UM ALGORITMO HEURÍSTICO DE COMPLEXIDADE LINEAR PARA PLANARIZAÇÃO DE GRAFOS POR REMOÇÃO DE VÉRTICES

Ademir Aparecido Constantino
Departamento de Informática,
Universidade Estadual de Maringá, CEP 87020-900, Maringá, PR
aaconstantino@uem.br

Candido F. X. de Mendonca
Escola de Artes, Ciência e Humanidades, USP-Leste, São Paulo, SP
cfxavier@usp.br

Rodrigo Lankaites Pinheiro
Departamento de Informática,
Universidade Estadual de Maringá, CEP 87020-900, Maringá, PR
rlpinheiro@ymail.com

RESUMO

Um grafo não planar só pode ser planarizado se o mesmo for modificado. Neste trabalho é apresentado um algoritmo que utiliza a remoção de vértice para obter um subgrafo planar. A remoção de vértice de um grafo G é o menor inteiro $k \geq 0$ tal que exista um subgrafo planar induzido de G obtido pela remoção de k vértices de G. Considerando que o problema de decisão correspondente é *NP*-completo e que não existe algoritmo de aproximação para planarização de grafos por remoção de vértices, este trabalho propõe um algoritmo heurístico de complexidade O(n+m) para planarização de grafos baseado na estrutura de dados de árvores-*PQ* e na operação de remoção de vértices. A complexidade do algoritmo é demonstrada e a sua eficiência é verificada por meio de alguns estudos de casos.

PALAVARAS CHAVE. planarização de grafos, árvores-*PQ*, remoção de vértices.

Área de Principal: Teoria e Algoritmos em Grafos

ABSTRACT

A non-planar graph can only be planarized unless it is modified. This work presents a new heuristic algorithm that uses vertexes deletion to modifier a non-planar graph in order to obtain a planar subgraph. The present algorithm aims to delete a minimum number of vertexes to get a planar subgraph. The vertex deletion number of a graph G is the smallest integer $k \geq 0$ such that there is an induced planar subgraph of G obtained by the removal of k vertexes of G. Considering that the corresponding decision problem is *NP*-complete and an approximation algorithm for graphs planarization by vertexes deletion does not exist, this work proposes a O(n+m) complexity heuristic algorithm for graph planarization based on the PQ-trees data structure and on the vertex deletion operation. The algorithm complexity is proven and its effectiveness is verified by means of some case studies.

KEYWORDS. graph planarization, PQ-Tree, vertexes deletion

Main Area: Teoria e Algoritmos em Grafos

1. Introdução

As aplicações práticas de Desenho de Grafo (*Graph Drawing*), tal como projeto VLSI, requerem técnicas de desenho para grafos não planares. Entretanto, o resultado de algoritmos de desenho é limitado a grafos planares, caso contrário o resultado pode ser inútil. Uma aproximação possível para trabalhar com a não planaridade em grafos é considerar suas invariantes topológicas, tal como o número de remoção de vértice, que pode ser usado como a medida de não planaridade.

Um *desenho simples* de um grafo G é um desenho de G no plano, sendo que nenhuma aresta se cruza, arestas adjacentes não se cruzam, o cruzamento de duas arestas ocorre somente uma vez, as arestas não cruzam sobre vértices e não mais de duas arestas cruzam em um ponto comum. Um grafo é *planar* quando há um desenho simples para este grafo no plano, sem cruzamento de arestas. Ao longo do texto que segue, todos os desenhos são considerados simples.

Se um desenho de um grafo G tiver o número mínimo de cruzamentos entre todos os desenhos de G, então, este desenho é denominado de desenho *ótimo*. Este número é denominado de *número de cruzamentos* (*crossing number*) do grafo G.

O *número de remoção de arestas* (*skewness*) $sk(G)$ é o menor inteiro $k > 0$ tal que a remoção de arestas de k de G torna o grafo planar.

O *número de divisão de vértices* (*splitting number*) $sp(G)$ de um grafo é o menor inteiro $k>0$ tal que um grafo planar pode ser obtido de G por operações de divisão de k vértices. Uma operação de divisão de vértice $v \in V(G)$, ou simplesmente divisão, particiona o conjunto de adjacentes de v em dois conjuntos não vazios P_1 e P_2 e adicionam a $G \backslash \{v\}$ os novos vértices v_1 e v_2 não adjacentes, tais que os vértices de P_1 são adjacentes a v_1 e os vértices de P_2 são adjacentes a v_2. Se um grafo H for obtido de G por uma sequência de k divisões, então, é dito que H é o grafo resultante deste conjunto de k divisões em G.

O número de remoção de vértices $\Phi(G)$ é o menor inteiro $k>0$ tal que a remoção de k vértices de G produz um grafo planar. O problema de decisão correspondente ao número de remoção de vértice, ao número de divisão de vértice, ao número de remoção de arestas ou ao número de cruzamento são todos *NP*-completos (Yannakakis, 1978; Geldmacher e Liu, 1979; Garey e Johnson, 1983; Faria *et al.*, 1998). Faria *et al* (2006) provou que não existe algoritmo de aproximação para planarização de grafos por remoção de vértices, logo, um algoritmo heurístico se torna uma importante alternativa para solucionar o problema. Também, foi mostrado que permanece *NP*-difícil, mesmo para grafos cúbicos (Faria *et al.*, 1998, 2001, 2004). Também foi mostrado (Figueiredo *et al.*, 1999) que o mesmo acontece para o número de divisão de vértices pelo resultado de Robertson e de Seymour (1995).

