(d+2)(N_a - N_1)(\mu_{ac}+\mu_{av}+\mu_c)$, $(d+2)(N_v-1)(\mu_{ac} + \mu_{at})$, e $(d+2)(N_a - N_1 - N_v+1)\mu_d$, além de $(d+2)(N_v-1)$

μ_s que são utilizadas na primeira atribuição à variável de controle a cada entrada.

Na linha 19 são executadas $(N_a - N_1)(d+2)\mu_{OL}$. Esse número poderia ser diminuído de (N_v-1) (em detrimento de um igual número de desvios) a cada uma das d+2 iterações, se a atribuição da linha 16 fosse substituída por Yaux := X[V[v].origem], entretanto por uma questão de clareza foi escolhida a forma utilizada no procedimento. Na linha 22 ocorrem $(d+2)(N_v-1)$ comparações de vetores de bits (μ_c), que serão mais tarde contadas juntamente com as operações lógicas sobre aqueles vetores.

Finalmente, o corpo do procedimento Visita-Sucessores é executado N_v vezes (é fácil comprovar que cada vértice é usado como argumento de chamada exatamente uma vez). O corpo da malha na linha 27 é executado no total N_a vezes, e como o teste ocorre antes da execução, serão necessárias $N_v(2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$ e $N_a(\mu_{ac} + \mu_{av} + \mu_c + \mu_d)$, além de N_v subscrições, uma a cada entrada na malha, para a primeira atribuição à variável de controle.

A figura 2.5 indica o número de operações por linha (ou grupo de linhas) no procedimento HU, e a figura 5.6 apresenta o número de vezes que cada operação básica é executada em todo o procedimento. Nesse último quadro, cada ocorrência de μ_{av} foi substituída por $(\mu_{ac} + \mu_{at} + \mu_{sel})$ e o custo de μ_{Inclui} foi traduzido por $(9\mu_{ac} + 2\mu_{ar} + 5\mu_{at} + \mu_c + \mu_d + 2\mu_{sel})$.

CUSTO DO PROCEDIMENTO HU

1.	N_v	$(2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$
2.	(N_v-1)	μ_{ar}
2+..+6+8.	N_v	$(2\mu_{at} + \mu_{AT} + 3\mu_s)$
7.	1	$(\mu_{AC} + \mu_{AT} + 2\mu_s)$
9.	1	$(\mu_{ac} + \mu_{at})$
10+31.	N_v	$(\mu_{cpl} + \mu_{rp})$
12.	$(d+2)$	μ_{at}
13.	$(d+2)$ N_v	$(2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$
	$(d+2)(N_v-1)$	$(\mu_{ar} + \mu_{ac})$
14+..+16+20+..+22		
	$(d+2)(N_v-1)$	$(7\mu_{AC} + \mu_{at} + 4\mu_{AT} + \mu_C + 2\mu_{OL} + 7\mu_s + \mu_{ac})$
22.	$(N_v - 1)$	μ_d
17.	$(d+2)(N_v-1)$	$(\mu_s + \mu_{ac} + \mu_{at})$
	$(d+2)(N_a-N_l)$	$(\mu_{av} + \mu_{ac} + \mu_c)$

Figura 2.5a

17.	$(d+2)(N_a-N_l-N_v+1)$	μ_d
18+19.	$(d+2)(N_a-N_l)$	$(2\mu_{AC} + \mu_{at} + \mu_{AT} + \mu_{OL} + \mu_s + \mu_{sel} + \mu_{ac})$
23.	$(d+1)(N_v-1)$	μ_{at}
24.	$(d+2)$	$(\mu_{ac} + \mu_c)$
	$(d+1)$	μ_d
26.	N_v	$(\mu_{at} + \mu_s)$
27.	N_v	$(\mu_s + 2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$
	N_a	$(\mu_{ac} + \mu_{av} + \mu_c + \mu_d)$
28.	N_a	$(\mu_{ac} + \mu_{at} + \mu_{sel})$
29.	N_a	$(\mu_s + \mu_{Inclui})$
30.	N_a	$(\mu_{ac} + \mu_c + \mu_s)$
	(N_a-N_v+1)	μ_d
32+33.	N_v	$(2\mu_{ac} + \mu_{ar} + 2\mu_{at} + \mu_s)$

Figura 2.5b

SOMA VERTICAL DE OPERAÇÕES NO PROCEDIMENTO HU

$(3d+19)$	N_a	$+ (5d+16)$	N_v	$- 3(d+2)N_l$	$- 2d$	$-$	3			μ_{ac}
$(2(d+2)$	N_a	$+ 7(d+2)$	N_v	$- 2(d+2)N_l$	$- 7d$	$-$	13			μ_{AC}
2	N_a	$+ (d+4)$	N_v		$- d$	$-$	3			μ_{ar}
$(2d+11)$	N_a	$+ (4d+14)$	N_v	$- 2(d+2)N_l$	$- 2d$	$-$	2			μ_{at}
$(d+2)$	N_a	$+ (4d+9)$	N_v	$- (d+2)N_l$	$- 4d$	$-$	7			μ_{AT}
$(d+5)$	N_a	$+ (d+4)$	N_v	$- (d+2)N_l$	$+ d$	$+$	2			μ_c
		$(d+2)$	N_v		$- d$	$-$	2			μ_C
			N_v							μ_{cpl}
$(d+5)$	N_a	$+ 2$	N_v	$- (d+2)N_l$	$+ 2d$	$+$	3			μ_d
$(d+2)$	N_a	$+ (2d+4)$	N_v	$- (d+2)N_l$	$- 2d$	$-$	4			μ_{OL}
			N_v							μ_{rp}
$(d+4)$	N_a	$+ (8d+22)$	N_v	$- (d+2)N_l$	$- 8d$	$-$	14			μ_s
$(2d+8)$	N_a	$+$		$- (2d+4)N_l$						μ_{sel}

FIGURA 2.6

CAPÍTULO III

A ABORDAGEM POR INTERVALOS

3.1. INTRODUÇÃO

Uma vez que o método aqui descrito só se aplica a grafos redutíveis, quando utilizarmos a palavra grafo nesse capítulo, estaremo-nos referindo a essa classe.

Ao estudar o algoritmo HU, é fácil observar que se o grafo não apresentasse circuitos, seria necessária uma única visita a cada um dos vértices v para determinar definitivamente os valores de X_v e Y_v.

Suponhamos agora que todas as arestas de retorno de um grafo têm como destino um único vértice u. Após uma iteração do algoritmo HU, é possível que alguns dos vértices v alcançáveis de u (inclusive u) tenham ainda valores incorretos para os seus X_v e Y_v. Isso porque Y_u foi baseado nos $X'_p s$ de seus predecessores, que podem ter mudado em função de um $X_t \neq U$ propagado pelas arestas. Entretanto, uma segunda visita a cada vértice será suficiente para determinar os $X'_p s$ e $Y'_p s$ corretos. (Note-se que, na realidade, o algoritmo HU faz uma iteração adicional em cada caso, para determinar se foi atingida a condição de parada).

Em geral, se forem conhecidos os $Y'_u s$ de todos os vértices de um grafo que são destino de arestas de retorno, será possível determinar os $X'_v s$ e $Y'_v s$ de todos os blocos de uma única iteração.

O método de solução por intervalos utiliza essas idéias, e pode ser descrito para um grafo $G = (VG, aG, n_o)$ por:

1. Se G é trivial, então $Y_{n_o} \leftarrow \phi$.
2. Caso contrário, particione VG em subconjuntos maximais tendo cada um deles um único vértice *raiz*, de forma que se fossem dados os $Y'_r s$ dessas raízes, seria possível determinar os $X'_v s$ e $Y'_v s$ de todos os outros vértices numa única iteração. Os subgrafos gerados por esses conjuntos serão chamados *intervalos*.
3. Considere um grafo derivado G' no qual os vértices correspondem a cada um dos intervalos de G, e as arestas correspondem à existência de arestas entre vértices de dois intervalos distintos, e representam os $X'_p s$ das origens dessas arestas.
4. Use o método recursivamente para determinar os $X'_v s$ e $Y'_v s$ dos vértices de G', obtendo assim os $Y'_v s$ dos vértices raízes de intervalos no grafo G.
5. Determine os $Y'_v s$ dos vértices de cada intervalo do grafo G na ordem em que foram admitidos nestes.

Ao mesmo tempo que há, neste algoritmo, uma transformação

dos grafos, deverá haver uma transformação das funções k e c associadas, de modo a possibilitar a obtenção da solução do problema no grafo original. Essas transformações serão indicadas na secção 3.3.

Passaremos agora às definições e propriedades do método.

3.2. INTERVALOS

DEFINIÇÕES

Dado um vértice r de um grafo $G = (VG, aG, n_o)$, chamaremos de *subintervalo* de *raiz* r o subgrafo maximal $I(r) = G[VI]$ tal que as seguintes condições são satisfeitas:

i) $r \in VI$;

ii) $G[VI \setminus \{r\}]$ não contém circuitos ou laços (isto é, todo circuito ou laço de $I(r)$ passa por r);

iii) Se $v \in VI \setminus \{r\}$ então $v \neq n_o$ e $\nabla v \subseteq VI$.

O subintervalo $I(r)$ será um *intervalo* em G, com *raiz* r, se $I(r)$ for maximal no conjunto dos subintervalos de G, ou seja, se não existir em G um subintervalo $I'(r')$ tal que $VI \subsetneq VI'$.

Quando não for relevante, omitiremos a raiz nesta notação, indicando apenas o grafo I que é o subintervalo.

A figura 3.1 apresenta um grafo G com seus oito intervalos,

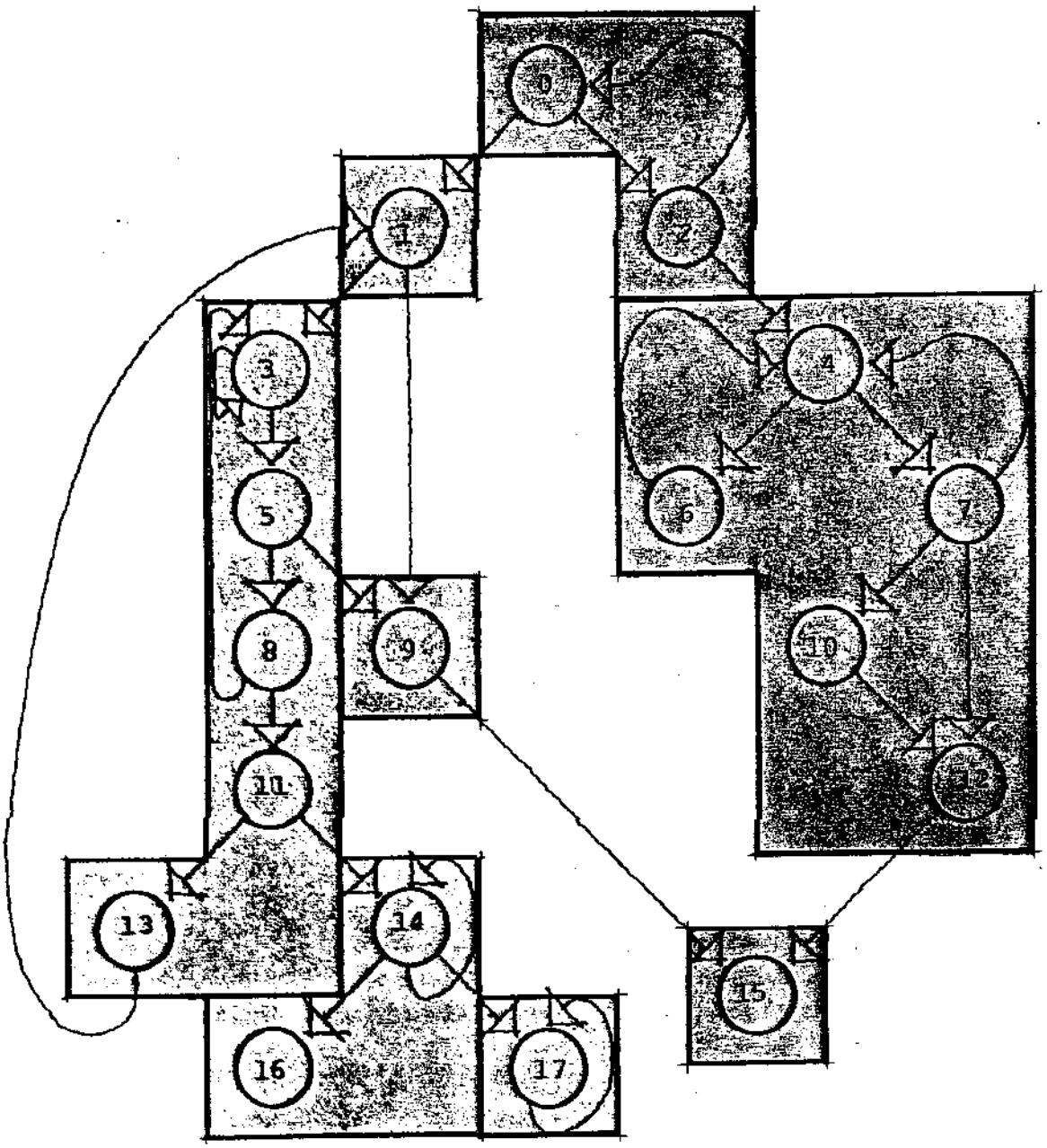


FIGURA 3.1

de raízes 0, 1, 3, 4, 9, 14, 15 e 17. Os subintervalos com raízes 1, 2, 6, 9, 10, 12, 13, 15,16 e 17 são grafos triviais. Os subintervalos de G que não são grafos triviais nem intervalos, são $I(5) = G[\{5,8,11,13\}]$, $I(8) = G[\{8,11,13\}]$, $I(11) = G[\{11,13\}]$, e $I(7) = G[\{7,10,12\}]$.

PROPRIEDADE 3.2.1.

Todos os vértices de um subintervalo são dominados pela raiz.

PROVA: É fácil verificar que a existência de um vértice no intervalo $I(r)$ não dominado por r, implica na existência de um vértice $v \neq r$ em VI com predecessor u que não pertence a VI. □

PROPRIEDADE 3.2.2.

Se $I(r)$ e $J(r)$ são dois subintervalos (com a mesma raiz), então $I(r) = J(r)$.

PROVA: Suponhamos que $I \neq J$. Então, pela maximalidade de ambos teremos $VI \setminus VJ \neq \phi$ e $VJ \setminus VI \neq \phi$. Consideremos o grafo $I = G[VI \cup VJ]$. Claramente I satisfaz as propriedades (i) e (iii) dos subintervalos para a raiz r. Se $(VI \cup VJ) \setminus \{r\}$ contém todos os vértices de um circuito C em G, então existe pelo menos um vértice v em VC que está em $VI \setminus VJ$. Seja $C = (v, w_1, w_2, \ldots, w_n, v)$ e seja $w = w_i$ tal que i é o menor elemento de $\{j \mid 1 \leq j \leq n \text{ e } w_j \in VJ\}$.

w não satisfaz a propriedade (iii) dos subintervalos em VJ, o que contradiz a hipótese. Logo, I e r satisfazem as três propriedades dos subintervalos. Conseqüentemente, I e J não são maximais, e portanto não são subintervalos. Temos então que I = J. □

COROLÁRIO 3.2.1.

Para cada vértice r num grafo G, existe um único subintervalo de raiz r.

PROPRIEDADE 3.2.3.

Dados dois subintervalos $I(r_I)$ e $J(r_J)$ quaisquer, ou $VI \subseteq VJ$, ou $VJ \subseteq VI$, ou $VI \cap VJ = \phi$.

PROVA: Sejam $I(r_I) \neq J(r_J)$ (isto é, $r_I \neq r_J$) tais que $VI \cap VJ \neq \phi$, e seja $v \in VI \cap VJ$. Então r_I domina v e r_J domina v, logo, pela definição da relação domina, r_I domina r_J ou r_J domina r_I. Sem perda de generalidade, suponhamos que r_I domina r_J. É fácil ver então que $r_J \in VI$.

Suponhamos agora que $VJ \setminus VI \neq \phi$. Como r_J domina v, para todo v em $VJ \setminus VI$, e $r_J \in VI$, existe um $w \in VJ \setminus VI$ com todos os predecessores em VI. Então, pela maximalidade de I e pela propriedade (ii) de subintervalos, w é um vértice de um circuito C em G, tal que todos os outros vértices de C estão

em VI, e C não passa por r_I. Mas nesse caso, o sucessor z de w em C não terá todos os seus predecessores em I, logo z não estará em I, e w poderia ser incluído nesse subintervalo. Donde se conclui que não existe um tal w, e portanto VJ $\subseteq$ VI.□

COROLÁRIO 3.2.2.

Dado um grafo G = (VG, aG, n_o), existe uma única partição de VG em conjuntos de vértices que geram intervalos.

PROVA: Segue imediatamente do corolário 3.2.1 e da propriedade 3.2.3.□

O fato de que os vértices dos intervalos de G definem uma partição de VG, será usado nos algoritmos que determinam essa partição, evitando que seja identificado o subintervalo cuja raiz é um vértice que já foi incluído em algum outro subintervalo.

ALGORITMO PARA IDENTIFICAÇÃO DE INTERVALOS

A própria caracterização dos subintervalos sugere um algoritmo para a sua identificação num dado grafo G. Sejam r a raiz dada e I o subintervalo procurado. VI pode ser obtido por:

1. Atribua inicialmente {r} a VI;
2. Enquanto houver (maximalidade) vértice v tal que $\nabla v \neq \phi$ e $\nabla v \subseteq$ VI, acrescente v a VI.

Note-se que a condição no passo 2 garante que todo circuito passará necessariamente pela raiz, porque todo vértice $v \neq r$ exige, em particular, a presença em VI do seu predecessor no circuito para só então entrar em VI. Essa condição garante também a possibilidade de cada vértice do subintervalo ser visitado apenas quando todos os seus predecessores já o foram. Esse efeito é semelhante ao da numeração em profundidade, local a cada subintervalo.

O passo 2 pode ser executado observando-se cada sucessor v dos vértices já incluídos em VI. Isso porque pela propriedade (iii), um vértice v que deve estar em VI tem todos e pelo menos um predecessor em VI, e depois que o último deles, u, for acrescentado ao subintervalo, a observação dos sucessores de u garantirá a inclusão de v em VI.

Naturalmente, se r é uma raiz de intervalo no grafo G, esse algoritmo identifica o intervalo com raiz r. Esse processo pode, portanto, ser utilizado para identificar os intervalos de um grafo G, sendo necessário apenas determinar as raízes de intervalo em G. Isso pode ser feito da seguinte maneira: como o vértice inicial do grafo é sempre raiz de intervalo, ele será a primeira raiz tomada. Após a identificação de cada intervalo J, aqueles vértices que são sucessores de vértices no intervalo J e não foram incluídos neste, serão tomados como raízes de outros intervalos. Esse algoritmo baseia-se no fato de que toda raiz

$r \neq n_o$ tem pelo menos um predecessor em um intervalo $J \neq I(r)$, que eventualmente será identificado (logo toda raiz de intervalo será detectada), e no fato de que se um subintervalo $I(v)$ não é maximal, então existe um subintervalo J tal que $VI \subset VJ$ e $\nabla v \subseteq VJ$, logo v jamais será tomado como sucessor de qualquer vértice em $VG \setminus VJ$.

Esse algoritmo para identificação dos intervalos de um grafo pode ser implementado pelo procedimento indicado na figura 3.2. Supõe-se que são disponíveis as estruturas de adjacência (Δ) e de incidência (∇) do grafo G. Tanto a lista Raízes como o conjunto (vetor característico) R, representam os vértices que serão raízes de intervalos,e essa redundância se deve à facilidade nas operações de seleção (Retira) e de teste de pertinência, respectivamente. A estrutura Intervalos é um vetor de listas, no qual cada componente I apresentará os vértices do I-ésimo intervalo identificado. É esse vetor Intervalos que será devolvido como resultado do procedimento.

Serão omitidos os procedimentos Inclui e Retira, responsáveis respectivamente pela inclusão e pela retirada de um vértice numa lista simplesmente ligada. A função booleana Vazio tem implementação e utilização óbvias.

Note-se que, apesar de aparentemente ser linear (as duas malhas mais internas, M_1 e M_2, são executadas no total $|aG|$ vezes cada uma), o procedimento da figura 3.2. executa da ordem de $|VG|$

```
procedimento Identifica-Intervalos (G = (VG,Δ,∇))
  Raízes := lista vazia
  Inclui (Raízes, 0)
  R      := {0}
  I      :=  0 /* I representará o número do intervalo corrente */
  repita
    Retira (Raízes, r)
    Intervalos [I] := lista vazia
    Inclui (intervalos [I], r)
    VI   := {r}
    para todo  u  em Intervalos [I] faça
      M1: para toda aresta  a  em  Δ[u] faça
            v := a.destino
            se  ∇v ⊆ VI  ∧  v ∉ VI
            então Inclui (Intervalos [I], v)
                  VI := VI ∪ {v}

    para todo  u  em Intervalos [I] faça
      M2: para toda aresta  a  em  Δ[u] faça
            v := a.destino
            se  v ∉ VI  ∧  v ∉ R
            então Inclui (Raízes, v)
                  R := R ∪ {v}
    I:= I + 1
  até Vazio (Raízes)
  devolva Intervalos
```

FIGURA 3.2.

operações para verificar a condição $\nabla v \subseteq VI$, da ordem de $|aG|$ vezes. Isso porque as operações entre conjuntos são efetuadas em termos de operações lógicas sobre vetores característicos, e portanto dependem do tamanho de VG.

Um outro algoritmo, adaptado de HECHT [1977], menos imediato porém linear, usa uma estratégia diferente para verificar a condição $\nabla v \subseteq VI$ para um dado vértice $v \neq n_o$: se durante a visita aos sucessores dos vértices num certo intervalo I, o vértice v for alcançado $|\nabla v|$ vezes, então v será incluído em VI. Um vértice v será acrescentado ao conjunto das raízes (R), se v for visitado pela primeira vez durante a identificação de um intervalo I, e ao final dessa identificação $v \notin VI$ (o que significa que v tem predecessores em mais de um intervalo). Esse algoritmo está indicado na figura 3.3.

Uma implementação desse algoritmo é apresentada na figura 3.4. Nela, a lista Raízes e o vetor característico R são usados conjuntamente porque um vetor característico não permite a seleção (Retira) de diversos elementos em tempo linear. Por outro lado, a utilização independente da lista Raízes requereria um vetor com apontadores para elementos da lista, para propiciar a eliminação de diversos elementos em tempo linear, além de ser uma operação mais demorada que a eliminação num vetor característico.

