Reconhecimento e Traçado de Grafos Planares

Maria Teresa Marques Leite Baldas

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

Aprovado por:

Lilian Markenzon, D.Sc.
Presidente

Jayme Luiz Szwarcfiter, Ph.D.

Paulo Oswaldo Boaventura Netto, Dr.Ing.

Paulo Roberto Oliveira, Dr.Ing.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1995

BALDAS, MARIA TERESA MARQUES LEITE

Reconhecimento e Traçado de Grafos Planares. Rio de Janeiro, 1995.

viii, 127 p., 29,7cm (COPPE / UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1995)

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1 - Traçado de Grafos Planares 2 - Algoritmos

I. COPPE/UFRJ II. Título (Série)

A Elisa,
minha filha.

Agradecimentos

À Lilian Markenzon, pela sua paciência em me deixar procurar meus caminhos; pela sua amizade e valiosa orientação.

A Paulo Roberto Oliveira, pelo seu apoio como co-orientador durante todo o desenvolvimento desta tese.

À Ana Sueli, minha irmã, pelo seu estímulo e constante participação na finalização deste trabalho.

A Samuel Jurkiewicz, pelo seu auxílio inicial, sem o qual eu teria desistido nos primeiros passos.

À Maria Aparecida, minha amiga, e a todos aqueles que acreditaram que eu seria capaz de concretizar este trabalho.

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

Reconhecimento e Traçado de Grafos Planares

Maria Teresa Marques Leite Baldas

Março, 1995

Orientador : Lilian Markenzon
Programa : Engenharia de Sistemas e Computação

Resumo

Este trabalho estuda o traçado automático de grafos planares. Inicialmente são descritos dois algoritmos de reconhecimento de planaridade. A seguir, com base nas árvores-PQ, são estudados dois algoritmos para embutir um grafo no plano, o que consiste em reconstruir sua listas de adjacências de forma a poder desenhá-lo sem cruzamento de arestas.

Finalmente são apresentados os algoritmos de traçado: o convexo, que não pode ser aplicado a qualquer grafo planar, e o agradável, que não tem tal restrição. Um algoritmo de teste de convexidade também consta deste estudo, para selecionar os grafos planares aos quais o traçado convexo pode ser aplicado.

São ainda comentados outros resultados: o traçado de grafos planares em grades, muito utilizado em desenho automático de grandes circuitos; um traçado de linhas retas, onde uma face triangular é escolhida como ciclo externo do grafo e o reconhecimento de grafos periplanares biconexos com desenho em um polígono regular.

A maioria dos algoritmos descritos neste trabalho tem tempo de execução e área de armazenamento lineares relativamente ao tamanho do grafo.

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

The Recognition and Drawing of Planar Graphs

Maria Teresa Marques Leite Baldas

March, 1995

Thesis Supervisor : Lilian Markenzon
Department : Computing and System Engineering

Abstract

This work studies the automatic drawing of planar graphs. Initially, two planarity testing algorithms are described. Then, based on the PQ-trees, two algorithms for embedding planar graphs in the plane are presented, constructing adjacency lists for the graph, such that it can be drawn without edge crossing.

Finally the drawing algorithms are presented: a convex one, that cannot be applied to every planar graph; and a *nice* drawing, that has no such constraint. A convex testing algorithm is also described, to determine which planar graphs can be drawn convex.

Other results commented: drawing planar graphs on a grid, that has many applications such as large circuits automatic design; a straight-line drawing, where a triangle is chosen as the boundary of the outer face and how to obtain a regular polygon drawing of a biconnected outerplanar graph, previously recognized.

Most of the algorithms described in this work run in both time and space which are linear in the size of the graph.

Índice

Capítulo 1

Introdução

Grafos têm sido utilizados na modelagem dos mais diversos problemas. Um conjunto de vértices e outro de arestas especifica perfeitamente um grafo considerado. No entanto, eles fornecem pouca informação estrutural sobre o grafo, o que torna indispensável um tratamento visual para o mesmo. Por esta razão, pode-se observar, nos últimos anos, o desenvolvimento de sistemas gráficos oferecendo recursos visuais tais como o traçado automático.

O traçado de um grafo submetido a imposições como a ausência de interseções nas arestas ou a igualdade de dimensões das mesmas, pode tornar o problema geral de *desenhar um grafo no plano de tal forma que todas as restrições sejam satisfeitas* de difícil solução.

No estudo teórico de problemas de grafos, algoritmos eficientes foram desenvolvidos solucionando alguns destes problemas e mostrando que outros são computacionalmente intratáveis. Pesquisas recentes têm se concentrado em estabelecer limites superiores e inferiores para os casos sem tratamento determinístico ou em achar valores exatos para grafos especiais [11], como, por exemplo, os planares.

1.1- Grafos Planares

Um grafo abstrato G=(V,E) consiste de um conjunto **V** de vértices e um conjunto **E**, contido em VxV, de arestas. A planaridade de um grafo G é uma propriedade de G que permite que ele seja desenhado no plano sem que suas arestas se cruzem, exceto nos vértices.

O objetivo deste trabalho é estudar o traçado de grafos planares que, além de ter muitos problemas resolvidos satisfatoriamente na teoria de grafos, tem aplicações práticas diversas, como desenho automático de circuitos VLSI (very large scale integration),

plantas de circuitos impressos, redes de comunicação por luz ou microondas, segmentação de programas em informática, isomorfismo de estruturas moleculares, interfaces gráficas, entre outras.

O problema de traçar um grafo planar pode ser decomposto em três subproblemas. Cada um deles tem importantes contribuições e resultados, que são revistos no decorrer deste trabalho. Os passos da apresentação são os seguintes:

1- reconhecer a planaridade do grafo em questão;

2- construir sua embutidura planar e

3- traçar o grafo no plano.

Reconhecer um grafo quanto à sua planaridade consiste em submetê-lo a um algoritmo de teste, que verifique se ele tem possibilidade de ser desenhado no plano sem que suas arestas se interceptem. Para introduzir um grafo em um programa de computador, são usadas estruturas de dados para representá-lo; entre as mais comuns, tem-se a matriz de adjacências e as listas de adjacências dos vértices [2].

Os algoritmos estudados devem-se a Hopcroft-Tarjan [1] e a Booth-Lueker [4] e estão no capítulo 2, precedidos pela discussão das técnicas empregadas nos mesmos: a exploração sistemática de grafos por busca em profundidade, a rotulação-st dos vértices de um grafo e a estrutura de dados árvore-PQ, definida por [4] e utilizada no seu trabalho, bem como em outros posteriores.

O segundo passo consiste em construir a embutidura de um grafo, reconhecido como planar. A embutidura, também chamada representação planar, refaz as listas de adjacências dos vértices do grafo, de tal forma que as arestas possam ser traçadas sem cruzamento, determinando, por exemplo, no sentido dos ponteiros do relógio, a ordem na qual os vizinhos de cada vértice devem aparecer no desenho. Ela já pode ser considerada um *desenho* planar do grafo, apesar de não existir qualquer critério imposto à forma como são traçadas as arestas ou ao posicionamento dos vértices.

São discutidos, no capítulo 3, os algoritmos de Chiba-Nishizeki-Abe-Ozawa [8] e de Jayakumar-Thulasiraman-Swamy [12], que se baseiam, ambos, na estrutura de dados árvore-PQ para determinar as novas listas de adjacências dos vértices.

Finalmente, o traçado de grafos planares é apresentado nos capítulos 4 e 5. O primeiro tipo de desenho discutido é o traçado convexo, cujos algoritmos constam do trabalho de Chiba-Nishizeki [15]; este, no entanto, não se aplica a todos os grafos planares; um algoritmo de teste é fornecido para verificar esta possibilidade. A Chiba-Onoguchi-Nishizeki [9] deve-se o desenvolvimento do algoritmo de traçado agradável, que abrange o desenho de qualquer grafo planar. No capítulo 5 é abordado outro tipo de traçado de grafos,

o traçado em grades, tema dos trabalhos de Storer [13], Tamassia-Tollis [10] e Tamassia-Tollis-Vitter [16]. Finalmente é apresentado o traçado poligonal regular para os grafos periplanares, caso particular dos grafos planares, com base em Mitchell [17] e Manning-Atallah [18].

Todos os algoritmos de traçado apresentados pressupõem que a planaridade do grafo foi reconhecida; eles utilizam e preservam a embutidura planar construída pelos algoritmos do segundo passo.

1.2- Critérios de Traçado

Um grafo abstrato pode ser desenhado de várias maneiras diferentes, todas igualmente *corretas*. Certamente algumas são melhores do que outras, no sentido de exibir claramente propriedades estruturais do grafo, tais como planaridade, simetria, biconectividade e outras, ou no sentido da sua legibilidade, isto é, a capacidade de transmitir o significado do diagrama rápida e claramente. São, então, definidos critérios de traçado, baseados em necessidades teóricas ou em aplicações práticas, aos quais o desenho do grafo deve satisfazer.

Um dos resultados da teoria dos grafos, obtido para grafos planares, é que todos admitem um desenho planar de linhas retas. Isto define um dos critérios mais utilizados no traçado de grafos planares, segundo o qual as arestas devem ser desenhadas como segmentos de reta; em decorrência deste critério, tem-se que o desenho fica inteiramente determinado pelo posicionamento dos vértices. O traçado convexo utiliza o critério de linhas retas e o incrementa, desenhando os limites de todas as faces como polígonos convexos. Também com base no mesmo critério, o algoritmo de traçado agradável tem como objetivo desenhar as componentes 3-conexas do grafo planar dado convexamente; o desenho final é freqüentemente mais bonito e claro do que o anterior. Ainda assim, tais desenhos são criticados por Fraysseix-Pach-Pollach [7] por sua tendência em agrupar os vértices do grafo, no sentido de que a razão da menor para a maior distância é muito pequena. O algoritmo proposto por [7] satisfaz o critério de linhas retas para as arestas, desenhando o ciclo externo do grafo como um triângulo de coordenadas (0,0), (2n-4,0) e (n-2,n-2), onde $\mathbf{n}=|V|$, e posicionando os demais vértices no interior deste triângulo, todos com coordenadas inteiras.

O tipo de traçado em grades introduz critérios diferentes, voltados para a sua utilização em aplicações práticas, onde desenhos automáticos de circuitos são muito

importantes; neste caso, eles podem também ser considerados medidas de qualidade. Estes critérios são a área do retângulo que contém o grafo, o comprimento total das arestas e o número de *curvas* existentes ao longo das arestas. Obviamente, neste caso, o critério de linhas retas é abandonado, pois as arestas devem *seguir* a grade, podendo ser flexionadas para atingir os vértices ao qual são incidentes. Para um traçado em grade, a restrição do grau máximo de G ser menor ou igual a 4 é necessária.

Um último critério - o da simetria -, aliado ao critério de linhas retas é escolhido, abrangendo a família de grafos periplanares e obtendo, para os grafos periplanares biconexos, desenhos poligonais regulares.

1.3- Complexidade dos Problemas

O problema de desenhar um grafo no plano sem cruzamento de arestas é resolvido na teoria de grafos por mais de um algoritmo, de tempo de execução e área de armazenamento lineares, que reconhece um grafo planar, o qual pode satisfazer tal restrição. No entanto, o problema geral de desenhar um grafo não planar de tal forma que o número total de cruzamentos de suas arestas seja menor ou igual a um dado inteiro **k**, é NP-completo [11].

Mesmo o grafo sendo planar, se à restrição de não cruzamento de arestas se acrescentar a de arestas com pesos ou dimensões iguais, tem-se um problema classificado como NP-difícil. Alterando esta segunda restrição para *o comprimento de cada aresta deve corresponder ao seu peso*, o problema, muito importante quando se desejam desenhos em escala, permanece na mesma classificação. Estes resultados valem para grafos 2-conexos e 3-conexos [11].

Nos traçados de grafos planares em grades, os problemas de minimização da área do desenho e do comprimento total das arestas são, na maioria dos casos, NP-difíceis [13]. Em contraste, um desenho em grade com mínimo número de curvas pode ser construído em tempo polinomial [16].

No decorrer deste trabalho, as discussões se restringem aos grafos biconexos, não direcionados e simples. De maneira geral, seus resultados podem vir a ser expandidos para os que não se enquadram nesta hipótese, seja adicionando arestas, ignorando direcionamentos ou eliminando multi-arestas e laços (aresta e=(v,v)). Os teoremas estabelecidos sem demonstração têm as referências devidamente fornecidas.

Capítulo 2

Reconhecimento da Planaridade de um Grafo

2.1- Introdução

O objetivo deste trabalho é estudar o traçado de grafos planares. O primeiro passo é o **reconhecimento da planaridade** do grafo; o segundo, chamado **embutidura** ou **representação planar**, é posicionar seus vértices no plano, de forma a poder traçar suas arestas sem que estas se interceptem; o passo final é **traçar** o grafo, usando critérios estéticos específicos.

O interesse em obter o traçado de um grafo no plano, sem cruzamento de arestas, se deve a uma vasta utilidade prática, como por exemplo desenho de circuitos VLSI (Very Large Scale Integration) [11,7,10], plantas de circuitos impressos [12], segmentação de programas em informática [2], análise de redes em engenharia elétrica [2], interfaces gráficas [10], isomorfismo de estruturas químicas [1,8], entre outras.

Um grafo G=(V,E), onde V é o conjunto de vértices de G, E o de arestas, n=|V| e m = |E|, é dito **planar** [2] se existe um mapeamento dos seus vértices e arestas num plano, de tal forma que:

(i) cada vértice **v** é transformado num ponto distinto **v'** no plano;

(ii) cada aresta (v,w) pode ser transformada numa curva simples com extremos **v'** e **w'**, de tal forma que

(iii) as arestas transformadas se encontram somente nos extremos comuns.

(DEF.2.1)

O reconhecimento da planaridade de um grafo está, pela definição acima, fortemente ligado à sua representação planar. No entanto, apesar desta interligação, a primeira parte do problema -o reconhecimento da planaridade- foi resolvida bem antes da segunda -a embutidura do grafo.

A primeira tentativa de estabelecer uma condição de planaridade para um grafo data de 1736, e é resultado da fórmula poliédrica de Euler (para um poliedro esférico com **n** vértices, **m** arestas e **f** faces, tem-se **n -m +f =2**) e é somente uma condição necessária:

.se G é planar, então $\mathbf{m} \leq 3\mathbf{n} - 6$ ou, se $\mathbf{m} > 3\mathbf{n} - 6$ então G é não planar.

Só em 1930, Kuratowski formulou e provou a seguinte condição necessária e suficiente para determinar a planaridade de um grafo:

.G é planar se e só se não contém nenhum subgrafo homeomorfo a $K_{3,3}$ ou a K_5 [1].

Embora corretamente formulada, esta caracterização nunca foi implementada com eficiência como teste de planaridade. Em 1963, Tutte usou esta idéia num algoritmo de traçado, obtendo um tempo de complexidade $O(\mathbf{n}^3)$ [11].

2.1.1- As Abordagens do Teste de Planaridade

O problema de testar a planaridade de um grafo foi abordado de duas maneiras distintas no desenvolvimento de algoritmos:

(i) a **adição de caminhos** - quando a cada passo do algoritmo, uma aresta ou um caminho é considerado no teste;

(ii) a **adição de vértices** - quando as considerações recaem sobre os vértices.

Os algoritmos de reconhecimento de planaridade analisados neste capítulo são de Hopcroft-Tarjan [1] e de Booth-Lueker [4]. Esta escolha possui algumas justificativas. Inicialmente ambos têm complexidade linear; além disso, [1] foi o primeiro com este tempo. Depois, o interesse especial no algoritmo de [2] se prende à utilização de uma estrutura de dados especial, muito utilizada posteriormente em algoritmos de embutidura de grafos planares.

A abordagem de Hopcroft-Tarjan foi a de **adição de caminhos**; sua publicação data de 1974. A base do algoritmo tem a seguinte história, segundo [1]:

(i) em 1961, Auslander-Parter desenvolveram um algoritmo recursivo que, achando um ciclo num grafo, o removia, partindo o grafo resultante em diversos pedaços. O procedimento tentava, então, reunir cada um dos pedaços juntamente com o ciclo inicial sem cruzamento de arestas. Este algoritmo continha um erro, que podia tornar seu tempo infinito;

(ii) em 1963, Goldstein formulou corretamente o algoritmo anterior, usando iteração ao invés de recursividade;

(iii) em 1969, Shirey, usando uma estrutura de listas para representação do grafo, implementou o mesmo método, obtendo um tempo limitado superiormente em $O(n^3)$;

(iv) em 1971, os próprios Hopcroft-Tarjan conseguiram um tempo limitado em O(n *log* n) para o algoritmo de Goldstein, usando a técnica da *busca em profundidade* para exploração das arestas do grafo dado.

Embora Hopcroft-Tarjan tenham afirmado que seu algoritmo poderia ser facilmente adaptado para obter a embutidura de um grafo, uma vez reconhecida sua planaridade, nenhuma implementação foi obtida com complexidade linear.

A abordagem de **adição de vértices** teve sucesso com Booth-Lueker, em 1976. Neste ano eles publicaram um algoritmo de reconhecimento de planaridade linear em relação ao número de vértices do grafo, usando o método originalmente proposto por Lempel-Even-Cederbaum em 1967, e uma nova estrutura de dados, criada por eles, chamada árvore-PQ.

Este algoritmo, apesar de não ter sido o primeiro publicado com tempo linear, foi o que teve desdobramentos no problema da embutidura de grafos:

(i) em 1985, Chiba, Nishizeki, Abe e Ozawa [8] apresentaram um algoritmo de embutidura de grafos planares com tempo linear;

(ii) em 1988, Jayakumar, Thulasiraman e Swamy [12] descreveram um procedimento sistemático para obter uma representação planar de um grafo, no qual foram calculadas coordenadas para os vértices, também com tempo linear.

Pode-se mencionar ainda uma **terceira abordagem** para o teste de planaridade, a de Fraysseix-Rosenstiehl [6], publicada em 1982 e aqui apresentada sem detalhamentos.

A técnica utilizada é uma busca em profundidade, que constrói inicialmente uma árvore geradora T para o grafo dado; a idéia é verificar, para pares de subárvores com um vértice inicial comum e arestas de retorno que se interceptam, se é possível comutá-las relativamente a T, eliminando os cruzamentos. Para atingir seu objetivo, [6] analisa, dois a dois, os ângulos que as arestas de retorno fazem com as arestas de T em dois vértices de entrada consecutivos; estes ângulos, aos pares, são agrupados nos conjuntos S e D, conforme estejam, respectivamente, do mesmo lado (S) ou em lados diferentes (D) de T.

O teorema principal de [6], que testa a planaridade do grafo dado, se baseia nestes pares de ângulos para ser formulado; um algoritmo não é explicitado por [6], mas este conclui que um tempo linear de computação pode ser atingido por um procedimento que determine tais ângulos.

2.2- Técnicas Empregadas nos Algoritmos

2.2.1- A Busca em Profundidade

Uma busca [14] é uma técnica utilizada em algoritmos de grafos para solucionar o problema da sua exploração sistemática, isto é, o caminhamento pelos vértices e arestas do mesmo.

Seja G=(V,E) um grafo conexo. Define-se:

- uma **aresta (v,w)** de G é dita **explorada**, quando ela é selecionada a partir de um vértice **v**; neste caso, o **vértice w** é dito **alcançado**;

- um **vértice v** torna-se **explorado** quando todas as suas arestas tiverem sido exploradas;

- **visitar** um **vértice v** ou uma **aresta (v,w)** é o mesmo que explorá-los.

(DEF.2.2)

Para caminhar num grafo, visitando todos os seus vértices e arestas, sem repetições desnecessárias, usa-se um rótulo (ou *marca*), que é associado a cada vértice, com a finalidade de registrar que ele já foi alcançado em algum passo da busca. Assim, um algoritmo de busca começa com todos os vértices *desmarcados*, para depois marcá-los, um a um, quando a primeira aresta incidente é explorada, isto é, quando o vértice é alcançado pela primeira vez.

Um vértice arbitrário é escolhido como inicial; ele se denomina **raiz** da busca.

A **busca em profundidade** obedece ao seguinte critério para selecionar o vértice a partir do qual será realizada a próxima exploração de aresta: **dentre todos os vértices já visitados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca** [14].

Para executar uma busca em profundidade, parte-se da raiz e seleciona-se um vértice da sua lista de adjacências; se este é *marcado*, seleciona-se outro; senão, ele recebe a *marca* de visitação e o procedimento continua a partir dele; em ambos os casos a aresta incidente ao vértice escolhido é explorada, se ainda não tiver sido. Quando toda a lista de adjacências de um vértice tiver sido percorrida, é porque todas as arestas incidentes a ele foram exploradas.

Para implementar este procedimento, usa-se uma pilha, na qual os vértices alcançados pela primeira vez são armazenados; um vértice só é retirado da pilha quando sua

exploração está completa.

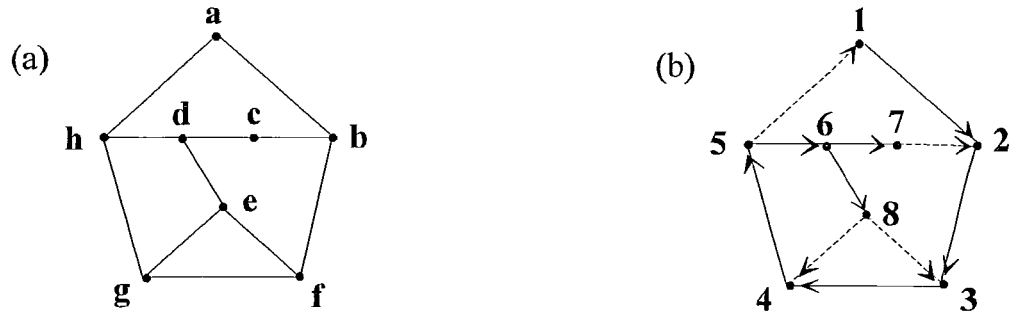
No decorrer do processo de busca, as arestas de G que alcançam vértices ainda não *marcados*, são denominadas **arestas de árvore** e seu conjunto forma uma árvore geradora de G ou uma floresta, se G não for conexo. As arestas que não pertencem à árvore geradora são chamadas **frondes** ou **arestas de retorno**, porque elas alcançam vértices já explorados e, portanto, *marcados* em algum passo anterior do processo.

Esta técnica vem sendo usada em algoritmos de grafos há bastante tempo; em 1972, Tarjan a utilizou [2] para construir uma rotulação para os vértices, numerando-os seqüencialmente de 1 a **n**, na ordem de sua seleção pela busca em profundidade, e aplicando-a mais tarde ao seu teste de planaridade [1]. Estes rótulos numéricos, atribuídos durante o processo de busca, são de tal forma que:

- se (v,w) é uma aresta de árvore, então num(v)<num(w), onde num(v) é o rótulo numérico de **v**, **v** ∈ V; analogamente para **w**;

- se (v,w) é uma aresta de retorno, então num(v)>num(w).

Fig. 2.1



Seja G o grafo da Fig.2.1a, com V={a,b,c,d,e,f,g,h} e listas de adjacências A(a)={b,h}, A(b)={f,c,a}, A(c)={b,d}, A(d)={c,e,h}, A(e)={f,g,d}, A(f)={g,e,b}, A(g)={h,e,f}, A(h)={d,g,a}. Para proceder uma busca em profundidade em G, os vértices são inicialmente *desmarcados*. Seja **a** o vértice escolhido como raiz; a ele é dado como rótulo o número 1.

A Fig.2.1b apresenta o mesmo grafo, após a busca, com os vértices rotulados de 1 a 8; as arestas de árvore estão representadas por linhas cheias e as de retorno por linhas tracejadas; a direção das setas indica a ordem de seleção dos vértices durante o processamento, de acordo com as listas de adjacências.

2.2.2- A Rotulação-ST

Lempel, Even e Cederbaum provaram [3], em 1967 , que sempre é possível associar aos vértices de um grafo G biconexo, com |V|=n, uma numeração de 1 a n, chamada de **rotulação-st**, com a seguinte característica:

- dada qualquer aresta (s,t) de G e associando ao vértice **s** o número 1, ao vértice **t** o número n, e aos demais vértices números entre 2 e n-1, **qualquer vértice de G, diferente de s e diferente de t, é adjacente a um vértice de número menor e a outro de número maior que ele próprio.** **(DEF 2.3)**

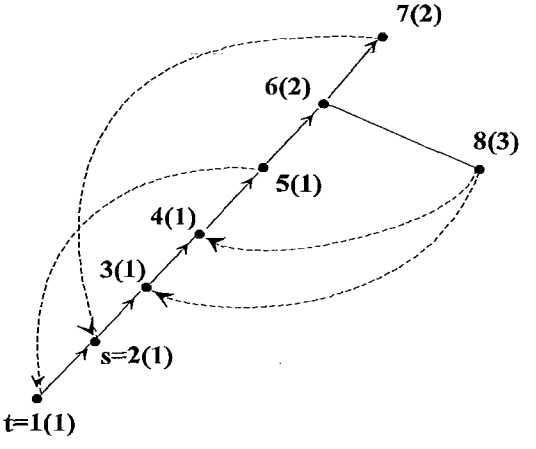
A prova da afirmativa de Lempel-Even-Cederbaum forneceu um algoritmo para construir a rotulação-st de um grafo G com **n** vértices e **m** arestas com tempo de processamento O(n.m).

Em 1976, Even e Tarjan [3] publicaram um algoritmo com tempo O(n+m) com a mesma finalidade, baseado na busca em profundidade. O procedimento definido em [3] pode ser subdividido em três fases, como é detalhado a seguir.

A **primeira fase** realiza uma busca em profundidade em G, construindo a árvore geradora T de G, enraizada no vértice **t** e escolhendo (t,s) como a primeira aresta da busca. Como **t** é pai de **s** em T, e sendo G biconexo, não existe outra aresta de árvore saindo de **t**. A busca rotula os vértices de T de com números de **1** a **n**, sendo 1 o rótulo de **t** e 2 o de **s**; durante o processo, calculam-se também os valores **L(v)**, para todo **v** de G, onde:

L(v) = menor entre num(v) e num(u), este sendo o vértice de menor rótulo acessável por **v**, num caminho em T, finalizado por uma única aresta de retorno.

Fig. 2.2



Seja G=(V,E) o grafo da Fig.2.1a com suas listas de adjacências; as Figs.2.1b e

2.2 ilustram a busca em profundidade realizada em G; a segunda figura apresenta, ao lado de cada vértice, entre parênteses, os valores calculados para L(v), **v** ∈ V, na primeira fase do procedimento de rotulação-st.

A **segunda fase** consiste do procedimento CAMINHOS, que se baseia na relação entre a busca em profundidade e a biconectividade, para subdividir o grafo biconexo em caminhos simples.

Inicialmente são marcados como *novos* todos os vértices e arestas do grafo, exceto **s**, **t** e **(s,t)**, aos quais é atribuída a marca *velho*. Na primeira chamada, a rotina acha uma caminho simples de **s** para **t**, que não contém (s,t), e marca as arestas e vértices deste caminho como *velhos*. Em cada chamada subseqüente, a rotina acha um caminho simples de novas arestas de um vértice **v** *velho* para algum vértice distinto **w**, também *velho*; marca arestas e vértices deste caminho como *velhos*, e retorna o caminho.

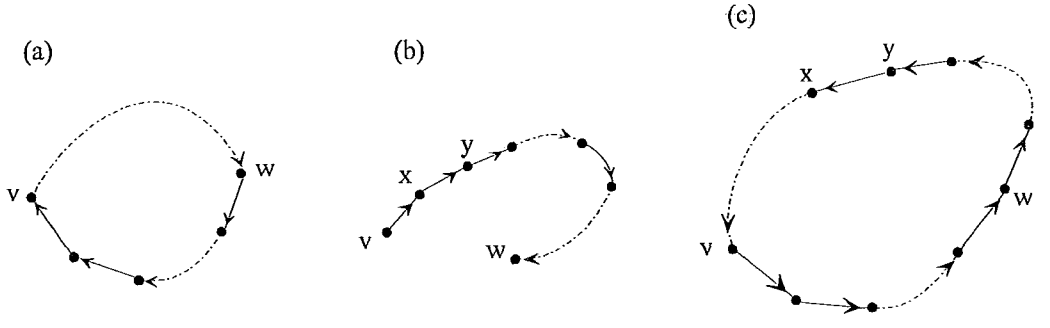
As possibilidades de construção de caminhos, analisadas pelo procedimento CAMINHOS e ilustradas na Fig.2.3, são as seguintes, tendo-se em **v** o vértice de partida:

(caso a) existe uma aresta de retorno (v,w) *nova*, com **v** descendente de **w** por algum caminho em T; o caminho consiste somente da aresta (v,w);

(caso b) existe uma aresta de árvore (v,x) *nova*, com **v** pai de **x**; o caminho construído vai conter uma ou mais arestas de árvore -(v,x) sendo a primeira- e no máximo uma aresta de retorno, que alcança o vértice **w**, encerrando o caminho; na primeira vez que este caso ocorre, v=s e w=t. As novas arestas de árvore ou de retorno, escolhidas a partir de **x** e alcançando **y**, são tais que L(y)=L(x) ou y=L(x);

(caso c) existe uma aresta de retorno (x,v) *nova*, com **v** antecessor de **x** por algum caminho em T; o caminho construído é formado por esta aresta e as arestas de árvore (y,x) *novas* encontradas, com o vértice **x** ainda marcado como *novo* e sendo **y** o pai de **x**.

Fig. 2.3



A seguir, a rotina CAMINHOS.

procedimento CAMINHOS;

```
{ subdivisão do grafo em caminhos simples }
começar
  { caso a }
  . se existe uma aresta de retorno (v,w) nova
    então
    começar
      . marcar aresta (v,w) velha;
      . seja o caminho v,w;
    fim
    senão { caso b }
      . se existe uma aresta de árvore (v,x) nova
        então
        começar
          . marcar aresta (v,x) velha;
          . inicializar o caminho como v,x;
          . enquanto x for novo, fazer
            começar
              . achar uma aresta (x,y) nova, com
                              y = L(x) ou L(y) = L(x);
              . marcar x e (x,y) velhos;
              . adicionar y ao caminho;
                    . x := y;
            fim
        fim
        senão { caso c }
          . se existe uma aresta de retorno (x,v) nova
            então
            começar
              . marcar aresta (x,v) velha;
              . inicializar o caminho como v,x;
              . enquanto x for novo, fazer
                começar
                  . achar uma aresta de árvore(y,x) nova
                  . marcar x e (y,x) velhos;
                  . adicionar y ao caminho;
                  . x := y;
                fim
            fim
            senão seja o caminho nulo;
fim;
```

O caminhos não vazios gerados pelo procedimento acima para o grafo da Fig.2.2 são 2,3,4,5,1; 2,7,6,5; 6,8,3 e 8,4, nesta ordem; o primeiro e o terceiro caminhos foram gerados no **caso b**, o segundo, no **caso c** e, o quarto, no **caso a**.

A **terceira fase** é o procedimento STNUMERO, que, usando a rotina CAMINHOS, calcula os novos rótulos para os vértices. O método mantém uma pilha com todos os vértices *velhos*; ela é inicializada com **s** sobre **t**. No passo geral do algoritmo, o vértice **v** do topo da pilha é eliminado e CAMINHOS é chamado. Se a rotina retorna um

caminho $v,v_1,\ldots,v_{k-1},v_k$, então v_{k-1}, $v_{k-2},\ldots$, v_1 e v são adicionados ao topo da pilha, nesta ordem. Se a rotina retorna um caminho nulo, então **v** recebe como rótulo-st o valor do próximo número disponível na rotina e não volta para a pilha. O processo se repete até a pilha estar vazia.

A seguir a versão do procedimento STNUMERO:

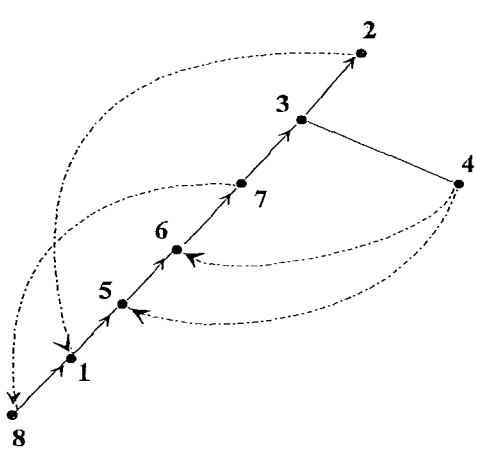
procedimento STNUMERO;

```
{ determinação dos valores da rotulação-st de G }
começar
  . marcar s, t e (s,t) velho e demais vértices e arestas novo;
  . inicializar pilha com s no topo de t;
  . i := 0;
  . enquanto pilha não estiver vazia, fazer
    começar
      . seja v o vértice topo da pilha;
      . eliminar v da pilha;
      . seja v,v1,...,vk-1,vk o caminho achado por CAMINHOS;
      . se o caminho não for nulo
        então adicionar vk-1,vk-2,...,v1 e v   à pilha
        senão num-st(v) := i := i+1;
    fim
fim;
```

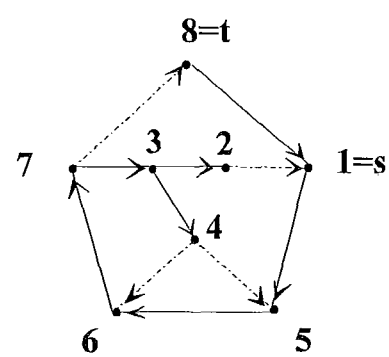
A Fig.2.4 mostra o grafo da Fig.2.2 com os vértices rotulados pela rotina STNUMERO, a partir dos caminhos obtidos por CAMINHOS. Observar que, após o procedimento STNUMERO, o rótulo de **s** é 1 e o de **t** é **n**, ou 8, no caso.

Fig. 2.4

(a)



(b)



2.2.3- As Árvores-PQ

As **árvores-PQ**, definidas por Booth e Lueker [4] em 1976, são eficientes estruturas de dados utilizadas para analisar as permutações possíveis de um conjunto U. Dentre as suas aplicações conhecidas podem ser citadas: verificar propriedades de matrizes, reconhecer grafos de intervalo e testar a planaridade de grafos.

Dado um conjunto universo $U = \{a_1, a_2, \ldots\ldots, a_m\}$, a classe de árvores-PQ sobre este conjunto é definida como sendo composta por todas as árvores enraizadas e ordenadas, cujas folhas são os elementos de U e cujos nós internos (não folhas) são P-nós ou Q-nós. **(DEF 2.4)**

As operações a seguir são utilizadas para construir uma árvore-PQ corretamente:

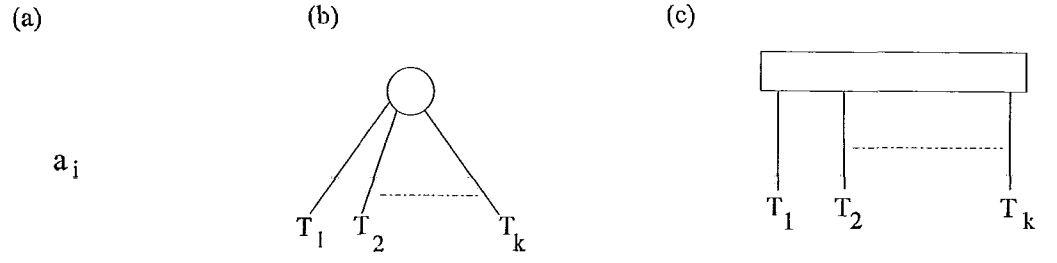
(i) qualquer elemento a_i de U é uma árvore-PQ, cuja raiz é o próprio elemento;

(ii) se T_1, T_2, ..., T_k são árvores-PQ, então, se todas forem reunidas como filhas de um P-nó, tem-se uma nova árvore-PQ, cuja raiz é este P-nó;

(iii) da mesma forma se forem reunidas como filhas de um Q-nó. **(DEF.2.5)**

A Fig. 2.5 ilustra as três árvores construídas, onde um P-nó é representado como um círculo e um Q-nó como um retângulo.

Fig. 2.5



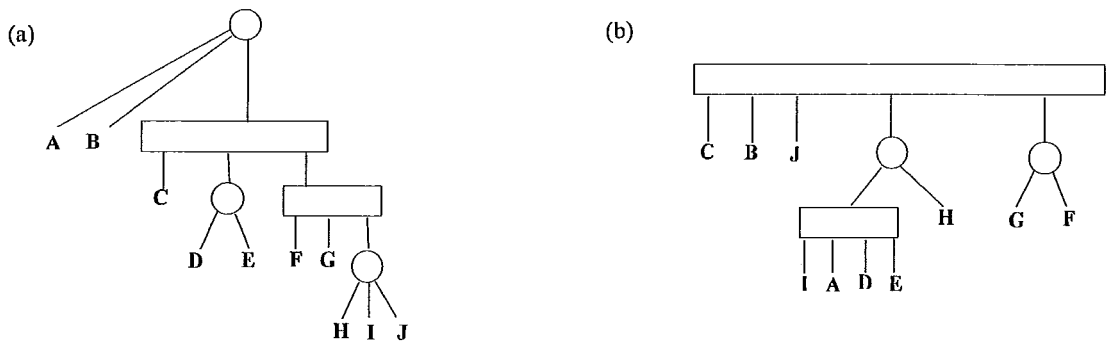
Uma árvore-PQ é dita **própria** se:

- cada elemento a_i de U aparece exatamente uma vez como folha;
- um P-nó tem no mínimo dois filhos;
- um Q-nó tem no mínimo três filhos. **(DEF.2.6)**

Lendo as folhas de uma árvore-PQ da esquerda para a direita, temos sua **fronteira** que é, obviamente, uma permutação do conjunto U. **(DEF.2.7)**

Seja o conjunto U={A,B,C,D,E,F,G,H,I,J}; a Fig.2.6 apresenta duas árvores-PQ que podem representá-lo. A fronteira da árvore (a) é ABCDEFGHIJ e a da árvore (b), é CBJIADEHGF.

Fig. 2.6



Duas árvores-PQ são ditas **equivalentes** se a segunda é obtida da primeira pela aplicação de uma ou mais transformações de equivalência, que são reorganizações consideradas válidas para os nós da árvore. Existem somente dois tipos de transformação de equivalência possíveis:

- permutar arbitrariamente os filhos de um P-nó;
- reverter os filhos de um Q-nó. **(DEF.2.8)**

Os filhos de um P-nó têm completa liberdade para serem permutados, o que quer dizer que não existe uma ordenação esquerda-direita obrigatória entre eles. Já os filhos de um Q-nó têm movimentação restrita a uma simples reversão, o que quer dizer que sempre os mesmos dois filhos serão extremos e que os outros filhos serão sempre internos; e mais, estes filhos internos terão sempre os mesmos irmãos imediatos. Visivelmente não há diferença real entre um P-nó e um Q-nó, ambos com somente dois filhos, pois a permutação dos filhos fica igual à sua reversão.

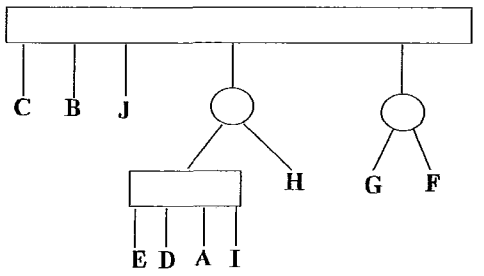
A Fig.2.7 mostra uma árvore equivalente à da Fig. 2.6b, obtida pelas reorganizações definidas acima.

Árvores equivalentes têm fronteiras diferentes; o conjunto de todas as fronteiras obtidas por transformações de equivalência de uma mesma árvore é chamado de conjunto de **permutações consistentes** e denotado por

$$\text{CONSISTENT}(T) = \{ \text{FRONTEIRA}(T') \mid T' \equiv T \}$$

(DEF.2.9)

Fig. 2.7



O problema mais importante que uma árvore-PQ se propõe a resolver não é representar um conjunto U, mas sim, o **agrupamento de todos os elementos de um particular subconjunto S de U.**

As **permutações permitidas** serão aquelas onde os elementos de S ocorrem, na fronteira da árvore, em uma seqüência consecutiva. **(DEF.2.10)**

Como os elementos do subconjunto S são restritos a aparecer juntos, o número de permutações consistentes fica reduzido. **S** também é conhecido como o **conjunto de restrições** impostas aos elementos de U.

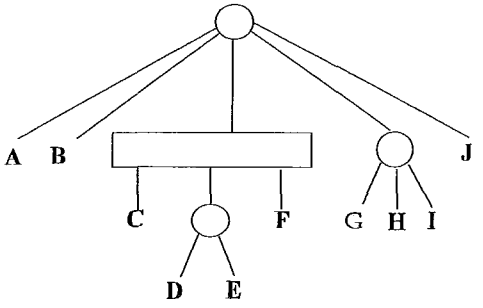
Seja o conjunto universo U={A,B,C,D,E,F,G,H,I,J} e o conjunto de restrições S={G,H,I}. As permutações permitidas são aquelas onde os elementos de S estão consecutivos; elas são, naturalmente, um subconjunto das permutações consistentes. Uma árvore-PQ pode representar **U restrito a S** se sua fronteira contiver uma das permutações dos elementos de S. A árvore da Fig.2.6a, cuja fronteira é ABCDEFGHIJ, tem os elementos de S consecutivos; ela, portanto, pode representar o conjunto de permutações permitidas para o conjunto U sob a restrição de S.