Na literatura são encontradas algumas invariantes de não planaridade para desenho de grafo, tais como o número de divisão de vértice e o número de remoção de arestas. O algoritmo PLANARIZE de Jayakumar, Thulasiraman e Swamy (1989) utiliza a operação de remoção de arestas para o desenho de grafo não planar. Eades e Mendonça (1993) consideraram o número de divisão de vértice adaptando o algoritmo PLANARIZE para o SPLIT-PLANARIZE. Isto foi feito substituindo a operação de remoção de aresta pela operação de divisão de vértice. Ambos os algoritmos têm complexidade de tempo O(n2) e complexidade de espaço O(n + m), onde n é o número de vértices e m é o número de arestas de G. Na seção seguinte será discutido como o algoritmo PLANARIZE (Jayakumar *et al.*, 1989) foi adaptado para um novo algoritmo proposto neste trabalho com complexidade de tempo e espaço O(n + m). Outro diferencial, consequente do uso da operação de remoção de vértice, é o fato de que o algoritmo PLANARIZE devolve um subgrafo planar enquanto que o algoritmo proposto devolve um subgrafo planar induzido.

A literatura relata muitos algoritmos que tentam remover um número mínimo das arestas para obter um subgrafo planar (Fisher e Wing, 1966; Pasedach, 1976; Marek-Sadowska, 1978; Chiba *et al.*, 1979; Ozawa e Takahashi, 1981). Uma das melhores tentativas é o algoritmo PLANARIZE de Jayakumar, Thulasiraman e Swamy (1989); resumidamente denotado por algoritmo JTS PLANARIZE. O algoritmo JTS PLANARIZE é baseado no algoritmo de teste de

planaridade de Lempel, Even e Cederbaum (1967) e Even (1979) (em resumo, o algoritmo de LEC) e sua implementação usando árvores-PQ (Booth e Lueker, 1976). O algoritmo proposto neste trabalho, denominado de VD-PLANARIZE, usa ideias similares do algoritmo PLANARIZE acima, porém, ele utiliza a operação de remoção de vértice ao invés da operação de remoção de aresta.

A seção 2 descreve o algoritmo de LEC e a estrutura de dados chamada árvores-PQ e, ainda, alguns conceitos associados a esta estrutura de dados. Na seção 3 será apresentado o algoritmo VD-PLANARIZE. Posteriormente, será feita a análise da complexidade do algoritmo e o seu desempenho (seção 4 e 5).

2. O Algoritmo de LEC e Árvores-PQ

Esta seção apresenta os princípios do algoritmo JTS PLANARIZE baseado no algoritmo de teste de planaridade de LEC executado com auxílio de árvores-PQ. As definições desta estrutura de dados e as operações sobre ela são descritas nesta seção. Entretanto, acerca dos detalhes de implementação das operações com árvores-PQ é recomendado ao leitor que consulte o trabalho de Booth e Lueker (1976).

O algoritmo de LEC trata somente de grafos biconexos. Considerando que é fácil dividir um grafo em uma árvore de componentes biconexos (blocos). Gibbons (1984) apresenta um algoritmo de complexidade linear que reconhece uma árvore de componentes biconexos de um grafo. Consideram-se, neste trabalho, somente grafos biconexos.

Considere um grafo biconexo $G=(V, E)$ com $n=|V|$ vértices e $m=|E|$ arestas. Uma *st*-numeração é uma rotulação dos vértices de G por números inteiros 1, 2,…, n sendo que "1" é adjacente a "n" e o vértice j é adjacente a um par de vértices i e k se $i < j < k$. O vértice "1" (resp. "n") é chamado fonte (resp. sumidouro) e denotado por s (resp. t). Cada grafo biconexo tem uma *st*-numeração (Lempel *et al.*, 1967) e tal rotulação pode ser encontrada em tempo linear (Even e Tarjan, 1976). O grafo G rotulado com *st*-numeração é chamado *st*-grafo.

Seja G_k, $1 \leq k \leq n$, um subgrafo de um *st*-grafo G induzido pelo conjunto de vértices $V_k = \{1, 2,\ldots, k\}$. Seja B_k um grafo associado ao subgrafo G_k e de todas as arestas de G incidentes com os vértices de V_k e de V-V_k em G. Estas arestas são chamadas *arestas virtuais* e os vértices de V-V_k são chamados de vértices virtuais. Os vértices virtuais são rotulados como seus vértices equivalentes em G; no entanto mantidos separados (uma folha para cada adjacente que não tenha sido imerso – *embeded*). Consequentemente, em B_k pode haver diversos vértices virtuais com o mesmo rótulo, cada um com exatamente uma aresta virtual. Um desenho de B_k é chamado de *arbusto* (*bush form*) de B_k se os vértices com rótulos menores aparecerem em um nível mais elevado e todos os vértices virtuais aparecerem no mesmo nível. Figura 1 (a) mostra o $K_{3,6}$ com uma *st*-numeração; (b) mostra, também, um arbusto B_5.