O vetor Cont controla o número de visitas a cada vértice, enquanto o vetor Assoc é usado para determinar a primeira visita a

Algoritmo Encontra-Intervalos (G)

1. $R \leftarrow \{n_o\}$

2. Para todo vértice r em R execute os passos 3 e 4

3. $VI \leftarrow \{r\}$

4. Para todo u em VI

   para todo v em Δu

   - se v é visitado pela primeira vez
     - então
       - associe v com o intervalo corrente
       - $R \leftarrow R \cup \{v\}$ /* temporariamente */
   - se v é visitado pela $|\nabla v|$-ésima vez
     - então se v está associado ao intervalo corrente
       - então
         - $VI \leftarrow VI \cup \{v\}$
         - $R \leftarrow R \setminus \{v\}$

FIGURA 3.3.

```
procedimento Encontra-Intervalos (G = (VG,Δ))
  Raízes := lista vazia
  Inclui (Raízes, 0)
  I := 0              /* I é o número do intervalo corrente */
  R := {0}
  para todo u em VG faça
    Cont [u] := 0
    Assoc [u] := ∞      /* valor indefinido */
  para todo u em VG faça
    para toda aresta a em Δ[u] faça
      v := a.destino
      Cont [v] := Cont [v] + 1
  repita
    Retira (Raízes, r)
    se r ∈ R
    então Intervalos [I] := lista vazia
          Inclui (Intervalos [I], r)
          para todo u em Intervalos [I] faça
              para toda aresta a em Δ[u] faça
                v := a.destino
                se Assoc [v] = ∞
                então Assoc [v] := I
                      Inclui (Raízes, v)
                      R := R ∪ {v} /* temporariamente */
                Cont [v] := Cont [v] - 1
                se Cont [v] = 0
                então se Assoc [v] = I
                        então Inclui (Intervalos [I],v)
                              R := R \ {v}
          I := I + 1
  até Vazio (Raízes)
  devolva Intervalos
```

FIGURA 3.4.

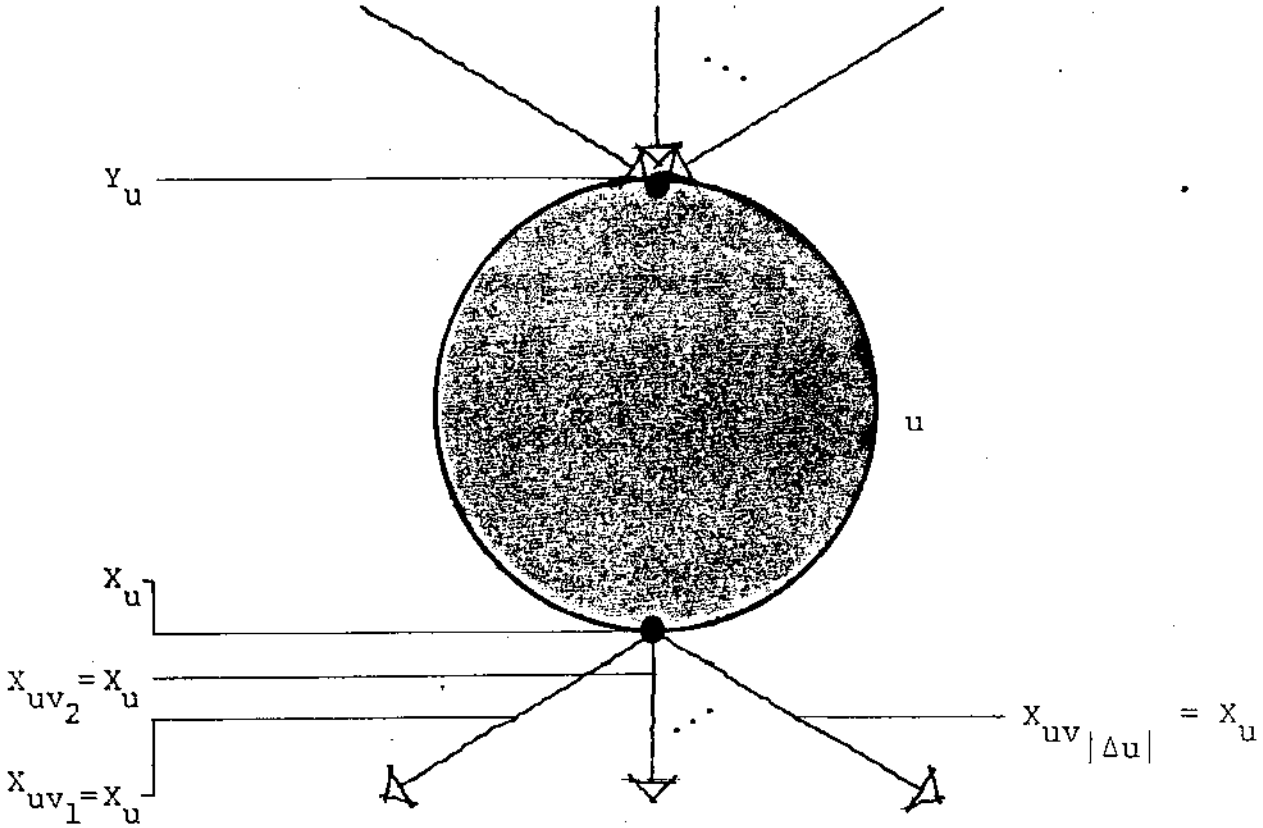
um vértice (se seu valor for indefinido), e para associá-lo a um dado intervalo nessa ocasião. Os procedimentos Inclui e Retira,e a estrutura Intervalos são compatíveis com o procedimento da figura 3.2.

3.3. O GRAFO DERIVADO G'

Como veremos adiante, é conveniente nesse método associar os valores k, c, e X com cada aresta de um grafo, e não com os vértices deste. É o que faremos daqui em diante nesse capítulo.

No grafo inicial dado, o par (k,c) associado a uma aresta será aquele a princípio associado à sua origem. A figura 3.5 ilustra essa alteração, e mostra o novo sistema de equações obtido. Essa transformação origina um sistema com um número de variáveis maior ou igual ao número de variáveis no sistema original, porém os dois sistemas são equivalentes, uma vez que as soluções obtidas para os Y'_us do novo sistema são claramente as mesmas obtidas para o sistema original, e que os X'_{uv}s obtidos são iguais aos X'_us do grafo original. Note-se que para um vértice sem sucessores, ou os valores de X são irrelevantes ou então podemos acrescentar um vértice fictício (vazio) introduzindo arestas convenientes.

Se fosse dado o Y_r do vértice raiz de um intervalo num grafo G, os Y'_us de todos os outros vértices desse intervalo



$$Y_u = \bigwedge_{w \in \nabla u} X_{wu}$$

$$X_{uv_i} = Y_u \wedge k_{uv_i} \vee c_{uv_i} \qquad i = 1,2,\ldots,|\Delta u|$$

$$\text{com} \quad k_{uv_i} = k_u \quad \text{e} \quad c_{uv_i} = c_u$$

FIGURA 3.5

poderiam ser determinados, pois todos os seus predecessores pertencem ao intervalo: bastaria aplicar as equações do sistema a esses vértices, na ordem em que foram incluídos. O método de análise por intervalos consiste justamente em determinar os Y_r's das raízes dos intervalos, e porteriormente encontrar a solução total.

Utilizando o sistema de equações para G, podemos mostrar, por indução na ordem de entrada de um vértice u num intervalo $I(r_I)$ de G, que $X_{uv} = Y_{r_I} \wedge \bar{k}_{uv} \vee \bar{c}_{uv}$, onde

$$\bar{k}_{uv} = \begin{cases} k_{uv} \vee c_{uv} & \text{se} \quad u = r_I \\ [\bigwedge_{w \in \nabla u} (\bar{k}_{wu} \vee \bar{c}_{wu}) \wedge k_{uv}] \vee c_{uv} & \text{se} \quad u \neq r_I \end{cases}$$

$$\text{e} \quad \bar{c}_{uv} = \begin{cases} c_{uv} & \text{se} \quad u = r_I \\ [(\bigwedge_{w \in \nabla u} \bar{c}_{wu}) \wedge k_{uv}] \vee c_{uv} & \text{se} \quad u \neq r_I \end{cases}$$

Claramente teremos $\bar{k}_{uv} \supseteq \bar{c}_{uv}$ para toda aresta (u,v) em aG.

(Note-se que se $E = \bigwedge_{i \in H} (\alpha_i \wedge z \vee \beta_i)$, onde α_i e β_i são expressões quaisquer e z não depende de i, então $E = \gamma \wedge z \vee \delta$, com $\gamma = \bigwedge_{i \in H} (\alpha_i \vee \beta_i)$ e $\delta = \bigwedge_{i \in H} \beta_i$.)

Seja $I(r_I)$ um intervalo, e (u,v) uma aresta com $u \in VI$, e denotemos por P o conjunto dos caminhos de r_I a v, contendo a aresta (u,v). Podemos dizer intuitivamente, em termos do problema das expressões disponíveis, que para uma expressão $q \in U$, $q \in \bar{c}_{uv}$ se em todos os caminhos de C de P, q é gerada numa dada aresta (s,t), e preservada por todas as arestas da seção de C que vai de t a v, e que $q \in \bar{k}_{uv}$ se ao longo de todos os caminhos C de P, ou q é preservada por todas as arestas de C, ou q é gerada em C e preservada posteriormente nesse caminho.

Como a única informação relevante na determinação de um Y_u é a "influência exercida" pelos X'_{wu}s (a topologia em si não é importante), para determinar os Y'_rs das raízes, pode-se definir um grafo G', dito derivado de G, no qual cada intervalo $I(r_I)$ de G é reduzido a um único vértice I, e as arestas de G que partem de vértices no intervalo I para a raiz r_J do intervalo J serão representadas pela aresta (I,J) de G'. (É fácil verificar que $I(n_o)$ é um intervalo, e será tomado como vértice inicial de G'). Note-se que o grafo derivado G' não tem laços. (Todas as quantidades associadas com G' receberão apóstrofo , inclusive os parâmetros N_v e N_a usados no cálculo do custo.)

Uma vez que a nova aresta (I,J) deve substituir o efeito das arestas por ela representadas, tomemos $X'_{IJ} = \bigwedge_{w \in \nabla r_J \cap VI} X_{wr_J}$. Como para duas raízes de intervalos distintos, r_J e r_L, temos

em geral $\nabla r_J \cap VI \neq \nabla r_L \cap VI$, obteremos $X'_{IJ} \neq X'_{IL}$ para dois sucessores distintos J e L do vértice I em G'. Daí a conveniência de associar os X's às arestas, mencionada anteriormente. (Veja a figura 3.6).

Por outro lado, pelo sistema de equações original,

$$Y_{r_I} = \bigwedge_{w \in \nabla r_I} X_{wr_I} = \bigwedge_{u \in (\nabla r_I \setminus V_I)} X_{ur_I} \wedge \bigwedge_{v \in (\nabla r_I \cap VI)} X_{vr_I} .$$

Tomemos $Y'_I = \bigwedge_{L \in \nabla'_I} X'_{LI}$. Assim, pela definição de X'_{LI}

$$Y'_I = \bigwedge_{u \in (\nabla r_I \setminus VI)} X_{ur_I} \quad \text{e} \quad Y_{r_I} = Y'_I \wedge \bigwedge_{v \in (\nabla r_I \cap VI)} X_{vr_I} .$$

$$\therefore Y_{r_I} = Y'_I \wedge \bigwedge_{w \in (\nabla r_I \cap VI)} (Y_{r_I} \wedge \bar{k}_{wr_I} \vee \bar{c}_{wr_I})$$

$$= Y'_I \wedge [Y_{r_I} \wedge \bigwedge_{w \in (\nabla r_I \cap VI)} (\bar{k}_{wr_I} \vee \bar{c}_{wr_I}) \vee \bigwedge_{w \in (\nabla r_I \cap VI)} \bar{c}_{wr_I}]$$

$$= Y'_I \wedge [Y_{r_I} \wedge \bigwedge_{w \in (\nabla r_I \cap VI)} \bar{k}_{wr_I} \vee \bigwedge_{w \in (\nabla r_I \cap VI)} \bar{c}_{wr_I}]$$

$$\therefore Y_{r_I} = Y'_I \wedge [Y_{r_I} \wedge K_i \vee C_I]$$

com $K_I = \bigwedge_{w \in (\nabla r_I \cap VI)} \bar{k}_{wr_I}$ e $C_I = \bigwedge_{w \in (\nabla r_I \cap VI)} \bar{c}_{wr_I}$.

A solução maximal desta equação é dada por:

$$Y_{r_I} = Y'_I \wedge [K_I \vee C_I]$$

Mas $K_I \vee C_I = K_I$, pois $K_I \supseteq C_I$, donde $Y_{r_I} = Y'_I \wedge K_I$.

Retomando a definição de X'_{IJ} , teremos:

$$X'_{IJ} = \bigwedge_{w \in (\nabla r_J \cap VI)} X_{wr_J}$$

$$= \bigwedge_{w \in (\nabla r_J \cap VI)} (Y_{r_I} \wedge \bar{k}_{wr_J} \vee \bar{c}_{wr_J})$$

$$= \bigwedge_{w \in (\nabla r_J \cap VI)} [(Y'_I \wedge K_I) \wedge \bar{k}_{wr_J} \vee \bar{c}_{wr_J}]$$

$$= (Y'_I \wedge K_I) \wedge \bigwedge_{w \in (\nabla r_J \cap VI)} (\bar{k}_{wr_J} \vee \bar{c}_{wr_J}) \vee \bigwedge_{w \in (\nabla r_J \cap VI)} \bar{c}_{wr_J}$$

$$= Y'_I \wedge K_I \wedge \bigwedge_{w \in (\nabla r_J \cap VI)} \bar{k}_{wr_J} \vee \bigwedge_{w \in (\nabla r_J \cap VI)} \bar{c}_{wr_J}$$

Colocando $k'_{IJ} = K_I \wedge \bigwedge_{w \in (\nabla r_J \cap VI)} \bar{k}_{wr_J}$ e $c'_{IJ} = \bigwedge_{w \in (\nabla r_J \cap VI)} \bar{c}_{wr_J}$,

temos $X'_{IJ} = Y'_I \wedge k'_{IJ} \vee c'_{IJ}$

e $Y'_I = \bigwedge_{L \in \nabla'_I} X'_{LI}$.

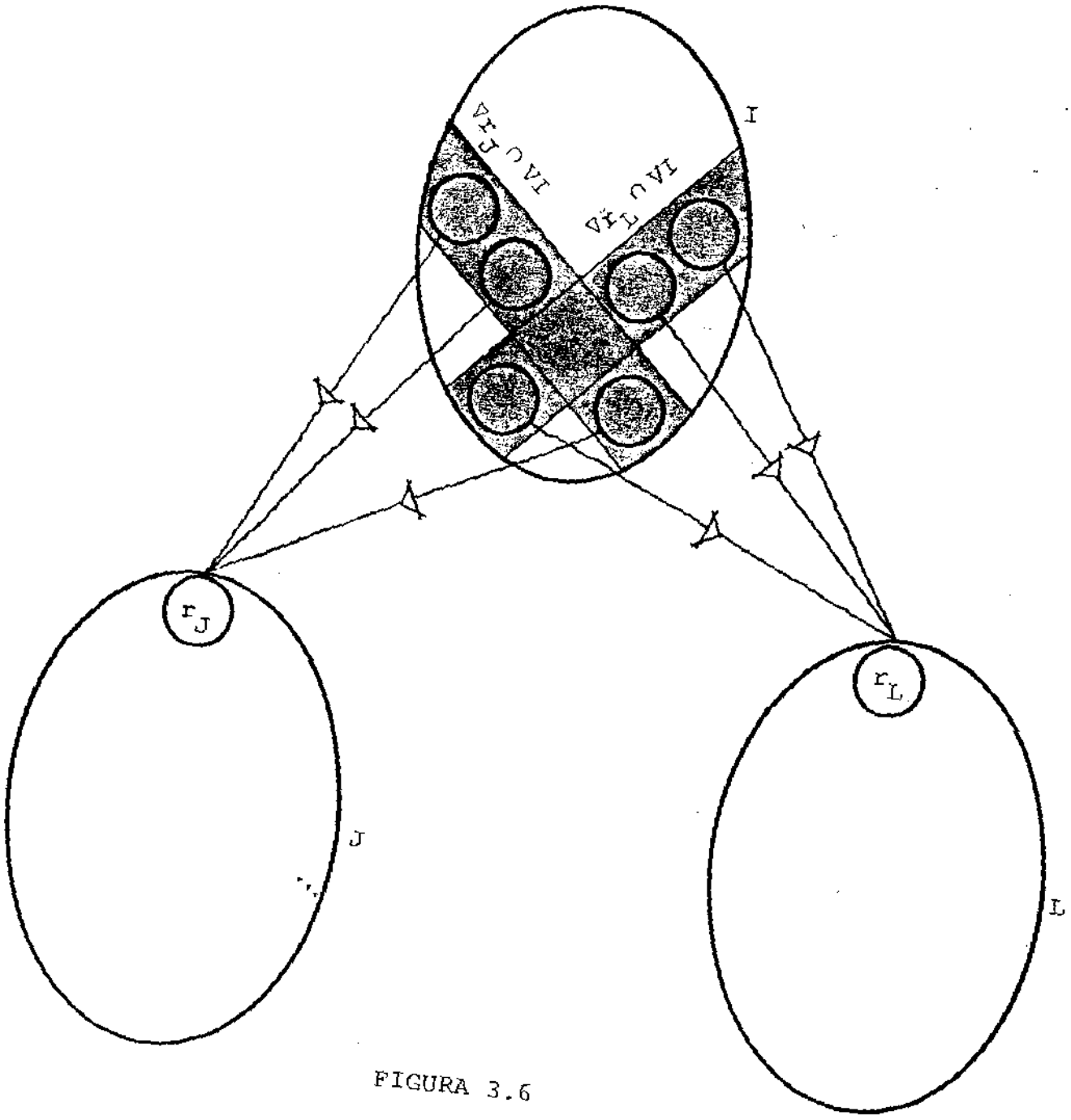


FIGURA 3.6

Dessa maneira obtivemos um sistema de equações semelhante ao original para o grafo derivado G', de forma que se G' não for trivial, podemos aplicar o processo recursivamente para encontrar os valores Y'_I para os vértices de G'.

A figura 3.7 apresenta um esquema de vizinhança entre dois intervalos I e J.

O algoritmo Grafo-Derivado, indicado na figura 3.8, é uma extensão do algoritmo Encontra-Intervalos que, além da partição dos vértices de G em subconjuntos que geram intervalos, fornece a estrutura de adjacência entre esses intervalos, a influência interna K_I de todos os vértices de cada intervalo I sobre a sua própria raiz, e o par (k',c') associado a cada aresta de G'.

A cada aresta (u,v) de G visitada, os valores de $\bar{k}_{uv}$ e de $\bar{c}_{uv}$ são calculados e acumulados nas variáveis κ_v e σ_v, que representam valores parciais das conjunções $\bigwedge_{u\in\nabla v} \bar{k}_{uv}$ e $\bigwedge_{u\in\nabla v} \bar{c}_{uv}$, respectivamente. Se v não é raiz de intervalo, κ_v e σ_v são usadas no cálculo de $\bar{k}$ e $\bar{c}$ das arestas de G com origem v. Se v é a raiz r_J do intervalo J, κ_v e σ_v serão reinicializados a cada visita a r_J através de uma aresta (u,r_J), onde u não pertence ao mesmo intervalo L que o predecessor de r_J anteriormente considerado. Nesse instante, κ_v e σ_v representam respectivamente as conjunções $\bigwedge_{w\in(\nabla r_J\cap VL)} \bar{k}_{wr_J}$ e $\bigwedge_{w\in(\nabla r_J\cap VL)} \bar{c}_{wr_J}$,

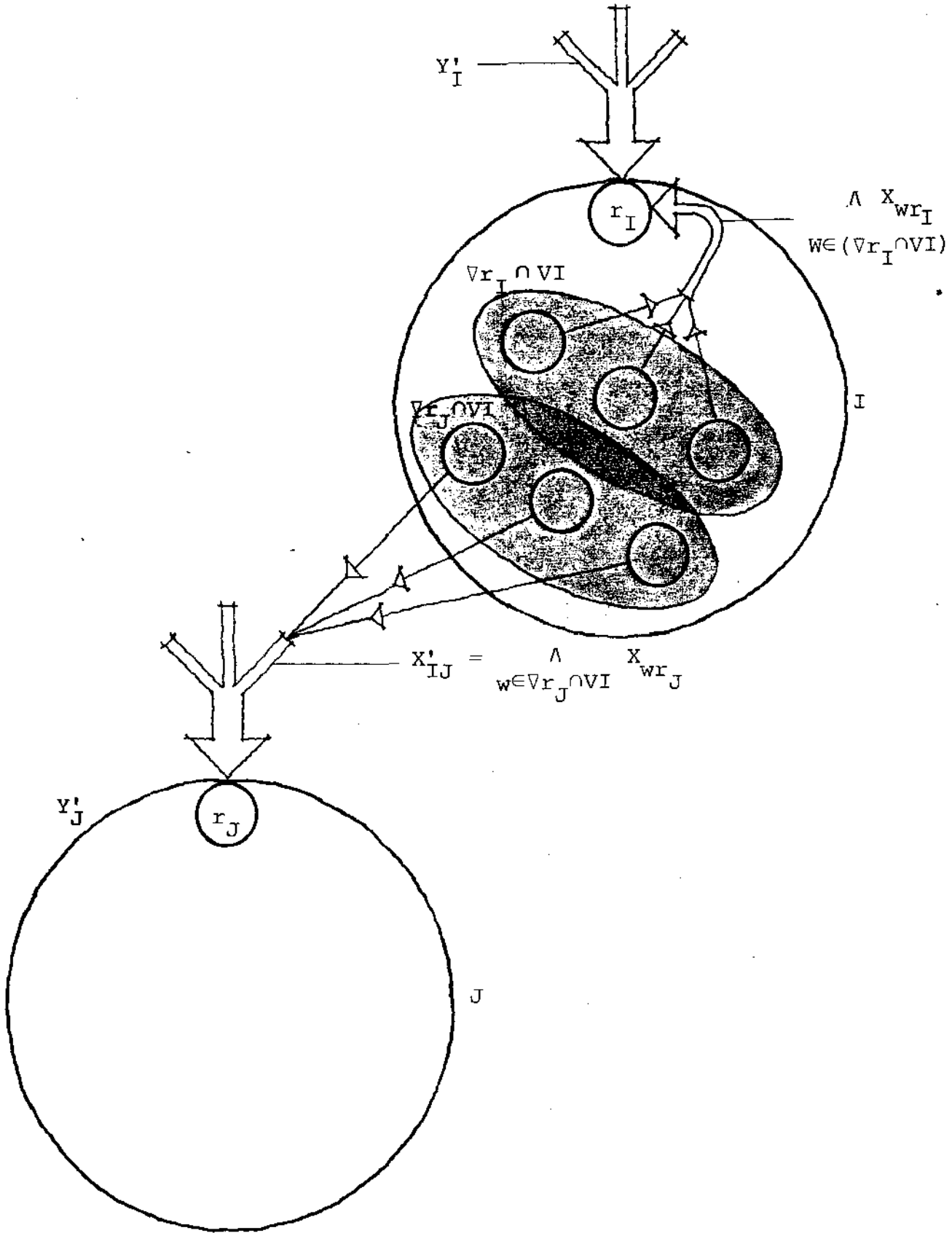

Y'_I
r_I
$\bigwedge_{w \in (\nabla r_I \cap VI)} X_{wr_I}$
$\nabla r_I \cap VI$
$\nabla r_J \cap VI$
I
$X'_{IJ} = \bigwedge_{w \in \nabla r_J \cap VI} X_{wr_J}$
Y'_J
r_J
J

FIGURA 3.7

que serão usadas no cálculo de k'_{LJ} e de c'_{LJ}, se $L \neq J$. Se $L = J$, o valor de κ_v será guardado na variável K_J para ser usado como fator no cálculo de k' para todos os sucessores de J em G'.