Para obter uma árvore-PQ onde os elementos do conjunto S apareçam consecutivos, será freqüentemente necessário reorganizá-la, alterando a ordem esquerda-direita de alguns filhos de alguns nós; utilizam-se, então, as transformações de equivalência definidas. A fronteira da árvore da Fig.2.6b não apresenta os elementos de S em uma seqüência consecutiva, mas ela pode ser reorganizada, revertendo o Q-nó **x**, passando a atender às restrições impostas pelo conjunto S (ver Fig.2.7).

Seja o conjunto de restrições S definido como anteriormente; suas permutações são GHI, GIH, HGI, HIG, IGH, IHG. Observa-se que, para qualquer conjunto universo U, as árvores-PQ que podem representar todas estas permutações devem ter os elementos de S como filhos de um único P-nó.

A árvore da Fig.2.8 e todas as suas equivalentes podem representar todas as permutações permitidas de U restrito a S, devido ao P-nó criado como pai das folhas.

Fig. 2.8



Nas Figs.2.6a e 2.7, pode-se notar que a organização dos P-nós e Q-nós destas árvores-PQ não impede que suas fronteiras venham a ser alteradas por alguma outra transformação de equivalência, necessária ao atendimento de uma nova restrição S'. Essa observação motiva a discussão a seguir.

Seja uma árvore-PQ construída para representar um conjunto universo U; dado um conjunto de restrições S, supõe-se que, utilizando as transformações de equivalência, é possível obter uma nova árvore-PQ cuja fronteira contém os elementos de S em alguma seqüência consecutiva. Esta árvore-PQ, portanto, satisfaz às restrições de S. O objetivo, agora, é **tornar definitivo** este resultado, isto é, torná-lo uma característica dos nós da árvore, através de P-nós e Q-nós que espelhem a obrigatoriedade dos agrupamentos conseguidos, de tal forma que, se for dado um outro conjunto de restrições S' e a árvore for transformada numa equivalente para satisfazer S', ela não deixe de atender às restrições de S. Define-se, então, a operação de **REDUÇÃO**, a ser aplicada sobre as árvores-PQ.

Seja um conjunto universo U, um subconjunto S de U e uma árvore-PQ T com os elementos de S consecutivos na sua fronteira. Deseja-se obter uma nova árvore, cujas permutações consistentes sejam exatamente aquelas que têm as folhas selecionadas por S *obrigadas* a ocorrer como uma seqüência consecutiva, isto é, as próprias permutações permitidas. Esta nova árvore é chamada **S-redução de T** e se denota REDUCE (T,S).

(DEF. 2.11)

O procedimento se aplica numa sucessão de MOLDES pré-definidos sobre os nós de uma árvore-PQ. Cada molde é constituído de um PADRÃO e de uma SUBSTITUIÇÃO. Um molde se aplica sobre um determinado nó (o que está sendo analisado) e somente ele e seus filhos são alterados. Um **padrão** é uma forma genérica de analisar um nó; a cada um deles corresponde uma **substituição**, que pretende consolidar determinados agrupamentos já obtidos. A escolha do molde conveniente a ser aplicado, isto

é, do padrão de análise e de sua correspondente substituição, depende do conjunto S e da fronteira da subárvore enraizada no nó em estudo.

São onze pares de PADRÃO x SUBSTITUIÇÃO, isto é, onze MOLDES de análise definidos inicialmente por Booth e Lueker e, depois, reduzidos a nove na implementação do algoritmo.

Antes define-se:

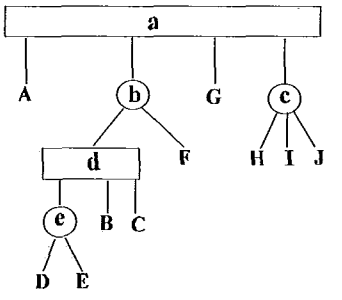
- **nó cheio** como aquele que tem todos os seus descendentes pertencentes a S;
- **nó vazio** como aquele que não tem nenhum descendente pertencente a S;
- **nó parcial** como aquele que tem parte dos seus descendentes, mas não todos, em S;
- **nó pertinente** como aquele que é cheio ou parcial;
- **subárvore pertinente de T com respeito a S** como a subárvore de altura mínima, cuja fronteira contém todos os elementos de S. **(DEF.2.12)**

Sejam U = {A,B,C,D,E,F,G,H,I,J} e S = {D,E,F}.

Na árvore-PQ da Fig.2.9, tem-se que:

- são nós cheios as folhas D,E,F e o P-nó **e**; são nós vazios as folhas A,B,C,G,H,I,J e o P-nó **c**; são nós parciais o Q-nó **a**, o P-nó **b** e o Q-nó **d**; são nós pertinentes os cheios e os parciais; a subárvore pertinente é a que tem raiz no P-nó **b**.

Fig. 2.9

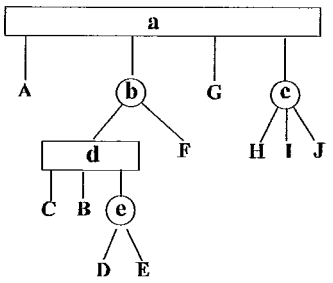


Aplicando sobre a subárvore pertinente da Fig.2.9 uma transformação de equivalência conveniente (reversão do Q-nó **d**), pode-se obter uma nova árvore, na qual os elementos do conjunto de restrições S ocorrem numa seqüência consecutiva, conforme ilustra a Fig. 2.10.

Agora o objetivo é *reduzir* a subárvore pertinente, de forma a espelhar o agrupamento dos elementos de S na sua construção, isto é, fazendo com que ela represente as permutações permitidas para o conjunto U restrito a S, mesmo se vier a ser submetida a alguma outra transformação de equivalência. No casamento com os padrões dos moldes e a

conseqüente substituição, deve-se ter certeza que a fronteira da árvore enraizada no nó em análise tem todas as suas folhas pertinentes ocorrendo como uma seqüência consecutiva. Assim, as transformações de equivalência necessárias para agrupar os elementos de S devem ser realizadas antes da aplicação dos moldes. Se não for possível chegar a uma seqüência consecutiva na fronteira da árvore-PQ, os moldes não poderão ser aplicados.

Fig. 2.10



O processo de redução se inicia nas folhas da árvore, que são os elementos de U; somente depois de todos os filhos terem sido analisados, o pai o será. Essa estratégia de baixo para cima é fundamental, porque permite que a informação contida nos filhos seja propagada para os pais. Para começar a análise dos moldes, as **folhas** são assinaladas como **cheias ou vazias.**

Os moldes estão enquadrados em onze casos, cada um apresentando um padrão e uma substituição correspondente. Eles são detalhados um a um, para os nós internos; esses podem ser P-nós ou Q-nós e, nesta ordem, os casos estão descritos. Os nós hachurados representam nós cheios e os não, vazios; as subárvores filhas aparecem como pequenos triângulos. O nome dos casos entre parênteses é o dado por Booth-Lueker. Seja U={A,B,C,D,E,F,G,H,I,J} o conjunto universo usado nas ilustrações.

CASO 1 (P0): este padrão espera que o nó em análise seja um P-nó e tenha todos os seus filhos vazios. Neste caso nenhuma alteração é feita na árvore; somente este P-nó é assinalado como vazio (ver Fig.2.11).

Fig. 2.11

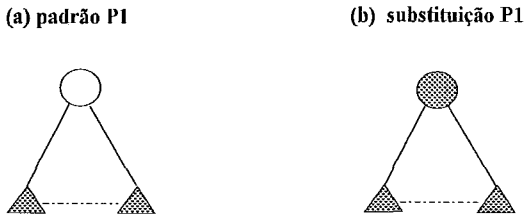
(a) padrão P0 **(b) substituição P0**

Tomando como nó para análise o P-nó **c** da Fig.2.9 e sendo S={D,E,F}, as

folhas H, I e J são vazias; o padrão de casamento se adapta e a substituição consistirá em assinalar o P-nó **e** como vazio.

CASO 2 (P1): O nó em análise é um P-nó que tem todos os seus filhos cheios. Como no caso 1, nenhuma alteração é feita na árvore, exceto assinalar o P-nó como cheio (ver Fig.2.12).

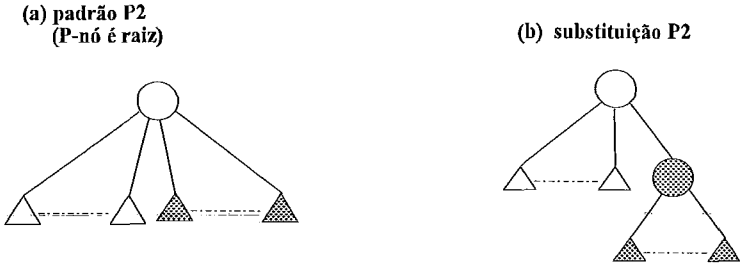
Fig. 2.12



O P-nó **e** da Fig.2.9, para S={D,E,F}, tem filhos D e E, ambos cheios.

CASO 3 (P2): este padrão espera que o nó em análise seja um P-nó com filhos cheios e vazios e *seja a raiz* da subárvore pertinente. A substituição correspondente consiste em agrupar os filhos cheios para assegurar a consecutividade de S (ver Fig.2.13).

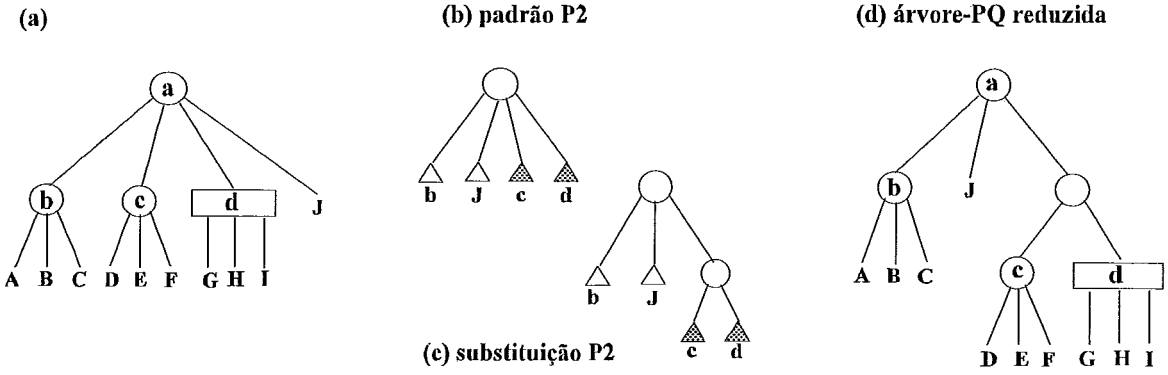
Fig. 2.13



É interessante observar que antes da substituição os nós cheios estavam livres para qualquer organização; após, não mais. Já os nós vazios permanecem livres.

A Fig.2.14 apresenta uma análise deste caso, para o conjunto de restrições S={D,E,F,G,H,I}; o P-nó **a** é a raiz da subárvore pertinente e o molde P2 pode ser aplicado, como mostram os itens **b** e **c** da mesma figura; no item **d** tem-se a árvore-PQ reduzida de T com relação a S.

Fig. 2.14



CASO 4 (P3): o padrão espera que o nó em análise seja um P-nó, tenha alguns filhos cheios e outros vazios (um P-nó com esta característica é chamado **isoladamente parcial)** e *não seja a raiz* da subárvore pertinente.

A substituição proposta gera um Q-nó com somente dois filhos, isto é, uma árvore-PQ imprópria; ela é aceita, pois no decorrer do processo outras folhas pertinentes, que certamente existem já que esse não é o nó raiz da subárvore pertinente, poderão se juntar a este Q-nó (ver Fig.2.15).

Fig. 2.15

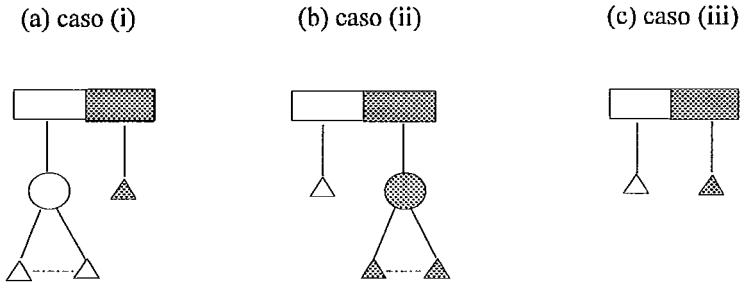
Este molde tem algumas variações ou casos especiais, apresentados na Fig.2.16 com as respectivas substituições:

(i) o nó em análise só tem 1 filho cheio;

(ii) o nó em análise só tem 1 filho vazio;

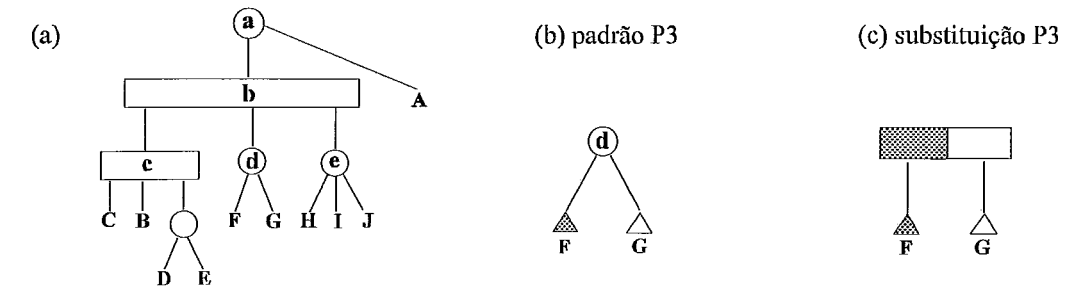
(iii) o nó em análise tem 1 filho e 1 filho vazio.

Fig. 2.16



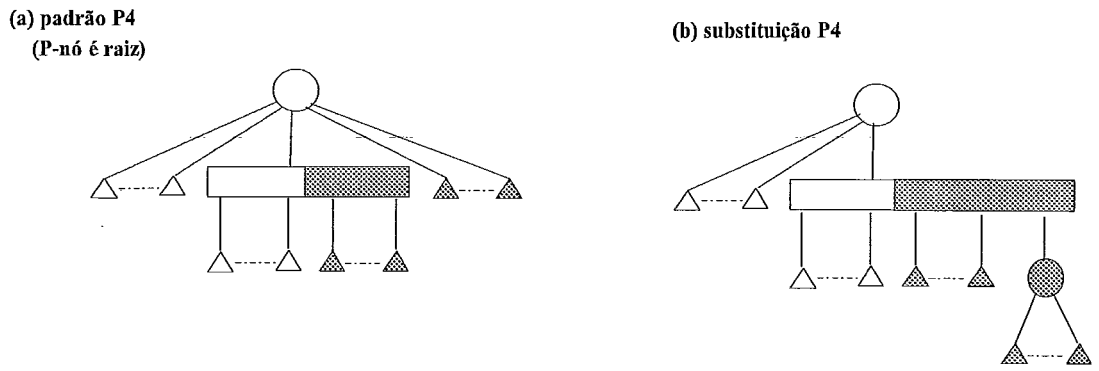
A Fig.2.17 mostra uma aplicação deste molde; considerando S={D,E,F}, o P-nó **d** é uma situação particular deste caso, pois G é filho vazio e F é filho cheio, o que recai na variação (iii).

Fig. 2.17



CASO 5 (P4): o padrão espera que o P-nó em análise tenha exatamente 1 filho isoladamente parcial e *seja a raiz* da subárvore pertinente (ver Fig.2.18).

Fig. 2.18



Como no caso anterior, algumas variações aparecem:

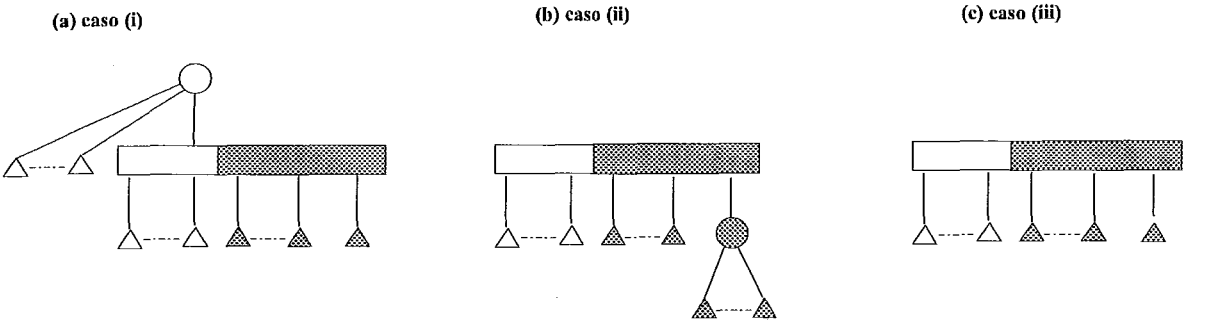
(i) o nó em análise só tem 1 filho cheio;

(ii) o nó em análise não tem nenhum filho vazio;

(iii) o nó em análise só tem 1 filho cheio e nenhum vazio.

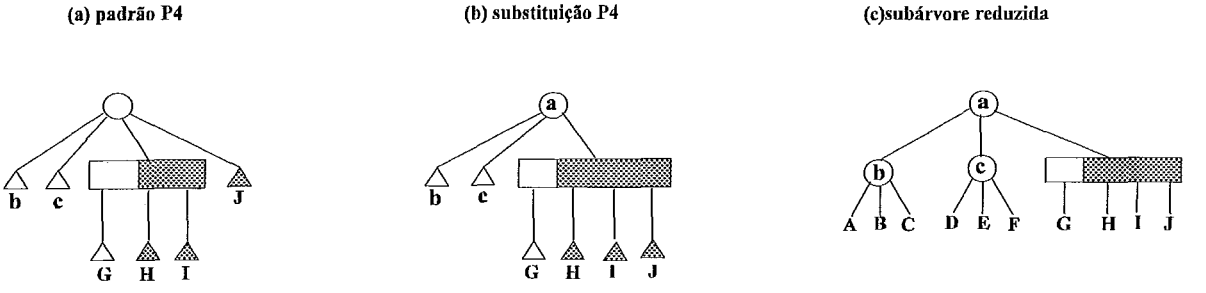
A Fig.2.19 mostra as substituições realizadas para estas variações.

Fig. 2.19



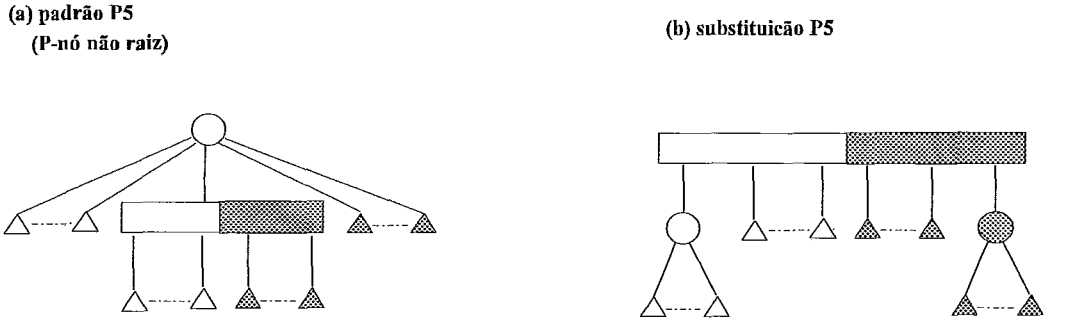
Tomando a árvore-PQ da Fig.2.14a e alterando o conjunto de restrições para S={H,I,J}, o P-nó **a** é agora a raiz da subárvore pertinente. Tem-se uma ilustração para este caso, mais especificamente para a variação (i), mostrada na Fig.2.20.

Fig. 2.20



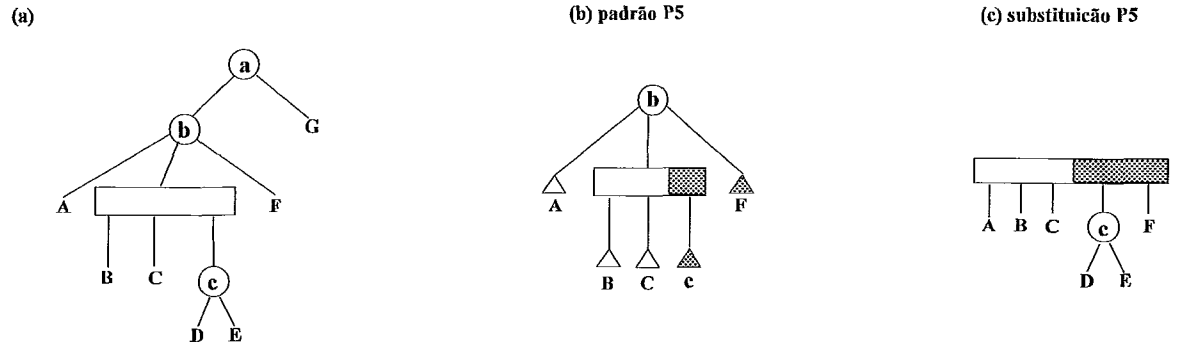
CASO 6 (P5): o padrão espera que o P-nó em análise tenha exatamente 1 filho isoladamente parcial, mas *não seja a raiz* da subárvore pertinente (ver Fig.2.21). Os casos especiais existentes são como no caso5 (P4) e as substituições são análogas.

Fig. 2.21



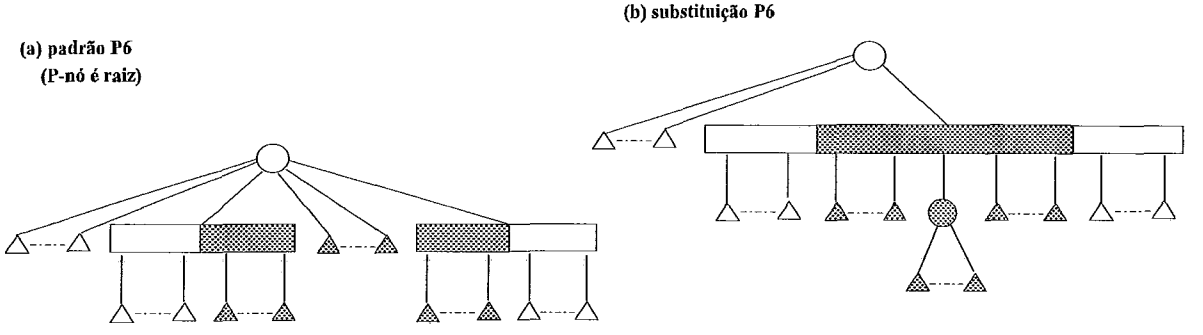
Seja a árvore-PQ da Fig.2.22a, com conjunto de restrições S={D,E,F,G}; a raiz da subárvore pertinente é o P-nó **a.** Ao analisar o P-nó **b**, ele recai no caso P5 (ver Figs.2.22b e 2.22c).

Fig. 2.22



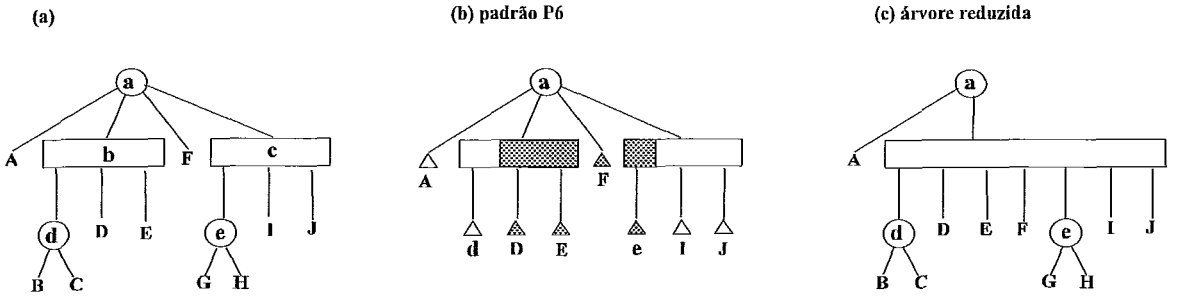
CASO 7 (P6): o último caso relativo a um P-nó tem como padrão um P-nó com exatamente 2 filhos isoladamente parciais e que é *a raiz* da subárvore pertinente. Neste caso o P-nó só pode ser a raiz da subárvore pertinente ou não haverá prosseguimento possível no casamento dos moldes, pois, conforme a substituição definida, os filhos cheios, que serão colocados como filhos interiores do Q-nó, ficarão impedidos de se agrupar a qualquer outro nó cheio (ver Fig.2.23).

Fig. 2.23



A árvore-PQ da Fig.2.24a, para S={D,E,F,G,H}, serve como exemplo. O nó em análise é o P-nó **a**, que é a raiz da subárvore pertinente e tem dois filhos isoladamente parciais; os itens (b,c) da mesma figura mostram a adequação do padrão P6 e a respectiva substituição.

Fig. 2.24

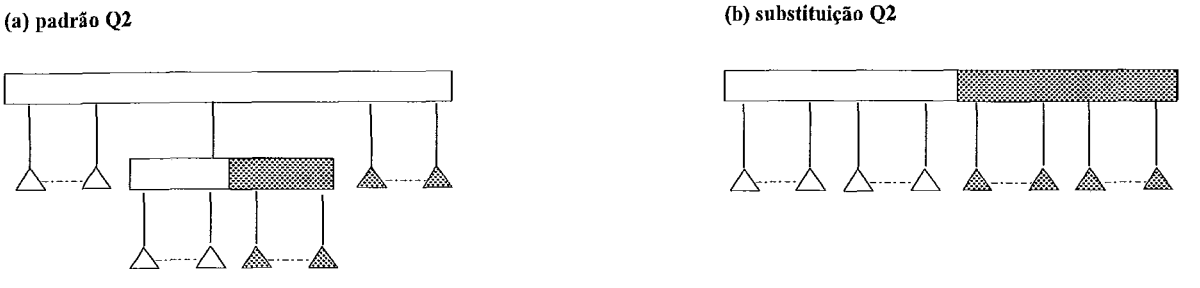


CASO 8 (Q0): o nó em análise é, agora, um Q-nó que tem todos os seus filhos vazios. Nenhuma alteração é feita na árvore, exceto assinalar o Q-nó como vazio.

CASO 9 (Q1): o nó em análise é um Q-nó que tem todos os seus filhos cheios. Nenhuma alteração é feita na árvore, exceto assinalar o Q-nó como cheio.

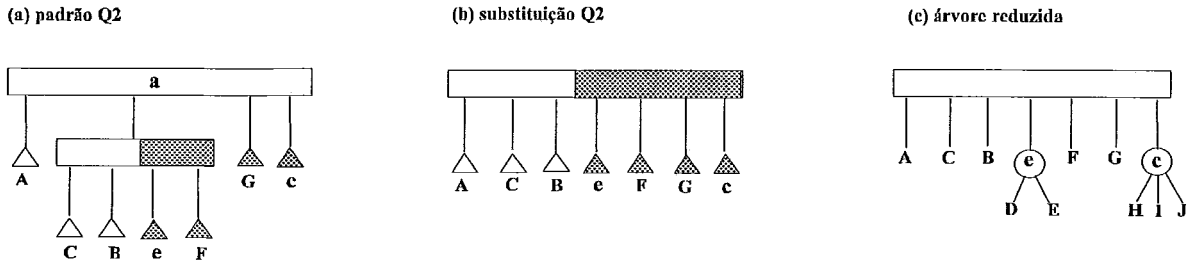
CASO 10 (Q2): o padrão de casamento espera um Q-nó isoladamente parcial; um Q-nó é dito **isoladamente parcial** se ele tem, no máximo, um Q-nó parcial como filho e tem um ou mais filhos cheios que podem ser agrupados todos à sua esquerda ou à sua direita, isto é, nas suas extremidades (ver Fig.2.25). Este caso não exige que o Q-nó seja raiz da subárvore pertinente, nem impede que ele o seja.

Fig. 2.25



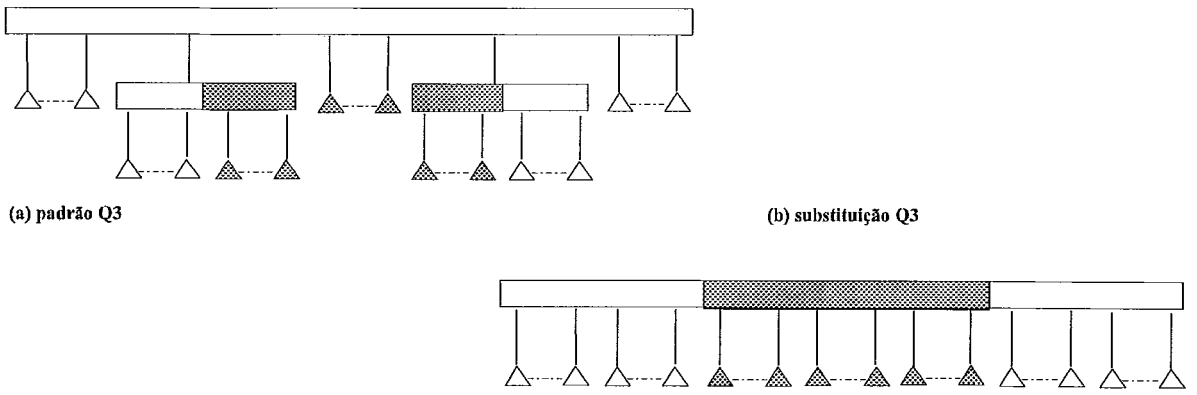
Tomando a árvore-PQ da Fig2.20c, com conjunto de restrições S'={D,E,F,G,H,I,J}, tem-se uma ilustração para o caso 10, demonstrado na Fig.2.26; o nó em análise é o Q-nó **a** e seu filho parcial é o Q-nó pai das folhas C, B e F e do P-nó **e**.

Fig. 2.26



CASO 11 (Q3): o padrão de casamento espera que o Q-nó em análise seja duplamente parcial, isto é, tenha um máximo de dois filhos parciais, e *seja a raiz* da subárvore pertinente. Como no caso 7, se o Q-nó for duplamente parcial e não for a raiz da subárvore pertinente, é porque existem outros filhos cheios, que não mais poderão ser agrupados, interrompendo a aplicação dos moldes (ver Fig.2.27).

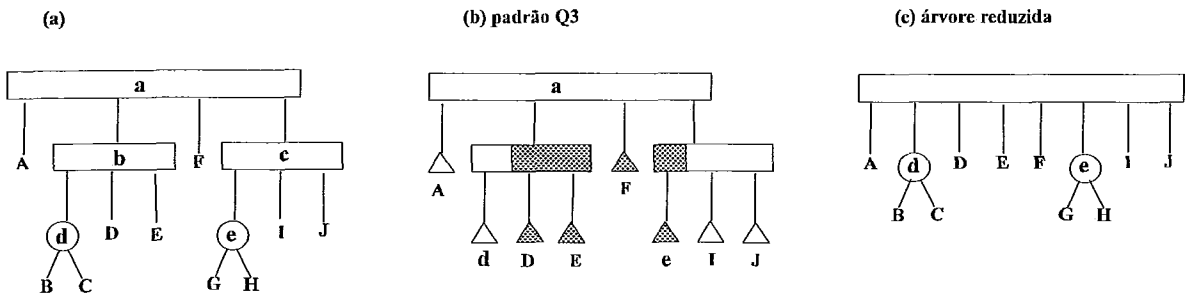
Fig. 2.27



Algumas variações no padrão deste molde são:
(i) um dos Q-nós filho é cheio;
(ii) ambos os Q-nós filhos são cheios;
(iii) não há filhos cheios entre os dois Q-nós parciais.

Na árvore-PQ da Fig.2.28a, para um conjunto de restrições S={D,E,F,G,H}, considera-se como nó em análise o Q-nó **a**, que é duplamente parcial. Os itens (b) e (c) da mesma figura mostram a adequação do padrão Q3 e a respectiva substituição.

Fig. 2.28



Os onze casos definidos por Booth-Lueker são os únicos considerados possíveis; qualquer outro é ilegal e resultaria na falha do processo de casamento de moldes.

Usando os moldes, procura-se e aplica-se um padrão e uma substituição para cada nó da árvore _sempre os filhos antes dos pais_, até a análise da raiz da subárvore pertinente ou até que, em algum momento, nenhum padrão se adapte, quando, então, o processo é interrompido.

Pode-se observar também que, após a aplicação de qualquer dos moldes descritos, os elementos do conjunto S de restrições continuam ocorrendo numa seqüência consecutiva dentro da fronteira da árvore reduzida e na **de qualquer árvore equivalente a ela**.

A S-redução de uma árvore pode ser caracterizada em termos de permutações consistentes, da seguinte forma:

- sejam $U=\{a_1,a_2,...,a_m\}$, $S=\{a_{i1},a_{i2},...,a_{it}\}$, um subconjunto qualquer de U, e T(U,S) a árvore-PQ da Fig.2.29.

Pode-se observar que CONSISTENT (T(U,S)) é exatamente o conjunto de todas as permutações onde os elementos de S ocorrem como uma subsequência consecutiva. Booth-Lueker provaram o seguinte teorema.

TEOREMA 2.1: Seja T qualquer árvore-PQ e seja S qualquer subconjunto do conjunto universal U. Então tem-se

CONSISTENT (REDUCE(T,S)) = CONSISTENT (T) $\cap$ CONSISTENT (T(U,S))

Este teorema de Booth-Lueker demonstra porque as árvores-PQ podem ser usadas para representar todas as permutações nas quais cada conjunto, numa família de subconjuntos, ocorre como uma subsequência consecutiva.

Fig. 2.29

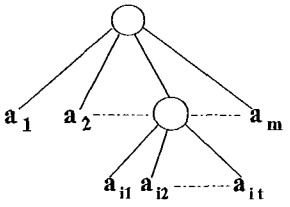
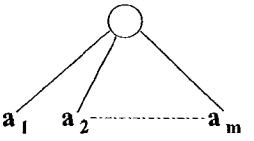


Fig. 2.30



Seja T a árvore-PQ da Fig.2.30 para um conjunto $U=\{a_1,a_2,\ldots,a_m\}$; ela é chamada **árvore universal**. Uma S-redução de U restringe as permutações consistentes àquelas onde os elementos de S são consecutivos. Várias reduções para subconjuntos S_1, S_2, ..., S_k de U vão poder produzir todas as permutações onde cada um dos conjuntos de restrição é consecutivo. Essa operação de redução será detalhada na seção 2.4, juntamente com o algoritmo de teste de planaridade de Booth-Lueker.

2.3- O Algoritmo de Hopcroft e Tarjan

O algoritmo de reconhecimento de planaridade de Hopcroft-Tarjan [1] aborda o problema usando o método da **adição de caminhos**. Sua idéia básica é *quebrar* o grafo em subgrafos, e tentar colocá-los no plano, um a um, até chegar a um desenho completo do grafo, sem interseção de arestas. Este método foi originalmente proposto por Auslander-Parter e depois corretamente reformulado por Goldstein, num algoritmo iterativo, usado para posicionar cada pedaço do grafo.

O algoritmo de teste de planaridade de Hopcroft-Tarjan foi o primeiro a obter tempo e espaço lineares para sua execução; sua publicação é de 1974.

O objetivo do algoritmo HT é testar a planaridade de um grafo G arbitrário; ele é descrito considerando G não direcionado, simples e biconexo, isto porque [2]:

(i) um digrafo é planar se e somente se o grafo não direcionado, obtido dele, ignorando a direção das arestas, é planar;

(ii) um grafo não direcionado é planar se e somente se todas as suas componentes conexas o são;

(iii) um grafo não direcionado é planar se e somente se todas as suas componentes biconexas o são; então, neste caso, o grafo pode ser decomposto em suas componentes biconexas, a serem consideradas separadamente;

(iv) laços e arestas paralelas podem ser incluídas ou eliminadas sem afetar a planaridade de um grafo.

2.3.1- Noções Básicas

Seja G=(V,E) um grafo simples, não direcionado, com |V|=n e |E|=m, representado por suas listas de adjacências.

O algoritmo HT tem os seguintes passos iniciais:

1- eliminar do teste de planaridade os grafos que tenham $m > 3n - 6$, que não são planares, de acordo com a fórmula de Euler;;

2- decompor o grafo G em suas componentes biconexas.

A seguir, em cada componente biconexa de G, procura-se um ciclo, que se desenha no plano como uma curva simples e fechada; depois, decompõe-se o restante da componente em caminhos de arestas disjuntas, e tenta-se localizar cada um destes caminhos inteiramente dentro ou fora do ciclo inicial. Se toda a componente puder ser colocada no plano sem cruzamento de arestas, então ela é planar; repetindo o processo para todas as componentes, e tendo sucesso, conclui-se que G é planar; caso para uma delas, pelo menos, o processo não possa ser realizado, então G não é planar. A geração e a localização desses caminhos são fundamentais para se chegar a uma conclusão correta sobre a planaridade de G. Eles devem ser gerados sistematicamente, posicionados em áreas apropriadas e, se necessário, remanejados de dentro para fora do ciclo, ou vice-versa, para permitir a acomodação de outros.

Antes de iniciar seu procedimento, para cada componente biconexa do grafo, o algoritmo obtém uma rotulação numérica para os vértices, usando uma busca em profundidade, e refaz as listas de adjacências dos vértices com base na nova rotulação; só então os caminhos podem ser gerados, agora numa ordem desejável.

2.3.2- Descrição do Algoritmo

Para detalhar os passos do algoritmo, G=(V,E) será considerado um grafo simples, biconexo e não direcionado, com |V|=n e |E|=m.

O **primeiro passo** do algoritmo HT realiza uma busca em profundidade em G; um vértice arbitrário é escolhido como inicial e a ele se dá como rótulo o valor 1. Cada aresta do grafo é explorada pelo procedimento, e caracterizada como aresta de

árvore ou de retorno; ao mesmo tempo, os **vértices** são rotulados seqüencialmente, de tal maneira que:

- se (v,w) é uma aresta de árvore, então num(v) < num(w);
- se (v,w) é uma aresta de retorno, então num(v) > num(w).

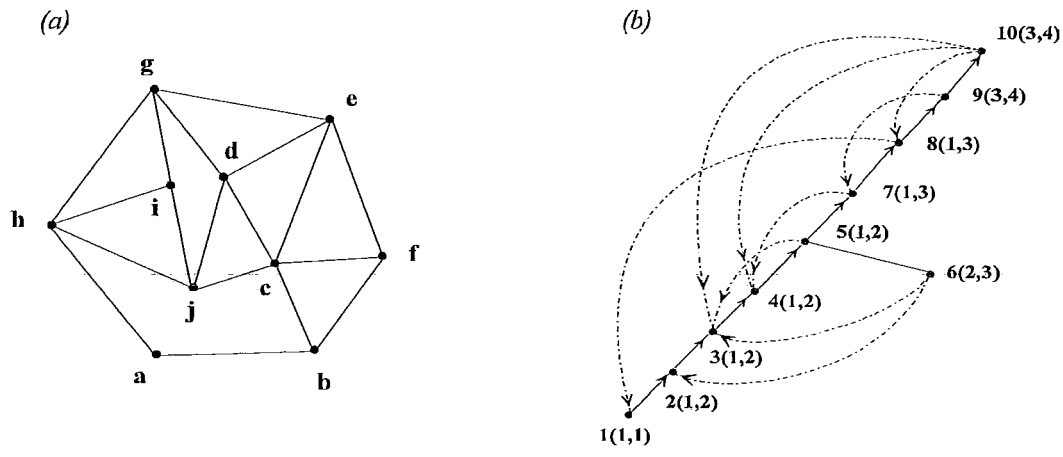
Também, durante esta busca, calculam-se os valores definidos abaixo, para cada vértice **v:**

- L1(v), que é o vértice de menor rótulo alcançado por **v**, seguindo um caminho de arestas de árvore terminado por uma única aresta de retorno;
- L2(v), que é o segundo vértice de menor rótulo alcançado por **v**, seguindo um caminho de arestas de árvore terminado por uma única aresta de retorno.

Observa-se que L1(v) é sempre diferente de L2(v), a menos que **v** seja igual a 1, quando L1(v) = L2(v) = 1. O cálculo de L1(v), que equivale a L(v), está descrito na seção 2.2 (busca em profundidade e primeira fase da rotulação-st).

Ao fim da busca, é obtido um grafo G_1, direcionado, com seus vértices rotulados numericamente, e com suas arestas identificadas como de árvore ou de retorno. As listas de adjacências são atualizadas para os novos rótulos.

Fig. 2.31



Seja G o grafo da Fig.2.31a; suas listas de adjacências são como a seguir:

a: b,h	e: f,g,d,c	i:j,h,g
b: c,f,a	f: e,c,b	j: c,d,i,h
c:d,f,e,b,j	g: h,e,d,i	
d: e,c,j,g	h: i,a,g,j	

Tomando **o vértice a** como inicial da pesquisa, a ele se dá o rótulo 1; o item (b) da mesma figura apresenta o grafo após a realização da busca, com seus vértices rotulados de 1 a 10, em substituição aos rótulos a,b,c,d,e,f,g,h,i,j, nesta ordem; as arestas de árvore estão representadas por linhas cheias e, as de retorno, por linhas tracejadas; entre parênteses, ao lado de cada vértice **v**, estão especificados os valores de L1(v) e L2(v).

No **segundo passo** do algoritmo, [1] define uma função **X**, a ser calculada para cada aresta (v,w) -**v** e **w** já com rótulos numéricos-, como abaixo:

$X[(v,w)] = 2\,w$, se (v,w) é aresta de retorno;

$X[(v,w)] = 2\,L1(w)$, se (v,w) é aresta de árvore e $v \leq L2(w)$;

$X[(v,w)] = 2\,L1(w)+1$, se (v,w) é aresta de árvore e $v > L2(w)$.