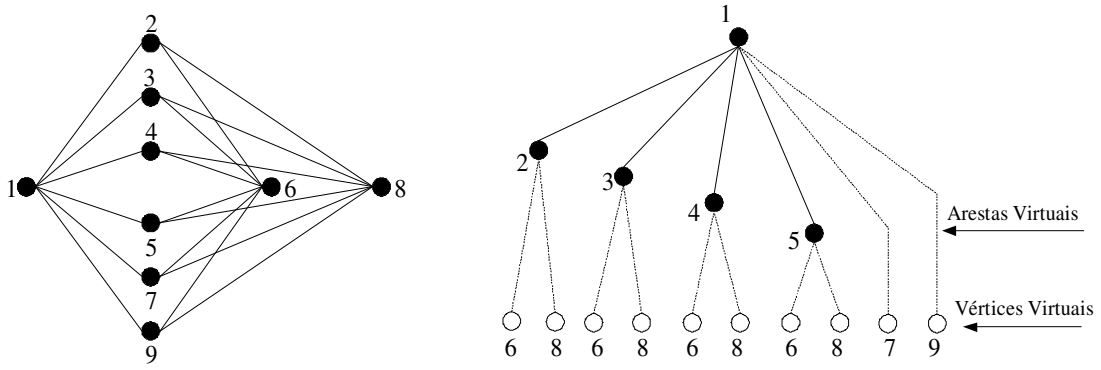

Figura 1: Um *st*-grafo e um arbusto B_5

Pode-se mostrar (Lempel *et al.,* 1967; Even, 1979) que um *st*-grafo é planar se e somente se para cada B_k, $2 \leq k \leq n - 2$, existir um grafo planar B'_k isomorfo a B_k tal que todos os vértices virtuais em B'_k rotulados k+1 aparecerem consecutivamente.

Uma árvore-*PQ* (Booth e Lueker, 1976) *T* é uma estrutura de dados que representa um conjunto das permutações em um conjunto *S*. Os nós de *T* são:

- folhas, representando os elementos de *S*;
- nós-P, convencionalmente representado como um círculo;
- nós-Q, convencionalmente representado como um retângulo.

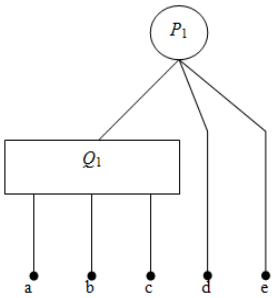

Figura 2: Uma árvore-*PQ*

Para este tipo de árvore é muito importante a ordem de como os filhos de um nó aparecem. Define-se a *fronteira* de *T* como sendo a permutação representada pela ordem da esquerda para direita de todas as folhas de *T*. Por exemplo, a fronteira da árvore-*PQ* na figura 2 é [abcde].

O conjunto das permutações representadas por *T* é gerado rearranjando os filhos de cada nó de P e de Q de acordo com duas regras:

(i) Os filhos de um nó-P podem ser permutados livremente;

(ii) A ordem dos filhos de um nó-Q só pode ser invertida.

O conjunto das permutações de *S* represento por *T* é o conjunto das fronteiras das árvores *T* obtidas de *T*, rearranjando os filhos de acordo com estas regras. Por exemplo, o conjunto das permutações representadas pela árvore-*PQ* na figura 2 é: [abcde], [abced], [cbade], [cbaed], [dabce], [dcbae], [eabcd], [ecbad], [deabc], [decba], [edabc], e [edcba]. Estas permutações correspondem às doze árvores da figura 3.

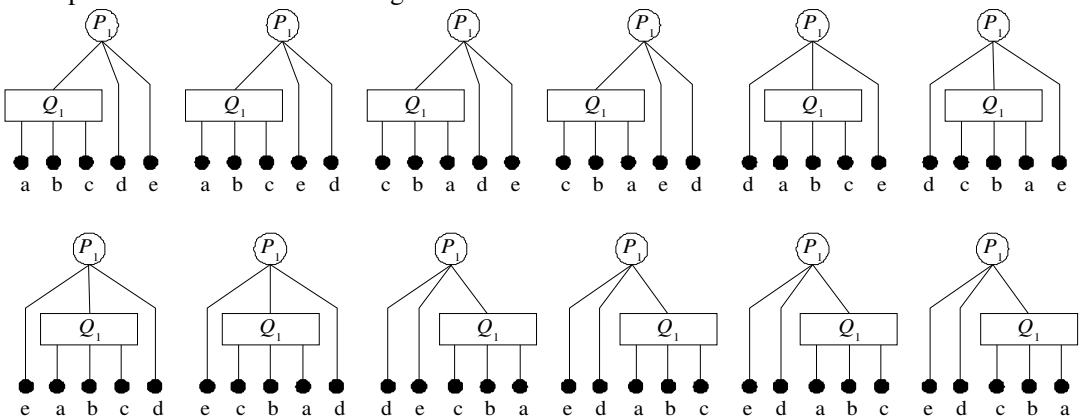

Figura 3: Doze permutações permitidas por uma árvore-*PQ*

Estas árvores-*PQ* provaram ser úteis em muitos problemas os quais envolvem a redução sucessiva dos conjuntos das permutações para encontrar uma permutação especial. Por exemplo, foram usados para reconhecer grafos planares, grafos de intervalo, matrizes com a propriedade de uns consecutivos (Booth e Lueker, 1976), os grafos planar hierárquicos (Di Battista e Nardelli, 1988), bem como os desenhos de dominância (Eades *et al.,* 1993).