Uma aresta (L,J) será incluída em G' se o último predecessor de r_J considerado pertencia a VL, e r_J é visitado através de uma aresta cuja origem está num intervalo $I \neq L$. Uma aresta (I,J) será incluída em G' se r_J for visitado pela $|\nabla r_J|$-ésima (última) vez durante a identificação do intervalo I, se I for diferente do intervalo ao qual pertencia o primeiro predecessor de r_J considerado. (Antes da inclusão de uma nova aresta verifica-se sempre se ela não será um laço).

No primeiro caso, a identificação do intervalo L já foi concluída, de forma que K_L já foi determinado. Como o predecessor anterior de r_J pertencia ao intervalo L, κ_{r_J} contém o fator $\bigwedge_{w \in (\nabla r_J \cap VL)} \bar{k}_{wr_J}$ de k'_{LJ}, e σ_{r_J} contém c'_{LJ} (uma vez que κ_{r_J} e σ_{r_J} foram reinicializadas durante a identificação de L, e todas as arestas com origem em L e destino r_J foram visitadas até o final dessa identificação). Por conseguinte, os coeficientes k'_{LJ} e c'_{LJ} estão disponíveis, e a aresta (L, J) pode ser incluída em G'.

No outro caso (inclusão de uma aresta (I,J)), no entanto, é possível que nem todas as arestas com origem em I e destino

Algoritmo Grafo-Derivado (G)

```
1. R ← {n_o}
2. Para todo r em R execute os passos 3, 4, 5.
3. VI ← {r}                    /* I é o intervalo corrente        */
4. Para todo u em VI           /* Identificação do intervalo I */
     para todo v em Δu
   [se v é visitado pela primeira vez
   então [associa v com I
          R ← R ∪ {v}
          inicializa κ_v e σ_v com U
   calcula k̄_uv e c̄_uv
   se v associado com I
   então [acumula κ_v e σ_v
          se v é visitado pela |Vv|-ésima vez
          então se v ≠ 0
                então [VI ← VI ∪ {v}
                       R ← R \ {v}
                senão K[0] := κ[0]
   senão [se v é visitado pela primeira vez em I
          então [/* Seja N o intervalo associado com v originalmente */
                 se v é visitado pela primeira vez após identificação de N
                 então [atribui a v o próximo intervalo M
                        inicializa K_M com U
                 se intervalo L do predecessor anterior de v≠intervalo J de v
                 então inclui aresta (L,J) com coeficientes
                                k' = κ_v ∧ K_L e c' = σ_v em G'
                 senão K_J ← σ_v
                 reinicializa κ_v e σ_v com U
          acumula κ_v e σ_v
          se v é visitado pela |Vv|-ésima vez
          então se intervalo J de v ≠ I
                then inclui (J,κ_v,σ_v) na lista de sucessores
                                        pendentes de I
                senão K_J ← κ_v
5. Para todo J em sucessores pendentes de I
     inclui aresta (I,J) com coeficientes k' = κ_v ∧ K_I e c' = σ_v em G'
```

FIGURA 3.8.

r_I tenham sido visitadas, de forma que o valor de K_I não é, em geral, conhecido nesse momento. Portanto, J é incluído numa lista de sucessores pendentes de I, que será processada após a identificação do intervalo I, quando K_I será conhecido, e o coeficiente k'_{IJ} pode ser calculado. Dessa maneira, a linearidade do algoritmo é preservada.

O processo de passagem por uma fila de arestas pendentes poderia ser geral a todas as arestas de G', porém isso só diminuiria a eficiência do algoritmo sem simplificação significativa.

No procedimento correspondente, apresentado na figura 3.9, supõe-se que a estrutura de adjacência Δ de G é um vetor de listas que suporta os vetores característicos k, c, $\bar{k}$ e $\bar{c}$, correspondentes a cada aresta (u,v), que ainda serão denotados por k_{uv}, c_{uv}, $\bar{k}_{uv}$, e $\bar{c}_{uv}$, respectivamente. Supõe-se também que os k's e c's de cada aresta são fornecidos nessa estrutura. As variáveis K, κ, e σ serão implementadas como vetores de vetores característicos. O vetor Int-com-Raiz conterá um valor indefinido para os vértices de VG que não são raízes de intervalos, ou a identificação do intervalo do qual o vértice é raiz, caso contrário. O vetor Último-Assoc indica, para cada raiz r_J de algum intervalo J em G, qual o último vértice L de G' ao qual J foi associado como sucessor, e é usado para controlar a inclusão de arestas em G'. Se um vértice u de G não é raiz de intervalo, então Último-Assoc[u] será sempre indefinido. A estrutura

```
procedimento Grafo-Derivado (G = (VG,Δ))

1  para todo u em VG faça
2       Cont[u]          := 0
3       Assoc[u]         := ∞
4       Int-com-Raiz[u]  := ∞
5       Último-Assoc[u]  := ∞
6  Sucessores-Pendentes := lista vazia
7  Raízes               := lista vazia
8  Inclui-Vértice (Raízes, 0)
9  R[0] := verdadeiro
10 Assoc[0] := 0
11 Int-com-Raiz[0] := 0
12 Último-Assoc [0] := 0
13 κ[0] := U
14 σ[0] := U
15 K[0] := U
16 I    := 0 /* I é o intervalo corrente */
17 M    := 0 /* M é o último número usado p/ identificar um intervalo */
18 para todo u em VG faça
19      para todo v em Δ[u] faça
20           Cont[v] := Cont[v]+1 /* conta predecessores de um vértice */
21 repita
22      Retira-Vértice (Raízes, r)
23      se r ∈ R
24      então Intervalos [I] := lista vazia
25            Δ'[I]          := lista vazia
26            Inclui-Vértice (Intervalos [I], r)
27            para todo u em Intervalos [I] faça
28                 para toda aresta a em Δ[u] faça
29                      v := a.destino
30                      Trata-Aresta
31            enquanto não Vazio (Sucessores-Pendentes) faça
32              Retira-Aresta (Sucessores-Pendentes, (J, kappa, sigma))
33              Inclui-Aresta (Δ'[I], (J, kappa ∧ K[I], sigma))
34            I := I+1
35 até Vazio (Raízes)
36 devolva (Intervalos, K, (M,Δ'))
```

FIGURA 3.9a.

```
     procedimento Trata-Aresta
37   se Assoc [v] =: ∞
38   então  Assoc [v] := I
39          Inclui-Vértice (Raízes, v)
40          R[v]       := verdadeiro
41          κ[v]       := U
42          σ[v]       := U
43   Cont [v] := Cont [v]-1
44   se  u = r
45   então  k̄_uv := k_uv
46          c̄_uv := c_uv
47   senão  k̄_uv := (κ[u] ∧ k_uv) ∨ c_uv
48          c̄_uv := (σ[u] ∧ k_uv) ∨ c_uv
49   se Assoc [v] = I
50   então  κ[v] := κ[v] ∧ k̄_uv
51          σ[v] := σ[v] ∧ c̄_uv
52          se Cont [v] = 0
53          então se  v ≠ 0
54                then  Inclui-Vértice (Intervalos [I] , v)
55                      R[v]  := falso
56                senão  K[0]  := κ[0]
57   senão  se Último-Assoc [v] ≠ I
58          então  se Int-com-Raiz [v] = ∞
59                 então  M                  := M + 1
60                        Int-com-Raiz [v]  := M
61                        Último-Assoc [v]  := Assoc [v]
62                        K[M]              := U
63                 se Último-Assoc [v] ≠ Int-com-Raiz [v]
64                 então Inclui-Aresta (Δ'[Último-Assoc [v]],
                                        (Int-com-Raiz [v] ,
                                         κ[v] ∧ K[Último-Assoc[v]],σ[v]))
65                 senão K[Int-com-Raiz [v]] := κ[v]
66                 Último-Assoc [v]  := I
67                 κ[v] := U
68                 σ[v] := U
69          κ[v] := κ[v] ∧ k̄_uv
70          σ[v] := σ[v] ∧ c̄_uv
71          se Cont [v] = 0
72          então se  v ≠ r
73                then Inclui-Aresta(Sucessores-Pendentes,(Int-com-Raiz [v] ,
                                                            κ[v] , σ[v] ))
74                     senão  K[Int-com-Raiz [v]] := κ[v]
```

FIGURA 3.9b.

devolvida Δ' é análoga a Δ, e ao final da execução do procedimento conterá a estrutura de adjacência de G', incluindo os valores de k' e c' para cada aresta.

A existência de dois procedimentos para incluir elementos em filas (Inclui-Vértice e Inclui-Aresta), e de outros dois para recuperar o próximo elemento (Retira-Vértice e Retira-Aresta), justifica-se pelas diferentes estruturas das células que compõem as listas e das informações manipuladas.

O procedimento Trata-Aresta foi criado apenas para facilitar a visão geral do procedimento Grafo-Derivado, e pode ser expandido no único ponto de ativação no corpo desse último.

3.4. ALGORITMO FINAL

O algoritmo de análise por intervalos sugerido inicialmente por Cocke e Allen (Algoritmo CA), é apresentado na figura 3.10. É fácil ver que esse algoritmo termina, pela verificação de que o tamanho de um grafo derivado é estritamente menor que o tamanho do seu grafo original, a menos que este seja o grafo trivial.

A figura 3.11 apresenta o procedimento correspondente ao algoritmo CA. Note-se que ele utiliza os valores $\kappa[v]$ e $\sigma[v]$ deixados como efeito lateral pelo procedimento Grafo-Derivado. Naquele ponto, $\kappa[v]$ e $\sigma[v]$ representam respectivamente as conjunções de $\overline{k}_{uv}$ e de $\overline{c}_{uv}$, para todos os $u \in \nabla v$, se v não é raiz de intervalo em G.

Algoritmo CA (G)

1. Se G é trivial então $Y_{n_o} \leftarrow \phi$

2. Caso contrário

   - determine o grafo G' derivado de G
   - use o algoritmo recursivamente para determinar Y'_I para os vértices I de G'
   - para todo I em VG'
     - calcule Y_{r_I} (r_I é a raiz do intervalo I do grafo G)
     - visite os vértices $v \neq r_I$ que pertencem ao intervalo I em G, na ordem em que foram colocados em I, determinando Y_v em função de Y_{r_I}

FIGURA 3.10.

```
procedimento  CA(G = (VG,Δ))

1   se  |VG| -1 = 0 ∧ Δ[0] = lista vazia
2   então devolva ϕ
3   senão (Intervalos, K,G' = (VG',Δ')) := Grafo-Derivado (G)
4         Y' := CA(G')
5         para todo I em VG' faça
6             Retira-Vértice (Intervalos [I], r)
                                  /*  r é raiz de  I */
7             z    := Y'[I] ∧ K[I]
8             Y[r] := z
9             para todo v em Intervalos [I] faça
10                Y[v] := z ∧ ⊀[v] ∨ σ[v]
11        devolva Y
```

FIGURA 3.11.

As tabelas da figura 3.12 apresentam o número de operações básicas executadas por linha (ou grupo de linhas), a cada ativação do procedimento CA, Grafo-Derivado e Trata-Aresta.

No procedimento CA consideramos que VG é dado por uma variável cujo valor é |VG|-1, de forma que o teste |VG| -1 = 0 exige apenas um acesso àquela variável e a comparação propriamente dita. As atribuições das linhas 3 e 4 constam apenas para designar as variáveis que conterão os valores retornados pelos procedimentos, logo serão contados apenas os custos de ativação e execução destes. Considerando o caso de uma entrada por ativação, a malha da linha 5 é executada $N_{v'}$ vezes, e o teste de fim de iteração é efetuado depois do corpo porque o grafo derivado tem pelo menos um vértice. Por causa do tratamento especial dado às raízes de intervalo, no total das $N_{v'}$ entradas na malha da linha 9, acontecem $(N_v - N_{v'})$ iterações, e como é possível que um intervalo contenha apenas a raiz, o teste de fim de iteração é feito antes de executar o corpo. Além das operações usuais de controle da malha, ocorrem ainda $N_{v'}$ subscrições, para localizar as cabeças de lista nas atribuições iniciais, e $(N_v - N_{v'})$ seleções, uma para cada vértice nas listas.

Se o argumento de chamada ao procedimento CA for o grafo trivial, o seu custo se resume às linhas 1 e 2, caso contrário a linha 2 não será executada.

No procedimento Grafo-Derivado, os corpos das malhas nas

linhas 1 e 18 são executados N_v vezes cada, sendo o teste de fim de iteração efetuado após cada execução. Nas N_v entradas na malha da linha 19, ocorrem N_a iterações, com o teste antes de cada iteração, uma vez que é possível haver vértices sem sucessores. Ocorrem ainda uma subscrição a cada entrada na malha, e uma seleção da identificação do vértice v na lista a cada iteração.

É fácil ver que a malha da linha 21 é executada N_v vezes, pois todo vértice $v \neq 0$ de G é argumento da chamada de procedimento da linha 39 exatamente uma vez, enquanto que a chamada na linha 22 se encarrega de eliminá-los. O teste da linha 23 resultará verdadeiro $N_{v'}$ vezes, porque R[v] se torna verdadeiro (linha 40) sempre que o vértice v é incluído na lista Raízes, porém é tornado falso pelo comando na linha 55 sempre que v é incluído num intervalo cuja raiz não é v, e isso tudo ocorre sempre numa única entrada na malha da linha 27. Considerando todas as $N_{v'}$ entradas na malha da linha 27, seu corpo é executado N_v vezes, com teste de fim de iteração após cada execução, uma vez que todo intervalo contém pelo menos a raiz. Além das operações de controle, são necessárias mais uma subscrição para cada atribuição inicial, e uma seleção do vértice u da lista de vértices do intervalo a cada iteração. No total das N_v entradas, ocorrem N_a iterações na malha da linha 28, com teste de saída anterior a cada iteração, e uma subscrição a cada entrada na malha. Finalmente, no pior caso, teremos $N_{v'}$ execuções do corpo da malha na linha 31. Isso porque cada vértice J do grafo

derivado só pode ser destino de uma aresta na lista de sucessores pendentes no máximo uma vez (pois a inclusão só ocorre eventualmente no momento em que o último predecessor da raiz de J em G é visitado (linha 73)). Como ocorrem $N_{v'}$ entradas nessa malha e o teste de controle (Sucessores-Pendentes vazia) é executado antes de cada iteração, teremos no máximo $2N_{v'}(\mu_{ac} + \mu_c + \mu_d)$ para controlar esse *enquanto*. Na linha 32 foi considerado o pior caso, de $N_{v'}$ execuções da chamada de procedimento. A instrução da linha 33, no entanto, foi contada conjuntamente com a da linha 64, totalizando o número exato de inclusões na estrutura Δ'.

No procedimento Trata-Aresta, das N_a vezes em que a condição na linha 37 é testada, ela é verdadeira uma vez para cada vértice de G, exceto a origem, de forma que os comandos associado ao *então* são executados $(N_v - 1)$ vezes, e ocorrem $(N_a - N_v + 1)\mu_d$.

Claramente, o resultado do teste da linha 44 é verdadeiro N_9 vezes, e falso $(N_a - N_9)$. As instruções das linhas 50, 51, 52 conjuntamente com aquelas das linhas 69, 70, 71, são executadas no total N_a vezes. O teste da linha 52 dá verdadeiro uma vez para cada vértice do grafo G que não é raiz de região, exceto possivelmente o vértice inicial, logo a comparação da linha 53 é executada no máximo $(N_v - N_{v'} + 1)$ vezes, com exatamente $N_v - N_{v'}$ resultados verdadeiros, e talvez 1 falso. O teste da linha 49 resulta falso para todos os predecessores de toda raiz r, que não estão no intervalo ao qual pertence o vértice u, origem da

primeira aresta incidente a r considerada na execução do procedimento Grafo-Derivado. Em outras palavras, para cada raiz r de intervalo em G, seja (u,r) a primeira aresta com destino r considerada na execução do procedimento Grafo-Derivado, e seja N o intervalo de G tal que $u \in VN$. O teste da linha 49 dá falso para toda aresta $(v,r) \in aG$ tal que $v \notin VN$. O número dessas arestas é dado pela soma $(N_{10} - N_{11}) + N_{13}$, que é o número de vezes que a comparação da linha 57 é executada. Note-se que essa quantidade não é função apenas da topologia do grafo, mas também da maneira como este foi representado. Ela só independe da representação quando todos os predecessores de um vértice J no grafo derivado contêm a mesma quantidade de predecessores da raiz r_J do intervalo J em G, ou quando algum desses predecessores domina r_J.

A comparação na linha 57 dá verdadeiro para uma dada raiz de intervalo v, sempre que (u,v) é a primeira aresta considerada cuja origem está num intervalo I qualquer, exceto a primeira aresta com destino v considerada durante a execução do procedimento. Esta situação ocorre $(N_{a'} - N_{v'} + N_{14} + 1)$ vezes, com um erro de 1, caso exista um vértice v tal que $(v,0) \in aG$ e v está no intervalo I(0). O teste da linha 58 dá resultado verdadeiro $(N_{v'} - 1)$ vezes, e o teste da linha 63 dá verdadeiro $(N_{a'} - N_{v'})$ vezes, considerando o pior caso de arestas não entrarem direto em Δ', mas passarem antes por Sucessores-Pendentes. As subscrições na linha 64 que não foram computadas junto com a

linha 33, serão contadas em conjunto com as subscrições da linha 73 ($3N_a$,, referentes aos vetores Int-com-Raiz, κ, σ), ou sozinhas, considerando o pior caso ($2(N_a, - N_v,)$, referentes às duas subscrições no vetor Último-Assoc). As linhas 65 e 74 foram agregadas para se considerar o pior caso, em que todas as raízes de intervalo r têm um predecessor em I(r). O teste da linha 71 dá verdadeiro uma vez para cada raiz de intervalo, podendo ocorrer um erro de 1 se $V0 \subseteq VI$, com $I = I(0)$. Considerando o pior caso, haverá N_v, chamadas ao procedimento Inclui-Aresta na linha 73.

Substituindo-se os custo de μ_{av}, $\mu_{Inclui\text{-}Vértice}$, $\mu_{Retira\text{-}Vértice}$, $\mu_{Inclui\text{-}Aresta}$ e $\mu_{Retira\text{-}Aresta}$, pelos custos de implementações típicas, como as apresentadas no apêndice 3, pode-se obter equações que expressam o número de vezes que cada operação básica é executada e cada ativação do procedimento CA. A figura 3.13 apresenta as equações que indicam o número de operações executadas numa ativação do procedimento CA, incluindo os procedimentos auxiliares e as chamadas recursivas. As equações obtidas têm a forma de somatórias nas quais o índice i indica parâmetros correspondentes aos grafos da sequência $G_o, G_1, \ldots, G_{m-1}$, onde G_o é o grafo original. Para todo $1 \leq i \leq m-1$, G_i é o grafo derivado de G_{i-1}, e o grafo G_m, derivado de G_{m-1}, é o grafo trivial, com $G_{m-1} \neq G_m$. O custo de CA para um grafo trivial é $(2\mu_{ac} + 3\mu_c + \mu_{ol} + \mu_s)$.