O objetivo desta função é auxiliar a **reorganização das listas de adjacências** dos vértices, para a posterior geração de caminhos. Os valores calculados são tais que entre duas arestas de retorno com mesma origem, é selecionada primeiro aquela que se dirige para o vértice de menor rótulo, isto é, o mais próximo da raiz; também, um caminho de arestas de árvore seguirá na direção que o conduz ao vértice de menor rótulo possível, alcançável utilizando uma única aresta de retorno.

As listas de adjacências de G são, então, reordenadas em ordem crescente dos valores obtidos pela função **X**, usando o procedimento de ordenação por distribuição a seguir, cujo tempo é limitado em O(n+m):

```
. começar
    . para i:=1 até (2n+1), fazer ORDEM(i):=lista vazia;
    . para cada aresta (v,w) de G, fazer
        . começar
            . calcular X((v,w));
            . adicionar (v,w) a ORDEM(X(v,w));
        . fim;
    . para v:=1 até n, fazer Padj(v):= lista vazia;
    . para i:=1 até (2n+1), fazer
        . para (v,w) ∈ ORDEM(i), fazer
            . adicionar w ao final de Padj(v);
. fim;
```

As listas Padj(v) obtidas ao final do procedimento acima são as listas de adjacências de cada vértice, reordenadas segundo a função **X**.

Para o grafo da Fig.2.31b, os valores de X[(v,w)], as arestas em cada ORDEM(i) e as novas listas de adjacências Padj(v) são apresentadas abaixo.

ARESTAS DE ÁRVORE	FUNÇÃO X	ARESTAS DE RETORNO	FUNÇÃO X	I	ORDEM (I)	V	PADJ(V)
(1,2)	2	(5,3)	6	2	(1,2),(2,3),(8,1)	1	2
(2,3)	2	(6,2)	4	3	(3,4),(4,5),(5,7),(7,8)	2	3
(3,4)	3	(6,3)	6	4	(6,2)	3	4
(4,5)	3	(7,4)	8	5	(5,6)	4	5
(5,6)	5	(8,1)	2	6	(5,3),(6,3),(10,3)	5	7,6,3
(5,7)	3	(9,7)	14	7	(8,9),(9,10)	6	2,3
(7,8)	3	(10,3)	6	8	(7,4),(10,4)	7	8,4
(8,9)	7	(10,4)	8	14	(9,7)	8	1,9
(9,10)	7	(10,8)	16	16	(10,8)	9	10,7
						10	3,4,8

Esta ordenação é fundamental para a otimização do algoritmo e, juntamente com a técnica de geração de caminhos, vai permitir formular a base para o teste de planaridade.

O **terceiro passo** do algoritmo HT é a **geração de caminhos**; partindo do vértice 1 e executando uma busca em profundidade, os caminhos são construídos, um a um, com base nas listas de adjacências reorganizadas, armazenadas em Padj(v). As arestas de árvore são selecionadas primeiro, pela definição da função **X**; quando uma aresta de retorno é encontrada, o caminho é encerrado. Cada caminho consiste, portanto, de uma seqüência de arestas de árvore e uma aresta de retorno, que o encerra, ou de uma única aresta de retorno. O primeiro caminho gerado é um ciclo, que se inicia e finaliza no vértice 1.

A rotina CAMINHOS, a seguir, é a estrutura básica do algoritmo HT. Ela requer um tempo O(n+m) para achar todos os caminhos.

```
Rotina CAMINHOS (v);
começar
  . para w ∈ Padj(v), fazer
  . se s = 0
    então
    começar
      . s := v;
      . iniciar um novo caminho p;
    fim;
  . adicionar (v,w) ao caminho corrente p;
  . se v < w
    então { (v,w) é uma aresta de árvore, com v pai de w }
      . CAMINHOS (w);
    senão
    começar  { (v,w) é uma aresta de retorno }
      . retornar o caminho corrente;
      . s := 0;
    fim
```

```
fim;
```

Os caminhos gerados por esta rotina para o grafo da Fig.2.31b, usando os valores de Padj(v) calculados anteriormente, são:

p_0 ou o ciclo C= 1,2,3,4,5,7,8,1;

p_1= 8,9,10,3; p_2= 10,4; p_3= 10,8; p_4= 9,7;

p_5= 7,4; p_6= 5,6,2; p_7= 6,3 e p_8= 5,3.

A técnica de geração dos caminhos lhes dá as seguintes propriedades, de acordo com lemas formulados e demonstrados por [1]:

(i) **LEMA 2.1:** Seja **p** um caminho gerado de **s** para **f;** quando a primeira aresta de **p** é atravessada, consideram-se todas as arestas de retorno que não foram usadas em nenhum caminho; então, **f** será o vértice de menor rótulo que pode ser alcançado por qualquer descendente de **s**, usando uma destas arestas de retorno. Se **v** é um vértice intermediário do caminho **p**, então **f** é o vértice de menor rótulo alcançável por qualquer descendente de **v**, através de uma aresta de retorno;

(ii) **LEMA 2.2:** Seja **p** um caminho gerado de **s** para **f**; então existe um caminho de arestas de árvore de **f** para **s** em G_1. Se **p** é o primeiro caminho, então ele é um ciclo; caso contrário, ele é simples e contém exatamente dois vértices (**f** e **s**) em comum com os caminhos gerados anteriormente;

(iii) **LEMA 2.3:** Sejam $\mathbf{p_1}$ e $\mathbf{p_2}$ dois caminhos gerados, com vértices iniciais e finais $\mathbf{s_1}$, $\mathbf{s_2}$ e $\mathbf{f_1}$, $\mathbf{f_2}$, respectivamente. Se $\mathbf{p_1}$ é gerado antes de $\mathbf{p_2}$ e $\mathbf{s_1}$ é um ancestral de $\mathbf{s_2}$, então $\mathbf{f_1} \leq \mathbf{f_2}$;

(iv) **LEMA 2.4:** Sejam $\mathbf{p_1}$ e $\mathbf{p_2}$ dois caminhos gerados, que tem o mesmo par **s** e **f** de vértices iniciais e finais. Seja $\mathbf{v_1}$ o segundo vértice do caminho $\mathbf{p_1}$ e $\mathbf{v_2}$ o segundo vértice do caminho $\mathbf{p_2}$. Se $\mathbf{p_1}$ é gerado antes de $\mathbf{p_2}$, $\mathbf{v_1}$ diferente de **f** (i.é, (s,v_1) não é aresta de retorno) e L2(vi) < **s**, então $\mathbf{v_2}$ é diferente de f (i.é, (s,v_2) também não é aresta de retorno) e L2 (v_2) < s.

Observa-se, nos caminhos gerados para o grafo da Fig.2.31b, que:

- em p_6=5,6,2, de **s**=5 para **f**=2, a aresta de retorno escolhida primeiro foi (6,2), pois alcança um vértice de menor rótulo do que (6,3); também, para o vértice 6, diferente de **s** e de **f**, **f**=2 é o vértice de menor rótulo que ele pode alcançar (propriedade (i));

- p_0=C é o primeiro caminho gerado e é um ciclo; todos os outros caminhos

são simples. O caminho p_6, o sétimo gerado, tem em comum com os anteriores somente seus vértices extremos, **f**=2 e **s**=5 (propriedade (ii));

- para os caminhos p_1 e p_2, gerados nesta ordem, onde o vértice 8, inicial de p_1, é ancestral de 10, inicial de p_2, tem-se que $\mathbf{f_1}=3 < 4=\mathbf{f_2}$ (propriedade (iii)).

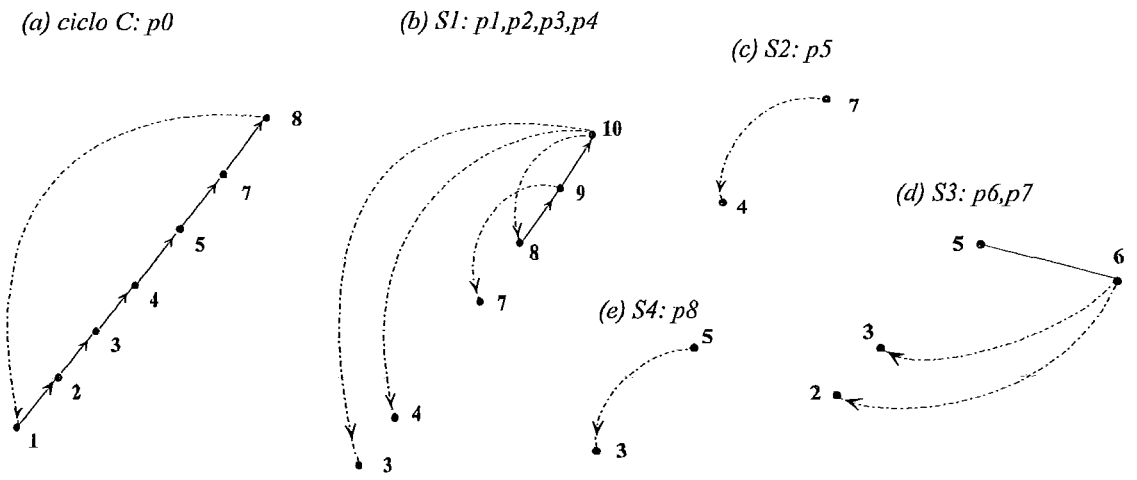
O **teste de planaridade** é realizado concomitantemente à geração de caminhos, no terceiro passo, tentando localizá-los um de cada vez no plano, sem interseção de arestas.

Para localizar os caminhos no plano, conceitua-se um **segmento**. Seja C o primeiro caminho gerado: um ciclo. Retirando C de G, os pedaços conexos do grafo resultante são agrupados em segmentos, cada um deles um subgrafo conexo em G-C. Um segmento possui as seguintes características, relativamente ao ciclo C:

(i) é composto por uma aresta de retorno solitária, não pertencente a C, mas com ambas as extremidades em C; ou

(ii) é um subgrafo iniciado por uma aresta de árvore (v,w), $\mathbf{v} \in C$ e $\mathbf{w} \notin C$, mais a subárvore enraizada em **w**, juntamente com todas as arestas de retorno que partem desta subárvore.

Fig. 2.32



A ordem da geração de caminhos é tal que:

- todos os caminhos de um segmento são gerados antes de qualquer caminho de um outro segmento;

- os segmentos são criados em ordem decrescente de **v**, vértice inicial ou *base* do primeiro caminho de um segmento S. Assim, qualquer segmento gerado após S, terá um vértice base menor ou igual a v.

Um segmento se liga ao ciclo C por uma aresta de árvore que parte de C e

por uma ou mais arestas de retorno que entram em C.

A Fig.2.32 apresenta o ciclo C e os segmentos S_1, S_2, S_3 e S_4 do grafo G-C; os vértices base dos segmentos são, respectivamente, 8, 7, 5 e 5.

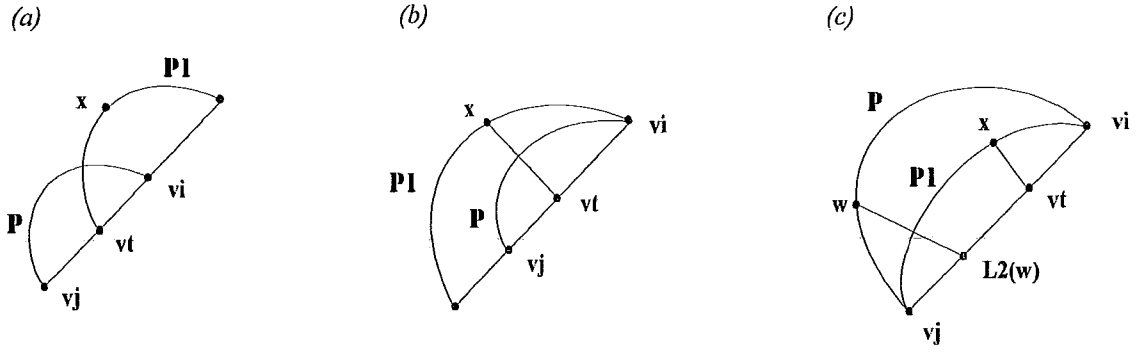
O objetivo de caracterizar um segmento é que, cada um deles, deve ser localizado completamente em um único lado do ciclo C, isto é, dentro ou fora de dele , conforme o Teorema de Jordan [1].

Com base na técnica de geração de caminhos e no conceito de segmento, pode-se formular o teorema abaixo, provado por [1] e por [2], base para o teste de planaridade, que restringe os critérios de verificação aos vértices inicial e final de cada caminho:

TEOREMA 2.2: Seja **p**, de $\mathbf{v_i}$ (o vértice base de S) para $\mathbf{v_j}$ (outro vértice em C), o primeiro caminho do segmento corrente S. Se todos os segmentos gerados antes de S já foram localizados no plano, então **p** pode ser colocado dentro de C se e somente se não existir nenhuma aresta de retorno (x,v_t) previamente desenhada por dentro que satisfaça $v_j < v_t < v_i$.

A Fig 2.33 ilustra os três casos de impedimento analisados por [1] na demonstração do teorema; nela, a aresta (x,v_t), final de um caminho $\mathbf{p_1}$ gerado antes de **p**, não permite colocar **p** dentro de C.

Fig. 2.33



Usando o teorema 2.2, o teste de planaridade de Hopcroft-Tarjan pode ser resumido da seguinte forma:

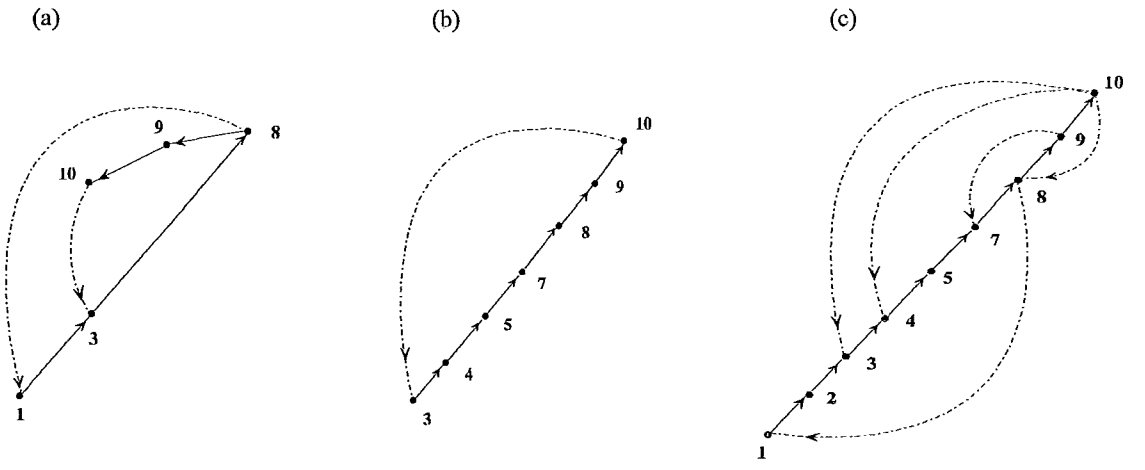
- inicialmente coloca-se o ciclo no plano;
- para cada segmento S a posicionado no plano:
  - acha-se um caminho **p** de **s** para **f**;
  - se não existe aresta de retorno (x,vt) previamente desenhada, tal que $f < vt < s$, então coloca-se **p** dentro de C;
  - se existem arestas de retorno impeditivas só por dentro, mas não por

fora de C, os segmentos anteriormente posicionados são remanejados de dentro para fora e vice-versa; coloca-se **p** dentro de C;

- se existem arestas de retorno entrando por dentro e por fora de C, que impedem a inclusão de **p**, e o remanejamento dos segmentos de dentro para fora de C não resolver, então G não é planar, ou seja, não é possível localizar o caminho **p** juntamente com o ciclo C sem cruzamento de arestas.

Na Fig.2.34a, estão desenhados no plano o ciclo C e o primeiro caminho do segmento S_1 (ver Fig.2.32b), este, dentro de C; observar que o vértice final de p_1 é 3 e não existe nenhuma aresta de retorno entrando em C entre 3 e 8, o vértice inicial de p_1.

Fig. 2.34



Essa idéia, a princípio simples, é trabalhada de forma recursiva, sempre que um segmento for composto por mais de um caminho. Isto porque, para adicionar o primeiro caminho ao desenho planar de G, o teorema 2.2 pode ser aplicado diretamente, o que não acontece ao se tentar incluir o segundo; neste caso, o ciclo de referência para o teste é outro, pois, certamente, uma ou mais arestas de árvore foram posicionadas juntamente com o primeiro caminho. Assim, num passo intermediário, após desenhar o ciclo C, todos os segmentos anteriores a S e o primeiro caminho **p** do segmento S, é necessário saber se o resto do segmento (S-p) pode ser adicionado ao desenho já existente.

Fica-se, então, frente ao teste de planaridade de $G' = S \cup C$. O novo ciclo inicial C' de G' é formado pelo caminho **p** e pela porção de C que liga **f** até **s**. Removendo o ciclo C' de G', o grafo G'-C' deverá ser desmembrado em segmentos, cuja embutidura será testada relativamente ao ciclo C'. Naturalmente, se um segmento relativo ao ciclo C' contiver mais de um caminho, um novo nível de recursão será utilizado.

Uma vez posicionado o segmento S (com relação ao ciclo C inicial), fica

faltando verificar se a porção de C, não incluída em C', pode ser adicionada ao conjunto; o teorema 2.2 é utilizado para esta análise.

Para posicionar o caminho p_2 do segmento S_1 da Fig.2.32b, o algoritmo usa o procedimento de recursão descrito. O ciclo C' achado é 3,4,5,7,8,9,10,3, composto pelo caminho p_1=8,9,10,3 e pela porção de C que liga **f**=3 a **s**=8 (ver Fig.2.34b). O grafo restante consiste de seis arestas, das quais três são parte de C ((1,2),(2,3) e (8,1)) e três ((10,4),(10,8) e (9,7)) são arestas de retorno, cada uma constituindo um segmento relativamente a C'. Não é necessário outro nível de recursão: as arestas de retorno são localizadas e, depois, verifica-se a porção de C restante. A Fig.2.34c apresenta um desenho de $S_1 \cup C$.

Observar que para posicionar a aresta (9,7), esta o último caminho de S_1, foi necessário remanejar a aresta (10,8) de dentro para fora de C'.

Para se constatar que todo o segmento S_1 foi localizado dentro (ou à esquerda) de C, usa-se a seguinte definição [2]: um segmento cuja primeira aresta é (v_i,w) é dito embutido **dentro** de C quando, alcançando $\mathbf{v_i}$ pela aresta (v_{i-1},v_i), a orientação das arestas que saem de $\mathbf{v_i}$, na ordem dos ponteiros do relógio, é $(v_i,w),(v_i,v_{i+1})$; caso contrário, quando a ordem é (v_{i-1},v_i), (v_i,v_{i+1}), (v_i,w), o segmento é dito embutido **fora** (ou à direita) de C.

Na Fig.2.34c, a primeira aresta de S_1 é (8,9); a ordem das arestas ao redor do vértice 8 é (7,8), (8,9), (8,1) e, portanto, S_1 está dentro de C.

2.3.3- Detalhes de Implementação

Para implementar o teste de planaridade conforme o método descrito, foram criadas estruturas de dados que mantêm informações sobre os vértices que estão no caminho de arestas de árvore de 1 a $\mathbf{v_i}$ (quando se quer localizar um segmento que inicia em $\mathbf{v_i}$) e que têm arestas de retorno entrando por dentro e/ou por fora dele. Definem-se, então:

- uma pilha L, que armazena os vértices cujas arestas de retorno entram por dentro de C ou na sua esquerda;
- uma pilha R, que, analogamente à primeira, armazena os vértices cujas arestas entram por fora de C ou na sua direita;
- uma pilha B que mantém pares (x,y) representando os blocos: um **bloco** é um conjunto maximal de entradas nas pilhas L e R que corresponde a arestas de retorno,

tais que a localização de qualquer uma delas determina a localização de todas as outras. No par (x,y), **x** aponta para a última entrada do bloco em L e **y** aponta para a última entrada do bloco em R. Se **x** = 0, o bloco não tem entradas em L; analogamente para **y**.

As pilhas L e R são atualizadas de quatro maneiras:

(i)- quando todos os segmentos que se iniciam em um vértice $\mathbf{v_i}$ tiverem sido explorados e localizados no plano, todas as ocorrências maiores ou iguais a $\mathbf{v_i}$ em L e R devem ser eliminadas, pois nenhum segmento ainda não explorado vai começar em um vértice maior do que $\mathbf{v_i}$;

(ii)- seja o caminho **p**, com vértice inicial **s** e final **f**; define-se: se **s** está em C, **p é normal** se e só se **f**>1. Se **p** é o primeiro caminho de um segmento S, e **p** é normal, **f** tem que ser adicionado a uma das pilhas;

(iii)- a aplicação recursiva do algoritmo deve adicionar entradas para os demais caminhos de S;

(iv)- as entradas de uma pilha devem ser movidas de uma para outra quando os correspondentes segmentos forem movimentados. O posicionamento de uma aresta de retorno à esquerda força a colocação à esquerda de todas as demais arestas de retorno deste mesmo segmento, e pode forçar arestas de retorno de outros segmentos a ficarem à direita.

A seguir, a movimentação das pilhas para a localização de S_1 (ver Fig.3.32b), cujo vértice base é 8, no ciclo C:

.p_1= 8,9,10,3	*.p_2= 10,4*
L: 3	L: 3,4
R:	R: 0
B: (0,0),(3,0)	B: (0,0),(3,0),(4,0)

.p_3=10,8	*.p_4= 9,7*
L: 3,4,8	L: 3,4,7
R: 0	R: 0,8
B: (0,0),(3,0),(4,0),(8,0)	B: (0,0),(3,0),(4,0),(7,8)

Observar que ao tentar localizar o segundo caminho de S_1, uma *marca de fim* (zero) é colocada na pilha L, para controle dos níveis de recursão. Abaixo alguns resultados do algoritmo HT no retorno da recursão de S_1; as entradas maiores ou iguais a 8 são eliminadas:

L: 3,4,7
R:
B: (0,0),(3,0)

Continuando o procedimento, posiciona-se S_2, que é o caminho $p_5=7,4$, sem impedimento, dentro de C; o vértice base de S_2 é 7 e, portanto, os vértices maiores ou iguais a ele são retirados da pilhas antes de processá-lo:

L: 3,4,4

R:

B: (0,0),(3,0),(4,0)

A localização de S_3 exige recursão; além disso, para serem posicionados dentro de C, os segmentos S_1 e S_2 deverão ser remanejados de dentro para fora de C e vice-versa. O vértice base de S_3 é 5:

.p_6= 5,6,2	*.p_7= 6,3*	*.retorno recursão S_2*
L: 2	L: 2,3	L: 2,3
R: 3,4,4	R: 3,4,4,0	R: 3,4,4
B: (0,0),(2,3)	B: (0,0),(2,3)	B: (0,0),(2,3)

O último segmento S_4 é o caminho $p_8=5,3$, uma aresta de retorno, e não há impedimento para localizá-la:

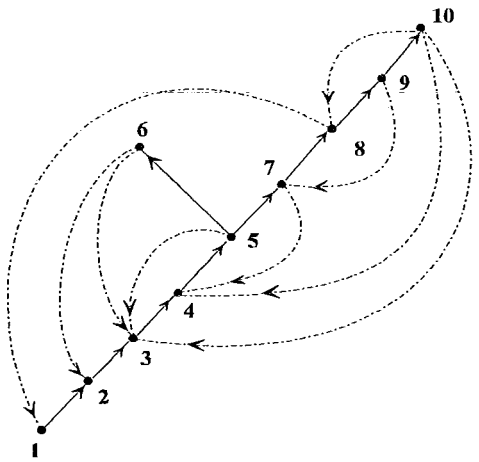
L: 2,3,3

R: 3,4,4

B: (0,0),(2,3)

A Fig.2.35 mostra um desenho planar do grafo da Fig.2.31, que é planar, pois chegou ao término do algoritmo com sucesso. As pilhas finais mostram a distribuição dos vértices ao redor do ciclo C.

Fig. 2.35



O algoritmo de Hopcroft-Tarjan é descrito, abaixo, organizado conforme [2]; a rotina PLANAR tem estrutura semelhante à da rotina CAMINHOS, acrescida do teste

de planaridade.

```
procedimento LOCALIZA:
{ Reconhecimento da Planaridade de um Grafo }
começar
 . procedimento PLANAR (v);
   começar
    . para w ∈ Padj(v), fazer
      começar
          . se s = 0
            então   { v é vértice inicial de um novo caminho }
            começar
              . iniciar novo caminho de número p=p+1;
              . seja v:= s, o vértice inicial do caminho
                           de número p;
            fim;
          . se v<w
            então  { (v,w) é aresta de árvore, com v pai de w}
            começar
              . seja path(w):= p, o número do caminho onde se
                                  encontra o vértice w;
              . PLANAR(w);
              . A;  { eliminar entradas das pilhas L, R e B,
                       que correspondem a vértices ≥ v }
              . se path(w) diferente path(v)
                então   {todos os caminhos do segmento com
                      primeira aresta (v,w) estão localizados}
                  . B;   { posicionar resto de C relativo ao
                           ciclo C'; verificar blocos de B e
                           movê-los da esquerda para a direita}
            fim
            senão        { (v,w) é aresta de retorno; caminho
                           de número p está completo }
            começar
              . seja f(p):= w, o vértice final do caminho p;
              . enquanto o topo de L > w, fazer
                  . C; {remanejar pilhas L e R para posicionar
                        caminho de número p por dentro }
              . se w > f(path(s))
                então       { w é maior que vértice final
                              do caminho iniciado em s}
                  . D;      { adicionar w à pilha L}
              . E;     { atualizar pilha B com o posicionamento
                       do caminho corrente terminado com (v,w)}
              . F;     { se o caminho corrente é mais que uma
                         aresta de retorno, adicionar uma marca
                         de fim à pilha R }
              . s:= 0;
      fim
   fim PLANAR;

 { Inicializações }
 . seja G(V,E) biconexo;
 . definir pilhas L, R, B;
 . inicializar pilhas L, R e B vazias;
 . PLANAR(1);
fim LOCALIZA;
```

Observações:

- **p** é o número do caminho corrente;
- path(v) contém o número do caminho onde se encontra o vértice **v**;
- f(p) contém o vértice final do caminho número **p**;
- ao invés de implementar as pilhas L e R, o algoritmo define dois vetores STACK e NEXT e os manipula de forma interligada; o objetivo é evitar eliminar e/ou acrescentar, desnecessariamente, elementos às pilhas, mas somente remanejar seus apontadores.

Resumo dos subprocedimentos da rotina PLANAR:

- **A:** ao retornar da recursão que posicionou um segmento S, com vértice base **v**, os vértices maiores que **v** podem ser eliminados das pilhas, porque nenhuma aresta de retorno que entra no ciclo C, em um vértice **v** ou acima dele, pode interferir no posicionamento de um segmento que começa em **v** ou abaixo dele (teorema 2.2); como o segmento em processamento tem vértice base **v**, o próximo terá vértice base menor ou igual a **v**.

- **B**: quando todo o segmento S (com respeito ao ciclo C) tiver sido posicionado com sucesso, verifica-se se porção de C, não incluída em C', pode ser adicionada ao desenho planar de $S \cup C'$, isto é, se existem entradas em ambos os lados do ciclo C' que impedem o posicionamento completo de C. Se for verdade, então G é não planar. Senão, remanejam-se as pilhas para localizar o restante de C.

- **C:** este passo é executado enquanto as pilhas L e R são não vazias e seus topos interferem com o posicionamento do caminho corrente de número **p**, que termina na aresta de retorno (v,w). Se não for possível remanejar os segmentos já localizados para colocar o caminho **p** por dentro do ciclo C, isto é, se ambos os topos das pilhas forem maiores que **w**, então G é não planar. Se somente o topo de uma das pilhas for maior que **w**, remanejam-se as entradas das pilhas L e R.

- **D**: o vértice **s** é o vértice inicial do caminho corrente **p** e path(s) é o caminho mais antigo que contém **s**. A menos que **p** seja o ciclo inicial, path(s) é menor que **p**; portanto **w**, que é o vértice final do caminho **p**, só é incluído em L se **w** for maior que o vértice final de path(s).

- **E**: atualiza a pilha B para refletir o posicionamento do caminho corrente. Esta atualização é requerida somente quando **w** foi adicionado à pilha L, ou quando os blocos foram remanejados no procedimento **C**, ou quando o segmento inteiro que contém o caminho corrente consiste de mais de uma aresta de retorno.

- **F:** se o segmento que contém o caminho corrente é mais que uma aresta de retorno, é necessário utilizar um nível de recursão para posicioná-lo; para controle, o algoritmo coloca uma marca de fim (zero) no topo da pilha R.

Hopcroft e Tarjan provaram que seu algoritmo LOCALIZA testa corretamente a planaridade de um grafo G em um tempo O(n + m) com o seguinte teorema [1].

TEOREMA 2.3: O algoritmo de planaridade requer tempo O(n) para testar um grafo.

Para realizar a demonstração deste teorema, [1] observa que o algoritmo termina se o número de arestas de G exceder 3n-3; então, o tempo para contar arestas é O(n). Se G tem O(n) arestas, a busca em profundidade inicial requer tempo O(n), a reorganização das listas de adjacências requer tempo O(n) e o algoritmo de geração de caminhos, juntamente com a algoritmo LOCALIZA, também. Então, o tempo total requerido é O(n).

Apesar de Hopcroft-Tarjan terem afirmado que seu algoritmo de teste de planaridade poderia, com pequenas alterações, obter a embutidura de um grafo G, as tentativas realizadas não alcançaram sucesso.

2.4- O Algoritmo de Booth e Lueker

O algoritmo de Booth-Lueker [4], aqui descrito para testar a planaridade de um grafo, foi definido de forma mais genérica, também para testar propriedades de matrizes ou reconhecer grafos de intervalo.

As bases desse algoritmo são a **adição de vértices** e uma **operação de redução**, praticamente idêntica à formulada por Lempel-Even-Cederbaum, em 1967.

A inovação implementada por Booth-Lueker é uma estrutura de dados chamada **árvore-PQ** (ver DEF.2.4), usada para representar as fórmulas criadas por Lempel-Even-Cederbaum e poder manipulá-las de forma eficiente.

Booth e Lueker atingiram um limite de tempo e espaço linear na sua proposta, o que Lempel-Even-Cederbaum não conseguiram.

O algoritmo de BL é definido para grafos biconexos, pelas mesmas razões do de Hopcroft-Tarjan (ver seção 2.3); o algoritmo exige, também, que se calcule uma rotulação-st (ver seção 2.2) para o grafo, antes de submetê-lo ao procedimento.

A idéia deste teste de planaridade é representar iterativamente um grafo G por árvores-PQ. O conjunto universo U é, a cada passo, um subconjunto de arestas do

grafo dado e, o conjunto de restrições S (ver DEF.2.10), um subconjunto de arestas de U que têm o mesmo vértice extremo. O objetivo deste procedimento é espelhar em P-nós e Q-nós a obrigatoriedade dos agrupamentos de arestas incidentes a um mesmo vértice.

2.4.1- Noções Básicas

Seja G=(V,E) um grafo biconexo e st-rotulado com números de 1 a **n** = |V|.

O processo se inicia no vértice **v** tal que num-st(v)=1 e vai agrupando arestas com mesmo vértice extremo, *reduzindo* a árvore-PQ e acrescentando outras arestas com vértice inicial no extremo anterior, até o processamento do último vértice ou a comprovação da não planaridade do grafo.

Uma árvore-PQ construída durante o teste de planaridade de [4], representa, nos seus P-nós, Q-nós e folhas, a parte do grafo em análise no decorrer de uma iteração do procedimento. As folhas dessa árvore-PQ são um subconjunto de arestas do grafo. Os P-nós e Q-nós, vértices internos da árvore, caracterizam a organização permitida para os seus filhos, que podem ser P-nós, Q-nós ou folhas.

Em cada passo do algoritmo, as folhas da árvore-PQ, que representam o conjunto U, são um subconjunto das arestas do grafo, ainda não analisadas. Mais especificamente, num passo **k**, os vértices **v** com num-st(v) $\leq$ **k** já foram processados; portanto, somente arestas incidentes a vértices **v**, tais que num-st(v) $>$**k**, constam do conjunto U e, exatamente aquelas incidentes a **k**, de num-st(v)=k+1, formam o conjunto S de restrições.

À árvore-PQ do passo **k** do algoritmo são aplicadas transformações de equivalência (ver DEF.2.8), isto é, permutações dos P-nós e reversões dos Q-nós, com a finalidade de agrupar as folhas incidentes ao vértice **v**, tais que num-st(v)=**k+1**, obtendo, para elas, uma seqüência consecutiva que venha a possibilitar a redução da árvore. Se, em algum passo do procedimento, não se conseguir agrupar os vértices extremos correspondentes à iteração ou, se obtido isto, a redução não for possível, então G não é planar.

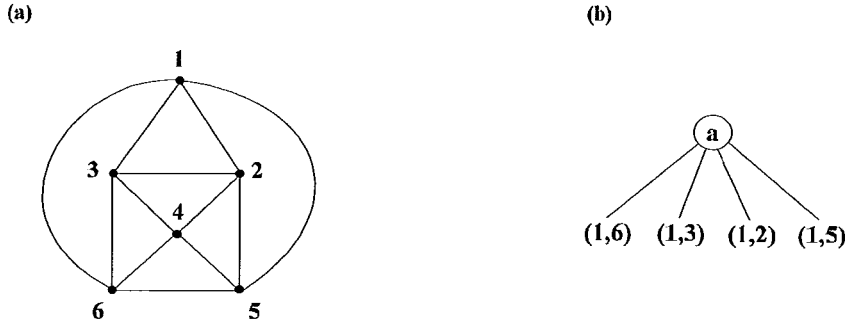
2.4.2- Descrição do Algoritmo

O algoritmo de Booth-Lueker está descrito no **procedimento PLANAR** que recebe um grafo G(V,E), biconexo e com rotulação-st de 1 a **n.**

O primeiro passo do algoritmo identifica as arestas que partem do vértice 1. Uma árvore-PQ é construída, tendo como raiz um P-nó e como folhas as arestas incidentes ao vértice 1; estas constituem o conjunto universo inicial.

O algoritmo pode ter melhor visualização utilizando um exemplo. A Fig.2.36 apresenta um grafo G, biconexo e st-rotulado, e a árvore-PQ construída no primeiro passo do algoritmo; suas folhas são as arestas incidentes ao vértice de rótulo 1. Seu conjunto universo é U = {(1,6), (1,3), (1,2), (1,5)}.

Fig. 2.36



A seguir, um procedimento iterativo de j=2 até n-1 identifica as folhas cheias, que são as arestas incidentes ao vértice **v** tal que num-st(v)=**j** e formam o conjunto S de restrições; a subárvore pertinente (ver DEF.2.12) é determinada e suas folhas cheias agrupadas numa seqüência consecutiva, se possível. Tenta-se, então, *reduzir* a árvore-PQ. No passo **j**=2, para o grafo da Fig.2.36, o conjunto de restrições é S={(1,2)}.

Se a redução for obtida com sucesso, as folhas cheias são substituídas por uma subárvore cuja raiz é um P-nó e cujos filhos são as arestas que partem do vértice **j** e apontam para outros de maior valor na rotulação-st. O conjunto U é atualizado para conter estas novas arestas e dele são eliminadas as arestas do conjunto S já processado.

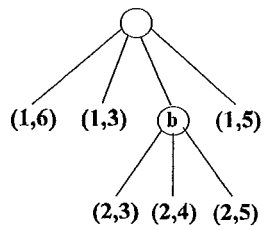
O procedimento de iteração se encerra em **j=n-1**, quando a árvore-PQ final só tem como folhas as arestas que apontam para o vértice de rótulo **n**, caso em que a redução foi bem sucedida e o grafo é planar.

A árvore-PQ da Fig.2.36, construída no passo 1 do algoritmo PLANAR, é a árvore analisada pelo passo **j**=2; ela tem somente uma folha cheia, (1,2). Após reduzi-la, o que, neste caso, consiste em *marcar* a folha como cheia, o procedimento substitui a folha (1,2) por um P-nó cujos filhos são as arestas que partem do vértice de rótulo 2 em

direção a outros rótulo maior do que 2 (ver Fig.2.37). O conjunto U é recalculado para U={(1,6),(1,3),(2,3),(2,4),(2,5),(1,5)}; o novo conjunto de restrições é S={(1,3), (2,3)}.

A essência do algoritmo PLANAR está na rotina **REDUÇÃO**, que é detalhada antes do mesmo; ela é a operação básica do teste de planaridade.O objetivo da operação de redução (ver DEF.2.11) é, dada uma árvore-PQ, com seus nós cheios já em uma seqüência consecutiva, obter uma outra com mesma fronteira (ver DEF.2.7), na qual a seqüência consecutiva dos nós cheios seja definitiva, isto é, possa ser expressa pela estrutura de P-nós e Q-nós da árvore. Para atingir esse objetivo são usados os moldes, constituídos de duas partes: o padrão de casamento e a substituição correspondente. São onze os moldes definidos por Booth-Lueker e eles estão detalhados na seção 2.2, casos de 1 a 11.

Fig. 2.37



Dada a árvore, procura-se um padrão no qual ela possa ser encaixada e realiza-se a substituição correspondente, obtendo uma árvore-PQ reduzida. Se, para a árvore-PQ, não se encontrar nenhum padrão é porque ela não é redutível. A única restrição imposta pelo casamento dos moldes é que todos os filhos de um nó sejam reduzidos antes do pai, porque o molde adequado para um nó só pode ser selecionado após ser conhecida a forma reduzida dos nós filhos. Assim, o processamento se inicia pelas folhas, que são reconhecidas como cheias ou vazias, conforme pertençam ou não ao conjunto S; depois, os moldes são aplicados aos pais das folhas, que já são nós internos, e, sucessivamente, aos pais destes.

O procedimento de redução usa uma fila para armazenar os elementos do conjunto U, que são as folhas da árvore-PQ em análise (T).

A primeira providência da rotina é colocar as folhas na fila, uma vez que o procedimento é realizado de baixo para cima. Feito isto, elas são comparadas com os moldes. Estes são triviais para folhas: consistem somente em marcá-las como cheias ou vazias, conforme pertençam ou não ao conjunto de restrições S, que é o subconjunto de U cujos elementos são as arestas incidentes ao vértice em análise. A cada vez que todos os filhos de um P-nó ou Q-nó forem reduzidos, o pai (P-nó ou Q-nó) é colocado na fila.

As folhas são as entradas iniciais da fila; ao serem selecionadas para análise suas entradas são eliminadas. Ao fim das folhas, e de acordo com a sua ordem de entrada na fila, os pais são analisados e reduzidos, conforme os moldes disponíveis. Seguindo a estratégia de baixo para cima, quando todos os filhos tiverem sido substituídos de acordo com o padrão, o pai, se existir, é colocado na fila, para um próximo passo do procedimento. Isto se repete até que:

- se constate que não existe molde adequado para ser aplicado. Neste caso o procedimento principal (PLANAR) informa que o grafo não é planar;
- se constate que S já está agrupado, ou seja, todos os elementos de S formam um subconjunto do nó que terminou de ser analisado; neste ponto, a árvore T da iteração está reduzida.

Voltando à analise da árvore da Fig.2.37, com os conjuntos U e S já definidos, e identificadas as folhas cheias e vazias, observa-se que a subárvore enraizada no P-nó **b** corresponde ao molde P3 (ver caso 4 da seção 2.2), pois sua raiz não é a raiz da subárvore pertinente (ver DEF.2.12) e ela tem filhos cheios e vazios. A Fig.2.38 mostra a aplicação do molde P3 (padrão e substituição) ao P-nó **b**; a subárvore reduzida não é própria (ver DEF.2.6), mas como a redução ainda não terminou, ela não é alterada.

Fig. 2.38

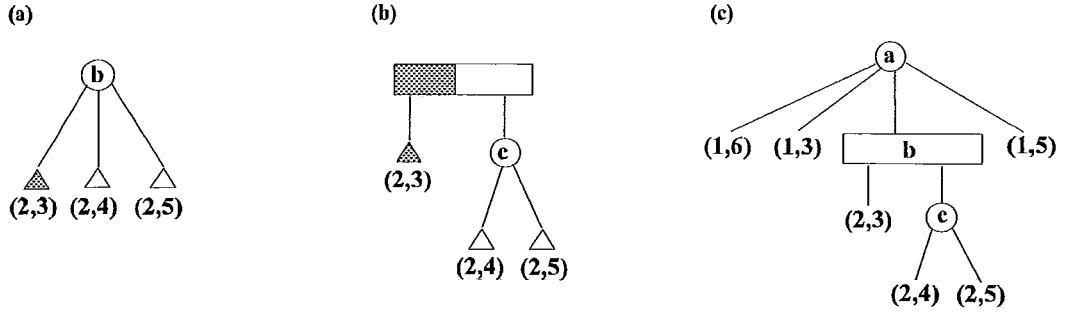
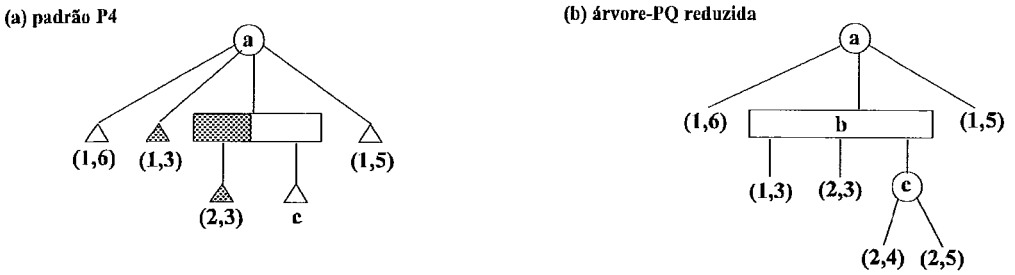


Fig. 2.39



A Fig.2.39 apresenta a continuação do procedimento; o nó em análise é,

agora, o P-nó **a**, cujos filhos, que são as folhas (1,6), (1,3) e (1,5) e o Q-nó **b**, já foram analisados. O molde que se adapta é P4; ele é aplicado e a redução deste passo está finalizado, pois o P-nó **a** é a raiz da subárvore pertinente. A árvore-PQ reduzida é mostrada na Fig.2.39.