Neste trabalho, uma árvore-*PQ* T_k é usada para representar o arbusto B_k no algoritmo. Os nós de T_k correspondem os seguintes:

(i) folhas: os vértices virtuais de B_k ,

(ii) nós-Q: os componentes biconexos maximal em B_k ,

(iii) nós-P: os vértices de articulação em B_k .

As folhas são chamadas *pertinentes* se corresponderem ao próximo vértice selecionado (rótulo k+1) com possibilidade de ser imerso; enquanto que as demais são chamadas folhas *não pertinentes*. Da mesma forma, um nó não folha X é dito pertinente se alguma folha descendente de X na árvore-*PQ* for pertinente. Se todas as folhas do descendente de um nó X na árvore-*PQ* forem pertinentes, então X é chamado de nó *cheio*. Se nenhuma folha descendente do nó X for pertinente, então X é dito *vazio*. A *fronteira* de X é definida pelo seu conjunto das folhas descendentes, lidas da esquerda para a direita. Um nó X é uma *raiz pertinente* se ele for o nó de mais baixo nível cuja fronteira contém todas as folhas pertinentes. A árvore enraizada em X é chamada de subárvore *pertinente*. Uma vez que uma raiz pertinente é identificada, pode-se usar uma série de testes de padrões e realocações descritos em (Booth e Lueker, 1976) para construir uma árvore nova em que todas as folhas pertinentes aparecem consecutivamente, se tal árvore existir. Caso isso ocorra, todas as folhas pertinentes na nova árvore aparecerão como filhos de um único nó. Por exemplo, vamos supor que a árvore-*PQ* da figura 4 representa um arbusto B_7; o nó P_1 é um nó pertinente; o nó Q_1 é um nó vazio; o nó P_2 é a raiz pertinente; o nó Q_2 é um nó cheio; as folhas pertinentes são rotuladas como 8; e as folhas não pertinentes são rotuladas como 9, 10, 11, 12.

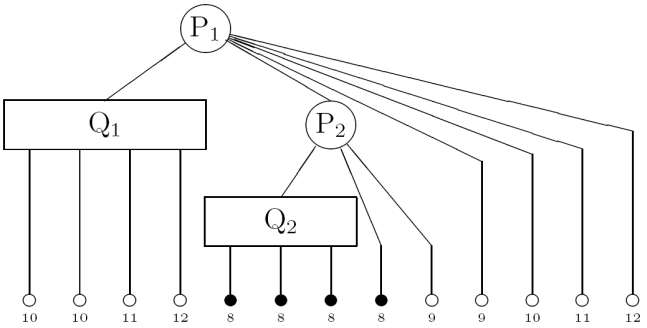

Figura 4: Uma árvore-*PQ* de um arbusto B_7

A *redução* é uma operação importante em uma árvore-*PQ*. Em um nível abstrato, a redução toma um conjunto das permutações Π de S e um subconjunto $S' \subseteq S$ e retorna um subconjunto Π' de Π tal que os elementos de S' aparecem consecutivamente em todas as permutações em Π'. Os elementos dos S' são ditos elementos *pertinentes* de S.

Both e Lueker (1976) criaram um algoritmo que reduz uma árvore T_k em uma árvore T_k^* de tal forma que todas as folhas pertinentes apareçam consecutivas na fronteira de T_k^* (quando for possível). A operação de redução pode ser executada eficientemente com uma implementação sofisticada de árvores-*PQ*. Este trabalho, no entanto, não discute essas operações que são descritas em detalhes em (Booth e Lueker, 1976). O processo de redução mencionado pode ser executado em duas etapas:

Bubble up: nesta etapa é identificada a subárvore pertinente (sobre a qual a redução deve ser processada).

Redução: nesta etapa é executada uma série de operações de teste de padrão e substituição usando as operações de inversão dos nós-*Q* e permutação dos nós-*P*.

Nota-se facilmente que nem sempre uma árvore-*PQ* T_k pode ser reduzida em uma árvore T_k^*, assim, Ozawa e Takahashi (1981) definiram alguns critérios para testar uma árvore antes de aplicar a redução. Sejam *G* um *st*-grafo biconexo e $T_1, T_2, \ldots, T_{n-1}$ as árvores-*PQ* correspondentes aos arbustos $B_1, B_2, \ldots, B_{n-1}$ de *G*. Um nó *X* de uma árvore-*PQ* é classificado de acordo com sua fronteira como segue:

Tipo A: se a subárvore enraizada em *X* puder ser rearranjada, de tal maneira que todas as folhas pertinentes descendentes de X apareçam consecutivamente no meio da fronteira, com pelo menos uma folha não pertinente em cada extremo da fronteira. Por exemplo, o nó P_1 na figura 4 é o tipo *A*.

Tipo B: se a fronteira da subárvore enraizada em X consistir somente em nós pertinentes, então *X* é um nó cheio. Por exemplo, o nó Q_2 na figura 4 é o tipo *B*.

Tipo H: se a subárvore enraizada em *X* puder ser rearranjada, de tal maneira que todas as folhas pertinentes do descendente de *X* apareçam consecutivamente em uma das extremidades da fronteira. Por exemplo, o nó P_2 na figura 4 é o tipo *H*.

Tipo W: se a fronteira da subárvore enraizada em *X* consistir somente de folhas não pertinentes, ou seja, *X* for um nó vazio. Por exemplo, o nó Q_1 na figura 4 é o tipo *W*.