CUSTO DO PROCEDIMENTO CA

1.	1	$(2\mu_{ac} + 3\mu_c + \mu_d + \mu_{ol} + \mu_s)$
2.	1	μ_{rp}
3+4.	1	$(2\mu_{cpl} + \mu_{Grafo\text{-}Derivado}(G) + \mu_{CA}(G'))$
5.	1	$(\mu_{ac} + \mu_{at})$
	$N_{v'}$	$(2\mu_{ac} + \mu_{ar} + \mu_{at} + \mu_c)$
	$(N_{v'}-1)$	μ_d
6+..+8.	$N_{v'}$	$(3\mu_{AC} + 2\mu_{AT} + 4\mu_s + \mu_{ol} + \mu_{Retira\text{-}Vértice})$
9.	$N_{v'}$	$(\mu_{ac} + \mu_{at} + \mu_s)$
	N_v	$(\mu_{ac} + \mu_c + \mu_d)$
	$(N_v-N_{v'})$	$(\mu_{av} + \mu_{sel})$
10.	$(N_v-N_{v'})$	$(3\mu_{AC} + \mu_{AT} + 2\mu_{OL} + 3\mu_s)$
11.	1	μ_{rp}

FIGURA 3.12a

CUSTO DO PROCEDIMENTO GRAFO DERIVADO

1.	N_v+1	$(\mu_{at} + \mu_{ac})$
	N_v	$(\mu_{ar} + \mu_c + \mu_{ac})$
	N_v-1	μ_d
2+..+5.	$4N_v$	$(\mu_s + \mu_{at})$
6+7.	2	μ_{at}
8.	1	$\mu_{Inclui\text{-}Vértice}$
9+..+17.	1	$7\mu_s + 6\mu_{at} + 3\mu_{AT}$
18.	N_v+1	$(\mu_{at} + \mu_{ac})$
	N_v	$(\mu_{ar} + \mu_c + \mu_{ac})$
	N_v-1	μ_d
19.	(N_a+N_v)	$(\mu_d + \mu_c + \mu_{ac})$
	N_v	$(\mu_{ac} + \mu_{at} + \mu_s)$
	N_a	$(\mu_{av} + \mu_{sel})$
20.	N_a	$(2\mu_s + \mu_{at} + \mu_{ar} + \mu_{ac})$

FIGURA 3.12b

21.	N_v	$(\mu_{ac} + \mu_c)$
	(N_v-1)	μ_d
22+23.	N_v	$(\mu_{ac} + \mu_{Retira\text{-}Vértice} + \mu_s + \mu_c)$
	$(N_v-N_{v'})$	μ_d
24+25.	$N_{v'}$	$2(\mu_s + \mu_{at})$
26.	$N_{v'}$	$(\mu_s + \mu_{Inclui\text{-}Vértice})$
27.	$N_{v'}$	$(\mu_s + \mu_{at} + \mu_{ac})$
	N_v	$(\mu_{sel} + \mu_{av} + \mu_c + \mu_{ac})$
	$(N_v-N_{v'})$	μ_d
28.	N_v	$(\mu_s + \mu_{at} + \mu_{ac})$
	N_v+N_a	$(\mu_c + \mu_d + \mu_{ac})$
	N_a	μ_{av}
29.	N_a	$(\mu_{ac} + \mu_{sel} + \mu_{at})$

FIGURA 3.12c

31.	$2N_v$,	$(\mu_{ac} + \mu_c + \mu_d)$	(PIOR CASO)
32.	N_v,	$\mu_{Retira\text{-}Vértice}$	(PIOR CASO)
33+64.	N_a,	$(2\mu_s + \mu_{OL} + 2\mu_{AC} + \mu_{Inclui\text{-}Aresta})$	
34.	N_v,	$(\mu_{ar} + \mu_{at} + \mu_{ac})$	
36.	1	μ_{rp}	
37.	N_a	$(\mu_{ac} + \mu_c + \mu_s)$	
	(N_a-N_v+1)	μ_d	
38+..+42.	(N_v-1)	$(\mu_{ac} + 2\mu_{at} + 2\mu_{AT} + 4\mu_s +$ Inclui-Vértice$)$	
43+44.	N_a	$(3\mu_{ac} + \mu_{ar} + \mu_{at} + \mu_c + \mu_d + 2\mu_s)$	
45+46.	N_9	$(2\mu_{AC} + 2\mu_{AT} + 4\mu_{sel})$	
47+48.	$(N_a - N_9)$	$(6\mu_{AC} + 2\mu_{AT} + 4\mu_{OL} + 2\mu_s + 6\mu_{sel})$	
49.	N_a	$(2\mu_{ac} + \mu_c + \mu_d + \mu_s)$	
50..52+69..71.	N_a	$(4\mu_{AC} + 2\mu_{AT} + \mu_c + \mu_{ac} + 5\mu_s + 2\mu_{sel} + 2\mu_{OL})$	
	(N_a-N_v+1)	μ_d	

FIGURA 3.12d

53.	$(N_v - N_{v'} + 1)$	$(\mu_{ac} + \mu_c + \mu_d)$
54+55.	$(N_v - N_{v'})$	$(\mu_{at} + 2\mu_s + \mu_{Inclui\text{-}Vértice})$
56.	1	$(\mu_{AC} + \mu_{AT} + 2\mu_s)$
57.	$(N_{10} - N_{11} + N_{13})$	$(2\mu_{ac} + \mu_c + \mu_s)$
	$(N_{10} - N_{11} + N_{13} - N_{a'} + N_{v'} - N_{14} - 1)$	μ_d
58+63+66+67+68.	$(N_{a'} - N_{v'} + N_{14} + 1)$	$(4\mu_{ac} + \mu_{at} + 2\mu_{AT} + 2\mu_c + 6\mu_s)$
	$2(N_{a'} - N_{v'} + N_{14} + 1) - (N_{v'} - 1)$	μ_d
59+..+62.	$(N_{v'} - 1)$	$(3\mu_{ac} + \mu_{ar} + 3\mu_{at} + \mu_{AT} + 4\mu_s)$
64.	$(N_{a'} - N_{v'})$	$2\mu_s$
64+73.	$N_{a'}$	$3\mu_s$
65+74.	$N_{v'}$	$(\mu_{AC} + \mu_{AT} + 3\mu_s)$
72.	$N_{v'}$	$(2\mu_{ac} + \mu_c + \mu_d)$
73.	$N_{v'}$	$\mu_{Inclui\text{-}Aresta}$

FIGURA 3.12e

SOMA VERTICAL DE OPERAÇÕES EM TODAS AS EXECUÇÕES DE CA

$$\sum_{i=0}^{m-1} (26N_{a_i} + 55N_{v_i} + 2N_{10_i} - 2N_{11_i} + 2N_{13_i} + 4N_{14_i}) - 13N_{a_o} - 19N_{v_o} + 6m + 21 \qquad \mu_{ac}$$

$$2\sum_{i=0}^{m-1} (7N_{a_i} + 4N_{v_i} - 2N_{9_i}) - 4N_{a_o} - 5N_{v_o} + m + 5 \qquad \mu_{AC}$$

$$\sum_{i=0}^{m-1} (4N_{a_i} + 11N_{v_i}) - 2N_{a_o} - 4N_{v_o} - m + 4 \qquad \mu_{ar}$$

$$\sum_{i=0}^{m-1} (11N_{a_i} + 40N_{v_i} + N_{14_i}) - 6N_{a_o} - 15N_{v_o} + 7m + 15 \qquad \mu_{at}$$

$$2\sum_{i=0}^{m-1} (4N_{a_i} + 4N_{v_i} + N_{14_i}) - 4N_{a_o} - 5N_{v_o} + 3m + 5 \qquad \mu_{AT}$$

$$\sum_{i=0}^{m-1} (9N_{a_i} + 13N_{v_i} + N_{10_i} - N_{11_i} + N_{13_i} + 2N_{14_i}) - 3N_{a_o} - 2N_{v_o} + 6m + 5 \qquad \mu_c$$

$$2m \qquad cpl$$

$$\sum_{i=0}^{m-1} (8N_{a_i} + 9N_{v_i} + N_{10_i} - N_{11_i} + N_{13_i} + N_{14_i}) - 2N_{a_o} + 2m \qquad \mu_d$$

$$m + 1 \qquad \mu_{ol}$$

$$\sum_{i=0}^{m-1} (7N_{a_i} + N_{v_i} - 4N_{9_i}) - N_{a_o} + N_{v_o} - 1 \qquad \mu_{OL}$$

$$2m \qquad \mu_{rp}$$

$$\sum_{i=0}^{m-1} (26N_{a_i} + 19N_{v_i} - 2N_{9_i} + N_{10_i} + N_{11_i} + N_{13_i} + 6N_{14_i}) - 13N_{a_o} - 3N_{v_o} + 8m + 4 \qquad \mu_s$$

$$2\sum_{i=0}^{m-1} (8N_{a_i} + 9N_{v_i} - N_{9_i}) - 4N_{a_o} - 8N_{v_o} + 8 \qquad \mu_{sel}$$

FIGURA 3.13

CAPÍTULO IV

O ALGORITMO DE GRAHAM E WEGMAN

4.1. INTRODUÇÃO

Como na abordagem por intervalos, o método de Graham e Wegman só é aplicável a grafos redutíveis, e associa as funções k e c às arestas do grafo.

A idéia básica é processar uma série de transformações nas arestas do grafo, até a obtenção de uma estrela. A solução do sistema de equações para uma estrela é trivial. As transformações são realizadas da seguinte maneira: escolhe-se um vértice $v \neq n_o$ que tenha um único predecessor u diferente dele próprio, elimina-se um eventual laço em v, e em seguida elimina-se cada aresta (v,w) do grafo, substituindo-a por uma aresta (u,w) se esta ainda não existir. (Essas transformações serão descritas mais rigorosamente na seção seguinte). Os valores k e c associados a cada uma das arestas (v,w) eliminadas serão "incluídos" nos valores da aresta (u,w) correspondente no novo grafo.

Verifica-se que, dado um grafo redutível G, tem-se (apresentaremos apenas a verificação de b e d):

a) ou G é uma estrela ou existe em G um vértice nas condições de v acima;

b) o grafo resultante após as transformações citadas também é redutível;

c) uma seqüência suficientemente grande dessas transformações sempre converge para uma estrela; e

d) a solução maximal para as variáveis Y do sistema de equações para o novo grafo é a mesma do grafo original G.

Apesar de convergir para uma estrela, uma seqüência qualquer de transformações pode tomar tempo quadrático. Para reduzir esse tempo, o algoritmo determina um subgrafo de G, semelhante a um intervalo, chamado de região, dentro do qual é imediato identificar um vértice v com a propriedade mencionada. Após aplicar uma transformação a cada aresta com origem v dentro da região, um novo vértice da região será tomado, até que todos já o tenham sido. A repetição desse processo transformará G num grafo acíclico, e ele será facilmente reduzido a uma estrela.

4.2. TRANSFORMAÇÕES NO GRAFO

DEFINIÇÕES

Seja $G = (VG, aG, n_o)$ um grafo. Diremos que uma aresta(u,v) satisfaz a condição PDU (predecessor distinto único) se $(u,v) \in aG$ e $\nabla v \setminus \{v\} = \{u\}$ (isto é, (u,v) é a única aresta incidente a v que não é um laço).

Se (u,v) satisfaz a condição PDU e $\alpha = (v,v) \in aG$, então $T1(G,\alpha) = (VG,\ aG \setminus \{\alpha\},\ n_o)$.

Se (u,v) satisfaz a condição PDU com $v \neq n_o$, e $\alpha = (v,w) \in aG$, com $v \neq w$, então $T2(G,\alpha) = (VG,(aG \cup \{(u,w)\}) \setminus \{\alpha\},\ n_o)$.

A figura 4.1 ilustra estas transformações. Note-se que, no caso de T2, as arestas (u,w) e (v,v) podem não existir no grafo original, e que não está excluído o caso $u = w$.

PROPRIEDADE 4.2.1.

Se $(u,v) \in aG$ satisfaz a condição PDU, a transformação $T1(G,(v,v))$ ou $T2(G,(v,w))$ não cria dominância de um vértice sobre qualquer outro vértice de G.

PROVA.

A prova é simples, notando-se que a transformação T1 elimina apenas laços, e esses não podem fazer parte de caminhos, e que todo caminho de n_o a um vértice z que não passava pela aresta (v,w) ainda permanece no grafo após uma transformação T2, e aqueles que passavam pela aresta (v,w) também passavam por (u,v), de forma que a seção (u,v,w) pode ser substituída por (u,w). Assim, se havia um caminho de n_o a z que não passava por um determinado vértice em G, existe um caminho (possivelmente o mesmo) nessas mesmas condições no grafo resultante.

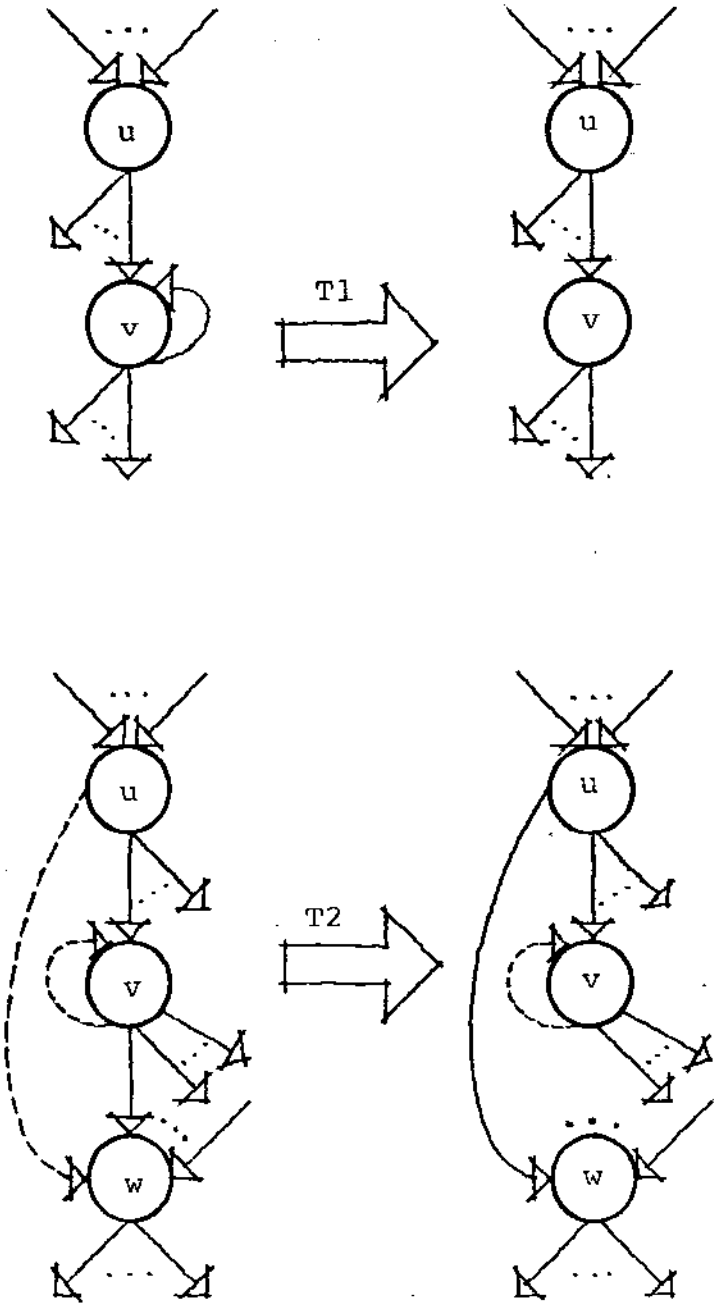


FIGURA 4.1

PROPRIEDADE 4.2.2.

Dado um grafo $G = (VG, aG, n_o)$, tem-se:

ou I) T1 ou T2 é aplicável a uma aresta de G,

ou II) G é uma estrela,

ou III) G é irredutível.

Por ser relativamente extensa, a prova será omitida.

PROPRIEDADE 4.2.3.

As transformações T1 e T2 preservam a redutibilidade de um grafo.

Claramente a transformação T1 não origina um grafo não redutível. A prova de que a transformação T2 também preserva a redutibilidade pode ser encontrada em GRAHAM [1976]. Apesar da transformação considerada na referência eliminar o vértice v e a aresta (u,v) se (v,w) era a única aresta com origem em v no grafo original, verifica-se facilmente que a manutenção de (u,v) e de v não interfere na redutibilidade do grafo resultante.

PROPRIEDADE 4.2.4.

Uma sequência suficientemente grande de transformações T1 e T2 num grafo redutível sempre converge para uma estrela.

A prova dessa propriedade também será omitida.

4.3. TRANSFORMAÇÃO DAS FUNÇÕES k E c

Como foi dito a princípio, os valores k e c associados à aresta eliminada a cada transformação T1 ou T2 são de certa forma acumulados numa aresta que tem o mesmo destino da aresta eliminada. Se uma aresta (u,v) satisfaz a condição PDU em G, e a transformação $T_i(G,(v,w))$ é executada, os valores de k e c propostos para a aresta (u,w) no grafo resultante G', irão representar a ação da aresta (u,w) se esta já existia em G, e ao mesmo tempo representar a ação da sequência (u,v), (v,w), mantendo a solução maximal para as variáveis Y do sistema para G'. (Utilizaremos apóstrofo para designar grandezas associadas ao grafo resultante após uma transformação T1 ou T2).

Consideremos a transformação T1 como indicada na figura 4.1. Inicialmente temos, devido à condição PDU:

$$Y_v = X_{uv} \wedge X_{vv} = (k_{uv} \wedge Y_u \vee c_{uv}) \wedge (k_{vv} \wedge Y_v \vee c_{vv})$$

A solução maximal dessa equação é dada por:

$$Y_v = (k_{uv} \wedge Y_u \vee c_{uv}) \wedge (k_{vv} \vee c_{vv}) = (k_{uv} \wedge Y_u \vee c_{uv}) \wedge k_{vv} .$$

Como u será o único predecessor de v no grafo resultante G', a solução maximal será mantida se substituirmos a equação

original para Y_v por:

$$Y_v = Y'_v = X'_{uv} = k'_{uv}\ Y_u \vee c'_{uv}$$

onde $k'_{uv} = k_{uv} \wedge k_{vv}$ e $c'_{uv} = c_{uv} \wedge k_{vv}$.

(Note-se que $k'_{uv} \supseteq c'_{uv}$.)

Para tratar o caso da transformação T2, notemos em primeiro lugar que a solução maximal do sistema de equações associado a um grafo G não se altera se introduzirmos arestas com $k = U$ e $c = U$. Assim podemos considerar o caso geral de T2, com a presença das arestas (u,w) e (v,v) no grafo original.

Temos $Y_w = X_{uw} \wedge X_{vw} \wedge \bigwedge_{z \in (\nabla w \setminus \{u,v\})} X_{zw}$

e $Y'_w = X'_{uw} \wedge \bigwedge_{z \in (\nabla' w \setminus \{u\})} X'_{zw}$.

Tomemos $X'_{uw} = X_{uw} \wedge X_{vw}$.

$X_{uw} = k_{uw} \wedge Y_u \vee c_{uw}$ e $X_{vw} = k_{vw} \wedge Y_v \vee c_{vw}$.

Usando a solução maximal obtida anteriormente para Y_v, na presença do laço (v,v), teremos:

$$X_{vw} = k_{vw} \wedge [(k_{uv} \wedge k_{vv}) \wedge Y_u \vee (c_{uv} \wedge k_{vv})] \vee c_{vw}$$

Logo, $X'_{uw} = k'_{uw} \wedge Y_u \vee c'_{uw}$

onde $k'_{uw} = k_{uw} \wedge k_{uv} \wedge k_{vv} \wedge k_{vw} \vee k_{uw} \wedge c_{vw}$

e $c'_{uw} = c_{uw} \wedge c_{uv} \wedge k_{vv} \wedge k_{vw} \vee c_{uw} \wedge c_{vw}$

(Mais uma vez, note-se que $k'_{uw} \geq c'_{uw}$.)

É fácil ver que preservando-se os pares (k,c) associados a cada aresta diferente de (u,w) no grafo $G' = T2(G,(v,w))$, a solução maximal do sistema é mantida. Em particular,

$$\bigwedge_{z \in (\nabla w \setminus \{u,v\})} X_{zw} = \bigwedge_{z \in (\nabla' w \setminus \{u\})} X'_{zw} .$$

4.4. REGIÕES

DEFINIÇÃO

Uma *super-região* R de um grafo $G = (VG, aG, n_o)$ é o subgrafo maximal gerado por um conjunto $VR \subseteq VG$ tal que:

a) $|VR| \geq 2$,

b) R é fortemente conexo,

c) todos os vértices de R são dominados por um vértice $r \in VR$, dito *raiz* da super-região (r será adotado como vértice inicial do subgrafo R).

Verificam-se facilmente as seguintes propriedades:

(1) O vértice r é único para uma super-região R, e se r é raiz de duas super-regiões R_1 e R_2, então $R_1 = R_2$. Portanto adotaremos a notação R(r).

(2) Toda super-região R contém pelo menos um circuito.

(3) Se R(r) é uma super-região, então todo circuito C em G ou está totalmente contido em R, ou se passa por $v \in VR$ também passa por r (isto é, se $VC \cap VR \neq \phi$, então $VC \subseteq VR$ ou $VC \cap VR \supseteq \{r\}$).

DEFINIÇÃO

Uma super-região R(r) é uma *região* se R é uma super-região minimal (isto é, se R não contém outra super-região).

A figura 4.2 apresenta um grafo com suas super-regiões, das quais apenas aquelas com raízes 3, 9, 12 e 14 são regiões. Note-se que os vértices 2, 6, 16 e 19 não estão em nenhuma super-região do grafo.

PROPRIEDADE 4.4.1.

Seja R(r) uma super-região do grafo redutível G. R é uma região sse todo circuito C de R passa por r (ou seja, se $VC \subseteq VR$, então $r \in VC$).

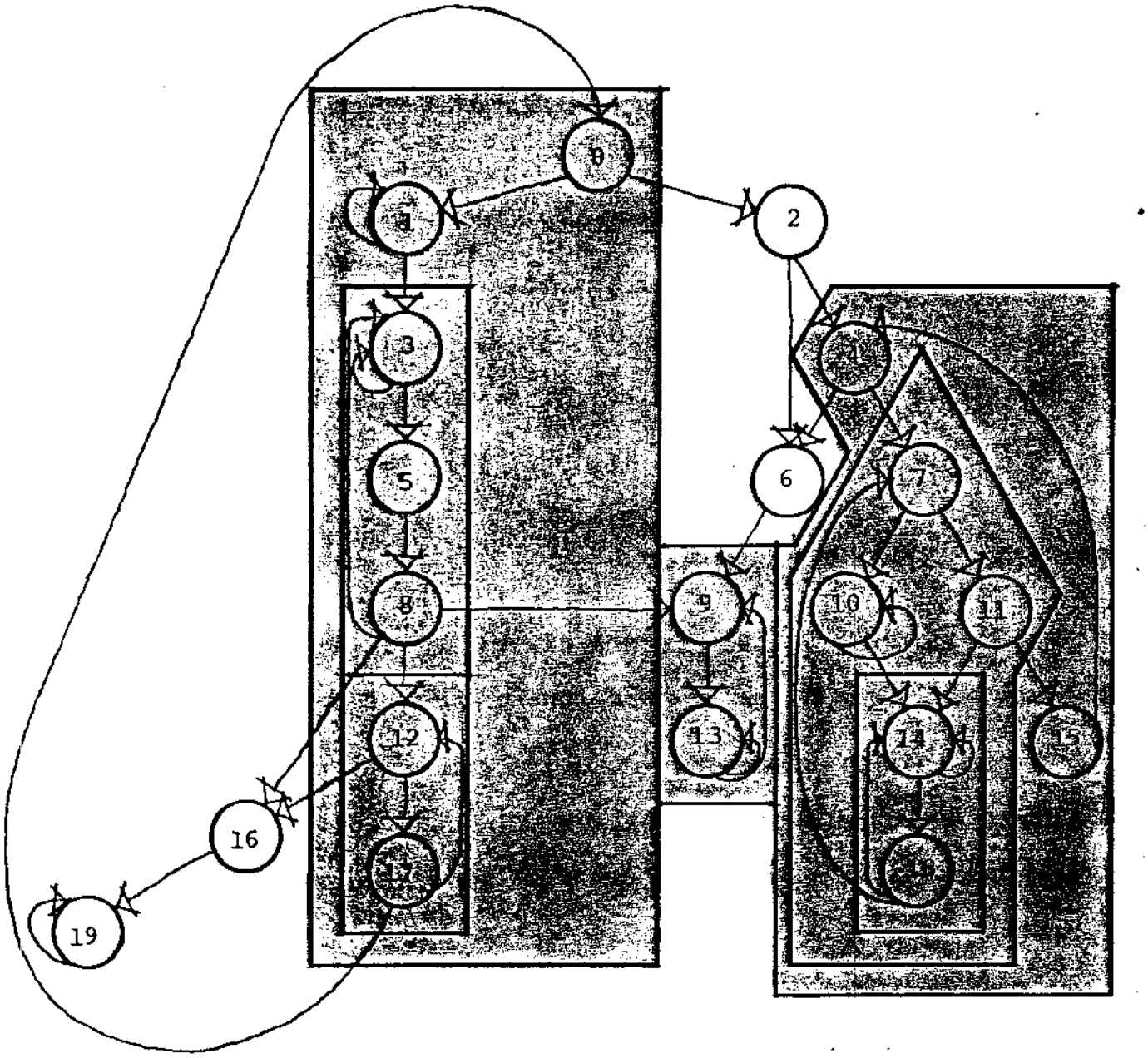


FIGURA 4.2

PROVA.

Se existe circuito C tal que $VC \subseteq VR$ e $r \notin VC$, então seja $r' \in VC$ que domina todos os vértices de VC (r' existe porque G é redutível). Considere o subgrafo maximal fortemente conexo R' de vértices dominados por r'. R'(r') é uma super-região pois contém pelo menos C, e portanto $|VR'| \geq 2$. Pela transitividade das relações de dominância e de ligação forte, $VR' \subseteq VR$. Pela anti-simetria de domina , $r \notin VR'$, donde $VR' \subsetneq VR$, e R não é uma super-região minimal.