Utilizando a fila definida pelo procedimento REDUÇÃO, é possível acompanhar as etapas descritas acima; inicialmente tem-se:

Fila = U= {(1,6),(1,3),(2,3),(2,4),(2,5),(1,5)}.

As folhas (1,3) e (2,3) são cheias e as demais, vazias; uma a uma são retiradas da fila. Todos os filhos do P-nó **b** estão analisados; então, ele é colocado na fila:

Fila = {P-nó **b**}.

Retira-se, agora, o P-nó **b** da fila; ele se adapta ao molde P3 e a redução é processada. O P-nó **a** é colocado na fila e, em seguida, selecionado e reduzido com a aplicação do molde P4.

A fila está vazia e o processamento termina.

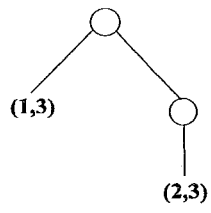
A implementação da rotina REDUÇÃO, conforme a descrição dada, percorre a árvore duas vezes, ambas a partir das folhas em direção à raiz. Para diminuir o seu custo foi criada a rotina **BOLHA**, que otimiza o procedimento da rotina REDUÇÃO, propiciando que o algoritmo BL atinja um tempo de execução linear.

Apesar de BOLHA não ser um procedimento trivial, ele é quase transparente como atuação. A única diferença visível para a rotina REDUÇÃO é que o conjunto de entrada conterá *somente os elementos de S* e, não mais todo o conjunto U. Assim, a redução pode trabalhar somente com nós cheios, prescindindo, inclusive, de transformações de equivalência para agrupá-los previamente. O procedimento BOLHA usa a seguinte definição:

- **subárvore pertinente aparada de T com respeito a S** como o menor subgrafo conexo que contém todas as folhas pertinentes, isto é, todos os elementos de S.

(DEF.2.13)

Fig. 2.40



A Fig.2.40 ilustra a subárvore pertinente aparada referente à Fig.2.37; como se pode ver, nem sempre esta será uma árvore-PQ própria.

A rotina BOLHA, para atingir seu objetivo, procura fazer *emergir* os nós cheios, construindo a subárvore pertinente aparada e repassando-a para a rotina REDUÇÃO. Os nós vazios deverão ser reconhecidos por sua ausência; mas, mesmo não querendo processar os nós vazios, é necessário manter controle sobre os seus ponteiros pais. A atualização destes, porém, pode gerar um trabalho excessivo, como, por exemplo, no seguinte caso: se um nó interno (não folha) tem um grande número de filhos vazios e ele é eliminado como parte da substituição dos moldes escolhidos, todos os filhos, inclusive os vazios, terão que receber um novo ponteiro pai. A saída eficiente é observar que só há necessidade de manter ponteiros pais para os filhos dos P-nós e para os filhos extremos dos Q-nós, pois os filhos interiores destes podem consultar os irmãos para saberem quem são seus pais. Para este controle, criam-se os conceitos de **nó bloqueado**, que é o nó interior de um Q-nó sem ponteiro pai, e de **bloco de nós bloqueados**, que é uma cadeia máxima da esquerda para a direita de irmãos bloqueados filhos de um mesmo Q-nó; a idéia é que, usando a estratégia dos blocos, cada nó pertinente tenha um ponteiro pai válido no início do passo de redução.

A rotina BOLHA manipula, através de contadores, as informações de número de nós bloqueados e de número de blocos de nós bloqueados. Ainda um outro contador guarda, em cada nó pai, quantos filhos existem para ser processados. Este é o contador repassado para a rotina REDUÇÃO que permite reconhecer quando o último filho pertinente de um nó foi processado e possa enfileirar corretamente o pai. A rotina BOLHA repassa também a subárvore pertinente aparada, que só contém nós cheios, fazendo com que o processo de redução prescinda de qualquer transformação de equivalência.

As duas rotinas descritas, básicas para o algoritmo de teste de planaridade de Booth-Lueker, podem ser resumidas, então, como a seguir:

- a rotina BOLHA identifica os nós a serem processados num passo do algoritmo, construindo e repassando a subárvore pertinente aparada para a rotina seguinte;

- a rotina REDUÇÃO seleciona os moldes a aplicar aos nós da subárvore pertinente aparada e realiza as respectivas substituições.

2.4.3- Detalhes de Implementação

O algoritmo BL é descrito nesta seção; ele é composto pelo procedimento PLANAR(G), que realiza o teste de planaridade de um grafo **G** biconexo, com vértices st-rotulados de 1 a **n**. Ele chama a rotina REDUÇÃO(T,S), que executa a redução da árvore-**PQ T** com respeito ao conjunto **S** de restrições, este contendo arestas com mesmo vértice extremo. Um exemplo é utilizado para melhor visualização da execução do algoritmo BL, sem utilizar a rotina BOLHA.

```
Procedimento REDUÇÃO(T,S);
{ redução de árvore-PQ T com respeito ao subconjunto S }
começar
  . inicializar Fila vazia;
  . para cada folha x de S, fazer
     começar
       . colocar x na Fila;
       . folhas_pertin(x) := 1;
     fim;
  . enquanto Fila não vazia, fazer
     começar
       . remover x do início da Fila;
       . se folhas_pertin(x) < |S|
         então
          começar      { neste caso x não é raiz de (T,S) }
            . y := pai(x);
            . folhas_pertin(y) := folhas_pertin(y) +
                                  folhas_pertin(x);
            . filhos_pertin(y) := filhos_pertin(y) - 1;
            . se filhos_pertin(y) = 0
              então  {todos os filhos de y foram analisados}
                     colocar y na Fila;
            . se molde_L1(x) não se aplica
              então se molde_P1(x) não se aplica
                então se molde_P3(x) não se aplica
                  então se molde_P5(x) não se aplica
                    então se molde_Q1(x) não se aplica
                      então se molde_Q2(x) não se aplica
                            então  { redução impossível }
                                   retornar T vazia;
          fim
         senão
          começar      { neste caso x é a raiz de (T,S) }
            . se molde_L1(x) não se aplica
              então se molde_P1(x) não se aplica
                então se molde_P2(x) não se aplica
                  então se molde_P4(x) não se aplica
                    então se molde_P6(x) não se aplica
                      então se molde_Q1(x) não se aplica
                        então se molde_Q2(x) não se aplica
                          então se molde_Q3(x) não se aplica
                                então  { redução impossível}
                                       retornar T vazia;
          fim;
```

```
  . retornar T reduzida;
fim REDUÇÃO;
```

Observações:

- o molde L1, não definido nos casos da seção 2.2, lida somente com as folhas cheias; sua substituição consiste em assinalá-las como cheias.

- os moldes P0 e Q0 não são utilizados, pois a rotina REDUÇÃO só processa nós cheios, conforme definição da rotina BOLHA; assim, somente nove dos onze moldes definidos inicialmente por BL, são implementados por esta rotina .

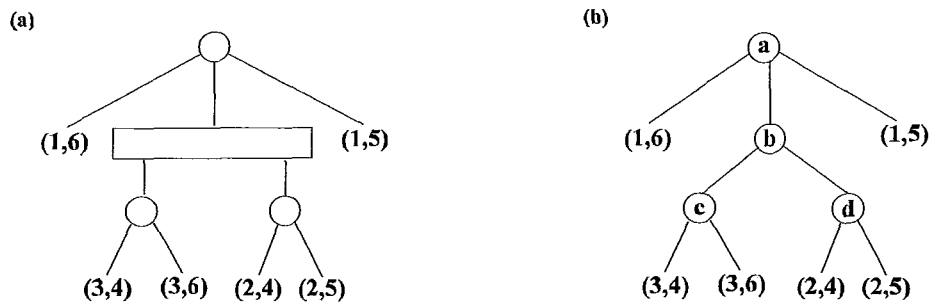
```
Procedimento  PLANAR(G);
{ Teste de planaridade de um grafo G(V,E) biconexo e st-rotulado com
valores de 1 a n }

começar
  . seja G(V,E);
  { construção da primeira árvore T }
  . U:= conjunto de arestas cujo menor vértice é 1;
  . T:= árvore-PQ enraizada em um P-nó e que tem como folhas
       as arestas que partem do vértice 1;
  . para j:= 2 até n-1, fazer
      começar
        { passo de redução }
        . S:= conjunto de arestas cujo maior vértice é j;
        . BOLHA(T com respeito a S);
        . REDUÇÃO(T com respeito a S);
        . se não há moldes a aplicar
          então    { redução falhou }
                retornar "G não é planar";
        { passo de adição de vértices }
        . S' := conjunto de arestas cujo menor vértice é j;
        . se raiz de T com respeito a S é um Q-nó
          então substituir os filhos cheios da raiz de T e
                seus descendentes por um P-nó, cujas folhas
                são os elementos de S';
          senão substituir a raiz de T e seus descendentes
                por um P-nó, cujas folhas são os elementos
                de S';
        { recálculo do conjunto U }
        . U := U - S + S';
      fim;
  . retornar "G é planar";
fim PLANAR;
```

As Figuras 2.41, 2.42, 2.43 e 2.44 ilustram a aplicação do procedimento PLANAR ao grafo da Fig. 2.36.

A árvore-PQ da Fig.2.39b corresponde ao término do procedimento de redução da iteração j=3 do algoritmo; a fase de adição de vértices a recebe como

entrada. Neste ponto, o conjunto S' é calculado como {(3,4),(3,6)} e as folhas (1,3) e (2,3) são substituídas por um P-nó cujos filhos são os elementos de S'. O resultado é a árvore-PQ da Fig.2.41a; como ela não é própria, ela é transformada na da Fig.2.41b, que é a árvore T de entrada para a iteração **j**=4. O conjunto U é recalculado ao final da iteração **j**=3, obtendo-se U = {(1,6),(1,5),(2,4),(2,5),(3,4),(3,5)}.

Fig. 2.41



O conjunto S é {(3,4),(2,4)}, calculado no início da iteração **j**=4. A raiz da subárvore pertinente é o P-nó **b** (ver Fig.2.41b) mas, primeiramente, são processados os P-nós **c** e **d**, aos quais é aplicado o molde P3. A Fig.2.42a mostra as subárvores reduzidas resultantes da aplicação do molde. O P-nó **b**, que já tem seus filhos analisados, recai no molde P6; a Fig.2.42b apresenta a árvore reduzida por este passo.

Fig. 2.42

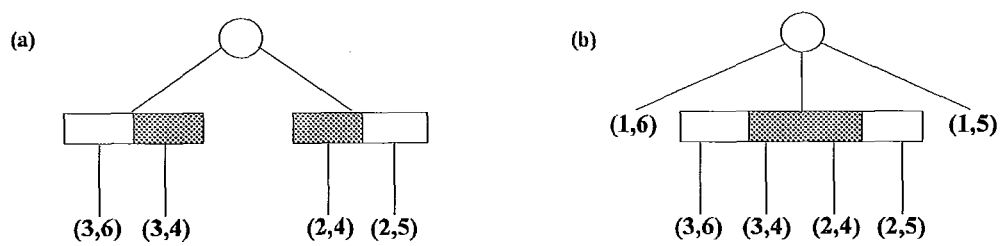
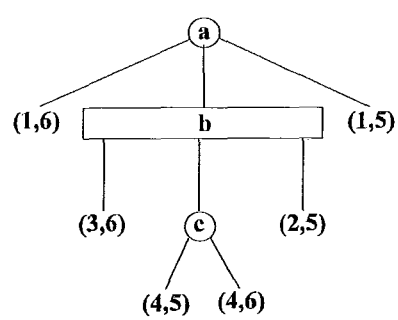


Fig. 2.43

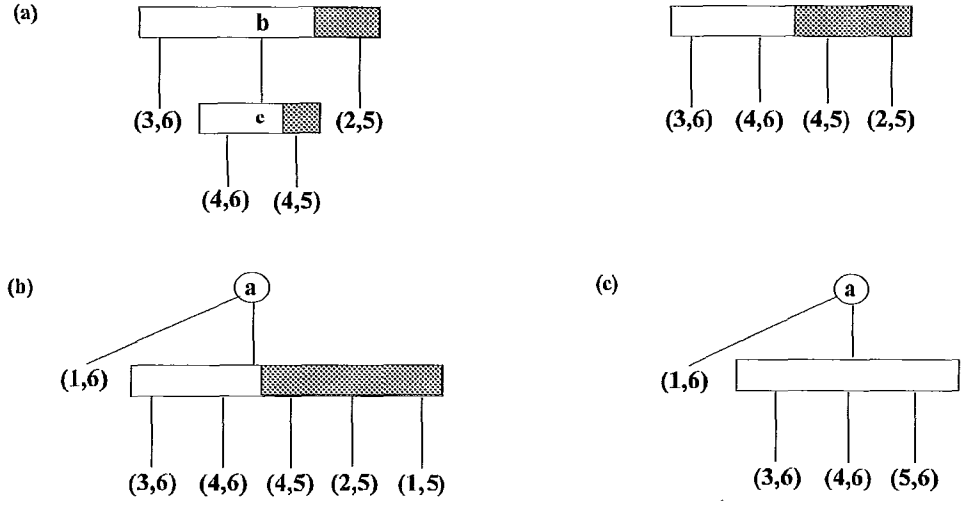


A fase de adição de vértices, para **j**=4, calcula S' = {(4,5),(4,6)} e substitui os nós cheios (3,4) e (2,4) por um P-nó cujos filhos são os elementos de S', obtendo a

árvore da Fig.2.43, que é a entrada T do passo **j**=5; o conjunto U é recalculado para {(1,6), (1,5), (2,5), (3,6), (4,5), (4,6)}.

O início da iteração **j**=5 calcula S={(1,5), (2,5),(4,5)}. O molde P3 se aplica ao P-nó **c** e o molde Q2 ao Q-nó **b**; esta redução intermediária é mostrada na Fig.2.44a. Finalmente, aplica-se o molde P4 sobre o P-nó **a**, que é a raiz de T, obtendo a árvore da Fig.2.44b.

Fig. 2.44



A etapa de adição de vértices completa a iteração **j**=5 e, também, o algoritmo, calculando S'={(5,6)} e substituindo as folhas (4,5), (2,5) e (1,5) pela própria folha (5,6), pois, seu P-nó pai, só teria a ela como filho.

A árvore obtida, que é a árvore-PQ final, está na Fig.2.44c. Nela, todas as folhas são arestas que apontam para o vértice **n**=6, o último vértice na rotulação-st. O procedimento, então, se completou com sucesso e pode-se concluir que o grafo é planar.

A simplicidade do algoritmo de Booth-Lueker, uma vez definidas as árvores-PQ, propiciou sua utilização em mais de um algoritmo de embutidura de grafos, o que será discutido no próximo capítulo.

O algoritmo PLANAR requer tempo e espaço O(n) para o seu processamento, conforme provado por Booth-Lueker no seu trabalho. A prova se baseia no mesmo argumento utilizado por Hopcroft-Tarjan, na demonstração do Teorema 2.3 (ver seção 2.3).

Capítulo 3

Embutidura de Grafos Planares

3.1- Introdução

A **embutidura** ou **representação planar** de um grafo é o passo seguinte ao reconhecimento da planaridade do mesmo; sua finalidade é posicionar os vértices do grafo no plano, de maneira que as arestas não se cruzem no desenho, não importando se são retas ou curvas.

O conceito de embutidura está muito próximo do de traçado, mas este último inclui também, como objetivo, tornar mais agradável a visualização do desenho, procurando, por exemplo, representar as arestas por segmentos de reta.

Seja G=(V,E), um grafo planar, com **n** vértices e **m** arestas. **Embutir um grafo no plano**, segundo Chiba e outros [8], é o mesmo que construir listas de adjacências A de G, de tal forma que, em cada A(v), $v \in V$, todos os vizinhos de **v** apareçam na ordem dos ponteiros do relógio no desenho que está sendo associado ao grafo. **(DEF.3.1)**

Os algoritmos sobre embutidura de grafos planares analisados neste capítulo são de Chiba-Nishizeki-Abe-Ozawa [8] e de Jayakumar-Thulasiraman-Swamy [12]. Ambos têm tempo linear e se baseiam no algoritmo de Lempel-Even-Cederbaum; sua implementação utiliza as árvores-PQ, já vista no capítulo anterior.

Os procedimentos definidos nos trabalhos de [8] e [12] têm como objetivo construir, no primeiro caso, as listas de adjacências A de G, onde, para cada **v**, seus vizinhos se apresentem na ordem dos ponteiros do relógio e, no segundo, uma ordenação de vértices que possibilite uma representação planar de G. Ambos supõem que G é não direcionado, simples (sem multi-arestas ou laços) e biconexo. Assumem, também, uma rotulação-st previamente calculada (ver seção 2.2), que é possível associar a qualquer grafo biconexo .

Os conceitos a seguir são utilizados para a descrição dos algoritmos.

3.1.1- Definições Preliminares

Seja $G_k=(V_k,E_k)$, $1 \leq k \leq n$, o subgrafo induzido pelo conjunto $V_k = \{1,2,....,k\}$, V_k subconjunto de V. Se $k<n$, então existe pelo menos uma aresta de G com uma extremidade em V_k e outra em $V\text{-}V_k$, pois, caso contrário, G não seria conexo.

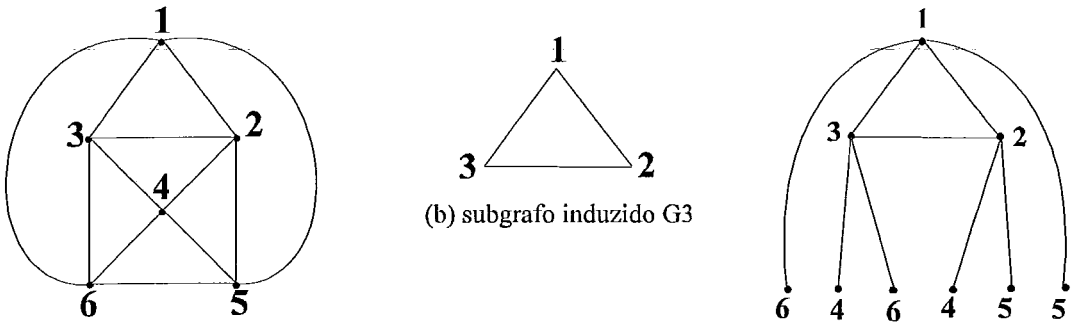
Seja G'_k o grafo decorrente da adição a G_k de todas as arestas que, partindo de V_k, são incidentes a vértices de $V\text{-}V_k$. Elas são chamadas **arestas virtuais** e suas extremidades em $V\text{-}V_k$, **vértices virtuais**. **(DEF.3.2)**

Cada vértice virtual é rotulado como sua contraparte em G e tem exatamente uma aresta entrando. Desta forma, podem existir vértices virtuais com rótulos repetidos, os quais serão mantidos separados até que, em algum passo do algoritmo, eles sejam processados.

Supondo G_k planar e G'_k desenhado no plano de tal forma que suas arestas não se interceptam (ver DEF.3.1), pode-se definir:

Seja B_k uma embutidura de G'_k, onde todos os vértices virtuais estão desenhados na face externa de G_k, em geral, posicionados numa mesma linha horizontal. B_k é chamado **forma de arbusto** de G'_k. **(DEF.3.3)**

Fig. 3.1



(a) grafo G

(b) subgrafo induzido G3

(c) forma de arbusto B3

A Fig.3.1 apresenta um grafo G biconexo e st-rotulado, um subgrafo induzido G_3 e a forma de arbusto B_3 correspondente à embutidura de G_3; os rótulos dos vértices virtuais são 4, 5 e 6 e as arestas virtuais são (1,6),(3,4),(3,6),(2,4),(2,5) e (1,5),

conforme mostra a Fig.3.1c. Observar que os vértices virtuais têm rótulos repetidos, mas cada um tem somente uma aresta entrando. Estes vértices são tais que, somente os com rótulo maior ou igual a 4 constam de B_3. E, ainda, **todos** os com rótulo igual a 4 estão presentes.

Um grafo G'_k e, portanto, sua forma de arbusto B_k, pode ser representado por uma árvore-PQ T_k (como definida na seção 2.2), desde que se faça a seguinte associação:

(i) os vértices ou as arestas virtuais de B_k são as folhas de T_k ;

(ii) os vértices de corte em B_k são representados por P-nós em T_k; quando um vértice de corte tiver um único filho, o P-nó é eliminado, ficando o filho diretamante ligado ao nó imediatamente acima, na hierarquia da árvore (ver DEF.2.6);

(iii) as componentes biconexas maximais em B_k são representadas por Q-nós em T_k.

Seja a forma de arbusto B_3 da Fig.3.1c. A árvore-PQ T_3 da Fig.3.2a, representa a forma de arbusto B_3, pois:

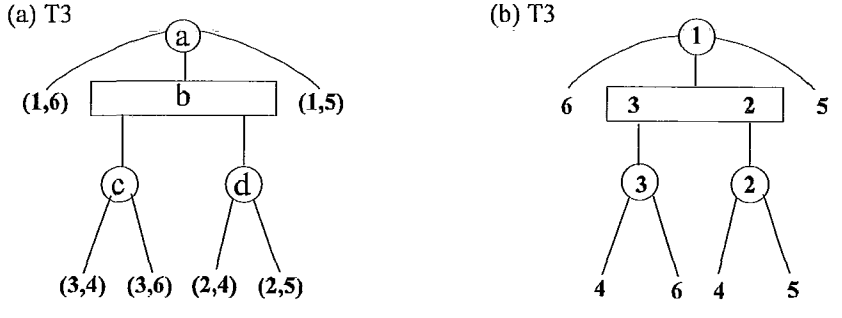
(i) as folhas de T_3 são as arestas virtuais de B_3;

(ii) sendo 1, 2 e 3 vértices de corte em B_3, eles são os P-nós **a**, **d** e **c** na árvore T_3;

(iii) a componente biconexa maximal de B_3 é o triângulo de vértices 1, 2 e 3, que é o Q-nó **b** em T_3.

As folhas da árvore T_3 também podem ser os vértices virtuais de B_3; esta representação está na Fig.3.2b.

Fig. 3.2



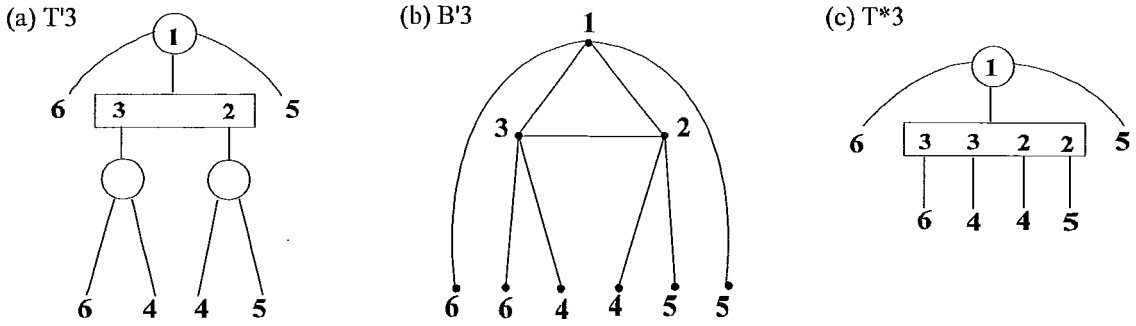
O objetivo de estabelecer uma correspondência entre B_k e T_k é a utilização do algoritmo de planaridade de Booth-Lueker, como base para os algoritmos de embutidura de [8] e [12]. Assim, ao executar a iteração j=k+1 do algoritmo PLANAR

(ver seção 2.4), a árvore T_k é a entrada do procedimento; todas as suas folhas são arestas incidentes a vértices com rótulo maior ou igual a k+1, sendo que todos os vértices **v** com num-st(v)=k+1 estão presentes. A última ação de cada passo é construir a árvore T_{k+1}, a ser processada pela próxima iteração, j=k+2.

Na árvore T_3 da Fig.3.2, todas as arestas incidentes ao vértice com rótulo igual a 4 estão presentes; ela é a árvore-PQ a ser analisada pelo passo k=3.

O procedimento de REDUÇÃO de árvores-PQ, visto na seção 2.4, consiste na aplicação de uma série de padrões e substituições, definidas por Booth-Lueker [4] e vistas na seção 2.2; ele é, então, utilizado. A partir de T_k, é possível obter uma árvore-PQ reduzida T^*_k, onde todas as folhas k+1, que são as folhas cheias (ver DEF.2.2) para o passo **k**, aparecem como filhas de um único nó. Para se chegar de T_k a T^*_k, passa-se antes por T'_k, que é uma árvore-PQ equivalente a Tk (ver DEF.2.8), na qual todas as folhas cheias são consecutivas. A Fig.3.3a apresenta a árvore-PQ T'_3, equivalente a T_3; nela as folhas cheias, que são os vértices virtuais de B_3 com rótulo 4, estão agrupadas consecutivamente.

Fig. 3.3



Seja B'_k a forma de arbusto isomorfa de B_k tal que em B'_k todos os vértices virtuais rotulados k+1 se posicionam de forma consecutiva. Prova-se, em [12], que B'_k existe se e somente se T_k puder ser convertida na árvore-PQ T'_k. Isto é também o que expressa o seguinte lema [8].

LEMA 3.1: -Seja B_k uma forma de arbusto do subgrafo G_k induzido de um grafo planar G. Então, existe uma seqüência de permutações e reversões que possibilitam que todos os vértices virtuais, denominados k+1, ocupem posições consecutivas nas linhas horizontais.

A forma de arbusto B'_3, isomorfa de B_3, onde os vértices **v** com num-st(v)=4 estão consecutivos, é ilustrada na Fig.3.3b; a árvore-PQ T^*_3 é apresentada na Fig.3.3c.

A relação entre $\mathbf{B_k}$ e $\mathbf{T_k}$ e $\mathbf{B'_k}$, $\mathbf{T'_k}$ e $\mathbf{T^*_k}$, esta última, reduzida de Tk e, portanto, com a mesma fronteira, permite que a embutidura de um grafo planar G seja acompanhada, passo a passo, durante o processamento do algoritmo, até o grafo inteiro estar embutido no plano. Neste ponto, a árvore-PQ T_n, correspondente a B_n, estará representada por um P-nó sem filhos, pois não há vértices com rótulo **n+1** a serem adicionados a T_n; B_n apresentará uma embutidura do grafo G.

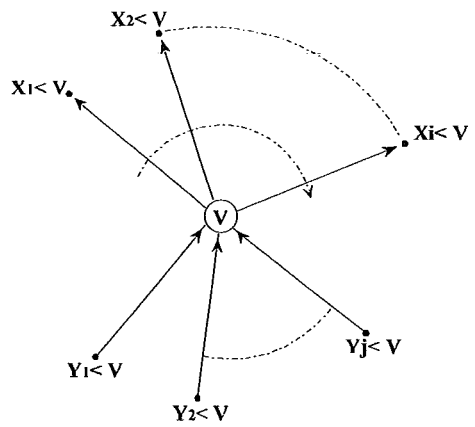
3.2- O Algoritmo de Chiba, Nishizeki, Abe e Ozawa

O algoritmo de embutidura de grafos planares biconexos de Chiba-Nishizeki-Abe-Ozawa [8] se baseia no teste de planaridade de Booth-Lueker. Seu objetivo é construir listas de adjacências A de G, onde as vizinhanças de cada vértice estejam organizadas na ordem dos ponteiros do relógio (ver DEF.3.1).

Para obter uma embutidura de G, é necessário observar como se apresenta a distribuição dos vizinhos com rótulos maiores e menores que o de **v** ao redor do vértice **v**. O lema a seguir formula a característica desta distribuição [8].

LEMA 3.2: Seja A uma embutidura do grafo planar G e **v** um vértice de G. Então todas as vizinhanças de **v** menores que **v** estão embutidas consecutivamente ao redor de **v** (ver Fig.3.4). Isto é, se w_1, w_2, w_3 e w_4, embutidos nesta ordem, pertencem a A(v), então não ocorrem $\mathbf{w_1, w_3 < v}$ e $\mathbf{w_2, w_4 > v}$.

Fig. 3.4



A ordem na qual os vizinhos de **v**, $\mathbf{v} \in V$, com rótulos menores que os de **v**, aparecem embutidos ao redor de **v**, é chamada **ordem dos ponteiros do relógio**.

Pode-se concluir imediatamente, a partir do Lema 3.2, que, se as vizinhanças menores do que **v** são embutidas consecutivamente ao redor de **v**, as maiores também o são.

Dado G=(V,E), não direcionado, simples, biconexo e st-rotulado, o algoritmo de [8] se subdivide em duas fases:

(i) construção das listas de adjacências do grafo D_u, obtido a partir de G, ordenadas na ordem dos ponteiros do relógio;

(ii) obtenção das listas de adjacências completas de G, na ordem dos ponteiros do relógio.

3.2.1- O Grafo Ascendente

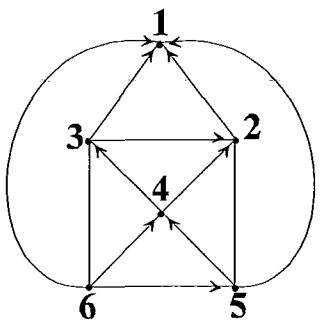
O **grafo ascendente D_u de G** é definido por [8] como um grafo direcionado obtido de G pela atribuição de uma direção a cada uma das arestas, de tal forma que, se **v** tem, pela rotulação-st, um valor maior do que **w**, então **v** é ascendente de **w**.

(DEF.3.4)

As listas de adjacências A_u de D_u, que vão definir uma embutidura de D_u, são subconjuntos das listas A do grafo G, não direcionado; assim, para qualquer aresta $<v,w>$, direcionada do vértice **v** para o vértice **w**, onde $v>w$, $A_u(v)$ contém **w**, mas $A_u(w)$ não contém **v**.

A Fig.3.5 apresenta, para o grafo da Fig.3.1a, o grafo ascendente D_u.

Fig. 3.4



O algoritmo de [8] constrói, inicialmente, as listas A_u de D_u, enquanto mantém, para cada uma, informações sobre as reversões aplicadas aos Q-nós das árvores-PQ analisadas. Ao final do processamento de G, as listas A_u são reorganizadas, de acordo com as informações mantidas sobre as reversões dos Q-nós. Finalmente, as listas A de G são refeitas, a partir das listas A_u.

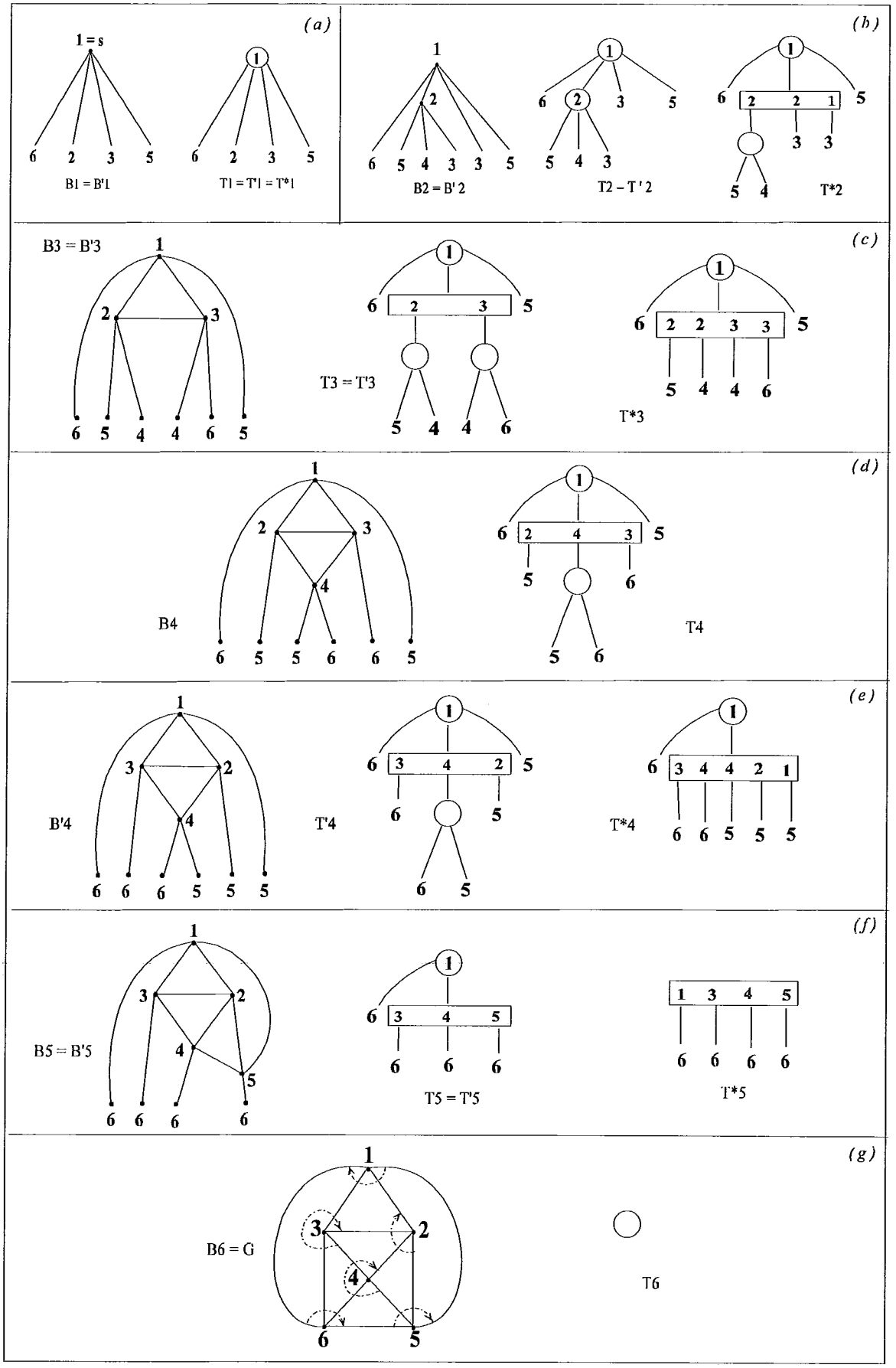
O algoritmo PLANAR de Booth-Lueker [4] é revisto nesta seção, pois ele é a estrutura básica do procedimento de embutidura de Chiba e outros. Ele testa, num passo **k**, a planaridade de G'_k, realizando permutações e reversões sobre os nós da subárvore-PQ pertinente, a fim de que vértices virtuais **k+1** ocupem posições consecutivas. Este posicionamento é tornado definitivo com o auxílio dos moldes de redução, e, depois, é construída a árvore-PQ de entrada para o passo seguinte, cujos nós cheios são os vértices **k+2**. O processo se inicia com a árvore T_1, que é a árvore-PQ correspondente à forma de arbusto B_1 e, portanto, uma embutidura de G'_1, e constrói uma seqüência de árvores-PQ $T_2,T_3,\ldots$, associadas, respectivamente, às formas de arbusto $B_2,B_3,\ldots$; se o grafo é planar, o algoritmo termina depois de construir T_{n-1}; caso contrário, ele termina depois de detectar a impossibilidade de reduzir alguma T_k a T^*_k, quando o grafo em análise não é planar.

Uma versão resumida do algoritmo PLANAR é apresentada a seguir para melhor visualização das alterações a serem implementadas por [8]. Uma delas é que a iteração varia de 2 até **n**, porque é neste último passo que o vértice com rótulo **n** é embutido na forma de arbusto B_{n-1}; o algoritmo original se encerra no passo **n-1**.

```
procedimento PLANAR (G);

começar
  . assinalar rotulação-st a todos os vértices de G;
  . construir a árvore-PQ correspondente a G'1;
  . para v:= 2 até n, fazer
    começar
      { passo de redução }
      . tentar agrupar todas as folhas cheias e aplicar os
        moldes, a partir das folhas até a raiz da subárvore
        pertinente (procedimentos BOLHA e REDUÇÃO);
      . se a redução falhar
        então retornar;              { G não é planar }
      { passo de adição de vértices }
      . substituir todos os nós cheios da árvore-PQ por
        um novo P-nó;
      . adicionar à árvore-PQ todas as vizinhanças de v
        como filhos do P-nó;        { para o grafo Du são
                              os vértices maiores do que v }
    fim
fim;
```

Na Fig.3.6, para o grafo da Fig.3.1a, são apresentadas as árvore-PQ construídas pelo algoritmo PLANAR e as formas de arbusto correspondentes, a cada passo; as folhas das árvores estão identificadas pelos vértices aos quais as arestas são incidentes A aplicação dos moldes está detalhada na seção 2.4, para o mesmo grafo.

Fig. 3.6


(a)
1 = s
6 2 3 5
B1 = B'1
1
6 2 3 5
T1 = T'1 = T*1
(b)
1
2
6 5 4 3 3 5
B2 = B' 2
1
6 2 3 5
5 4 3
T2 – T ' 2
1
6 2 2 1 5
3 3
5 4
T*2
(c)
B3 = B'3
1
2 3
6 5 4 4 6 5
1
6 2 3 5
T3 = T'3
5 4 4 6
1
6 2 2 3 3 5
5 4 4 6
T*3
(d)
1
2 3
4
B4
6 5 5 6 6 5
1
6 2 4 3 5
5 6
5 6
T4
(e)
1
3 2
4
B'4
6 6 6 5 5 5
1
6 3 4 2 5
6 5
6 5
T'4
1
6 3 4 4 2 1
6 6 5 5 5
T*4
(f)
1
3 2
4 5
B5 = B'5
6 6 6 6
1
6 3 4 5
6 6 6
T5 = T'5
1 3 4 5
6 6 6 6
T*5
(g)
1
3 2
4
B6 = G
6 5
T6

Observar a relação entre os vértices de corte e os P-nós e entre as componentes biconexas maximais e os Q-nós; a última árvore-PQ reduzida, T^*_5, é um Q-nó cujos filhos, todos, apontam para o vértice de rótulo 6; isto porque G é biconexo e planar.

3.2.1.1- Os Indicadores de Direção

Para obter a lista $A_u(v)$, $v \in V$, um procedimento *ingênuo* , no passo de adição de vértices do algoritmo PLANAR, armazenaria os pais das folhas rotuladas **v**, e processaria sobre eles as reversões subseqüentes, visando corrigir sua ordenação. No entanto, este procedimento consumiria tempo $O(n^2)$.

Chiba e outros [8] criaram, então, um vértice especial chamado **indicador de direção**, cujo objetivo é registrar as reversões aplicadas aos Q-nós de uma árvore-PQ durante o processo de redução. No passo de adição de vértices **v=k+1**, quando os nós cheios são substituídos por um P-nó, este vértice especial é adicionado como seu irmão, isto é, como filho do mesmo Q-nó pai, que é a raiz da subárvore pertinente, neste ponto, já reduzida para **v=k+1**.

Um indicador de direção se associa a Q-nós, e não a P-nós, porque cada bloco ou componente biconexa maximal que se forma em G_k e pode ser girado no decorrer do processo, se espelha em um Q-nó da árvore-PQ correspondente. Como o algoritmo de BL se propõe a testar a planaridade de grafos biconexos, os P-nós, inicialmente livres para serem permutados, são incluídos como filhos de algum Q-nó em um passo do procedimento.

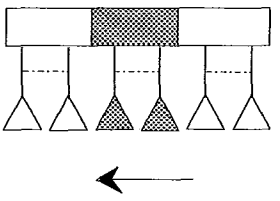
As funções de um indicador de direção são, portanto:

(i) controlar as reversões subseqüentes de $A_u(v)$, uma vez que ela é invertida a cada reversão do pai de **v**;

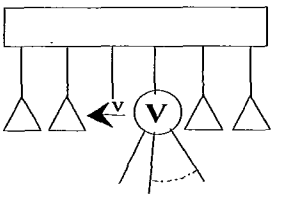
(ii) manter a direção do vértice **v** com relação aos seus irmãos.

Um indicador de direção é representado pelo símbolo -> e tem rótulo **v**. Sua orientação inicial é determinada pela travessia do Q-nó, a partir de um filho cheio e em direção a um dos filhos extremos (ver Fig.3.7); essa direção inicial deve ser a mesma para os próximos indicadores incluídos. Todo indicador de direção é uma folha, isto é, nunca tem filhos.

Fig. 3.7



direção da pesquisa



indicador v no passo de adição de vértices

A proposta de [8] para implementar os indicadores de direção se concretiza com a inserção de algumas instruções no algoritmo PLANAR. Este ignora a sua presença quando deseja acessar um irmão imediato de um vértice **v**; durante o passo de redução, os indicadores de direção são tratados como vértices usuais da árvore-PQ, sendo invertidos sempre que o Q-nó pai for submetido a uma reversão.