Sabe-se que nem sempre uma árvore-*PQ* é um dos tipos *A*, *B*, *H* ou *W*. Entretanto, será visto mais adiante a necessidade de transformar (essencialmente pela remoção de vértice) toda árvore em uma árvore do tipo *W*.

Um grafo *G* de *n* vértices é planar se e somente se as raízes pertinentes em todas as árvores-*PQ*, $T_2, T_3 \ldots, T_{n-2}$ de *G* forem do tipo *B*, *H* ou *A*. Uma árvore-*PQ* é *redutível* se sua raiz pertinente for o tipo *B*, *H* ou *A*, caso contrário é irredutível (Ozawa e Takahashi, 1981).

As duas árvores T_1 e T_{n-1} são *redutíveis*. A primeira porque tem somente uma folha pertinente correspondente à aresta (1, 2) e a última árvore porque tem somente um tipo de folha que é o nó que corresponde ao vértice virtual *n*.

3. Planarização por Remoção de Vértice

Nesta seção será introduzida a proposta do algoritmo de planarização de grafos por remoção de vértices, denominado de VD-PLANARIZE.

Em linhas gerais, o algoritmo VD-PLANARIZE inicia com um vértice e prossegue com a imersão de um vértice de cada vez construindo um subgrafo planar induzido *G*' de *G*. Os vértices são selecionados seguindo a ordem da rotulação introduzida pelo algoritmo de *st-numeração* (Lempel *et al.,* 1967). Seja *v* o próximo vértice selecionado cuja imersão será analisada no subgrafo planar *G*'. Seja E_v o subconjunto das arestas que incidem em *v* e em vértices de G' e seja $\hat{E}_v$ o subconjunto das outras arestas incidentes com *v*. Em cada iteração, se um vértice *v* não puder ser imerso em *G*', então ele é removido. O conjunto restante de arestas $vu \in \hat{E}_v$ é adicionado (como arestas *dummy*) ao primeiro vértice imerso (vértice 1). Isto será feito para cada vértice *u* que não tenha outro adjacente com rótulo menor que o rótulo de *v*, com o objetivo de manter a propriedade da *st*-numeração.

O algoritmo de LEC inicia com árvore T_1 e constrói a sequência de árvores-PQ $T_1, T_2, \ldots$ Se um grafo for planar, o algoritmo de LEC termina depois de construir a árvore T_{n-1}, caso contrário ele termina quando detecta a impossibilidade de reduzir uma árvore T_k em T_k^*.

Considere T_k uma árvore-*PQ* irredutível de um grafo não planar, ou seja, é impossível reduzir uma árvore T_k em T_k^*. Neste caso, o algoritmo proposto adiciona uma nova operação denominada *Update*(k). Esta operação remove todas as folhas pertinentes transformando a árvore T_k no tipo *W*. Além disso, se algum vértice u com rótulo maior que k+1, adjacente ao vértice equivalente das folhas pertinentes removidas, não tiver outro adjacente com rótulo menor, uma nova aresta (*dummy*) é adicionada ao grafo de tal modo que u seja adjacente a s. Isto é necessário para manter a propriedade da *st*-numeração. Cada iteração de imersão do algoritmo pode aumentar o número dos adjacentes de s. Entretanto, este número não excede o número dos vértices de *G*. A pergunta principal é como inspecionar os adjacentes do vértice a ser removido para assegurar a propriedade da *st*-numeração sem aumentar a complexidade de tempo. Isto pode ser feito adicionando um campo *small*(u) para cada vértice u. Este campo informa quantos adjacentes de u têm rótulos menores do que o rótulo de u estabelecido na etapa da *st*-numeração. Assim, quando os nós pertinentes – correspondentes ao vértice v_{k+1} – são removidos para fazer todas as subárvores T_k do tipo *W*, o campo *small*(u) é reduzido de uma unidade para cada adjacente u de v_{k+1} onde u tem a rótulo maior que k+1. Quando o valor do campo *small*(u) alcança o valor zero, uma aresta *dummy* é adicionada de s para u. Depois da execução da última iteração são removidas todas as arestas *dummy* de s para u onde o campo *small*(u) é zero.

O pseudocódigo do algoritmo proposto é apresentado como segue:

VD-PLANARIZE
Entrada: grafo G
Saída: subgrafo induzido planar de G
Pré-processamento: obter uma *st*-numeração válida para *G,* obter a lista *small* para todos os vértices de *G.*

Início:
  construir a árvore inicial T_1;
  para $k := 2$ até $n-2$ faça: [seguindo a st − numeração]
    se T_{k-1} for redutível {teste de Ozawa e Takahashi} então:
      aplica a redução {algoritmo de Booth e Lueker};
    senão
      $Update(v_k)$;

  obter T_k substituindo todos os nós pertinentes de T_{k-1}^* por um novo nó-P P_k com todas as arestas saindo do vértice v_k com rótulo maior do que k aparecendo como filho de P_k;
  retorna G;
Fim;

4. A Complexidade do Algoritmo VD-PLANARIZE

A redução Booth e Lueker de todas as árvores-PQ redutíveis T_k pode ser executada em tempo total de O(n+m) (Jayakumar *et al.,* 1989). Se uma árvore-PQ T_k não for redutível, executa-se a operação *Update*(k) que fará a remoção dos vértices pertinentes. Vamos supor o pior caso com o máximo de vértices removidos (note que isto é verdadeiro para o grafo K_n). Neste caso, para cada vértice v removido, o algoritmo inspeciona os rótulos de cada adjacente u de v. Se o rótulo de u for maior do que o rótulo de v, o valor de *small*(u) é reduzido uma unidade. Portanto, a operação *Update*(k) inspecionará cada vértice e seus adjacentes. Logo, no pior caso, o tempo total dessa operação é O(n + m) (similar à busca em largura). A adição de arestas *dummy* ao

vértice s será feita, no pior caso, n vezes. Consequentemente, a complexidade de tempo do VD-PLANARIZE é $O(n + m)$.