Suponha agora que R contém a super-região R'(r'), com $r' \neq r$. Pela anti-simetria de domina, $r \notin VR'$. Mas R' contém um circuito, logo existe em R um circuito que não passa por r. □

Com base na propriedade 4.4.1, pode-se construir um algoritmo simples para identificar a região R(r), dado o vértice r e a estrutura de incidência (V) do grafo.

Como R é fortemente conexo, todo vértice ou aresta (que não é laço) de R está num circuito. Pela propriedade 4.4.1, basta identificar, então, todos os circuitos que passam por r e têm todos os vértices dominados por r. Uma vez que num grafo redutível todo circuito C passa por uma única aresta de retorno (u,v) e v domina todos os vértices do circuito, é necessário apenas usar a estrutura de incidência de G para identificar os vértices que são origens das arestas de retorno com destino r,

e para fazer "busca em profundidade inversa" a partir desses vértices (isto é, evoluindo no sentido oposto ao das arestas). Essa busca em profundidade inversa (executada pelo algoritmo Busca-por-Arestas-Incidentes) é sempre limitada pelo vértice r que domina todos os vértices da região. A figura 4.3 apresenta o algoritmo Identifica-Região. O algoritmo assinala os vértices da região que são destino de aresta com origem em r, a fim de evitar a inclusão de arestas múltiplas quando da aplicação de uma transformação T2, como será visto posteriormente.

PROPRIEDADE 4.4.2.

Seja R(r) uma região no grafo G, e *num* uma numeração em profundidade dos vértices de G. Se $v \in VR \setminus \{r\}$ e $num(v) < num(w)$, para todo $w \in VR \setminus \{r,v\}$, então a aresta (r,v) existe e satisfaz a condição PDU.

PROVA.

Como $v \in VR \setminus \{r\}$, todo predecessor u de v em G está em VR (porque r domina v, logo r domina u, e existem caminhos de r a u, de u a v, e de v a r, de forma que r e u são fortemente ligados). Suponha que existe $(u,v) \in aR$, com $u \neq r$. (u,v) não é uma aresta de retorno, caso contrário haveria em R um circuito que não passa por r. Pela propriedade NP, $num(u) < num(v)$, o que contradiz a hipótese. Portanto, (r,v) existe e satisfaz a condição PDU. □

(esta página foi deixada em branco propositalmente)

```
                Algoritmo Identifica-Região (r)

1. VR ← {r}
2. aR ← ϕ
3. Para cada aresta de retorno (w,r) em  aG
       execute o algoritmo Busca-por-Arestas-Incidentes (w,r)


        Algoritmo Busca-por-Arestas-Incidentes (u,v)

1. aR ← aR ∪ {(u,v)}
2. Se  u ∉ VR
       então ┌ VR ← VR ∪ {u}
             │ para cada aresta (w,u)
             └     aplique  recursivamente o algoritmo para (w,u)
       senão  se  u = r
                  então assinale que  v  recebe aresta de  r.
```

FIGURA 4.3.

DEFINIÇÃO. Sejam $G = (VG, aG, n_o)$ e $(r,v) \in aG$ que satisfaz a condição PDU, com v em $R(r) = (VR, aR, r)$. Seja G' o grafo obtido após a aplicação de T1 ou T2 a todas as arestas com origem v em aR. Diremos que $G' = T12(G,v)$. É fácil mostrar que T12 está bem definida, pois G' não depende da ordem em que foram escolhidas as arestas para a aplicação das transformações.

PROPRIEDADE 4.4.3.

Seja $R(r) = (VR, aR, r)$ uma região do grafo G, e $(r,v) \in aR$ que satisfaz a condição PDU. Se $G' = T12(G,v)$, temos que em G', $\nabla'v = \{r\}$ e $\Delta'v \cap VR = \phi$ (conseqüentemente, $(v,v) \notin aG'$). Além disso,

ou I) $G'[VR \setminus \{v\}] = (\{r\}, \{(r,r)\}, r)$, que não é uma região,

ou II) $R' = G'[VR \setminus \{v\}]$ é uma região de G' com raiz r.

PROVA.

Claramente a condição PDU é sempre satisfeita pela aresta (r,v) após cada transformação $T_i(G,(v,w))$, e pela definição de T1, T2 e T12, não existe aresta (v,w) com $w \in VR$ em G'. Se $VR = \{r,v\}$, então $G'[VR \setminus \{v\}] = (\{r\},\{(r,r)\}, r)$. Senão, $|VR \setminus \{v\}| \geq 2$. As transformações mantêm a dominância de r sobre todos os vértices da região (uma vez que só são acrescentadas eventualmente arestas partindo de r), e não se originam a

dominância de r sobre qualquer vértice do grafo (propriedade 4.2.1). Uma vez que toda aresta incluída liga vértices já fortemente ligados, as transformações não originam ligação forte entre quaisquer vértices de G. Todo circuito de G que não passa por alguma aresta com origem v em aR é mantido, enquanto os demais são "curto-circuitados" pela substituição da seção (r,v,w) pela aresta (r,w). Logo, todas as ligações fortes são mantidas, exceto aquelas que envolvem o vértice v e qualquer vértice de VG não alcançável a partir de v através de um caminho cuja primeira aresta não pertence a aR. Dessa maneira, G'[VR \ {v}] é uma região de G' com raiz r. □

DEFINIÇÃO

Como conseqüência da propriedade 4.4.3, podemos definir a transformação $T12^{+}(G,R)$, que é o resultado da aplicação de T12 a todos os vértices de VR \ {r}, onde R = R(r). Também se pode demonstrar que $T12^{+}$ está bem definida, pela independência do grafo resultante quanto à ordem de tomada dos vértices v em VR \ {r}, a menos da necessidade de (r,v) satisfazer a condição PDU.

PROPRIEDADE 4.4.4.

Sejam G = (VG, aG, n_o), R(r) uma região de G, e $G' = T12^{+}$ (G,R). Então G'[VR] é uma estrela com vértice inicial r e com

laço (r,r).

PROVA.

Aplicações repetidas da propriedade 4.4.3. □

Com base nas propriedades 4.4.2, 4.4.3 e 4.4.4, pode-se desenvolver um algoritmo para reduzir uma região R(r) a uma estrela, associando a cada aresta (r,v) com $v \in VR$, os valores k e c resultantes de todos os caminhos de r a v em R(r), de forma a manter a solução maximal dos Y no sistema de equações para o novo grafo obtido igual à solução para o grafo original. Esse algoritmo é apresentado na figura 4.4. Supõe-se que é conhecida uma numeração em profundidade dos vértices de G, e que a estrutura de R(r) suporta os valores k e c associados a cada aresta em aR. A transformação T1(G(v,v)) estabelece um novo par (k,c) para a única aresta com destino v que não é laço, e elimina o laço em questão. A transformação T2(G(v,w)) elimina a aresta (v,w). Se (r,w) já é uma aresta do grafo, os valores k e c correspondentes a essa aresta são atualizados, caso contrário a aresta (r,w) é adicionada com os valores k e c calculados. Todos os novos valores de k e c são calculados como foi apresentado na seção 4.3. Note-se que nesse algoritmo as transformações T1 e T2, ao invés de criarem um novo grafo G', simplesmente modificam o grafo G dado.

```
Algoritmo Reduz-Região (R(r))    /* T12+(G,R(r)) */

1. Para cada vértice  v ∈ VR \ {r}  em ordem crescente da
   numeração em profundidade
   [ se existe laço  (v,v)
        então execute  T1(G(v,v)), alterando  k_rv  e  c_rv
     para cada aresta  (v,w) em  aR
        execute  T2(G,(v,w)), dando valores convenientes
                              a  k_rw  e  c_rw .
```

FIGURA 4.4.

PROPRIEDADE 4.4.5.

Dada uma numeração em profundidade dos vértices de um grafo redutível $G = (VG, aG, n_o)$, o vértice de maior numeração em VG que é destino de uma aresta de retorno, é raiz de uma região.

PROVA.

Seja r o vértice de maior numeração em VG que é destino de uma aresta de retorno (u,r). Como (u,r) é uma aresta de retorno, existe um circuito C em G que passa por (u,r). Como r domina u, r domina v, para todo v em VC. Seja R o subgrafo

maximal fortemente conexo de G, cujos vértices são todos dominados por r. $|VR| \geq 2$ pois R contém C, e R(r) é uma super-região.

Suponha que existe um circuito C' em R que não passa por r. Como G é redutível, existe um vértice $v' \in VC'$ que domina todos os vértices em VC', e v' é destino de uma aresta de retorno. Como $v' \in VR$, r domina v', e, pela propriedade NP, v' tem numeração maior que r, o que contradiz a hipótese. Logo, não existe um tal circuito C', e pela propriedade 4.4.1, R é uma região de G. □

PROPRIEDADE 4.4.6.

Seja R(r) uma região do grafo redutível G, e seja G' = $= T12^{+}(G, R(r))$. Uma numeração em profundidade *num* dos vértices de G também é válida para os vértices de G'.

PROVA.

Se (u,v) é uma aresta de retorno em G', então v domina u em G', e pela propriedade 4.2.1, v domina u em G. Pela propriedade NP, $num(v) < num(u)$.

Se (u,v), $u \neq v$, não é aresta de retorno em G', então:

CASO 1: Se $v \in VR \setminus \{r\}$, então pela propriedade 4.4.4, u=r. Pela definição de região, r domina v em G, e pela propriedade NP, $num(u) < num(v)$.

CASO 2: Se $v \in VG \setminus (VR \setminus \{r\})$, então $(u,v) \in aG$. Como v não domina u em G' (caso contrário (u,v) seria uma aresta de retorno), existe um caminho C' de n_o a u em G', que não passa por v.

CASO 2.a. Se C' não passa por uma aresta (r,w), com w em VR, C' é um caminho em G.

CASO 2.b. Senão existe pelo menos um caminho C de n_o a u em G, tal que C não passa por v. C é facilmente identificado usando seções de C' que vão de n_o a r, e de w a u, e um caminho C" de r a w em G[VR] tal que C" não passa por v. (C" existe porque se todo caminho de r a w em G passasse por v, então v pertenceria a VR, logo v = r, o que seria uma contradição, pois C' passa por r mas não passa por v).

Tanto no caso (a) como no caso (b), v não domina u em G, e portanto (u,v) não é aresta de retorno em G, donde $num(u) < num(v)$. □

PROPRIEDADE 4.4.7.

Seja $G = (VG, aG, n_o)$ um grafo redutível, e *num* uma numeração em profundidade dos vértices de G. Seja r o vértice com

num(r) máximo que é destino de aresta de retorno em G. Seja G' = $T12^{+}$(G, R(r)). Tem-se então que (u,v) é uma aresta de retorno em aG' sse (u,v) ∈ aG \ aR e (u,v) é uma aresta de retorno em G.

PROVA.

Pela maximalidade de R, todas as arestas de retorno com destino r no grafo G estão em aR, e pela propriedade 4.4.4 elas são todas eliminadas pela transformação $T12^{+}$(G,R). É fácil ver que se um vértice v ∈ VG \ (VR \ {r}) domina u ∈ VG em G, então v domina u em G', e como arestas em aG \ aR não são eliminadas por $T12^{+}$(G,R), nenhuma outra aresta de retorno de G deixa de sê-lo em G'.

Agora, se v ∈ VR \ {r}, então V'v = r e r domina v em G'. Logo não existe aresta de retorno cujo destino é um vértice em VR \ {r} no grafo G'. Como $T12^{+}$ só afeta arestas que têm ambos os extremos em VR, e além disso as propriedades 4.2.1 e 4.4.4 valem, essa transformação não acrescenta arestas de retorno com destino v ∈ VG \ (VR \ {r}). □

COROLÁRIO 4.4.1.

Uma sequência de aplicações da transformação $T12^{+}$ a um grafo redutível sempre converge para um grafo acíclico.

PROVA.

Segue imediatamente da propriedade 4.4.7 (pois diminui o número de arestas de retorno), e da propriedade 4.4.5.

PROPRIEDADE 4.4.8.

Seja $G = (VG, aG, n_o)$ um grafo acíclico que não é uma estrela e seja *num* uma numeração em profundidade dos vértices de G. Seja $VS = \{v \in VG \setminus \{n_o\} \mid \Delta v \neq \phi\}$. (Pela definição de uma estrela, $VS \neq \phi$.) Se para um $v \in VS$, $num(v) < num(w)$, para todo $w \in VS \setminus \{v\}$, então a aresta (n_o, v) existe e satisfaz a condição PDU.

PROVA.

Seja um $v \in VS$ na condição acima, e suponhamos que existe $(u,v) \in aG$, com $u \neq n_o$. Como não existe aresta de retorno em G, v não domina u, e pela propriedade NP, $num(u) < num(v)$, o que contradiz a hipótese, uma vez que $u \in VS$. Logo, (n_o, v) existe e satisfaz a condição PDU. □

PROPRIEDADE 4.4.9.

Seja $G = (VG, aG, n_o)$ um grafo acíclico. G pode ser reduzido a uma estrela com um número finito de transformações T1 ou T2.

PROVA.

Seja $f(G) = |\{(v,w) \in aG \mid v \neq n_o\}|$. Provaremos a propriedade por indução em $f(G)$.

BASE : Se $f(G) = 0$ então G é uma estrela.

PASSO: Se $f(G) \geq 1$ então, pela propriedade 4.4.8, existe um vértice v em $VG \setminus \{n_o\}$ tal que (n_o, v) satisfaz a condição PDU e $\{(v,w) \in aG\} \neq \phi$. Com $|\Delta v|$ transformações T1 ou T2, obtém-se um grafo $G' = (VG, aG', n_o)$, com $aG' = (aG \setminus \{(v,w) \mid w \in VG\}) \cup \{(n_o,w) \mid (v,w) \in aG\}$. É fácil ver que G' é acíclico, e que $f(G') = f(G) - |\{(v,w) \mid w \in VG\}| < f(G)$. □

Note-se que as propriedades 4.4.8 e 4.4.9 são semelhantes às propriedades 4.4.2 e 4.4.4 respectivamente, mas são aplicáveis a grafos acíclicos ao invés de regiões.

4.5. IMPLEMENTAÇÃO DO ALGORITMO GW

Se o grafo G dado não for acíclico, utilizando a propriedade 4.4.5, um vértice r que é raiz de região em G é escolhido, e a região $R(r)$ é identificada. O algoritmo Reduz-Região efetua a transformação $T12^+(G,R(r))$, gerando um grafo redutível (propriedade 4.2.3), com menos arestas de retorno (propriedade 4.4.7), mas com a mesma solução maximal para os Y do sistema

associado a G (seção 4.3). Pelo corolário 4.4.1, o processo pode ser repetido um número finito de vezes, resultando num grafo acíclico. Devido à propriedade 4.4.6 não é necessário renumerar os vértices.

Pela propriedade 4.4.9, um grafo acíclico G pode ser reduzido a uma estrela com um número finito de transformações T1 ou T2. Essa sequência final de transformações é semelhante a uma transformação $T12^{+}$, e verifica-se facilmente que o algoritmo Reduz-Região pode ser aplicado neste caso. Ao final, cada aresta (n_o, v) terá associados valores de k e c resultantes de todos os caminhos de n_o a v no grafo original. Usando o valor de contorno y dado, o valor de Y_{n_o} será: $Y_{n_o} = y \wedge (k_{n_o n_o} \wedge Y_{no} \vee c_{n_o n_o})$. A solução maximal dessa equação é dada por: $Y_{n_o} = y \wedge (k_{n_o n_o} \vee c_{n_o n_o})$, e como $k_{n_o n_o} \geq c_{n_o n_o}$, $Y_{n_o} = y \wedge k_{n_o n_o}$.

(Lembrando que se a aresta (n_o, n_o) não existe, $k_{n_o n_o} = U$ e $c_{n_o n_o} = U$).

Para cada $v \in VG \setminus \{n_o\}$, $Y_v = k_{n_o v} \wedge Y_{n_o} \vee c_{n_o v}$.

A figura 4.5 apresenta o algoritmo GW, sem o valor de contorno. A seguir são apresentados o procedimento GW, que implementa esse algoritmo, e uma série de procedimentos auxiliares, na figura 4.6.

Algoritmo GW(G)

1. Numere em profundidade os vértices de G.
2. Crie a estrutura de incidência, identificando os vértices que são destino de arestas de retorno e os que têm laços.
3. Para cada vértice que é destino de arestas de retorno, na ordem decrescente de numeração

   execute Identifica-Região (r)

   execute Reduz-Região (R(r))
4. Execute o algoritmo Reduz-Região ($G(n_o)$)
5. $Y_{n_o} \leftarrow \phi$
6. Para todo vértice v em $VG \setminus \{n_o\}$

   $Y_v \leftarrow Y_{n_o} \wedge k_{n_o v} \vee c_{n_o v}$

FIGURA 4.5

No procedimento GW, supõe-se que a estrutura de adjacência Δ dada suporta os valores k e c associados a cada aresta do grafo, valores esses que poderão ser alterados durante a sua execução. O procedimento GW utiliza ainda a estrutura de incidência de G, particionada em dois vetores de listas: ∇ret, que contém as arestas de retorno e os laços, e ∇av, que contém as demais arestas incidentes a cada vértice. (Os laços são colocados

nas listas Vret por conveniência de programação.) Cada aresta nessa estrutura contém informação quanto à localização dessa mesma aresta na estrutura Δ.

A cada nova raiz r considerada, a estrutura de adjacência ΔR da região R(r) é determinada. Cada aresta em ΔR também contém a localização da sua correspondente na estrutura Δ (copiada da estrutura de incidência), de forma a possibilitar uma atualização dos valores k e c associados a esta em tempo constante.

Quatro vetores característicos são usados para indicar, para cada vértice v do grafo, se este é destino de arestas de retorno (v será raiz de região), se contém laço (será necessário executar T1(G,(v,v))), se pertence à região corrente (usado na identificação da região),e se é destino de uma aresta com origem na raiz r da região corrente (ou com origem no vértice inicial). Esse último vetor é usado quando da execução de T2(G,(v,w)), para algum sucessor v de r em R(r). Se o componente correspondente ao vértice w contiver "verdadeiro", então Localiza-Aresta-de-r[w] conterá o localizador (ou "apontador") para a aresta (r,w) na estrutura de adjacência Δ de G, onde estão os valores k_{rw} e c_{rw} a serem atualizados. Um quinto vetor lógico, Por-Visitar, é utilizado durante o procedimento Numera-em-Profundidade, para indicar os vértices que ainda não foram visitados.

Os vetores Numeração-G e Vértice-G-com-Numeração conterão

respectivamente uma numeração em profundidade dos vértices de G, e a função inversa dessa numeração, ou seja, o vértice que foi associado a um dado número.

O procedimento GW consiste basicamente em identificar os vértices que são destino de arestas de retorno, e tomá-los na ordem inversa da numeração em profundidade, reduzindo as regiões a eles associadas, a estrelas. A seguir é feita a redução do grafo acíclico resultante a uma estrela, e finalmente são calculados os Y do vértice original e dos demais vértices do grafo.

Os procedimentos Inicializa-Vetores-Auxiliares, Numera-em-Profundidade, Constrói-Estrutura-de-Incidência-e-Identifica-Raízes-e-Laços, e Reduz-Grafo, existem apenas para dar mais clareza ao procedimento GW, mas só são chamados em um ponto, e na prática poderiam ser expandidos ali.

Apresentamos uma única versão do procedimento Numera-em-Profundidade, que será usada tanto para regiões como para o grafo acíclico. A diferença entre as duas versões necessárias na prática é apenas quanto à estrutura dos elementos das listas de adjacência (ΔR ou Δ).

O procedimento Constrói-Estrutura-de-Incidência-e-Identifica-Raízes-e-Laços toma cada aresta (v,w) do grafo e determina, através da numeração em profundidade de v e de w, se ela é de retorno ou não, colocando-a respectivamente numa lista de arestas de retorno incidentes a w (∇ret[w]), ou numa lista de "arestas

de avanço" incidentes a w (∇av[w]). (Como já foi dito, os laços serão colocados nas listas ∇ret por conveniência). Essas listas serão usadas para identificar rapidamente uma região do grafo, dada a raiz.

As arestas são tomadas em ordem decrescente de numeração em profundidade dos seus vértices de origem, para que as estruturas de incidência sejam ordenadas crescentemente a partir da cabeça. Dessa forma, se houver um laço em um vértice w, (w,w) será a primeira aresta da lista ∇ret[w]. Uma eventual aresta (r,w), onde r é a raiz da região R e $w \in VR$, ou uma aresta (n_o, w), será a primeira aresta da lista ∇av[w].

Os vértices w que são destino de uma aresta com origem no vértice inicial (0), são assinalados, e a localização dessa aresta na estrutura de adjacência Δ do grafo é colocada em Localiza-Aresta-de-r [w].

O procedimento Região identifica a região com raiz r, como descrito no algoritmo da figura 4.3, criando a estrutura de adjacência ΔR, ao mesmo tempo em que assinala os vértices que são destino de aresta com origem na raiz, e a localização dessas arestas na estrutura Δ. Em seguida é feita uma numeração em profundidade dos vértices e a redução da região a uma estrela. Finalmente são restauradas as estruturas que descrevem a região, e são eliminadas todas as arestas de retorno na entrada correspondente a r na estrutura de incidência do grafo (∇ret[r]). A eliminação

é efetuada nesse ponto por questões de simplicidade e desempenho.) A função Torna-Lista-Unitária é utilizada apenas para documentar a permanência do laço (r,r). Na prática, porém, a atribuição será substituída por: ∇ret[r].próximo ← lista vazia.

O procedimento Reduz-Região implementa o algoritmo apresentado na figura 4.4. Todas as arestas incidentes aos vértices v da região, com $v \neq r$, são eliminadas da estrutura ∇av, exceto aquelas com origem r. Mais uma vez se aplicam os comentários anteriores quanto ao ponto de eliminação dessas arestas e quanto à utilização da função Torna-Lista-Unitária.

O procedimento Reduz-Grafo é semelhante ao procedimento Reduz-Região, diferenciando-se deste pela estrutura de adjacência do grafo a ser reduzido. As arestas incidentes aos vértices do grafo não são eliminadas por ser desnecessário.

O procedimento T1 recebe como parâmetros o vértice v, que contém o laço a ser eliminado, e a localização da aresta (r,v) na estrutura Δ. A localização da aresta (v,v) em Δ é determinada utilizando a informação contida na primeira aresta da lista ∇ret[v]. A função Primeiro-Elemento objetiva apenas evidenciar a referência ao primeiro elemento de uma lista, o que representa uma simples consulta à cabeça da lista. Os valores de k e c correspondentes à aresta (r,v) são alterados em Δ, conforme descrito na seção 4.3. Para eliminar o laço (v,v), só é necessário alterar o vetor característico Existe-Laço.

O procedimento T2 aplicado à aresta (v,w), quando o único predecessor de v é o vértice r, após usar o k e o c associados à aresta (v,w), elimina-a da estrutura Δ. Se a aresta (r,w) já existe, então ela é localizada através do vetor Localiza-Aresta-de-r, e os seus valores k e c são alterados. Caso contrário, uma nova aresta (r,w) é incluída em Δ e na estrutura de incidência de G, além de serem atualizados os vetores Existe-Laço (se w = r), Recebe-Aresta-de-r, e Localiza-Aresta-de-r.