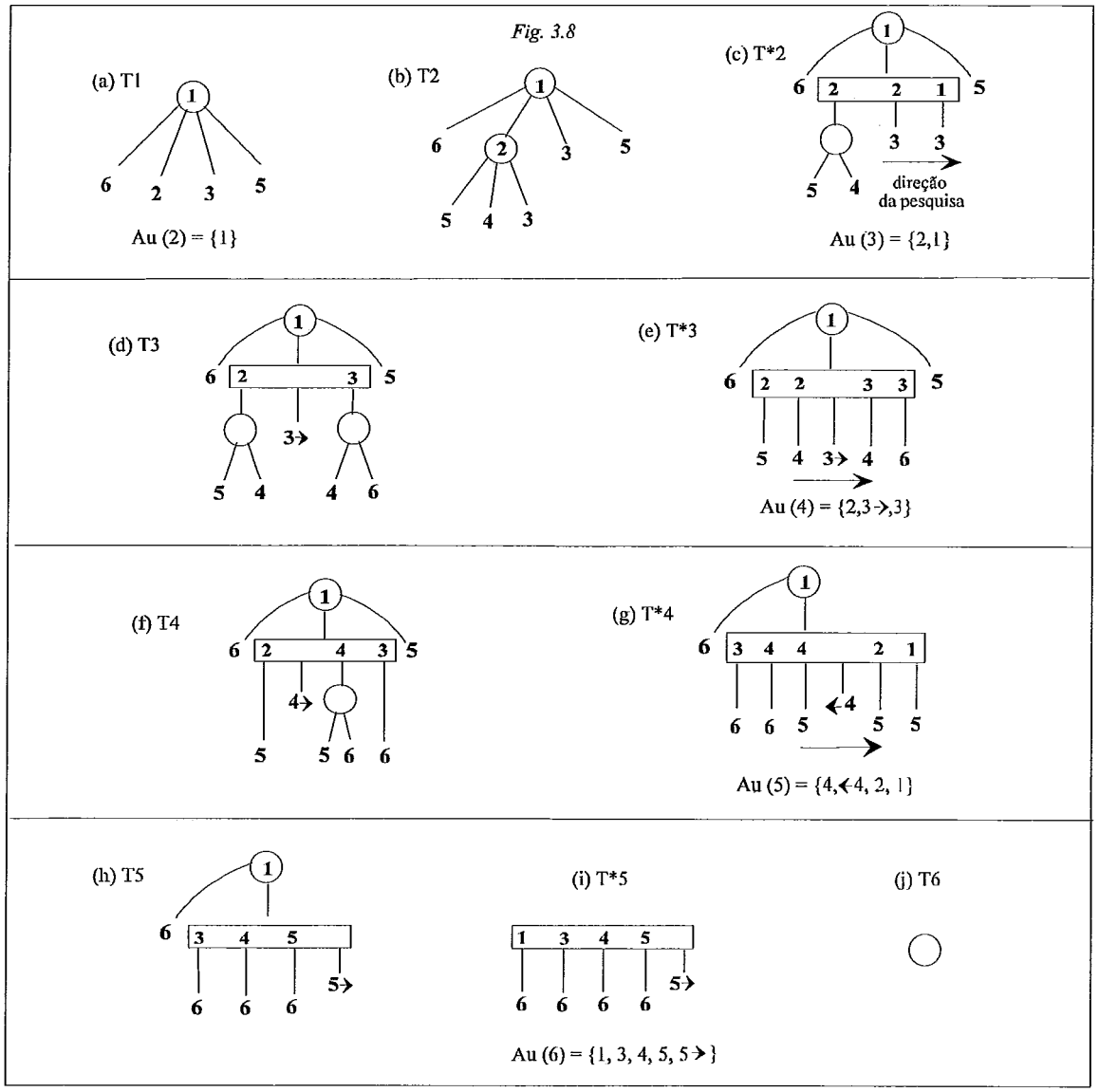
Em um passo **k** do algoritmo, assume-se que a raiz **r** da subárvore pertinente não é cheia e que já existem alguns indicadores $\mathbf{f_1}$, $\mathbf{f_2}$, ..., $\mathbf{f_k}$, criados em passos anteriores como filhos de **r**, para vértices com rótulo menor que **v=k+1**. Na fase de adição de vértices para **v=k+1**, o indicador com rótulo **v** é acrescentado à árvore-PQ; a partir daí, ele é invertido quando o seu Q-nó pai sofrer uma reversão. É, portanto, suficiente guardar as direções de f_1, f_2,..., f_k, relativamente a **v**; quando eles são eliminados da árvore-PQ, são armazenados em $A_u(v)$, juntamente com os demais vizinhos do vértice **v**.

Ao final do processamento do grafo G, as listas $A_u(v)$, para cada $v \in V$, contêm as adjacências do grafo direcionado D_u e, possivelmente, alguns indicadores de direção. Cada um deles tem uma direção igual ou inversa à inicial, correspondente às reversões do Q-nó pai. Um procedimento de correção é então definido, a partir do vértice **n** até o vértice **1**, invertendo as listas cujos indicadores estão na direção oposta à inicial. Para cada **v**, os indicadores armazenados em $A_u(v)$ têm rótulo menor que num-st(v) e, portanto, somente as listas $A_u(w)$, onde o rótulo de **w** é menor o rótulo de **v**, são atualizadas.

O procedimento EMBUT_ASCEND, detalhado a seguir, calcula as listas A_u de D_u com base nos indicadores de direção. No máximo O(n) indicadores são criados durante sua execução, requerendo no máximo um tempo O(n) a mais do que o do algoritmo PLANAR, no qual está inserido; isto porque, cada indicador processado num passo de redução, é retirado da árvore-PQ no próximo passo de adição de vértices, implicando que cada indicador é processado apenas uma vez.

```
procedimento EMBUT_ASCEND (G);

começar
  . assinalar rotulação-st a todos os vértices de G;
  . construir a árvore-PQ correspondente a G'1;
  . para v:= 2 até n, fazer
    começar
      { passo de redução }
      . tentar agrupar todas as folhas cheias e aplicar os
        moldes, a partir das folhas até a raiz da subárvore
        pertinente (procedimentos BOLHA e REDUÇÃO),
        ignorando todos os indicadores de direção,
        filhos da subárvore pertinente;
      { passo de adição de vértices }
      . eliminar as folhas incidentes ao vértice v e os
        indicadores de direção, irmãos das folhas cheias,
        da árvore-PQ e armazená-los em Au(v);
      . se a raiz r da subárvore pertinente não está cheia
        então começar
                . adicionar um indicador "v", orientado con-
                  forme a direção de pesquisa do Q-nó, como
                  filho da raiz r, numa posição arbitrária
                  entre os demais filhos;
                . substituir todos os filhos cheios da
                  árvore-PQ por um novo P-nó
              fim
        senão substituir a subárvore pertinente por um P-nó;
      . adicionar à árvore-PQ todas as vizinhanças de v
        como filhos do P-nó;        { para o grafo Du são
                              os vértices maiores do que v }
    fim;
  { passo de correção }
  . para v:= n até 1, fazer
      . para cada elemento x em Au(v), fazer
          . se x é um indicador de direção
            então começar
                    . eliminar x de Au(v);
                    . w:= rótulo de x;
                    . se a direção do indicador x for oposta
                      àquela inicial de Au(v)
                      então reverter a lista Au(w);
                  fim
fim;
```

A Figura 3.8 ilustra os passos seguidos pela primeira fase algoritmo EMBUT_ASCEND, aplicado ao grafo da Figura 3.1a; ela apresenta as árvores-PQ referentes a cada iteração e as listas de adjacências construídas no início de cada passo de adição de vértices.



O passo de correção, que finaliza o algoritmo, alcança o seguinte resultado:

. para v=6 , $A_u(6)$= {1,3,4,5, 5-> } contém o indicador 5 na direção inicial, o que não ocasiona inversão de nenhuma lista $A_u(v)$;

. para v=5 , $A_u(5)$= {4, <-4 ,2,1} apresenta o indicador 4 na direção oposta à inicial. Então, a lista $A_u(4)$ é invertida, incluindo o indicador que ela contém, ficando $A_u(4)$= {3, <-3, 2);

. para v=4 , $A_u(4)$= {3, <-3 ,2} tem agora o indicador 3 na direção oposta à original. A lista $A_u(3)$ é invertida, obtendo-se $A_u(3)$={1,2};

. as listas para v=3, 2 ou 1 não contêm indicadores; logo, nenhuma outra lista é alterada.

As listas de adjacências de D_u são, finalmente: $A_u(1)$={}; $A_u(2)$={1}; $A_u(3)$={1,2}; $A_u(4)$={3,2}; $A_u(5)$={4,2,1}; $A_u(6)$={1,3,4,5}.

3.2.2- A Embutidura Planar

O objetivo agora é expandir A_u em A, obtendo A(v), para cada $v \in V$, na ordem dos ponteiros do relógio, o que define uma embutidura do grafo G inicial, que é não direcionado.

O Lema 3.2 mostra que as vizinhanças de **v** estão embutidas consecutivamente ao redor de **v** e, ainda, que os vizinhos de rótulo menor que **v** formam um grupo que não se intercala com o dos vizinhos maiores. Com base neste lema, define-se a construção das listas A de G.

Este novo procedimento, chamado EMBUT_COMPLETO, executa uma busca em profundidade no grafo direcionado D_u, a partir do vértice **n**. As listas de adjacências A(v) são inicializadas com os elementos de $A_u(v)$, para cada **v**. Depois, durante a busca, quando uma aresta direcionada $<y_k,v>$ é visitada, o vértice y_k é adicionado ao topo de A(v), isto é, como primeiro elemento de A(v).

```
procedimento EMBUT_COMPLETO (G);

começar
  . copiar as embutiduras ascendentes Au para as listas A;
  . marcar cada vértice como novo;
  { T é uma árvore de busca construída pelo algoritmo }
  . T:= vazio;
  { t é o vértice sumidouro ou n }
  . DFS(t);
fim;


procedimento DFS (y);

começar
  . marcar vértice y como velho;
  . para cada vértice v ∈ Au(y), fazer
      começar
        . inserir vértice y no topo de A(v);
        . se v estiver marcado como novo
          então começar
                  . adicionar a aresta <y,v> a T;
                  . DFS(v)
                fim
      fim
fim;
```

A embutidura completa do grafo G da Fig.3.1a é, então, representada pelas listas de adjacências A(v) obtidas pelo procedimento EMBUT_COMPLETO:

A(1) = {5,2,3,6} A(4) = {5,6,3,2}
A(2) = {5,4,3,1} A(5) = {6,4,2,1}
A(3) = {4,6,1,2} A(6) = {1,3,4,5}

LEMA 3.3: Seja D_u um grafo ascendente de um grafo G, e seja A_u uma embutidura ascendente de D_u. Então, o algoritmo EMBUT_COMPLETO expande A_u em uma embutidura A de G em tempo linear [8].

O algoritmo completo é definido como a seguir:

```
procedimento EMBUTIDURA (G);
começar
  . EMBUT_ASCEND(G);
  . EMBUT_COMPLETO(G);
fim EMBUTIDURA;
```

Na Fig.3.6, apresentada para melhor visualização da execução do algoritmo PLANAR, o item (g) contém a forma de arbusto B_6 correspondente a uma embutidura planar de G. O algoritmo EMBUTIDURA de [8] calcula as listas de adjacências de G na ordem dos ponteiros do relógio, possibilitando a construção imediata do mesmo desenho, sem necessidade de utilizar as formas de arbusto intermediárias para acompanhar o procedimento.

3.3- O Algoritmo de Jayakumar, Thulasiraman e Swamy

Jayakumar, Thulasiraman e Swamy [12] baseiam seu trabalho no algoritmo de Lempel-Even-Cederbaum e na sua implementação utilizando as árvores-PQ. Seu objetivo é estabelecer uma ordenação para os vértices do grafo em estudo, de forma a poder distribuí-los em uma linha horizontal; depois, associando a cada vértice **v**, como ordenada, o num-st(v), obter um posicionamento dos vértices no plano, de tal forma que as arestas possam ser desenhadas sem cruzamento. O resultado do algoritmo de [12] é bem mais próximo de um traçado que o de [8]; no entanto, nenhum critério estético é ainda considerado, o que pode ser observado, por exemplo, no desenho das arestas, que na sua maioria são curvas.

Dado G=(V,E), não direcionado, simples, biconexo e st-rotulado, o algoritmo de [12] pode ser subdividido em três fases:

(i) construção da ordenação-f dos vértices;

(ii) determinação da ordenação M dos vértices;

(iii) colocação das arestas no plano.

Uma **ordenação-f** de um vértice **v**, ou **f(v)**, v ∈ V, é a lista de adjacências de **v** que contém todos os vizinhos de **v** com rótulo menor que num-st(v), na ordem contrária a dos ponteiros do relógio; uma **ordenação-f'** ou **f'(v)**, define-se como f(v) , sem ordenação para os vértices.

Para atingir o objetivo da primeira fase, [12] utiliza o algoritmo PLANAR, visto na seção 2.4. Durante seu processamento, no passo **v=k+1**, após a redução da árvore-PQ T_k, isto é, a partir de T^*_k, é construída a lista **f'(v)**, ou a **ordenação-f'** de **v**; seus elementos são as vizinhanças de **v** ou os filhos cheios de T^*_k.

Fig. 3.9

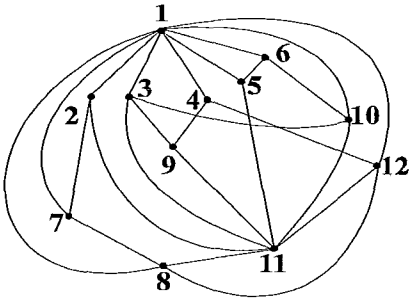


Fig. 3.10 - Forma de arbusto B9

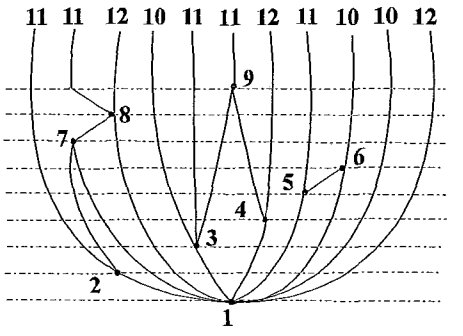
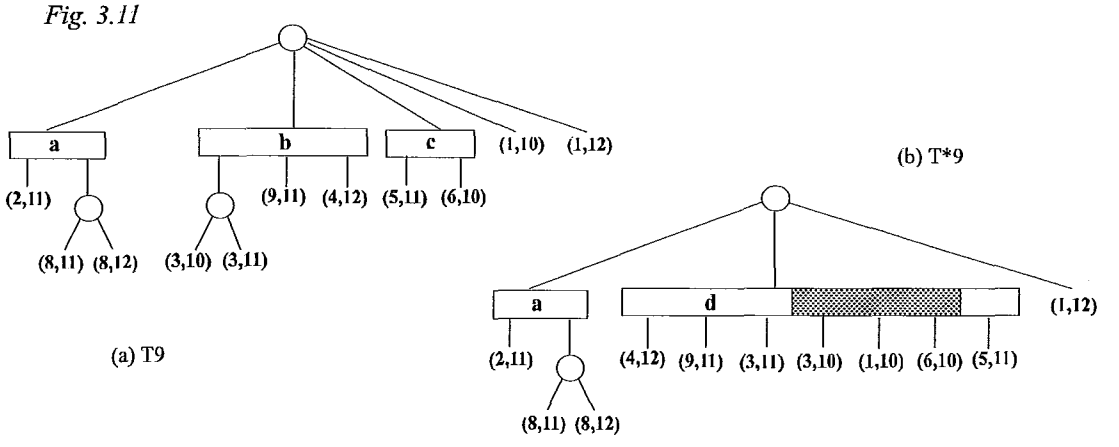


Fig. 3.11



Seja G o grafo não direcionado, simples, biconexo e st-rotulado da Fig.3.9.

Para **k=9** e, portanto, **v=10**, a forma de arbusto B_9, como construída por [12], é mostrada na Fig.3.10. Seja T_9 (ver Fig.3.11a) a árvore correspondente a B_9 e T^*_9 (ver Fig.3.11b), sua reduzida. As folhas das árvores-PQ representam as arestas de G, cuja extremidade de maior rótulo é ≥10, que é o num-st(v), e ainda não foram processadas. Neste passo do procedimento PLANAR, as folhas cheias são (3,10), (1,10) e (6,10), em T^*_9. A lista f'(v) é obtida selecionando os vizinhos de v=10, na ordem em que aparecem em T^*_9. Então tem-se f'(10) = (3,1,6).

Ao final do processamento de G, as listas f'(v), para todo $v \in V$, estão calculadas. Para obter f(v), isto é, as vizinhanças de **v** ordenadas no sentido contrário aos ponteiros do relógio, [12] utiliza o grafo de blocos, definido na seção a seguir.

A segunda fase do algoritmo de [12] reúne todas as listas f(v) adequadamente para determinar a ordenação M dos vértices de G, que corresponde à distribuição dos mesmos no eixo horizontal.

No início da terceira fase as coordenadas dos vértices de G estão estabelecidas; o algoritmo define, então, as regiões onde as arestas podem ser desenhadas e obtém a embutidura planar de G.

3.3.1- O Grafo de Blocos

Durante o processo de crescimento dos arbustos, quando um bloco é revertido, a ordenação-f' dos vértices que fazem parte deste bloco também deveria ser invertida; assim, ao final do processamento PLANAR, as listas f'(v) seriam as próprias listas f(v). Para poder obter as listas f(v), sem acompanhar as reversões de f'(v) passo a passo, são definidos os **grafos de blocos**, cuja função é semelhante à dos indicadores de direção de [8].

Sejam a forma de arbusto B_k e a árvore-PQ T_k. Os blocos de B_k são representados pelos Q-nós da subárvore pertinente de T_k; quando as arestas incidentes ao vértice **v=k+1** são agrupadas e a árvore-PQ T'_k é reduzida em T^*_k, os blocos de B_k _sejam eles $C_k(1)$, $C_k(2)$, ...,$C_k(i)$_ se agrupam para formar um novo bloco. Eles são considerados como *englobados* por esse novo bloco C_{k+1}, deixando de ser maximais nas formas de arbusto subseqüentes.

Em paralelo aos indicadores de direção definidos por [8], este algoritmo constrói um grafo direcionado e rotulado chamado **grafo de blocos**, cujos vértices

representam blocos não triviais (aqueles que têm no mínimo três arestas) e cujas arestas representam a relação de pertinência entre eles. Como notação, tem-se:

- o vértice **c** no grafo de blocos representa um bloco C de B_k;
- uma aresta se dirige de **c** para **c'**, vértices do grafo de blocos, se o bloco C englobou o bloco C';
- a cada vértice **c** se associa um rótulo, chamado **status**, que recebe os valores **R** ou **NR**, respectivamente, revertido ou não revertido.

Cada bloco não trivial em B_k, criado por uma redução, é incluído como vértice do grafo de blocos que, durante alguns passos do algoritmo, pode ser desconexo.

Observando a Fig.3.11a, tem-se na árvore T_9 que:

- o Q-nó **a** representa um bloco formado pelos vértices 1,2,7 e 8, este, o último a ser incluído, e as arestas que os ligam; este bloco, denominado C_8, em alguma redução anterior anexou o bloco C_7;
- o Q-nó **b**, formado pelos vértices 1,3,4 e 9, é o bloco C_9;
- o Q-nó **c**, formado pelos vértices 1,5 e 6, é o bloco C_6.

Na árvore T^*_9 (ver Fig.3.11b), que é a entrada do passo de adição de vértices que constrói T_{10} correspondente a B_{10}, observa-se que:

- o bloco C_8 permanece inalterado;
- os blocos C_9 e C_6 estão ambos revertidos na formação de T^*_9;
- está criado o bloco C_{10}, incorporando os blocos C_9 e C_6.

O grafo de blocos, na iteração $v=10$, terá vértices c_6, c_9 e c_{10}, com arestas dirigidas de c_{10} para os demais; os status de c_6 e c_9 será **R** e o de c_{10}, **NR**.

A construção do grafo de blocos está apenas delineada. Para uma definição correta, é necessário considerar o caso mais geral, isto é, quando a formação de um bloco incorpora dois ou mais Q-nós que, na sua hierarquia, têm filhos que também são Q-nós ou, ainda, um ou mais Q-nós intermediários são criados e, depois, eliminados durante o processo de redução.

O procedimento de construção do grafo de blocos é descrito em três etapas:

(i) construção do grafo de blocos intermediário;

(ii) atualização do grafo de blocos;

(iii) correção do status dos blocos.

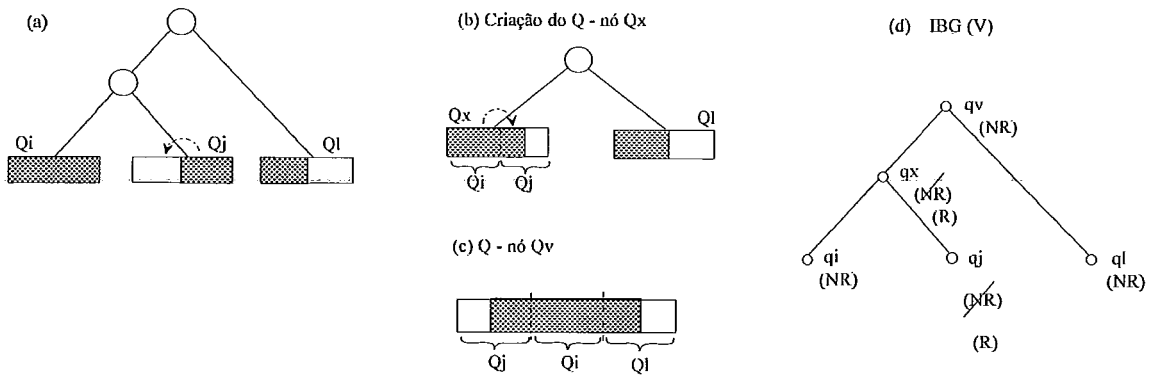
Seja IBG(v), para cada $v \in V$, o grafo de blocos intermediário. IBG(v) é uma árvore direcionada, usada para controlar os giros a que são submetidos os blocos durante a redução $\mathbf{v=k+1}$; ele é criado e utilizado somente durante este passo; cada uma

de suas folhas corresponde a um Q-nó em T_k.

Inicialmente adicionam-se a IBG(v) os vértices que correspondem aos Q-nós da subárvore pertinente T_k, associando-se a cada um deles o status NR. Quando, durante o procedimento de redução de T_k, um Q-nó é criado, englobando outros, ele é adicionado a IBG(v) com o rótulo NR e os seus filhos tem os rótulos atualizados (de NR para R), caso os Q-nós tenham sido revertidos. Observar que os Q-nós que inicializam IBG(v) têm status NR porque, criados em passos anteriores com este status, eles ainda não foram processados até este momento.

Seja G=(V,E) um grafo genérico; num passo v=k+1, a árvore-PQ T_k, a ser reduzida, tem como Q-nós pertinentes Q_i, Q_j e Q_l; a subárvore pertinente de T_k é apresentada na Fig.3.12a. O grafo IBG(v) é inicializado com três vértices q_i, q_j e q_l, todos com status NR. Um nó Q_x, intermediário, é criado durante o processo de redução, englobando os blocos Q_i e Q_j; supõe-se que o nó Q_j é revertido, para atingir o agrupamento das folhas cheias (ver Fig.3.12b). Essas informações são armazenadas em IBG(v) com a criação de um vértice q_x com rótulo NR e arestas dirigidas para os vértices q_i e q_j; o status do vértice q_j é atualizado para R. Ainda no passo **k**, os blocos Q_x e Q_l são englobados por um outro bloco final, seja Q_v; agora Q_x é revertido. IBG(v) é atualizado com a inclusão do vértice q_v, com arestas direcionadas para q_x e q_l e com a alteração do status de q_x para R.

Fig. 3.12



A Fig.3.12c ilustra a criação do Q-nó Q_v e a Fig.3.12d, a do grafo de blocos intermediário IBG(v). É importante observar que o nó q_x de IBG(v) é um nó intermediário do processo e, portanto, não vai constar do grafo de blocos.

O procedimento definido é chamado CALCULA_RÓTULO e possibilita a

atualização do grafo de blocos, que é a segunda etapa descrita.

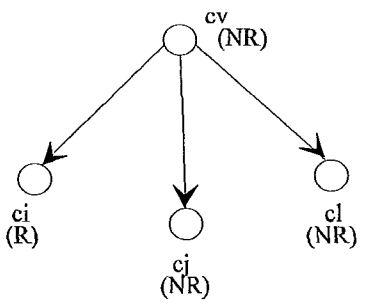
O grafo de blocos, no início o passo **k**, contém todos os blocos não triviais criados em passos anteriores. A primeira providência desta segunda etapa é incluir o bloco C_v, v=k+1, criado neste passo, no grafo de blocos e ligá-lo aos blocos incorporados a ele. Depois, a partir de IBG(v), os status dos vértices do grafo de blocos, filhos de C_v, são atualizados da seguinte maneira:

- procede-se uma busca em profundidade em IBG(v) a partir da raiz, que é o vértice q_v, correspondente ao bloco C_v;
- quando um vértice com rótulo R é encontrado, os rótulos dos seus filhos são trocados (de NR para R ou vice-versa);
- quando termina a busca em IBG(v), transfere-se para o grafo de blocos o resultado, isto é, o rótulo atualizado das folhas de IBG(v), que será o status dos vértices correspondentes no grafo de blocos.

A Fig.3.13 apresenta um grafo de blocos parcial, correspondente ao passo **v=k+1** de redução de T_k, cuja subárvore pertinente é a da Fig.3.12a. O bloco C_v, relativo ao Q-nó Q_v, é criado neste passo e incluído no grafo de blocos; suas arestas são dirigidas para C_i, C_j e C_l, que se referem a blocos advindos de passos anteriores, englobados por C_v. O status inicial de todos os vértices é NR.

O procedimento de busca se inicia no vértice q_v do grafo de blocos intermediário da Fig.3.12d, que tem status NR; portanto, seus filhos q_x e q_l mantêm seus status, respectivamente, R e NR. Já os de q_i e de q_l são trocados, de NR para R e de R para NR, pois seu pai, q_x, tem R como status. As informações finais da busca em IBG(v) atualizam o status dos vértices do grafo de blocos. Assim, o status de C_v, Cj e C_l é NR e o de C_i é R.

Fig. 3.13



A seguir detalha-se o procedimento GRAFO_DE_BLOCOS, que abrange as duas primeiras etapas da construção do grafo de blocos. Ele se referencia ao procedimento CALCULA_RÓTULO, descrito na primeira fase:

```
procedimento GRAFO_DE_BLOCOS;
{ construção do bloco de grafos e da informação do status
  de cada bloco, referente às reversões sofridas durante
  o processo de redução da árvore-PQ }

começar
  .para v :=2 até n-1, fazer
     começar
       { obtenção dos status dos blocos }
       . CALCULA_RÓTULO;
       . para cada Q-nó Qi pertinente em Tk, fazer
           começar
             . traçar uma aresta direcionada de cv para ci;
             . STATUS(i) := rótulo de Qi; { obtido por
                                  CALCULA_RÓTULO }
           fim
       { O Q-nó raiz da subárvore pertinente T*k }
       { representa o bloco Cv }
       . obter Tk+1;
       . STATUS(v) := NR;
     fim
fim;
```

Resumidamente, a construção do grafo de blocos pode ser apresentada da seguinte maneira:

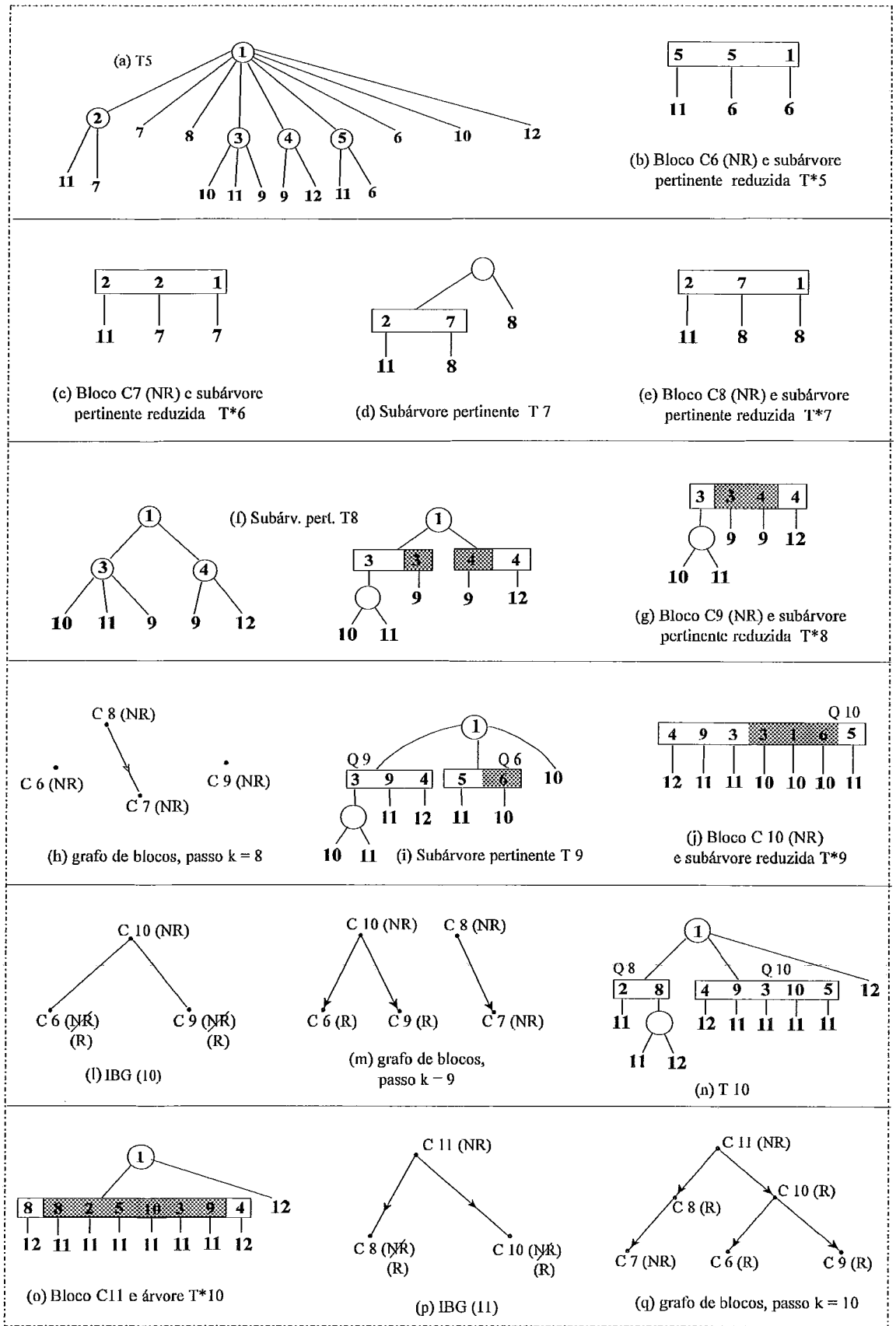
- cada Q-nó pertinente de T_k é um vértice no grafo de blocos;
- para o Q-nó resultante do passo de redução é criado um vértice no GB, com arestas dirigidas para os vértices que representam os blocos agrupados por ele;
- este último vértice tem status NR;
- seus filhos têm os status trocados, se os Q-nós que eles representam tiverem sido revertidos na conclusão do passo de redução **k**.

Os procedimentos CALCULA_RÓTULO e GRAFO_DE_BLOCOS podem ser incluídos no algoritmo de redução de árvores-PQ, definido por Booth-Lueker (ver seção 2.4).

A Fig.3.14 ilustra a construção do grafo de blocos referente ao grafo st-rotulado da Fig.3.10, paralelamente à construção e redução das árvores-PQ. Para simplificar os desenhos, explicitam-se, como abaixo, as listas de adjacências de G, utilizadas para as adições de vértices:

A(1)={2,7,8,3,4,5,6,10,12}; A(2)={11,7}; A(3)={10,11,9}; A(4)={9,12}; A(5)={11,6}; A(6)={10}; A(7)={8}; A(8)={11,12}; A(9)={11}; A(10)={11}; A(11)={2}; A(12)={}.

Fig. 3.14


(a) T5
(b) Bloco C6 (NR) e subárvore pertinente reduzida T*5
(c) Bloco C7 (NR) e subárvore pertinente reduzida T*6
(d) Subárvore pertinente T 7
(e) Bloco C8 (NR) e subárvore pertinente reduzida T*7
(f) Subárv. pert. T8
(g) Bloco C9 (NR) e subárvore pertinente reduzida T*8
C 8 (NR)
C 6 (NR)
C 9 (NR)
C 7 (NR)
(h) grafo de blocos, passo k = 8
Q 9
Q 6
(i) Subárvore pertinente T 9
Q 10
(j) Bloco C 10 (NR) e subárvore reduzida T*9
C 10 (NR)
C 6 (NR) (R)
C 9 (NR) (R)
(l) IBG (10)
C 10 (NR)
C 8 (NR)
C 6 (R)
C 9 (R)
C 7 (NR)
(m) grafo de blocos, passo k – 9
Q 8
Q 10
(n) T 10
C 11 (NR)
(o) Bloco C11 e árvore T*10
C 11 (NR)
C 8 (NR) (R)
C 10 (NR) (R)
(p) IBG (11)
C 11 (NR)
C 8 (R)
C 10 (R)
C 7 (NR)
C 6 (R)
C 9 (R)
(q) grafo de blocos, passo k = 10

O primeiro bloco criado é C_6 no passo **k**=5; o segundo é C_7; o bloco C_8 engloba C_7, sem o girar; o quarto bloco é C_9. A Fig.3.14h apresenta o grafo de blocos para **k**=8. Nas Figs.3.14l e 3.14m, o grafo de blocos intermediário mostra o status invertido dos vértices C_6 e C_9, na criação de C_{10}, e o grafo de blocos atualizado para estas informações. Nas Figs.3.14p e 3.14q tem-se IBG(11) e o grafo de blocos do passo **k**=10, com C_{11} incluído e os status de C_8 e de C_{10} corrigidos. Não há necessidade de construir um grafo de blocos para a inclusão de C_{12}, pois, neste último passo, existirá um único bloco, que é C_{11}, a ser incorporado a C_{12}, uma vez que G é biconexo; o status de ambos é NR.

A primeira e a segunda etapas da construção do grafo de blocos estão concluídas. No entanto, se cada passo k do procedimento de redução do grafo G tivesse sido acompanhado com a forma de arbusto B_k, seria possível verificar que os status dos blocos C_7, C_6 e C_9 não estão corretos. Isto acontece porque a árvore IBG(v), v=k+1, só controla a reversão dos blocos englobados e criados durante o passo k.

A terceira etapa é executada após o término da redução de G. Ela visa corrigir os status dos vértices do grafo de blocos obtido pelas etapas anteriores.

Define-se, então, o procedimento CALCULA_STATUS, onde a reversão de cada bloco, armazenada no seu status, é propagada para os blocos filhos no grafo de blocos. Utilizando uma busca em profundidade, percorre-se a árvore direcionada do bloco de grafos, a partir da raiz, trocando o status de cada filho cujo pai tem um status R; naturalmente mantêm-se inalterados os status dos filhos cujo pai tem status NR. Com este procedimento determina-se o status de cada bloco no final da embutidura de G.

Com o bloco de grafos construído e corrigido, pode-se atualizar a ordenação-f' de cada vértice **v**, invertendo-a ou não, conforme a informação do status final de cada bloco, e obtendo f(v).

Na Fig.3.15, tem-se o grafo de blocos da Fig.3.14q corrigido pelo procedimento CALCULA_STATUS. Os blocos C_8 e C_{10}, têm ambos status R e seus filhos estão com seus status ainda trocados.

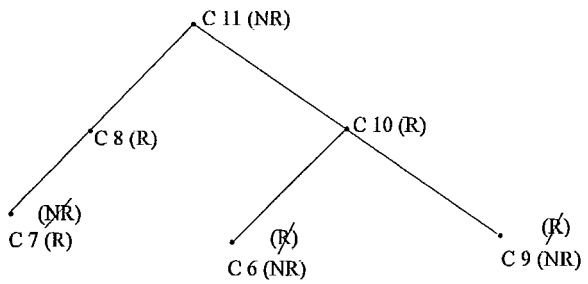
Utilizando as suas informações atualizadas, corrige-se a ordenação-f' dos vértices, como apresentado a seguir, a partir das Fig.3.14.:

o: f'(12)=(8,11,4,1); status C_{12}=NR, então f(12)=f'(12);

o: f'(11)=(8,2,5,10,3,9); status C_{11}=NR, então f(11)=f'(11);

j: f'(10)=(3,1,6); status C_{10}=R, então f(10)=(6,1,3);
g: f'(9)=(3,4); status C_9=NR, então f(9)=f'(9);
e: f'(8)=(7,1); status C_8=R, então f(8)=(1,7);
c: f'(7)=(2,1); status C_7=R, então f(7)=(1,2);
b: f'(6)=(5,1); status C_6=NR, então f(6)=f'(6).
As listas f(v) para v=5,4,3,2 contêm somente o vértice 1.

Fig. 3.15 Grafo de blocos final, corrigido



Observa-se que o vértice 1 pode ser omitido de f'(n), pois, ele, certamente, aparece na ordenação-f' de algum vértice menor do que **n**. Como, na ordenação final dos vértices, a aresta (1,n) tem a liberdade de ser colocada na esquerda ou na direita do conjunto de arestas, é dispensável controlá-lo. Assim, f(12)=f'(12)=(8,11,4).

Prova-se, em [12], que:

(i) o custo de CALCULA_RÓTULO é $O(N_{k+1})$, onde N_{k+1} é o número de folhas pertinentes de T_k;

(ii) o algoritmo GRAFO_DE_BLOCOS constrói corretamente o grafo de blocos e determina a informação de status de cada bloco em tempo O(n);

(iii) o algoritmo CALCULA_STATUS determina f(v), $2 \leq v \leq n$, corretamente, em tempo O(n).

3.3.2- A Ordenação dos Vértices

Para atingir o objetivo de distribuir os vértices em uma linha horizontal, define-se **M**, ou a **ordem dos vértices** de G, como uma ordenação esquerda-direita que determina o posicionamento de cada vértice **v** nos níveis verticais. Essa ordem, em geral, não é única. Por isso, a ela é imposta uma propriedade que reflete as informações fornecidas pelo processo de redução das árvores-PQ e é **o posicionamento dos vértices**

de f(v) relativamente à posição de v.

A ordem dos vértices, **M**, é inicializada com o vértice **n** e incrementada, sucessivamente, com as vizinhanças de **n, n-1, ..., 2**, nesta ordem. Algumas considerações são relevantes quanto à inclusão dos elementos de f(v) em M:

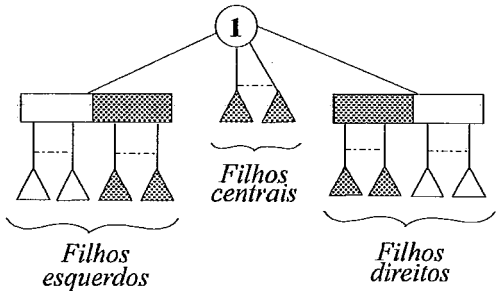
(i) quando os vizinhos de **v** são adicionados a M, o vértice **v** já está presente, pois ele pertence a f(w), para algum **w > v**;

(ii) dentre os vizinhos de **v**, elementos de f(v), a serem incluídos, constata-se que alguns já constam de M; isto porque, eles são adjacentes a vértices maiores do que **v** e, portanto, pertencem à lista f(v), para **w > v**, já processada. Estes vértices são chamados **TIPO2**, relativamente a **v**; os demais são **TIPO1** e, somente estes são selecionados de f(v) e acrescentados em M;

(iii) durante o processo de redução de uma árvore-PQ T_k em outra T^*_k, no momento em que o nó em análise é a raiz da subárvore pertinente e o molde escolhido é P6 ou Q3 (ver seção 2.2), um dos filhos parciais tem todos os nós cheios na sua direita, e o outro, todos os nós cheios na sua esquerda; os demais filhos pertinentes são cheios e estão entre os filhos parciais. Os filhos parciais são chamados, respectivamente, FILHOS ESQUERDOS e FILHOS DIREITOS; os demais são FILHOS CENTRAIS. Se o último molde aplicado na redução é P1, P2 ou Q1, os nós cheios são todos considerados centrais; se o molde é P4 ou Q2, o nó parcial pode ser esquerdo ou direito e, os demais, centrais; os moldes P3 e P5 não se aplicam à raiz da subárvore pertinente.

A Fig.3.16 apresenta o caso geral do padrão P6, destacando os filhos esquerdos, centrais e direitos.

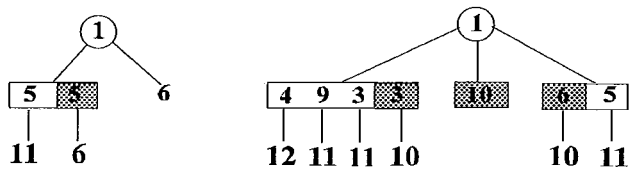
Fig. 3.16



O conceito definido em (iii) pode ser aplicado aos elementos de f'(v), agrupando-os condizentemente, e subdividindo f'(v) em $f'_e(v)$, $f'_c(v)$ e $f'_d(v)$; como a lista f'(v) nunca é vazia, pelo menos uma das subdivisões contém elementos. Observa-se que, para obter a ordenação-f, se f'(v) for revertida então f(v) será igual a $f'_d(v)$ revertida, $f'_c(v)$ revertida, $f'_e(v)$ revertida, nesta ordem.