5. O Desempenho do VD-PLANARIZE

Por se tratar de um algoritmo heurístico, o questionamento que pode ser feito é quanto à qualidade das soluções obtidas, ou seja, a quantidade de vértices removidos. A solução ótima implica em remover o número mínimo de vértices. A eficiência do algoritmo, com exceção de poucos casos tal como para grafo completo K_n, é fortemente dependente da *st*-numeração. Esta seção mostra a execução do algoritmo sobre um grafo K_{nm}, $3 < n \leq m$, com duas *st*-numerações diferentes sobre um grafo $C_4 \times C_4$.

A figura 5 ilustra a *st*-numeração e as iterações do VD-PLANARIZE para um grafo K_{nm} (a árvore-*PQ* foi omitida para as iterações 1, 2, 3 e 4). Neste primeiro caso, a partição inferior com n vértices recebe os rótulos {1, 3, 5, 6,..., n + 2} e a partição superior recebe os rótulos {2, 4, $n + 3$, $n + 4$, ...,$n+m$}. Todos os vértices rotulados 5, 6,..., $n + 2$ serão removidos e todos os outros vértices serão imersos pelo algoritmo. Assim, o algoritmo executará o melhor caso removendo n-2 vértices.

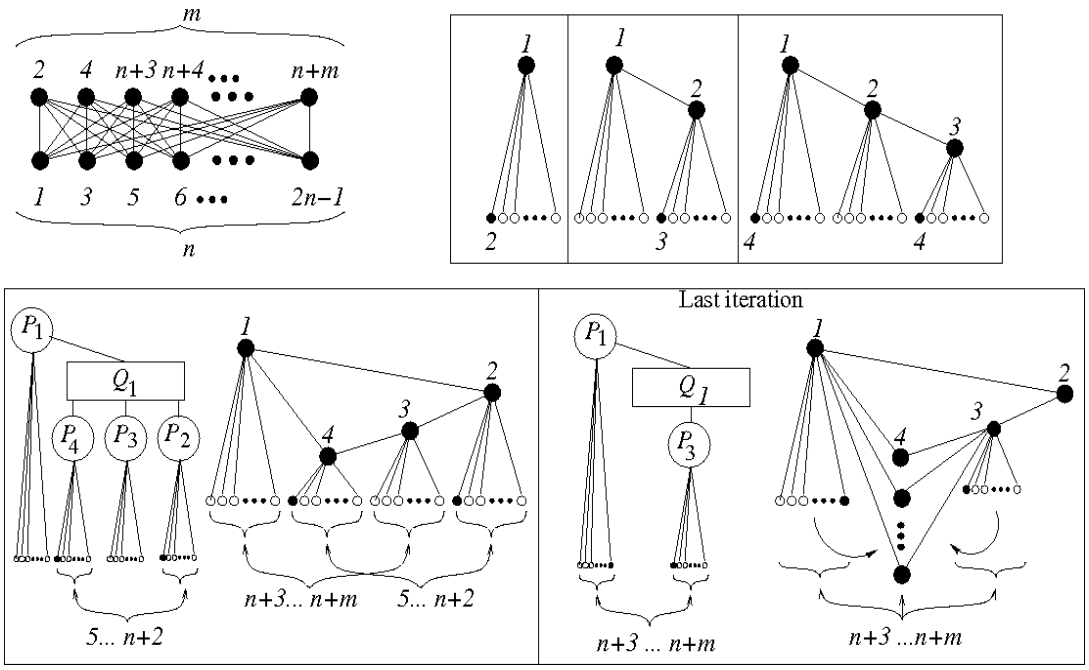

Figura 5: O caso ótimo para um K_{nm}

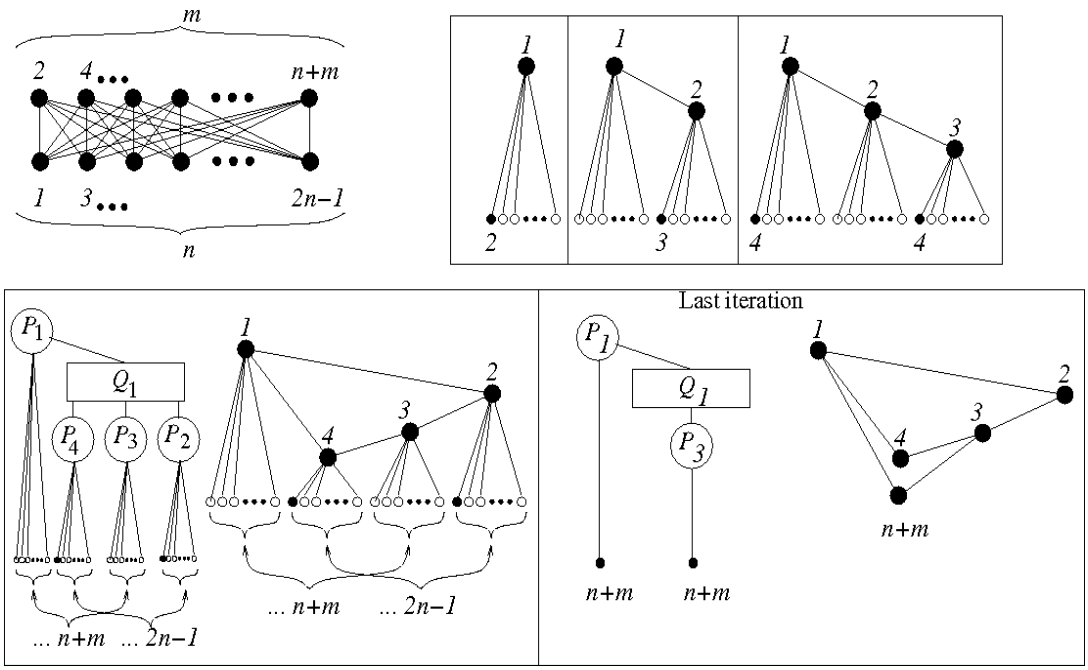

Figura 6: O pior caso para um K_{nm}