Note-se que a variável representada por r pode ser tanto uma raiz de região como o vértice inicial n_o do grafo, e que se um vértice v é sucessor de n_o no grafo original, então Recebe-Aresta-de-r [v] contém "verdadeiro" desde a execução de Constrói-Estrutura-de-Incidência-e-Identifica-Raízes-e-Laços. Isso porém não interfere durante as execuções de Reduz-Região, uma vez que a única região na qual w poderia entrar sem ser raiz seria a região com raiz n_o.

A figura 4.7 indica as operações básicas necessárias para executar cada linha (ou grupo de linhas) do procedimento GW e seus auxiliares.

Uma vez que os procedimentos ativados nas linhas 1, 2, 3 e 7 do procedimento GW poderiam ter sido expandidos naqueles pontos, o custo associado a cada uma daquelas linhas irá representar o custo de execução de cada procedimento respectivamente, sem considerar as chamadas e retornos. A malha da linha 4 é

```
procedimento  GW(G = (VG,Δ))
1.  Inicializa-Vetores-Auxiliares
2.  Numeração-G, Vértice-G-com-Numeração := Numera-G-em-Profun-
                                            didade (VG,Δ)
3.  Constrói-Estrutura-de-Incidência-e-Identifica-Raízes-e-Laços
4.  para  i:= |VG|-1   até  0  faça
5.    se Recebe-Aresta-de-Retorno [Vértice-G-com-Numeração [i]]
6.    então  Região(Vértice-G-com-Numeração [i])
7.  Reduz-Grafo
8.  se Existe Laço[0]
9.  então  loc := Primeiro-Elemento (∇ret[0]).localiza
10.        Elimina-Aresta (loc)
11. Y[0]  := ϕ
12. Z     := ϕ
13. para toda aresta a em Δ[0] faça /* Δ[0] não contém (0,0) */
14.     Y[a.destino]  := z ∧ a.k ∨ a.c
15. devolva  Y
```

FIGURA 4.6a

```
procedimento  Inicializa-Vetores-Auxiliares

1.  para todo  v  em  VG  faça
2.      Recebe-Aresta-de-Retorno [v]  := falso
3.      Existe-Laço [v]               := falso
4.      Pertence-a-R-Corrente [v]     := falso
5.      Recebe-Aresta-de-r [v]        := falso
6.      Por-visitar [v]               := verdadeiro
7.      ∇av [v]                       := lista vazia
8.      ∇ret [v]                      := lista vazia
9.      ΔR [v]                        := lista vazia

procedimento  Numera-em-Profundidade (G = (VG,Δ))

1.  i := |VG| -1
2.  Observa-Sucessores (0)
3.  devolva (Numeração, Vértice-com-Numeração)

procedimento Observa-Sucessores (u)

4.  Por-Visitar [u] := falso
5.  para toda aresta  a  em  Δ[u]  faça
6.      v := a.destino
7.      se Por-Visitar [v]
8.      então Observa-Sucessores (v)
9.  Numeração [u] := i
10. Vértice-com-Numeração [i] := u
11. i := i - 1
```

FIGURA 4.6b

```
procedimento Constrói-Estrutura-de-Incidência-e-Identifica-Raí-
             zes-e-Laços
1.  para i := |VG|-1 até 1 faça
2.    v := Vértice-G-com-Numeração [i]
3.    Por-Visitar [v] := verdadeiro
4.    para toda aresta a em Δ[v] faça
5.      w := a.destino
6.      se Numeração-G [w] > Numeração-G [v]
7.      então Inclui (∇av [w], (v, localizador (a)))
8.      senão Inclui (∇ret [w], (v, localizador (a)))
9.            se v = w
10.           então Existe-laço [w]                    := verdadeiro
11.           senão Recebe-Aresta-de-Retorno [w] := verdadeiro
12. Por-Visitar [0] := verdadeiro
13. para toda aresta a em Δ[0] faça
14.   w := a.destino
15.   se w = 0
16.   então Inclui (∇ret [0], (0, localizador (a)))
17.         Existe-Laço [0] := verdadeiro
18.   senão Inclui (∇av [w], (0, localizador (a)))
19.         Recebe-Aresta-de-r [w]  := verdadeiro
20.         Localiza-Aresta de r [w] := localizador (a)
```

FIGURA 4.6c

```
procedimento Região (r)
1.  Pertence-a-R-Corrente [r]  := verdadeiro
2.  m                          := 0 /* m nomeia os vértices da região */
3.  VR[0]                      := r
4.  para toda aresta a em ∇ret[r] faça /* Identifica R(r) */
5.      Busca-por-Arestas-Incidentes (r, localizador (a))
6.  Numeração-R, Vértice-R-com-Numeração := Numera-R-em-Profun
                                            didade (VR,ΔR)
7.  Reduz-Região
8.  ∇ret [r] : = Torna-Lista-Unitária (∇ret [r])
9.  para i := m até 0 faça
10.     v                          :=VR[i]
11.     Pertence-a-R-Corrente [v]  := falso
12.     Recebe-Aresta-de-r [v]     := falso
13.     ΔR[v]                      := lista vazia
14.     Por-Visitar [v]            := verdadeiro

procedimento Busca-por-Arestas-Incidentes (w,loc-v-w)
15. v := loc-v-w.origem
16. ℓ := loc-v-w.localiza
17. Inclui (ΔR[v] , (w,ℓ))
18. se Pertence-a-R-Corrente [v]
19. então se v = r
20.       então Recebe-Aresta-de-r [w]    := verdadeiro
21.             Localiza-Aresta-de-r [w] := ℓ
22.   senão Pertence-a-R-Corrente [v] := verdadeiro
23.         m                         := m + 1
24.         VR[m]                     := v
25.         para toda aresta a em ∇av[v] faça
26.             Busca-por-Arestas-Incidentes (v, localizador (a))
```

FIGURA 4.6d

```
procedimento  Reduz-Região  /* T12+(G,R(r))  */
1.  para  i := 1 até |VR|-1  faça  /* r  tem numeração zero */
2.      v       : Vértice-R-com-Numeração [i]
3.      ∇av[v]  := Torna-Lista-Unitária (∇av [v] )
4.      loc-r-v:= Localiza-Aresta-de-r [v]
5.      se  Existe-Laço [v]
6.      então  T1 (v, loc-r-v)
7.      para toda aresta  a  em  ΔR[v]  faça
8.          T2(r,v, loc-r-v, a.localiza)
```

```
procedimento  Reduz-Grafo
1.  para  i := 1  até  |VG|-1  faça
2.      v       :=Vértice-G-com-Numeração [i]
3.      loc-0-v := Localiza-Aresta-de-r [v]
4.      se  Existe-Laço [v]
5.      então  T1(v, loc-0-v)
6.      para toda aresta  a  em  Δ[v]  faça
7.          T2(0, v, loc-0-v, localizador (a))
```

FIGURA 4.6e

```
procedimento T1 (v, loc-r-v)

1.  loc-v-v          := Primeiro-Elemento (∇ret [v]).localiza
2.  c_rv             := c_rv ∧ k_vv   /* loc-r-v.c := loc-r-v.c∧ loc-v-v.k */
3.  k_rv             := k_rv ∧ k_vv   /* loc-r-v.k := loc-r-v.k ∧ loc-v-v.k */
4.  Existe-Laço[v]   := falso
```

```
procedimento T2 (r,v, loc-r-v, loc-v-w)

1.  C := c_rv ∧ k_vw ∨ c_vw
2.  K := k_rv ∧ k_vw ∨ c_vw
3.  w := loc-v-w.destino
4.  Elimina-Aresta (loc-v-w)
5.  se Recebe-Aresta-de-r [w]
6.  então  loc-r-w   := Localiza-Aresta-de-r [w]
7.         c_rw      := c_rw ∧ C
8.         k_rw      := k_rw ∧ K
9.  senão  Inclui-G(Δ[r], (w,C,K))
10.        se  w ≠ r
11.        então  Inclui (∇av[w],(r, Primeiro-Elemento (Δ[r])))
12.        senão  Inclui (∇ret[w],(r,Primeiro-Elemento (Δ[r])))
13.               Existe-Laço [r]  := verdadeiro
14.        Recebe-Aresta-de-r [w]  := verdadeiro
15.        Localiza-Aresta-de-r [w]  := Primeiro-Elemento (Δ[r])
```

FIGURA 4.6f

executada uma vez para cada vértice do grafo, com o teste de fim de iteração sendo efetuado após cada execução. O teste da linha 6 resulta verdadeiro para cada raiz de região em G. Considerando o pior caso (existência de laço em n_o), será desprezado o eventual desvio na comparação da linha 8 e será computada a execução dos comandos nas linhas 9 e 10. Como o grafo resultante após a execução de Reduz-Grafo é uma estrela com vértice inicial n_o, ocorrem N_v-1 iterações na malha da linha 13, sendo o teste de controle efetuado antes de cada execução. É necessária ainda uma subscrição no vetor Δ antes da atribuição inicial à variável de controle.

Ocorrem N_v iterações na malha da linha 1 do procedimento Inicializa-Vetores-Auxiliares com o teste de fim de iteração sendo efetuado após cada execução do corpo da malha, de forma que seu controle demanda $(2N_v+1)\mu_{ac}$, $N_v\mu_{ar}$, $(N_v+1)\mu_{at}$, $N_v\mu_c$ e $(N_v-1)\mu_d$.

Na linha 2, mais uma vez foi utilizada uma atribuição apenas para indicar as variáveis que contêm o resultado do procedimento ativado. Os custos dos procedimentos Numera-G-em-Profundidade e Observa-Sucessores são avaliados de maneira análoga ao das linhas 9, 10, 26, 27, 28, 30, 31, 32 e 33 do procedimento HU, acrescentando-se $N_v(\mu_{ac} + \mu_{at} + \mu_s)$, referentes à linha 9 de Numera-G-em-Profundidade.

No procedimento Constrói-Estrutura-de-Incidência-e-Identifica-Raízes-e-Laços, a malha da linha 1 é executada para todo

vértice que não é o inicial, logo ocorrem N_v-1 iterações, com teste de controle efetuado antes de cada execução do corpo. Nas N_v entradas nas malhas das linhas 4 e 13, ocorrem N_a iterações sendo o teste efetuado antes de cada uma delas. A cada entrada é necessário uma subscrição para localizar a cabeça da lista. O teste da linha 6 ocorre para toda aresta do grafo cuja origem não é o vértice inicial. Cada aresta é, então incluída na fila Vav ou na fila Vret (linhas 7, 8, 16, 18). Os laços que não estão no vértice inicial e as arestas de retorno são submetidas ao teste da linha 9, e as arestas com origem em n_o são submetidas ao teste da linha 15. As atribuições das linhas 10 e 17 são efetuadas ao todo N_L vezes, e a da linha 11 N_4 vezes. Para todo sucessor do vértice inicial que não é o próprio, ocorrem as atribuições das linhas 19 e 20.

O procedimento Reduz-Grafo consiste de uma malha na qual ocorrem N_v-1 iterações sendo o teste de controle efetuado antes de cada execução. As eventuais chamadas ao procedimento T1 na linha 5 são contadas juntamente com as chamadas na linha 6 do procedimento Reduz-Região. No total das N_v-1 entradas, a malha da linha 6 de Reduz-Grafo é executada tantas vezes quanto seja o número de arestas que não são laços no grafo acíclico resultante após todas as reduções de regiões, menos o número de sucessores de n_o naquele grafo que não são laços. O teste de fim de iteração é efetuado antes de cada execução, e ocorre uma subscrição a cada entrada na malha.

Na avaliação do procedimento Região e do seu auxiliar Busca-por-Arestas-Incidentes, serão considerados os custos referentes a todas as N_R ativações de Região dentro de uma execução de GW. Para cada vértice v que entra numa região, são executadas as instruções nas linhas 1, 2, 3, 4 (se v for raiz de região) ou nas linhas 22, 23, 24, 25, respectivamente. Nas N_5 entradas que ocorrem nas malhas das linhas 4 e 25, conjuntamente, o procedimento Busca-por-Arestas-Incidentes é ativado N_6 vezes. Como toda raiz de região tem pelo menos um predecessor em Vret, e os demais vértices da região têm pelo menos um predecessor em Vav, ambas as malhas têm teste de fim de iteração executado apenas após o corpo da malha.

Como o procedimento Numera-R-em-Profundidade poderia ser estendido na linha 6 de Região, não serão considerados os custos de ativação e retorno desse procedimento, assim como os custos da atribuição que foi utilizada para indicar as variáveis que conteriam os resultados do procedimento. Parte dessa observação também se aplica ao procedimento Reduz-Região, que poderia ser expandido na linha 7 de Região. A malha da linha 9 é executada N_5 vezes, com teste de fim de iteração apenas após a execução do corpo. Finalmente, o teste da linha 18 dá falso uma vez para cada vértice de cada região que não é a raiz da região, e a comparação da linha 19 resulta verdadeiro para todas as arestas de todas as regiões com origem na raiz.

O custo de Numera-R-em-Profundidade é análogo ao custo de

Numera-G-em-Profundidade, trocando-se apenas os coeficientes 1, N_v, N_a por N_R, N_5, N_6, respectivamente.

Na malha mais externa de Reduz-Região, ocorrem $(N_5 - N_R)$ iterações, e como toda região tem pelo menos dois vértices, o teste de controle só é efetuado após cada iteração. Considerando conjuntamente as comparações da linha 5 de Reduz-Região e da linha 4 de Reduz-Grafo, o resultado será verdadeiro N_{12} vezes. O corpo da malha na linha 7 de Reduz-Região é executado para cada aresta de cada região que não é laço e que não tem origem na raiz (ou seja, $(N_6 - N_7)$ vezes). Uma vez que toda região é fortemente conexa, toda lista ΔR tem pelo menos um elemento e o teste de fim de iteração só é efetuado após cada execução do corpo da malha. Ocorre ainda uma subscrição a cada entrada.

O procedimento T2 é ativado $(N_6 - N_7)$ vezes durante o processo de redução de regiões, e $(N_8 - N_3)$ vezes durante a redução do grafo acíclico resultante após aquelas transformações. Durante as reduções de regiões, o teste da linha 5 dará falso exatamente uma vez para cada vértice de cada região que não é sucessor da raiz, ou seja $(N_5 - N_7)$ vezes, e durante a redução do grafo acíclico a uma estrela, esse mesmo teste dará falso uma vez para cada vértice do grafo que não é destino de uma aresta com origem n_o, o que é dado por $(N_v - N_3)$ (com um erro de 1 se n_o não tiver predecessor). Dessa forma, as instruções associadas ao *então* são executadas $(N_6 - N_7) + (N_8 - N_3) - (N_5 - N_7) - (N_v - N_3)$, ou $(N_6 + N_8 - N_5 - N_v)$ vezes, enquanto que as associadas ao *senão*

são executadas $(N_5 - N_7 + N_v - N_3)$ vezes, assim como as chamadas de procedimento nas linhas 11 e 12 conjuntamente. Como numa transformação T2 temos $v \neq w$, se a comparação da linha 10 resulta falso, (isto é, se $w = r$) então (v,w) é uma aresta de retorno, logo a instrução na linha 13 é executada N_4 vezes.

Pelo fato de T1 e T2 serem facilmente expandidos nos pontos de ativação, as chamadas e retornos desses procedimentos serão desconsideradas, assim como para todos os procedimentos para inclusão e retirada em lista que foram omitidos no texto. Implementações típicas desses procedimentos aparecem no apêndice 3.

A figura 4.8 apresenta equações que definem o número de operações de cada tipo executadas numa ativação do procedimento GW, nas quais os custos de μ_{av}, μ_{Inclui}, e $\mu_{Inclui-G}$ foram substituídos pelos custos das implementações no apêndice 3.

CUSTO DO PROCEDIMENTO GW

1.	(N_v+1)	$(\mu_{at} + \mu_{ac})$
	N_v	$(\mu_{ar} + \mu_{ac} + \mu_c + 8\mu_s + 8\mu_{at})$
	(N_v-1)	μ_d
2.	1	$\mu_{Numera-Grafo}$
3.	1	$\mu_{Constrói-Estrutura...}$
4.	(N_v+1)	$(\mu_{at} + \mu_{ac})$
	N_v	$(\mu_{ar} + \mu_{ac} + \mu_c)$
	(N_v-1)	μ_d
5.	N_v	$(2\mu_s + \mu_{ac} + \mu_c)$
	N_v-N_R	μ_d
6.	N_R	$(\mu_s + \mu_{cpl} + \mu_{rp}) + \mu_{Região}$
7.	1	$\mu_{Reduz-Grafo}$
8.	1	$(\mu_s + \mu_{ac} + \mu_c)$

FIGURA 4.7a

9+10.	1	$(\mu_{ac} + \mu_{s} + \mu_{sel} + \mu_{at} + \mu_{Elimina\text{-}Aresta})$
11+12.	1	$(\mu_{s} + 2\mu_{AT})$
13.	1	$(\mu_{s} + \mu_{at} + \mu_{ac})$
	N_v	$(\mu_{c} + \mu_{d} + \mu_{ac})$
	(N_v-1)	μ_{av}
14.	(N_v-1)	$(3\mu_{AC} + \mu_{s} + 3\mu_{sel} + 2\mu_{OL} + \mu_{AT})$

FIGURA 4.7b

CUSTO DO PROCEDIMENTO NUMERA-EM-PROFUNDIDADE (+ OBSERVA - ...)

	GRAFO	REGIÕES	
1.	1	N_R	$(\mu_{ac} + \mu_{at})$
2+8.	N_v	N_5	$(\mu_{cpl} + \mu_{rp})$
4.	N_v	N_5	$(\mu_{at} + \mu_s)$
5.	N_v	N_5	$(\mu_{ac} + \mu_{at} + \mu_s)$
	N_a	N_6	μ_{av}
	(N_v+N_a)	(N_5+N_6)	$(\mu_{ac} + \mu_c + \mu_d)$
6.	N_a	N_6	$(\mu_{ac} + \mu_{at} + \mu_{sel})$
7.	N_a	N_6	$(\mu_{ac} + \mu_c + \mu_s)$
	(N_a-N_v+1)	$(N_6-N_5+N_R)$	μ_d
9+..+11	N_v	N_5	$(3\mu_{ac} + \mu_{ar} + 3\mu_{at} + 2\mu_s)$

FIGURA 4.7c

CUSTO DO PROCEDIMENTO CONSTRÓI - ESTRUTURA - ...

1.	N_v	$(\mu_{at} + \mu_c + 2\mu_{ac} + \mu_d)$
	(N_v-1)	μ_{ar}
2.	(N_v-1)	$(\mu_s + \mu_{at} + \mu_{ac})$
3+12.	N_v	$(\mu_s + \mu_{at})$
4+13.	N_v	$(\mu_s + \mu_{at} + \mu_{ac})$
	(N_a+N_v)	$(\mu_{ac} + \mu_c + \mu_d)$
	N_a	μ_{av}
5+14.	N_a	$(\mu_{ac} + \mu_{sel} + \mu_{at})$
6.	(N_a-N_2)	$(2\mu_s + 2\mu_{ac} + \mu_c + \mu_d)$
7+8+16+18.		
	N_a	$(\mu_s + \mu_{Inclui})$
9.	(N_4+N_L)	$2\mu_{ac}$
9+15.	$(N_L+N_2+N_4)$	$(\mu_c + \mu_d)$
10+17.	N_L	$(\mu_s + \mu_{at})$
11.	N_4	$(\mu_s + \mu_{at})$
15.	N_2	μ_{ac}
19+20.	N_2	$2(\mu_s + \mu_{at})$
20.	N_2	μ_{ac}

FIGURA 4.7d

CUSTO DO PROCEDIMENTO **REGIÃO** (+ BUSCA - POR - ...)