A Fig.3.17a apresenta a árvore T'_5; sobre ela, aplicando o molde P4, obtém-se T^*_5; a lista f'(6) subdivide-se em $f'_e(6)=(5)$ e $f'_c(6)=(1)$. Como o status do bloco C_6, (ver Fig.3.15), é NR, a ordenação-f de v=6 se mantém igual a f'(v); isto é, $f_e(6)=(5)$ e $f_c(6)=(1)$.

Fig. 3.17



Na Fig.3.17b tem-se T'_9; o molde de redução a ser aplicado é P6; assim $f'_e(10)=(3)$, $f'_c(10)=(1)$ e $f'_d(10)=(6)$. Neste caso, o status do bloco C_{10} é R e, portanto, a ordenação-f' do vértice 10 é invertida, obtendo-se $f_e(10)=(6)$, $f_c(10)=(1)$ e $f_d(10)=(3)$.

Para os demais vértices, tem-se:

$f_e(12)=()$,	$f_c(12)=(8,11,4)$,	$f_d(12)=()$;
$f_e(11)=(8,2)$,	$f_c(11)=()$,	$f_d(11)=(5,10,3,9)$;
$f_e(9)=(3)$,	$f_c(9)=()$,	$f_d(9)=(4)$;
$f_e(8)=()$,	$f_c(8)=(1)$,	$f_d(8)=(7)$;
$f_e(7)=()$,	$f_c(7)=(1)$,	$f_d(7)=(2)$;f

para v=5,4,3 e 2, as listas $f_e(v)$ e $f_d(v)$ são vazias e $f_c(v)=(1)$.

As ordenações-f esquerda, central e direita, para cada v, são utilizadas para posicionar os vértices de f(v), relativamente a v, na ordenação M dos vértices. Define-se, então, o procedimento de construção de M da seguinte maneira:

- colocam-se os vértices do TIPO1 de $f_e(v)$ imediatamente à esquerda de v, os de $f_d(v)$ imediatamente à direita e os de $f_c(v)$, ao redor de v, isto é, uma parte à esquerda e outra à direita de v; nos três casos a ordem esquerda-direita dos vértices de f(v) é respeitada ao incluí-los em M.

Construindo M desta forma, a embutidura final de G apresenta os blocos que contêm os vértices de $f_e(v)$ à esquerda de v, e os que contêm os vértices de $f_d(v)$ na sua direita.

Para o grafo da Fig.3.10a, utilizando as listas f(v) subdivididas em esquerda, central e direita, uma ordenação dos vértices, possível de ser calculada é:

M = (8,7,12,2,11,5,6,10,1,3,9,4),

que é uma ordem na qual os vértices serão distribuídos no eixo horizontal. Para sua obtenção, inicializa-se M com o vértice 12. A seguir, como $f_c(12)$ é não vazia, seus elementos são colocados ao redor de v=12. Tem-se M=(8,12,11,4).

Na lista $f_e(11)$, o vértice 8 é do TIPO2, isto é, ele é adjacente a um vértice maior do que 12, no caso, 11, e não é selecionado. Tem-se M=(8,12,2,11,5,10,3,9,4); observar que o vértice 2 está imediatamente à esquerda de 11 e os demais, 5,10,3 e 9, à sua direita.

Para v=10, o vértice 3 é do TIPO2. Então, tem-se M=(8,12,2,11,5,6,10,1,3,9,4). Observa-se que, apesar do vértice 3 não ter sido selecionado, ele está corretamente colocado à direita do vértice 10.

As listas f(9) só têm vértices do TIPO2. Em $f_d(8)=(7)$, a última atualização é realizada em M, chegando ao resultado calculado acima.

Para implementar o procedimento e garantir a recuperação de qualquer vértice em tempo constante, ao acrescentar um vértice em M, armazena-se, junto com ele, o seu endereço. Observar que o vértice 1 não é considerado adjacente a **n**, pois não aparece em f(n).

O procedimento ORDENA_VERTICES, usado para obter M, é o seguinte:

```
procedimento ORDENA_VERTICES;
{ determinação da ordem dos vértices M a partir de }
{ f(v) = (fe(v), fc(v), fd(v)), 2≤v≤n }

começar
  . inicializar M para conter o vértice n;
  . para v:= n até 2, fazer
      começar
          . se fe(v) é não vazia
            então posicionar em M os vértices TIPO1
                  de fe(v) à esquerda de v;
          . se fd(v) é não vazia
            então posicionar em M os vértices TIPO1
                  de fd(v) à direita de v;
          . se fc(v) é não vazia
            então posicionar em M os vértices TIPO1
                  de fc(v) ao redor de v, mantendo a ordem
                  esquerda-direita igual a de fc(v);
      fim
fim;
```

Prova-se, em [12], que o procedimento ORDENA_VERTICES posiciona os vértices de $f_e(v)$ à esquerda de **v** e os de $f_d(v)$ à sua direita; e que a ordem dos vértices é determinada em tempo O(n).

3.3.3- A Embutidura Planar

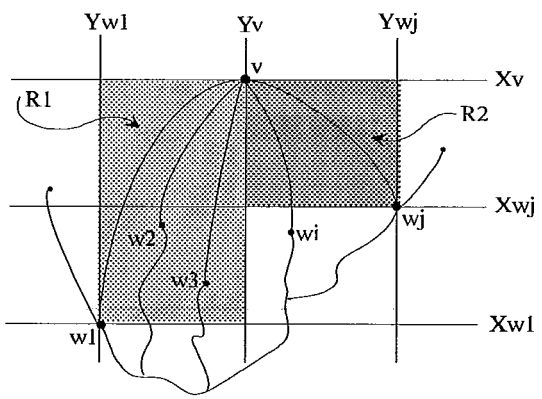
Os vértices de G podem, agora, ser posicionados no plano, em diferentes níveis horizontal e vertical, da seguinte maneira:

- o nível horizontal, ou ordenada, de cada vértice **v** é dado pela sua rotulação-st;
- o nível vertical, pela sua posição em M.

De acordo com [12], ***embutir um vértice v*** é conectá-lo aos seus vizinhos menores, na ordem especificada por f(v); então, para se poder assegurar que o desenho está livre de interseções, é necessário mostrar que a cada passo da embutidura de G, que consiste da construção das embutiduras planares dos subgrafos induzidos G_2, G_3, ..., $G_n=G$, os vértices TIPO2 de cada um aparecerão nas suas faces externas.

Considera-se o subgrafo G_k embutido no plano; para $\mathbf{v=k+1}$, seja f(v) = (w_1, w_2, ..., w_j). Ao ser embutido, o vértice **v** está claramente na face externa de G_{k+1}. Seja R_1 a região delimitada pelas retas X_{w1}, X_v, Y_{w1} e Y_v e, R_2, pelas retas X_{wj}, X_v, Y_{wj} e Y_v. Seja p_k o caminho de w_1 a w_j traçado ao longo da janela externa de G_k (ver Fig.3.18).

Fig. 3.18



Prova-se, em [12], que as arestas (v,w_1) e (v,w_j) podem ser traçadas nas regiões R_1 e R_2, respectivamente, de tal forma que o circuito formado em G_{k+1} pela adição das arestas (w_1,v) e (w_j,v) não contenha, dentro dele, nenhum vértice TIPO2 que esteja localizado em R_1 ou R_2. São os seguintes os argumentos que servem de base:

(i) a construção de M, por ORDENA_VERTICES, não admite que um vértice TIPO2 apareça entre o vértice **v** e um vértice TIPO1;

(ii) em M, todos os vértices de f_e(v) se encontram à esquerda de **v** e os de

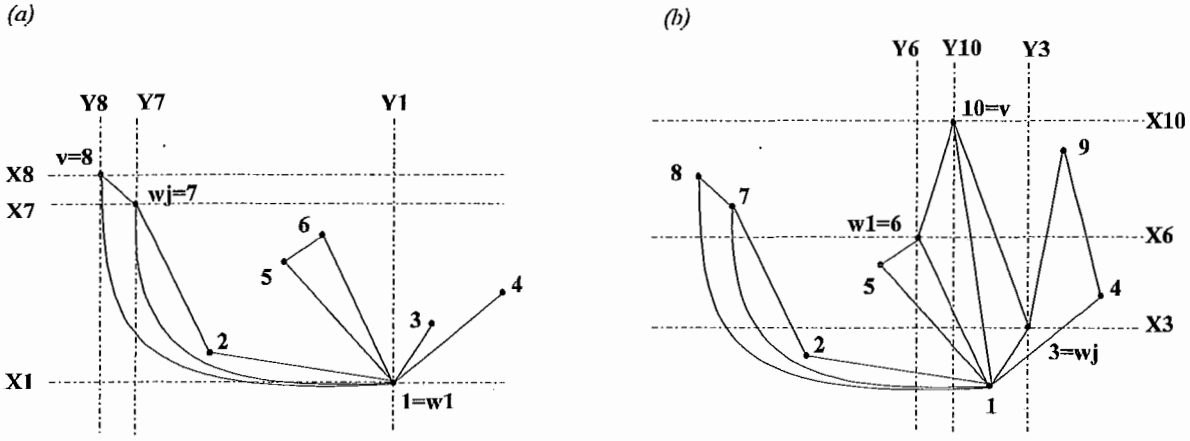
$f_d(v)$, à direita;

(iii) f(v) pode ter, no máximo, dois vértices TIPO2, originados de cada bloco de G_k, que são os extremos do Q-nó correspondente ao bloco;

(iv) quando a raiz da subárvore pertinente de T_k é processada, ela pode ter, no máximo, dois filhos parciais.

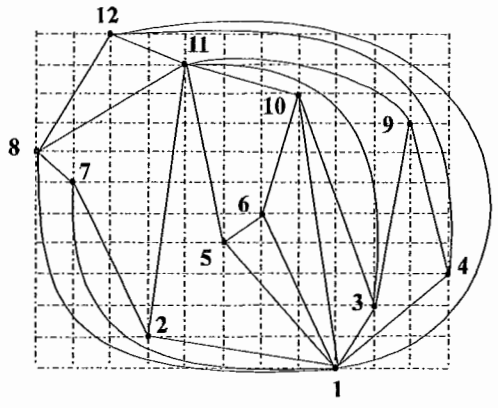
Para embutir um vértice **v**, então, primeiro desenha-se a aresta (w_1,v) dentro da região R_1, de tal forma que ela se coloque à direita de todos os vértices TIPO2 de f(v) localizados em R_1-R_2. Depois, desenham-se as arestas (w_2,v), (w_3,v), ..., (w_j,v), entrando em **v**, cada uma imediatamente à direita da sua predecessora na seqüência. A aresta (w_j,v) deverá ficar desenhada à esquerda de todos os vértices TIPO2 de f(v) que se localizam em R_2.

Fig. 3.19



As Figs.3.19a e 3.19b apresentam, respectivamente, a embutidura de v=8, com f(8)=(1,7), em G_7, e a de v=10, com f(10)=(6,1,3), em G_9.

Fig. 3.20



A embutidura final do grafo da Fig.3.10a é ilustrada na Fig.3.20. Pode-se observar que, para cada **v**, seus vizinhos menores estão na ordem contrária a dos ponteiros do relógio, a ordem de f(v); que os níveis verticais dos vértices correspondem à ordenação M calculada e que as arestas estão desenhadas nas regiões delimitadas pelo algoritmo.

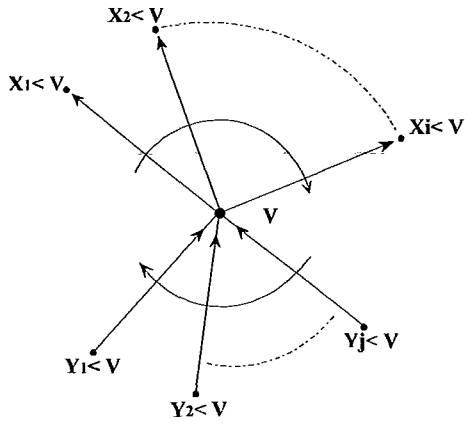
3.4- Algumas Observações sobre os Dois Algoritmos

Tanto Chiba e outros [8] como Jayakumar e outros [12] têm, como primeira etapa de seus procedimentos, calcular, para cada vértice **v**, suas vizinhanças com rótulo menor que **v**. Eles obtêm estas listas de adjacências, para cada vértice, com a mesma ordenação, sendo elas, $A_u(v)$ para o primeiro e f(v) para o segundo.

Jayakumar e outros constroem o grafo de blocos e Chiba e outros, os indicadores de direção, com o objetivo de obter, respectivamente, as listas f(v) e $A_u(v)$ corrigidas; só então, as vizinhanças de cada **v** refletem exatamente as reversões sofridas pelos Q-nós durante o processo de redução das árvores-PQ.

Depois, para chegar à embutidura final do grafo G, Chiba expande as listas de adjacências $A_u(v)$, obtendo as listas A de G, com os vizinhos maiores e menores de cada vértice na ordem dos ponteiros do relógio (ver Fig.3.21).

Fig. 3.21



Para Jayakumar, f(v) está na ordem contrária a dos ponteiros do relógio, pois ele associa ao vértice 1 a ordenada mais baixa e ao vértice **n**, a mais alta. Ele alcança a embutidura completa de G utilizando os posicionamentos dos vértices de f(v)

relativamente a **v**, com a construção da ordenação M dos vértices, e definindo regiões para o desenho das arestas.

A representação planar obtida por Jayakumar e outros se aproxima bastante de um traçado, pois G é embutido no plano com o auxílio de *coordenadas*, definidas pela rotulação-st e pela ordenação M dos vértices.

Capítulo 4

Traçado de Grafos Planares

4.1- Introdução

O último aspecto a ser abordado neste trabalho é o traçado de grafos, reconhecidos como planares, dada sua embutidura. A clareza ou uma visualização agradável podem ser alguns dos quesitos a serem atendidos por um desenho, associadas, ainda, a necessidades mais práticas. Algumas aplicações como o Desenho Automático de Circuitos VLSI, redes de comunicação diversas e interfaces gráficas, dentre outras já citadas, por consistirem de um grande número de vértices e arestas, utilizam ferramentas automáticas para o seu desenho, introduzindo, ainda, exigências, como a minimização da área total do desenho ou a do comprimento das arestas. Problemas como estes têm sido amplamente estudados, obtendo-se soluções gerais ou específicas ou provando-se que não existe solução possível em tempo polinomial.

Um grafo planar pode ser apresentado de muitas maneiras. Um tipo de traçado, abordado neste capítulo, é o que desenha o grafo dentro de um círculo, com um ciclo externo representado de forma poligonal e as demais arestas como segmentos de reta localizados dentro do polígono; um outro tipo é o traçado em grades retilíneas ortogonais, discutido no capítulo seguinte, onde as arestas são cadeias poligonais de segmentos horizontais e verticais e os vértices e as curvas das arestas têm coordenadas inteiras. Também aí, o desenho de alguns grafos planares especiais é considerado.

4.1.1- Alguns Critérios de Traçado

Certamente não há critérios absolutos que norteiam o traçado de um grafo.

No entanto, dado que o grafo em questão é planar, sabe-se que seu desenho pode ser realizado sem que suas arestas se interceptem. E mais, todo grafo planar pode ser desenhado por **segmentos de linha reta**, sem cruzamento dos mesmos. Assim, define-se um primeiro critério para o traçado de grafos planares.

CRITÉRIO 1: todas as arestas do grafo devem ser desenhadas como segmentos de reta, sem cruzamento.

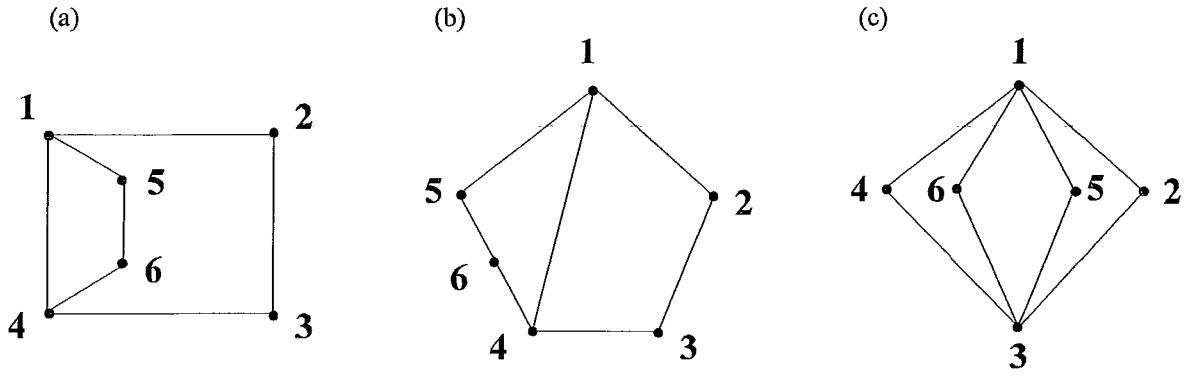
Uma das vantagens deste critério é que basta determinar o posicionamento dos vértices do grafo para obter seu desenho final.

A realização de um desenho de linhas retas sugere a utilização de formas poligonais para traçar o grafo. Assim, sendo uma *face* o interior de cada região delimitada pelas arestas de uma representação planar de um grafo G, tenta-se desenhar G de tal forma que todas as faces sejam polígonos convexos. Um desenho como este é chamado **convexo** e origina um segundo critério de traçado de grafos planares.

CRITÉRIO 2: as fronteiras de cada face devem ser desenhadas como polígonos convexos.

Este critério, no entanto, não se aplica a todos os grafos planares, pois há alguns que não podem ser desenhados de forma convexa (ver Fig.4.1c). Outros, ainda, têm traçado convexo somente para um ciclo facial externo convenientemente escolhido, definindo-se *ciclo facial externo* como a fronteira da face externa.

Fig. 4.1



Neste capítulo são apresentados um algoritmo que realiza o traçado convexo de um grafo dado, se isto for possível, e um outro que verifica se este traçado convexo existe; no caso afirmativo, é obtido um ciclo externo conveniente para o desenho (observar as Figs. 4.1a e 4.1b). Ambos os algoritmos são lineares e foram desenvolvidos

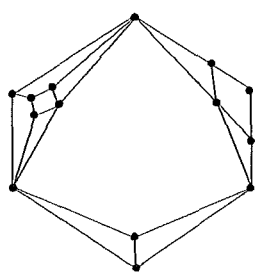
por Chiba-Yamanouchi-Nishizeki [15,9]. Eles se baseiam em resultados de Tutte e Thomassen [15,9], ali revisados, que estabelecem uma condição necessária e suficiente para um grafo planar ter um traçado convexo.

O fato de todo grafo planar 3-conexo ter traçado convexo, resultado provado por Tutte e revisado por [15], dá origem a uma outra forma de desenhar um grafo planar, que é chamada **traçado agradável**. Ela consiste em decompor o grafo planar dado nas suas componentes 3-conexas e desenhar, cada uma delas, de forma convexa. O resultado final, isto é, o traçado do grafo completo, provavelmente não atenderá à definição de *convexo*, mas, em geral, será um desenho visualmente mais agradável. Observar as Figs. 4.2a e 4.2b, que apresentam, respectivamente, um traçado convexo e um agradável para o mesmo grafo. Define-se, então, um terceiro critério:

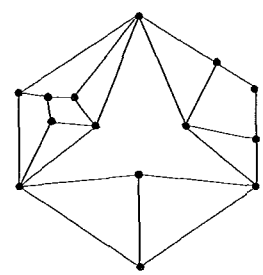
CRITÉRIO 3: o ciclo facial externo de uma componente 3-conexa deve ser desenhado como um polígono convexo.

Fig. 4.2

(a)



(b)



Este critério tem a vantagem de poder ser aplicado a qualquer grafo planar 2-conexo, pois as componentes 3-conexas deste, que certamente têm traçado convexo, são desenhadas isoladamente.

Os tipos de traçado apresentados, bem como os critérios definidos, não são os únicos existentes para o desenho de um grafo planar. Um outro tipo - a grade - é comentado no capítulo seguinte.

4.2- Traçado Convexo

O algoritmo aqui apresentado foi desenvolvido por Chiba e outros [15,9] com

o objetivo de obter um traçado convexo para uma representação planar 2-conexa G dada, procurando atender aos critérios 1 e 2 definidos na seção 4.1. Este procedimento assume que o grafo G pode ter um traçado convexo para um ciclo facial externo previamente determinado, embora isto não seja verdade para qualquer grafo planar. A seção seguinte deste capítulo apresenta um teste de convexidade para G, que verifica se a ele é possível associar um traçado convexo e determina, no caso afirmativo, um ciclo facial externo adequado para o desenho.

Seu tempo de processamento e espaço de armazenamento são lineares com relação ao número de vértices do grafo.

4.2.1- Definições Preliminares

A seguir, algumas definições:

-**Faces** são as regiões conexas do plano delimitadas pelas arestas de uma representação planar do grafo;

-**Ciclo Facial** é a fronteira de uma face;

-**Ciclo Facial Externo** ou **Ciclo Externo** é a fronteira da face externa;

-**Componentes Conexas** são os subgrafos conexos maximais;

-**Bloco** é um subgrafo maximal 2-conexo ou uma componente conexa que não tem vértice de corte. **(DEF.4.1)**

Define-se também: o **traçado convexo** de um grafo planar é uma representação do grafo no plano, onde todas as arestas são desenhadas por segmentos de reta sem cruzamento e as fronteiras de todos os ciclos faciais são polígonos convexos. **(DEF.4.2)**

Seja **S** o ciclo facial externo de um grafo G e seja **S*** um desenho poligonal de S ou, simplesmente, um polígono convexo de S. Diz-se que **S*** é **ampliável** se existe um desenho convexo de G que tenha S* como polígono externo. **(DEF.4.3)**

É importante observar que nem todos os vértices do ciclo S precisam, necessariamente, ser ápices (ou vértices geométricos) do polígono convexo S*. E, ainda, uma vez que o ciclo S é assumido como já determinado, se G não tiver um traçado convexo para um dado S* de S, não quer dizer que não possa tê-lo para um outro polígono convexo S'* de outro ciclo facial externo S'; estas considerações são objeto da seção 4.3.

A idéia básica do algoritmo de [15,9] é *expandir* um polígono convexo S* de um ciclo facial externo S de uma representação planar 2-conexa G em um traçado convexo de G; assim, inicialmente, desenha-se o polígono convexo S*, onde se posicionam os vértices de S; a partir daí, o algoritmo procura localizar os vértices restantes de G no interior do polígono, de forma a obter faces convexas.

4.2.2- Descrição do Algoritmo

Em [9], encontram-se revisados alguns resultados de outros autores. Dentre eles, o lema a seguir, que estabelece uma condição necessária e suficiente para uma representação planar ter um traçado convexo.

LEMA 4.1: Seja G uma representação planar 2-conexa; seja S um ciclo facial externo de G com polígono convexo S* e com conjunto de vértices denotado por V(S); sejam p_1, p_2,..., p_k os caminhos em S, cada um correspondendo a um lado do polígono S*. Então **S* é ampliável** se e somente se a **CONDIÇÃO I** abaixo for satisfeita:

(i) para cada vértice **v** de G, que tenha grau mínimo 3 e que não esteja em S , existem três caminhos disjuntos, exceto para **v**, unindo **v** e um vértice de S;

(ii) G-V(S) não tem nenhuma componente conexa C, tal que todos os vértices de S, adjacentes a vértices de C, estejam localizados num único caminho p_i; e nunca dois vértices, num mesmo caminho p_i, são unidos por uma aresta que não pertença a S;

(iii) qualquer ciclo de G, que não tenha arestas em comum com S, tem no mínimo três vértices de grau maior ou igual a 3 em G.

A seguir são discutidas algumas aplicações do lema anterior. O grafo da Fig.4.3a tem ciclo facial externo S=1,2,3,4,5,1 e polígono convexo S* com ápices nos mesmos vértices. S* é ampliável, pois obedece a todos os itens da Condição I:

(i) os vértices 6 e 7 não estão em S e têm grau 3; cada um tem três caminhos disjuntos para vértices de S;

(ii) a aresta (8,9), componente conexa de G-V(S), tem vértices adjacentes a dois caminhos distintos de S; não há vértices, em nenhum caminho p_i de S, ligados por uma aresta que não esteja em S;

(iii) o ciclo 6,2,9,8,5,6 tem três vértices com grau maior ou igual a 3.

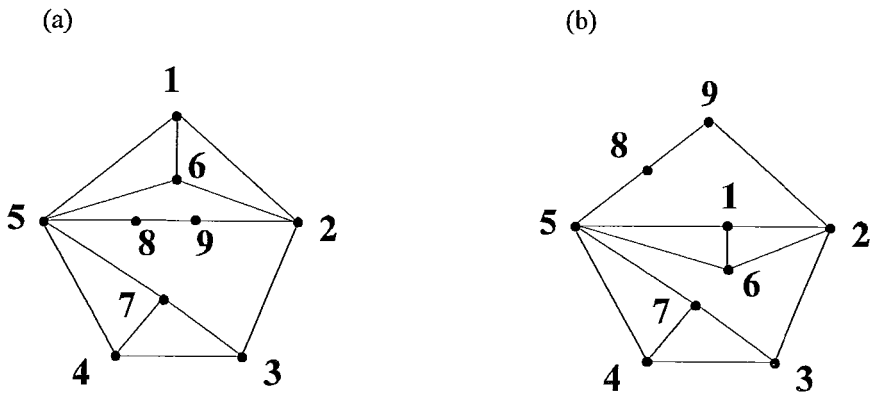
Já no grafo da Fig.4.1c, o polígono convexo S* de ápices 1, 2, 3 e 4 não é ampliável, pois não atende à Condição I no item (iii): o ciclo 1,5,3,6, que não tem

arestas em comum com S, só tem vértices de grau 2.

O grafo da Fig.4.1a, também tem um polígono externo S*, de ápices 1, 2, 3 e 4, não ampliável, não obedecendo ao item (ii) da Condição I: a componente conexa (5,6) tem todos os vértices de S adjacentes a ela num mesmo caminho.

Na Fig.4.3b o mesmo grafo da Fig.4.3a é apresentado com outro ciclo facial externo. Seu polígono convexo S*, com ápices 2, 3, 4, 5 e 9, não é mais ampliável, pois os vértices 1 e 6 não obedecem ao item (i) do lema. Com este ciclo externo não se pode obter, portanto, um traçado convexo para o grafo.

Fig. 4.3



Por outro lado, a Fig.4.1b ilustra um desenho convexo para o grafo da Fig.4.1a; o polígono convexo S* com ápices 1, 2, 3, 4 e 5 é, agora, ampliável.

O Lema 4.1 é a base do teste de convexidade, a ser detalhado na seção seguinte deste capítulo. Segundo ele, somente se o polígono convexo S* do ciclo facial externo S de G for ampliável, é que se pode obter um traçado convexo para G. Como o algoritmo de traçado de [15,9] é apresentado antes do teste de convexidade então, para delinear os passos do procedimento, assume-se que, dados uma representação planar G 2-conexa e um ciclo externo S de G, S* é ampliável.

Inicialmente, tem-se determinada a posição dos vértices de S nos lados e ápices de S*; para localizar os vértices de G-V(S), o algoritmo reduz G a diversos subgrafos, obtidos da seguinte forma:

- elimina-se um vértice v arbitrário de S*, juntamente com as arestas que lhe são incidentes;

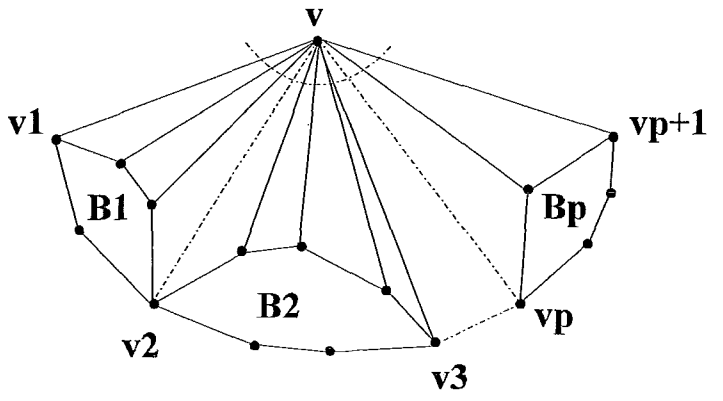
- divide-se o subgrafo G' resultante, G'=G-v, nos blocos B_1, B_2, ..., B_p, $p \geq 1$ (ver Fig.4.4);

- determina-se um polígono convexo S^*_i do ciclo externo de cada B_i, de tal forma que B_i com S^*_i satisfaça à Condição I do Lema 4.1; neste ponto tem-se a

localização dos vértices de S_i;

- recursivamente, aplica-se o algoritmo a cada B_i com polígono convexo S^*_i, para determinar a posição dos vértices de B_i que não estão em S^*_i.

Fig. 4.4



Antes de detalhar o algoritmo em questão, duas considerações merecem atenção. Primeiramente, se a representação planar G dada não for 2-conexa, esta deve ser tratada de uma das duas maneiras abaixo:

(i) subdividir o grafo nas suas componentes 2-conexas e utilizar o algoritmo, separadamente, para traçar de forma convexa cada uma delas; depois combiná-las para obter o traçado inteiro de G; ou

(ii) aumentar a representação dada para uma 2-conexa; traçar o grafo resultante através do algoritmo e, depois, eliminar as arestas que foram adicionadas, obtendo o traçado do grafo original.

O segundo ponto consiste em analisar se existem vértices de grau 2 fora do ciclo facial externo S na representação planar de G; neste caso, para cada v de grau 2 em G-V(S), substituir v e as arestas que lhe são incidentes por uma única aresta, unindo os dois vértices adjacentes a v. Feito isto, o algoritmo pode ser chamado para executar o traçado de G; após seu término, cada vértice eliminado é localizado no segmento de reta que une os dois vértices adjacentes a ele.

O algoritmo completo é detalhado como se segue.

```
procedimento TRAÇO_CONV (G,S,S*);
{ dada uma representação planar 2-conexa G e um polígono convexo
ampliável S* do ciclo externo S, este processo expande S* num traçado
convexo de G; supõe-se que G não tem vértices de grau 2 fora de S}
```

```
começar
  .se G tem pelo menos 4 vértices
   {caso contrário, o traçado de G já foi obtido}
     começar
       . selecionar um ápice v arbitrário de S*;
       . seja G'= G - v;
       . dividir G' nos blocos B_i, 1≤i≤p;
       { ver Fig.4.4 }
       . sejam v_1 e v_p+1 os dois vértices em S, adjacentes a
         v, com v_1 ∈ V(B_1) e v_p+1 ∈ V(B_p);
       . sejam v_2,v_3,...,v_p os vértices de corte de G';
       . seja o vértice v_i = V(B_i-1) ∩ V(B_i), 2≤i≤p;
       . para cada bloco B_i, 1≤i≤p, fazer
           começar
             . seja o triângulo de vértices v, v_i, v_i+1;
             . selecionar os vértices de V(S_i) que não
               pertencem a V(S);
             . posicioná-los no interior do triângulo v, v_i,
               v_i+1, de tal maneira que:
                 . os vértices adjacentes a v sejam ápices
                   do polígono convexo S*_i;
                 . os vértices não adjacentes a v estejam
                   nos segmento de reta de S*_i;
             { expansão de S*_i em um desenho convexo de B_i }
             . TRAÇO_CONV (B_i,S_i,S*_i);
           fim
     fim
fim;
```

Observações:

- os vértices v_i, $1 \leq i \leq p+1$, pertencem necessariamente a S, pois o polígono convexo S* em G satisfaz à Condição I e todos os vértices de G, que não se encontram em S, têm grau maior que 2;

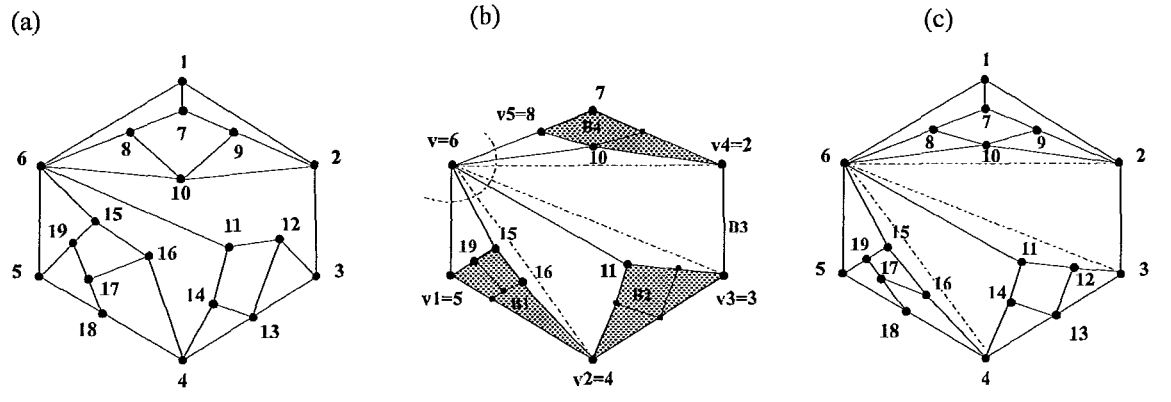
- para cada bloco B_i, com ciclo facial externo S_i, a segunda parte do procedimento determina um polígono convexo S^*_i, pelo posicionamento dos vértices de $V(S_i)$ que não pertencem a V(S), uma vez que os vértices comuns já têm posição definida em S*.

O grafo G da Fig.4.5a é planar e biconexo. Sejam o ciclo externo S=1,2,3,13,4,18,5,6,1 e o polígono S*, de S, com ápices 1, 2, 3, 4, 5 e 6. S* é ampliável, pois satisfaz à Condição I; também G não tem vértices de grau 2 fora de S. A posição dos vértices de S está perfeitamente determinada; basta, por exemplo, traçar uma circunferência e nela localizar os vértices de S*; os demais vértices de S ficam nos segmentos de reta de S*.

Aplicando o algoritmo TRAÇO_CONV sobre G, S e S*, elimina-se o vértice v=1. Neste caso, o grafo G' resultante só tem um bloco, pois é, ele próprio, biconexo.

Seu ciclo externo S_1 é 2,3,13,4,18,5,6,8,7,9. Em relação ao ciclo S anterior, os vértices a posicionar são 7, 8 e 9. Destes, só o vértice 7 é adjacente a v=1 sendo, portanto, ápice do polígono convexo S^*_1 e localizado no interior do triângulo de vértices 1, 6 e 2; os vértices 8 e 9 ficam nos segmentos de reta de S^*_1, cujos ápices são 2,3,4,5,6 e 7.

Fig. 4.5

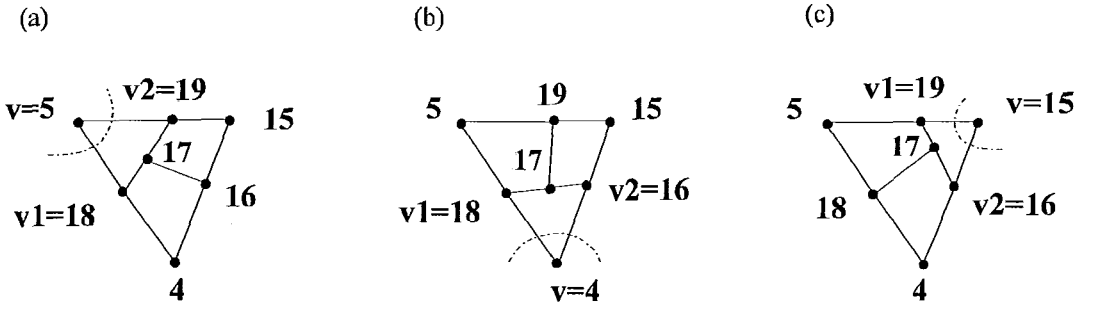


O procedimento recursivo recebe agora o grafo G' com ciclo externo S_1 e polígono convexo S^*_1 para análise.

Selecionando o ápice v=6 de S^*_1 e eliminando-o, o grafo resultante se resume a quatro blocos, cujos vértices de corte são 4, 3 e 2; os vértices extremos, adjacentes a v=6 são 5 e 8 (ver Fig.4.4b).

O terceiro bloco se resume a uma aresta, não sendo tratada pelo algoritmo, pois todos os seus vértices estão posicionados. Os blocos restantes vão ser localizados nos triângulos 6-5-4, 6-4-3 e 6-2-8; observa-se que as arestas (6,4), (6,3) e (6,2) não existem de fato no grafo, mas sendo 4, 3 e 2 os vértices de corte, são elas que definem os triângulos.

Fig. 4.6



No primeiro bloco (B_1, na Fig.4.5b), os vértices a posicionar são 15, 16 e 19; o único vértice do ciclo externo ligado a v=6 é o 15 que, portanto, vai ser ápice do polígono externo de B_1 e colocado no interior do triângulo 6-5-4; os vértices 16 e 19

ficam nos segmentos de reta do polígono externo de B_1.

O bloco B_1 vai gerar ainda outra recursão para localizar o vértice 17. Este poderá ter posicionamentos um pouco diferentes, dependendo do ápice do polígono externo selecionado para eliminação (ver Fig.4.6), mas sempre estará nos segmentos de reta do polígono externo, pois não é ápice em nenhum dos casos.

O procedimento de localização dos vértices dos demais blocos é análogo.

Um traçado convexo para o grafo da Fig.4.5a é apresentado na Fig.4.5c.

TEOREMA 4.1: Seja G uma representação planar 2-conexa; sejam S um ciclo externo e S* um polígono convexo ampliável de S. Então o algoritmo TRAÇO_CONV expande S* num desenho convexo de G e usa tempo e espaço lineares [15,9].

A prova da correção do algoritmo é dada por indução; a base é que cada bloco B_i com S^*_i satisfaz à Condição I do Lema 4.1. Para armazenar a representação planar são usadas listas de adjacências duplamente ligadas, que mantêm a ordem dos ponteiros do relógio ao redor dos vértices; esta estrutura orienta a prova de utilização de espaço linear. A prova de tempo linear se baseia na quantidade de arestas atravessadas durante a execução do algoritmo, que será, no máximo, igual a |E|, no total.

4.3- Teste de Convexidade

Há grafos que só têm traçado convexo para determinados ciclos faciais externos (ver Figs 4.1a e 4.3a), mas não para todos. Outros, apesar de biconexos, não obtêm traçado convexo para nenhum ciclo externo (ver Fig.4.1c). Ainda outros tem traçado convexo para qualquer ciclo externo, como é o caso, por exemplo, dos grafos 3-conexos, cujos ciclos faciais são todos ampliáveis [9]. O objetivo deste item é estabelecer condições de existência [15] de algum ciclo facial de um grafo G 2-conexo, sob o qual ele tem um traçado convexo.

A Condição I do Lema 4.1 define um teste de convexidade para um polígono convexo S* de um ciclo facial externo de uma representação planar 2-conexa G. Se S* não for ampliável, outros ciclos externos de G e seus polígonos convexos poderiam ser testados. No entanto, uma representação planar G pode ter um número exponencial de ciclos faciais, o que tornaria tal procedimento impraticável. Aqui é, então, apresentado

um algoritmo linear que determina se um grafo planar 2-conexo pode ter um traçado convexo; para tanto, uma nova condição será estabelecida, com base nas definições apresentadas a seguir.

4.3.1- Definições Preliminares

Seja G=(V,E) um grafo 2-conexo.

Um par de vértices {x,y} de G é um **par de separação** se existem dois subgrafos $G'_1=(V_1,E'_1)$ e $G'_2=(V_2,E'_2)$, que satisfazem às seguintes condições:

(i) $V = V_1 \cup V_2$, $V_1 \cap V_2 = \{x,y\}$;

(ii) $E = E'_1 \cup E'_2$, $E'_1 \cap E'_2$ é um conjunto vazio, $|E'_1| \geq 2$ e $|E'_2| \geq 2$.

(DEF.4.4)

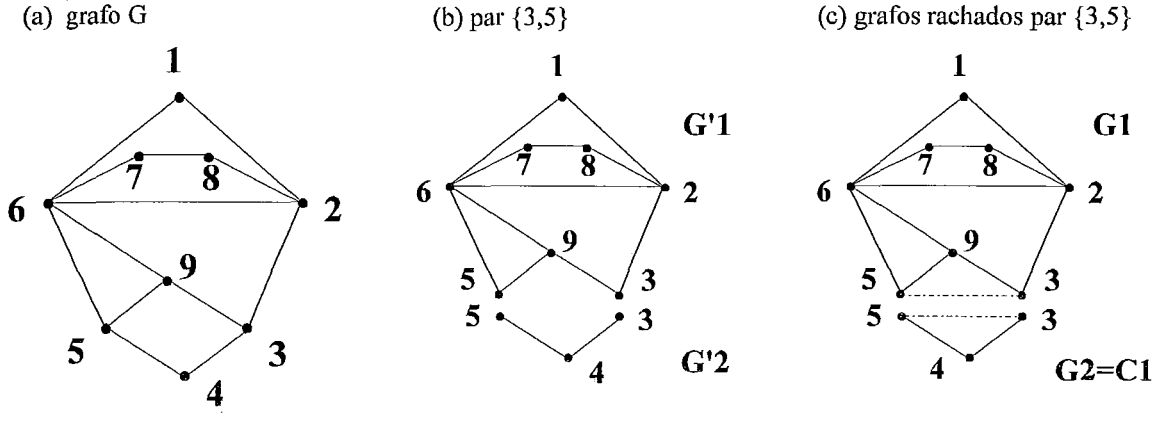
Observar que, se a aresta (x,y) não existe em G, G'_1 é o subgrafo induzido por V_1 e G'_2 o subgrafo induzido por V_2. Se (x,y) existe, a aresta pertencerá a um dos dois subgrafos.