Já neste outro caso da *st*-numeração, o algoritmo remove o maior número de vértices. A figura 6 ilustra a *st*-numeração e as iterações do algoritmo VD-PLANARIZE para um grafo K_{nm} (as árvores-*PQ* para passos 1, 2, 3 e 4 são omitidas). Neste caso, a partição inferior com n tem os seguintes rótulos {1, 3,..., $2n$-1} e a partição superior com $n+m$ tem os seguintes rótulos {2, 4, ...,$n+m$}. Os vértices rotulados 5, 6,..., $2n$ - 1 são removidos e somente os vértices rotulados 1, 2, 3, 4 e $n+m$ são imersos pelo algoritmo. Assim, neste caso a execução do algoritmo não é boa removendo $2n$ - 5 vértices.

Na sequência, a Figura 7 ilustra a execução do algoritmo para o grafo $C_4 \times C_4$. A *st*-numeração foi escolhida de tal forma que o algoritmo encontre a solução ótima (removendo o mínimo de vértices).

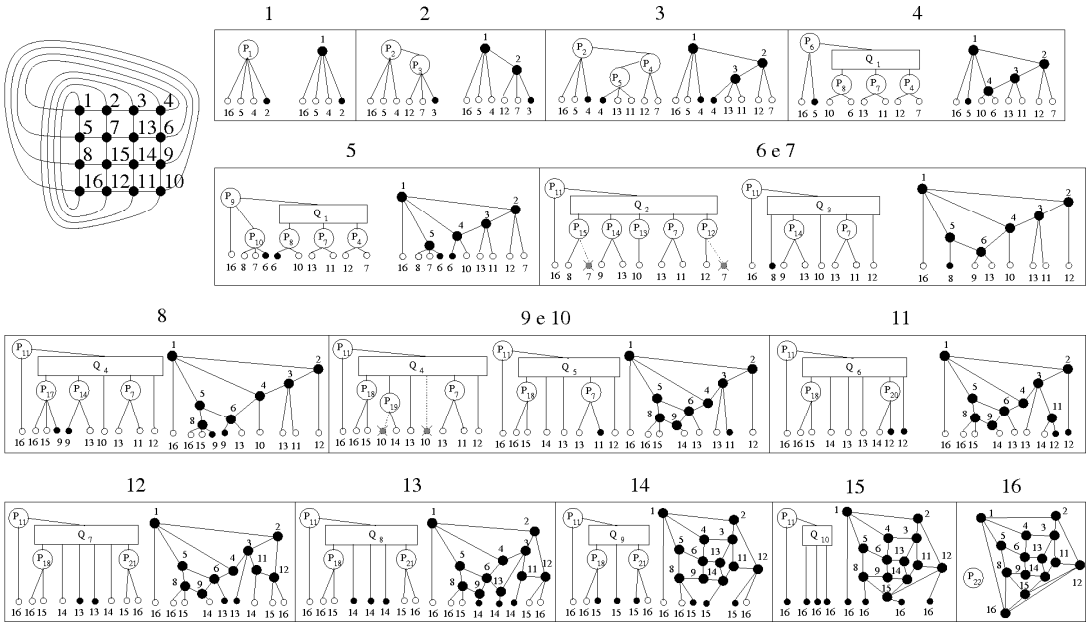

Figura 7: O caso ótimo para um grafo $C_4 \times C_4$

7. Conclusões

É importante salientar que não foi encontrado na literatura algoritmo algum com complexidade linear para planarização de grafos por remoção de vértice. Além do mais, o uso da remoção de vértice é muito raro como ferramenta para planarização de grafo. Note que o algoritmo encontra um subgrafo planar induzido, de componentes biconexos não planar. No entanto, é possível encontrar uma árvore (ou uma floresta se o gráfico não for conexo) de componentes biconexos em tempo linear e construir um subgrafo planar induzido a partir de aplicações sucessivas do algoritmo VD-PLANARIZE.