1+22.	N_5	$(\mu_s + \mu_{at})$
2+23.	N_5	μ_{at}
	(N_5-N_R)	$(\mu_{ac} + \mu_{ar})$
3+24.	N_5	$(\mu_s + \mu_{at} + \mu_{ac})$
4+25.	N_5	$(\mu_s + \mu_{at} + \mu_{ac})$
	(N_6-N_5)	μ_d
5+26.	N_6	$(\mu_{cp2} + \mu_{rp})$
6.	1	$\mu_{Numera-Região}$
7.	1	$\mu_{Reduz-Região}$
8.	N_R	$(\mu_s + \mu_{sel} + \mu_{at})$
9.	(N_5+N_R)	$\mu_{at} + \mu_{ac}$
	N_5	$(\mu_{ac} + \mu_{ar} + \mu_c)$
	(N_5-N_R)	μ_d

FIGURA 4.7e

10+..+14.	N_5	$(\mu_{ac} + 5\mu_{at} + 5\mu_s)$
15+16.	N_6	$2(\mu_{sel} + \mu_{at} + \mu_{ac})$
17.	N_6	$(\mu_s + \mu_{Inclui})$
18.	N_6	$\mu_s + \mu_{ac} + \mu_c + \mu_d$
19.	$(N_6-N_5+N_R)$	$(2\mu_{ac} + \mu_c)$
	$(N_6-N_5+N_r-N_7)$	μ_d
20+21.	N_7	$[2(\mu_s + \mu_{at}) + \mu_{ac}]$

FIGURA 4.7f

CUSTO DO PROCEDIMENTO REDUZ-REGIÃO

1.	N_5	μ_{at}
	N_5-N_R	$(\mu_{ar} + 3\mu_{ac} + \mu_c)$
	(N_5-2N_R)	μ_d
2.	N_5-N_R	$(\mu_s + \mu_{at} + \mu_{ac})$
3.	N_5-N_R	$(\mu_s + \mu_{sel} + \mu_{at})$
4.	N_5-N_R	$(\mu_s + \mu_{at} + \mu_{ac})$
5.	N_5-N_R	$(\mu_s + \mu_{ac} + \mu_c)$
	$N_5-N_R-N_{12}$	μ_d
6.	N_{12}	μ_{T1}
7.	N_5-N_R	$(\mu_s + \mu_{ac} + \mu_{at})$
	N_6-N_7	$(\mu_{av} + \mu_c + \mu_{ac})$
	$(N_6-N_7-N_5+N_R)$	μ_d
8.	N_6-N_7	μ_{sel} + μ_{T2}

FIGURA 4.7g

CUSTO DO PROCEDIMENTO REDUZ-GRAFO

1.	N_v	$(\mu_{at} + 2\mu_{ac} + \mu_c + \mu_d)$
	N_v-1	$(\mu_{ar} + \mu_{ac})$
2.	N_v-1	$(\mu_s + \mu_{at} + \mu_{ac})$
3.	N_v-1	$(\mu_s + \mu_{at} + \mu_{ac})$
4.	N_v-1	$(\mu_s + \mu_c + \mu_d + \mu_{ac})$
6.	N_v-1	$(\mu_s + 2\mu_{ac} + \mu_{at} + \mu_c + \mu_d)$
	N_8-N_3	$(\mu_{av} + \mu_{ac} + \mu_c + \mu_d)$

CUSTO DO PROCEDIMENTO T1

1. $\mu_{ac} + \mu_{sel} + \mu_{at} + \mu_s$
2. $2\mu_{AC} + 3\mu_{sel} + \mu_{OL} + \mu_{AT}$
3. $2\mu_{AC} + 3\mu_{sel} + \mu_{OL} + \mu_{AT}$
4. $\mu_s + \mu_{at}$

FIGURA 4.7h

CUSTO DO PROCEDIMENTO T2

1.	$(N_6-N_7+N_8-N_3)$	$(3\mu_{sel} + 2\mu_{OL} + \mu_{AT} + 3\mu_{AC})$
2.	$(N_6-N_7+N_8-N_3)$	$(3\mu_{sel} + 2\mu_{OL} + \mu_{AT} + 3\mu_{AC})$
3.	$(N_6-N_7+N_8-N_3)$	$(\mu_{sel} + \mu_{at} + \mu_{ac})$
4.	$(N_6-N_7+N_8-N_3)$	$\mu_{Elimina\text{-}Aresta}$
5.	$(N_6-N_7+N_8-N_3)$	$(\mu_s + \mu_c + \mu_d + \mu_{ac})$
6+7+8.	$(N_6+N_8-N_5-N_v)$	$(\mu_s + \mu_{at} + 4\mu_{AC} + 4\mu_{sel} + 2\mu_{OL} + 2\mu_{AT} + \mu_{ac})$
9+10.	$(N_5-N_7+N_v-N_3)$	$(\mu_s + \mu_{Inclui\text{-}G} + \mu_d + 2\mu_{ac} + \mu_c)$
11+12.	$(N_5-N_7+N_v-N_3)$	$(2\mu_s + \mu_{il})$
13.	N_4	$(\mu_s + \mu_{at})$
14+15.	$(N_5-N_7+N_v-N_3)$	$(3\mu_s + 2\mu_{at} + \mu_{ac})$

FIGURA 4.7i

SOMA VERTICAL DE OPERAÇÕES NO PROCEDIMENTO GW

$$19N_a + 47N_v + 4N_R + 2N_L - 31N_3 + 2N_4 + 38N_5 + 30N_6 - 30N_7 + 9N_8 + N_{12} + 2 \quad \mu_{ac}$$

$$N_v - 8N_3 - 2N_5 + 10N_6 - 8N_7 + 10N_8 + 4N_{12} - 3 \quad \mu_{AC}$$

$$2N_a + 9N_v - 2N_R - 4N_3 + 8N_5 + 2N_6 - 4N_7 - 2 \quad \mu_{ar}$$

$$10N_a + 40N_v - N_R + N_L + 2N_2 + 21N_3 + 2N_4 + 36N_5 + 16N_6 - 19N_7 + 5N_8 + 2N_{12} + 2 \quad \mu_{at}$$

$$-N_v - 2N_3 - 2N_5 + 4N_6 - 2N_7 + 4N_8 + 2N_{12} + 1 \quad \mu_{AT}$$

$$5N_a + 13N_v - N_R + N_L - 7N_3 + N_4 + 6N_5 + 10N_6 - 7N_7 + 4N_8 + 1 \quad \mu_c$$

$$N_v + N_R + N_5 \quad \mu_{cp1}$$

$$N_6 \quad \mu_{cp2}$$

$$5N_a + 12N_v - 2N_R + N_L - 6N_3 + N_4 + 3N_5 + 9N_6 - 7N_7 + 3N_8 - N_{12} - 2 \quad \mu_d$$

$$-4N_3 - 2N_5 + 6N_6 - 4N_7 + 6N_8 + 2N_{12} - 2 \quad \mu_{OL}$$

$$N_v + N_R + N_5 + N_6 \quad \mu_{rp}$$

$$4N_a + 27N_v - 3N_R + N_L - 7N_3 + 2N_4 + 22N_5 + 5N_6 - 5N_7 + 2N_8 + 2N_{12} - 2 \quad \mu_s$$

$$7N_a + 9N_v - 25N_3 + 6N_5 + 29N_6 - 26N_7 + 20N_8 + 7N_{12} + 5 \quad \mu_{sel}$$

FIGURA 4.8

CAPÍTULO V

COMPARAÇÃO DOS MÉTODOS

Devido aos diferentes parâmetros do grafo utilizados na avaliação do número de operações necessário na realização de cada um dos métodos, não existe nenhuma comparação conclusiva quanto ao algoritmo que apresenta o melhor desempenho.

5.1. OS GRAFOS DE FAMÍLIAS AUTO-REPLICANTES

Em KENNEDY [1976], algumas famílias de grafos ditos auto-replicantes são utilizadas para tornar possível a obtenção de um parâmetro único, através do qual podem ser expressos todos os outros parâmetros do grafo. Utilizamos aqui seis das sete famílias de grafos apresentadas por Kennedy, que podem ser definidas recursivamente da seguinte maneira:

1. GRAFO CONCHA ("SEASHELL GRAPH")

   de ordem 1: $G_1 = (\{0,f\}, \{(0,f), (f,0)\}, 0)$

   de ordem $p > 1$: Seja $G_{p-1} = (VG_{p-1}, aG_{p-1}, 0)$ um grafo concha de ordem $p-1$. $G_p = \{VG_{p-1} \cup \{0'\}, aG_{p-1} \cup \{(0',0), (f,0')\}, 0')$, onde $0' \notin VG_{p-1}$, e f é o único vértice tal que $(f,0) \in aG_{p-1}$.

2. GRAFO ESPIRAL ("SPIRAL GRAPH")

   de ordem 1: $G_1 = (\{0,v,f\},\{(0,v),(0,f),(v,0),(v,f)\}, 0)$

de ordem $p > 1$: Seja $G_{p-1} = (VG_{p-1}, aG_{p-1}, 0)$ um grafo espiral de ordem p-1. $G_p = (VG_{p-1} \cup \{0',f'\}$, $aG_{p-1} \cup \{(0',0), (0',f'),(f,0'),(f,f')\},0')$, onde $0',f' \notin VG_{p-1}$, e f é o único vértice em VG_{p-1} que não tem sucessores em G_{p-1}.

3. GRAFO BINÁRIO DE HECHT E ULLMAN ("HECHT-ULLMAN BINARY GRAPH")

de ordem 1: $G_1 = (\{0,f\},\{(0,f), (f,0)\}, 0)$

de ordem $p > 1$: Sejam $G_A = (VG_A, aG_A, 0_A)$ e $G_B = (VG_B, aG_B, 0_B)$ dois grafos binários de Hecht e Ullman de ordem p-1, tais que $VG_A \cap VG_B = \phi$. $G_p = (VG_A \cup VG_B, aG_A \cup aG_B \cup \{(f_A,0_B), (f_B,0_A)\}, 0_A)$, onde f_A e f_B são os únicos vértices origem de p-1 arestas de retorno em G_A e em G_B, respectivamente.

4. GRAFO CADEIA-DUPLA("DOUBLE-CHAIN GRAPH")

de ordem 1: $G_1 = (\{0,f\}, \{(0,f), (f,0)\}, 0)$

de ordem $p > 1$: Seja $G_{p-1}=(VG_{p-1},aG_{p-1},0)$ um grafo cadeia-dupla de ordem p-1. $G_p = (VG_{p-1} \cup \{0'\}, AG_{p-1} \cup \{(0',0), (0,0')\},0')$, onde $0' \notin VG_{p-1}$.

5. GRAFO LOSANGO ("DIAMOND GRAPH")

de ordem 1: $G_1 = (\{0,v_1,v_2,f\},\{(0,v_1), (0,v_2), (v_1,f), (v_2,f), (f,0)\}, 0)$

de ordem $p > 1$: Sejam $G_A = (VG_A, aG_A, 0_A)$ e $G_B = (VG_B, aG_B, 0_B)$ dois grafos losango de ordem p-1, com $VG_A \cap VG_B = \phi$. $G_p = (VG_A \cup VG_B \cup \{0,f\}, aG_A \cup aG_B \cup \{(0,0_A), (0,0_B), (f_A,f), (f_B,f), (f,0)\}, 0)$, onde $0,f \notin VG_A \cup VG_B$, e $f_A(f_B)$ é o único predecessor de $0_A(0_B)$ em $G_A(G_B)$.

6. GRAFO LEQUE ("FAN GRAPH")

de ordem 1: $G_1 = (\{0,v_1,v_2\}, \{(0,v_1), (0,v_2), (v_1,0), (v_2,0)\}, 0)$

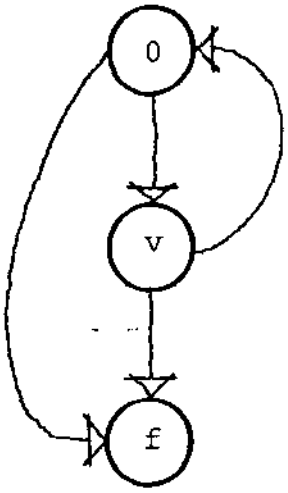
de ordem $p > 1$: Sejam $G_A = (VG_A, aG_B, 0_A)$ e $G_B = (VG_B, aG_B, 0_B)$ dois grafos leque de ordem p-1, tais que $VG_A \cap VG_B = \phi$. $G_p = (VG_A \cup VG_B \cup \{0\}, aG_A \cup aG_B \cup \{(0,0_A),(0,0_B),(0_A,0),(0_B,0)\}, 0)$, onde $0 \notin VG_A \cup VG_B$.

A figura 5.1 apresenta uma interpretação pictórica dos grafos dessas famílias, com ordem p=1 e p=3, enquanto a figura 5.2 mostra os valores dos parâmetros usados no cálculo dos custos dos procedimentos, em função da ordem p.

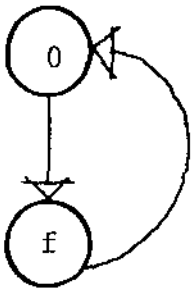
Intuitivamente, a ordem p corresponde ao maior nível de encaixamento de malhas no programa, e ao número de grafos sucessivamente derivados pelo método dos intervalos a partir do grafo original até a obtenção do grafo trivial. Sabe-se que o nível de

GRAFOS DE FAMÍLIAS AUTO-REPLICANTES DE ORDEM p = 1

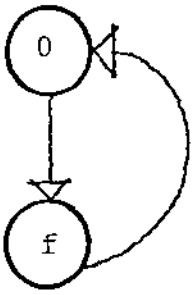
GRAFO ESPIRAL



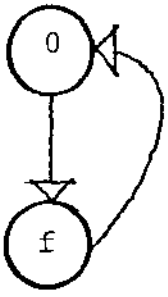
GRAFO CONCHA



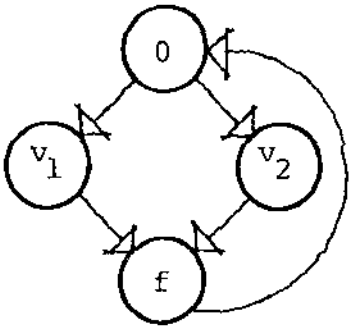
GRAFO BINÁRIO



GRAFO CADEIA-DUPLA



GRAFO LOSANGO



GRAFO LEQUE

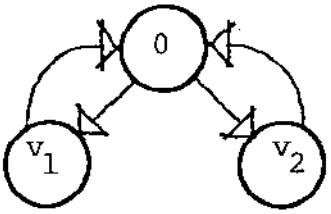
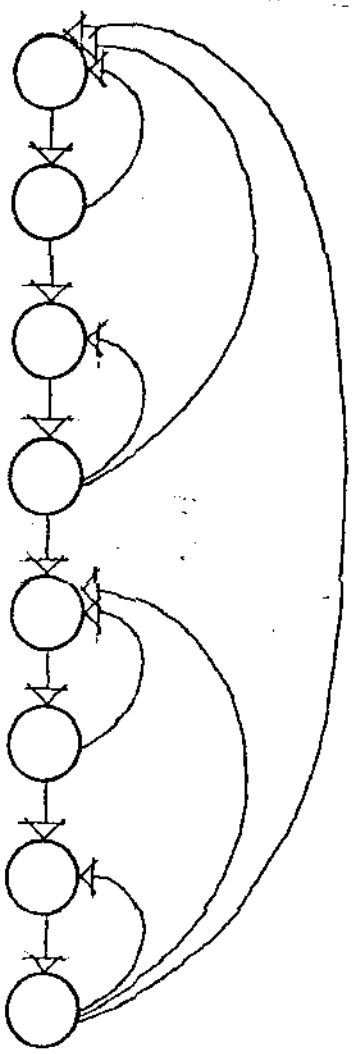


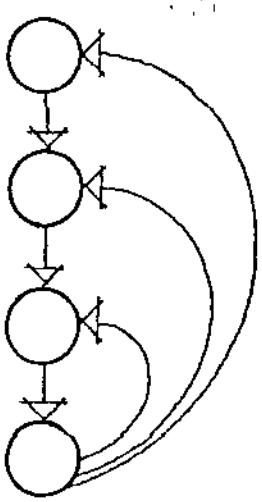
FIGURA 5.1a

GRAFOS DE FAMÍLIAS AUTO-REPLICANTES DE ORDEM p = 3

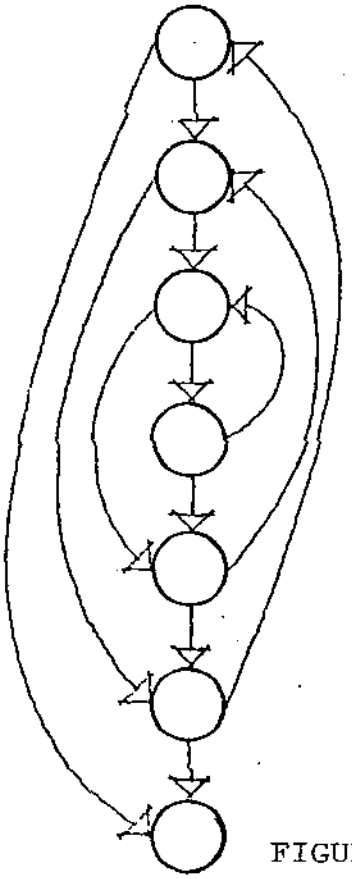
GRAFO BINÁRIO



GRAFO CONCHA



GRAFO ESPIRAL



GRAFO CADEIA-DUPLA

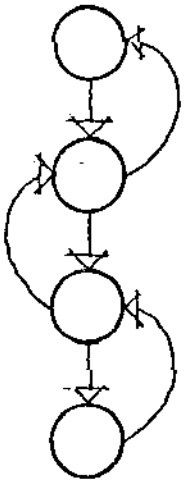
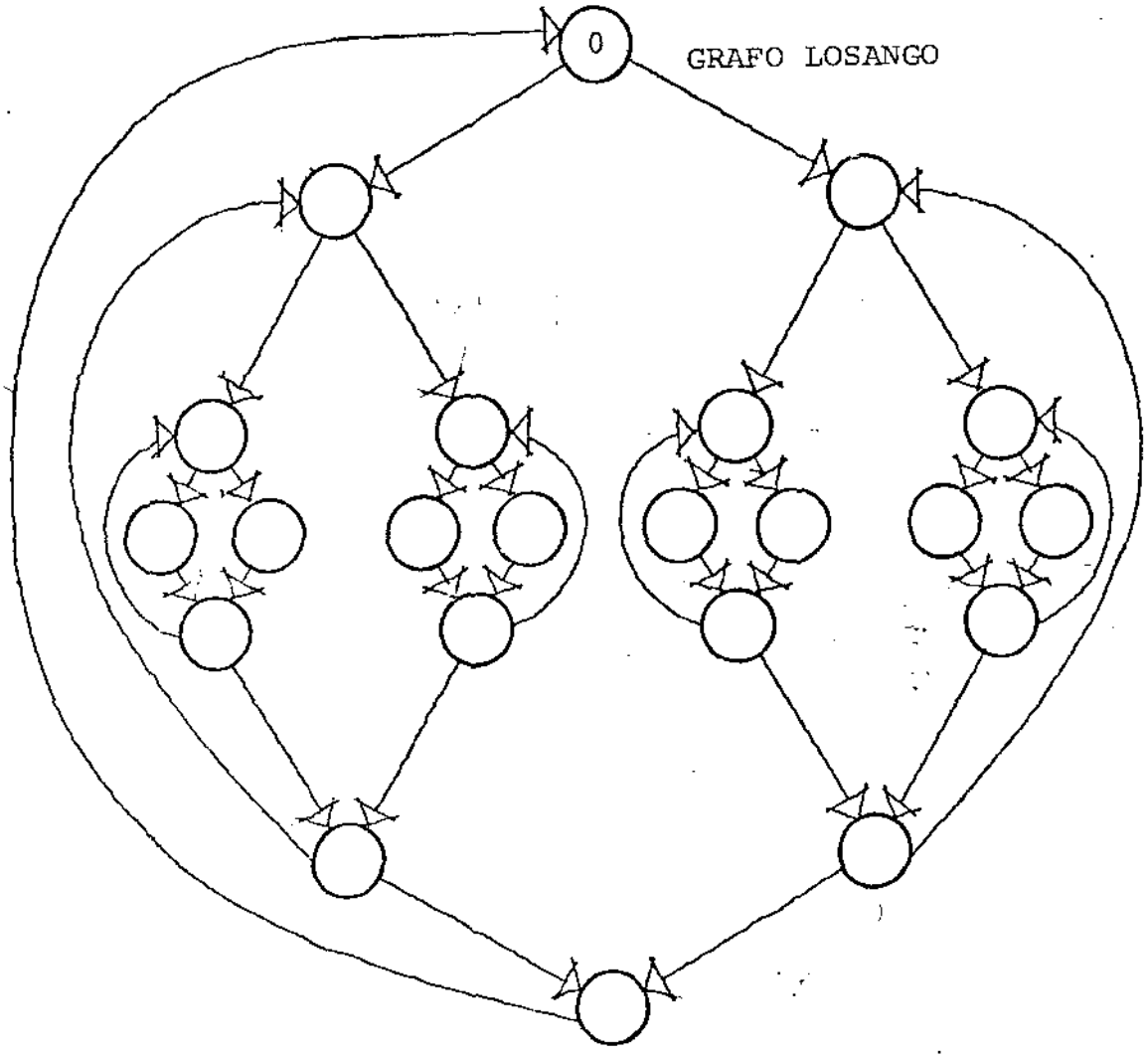


FIGURA 5.1b



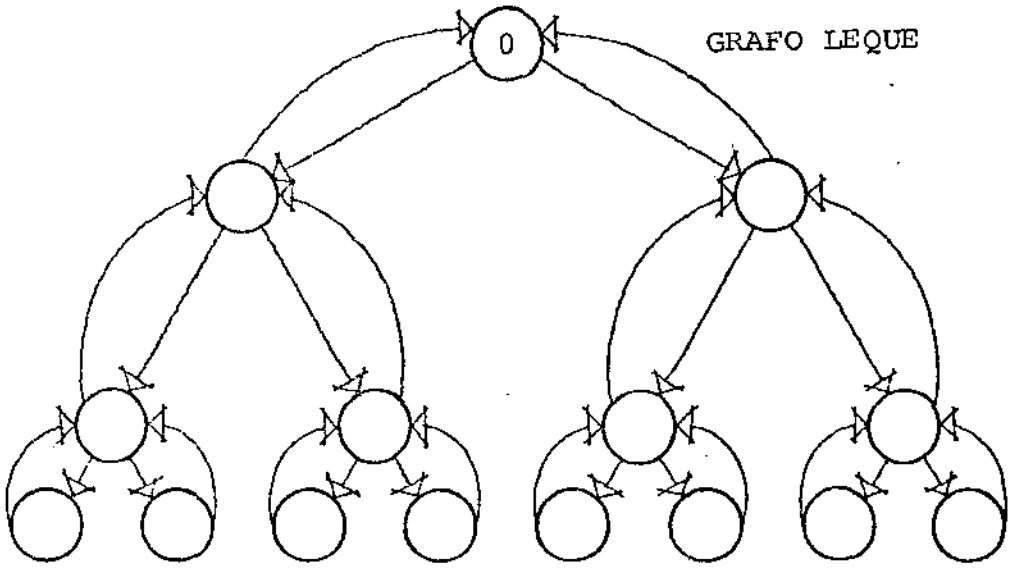


FIGURA 5.1c

VALOR DOS PARÂMETROS PARA OS GRAFOS DE FAMÍLIA AUTO-REPLICANTE

	CONCHA	ESPIRAL	BINÁRIO	CADEIA-DUPLA	LOSANGO	LEQUE
N_v	$p+1$	$2p+1$	2.2^{p-1}	$p+1$	$6.2^{p-1}-2$	$4.2^{p-1}-1$
N_a	$2p$	$4p$	$4.2^{p-1}-2$	$2p$	$10.2^{p-1}-5$	$8.2^{p-1}-4$
N_L	0	0	0	0	0	0
N_R	p	p	2^{p-1}	p	$2.2^{p-1}-1$	$2.2^{p-1}-1$
d	1	p	1	p	1	p
N_1	1	1	p	1	1	2
N_2	1	2	1	1	2	2
N_3	1 SE p=1 2 SE p>1	2 SE p=1 4 SE p>1	$2p-1$	1	3 SE p=1 5 SE p>1	2
N_4	p	p	$2.2^{p-1}-1$	p	$2.2^{p-1}-1$	$4.2^{p-1}-2$
N_5	$3p-1$	$4p-2$	$4.2^{p-1}-2$	$2p$	$10.2^{p-1}-6$	$6.2^{p-1}-3$
N_6	$3p-1$	$5p-3$	$5.2^{p-1}-3$	$2p$	$12.2^{p-1}-7$	$8.2^{p-1}-4$
N_7	p	p	2^{p-1}	p	$4.2^{p-1}-2$	$4.2^{p-1}-2$
N_8	p	$2p+1$	$2.2^{p-1}-1$	p	$6.2^{p-1}-3$	$4.2^{p-1}-2$
N_9	p	$4(p-1)$	2^{p-1}	$2p-1$	2 SE p=1 $6.2^{p-1}-5$ SE p>1	$6.2^{p-1}-4$
N_{10}	$2(p-1)$	$4(p-1)$	$2.2^{p-1}-2$	$2p-2$	$5.2^{p-1}-5$	$4.2^{p-1}-4$
N_{11}	$p-1$	$2(p-1)$	$2^{p-1}-1$	$p-1$	$3.2^{p-1}-3$	$2.2^{p-1}-2$
N_{12}	p	p	2^{p-1}	p	$2.2^{p-1}-1$	$2.2^{p-1}-1$
N_{13}	1	1	2^{p-1}	1	2^{p-1}	2.2^{p-1}
N_{14}	0 SE p=1 1 SE p>1	0 SE p=1 1 SE p>1	$2^{p-1}-1$	0 SE p=1 1 SE p>1	0 SE p=1 2^{p-1} SE p>1	0 SE p=1 2^{p-1} SE p>1

Figura 5.2

encaixamento de malhas num programa tende a ser pequeno, e KNUTH [1971] observou que de 50 programas escritos em FORTRAN, todos tinham no máximo 6 grafos sucessivamente derivados, e em média 2,75. Dessa maneira, valores pequenos de p devem merecer consideração especial.