Chama-se **aresta virtual** a uma nova aresta (x,y) a ser acrescentada aos subgrafos G'_1 e G'_2. Assim, para um par de separação {x,y}, os **grafos rachados** de G são G_1 e G_2, obtidos, respectivamente, de G'_1 e G'_2 pelo acréscimo da aresta virtual (x,y). **(DEF.4.5)**

Observa-se, ainda, que:

- mesmo que G seja um grafo simples, seus grafos rachados G_1 ou G_2 podem ter arestas múltiplas;
- um grafo 3-conexo não tem pares de separação.

Fig. 4.7



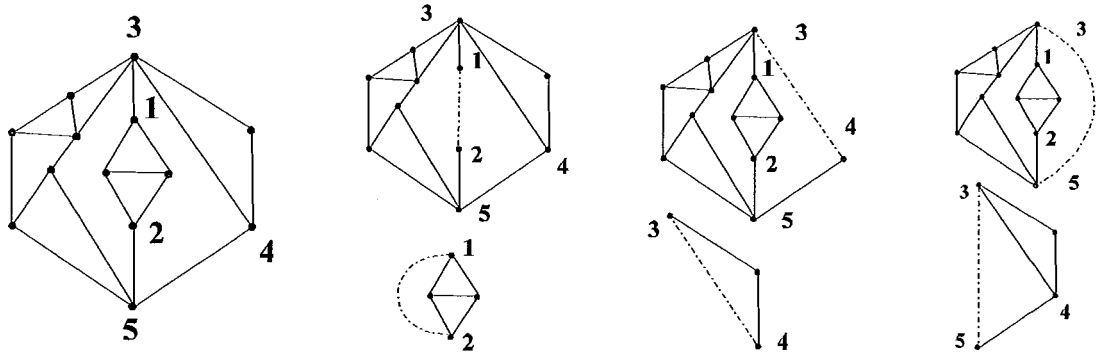
A Fig.4.7a apresenta um grafo G. O par de vértices {5,3} é um par de

separação, de acordo com a DEF.4.4; os subgrafos G'_1 e G'_2 estão na Fig.4.7b e os subgrafos rachados G_1 e G_2, nos quais a aresta virtual (3,5) é representada por uma linha tracejada, na Fig.4.7c.

A Fig.4.8a apresenta um outro grafo G, cujos pares de separação são {1,2}, {3,4} e {3,5}. Os subgrafos rachados para cada par de separação estão nas Figs.4.8b, 4.8c e 4.8d.

Fig. 4.8

(a) G (b) grafos rachados{1,2} (d) grafos rachados{3,5}



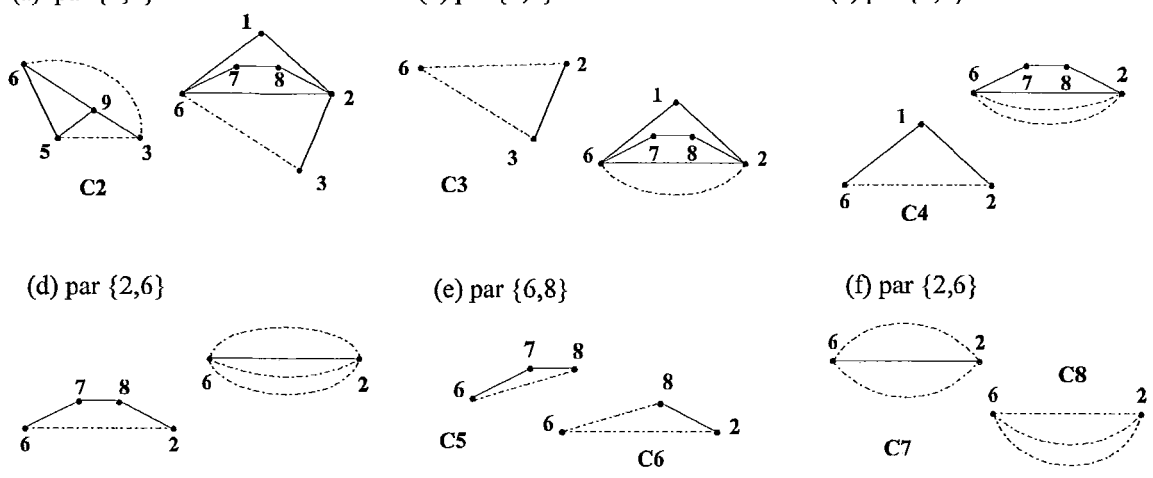
(c) grafos rachados{3,4}

Se o grafo G é rachado, e esses grafos são rachados de novo e, assim por diante, até não existir mais nenhum par de separação que permita continuar o processo, os grafos obtidos são chamados **componentes rachadas de G**. Elas podem ser de três tipos:

-laços triplos, isto é, um conjunto de três arestas múltiplas;
-triângulos, isto é, um ciclo de três arestas; ou
-grafos 3-conexos. **(DEF.4.6)**

Fig. 4.9

(a) par {3,6} (b) par {2,6} (c) par {2,6}



(d) par {2,6} (e) par {6,8} (f) par {2,6}

Na Fig.4.7c, o subgrafo G_2 já é uma componente rachada do grafo G da Fig.4.7a, pois é um triângulo. A Fig.4.9 ilustra a continuação do processo, a partir do subgrafo G_1 da Fig.4.7c, mostrando todas as demais componentes rachadas de G. Elas são ao todo oito: dois laços triplos (C_7,C_8), cinco triângulos (C_1,C_3,C_4,C_5,C_6) e um grafo 3-conexo (C_2).

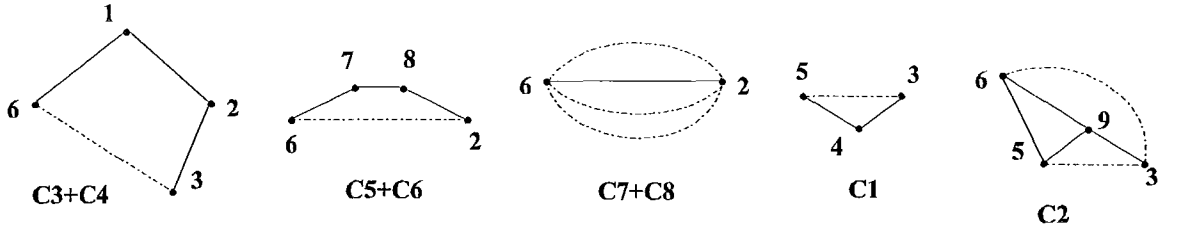
Um **laço** é um conjunto de arestas múltiplas; um **anel** é um ciclo. As **componentes 3-conexas de G** são obtidas das componentes rachadas de G, juntando laços triplos, oriundos de um mesmo par de separação, em um laço, e triângulos, também de um mesmo par de separação, em um anel, tanto quanto possível.

(DEF.4.7)

As componentes rachadas de G não são únicas, mas as componentes 3-conexas, sim.

Agrupando as componentes rachadas do grafo da Fig.4.7a, conforme a DEF.4.7, obtêm-se as suas componentes 3-conexas, ilustradas na Fig. 4.10.

Fig. 4.10 - Componentes 3-conexas



Seja {x,y} um par de separação de um grafo G; se G é rachado sucessivamente em {x,y}, até não ser mais possível, os grafos construídos desta maneira e que têm, pelo menos, uma aresta real são chamados **componentes rachadas-{x,y}**.

(DEF.4.8)

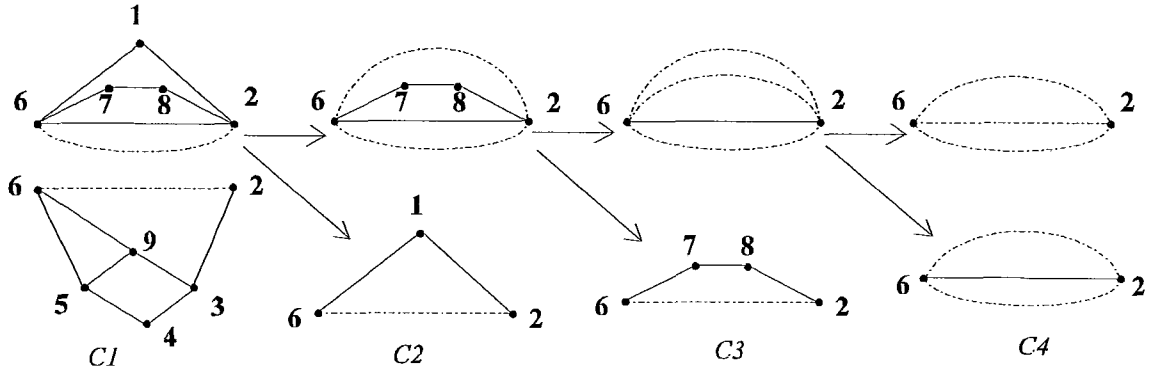
A Fig.4.11 mostra a construção das componentes rachadas-{2,6} do grafo da Fig.4.7a; são elas C_1, C_2, C_3 e C_4, pois, somente elas tem pelo menos uma aresta real.

Um par de separação {x,y} é dito **original** se **x** e **y** são vértices extremos de uma aresta virtual de uma componente 3-conexa. Um par de separação original {x,y} é **proibido** se produz quatro ou mais componentes rachadas-{x,y} **ou** três componentes rachadas-{x,y}, nenhuma delas sendo laço ou anel. Um par de separação original {x,y} é

crítico se produz três componentes rachadas-{x,y}, sendo uma delas um laço ou um anel **ou** duas componentes rachadas-{x,y}, nenhuma delas sendo anel. **(DEF.4.9)**

No grafo da Fig.4.7a, os pares de separação {2,6}, {3,6} e {3,5} são originais. O par de separação {2,6} é proibido e o par {3,6} é crítico; já o par {3,5} não é nem crítico nem proibido, apesar de ser um par original. O par de separação {6,8} não é original, pois os vértices 6 e 8 não são extremos de aresta virtual de nenhuma componente 3-conexa, e, portanto, não se enquadra nas definições de proibido ou crítico.

Fig. 4.11 - Componentes Rachadas-{2,6}



4.3.2- Fundamentos e Descrição do Algoritmo

O lema 4.1 estabelece como condição necessária e suficiente para qualquer grafo planar ter um traçado convexo, que o polígono convexo S* do ciclo facial externo do grafo seja ampliável. O lema a seguir define a **Condição II**, que é equivalente à Condição I, conforme demonstra [15], desde que se observe a restrição de S* ser *estrito*. Define-se: S* é **estrito** se todo vértice de S for ápice de S*. **(DEF.4.10)**

LEMA 4.2: Seja G=(V,E) uma representação planar 2-conexa com ciclo facial externo S; seja S* um polígono convexo estrito de S. Sejam p_1, p_2,..., p_k os caminhos em S, cada um correspondendo a um lado do polígono S*. Então **S* é ampliável** se e somente se a **CONDIÇÃO II** abaixo for satisfeita:

(i) G não tem nenhum par de separação proibido;

(ii) para cada par de separação crítico {x,y} de G, existe no máximo uma componente rachada-{x,y} que não tem nenhuma aresta em comum com S. E mais, essa componente deve ser um laço, se (x,y) pertencer a E, ou um anel, se (x,y) não pertencer a E.

Observa-se que, se G é uma representação planar 2-conexa com ciclo facial externo S e este tem um polígono convexo ampliável, então o polígono convexo estrito de S é ampliável [15].

Os seguintes teoremas, corolários e lemas, somente enunciados, fornecem a base para que o algoritmo de teste de convexidade possa ser formulado posteriormente. Todos estes resultados, e suas respectivas provas, se encontram em [15].

TEOREMA 4.2: Um ciclo facial S, de um grafo planar 2-conexo G, é ampliável se e somente se S e G satisfazem à Condição II.

COROLÁRIOS:

4.1- Um grafo planar 2-conexo G não tem traçado convexo se G tem um par de separação proibido.

4.2- Um grafo planar 2-conexo G tem traçado convexo para qualquer ciclo facial, se G não tem nem par de separação crítico nem par de separação proibido.

4.3- Todo grafo planar 3-conexo G tem traçado convexo para qualquer ciclo facial de G.

4.4- Toda representação planar G tem um desenho de linhas retas.

4.5- Se um ciclo facial S de um grafo planar 2-conexo G satisfaz à Condição II, então S contém todos os vértices dos pares de separação críticos de G.

A partir dos resultados vistos até aqui, pode-se observar que os pares de separação originais desempenham um papel fundamental para o teste de convexidade de G, pois, se há algum par de separação proibido, G não tem traçado convexo (corolário 4.1), como é o caso do grafo da Fig.4.7a; se não há pares críticos e nem proibidos, G tem traçado convexo para qualquer ciclo facial (corolário 4.2). Ainda conclui-se que, se um ciclo facial tem polígono convexo estrito ampliável, nele se localizam todos os vértices dos pares críticos de G (corolário 4.5). Este último resultado é reforçado pelo teorema a seguir; antes, [15] formula dois lemas.

LEMA 4.3: Seja G um grafo planar 2-conexo que não tem par de separação proibido e tem exatamente um par de separação crítico. Então G tem um traçado convexo.

LEMA 4.4: Seja G um grafo planar 2-conexo e seja {x,y} um par de separação original de G. Se um ciclo facial S de G contém os vértices **x** e **y**, então exatamente duas componentes rachadas-{x,y} contêm arestas de S.

Seja G=(V,E) um grafo planar 2-conexo que não tem par de separação proibido e tem dois ou mais pares de separação críticos. Aplicar a seguinte operação a

todos os pares de separação críticos {x,y} de G:

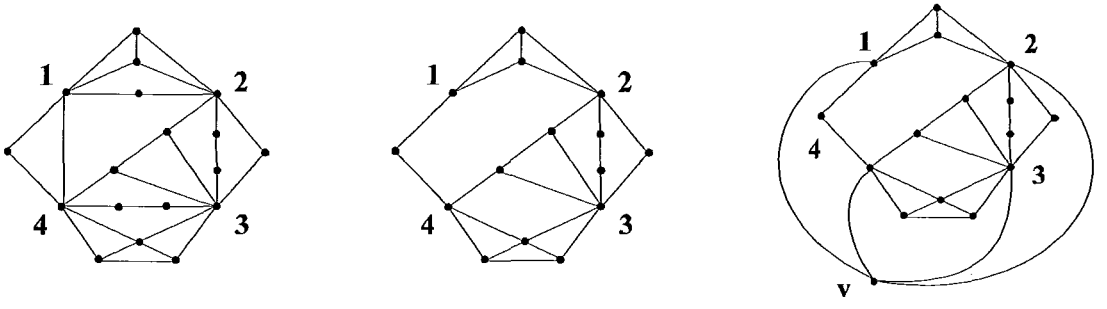
- se (x,y) pertence a E então eliminar a aresta (x,y) de G;

- se (x,y) não pertence a E e dentre as componentes rachadas-{x,y} só há exatamente um anel, então eliminar o caminho de **x** para **y** que se encontra na componente de G.

TEOREMA 4.3: Seja G_1 o grafo resultante da operação acima, aplicada a G. Então **S é um ciclo facial ampliável de G se e somente se S é um ciclo facial de G_1 que contém todos os vértices dos pares de separação críticos de G.**

Fig. 4.12

(a) grafo G (b) G1 (c) G2



A Fig 4.12a apresenta um grafo G=(V,E) 2-conexo, que não tem par de separação proibido e tem cinco pares de separação críticos. Seguindo a operação definida no Teorema 4.3, para o par de separação crítico {1,4} a aresta (1,4) será retirada de G, pois pertence a E. Para os pares {1,2}, {3,4}, {2,3} e {2,4} não existem arestas correspondentes. No entanto, os dois primeiros têm, ambos, entre suas componentes rachadas-{1,2} e {3,4} exatamente um anel; então os caminhos 1-2 e 3-4 serão eliminados. O par {2,3} tem dois anéis entre suas componentes rachadas-{2,3} e o par {2,4} nenhum; então nada, referente a estes dois pares, poderá ser eliminado.

O grafo G_1, resultante da operação acima, é mostrado na Fig. 4.12b. O ciclo facial S de G_1 tem, por enquanto, somente quatro vértices conhecidos; ele será totalmente determinado a partir dos corolários do Teor.4.3.

Retornando ao grafo da Fig.4.1a e retirando a aresta (1,4), referente ao par de separação crítico {1,4}, o polígono de ápices 1,2,3,4,6 e 5 propicia um traçado convexo.

O teorema 4.3 é, agora, o resultado básico para a definição do teste de convexidade. Sob o condicionamento de todos os pares críticos estarem num único ciclo facial, os dois corolários a seguir vão reduzir o algoritmo de [15] a um teste de planaridade.

Antes, define-se: Seja **v** um vértice de uma representação planar 2-conexa G_2; seja $G_1 = G_2 - v$, também 2-conexo. Então, o **v-ciclo de G_2** (ver Fig.4.12c) é o ciclo da representação planar do subgrafo G_1 de G_2, que limita a face de G_1 na qual **v** está localizado. **(DEF.4.11)**

Seja G_2 grafo da Fig.4.12c; o ciclo que limita a face de $G_1 = G_2 - v$, onde **v** está localizado, é o próprio ciclo facial externo de G_1, o qual é, portanto, o v-ciclo de G_2.

Seja G um grafo planar 2-conexo sem par de separação proibido e com dois ou mais pares de separação críticos. Os corolários do teorema 4.3 são:

COROLÁRIO 4.6: Seja G_1 o grafo definido no teorema 4.3; seja G_2 o grafo obtido a partir de G_1, adicionando um novo vértice **v** e unindo-o a todos os vértices dos pares de separação críticos de G. Então S é um ciclo facial ampliável de G se e somente se:

(i) G_2 é planar; e

(ii) S é o v-ciclo de uma embutidura planar de G_2.

COROLÁRIO 4.7: Seja G2 o grafo definido no corolário 4.6. Então G tem um traçado convexo se e somente se G_2 é planar.

Finalmente, com base nos Corolários 4.1, 4.2, 4.6 e 4.7 e no Lema 4.3, o algoritmo pode ser formulado; assim, para decidir se G pode ter um traçado convexo, é testada a planaridade do grafo G_2, definido no Corolário 4.6. O teste de convexidade de um grafo G se resume, então, ao teste de planaridade do grafo G_2, construído a partir de G.

Sejam G o grafo da Fig.4.12a e G_1 o da Fig.4.12b; adicionando um vértice **v** e ligando-o aos vértices dos pares de separação críticos de G, obtém-se G_2, apresentado na Fig.4.12c; como G_2 é planar, então o teste de convexidade pode concluir que G tem traçado convexo para o v-ciclo S da embutidura de G_2.

4.3.3- Detalhes de Implementação

O procedimento CONVEX_TESTE é apresentado a seguir. Ele verifica se, dado um grafo planar 2-conexo G, este pode ou não ter um traçado convexo; em caso afirmativo, o algoritmo encontra o ciclo facial ampliável de G.

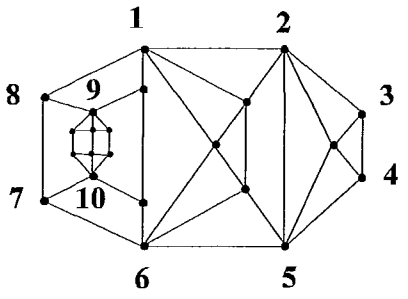
```
procedimento CONVEX_TESTE (G);
{ G é um grafo planar 2-conexo dado; este procedimento verifica se G pode
ou não ter um traçado convexo; em caso afirmativo, ele encontra o ciclo
facial ampliável de G}

começar
  . achar todos os pares de separação de G, a partir da
      determinação das suas componentes 3-conexas;
  . guardar no conjunto CSP os pares de separação críticos;
  . guardar no conjunto FSP os pares de separação proibidos;
  . se |FSP| = 0
    então imprimir  " G não tem traçado convexo ";
    senão
      .se |FSP| = |CSP| = 0
       então imprimir  "  Todos os ciclos faciais são
                        ampliáveis. G tem traçado convexo.";
       senão
         . se |CSP| = 1
           então imprimir  " G tem traçado convexo ";
           senão
             começar
               . obter o grafo G1 de G, aplicando a operação
                   definida no Teor.4.3 aos vértices de CSP;
               . construir o grafo G2 a partir de G1,
                   adicionando um vértice v, ligado a todos
                   os vértices de CSP;
               . testar a planaridade de G2;
               . se G2 for não planar
                 então
               { Não há nenhum ciclo facial contendo
                 todos os vértices de CSP }
                   . imprimir   "G não tem traçado convexo";
                 senão
                   começar
                    . seja S o v-ciclo de G2 que, portanto,
                       satisfaz à Condição II;
                    . imprimir  "G tem traçado convexo.
                           Seu ciclo facial ampliável é S";
                   fim
             fim
fim.
```

O grafo da Fig.4.12c, G_2, é planar e, portanto, o ciclo facial de G_1 (ver Fig.4.12b) é ampliável e contém todos os pares de separação críticos de G (ver Fig.4.12a).

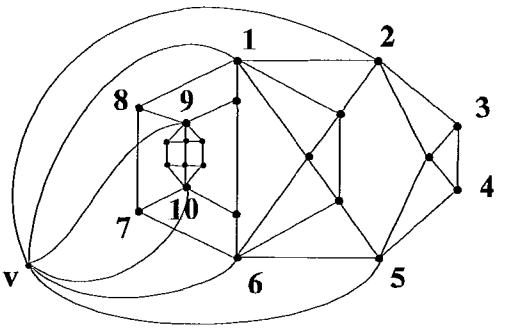
Já o grafo da Fig.4.13, apesar de 2-conexo, planar, sem par de separação proibido e com três pares de separação críticos ({1,6},{2,5},{9,10}), não tem traçado convexo possível, pois o grafo G_2, construído conforme o Teorema 4.3 e o Corolário 4.6, não é planar.

Fig. 4.13

(a) grafo G



(b) G2



4.4- Traçado Agradável

O algoritmo de traçado convexo, detalhado na seção 4.2, não atinge seu objetivo se a representação planar 2-conexa G for rejeitada pelo teste de convexidade, isto é, se existirem pares de separação proibidos ou, mesmo que estes não existam, se os pares de separação críticos não puderem ser colocados num mesmo ciclo facial externo do grafo G_1, obtido de G pela operação definida pelo teorema 4.3. Além disso, o desenho obtido pelo algoritmo TRAÇO_CONV nem sempre é *bonito*, pois, freqüentemente, força as componentes 3-conexas que tem arestas em comum com o polígono externo a serem desenhadas como lâminas finas (ver Figs.4.2b e 4.5c).

O algoritmo aqui apresentado foi desenvolvido por Chiba-Onoguchi-Nishizeki [9] e se propõe a obter um traçado mais *agradável* para o grafo dado, além de não exigir que ele seja aceito pelo teste de convexidade. Ele se baseia no corolário 4.3 do teorema 4.2, para o qual **todo grafo planar 3-conexo tem traçado convexo para qualquer ciclo facial**. A partir deste resultado, o algoritmo procura isolar as componentes 3-conexas do grafo inicial e desenhá-las convexamente.

O traçado agradável obedece ao critério de linhas retas para as arestas e procura, tanto quanto possível, que as faces sejam desenhadas como polígonos convexos. Mas é o terceiro critério, que propõe que os ciclos faciais externos das componentes 3-conexas sejam polígonos convexos, que o algoritmo de [9] prioriza.

Sua restrição de entrada é, como na seção 4.2, que a representação planar G seja 2-conexa. No entanto, não é mais necessário que o polígono externo do grafo seja ampliável, ou que atenda às condições impostas na seção 4.3; mesmo um grafo com um ou mais pares de separação proibidos pode ser submetido ao procedimento em questão,

que obtém para ele um traçado agradável.

As Figs.4.2a e 4.2b mostram o mesmo grafo desenhado, respectivamente, de forma convexa e de forma agradável. Na Fig.4.2b as componentes 3-conexas estão certamente bem mais visíveis.

O grafo da Fig.4.13a foi rejeitado pelo teste de convexidade, mas ele pode obter um traçado agradável, se for submetido ao algoritmo em questão.

O algoritmo agradável se subdivide nos seguintes passos, a serem detalhados posteriormente um a um:

(i) decompor o grafo em uma estrutura básica que tenha traçado convexo e nas restantes componentes 3-conexas; desenhar a estrutura básica do grafo, usando a rotina TRAÇO_CONV, definida em 4.2;

(ii) determinar as regiões a serem utilizadas para encaixar as componentes 3-conexas retiradas no passo 1;

(iii) desenhar as componentes 3-conexas usando a rotina TRAÇO_CONV, se for necessário, e embuti-las nas regiões previamente determinadas.

4.4.1- Estrutura Básica do Desenho

O objetivo deste passo é determinar uma estrutura básica no grafo dado que admita um traçado convexo; algumas partes do grafo inicial são, para isso, eliminadas e tratadas nos passos seguintes.

Pelas definições 4.4 e 4.5, se {x,y} é um par de separação de um grafo G, é possível rachar G em dois subgrafos, de tal forma que se tenha como interseção de vértices somente o par {x,y} e como interseção de arestas o conjunto vazio; o mínimo de arestas de cada subgrafo é dois.

Seja G uma representação planar 2-conexa e seja S o ciclo externo. Dado um par de separação de G, rachar G em dois subgrafos; se exatamente um dos subgrafos rachados não contiver nenhuma aresta de S, então eliminá-lo do grafo G. Repetir a operação para todos os pares de separação de G.

O lema abaixo, proposto e demonstrado por [9], garante a possibilidade de um traçado convexo para o grafo G', resultante de G, após a retirada dos subgrafos rachados que não contêm arestas do ciclo externo.

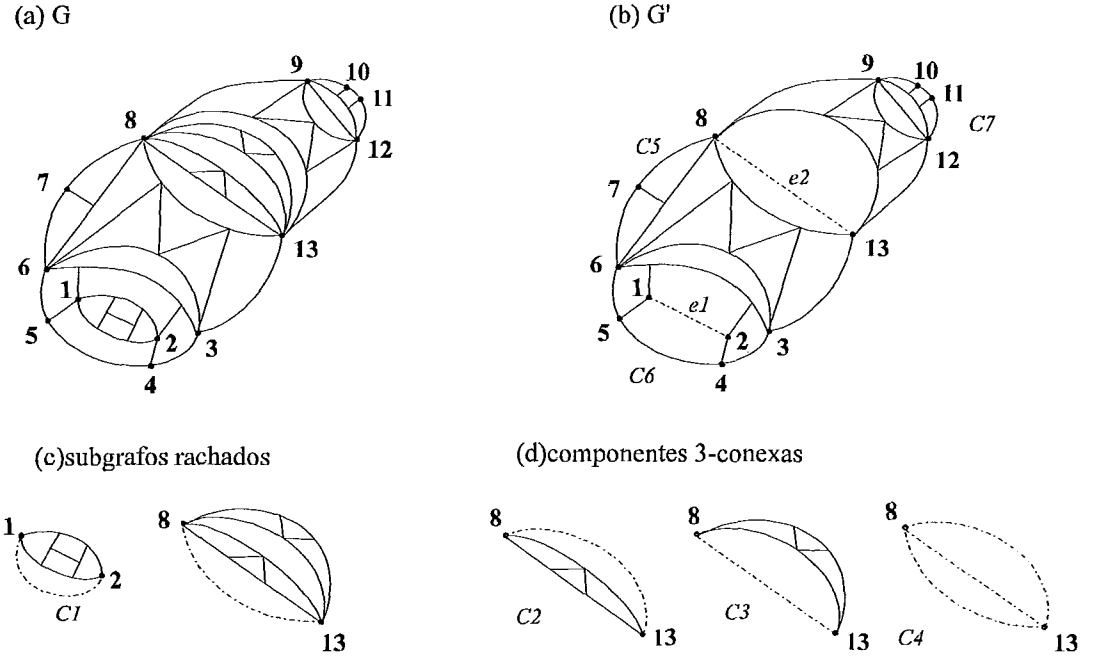
LEMA 4.5: Seja G' o grafo resultante da operação acima, o qual não tem

pares de separação cujos grafos rachados tenham, ambos, arestas de S. Então G' contém S e tem um traçado convexo.

A prova é conseguida diretamente do Lema 4.1.

Para ilustrar o Lema 4.5, seja G o grafo planar 2-conexo da Fig.4.14a. Os pares de separação {1,2} e {8,13} produzem subgrafos rachados que não contêm arestas de S, este com vértices 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 e 13. Eles são, então, eliminados de G e substituídos pelas arestas virtuais e_1 e e $_2$, resultando o grafo G', da Fig.4.14b, para o qual existe um traçado convexo.

Fig. 4.14



A estrutura básica do desenho, no entanto, nem sempre será o grafo G', pois ele pode ter componentes 3-conexas que serão desenhadas como lâminas finas pelo algoritmo de traçado convexo.

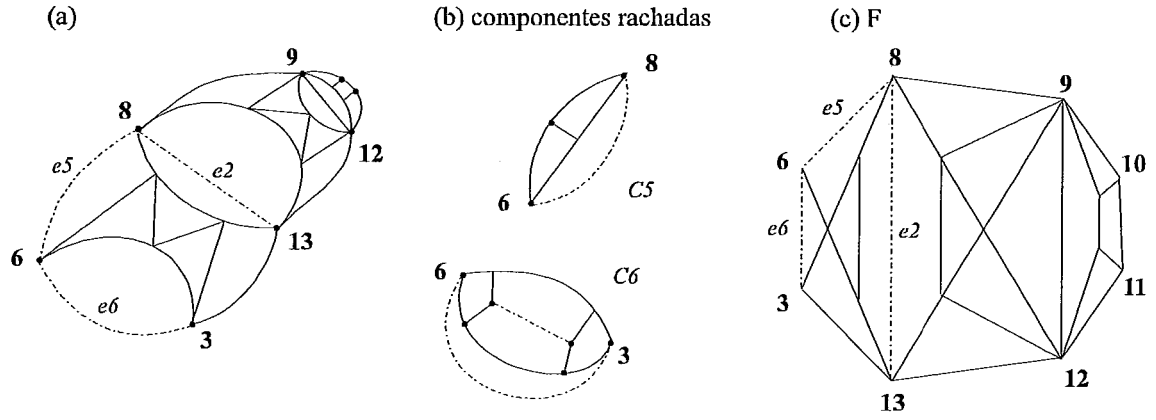
Num traçado convexo para o grafo G' da Fig.4.14b, certamente as componentes C_5, C_6 e C_7, relativas, respectivamente, aos pares de separação {6,8}, {3,6} e {9,12}, se enquadram no caso acima.

Para chegar à estrutura desejada, então, os demais pares de separação são analisados e as componentes 3-conexas, relativas a cada par {x,y}, eliminadas de G e substituídas por uma aresta virtual, a menos que a aresta (x,y) pertença ao grafo. O objetivo é traçar cada componente de forma convexa, mas, se a aresta existir, o desenho em lâmina fina não pode ser melhorado, o que explica porque a componente, neste caso,

não é retirada.

A estrutura básica do grafo da Fig.4.14a está ilustrada na Fig.4.15a; somente as componentes C_5 e C_6 (ver Fig.4.15b) foram eliminadas, sendo substituídas pelas arestas e_5 e e_6. A componente C_7 permanece, pois a aresta (9,12) está presente em G.

Fig. 4.15



Seja S_f o ciclo externo da estrutura básica de G, chamada F, e S^*_f um polígono convexo ampliável de S_f, escolhido apropriadamente. A rotina TRAÇO_CONV é executada, gerando um traçado convexo para F.

Na Fig.4.15c tem-se a estrutura F traçada convexamente, para o grafo da Fig.4.14a; o ciclo externo 3,6,8,9,10,11,12,13,3 contém três arestas virtuais neste ponto do procedimento.

4.4.2- Determinação das Regiões Utilizáveis

Cada aresta virtual precisa ser substituída pelo desenho da respectiva componente, eliminada no passo 1; a região utilizável, a ser definida, é a área do traçado convexo de F onde o desenho de cada um delas será encaixado.

Uma aresta virtual e_i pode:

(i) pertencer a F, como é o caso de e_2, e_5 e e_6 na Fig.4.15a; ou

(ii) não pertencer a F, como é o caso de e_1, que pertence à componente C_6, conforme a Fig.4.14b.

A determinação das regiões utilizáveis é realizada como a seguir:

caso a) e_i pertence a F:

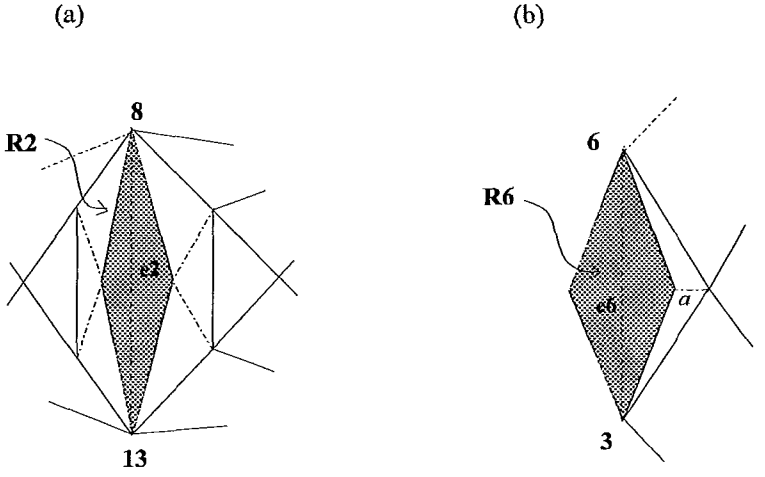
- colocar um ponto no interior de cada face que cerca e_i, por exemplo, os centróides dos polígonos convexos;

- unir estes pontos aos ápices dos polígonos convexos que os contêm;

-a.1) se e_i não pertence ao ciclo externo S_f de F, então a região utilizável é o quadrilátero definido pelos dois triângulos adjacentes a e_i (ver Fig.4.16a);

-a.2) se e_i pertence a S_f, então a região utilizável é a união do triângulo adjacente a e_i e do seu espelho, relativamente a e_i (ver Fig.4.16b).

Fig. 4.16



Nas Figs.4.16a e 4.16b, as áreas hachuradas são as regiões utilizáveis R_2 e R_6 determinadas para o desenho das componentes eliminadas no passo 1 e substituídas pelas arestas virtuais e_2 e e_6 (ver Fig.4.15b).

caso b) e_i não pertence a F; neste caso, assume-se que e_i está contida numa componente 3-conexa C_j, já desenhada:

- seja R_j a região utilizável definida para o desenho de C_j; R_j é um quadrilátero dividido em duas partes por e_j, conforme visto no caso anterior;

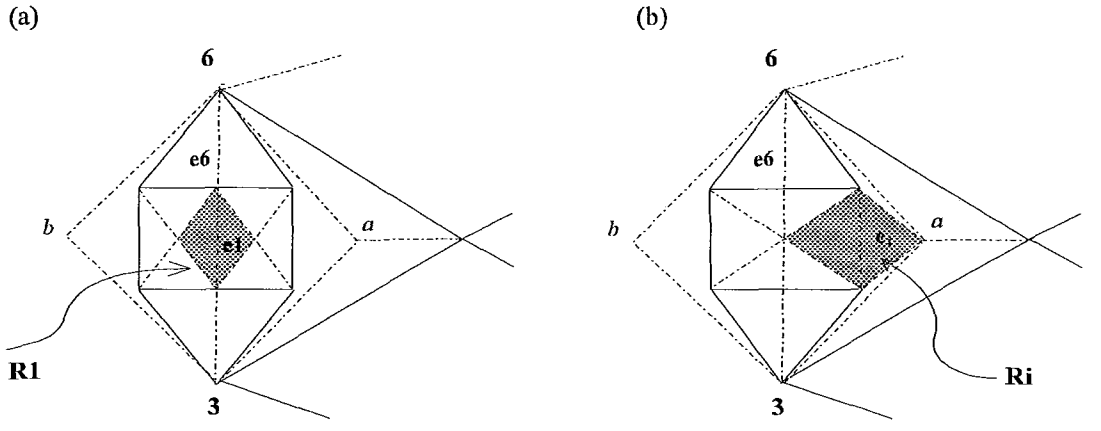
-b.1) se e_i não pertence ao ciclo externo da componente, então determinar R_i da forma descrita no caso a.1. A região utilizável R_1, para a aresta e_1, está ilustrada na Fig. 4.17a, onde R_6 é o quadrilátero de vértices 6, a, 3, b;

-b.2) se e_i pertence ao ciclo externo da componente, então R_i é determinada pela união de dois triângulos adjacentes a e_i; o primeiro é definido como no caso anterior e o outro tem ápices nos extremos de e_i e no ápice de R_j que está do mesmo lado de e_i (ver Fig.4.17b, onde e_i, supostamente, pertence a C_6).

Quando e_i não pertence a F, sua região utilizável só é determinada após o traçado da componente que a contém, o que é traduzido por uma recursividade dentro do

algoritmo de traçado agradável.

Fig. 4.17



4.4.3- Desenho das Componentes 3-Conexas

As partes eliminadas do grafo G, no primeiro passo do algoritmo, correspondentes a subgrafos rachados, são agora decompostas nas suas componentes 3-conexas, relativamente ao par de separação que as gerou; elas são processadas, neste passo, em conjunto com as demais componentes 3-conexas, que contêm arestas de S.

Para o grafo G da Fig.4.14a, as componentes 3-conexas são C_1, que é o próprio subgrafo rachado do par de separação {1,2}; C_2, C_3 e C_4, que são a decomposição do subgrafo rachado pelo par {8,13} e C_5 e C_6, retiradas na segunda etapa do primeiro passo. Elas são ilustradas nas Figs.4.14d e 4.15b.

As componentes 3-conexas de um grafo podem ser de três tipos, conforme visto na DEF.4.7: anéis, laços ou grafos 3-conexos.

Para cada par de separação {x,y}, podem ocorrer as seguintes situações:

caso a) se uma aresta virtual e_i substitui um anel C_i, o desenho de C_i-e_i consiste em localizar os seus vértices no segmento de reta que une os dois extremos de e_i;

caso b) se a aresta virtual e_i substitui um subgrafo 3-conexo C_i, então C_i-e_i tem traçado convexo. A rotina TRAÇO_CONV é executada para desenhar a componente, que, depois é embutida na região utilizável já definida. Este é o caso das componentes C_5 e C_6, assim como o de C_1, sendo que esta será tratada com recursividade (ver Figs. 4.14d e 4.15b);

caso c) se e_i substitui um laço C_i, entre outras componentes 3-conexas, realizar o procedimento a seguir.

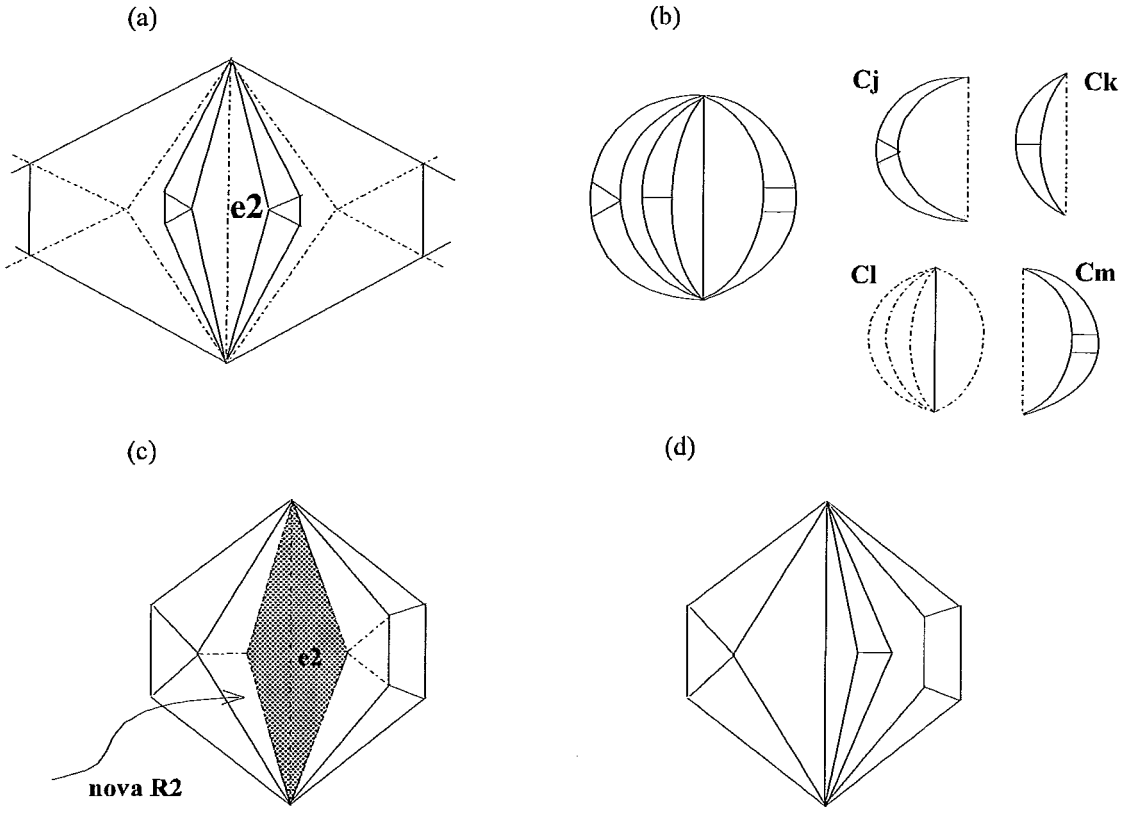
```
procedimento DESENHA_LAÇO:
começar
  . enquanto existirem 3 ou mais componentes 3-conexas a
    desenhar, fazer
    começar
      . seja Ck a união de duas dessas componentes;
      { desenhá-las, usando TRAÇO_CONV, na região
        utilizável, já determinada, Ri}
      . TRAÇO_CONV (Ck-ei,Sk,S*k);
      . embutir o traçado convexo de Ck-ei em Ri;
      { determinar uma nova região utilizável Ri no
        interior do par recém-desenhado, usando caso b.2 }
      . redefinir a região utilizável Ri;
    fim;
  { Existem, agora, zero, uma ou duas componentes 3-conexas
    ainda não desenhadas e somente o seguinte pode ocorrer }
  . se a componente restante é exatamente uma aresta real
    então traçá-la como um segmento de reta;
  . se a componente restante é um grafo 3-conexo
    então
    começar
      . seja Ck a componente restante;
      . TRAÇO_CONV (Ck-ei,Sk,S*k);
      { Ri é a região definida no fim da iteração acima }
      . embutir o traçado convexo de Ck-ei em Ri;
    fim;
  { se restam exatamente uma aresta real er=(x,y) e uma
    componente 3-conexa C com aresta virtual e=(x,y),
    então desenhar (C-e) ∪ er convexamente na última
    região Ri definida; isto pode ser feito pois (C-e) ∪ er
    obedece à Condição I do Lema 4.1 }
  . se restam uma aresta real e um grafo 3-conexo
    então
    começar
      . seja Ck a união dessas componentes;
      . TRAÇO_CONV (Ck-ei,Sk,S*k);
      . embutir o traçado convexo de Ck-ei em Ri;
    fim;
fim;
```

As componentes C_2, C_3 e C_4 podem ilustrar o caso **c**; C_4 é um laço e C_2 e C_3 são componentes 3-conexas para um mesmo par de separação {x,y}. Então, tomando C_2 e C_3, é possível desenhá-las com TRAÇO_CONV na região R_2 (ver Figs.4.16a e 4.18a). O procedimento se encerra, pois não há mais componentes com arestas reais a desenhar.