Um aspecto muito importante a ser destacado é a justificativa do uso de um algoritmo heurístico para planarização de grafos por remoção de vértices. Um trabalho recente (Faria *et al.*, 2006) prova que não existe algoritmo de aproximação para planarização de grafos por remoção de vértices. Portanto, esta é uma forte razão para o uso de algoritmos heurísticos como alternativa para o problema.

A *st*-numeração é a chave para encontrar a melhor sequência de remoção de vértices. Isto também é verdadeiro para a sequência de remoções de arestas no algoritmo de JTS. Este aspecto sugere uma investigação mais minuciosa de como gerar uma *st*-numeração que remova o número mínimo de vértices.

Referências

Booth, K. S. e Lueker, G. S. (1976), Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using *PQ*-Tree Algorithms. *J. Comp. Syst. Sci.*, (3):335–379.

Chiba, T., Nishioka, I. e Shirakawa, I. (1979), An Algorithm of Maximal Planarization of Graphs. *In Proc. 1979 IEEE Int. Symp. on Circuits and Systems*, 648–652.

Di Battista, G. e Nardelli, E. (1988), Hierarchies and Planarity Theory. *IEEE Trans. on Systems, Man., and Cybernetics*, (18): 1035-1046, 1988.

Eades, P. e de Mendonça, C. F. X. (1993), Heuristics for planarization by vertex splitting. In Proc. *ALCOM Int. Workshop on Graph Drawing*, GD'93, 83–85.

Eades, P., Elgindy, H., Houle, M., Lenhart, B., Miller, M., Rappaport, D. e Whitesides, S. (1993), Dominance Drawings of Bipartite Graphs. *Workshop on Layout and Optimal Path Problems at Hawks Nest*, Australia, sponsored by the Department of Computer Science, The University of Newcastle, New South Wales, Australia, 1993.

Even, S. *Graph Algorithms*. Computer Science Press, Potomac, Maryland, (1979).

Even¸S. e Tarjan, R. E. (1976), Computing and st-Numbering. *Theoret. Comp. Sci*., 2:339–344.

Faria, L., Figueiredo, C. M. H. e Mendonça, C. F. X. (1998), Splitting number is Np-complete. 24th Workshop on Graph-Theoretic Concepts in Computer Science, 1998, *Lecture Notes in Computer Science*, 1517: 285-297.

Faria, L., Figueiredo, C. M. H. e Mendonça, C. F. X. (2001), On the Complexity of Approximation of Nonplanarity Parameters for Cubic Graphs. In Proc. Brazilian Symposium on Graphs, Algorithms and Combinatorics, GRACO'2001, *Eletronic Notes in Mathematics*, Elsevier, Fortaleza, Brazil.

Faria, L., Figueiredo, C. M. H. e Mendonça, C. F. X. (2004), On the complexity of the approximation of nonplanarity parameters for cubic graphs. *Discrete Applied Mathematics*, 141(1-3):119–134.

Faria, L., Figueiredo, C. M. H., Mendonça, C. F. X. **e Stolfi, J.** (2006), On maximum planar induced subgraphs, *Discrete Applied Mathematics*, 154: 1774-1782.

Faria, L., Figueiredo, C. M. H. e Mendonça, C. F. X. (1999), Optimal node-degree bounds for the complexity of nonplanarity parameters. In Proc. 10th. Ann ACM- SIAM Symp. on Discrete Algorithms SODA'99, pages 887–888.

Fisher, G. J. e Wing, O. (1966), Computer Recognition and Extraction of Planar Graphs from the Incidence Matrix. *IEEE Trans. Circuit Theory*, CT-13:154–163.

Garey, M. R. e Johnson, D. S. (1983), Crossing number is NP-complete. SIAM J. *Algebraic and Discrete Methods*, 4(3):312–316.

Geldmacher, R. C. e Liu, P. C. (1979), On the deletion of nonplanar edges of a graph. *Congressus Numerantium*, 24:727–738.

Gibbons, A. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge, 1984.

Jayakumar, R., Thulasiraman, K. e Swamy, M. N. S. (1989), O(n2) Algorithms for Graph Planarization. *IEEE Trans. Computer-Aided Design*, 8(3):257–267.

A. Lempel, A., Even, S. e Cerderbaum, I. (1967), An Algorithm for Planarity Testing of Graphs, *Proceddings International Symposium on Theory of Graphs*, Rome, Italy, July 1966. P. Rosenstiel (Ed.), Gordon & Breach, (1):215–232.

Marek-Sadowska, M. (1978). Planarization Algorithm for Integrated Circuits Engineering. In Proc. 1978 *IEEE Int. Symp. on Circuits and Systems*, (1): 919–923.

Ozawa, T. e Takahashi, H. (1981), A Graph-Planarization Algorithm and its Applications to Random Graphs. *Graph Theory and Algorithms, Lectures Notes in Computer Science*, Springer-Verlag, 108: 95–107.

Pasedach, K. Criterion and Algorithms for Determination of Bipartite Subgraphs and their Application to Planarization of Graphs. in *Graphen-Sprachen und Algorithmen in Graphen*, 175-183, Carl Hanser Verlag, Germany, (1976).

Robertson, N. e Seymour, P. D. (1995), Graph minors. XIII. the disjoint paths problems. *Journal of Combinatorial Theory Ser*. B, 63:65–110.

Yannakakis, M. (1079), Node- and edge deletion NP-complete problems. In Proc. 10th. Ann ACM Symp. on Theory of Computing, New York, (1): 253–264.