5.2. COMPARAÇÃO DOS MÉTODOS PARA GRAFOS DE FAMÍLIAS AUTO-REPLICANTES

Os resultados obtidos na comparação do método iterativo com o método dos intervalos em KENNEDY [1976] parecem indicar que o método dos intervalos é mais eficiente. Entretanto, na construção do grafo de fluxo de controle daquela comparação, cada vértice representa um "bloco estendido" que equivale a uma árvore de blocos como definidos neste texto. Acreditamos que o argumento apresentado quanto à redução do tamanho do problema pela utilização de tais blocos estendidos não é válido, uma vez que para diferentes sucessores v de um vértice u no grafo assim formado, X_{uv} é potencialmente diferente, e esses diferentes valores precisam ser calculados, num processo similar a uma redução do grafo original a um grafo derivado, que envolve, entre outras, operações com vetores de bits, que foram desconsideradas. Também as operações de atribuição sobre vetores de bits, que tomam tempo proporcional ao tamanho do universo do problema, não foram avaliadas, assim como as operações que não manipulam diretamente

aqueles vetores. Nessa seção, essa avaliação será feita, considerando que os vértices dos grafos representam blocos, como descritos no capítulo 1.

Utilizando a tabela da figura 5.2 e as equações apresentadas ao final dos capítulos 2, 3 e 4, que definem o número de operações básicas efetuadas pelos procedimentos HU, CA e GW, respectivamente, em função de parâmetros do grafo G dado, pode-se construir tabelas com o número de operações necessárias aos três procedimentos para grafos de cada uma das famílias auto-replicantes, em função apenas da ordem p do grafo G. O exemplo de uma tabela desse tipo para os grafos concha pode ser visto na figura 5.3.

Ainda com uma tabela assim, um confronto direto dos procedimentos é dificultado pela ocorrência de diferentes tipos de operação. Associando-se a cada operação básica um custo relativo (tomado sem rigor, porém com base em um computador típico), é possível obter uma equação única de custo para cada procedimento quando aplicado a um grafo de uma dada família. (Os custos relativos das operações básicas são apresentados no apêndice 2.) Essa equação de custo compõe-se de duas parcelas, uma referente às operações sobre vetores de bits, e outra referente às demais operações. A figura 5.4 mostra as equações obtidas, nas quais o valor de θ representa uma relação entre o tamanho do universo considerado e o tamanho da palavra de memória do computador.

CUSTO DOS ALGORITMOS PARA UM GRAFO CONCHA

	ALGORITMO HU	ALGORITMO CA	ALGORITMO GW	(SE p=1)
μ_{ac}	65p + 7	$54,5p^2$ + 74,5p-2	275p - 81	(+31)
μ_{AC}	33p - 5	$16p^2$ + 12p	31p - 26	(+8)
μ_{ar}	9p + 1	$9,5p^2$ + 11,5p	37p - 11	(+4)
μ_{at}	44p + 8	$31p^2$ + 52p-1	205p - 50	(+21)
μ_{AT}	19p - 1	$12p^2$ + 12p-2	9p - 6	(+2)
μ_{c}	17p + 5	$16p^2$ + 29p+1	68p - 16	(+7)
μ_{C}	3p	——	——	
μ_{cp1}	p + 1	2p	5p	——
μ_{cp2}	——	——	3p - 1	——
μ_{d}	14p + 4	$13p^2$ + 13p+7	52p - 14	(+6)
μ_{ol}	——	p+1	——	
μ_{OL}	12p - 3	$5,5p^2$ + 5,5p	16p - 14	(+4)
μ_{rp}	p + 1	2p	8p - 1	——
μ_{s}	40p + 5	$35p^2$ + 39p-5	114p - 16	(+7)
μ_{sel}	20p - 6	$24p^2$ + 26p	129p - 71	(+25)

FIGURA 5.3

CUSTO DOS PROCEDIMENTOS PARA UM GRAFO DE FAMÍLIA AUTO-REPLICANTE DE ORDEM p	
CONCHA	HU: $(1087p + 244) + (307p - 44)\theta$ CA: $(823p^2 + 1377p-7) + (155,5p^2 + 135,5p-8)\theta$ GW: $(5177p +1344) + (271p - 224)\theta$ $(+(457+68\theta)$ SE $p = 1)$
ESPIRAL	HU: $(352p^2 + 1813p + 163) + (202p^2 + 393p - 25)\theta$ CA: $(1626p^2 + 1641p - 39) + (271p^2 + 289p - 8)\theta$ GW: $(8588p - 3966) + (532p - 428)\theta$ $(+(914+136\theta)$ SE $p = 1)$
BINÁRIO	HU: $(2174.2^{p-1} - 132p - 711) + (614.2^{p-1} - 57p - 294)\theta$ CA: $(5488.2^{p-1} - 574p - 2721) + (1096.2^{p-1} - 265p - 548)\theta$ GW: $(8622.2^{p-1} - 914p - 3418) + (532.2^{p-1} - 136p - 281)\theta$
CADEIA--DUPLA	HU: $(176p^2 + 902p + 253) + (101p^2 + 187p - 25)\theta$ CA: $(813p^2 + 1419p - 39) + (135,5p^2 + 155,5p - 8)\theta$ GW: $(3877p + 413) + (203p - 88)\theta$
LOSANGO	HU: $(5956.2^{p-1} - 2729) + 1728.2^{p-1} - 908)\theta$ CA: $(14550.2^{p-1} - 3269p - 7334) + (2550.2^{p-1} - 622p - 1323)\theta$ GW: $(21430.2^{p-1} - 13422) + (1258.2^{p-1} - 1057)\theta$ $(+(914+136\theta)$ SE $p=1)$
LEQUE	HU: $[(704p + 3644).2^{p-1} - 389p - 1657] + [(404p+824).2^{p-1} - 240p - 475]\theta$ CA: $(10692.2^{p-1} - 2098p - 5385) + (1856.2^{p-1} - 448p - 936)\theta$ GW: $(13758.2^{p-1} - 6915) + (792.2^{p-1} - 552)\theta$

FIGURA 5.4

Nas figuras 5.5 e 5.6 são tabelados os valores das duas parcelas quando $p \leq 6$.

CONSIDERAÇÕES ACERCA DE OPERAÇÕES SOBRE VETORES DE BITS

Com base nas equações de custo dos procedimentos para os grafos de famílias auto-replicantes, pode-se verificar que, para valores grandes de p, o custo de operações sobre vetores de bits é sempre menor para o procedimento GW que para os demais. Para o procedimento HU, esse custo é maior que para o procedimento CA em apenas uma das famílias. Considerando valores de $p \leq 6$ (figura 5.5), todas essas relações se mantêm, com exceção apenas para o grafo espiral com $p = 1$ e para o grafo losango com $p = 1$ e 2, quando o procedimento HU apresenta um custo maior que o procedimento CA.

OPERAÇÕES INDEPENDENTES DO TAMANHO DO UNIVERSO

As operações que não manipulam os vetores de bits são importantes porque são um indicativo de "overhead" necessário para implementar cada algoritmo e espelham de certa forma a complexidade das estruturas de dados.

As equações obtidas para os grafos das famílias auto-replicantes indicam que quando p é grande, o procedimento HU apresenta um custo menor que os demais em três das famílias, tendo

CUSTO RELATIVO A OPERAÇÕES SOBRE VETORES DE BITS (θ)

GRAFO CONCHA

p	HU	CA	GW
1	263	283	115
2	570	885	318
3	877	1.798	589
4	1.184	3.022	860
5	1.491	4.557	1.131
6	1.798	6.403	1.402

GRAFO ESPIRAL

p	HU	CA	GW
1	570	552	240
2	1.569	1.654	636
3	2.972	3.298	1.168
4	4.779	5.484	1.700
5	6.990	8.212	2.232
6	9.605	11.482	2.764

GRAFO BINÁRIO

p	HU	CA	GW
1	263	283	115
2	820	1.114	511
3	1.991	3.041	1.439
4	4.390	7.160	3.431
5	9.245	15.663	7.551
6	19.012	32.934	15.927

GRAFO CADEIA-DUPLA

p	HU	CA	GW
1	263	283	115
2	753	845	318
3	1.445	1.678	521
4	2.339	2.782	724
5	3.435	4.157	927
6	4.733	5.803	1.130

GRAFO LOSANGO

p	HU	CA	GW
1	820	605	337
2	2.548	2.533	1.459
3	6.004	7.011	3.975
4	12.916	16.589	9.007
5	26.740	36.367	19.071
6	54.388	76.545	39.199

GRAFO LEQUE

p	HU	CA	GW
1	513	472	240
2	2.309	1.880	1.032
3	6.949	5.144	2.616
4	18.085	12.120	5.784
5	43.829	26.520	12.120
6	102.021	55.768	24.792

FIGURA 5.5

um custo maior que o procedimento GW em outras duas, e sendo o de maior custo para a família de grafos leque, que é a única família de grafos para a qual o procedimento CA é melhor que os demais. Ainda para valores grandes de p, o procedimento CA tem desempenho pior que os outros dois para três das famílias.

Já para valores de $p \leq 6$ (figura 5.6), o procedimento HU apresenta um custo relativo menor que os outros dois em todas as famílias, e o procedimento CA só apresenta um custo maior que o procedimento GW em três das famílias quando $p = 5$ ou $p = 6$, e para duas delas quando $p = 4$.

5.3. CONSIDERAÇÕES GERAIS

Em resumo, usando como referência os grafos de famílias auto-replicantes, pode-se dizer que o procedimento HU apresenta menor custo de "overhead", e que o procedimento GW é o mais eficiente em termos de operações sobre vetores de bits. Cabe lembrar aqui, entretanto, que esses resultados não podem ser estendidos diretamente para um grafo qualquer.

Como observações de caráter geral, pode-se dizer que na avaliação do método iterativo foi utilizada a pior situação de propagação de dados no grafo. Esse limite é, porém, atingível. No caso dos intervalos foi efetuada uma estimativa real. Em ambos, o pior caso toma tempo quadrático no tamanho do grafo.

CUSTO RELATIVO A OPERAÇÕES INDEPENDENTES DO TAMANHO DO UNIVERSO

GRAFO CONCHA			
p	HU	CA	GW
1	1.331	2.193	4.290
2	2.418	6.039	9.010
3	3.505	11.531	14.187
4	4.592	18.669	19.364
5	5.679	27.453	24.541
6	6.766	37.883	29.718

GRAFO ESPIRAL			
p	HU	CA	GW
1	2.328	3.228	5.536
2	5.197	9.747	13.210
3	8.770	19.518	21.798
4	13.047	32.541	30.386
5	18.028	48.816	38.974
6	23.713	68.343	47.562

GRAFO BINÁRIO			
p	HU	CA	GW
1	1.331	2.193	4.290
2	3.373	7.107	11.998
3	7.589	17.509	28.328
4	16.153	38.887	61.902
5	33.413	82.217	129.964
6	68.065	169.451	267.002

GRAFO CADEIA-DUPLA			
p	HU	CA	GW
1	1.331	2.193	4.290
2	2.761	6.051	8.167
3	4.543	11.535	12.044
4	6.677	18.645	15.921
5	9.163	27.381	19.798
6	12.001	37.743	23.675

GRAFO LOSANGO			
p	HU	CA	GW
1	3.227	3.947	8.922
2	9.183	15.228	29.438
3	21.095	41.059	72.298
4	44.919	95.990	158.018
5	92.567	209.121	329.458
6	187.863	438.652	672.338

GRAFO LEQUE			
p	HU	CA	GW
1	2.302	3.209	6.843
2	7.669	11.803	20.601
3	20.200	31.089	48.117
4	48.467	71.759	103.149
5	111.022	155.197	213.213
6	247.785	324.171	433.341

FIGURA 5.6

Para o número de transformações T2 executadas na redução das regiões fortemente conexas do procedimento GW, que foi descrito neste trabalho como $(N_6 - N_7)$, existem dois limites superiores apresentados em GRAHAM [1976], que em geral são bem maiores que os números reais: $0(N_a \log N_a)$ e $N_a + \sum_{u \in VG} f(u,G) - \sum_{r \in RG} g(r,G)$, onde $VR(u,G)$ é conjunto dos vértices da região R com raiz u em G, $f(u,G) = |\{(v,w) \in aG \mid v \in VR(u,G) \text{ e } w \notin VR(u,G)\}|$, $RG = \{v \in VG \mid$ existe uma aresta de retorno (u,v) em $AG\}$, e $g(r,G) = |\{(r,v) \in aG\}|$.

Intuitivamente, a função $f(u,G)$ conta as arestas cuja origem pertence aos vértices da região com raiz u em G, mas o destino não está nesse conjunto, o que corresponde a possibilidades de saída das malhas do programa, e a função $g(r,G)$ conta os sucessores de um vértice r que é raiz de região.

• COMENTÁRIOS FINAIS

Certamente cada um dos métodos discutidos tem importância considerável. O algoritmo HU pela sua extrema simplicidade e também pela generalidade, manipulando grafos irredutíveis sem esforços adicionais (requeridos pelos outros métodos). O algoritmo CA representa um primeiro avanço no sentido de orientar a análise pela forma do grafo, e dividir o problema em outros menores. E o algoritmo GW refina a consideração à forma, procurando

adaptar o trajeto da análise a partir das malhas mais internas até as mais externas, sem no entanto tomar tempo quadrático (o algoritmo GW para programas sem saídas múltiplas de malhas é linear).

Para procedimentos escritos em uma linguagem estruturada, essa "consideração à forma" pode ser feita de maneira natural, pela análise estruturada de fluxo de dados, que explora as estruturas de controle da linguagem (KOWALTOWSKI [1984], ROSEN [1977], SHARIR [1980]).

APÊNDICE 1

TABELA DE PARÂMETROS DO GRAFO

PARÂMETRO	NÚMERO (DE)
N_v	vértices do grafo .
N_a	arestas do grafo .
N_L	laços do grafo .
N_R	vértices destinos de arestas de retorno (número de regiões).
d	máximo de arestas de retorno num caminho do grafo.
N_1	predecessores de n_o .
N_2	sucessores de n_o .
N_3	sucessores v de n_o, $v \neq n_o$, no grafo acíclico resultante de todas as reduções de regiões no grafo G.
N_4	arestas de retorno.
N_5	total de vértices em todas as regiões.
N_6	total de arestas em todas as regiões menos laços que não estão nas raízes.

N_7 arestas que saem de todas as raízes para vértices na sua própria região.

N_8 arestas no grafo acíclico resultante de todas as reduções de regiões no grafo G, que não são laços.

N_9 sucessores de raízes de intervalos, incluindo n_o.

N_{10} arestas entre vértices de intervalos distintos.

N_{11} predecessores de cada raiz $r \neq n_o$ no "primeiro" intervalo predecessor de $I(r)$.

N_{12} vértices destino de arestas de retorno ou laços.

N_{13} predecessores de cada raiz $r \neq n_o$ no seu próprio intervalo, incluindo laços.

N_{14} raízes de intervalo $r \neq n_o$ que têm predecessores no seu próprio intervalo.

APÊNDICE 2

AS OPERAÇÕES BÁSICAS E SEUS CUSTOS RELATIVOS

μ_{ac} acesso à memória, ou seja uma transferência da memória para um registrador. Custo relativo 5.

μ_{ar} aritmética, como soma ou subtração, sem incluir acessos à memória e deixando o resultado num registrador. Custo relativo 5.

μ_{at} atribuição, ou transferência de um registrador para a memória. Custo relativo 4.

μ_{av} avanço em lista, que corresponde a p := p·próximo. Seu custo nas somas verticais é substituído por $(\mu_{ac}+\mu_{at}+\mu_{sel})$.

μ_{c} comparação do tipo =, $\neq$, >, <, $\geq$, $\leq$, sem incluir acessos à memória. Custo relativo 2.

μ_{cpi} chamada de procedimento com i parâmetros, incluindo salvamento de contexto (registradores de índice e de base, acumulador e "flags") e desvio, além da passagem dos parâmetros. Custo relativo $\mu_{cp1} = 80$, $\mu_{cp2} = 90$.

μ_{d} desvio na seqüência de execução das instruções. Custo relativo 4.

μ_{ol} operação lógica do tipo $\wedge$ ou $\vee$. Custo relativo 2.

μ_{rp} retorno de procedimento, incluindo a restauração do contexto e desvio. Custo relativo 71.

μ_{s} subscrição, incluindo o acesso ao valor do subscrito além da alteração do endereço propriamente dito. Custo relativo 5.

μ_{sel} seleção de campo, incluindo o acesso ao valor do deslocamento referente ao campo selecionado, além da alteração de endereço propriamente dita. Custo relativo 5.

As operações sobre vetores de bits ($\mu_{AC}, \mu_{AT}, \mu_{C}, \mu_{OL}$) têm, em geral, custo relativo igual ao custo das operações sobre variáveis simples, multiplicada pelo número de palavras de memória ocupadas pelo universo considerado, que no texto foi representado por θ. Exceção é feita à operação μ_{OL}, que por ser implementada através de uma instrução diferente de μ_{ol}, tem custo relativo 5θ.

APÊNDICE 3

IMPLEMENTAÇÕES TÍPICAS E CUSTOS DE PROCEDIMENTOS SIMPLES

Em todos os procedimentos, supõe-se que foram declaradas estruturas de dados convenientes.

Um trecho de procedimento usado por todas as rotinas de inclusão em lista, denominado GERA, e responsável pela alocação de espaço, pode ser traduzido por:

```
procedimento Gera (p)
  se (topo + tam) ≤ lim
  então  p    := topo
         topo := topo + tam
  senão  ERRO
```

que, considerando a não ocorrência de erro, apresenta um custo de: $(6\mu_{ac} + 2\mu_{ar} + 2\mu_{at} + \mu_c + \mu_d)$.

• PROCEDIMENTO USADO NO CAPÍTULO II

```
procedimento  Inclui(lista, v)

    Gera (p)
    p·vértice  := v
    p·próximo  := lista
    lista      := p
```

Custo associado: $(9\mu_{ac} + 2\mu_{ar} + 5\mu_{at} + \mu_c + \mu_d + 2\mu_{sel})$

• PROCEDIMENTOS USADOS NO CAPÍTULO III

```
procedimento  Inclui-Vértice(lista, v)

    Gera (p)
    p·vértice  := v
    p·próximo  := lista
    lista      := p
```

Custo associado: $(9\mu_{ac} + 2\mu_{ar} + 5\mu_{at} + \mu_c + \mu_d + 2\mu_{sel})$

```
procedimento  Retira-Vértice (lista, v)

    v      := lista·vértice
    lista  := lista·próximo
```

Custo associado: $(2\mu_{ac} + 2\mu_{at} + 2\mu_{sel})$

procedimento Inclui-Aresta (lista, v, κ, σ)

```
Gera (p)
p.destino := v
p.k       := κ
p.c       := σ
p.próximo := lista
lista     := p
```

Custo associado: $(9\mu_{ac} + 2\mu_{AC} + 2\mu_{ar} + 5\mu_{at} + 2\mu_{AT} + \mu_{c} + \mu_{d} + 4\mu_{sel})$

procedimento Retira-Aresta (lista, (v, κ, σ))

```
v     := lista.destino
κ     := lista.k
σ     := lista.c
lista := lista.próximo
```

Custo associado: $(2\mu_{ac} + 2\mu_{AC} + 2\mu_{at} + 2\mu_{AT} + 4\mu_{sel})$

• PROCEDIMENTOS USADOS NO CAPÍTULO IV

procedimento Inclui(lista,(v, loc))

```
Gera (p)
p·vértice  := v
p.localiza := loc
p·próximo  := lista
lista      := p
```

Custo associado: $(10\mu_{ac} + 2\mu_{ar} + 6\mu_{at} + \mu_{c} + \mu_{d} + 3\mu_{sel})$

procedimento Inclui-G(lista, (v, C, K))

```
Gera (p)
p.destino    := v
p·c          := C
p·k          := K
lista·predec := p
p·próximo    := lista
p·predec     := nil
lista        := p
```

Custo associado: $(10\mu_{ac} + 2\mu_{AC} + 2\mu_{ar} + 5\mu_{at} + 2\mu_{AT} + \mu_{c} + \mu_{d} + 6\mu_{sel})$

```
procedimento  Elimina-Aresta(loc)

     se loc.predec  = nil

     então ⌈Δ[v]    := loc.próximo
           |se loc.próximo ≠ nil
           |então  loc.próximo.predec := nil

     senão ⌈loc.predec.próximo := loc.próximo
           |se loc.próximo ≠ nil
           |então  loc.próximo.predec := loc.predec /* PIOR CASO */
```

Custo associado: $(4\mu_{ac} + 2\mu_{at} + 2\mu_{c} + \mu_{d} + 8\mu_{sel})$

REFERÊNCIAS BIBLIOGRÁFICAS

AHO [1977]

ALFRED V. AHO & JEFFREY D. ULLMAN, *Principles of Compiler Design*, Addison-Wesley.

COHEN [1982]

JACQUES COHEN, "Computer-Assisted Microanalysis of Programs", *Comm. ACM* 25, 10, 724-733.

FOSDICK [1976]

L.D. FOSDICK & L.J. OSTERWEIL, "Data Flow Analysis in Software Reliability", *Computing Surveys*, 8, 3, 305-330.

GRAHAM [1976]

SUSAN GRAHAM & MARK WEGMAN, "A Fast and Usually Linear Algorithm for Global Flow Analysis", *Journal of the ACM* 23, 1, 172-202.

HECHT [1974]

MATTHEW S. HECHT & JEFFREY D. ULLMAN, "Characterizations of Reducible Flow Graphs", *Journal of the ACM* 21, 3, 367-375.

HECHT [1975]

MATTHEW S. HECHT & JEFFREY D. ULLMAN, "A Simple Algorithm for Global Data Flow Analysis Problems", *SIAM J. Computing* 4, 4, 519-532.

HECHT [1977]

MATTHEW S. HECHT, *Flow Analysis of Computer Programs*, North-Holland.

HENDERSON [1976]

PETER HENDERSON, "Flowcharts, Control-Structures and Flow Graph Reducibility", *State University of New York at Stony Brook, Technical Report* #50.

KENNEDY [1976]

KEN KENNEDY, "A Comparison of Two Algorithms for Global Flow Analysis", *SIAM J. Computing* 5, 1, 158-180.

KNUTH [1971]

DONALD E. KNUTH, "An Empirical Study of FORTRAN Programs", *Software Practice and Experience* 1, 12, 105-134.

KNUTH [1973]

DONALD E. KNUTH, *The Art of Computer Programming*, vol. 1, Addison-Wesley.

KOWALTOWSKI [1984]

TOMASZ KOWALTOWSKI, "Implementation of Structured Data Flow Analysis", *Universidade Estadual de Campinas, IMECC, Relatório Interno Nº 258.*

ROSEN [1977]

BARRY K. ROSEN, "High-Level Data Flow Analysis", *Comm. ACM* 20, 10, 712-724.

SHARIR [1980]

MICHA SHARIR, "Structutal Analysis: A New Approach to Flow Analysis in Optimizing Compilers", *Computer Languages* 5, 3/4, 141--153.