Considera-se, então, o conjunto hipotético de componentes 3-conexas da Fig.4.18b, para um mesmo par de separação {x,y}, para substituir a aresta e_2. Usando o procedimento descrito, toma-se C_j e C_m, por exemplo, e desenha-se de forma convexa; redefine-se a região utilizável R_2, mostrada na Fig.4.18c; restam duas componentes: uma

aresta real (C_l) e uma componente 3-conexa (C_k), que são desenhadas na região R_2 com TRAÇO_CONV (ver Fig.4.18d).

Fig. 4.18



4.4.4- Detalhes de Implementação

A seguir é apresentado o algoritmo DESENHA, cuja entrada é uma representação planar G 2-conexa; são utilizados os casos descritos na seção anterior.

```
procedimento DESENHA(G);
{ G é uma representação planar 2-conexa }

começar
  . determinar a estrutura básica F de G e as componentes
    3-conexas restantes;
  . seja Sf o ciclo externo de F;
  . escolher S*f, um polígono convexo ampliável de S;
  . TRAÇO_CONV (F,Sf,S*f);
  . para cada aresta virtual ei, pertencente a F, fazer
```

```
      . achar a região utilizável Ri;
  . enquanto existir uma aresta virtual ei, fazer
    começar       {Passo 3- casos a, b e c}
      . se Ci é um anel
        então traçar a aresta e posicionar os vértices;
      . se Ci é um grafo 3-conexo
        então
        começar
          . TRAÇO_CONV (Ci-ei,Si,S*i);
          . embutir o traçado convexo de Ci-ei em Ri;
        fim;
      . se Ci é um laço
        então DESENHA_LAÇO(Ci-ei);
      . para cada nova aresta virtual ej, fazer
         . achar a região utilizável Rj;
    fim
fim;
```

Observações:

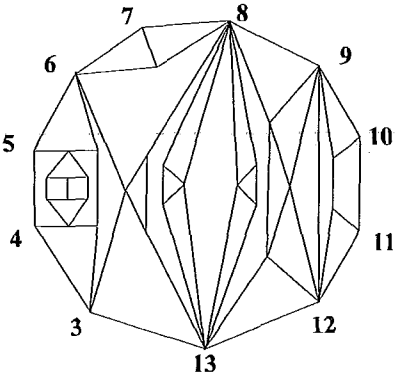
- S_i e S_k são os ciclos externos, respectivamente, de C_i-e_i e de C_k-e_i;

- S^*_i e S^*_k são os polígonos convexos ampliáveis apropriados, respectivamente, dos ciclos externos S_i e S_k.

Chiba e outros [9] provam que seu algoritmo produz um traçado de linhas retas para uma representação planar 2-conexa G em tempo linear.

TEOREMA 4.4: Dada uma representação planar 2-conexa G, o algoritmo DESENHA(G) produz um traçado de linhas retas de G em tempo linear.

Fig. 4.19



Como ilustração do resultado final do algoritmo, a Fig.4.19 apresenta o traçado agradável para o grafo G da Fig.4.14a. Observa-se que as componentes 3-conexas, sempre que possível, são traçadas de forma convexa, sendo o desenho visualmente agradável. Além disso, o grafo G não poderia ser desenhado pelo algoritmo

TRAÇO_CONV; o teste de convexidade o rejeitaria, uma vez que o par de separação {8,13} é proibido.

Pode-se considerar o algoritmo de traçado agradável como um aperfeiçoamento do traçado convexo. Além dele poder ser utilizado para traçar qualquer grafo planar 2-conexo, a visualização obtida para o grafo é, em geral, mais clara que a do anterior.

Capítulo 5

Outras Alternativas de Traçado

5.1- Introdução

Este capítulo apresenta alguns resultados mais recentes sobre traçado de grafos, que mostram as tendências de pesquisa na área e complementam os capítulos anteriores. De início é apresentado o traçado em grades, cuja importância decorre de suas aplicações na área de VLSI. Em seguida, resultados referentes à família dos grafos periplanares são abordados.

Os trabalhos de Storer [13], Tamassia-Tollis [10] e Tamassia-Tollis-Vitter [16] abordam o traçado de grafos planares, com grau máximo de vértices igual a 4, em *grades ortogonais*, estabelecendo como critérios as medidas de avaliação da qualidade do desenho produzido, ou seja, a área do retângulo que contém o desenho ou o número total de flexões existentes ao longo das arestas. Seu maior interesse é o traçado automático de *esquemas gráficos* para circuitos VLSI e redes de comunicação por luz ou microondas. Neste tipo de traçado todos os critérios definidos no capítulo anterior são abandonados.

O trabalho de Fraysseix-Pach-Pollack [7] critica os desenhos convexos (ver CAP.4), apontando como desvantagem a ilegibilidade ocasionada pela proximidade dos nós, obtida nos desenhos automáticos produzidos; ele propõe um algoritmo linear que embute um grafo planar num polígono externo triangular, contido numa grade (2n-4,0) x (n-2), onde **n** é o número de vértices do grafo.

Os trabalhos de Mitchell [17] e Manning-Atallah [18] concentram seus estudos na família de grafos *periplanares*, sendo que o segundo aborda o critério de simetria para o traçado dos mesmos, escolhendo-o como o melhor e mais geral entre todos a serem exibidos por um desenho. Esta é uma abordagem onde o critério está associado às propriedades do grafo e, portanto, não se aplica a todos os grafos planares; no entanto, para a família dos periplanares biconexos todos os critérios anteriormente

definidos são preservados.

5.2- Traçado de Grafos Planares em Grades

5.2.1- Grades Ortogonais Planares

Um desenho é uma **grade** se os vértices e as flexões das arestas, caso existam, têm coordenadas inteiras. Um **desenho ortogonal** de um grafo é tal que todas as suas arestas são cadeias poligonais, que consistem de segmentos horizontais e verticais. Um **esquema gráfico** ou um **desenho ortogonal planar** é um desenho ortogonal de um grafo em uma grade [16]. Os grafos que admitem este tipo de traçado são planares, com grau máximo 4 para os vértices e podem ter multi-arestas ou laços, aqui definidos como arestas do tipo e=(v,v).

No desenvolvimento dos seus trabalhos, [10], [13] e [16] assumem que é dada uma representação planar ou uma embutidura para o grafo, isto é, um conjunto de listas de adjacências de cada vértice do grafo, onde as arestas se apresentam na ordem dos ponteiros do relógio (ver CAP.3), a qual é preservada pelo esquema gráfico.

Os critérios ou medidas de avaliação de qualidade de um esquema gráfico estudados são:

(i) a área do retângulo que contém o desenho;

(ii) o comprimento total das arestas e o comprimento da aresta mais longa;

(iii) o número de flexões presentes ao longo das arestas.

Este último critério é chamado **custo de curvas** e foi inicialmente estudado por Storer [13], que ressalta algumas de suas aplicações práticas:

(i) mais simplicidade e clareza para o desenho;

(ii) a sua minimização pode ser uma aproximação para a minimização das outras duas medidas. Ele prova, no entanto, que isto não só não é verdadeiro sempre, como pode, no estudo dos piores casos, levar a resultados ruins para as outras duas medidas;

(iii) cada flexão de aresta tem um custo real, como, por exemplo, o de um dispositivo especial necessário para realizar curvas em comunicação por luz ou microondas e em alguns problemas de transporte.

Storer [13] apresenta três estratégias básicas para embutir um grafo em uma grade, que funcionam bem quanto ao número de flexões geradas; no entanto, os tempos dos algoritmos não são considerados. Definindo um **grafo não trivial** como aquele que contém no mínimo dois ciclos distintos e **grafo k-planar** como aquele que é planar e tem todos os vértices com grau menor ou igual a **k**, os resultados finais de [13] estabelecem que é possível obter um esquema gráfico para um grafo planar G=(V,E) com **n** vértices:

(i) com número total de flexões de arestas menor ou igual a **n**, para qualquer grafo 3-planar não trivial;

(ii) com número total de flexões menor ou igual a **2,4n + 4**, para qualquer grafo 4-planar não trivial; neste caso, se o grafo for biconexo, o custo de curvas é menor ou igual a **2n + 4**.

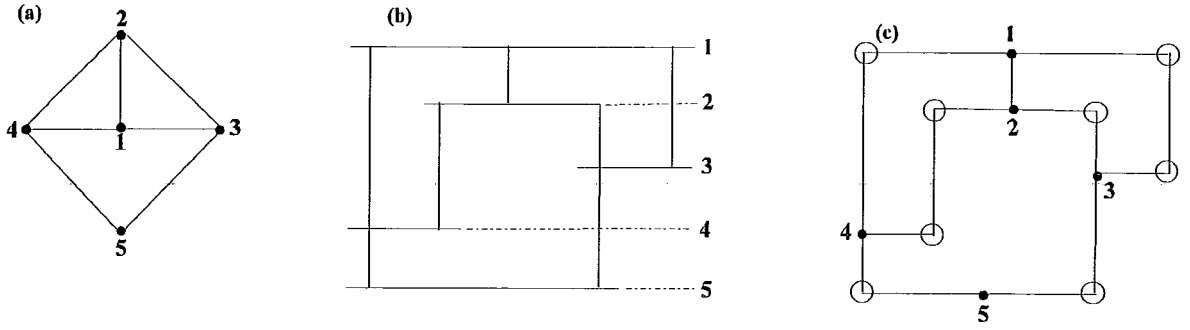
Os resultados de [10] são mais precisos; ele apresenta um algoritmo que desenha um grafo G=(V,E), planar com **n** vértices e grau máximo de vértices igual a 4, numa grade ortogonal planar, com as seguintes propriedades:

(i) tempo linear de execução;

(ii) desenhos com área ocupada de $O(n^2)$, que é a melhor possibilidade no pior caso, conforme demonstra [13];

(iii) custo de curvas comparável ao obtido por [13], isto é, no máximo **2n+4** para G biconexo e **2,5n+4**, em caso contrário.

Fig 5.1



Para obter um esquema gráfico de um grafo planar G=(V,E), [10] constrói, inicialmente, uma *representação de visibilidade fraca* para G, que é uma transformação dos vértices de G em segmentos horizontais, sem superposição, chamados **segmentos de vértices**, e das arestas de G em segmentos verticais, chamados **segmentos de arestas**, tal que, para cada aresta (u,v) pertencente a E, o segmento de aresta associado tem seus extremos nos segmentos de vértices correspondentes a **u** e a **v**, e não cruza nenhum outro

segmento de vértice. Nesta representação, os vértices de G são posicionados e pode-se obter um primeiro esquema gráfico para G.

As Figs. 5.1a, 5.1b e 5.1c apresentam, respectivamente, um grafo G, sua representação de visibilidade fraca e um esquema gráfico de G; neste é possível contar um total de oito flexões existentes ao longo das arestas.

A seguir, [10] define três transformações, chamadas por ele de *esticamento de curvas*, a serem aplicadas sobre um esquema gráfico de G, repetidamente; os ângulos de 90 graus formados pelas flexões das arestas são chamados **convexos**, os de 180 graus, **planos** e os de 270 graus, **côncavos**. As transformações de esticamento de curvas são:

(i) **TIPO1** - se uma aresta (u,v), no caminho de **u** para **v**, tem ângulos convexos tanto à esquerda como à direita, eles, aos pares, podem ser eliminados.

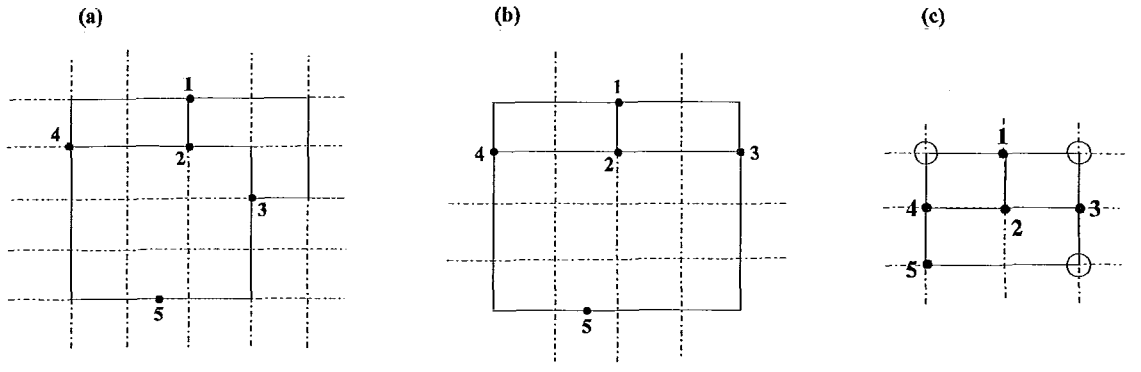
Na Fig.5.1c, a aresta (4,2) tem um ângulo convexo à esquerda e outro à direita; estes podem ser retirados, dando origem a um novo esquema, com um total de duas flexões a menos, mostrado na Fig.5.2a.

(ii) **TIPO2** - se todas as arestas (u,x), incidentes a um vértice **u**, têm ângulo convexo para o mesmo lado, isto é, todos para a esquerda ou todos para a direita, estes podem ser eliminados.

Na Fig.5.2a, as arestas (3,2) e (3,1) têm, ambas, um ângulo convexo para o lado esquerdo. Retirando-os, obtém-se o esquema da Fig.5.2b, que tem duas curvas a menos.

(iii) **TIPO3** - se, num vértice **v** de grau 2, as arestas incidentes formam um ângulo plano e, além disso, pelo menos uma delas tem um ângulo convexo, este pode ser eliminado.

Fig 5.2



Na Fig.5.2b, o vértice 5 ilustra este caso; eliminando o ângulo convexo da aresta (5,4), obtém-se o esquema da Fig.5.2c, com área também reduzida. Seu custo de curvas é 3, num total de cinco flexões a menos do que no esquema inicialmente

construído.

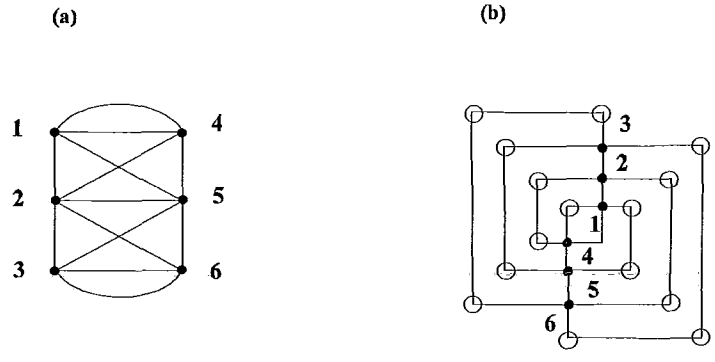
Aplicando as transformações acima definidas a todos os vértices de um esquema gráfico de G, [10] prova que seu algoritmo atinge as metas anteriormente citadas.

Os resultados obtidos por [10] são revistos por [16], que fornece limites inferiores mais precisos para o mínimo número de curvas, considerando os piores casos por famílias. Ele define G_n como o multigrafo 4-planar, biconexo e com **n** vértices, no qual todos os vértices têm grau 4, provando que, para G_n, o menor custo de curvas possível, em qualquer esquema gráfico, é **2n+4**.

Seja H_n' o grafo obtido de G_n pela eliminação das multi-arestas e a introdução de dois vértices extras para cada par delas. O grafo H_n' é biconexo e simples; seu número de vértices é n'=n+2 e seu esquema, obtido de G_n, tem duas curvas a menos.

A Fig.5.3 apresenta um grafo G_n para n=6 e seu esquema. As multi-arestas são (1,4) e (3,6); seu custo de curvas é 16. Ainda na Fig.5.3, H_8 pode ser obtido de G_6 introduzindo um vértice para a aresta (1,4) e outro para a aresta (3,6). Isto corresponde, no esquema da Fig.5.3b, à eliminação de duas curvas nas respectivas arestas, obtendo quatorze curvas.

Fig 5.3



Os resultados de [16], para um grafo G biconexo de **n** vértices, podem ser resumidos da seguinte forma:

(i) se G tem duas arestas múltiplas e não tem laços, seu número mínimo de curvas, no pior caso, que é G_n, é **2n+4**;

(ii) se G não tem arestas múltiplas e nem laços, seu número mínimo de curvas, no pior caso, que é H_n, é **2n-2**.

Quanto à área do retângulo que contém o desenho, além de ter demonstrado que qualquer grafo 4-planar com **n** vértices pode ser desenhado numa grade **n x n**, [13] prova, também, que há uma quantidade infinitamente grande de grafos 4-planares que requerem uma grade **(n-2) x (n-2)** e outra de grafos 3-planares que requerem uma grade **n/2 x n/2**.

O grafo da Fig.5.1a é 3-planar com cinco vértices; ele necessita de uma grade 2 x 2, conforme mostra a Fig.5.2c, após ter sido submetido às transformações de esticamento de curvas, definidas por [10].

Os trabalhos de [13], [10] e [16] se concentram em analisar o problema de minimização do custo das curvas, estabelecendo limites superiores e inferiores para os piores casos e para alguns casos especiais. Isto porque, [13] prova que, na maioria dos casos, o problema da minimização de qualquer uma das três medidas é computacionalmente intratável ou NP-difícil. E, ainda, que minimizar o custo de curvas pode, com respeito ao pior caso, levar a uma resultado ruim para a área do desenho ou para o comprimento total das arestas, sendo verdadeiro, também, o inverso. Para o custo de curvas, no entanto, como visto anteriormente, o algoritmo de [10] tem tempo de execução O(n) e constrói um esquema gráfico com um pequeno número de curvas.

5.2.2- Um Traçado de Linhas Retas

Um outro tipo de traçado é proposto por Fraysseix-Pach-Pollack [7], que utiliza como suporte para o posicionamento dos vértices uma grade. Esta tem dimensões (2n-4) x (n-2), onde **n** é o número de vértices do grafo dado; é apresentado um algoritmo com tempo O(n*log*n) e espaço O(n), que produz um desenho de linhas retas para o grafo, no qual o ciclo externo é representado por um triângulo com vértices de coordenadas (0,0), (2n-4,0) e (n-2,n-2). Também aqui, assume-se uma embutidura planar previamente calculada, onde a lista de adjacências de cada vértice se apresenta na ordem dos ponteiros do relógio.

O algoritmo de [7] se baseia no lema a seguir, conhecido como *representação canônica de uma embutidura planar*, formulado para **grafos planares maximais**, que são grafos planares nos quais a adição de uma aresta destrói a planaridade; grafos como estes são chamados *triangularizados*, pois todas as suas faces internas são triângulos. O lema seleciona um triângulo arbitrário e o elege como a face externa do grafo.

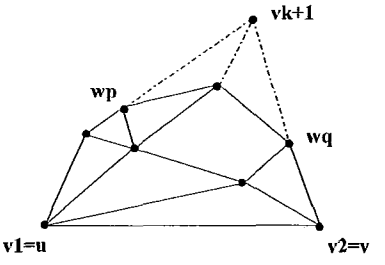
LEMA 5.1: Seja G um grafo planar maximal desenhado no plano com face externa **uvw**. Então existe uma rotulação para os vértices $\mathbf{v_1}$=u, $\mathbf{v_2}$=v, v_3, ..., $\mathbf{v_n}$=w

que satisfaz as requisições abaixo, para $4 \leq k \leq n$:

(i) o subgrafo G_k de G, induzido por $v_1, v_2, \ldots, v_k$ é 2-conexo e o limite da sua face externa é um ciclo que contém a aresta (u,v);

(ii) v_{k+1} está na face externa de G_k e seus vizinhos em G_k formam um subintervalo, de no mínimo dois elementos, no caminho C_k-uv.

Fig 5.4



A Fig.5.4 ilustra o posicionamento de $\mathbf{v_{k+1}}$ na face externa de G_k e o subintervalo no caminho C_k-uv formado por seus vizinhos em G_k.

O objetivo de [7] é construir um desenho **f** do grafo G, dada sua representação planar, tal que se **uvw** é sua face externa e $v_1=u$, $v_2=v$, v_3, ..., $v_n=w$ é sua rotulação canônica, então todo vértice de G está localizado em algum ponto da grade delimitada pelo triângulo determinado por

$f(v_1)=f(u)=(0,0)$, $f(v_2)=f(v)=(2n-4,0)$ e

$f(v_n)=f(w)=(n-2,n-2)$.

O teorema 5.1 e a sua prova possibilitam o desenvolvimento do algoritmo de desenho de [7].

TEOREMA 5.1: Qualquer representação planar de **n** vértices tem um desenho em uma grade 2n-4 por n-2, onde os vértices são pontos da grade e as arestas são segmentos de linha reta.

A prova é feita para uma representação planar máxima arbitrária de um grafo G, tomando uma face triangular qualquer e produzindo um desenho com este triângulo como face externa. A base da prova é a representação canônica definida no lema 5.1, que fornece uma ordenação apropriada para os vértices, tal que, supondo o subgrafo induzido pelos primeiros **k** vértices desenhado na grade, então, movendo alguns vértices de maneira controlada, pode-se adicionar o próximo vértice e continuar tendo um traçado de linhas retas na grade.

Assim, seja G com face externa uvw e $v_1=u$, $v_2=v$, v_3, ..., $v_n=w$ a rotulação canônica dos vértices. Supõe-se que, no passo **k**, o algoritmo já embutiu G_k na

grade de tal maneira que:

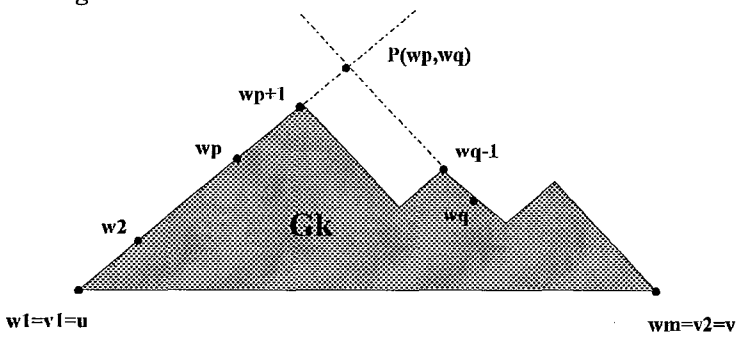
(i) v_1 está em (0,0) e v_2 em (2k-4,0);

(ii) se $v_1=w_1, w_2, \ldots, w_m=v_2$ são os vértices da face externo de G_k, nesta ordem, então $x(w_1) < x(w_2) < \ldots < x(w_m)$;

(iii) as arestas (w_i,w_{i+1}), para $1 \leq i \leq m$, têm todas inclinação +1 ou -1.

Sejam $w_p, w_{p+1}, \ldots, w_q$ os vizinhos de v_{k+1} em G_{k+1}, com $1 \leq p < q \leq m$. Seja $P(w_p,w_q)$ um ponto da grade definido pela interseção das retas que passam por w_p e w_q, respectivamente, com inclinação +1 e -1; $P(w_p,w_q)$ é um candidato para localizar v_{k+1} (ver Fig.5.5). Algumas vezes, no entanto, o desenho deve ser *deformado* para garantir que todas as arestas incidentes a v_{k+1} podem ser traçadas sem que nenhuma sobreponha outra.

Fig 5.5



O algoritmo, para uma embutidura planar de um grafo G não máximo, acrescenta inicialmente as arestas suficientes para triangularizá-lo.

O tempo de execução do algoritmo é calculado por [7] em O(n*log*n) e o espaço ocupado em O(n).

5.3- Grafos Periplanares Biconexos

Um grafo é **periplanar** se ele tem um desenho planar, no qual todos os vértices se localizam na face externa. Um grafo periplanar é **maximal** se a adição de uma aresta entre dois vértices quaisquer resulta em um grafo não periplanar.

Em seus trabalhos, Mitchell [17] e Manning-Atallah [18], citam dois resultados importantes que interligam a periplanaridade e a biconectividade:

(i) um grafo é periplanar se e somente se todas as suas componentes biconexas são periplanares;

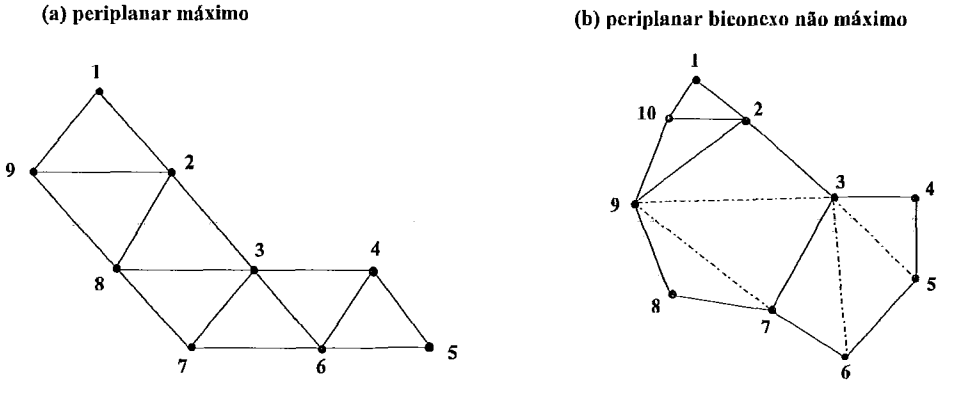
(ii) todo grafo periplanar biconexo, com um mínimo de três vértices, tem um único ciclo hamiltoniano.

Tendo em vista estes resultados, esta discussão se detém somente em grafos periplanares biconexos.

5.3.1- Reconhecimento de Grafos Periplanares Biconexos

Dois algoritmos lineares quanto ao número de vértices do grafo são desenvolvidos por [17] para reconhecer um grafo periplanar máximo e um periplanar biconexo. A base do primeiro algoritmo é que, tendo um grafo periplanar máximo um único ciclo hamiltoniano (resultado (ii) acima), suas arestas restantes *triangularizam* este ciclo, isto é, todas as faces internas de um grafo periplanar máximo são triângulos. E, mais, qualquer aresta nunca pertence a mais de dois triângulos. Para o segundo algoritmo a idéia é semelhante: uma vez que as arestas que não pertencem ao ciclo externo formam polígonos menores no seu interior, estes podem ser triangularizados, cada um independentemente do outro, pois, também neste caso, uma aresta não pode pertencer a mais de dois polígonos.

Fig 5.6



As Figs.5.6a e 5.6b exibem dois grafos periplanares; o da primeira figura é máximo e o da segunda não; neste, as arestas pontilhadas mostram uma possível triangularização dos seus polígonos internos.

O algoritmo de [17] que reconhece se um grafo é periplanar máximo utiliza o teorema a seguir e o seu corolário, demonstrados por ele no seu trabalho. Antes define-se como **2-vértice** um vértice de grau 2 cujos vizinhos são adjacentes.

TEOREMA 5.2: um grafo G biconexo com **n** vértices é periplanar máximo se e somente se ou G é um triângulo ou

(i) G tem exatamente 2n-3 arestas;

(ii) G tem, no mínimo, dois 2-vértices;

(iii) nenhuma aresta de G pertence a mais de dois triângulos; e

(iv) para qualquer 2-vértice **u**, o grafo G-u é periplanar máximo.

COROLÁRIO 5.1: Se G biconexo é periplanar máximo, então G não tem um vértice **w** de grau 2 cujos vizinhos não são adjacentes.

Dado um grafo G=(V,E) arbitrário, com **n** vértices, descrito pelas suas listas de adjacências, o algoritmo de [17] verifica inicialmente os itens (i) e (ii) do teorema 5.2; depois, iterativamente, com base no item (iv), elimina os vértices de grau 2 e as arestas a ele incidentes, pretendendo que o grafo resultante mantenha suas propriedades, isto é, que continuem existindo pelo menos dois vértices de grau 2. A iteração se encerra quando o grafo original ficar reduzido a dois vértices somente, os quais devem constituir uma aresta, comum a dois triângulos eliminados durante a iteração, conforme o item (iii) do teorema 5.2.

Os passos do algoritmo de reconhecimento de um grafo periplanar máximo são os seguintes.

```
procedimento PERIMAX;
começar;
  . se o quantidade de arestas é diferente de 2n-3
    então { número de arestas não confere }
          . parar;
  . seja L:= lista de vértices de grau 2;
  . seja Pares:= lista vazia;
  . se |L| < 2
   então { mínimo de dois vértices de grau 2 não verificado}
          . parar;
  . para i:= 1 até n-2, fazer
    começar
      . selecionar e eliminar o primeiro vértice de L;
      . seja v este vértice e u, w seus vizinhos;
      . adicionar a aresta (u,w) à lista Pares;
      . remover v de G;
      . acrescentar à lista L os vértices de G-v que
        ficaram com grau 2;
      . se |L| < 2
        então { G-v não tem dois vértices de grau 2 }
             . parar;
    fim;
   { a última aresta (u,v) deve ser comum a dois triângulos}
  . adicionar a aresta (u,v) à lista de arestas de G;
```

```
  . se algum elemento da lista Pares não consta da lista de
      arestas de G
    então { o grafo não é periplanar máximo }
         . parar;
    senão { o grafo é periplanar máximo }
         . parar;
fim.
```

Observar que, a cada vez que um vértice de grau 2 é eliminado, o algoritmo volta a testar quantos restaram, verificando simultaneamente os itens (ii) e (iv) do teorema 5.2; o número de arestas não precisa ser testado a cada iteração, pois à eliminação de um vértice corresponde a de duas arestas, o que mantém a igualdade.

É relevante notar que, ao final do algoritmo, a lista Pares contém todas as arestas interiores (ou cordas) do grafo, se este for periplanar máximo; neste caso, suprimindo da lista de arestas de G as da lista Pares, têm-se as arestas do único ciclo hamiltoniano do grafo.

O reconhecimento de um grafo periplanar biconexo utiliza o lema a seguir, que prova que é possível subdividir seus polígonos internos em triângulos.

LEMA 5.2: Um grafo é periplanar se e somente se ele pode ser transformado em um grafo periplanar máximo por triangularização.

O segundo algoritmo de [17] que se propõe a reconhecer um grafo periplanar biconexo é muito semelhante ao anterior. Neste caso, a quantidade **máxima** de arestas é 2n-3, mas o número mínimo de vértices de grau 2 continua sendo dois. A diferença entre eles é a adição de arestas que triangularizam os polígonos internos. Assim, no passo 4, a cada vez que uma aresta (u,v) é adicionada à lista Pares, se ela não consta da lista de arestas de G, ela é uma aresta *criada* pelo procedimento; as listas de adjacências de **u** e de **v** devem ser incrementadas dessa nova vizinhança. E, quando durante a eliminação de um vértice de grau 2, uma aresta *criada* for uma das adjacentes ao vértice, ela deve ser adicionada à lista de arestas de G. O resto do algoritmo é idêntico e tem-se ao final o reconhecimento de um grafo periplanar biconexo.

Também neste caso o ciclo hamiltoniano de G pode ser obtido da lista de arestas final, pela eliminação dos elementos da lista Pares; todas as arestas *criadas* são internas e, portanto, são abandonadas.

5.3.2- Exibição de Simetrias Planares

A simetria é uma propriedade dos grafos abstratos, independente de qualquer desenho particular do grafo. Ela pode ser considerada como um dos principais critérios de qualidade para o traçado de um grafo, por vezes realçando outras propriedades como a planaridade, a biconectividade ou o diâmetro do grafo.

Um grafo tem **simetria axial** se existe algum desenho para ele que exiba o eixo de simetria no sentido geométrico. Diz-se o mesmo para **simetria rotacional**. De acordo com [18], no entanto, os problemas de determinar se um grafo arbitrário tem alguma simetria axial ou rotacional são ambos NP-completos. Ele trata, então, em seu trabalho, as simetrias em grafos periplanares; mais ainda, somente as simetrias planares, que são aquelas que podem ser exibidas em um desenho planar, desejável para um grafo periplanar. Primeiro são considerados os grafos periplanares biconexos, onde as simetrias são detectadas e, depois, exibidas. Os grafos periplanares não biconexos são também discutidos, detalhadamente, por [18]. Todos os seus resultados podem ser expandidos para grafos com multi-arestas ou laços, estes definidos como arestas do tipo (v,v).

Seja G=(V,E) um grafo periplanar biconexo com **n** vértices e **m** arestas e H seu ciclo hamiltoniano, obtido pelo algoritmo de [17]. Apesar da simetria ter sido definida geometricamente, [18] utiliza um procedimento que consiste em *codificar* as adjacências de cada vértice de G por uma cadeia de inteiros $\mathbf{S_V}$, $\mathbf{v} \in V$, de tal forma que as simetrias planares de G correspondem a certas simetrias na cadeia $\mathbf{S_V}$.

Para identificar e enumerar as simetrias axial e rotacional planares de G, [18] se baseia na cadeia de inteiros **S**, união das cadeias S_V, que codifica G e observa o deslocamento de seus elementos. A partir de **S**, ele caracteriza as simetrias, desenvolvendo seus algoritmos.

Para exibir as simetrias axial e rotacional planares de um grafo periplanar biconexo G, com um mínimo de três vértices, [18] utiliza um polígono regular para desenhar o único ciclo hamiltoniano de G, traçando as demais arestas como cordas no interior do polígono. O lema a seguir garante que qualquer simetria de um desenho poligonal regular representa uma simetria planar de G.

LEMA 5.3: Seja G um grafo periplanar biconexo com $n \geq 3$ vértices. Então um desenho poligonal regular de G é um desenho periplanar de G.

O teorema abaixo reforça o lema 5.3 e associa a exibição de simetrias

planares ao desenho de polígonos regulares.

TEOREMA 5.3: Seja G um grafo periplanar biconexo com $n \geq 3$ vértices. Então um desenho poligonal regular de G exibe simultaneamente **todas** as simetrias axiais planares e **todas** as simetrias rotacionais planares de G.

Como consequência do teorema 5.3 tem-se que todas as simetrias de um grafo periplanar biconexo podem ser exibidas simultaneamente em um único desenho, desde que sejam mantidas as restrições de periplanaridade e biconectividade. No entanto, [18] observa que a enumeração dessas simetrias é importante porque:

- a aparência de um desenho é geralmente realçada se seu alinhamento for realizado por um eixo de simetria vertical;
- a indicação das simetrias em um desenho, por linhas tracejadas ou setas curvas pode ser desejável;
- simetrias geométricas de grafos podem ter interesse teórico independente, à parte da sua aplicação de guiar a construção de desenhos.

Os resultados sobre traçado em grades e traçado de grafos periplanares biconexos apresentados neste capítulo são recentes e pode-se observar a tendência de focalizar casos especiais no desenvolvimento dos algoritmos.

Capítulo 6

Conclusão

Foi apresentado neste trabalho um estudo do traçado de grafos planares, precedido pela discussão do reconhecimento da sua planaridade e da construção da embutidura, que é assumida como já calculada pelos algoritmos de traçado.

Os algoritmos de reconhecimento de planaridade escolhidos para apresentação foram o de Hopcroft e Tarjan, os primeiros a alcançar um tempo de execução linear, e o de Booth e Lueker, que criaram uma estrutura de dados denominada **árvore-PQ** e a utilizaram no desenvolvimento de um teste de planaridade bem simples.

A árvore-PQ possibilita, entre outros objetivos, acompanhar sistematicamente a construção da representação planar de um grafo e, portanto, foi muito importante no desenrolar do processo de traçado de grafos. Os algoritmos de construção de embutidura de grafos, apresentados neste trabalho, nela se fundamentaram para serem desenvolvidos.

O traçado de grafos foi subdividido em dois grandes tipos nesta exposição: o que objetiva a convexidade e as grades, de grande aplicação prática.

O traçado convexo não se mostrou abrangente, pois nem todo grafo planar se enquadra nas suas exigências. O traçado agradável, no entanto, o utilizou para desenhar convexamente as componentes 3-conexas do grafo dado; além de poder ser aplicado a qualquer grafo planar, obteve um desenho final visualmente mais bonito.

Foi estudado, juntamente com as grades, um traçado que escolhe um triângulo para desenhar o ciclo facial externo do grafo planar dado. Como resultado final, este algoritmo desenha o grafo em uma grade ortogonal, com vértices posicionados em coordenadas inteiras e distribuídos equilibradamente no plano.

Finalmente, uma abordagem dos grafos periplanares biconexos, uma família dos grafos planares, mostrou algoritmos simples para desenhá-los em polígonos regulares e exibir suas simetrias axiais e rotacionais com clareza.

A preocupação desta tese foi sedimentar todos os trabalhos que servem de sustentação para a pesquisa atual na linha de traçado de grafos planares e de suas famílias, procurando mostrar todos os aspectos básicos sobre os desenvolvimentos ocorridos nesta área. Pode-se observar que o tema escolhido é muito rico, apresentando, ainda hoje, inúmeras possibilidades de pesquisa.

Bibliografia

[1] HOPCROFT, J., TARJAN, R., "Efficient Planarity Testing", *Journal of ACM*,. 21, 4 (1974), 549-568.

[2] REINGOLD , NIEVERGELT , DEO, "Combinatorial Algorithms: Theory and Practice", Prentice Hall, 1977.

[3] EVEN, S., TARJAN, R., "Computing an st-Numbering", *Theoretical Computer Science*, 2 (1976), 339-344.

[4] BOOTH, K. S., LUEKER, G. S., "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity using PQ-Tree Algorithms", *Journal of Computer and System Sciences*, 13 (1976), 335-379.

[5] REINGOLD, E. M., TILFORD, J. S., "Tidier Drawing of Trees", *IEEE Transactions on Software Engineering*, SE-7, 2 (1981), 223-228.

[6] FRAYSSEIX, H. DE; ROSENSTIEHL, P., "A Depth First Search Characterization of Planarity", *Annals of Discrete Mathematics*, 13 (1982), 75-80.

[7] FRAYSSEIX, H. DE, PACH, J., POLLACK, R., "How to Draw a Planar Graph on a Grid", *Combinatorica*, 10 (1990), 41-51.

[8] CHIBA, NISHIZEKI, ABE, OZAWA, "A Linear Algorithm for Embedding Planar Graphs using PQ-Trees", *Journal of Computer and System Sciences*, 30, 1 (1985), 54-76.

[9] CHIBA, ONOGUCHI, NISHIZEKI, "Drawing Plane Graphs Nicely", *Acta Informatica*, 22 (1985), 187-201.

[10] TAMASSIA, R., TOLLIS, I. G., "Efficient Embedding of Planar Graphs in Linear Time", *Proc. IEEE - International Symp. of Circuits and Systems*, Philadelphia (1987), 495-498.

[11] ESPOSITO, C., "Graph Graphics: Theory and Practice", *Comput. Math. Applic.*, 15, 4 (1988), 247-253.

[12] JAYAKUMAR, R., THULASIRAMAN, K., SWAMY, M. N. S., "Planar Embedding: Linear-Time Algorithms for Vertex Placement and Edge Ordering", *IEEE Transactions on Circuits and Systems*, 35, 3 (1988), 334-344.

[13] STORER, J. A., "On Minimal-Node-Coast Planar Embeddings", *Networks*, 14 (1984), 181-212.

[14] SZWARCFITER, J. L., "Grafos e Algoritmos Computacionais", Editora Campus (1984).

[15] NISHIZEKI, T., CHIBA, N., "Planar Graphs: Theory and Algorithms", North Holland (1988).

[16] TAMASSIA, R., TOLLIS, I. G., VITTER, J. S., "Lower Bounds for Planar Orthogonal Drawing of Graphs", *Information Processing Letters*, 39 (1991), 35-40.

[17] MITCHELL, S., "Linear Algorithms to Recognize Outerplanar and Maximal Outerplanar Graphs", *Information Processing Letters*, 9, 5 (1979), 229-232.

[18] MANNING, J., ATALLAH, M. J., "Fast Detection and Display of Symmetry in Outerplanar Graphs", *Discrete Applied Mathematics*, 39, 1 (1992), 13-35.