UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO LANKAITES PINHEIRO

Planarização de grafos por remoção de vértices

Maringá
2009

RODRIGO LANKAITES PINHEIRO

Planarização de grafos por remoção de vértices

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.
Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Ademir Aparecido Constantino

Maringá
2009

"Dados Internacionais de Catalogação-na-Publicação (CIP)"
(Biblioteca Setorial - UEM. Nupélia, Maringá, PR, Brasil)

P654p

Pinheiro, Rodrigo Lankaites, 1983-
Planarização de grafos por remoção de vértices / Rodrigo Lankaites Pinheiro. – Maringá, 2009.
92 f.: il.

Dissertação (Mestrado em Ciência da Computação)--Universidade Estadual de Maringá, Dep. de Informática, 2009.
Orientador: Prof. Dr. Ademir Aparecido Constantino.

1. Planarização de grafos. 2. Grafos - Algoritmos - st-numeração. I. Universidade Estadual de Maringá. Departamento de Informática. Programa de Pós-Graduação em "Ciência da Computação".

CDD 22. ed. -005.1015115
NBR/CIP - 12899 AACR/2

Maria Salete Ribelatto Arita CRB 9/858
João Fábio Hildebrandt CRB 9/1140

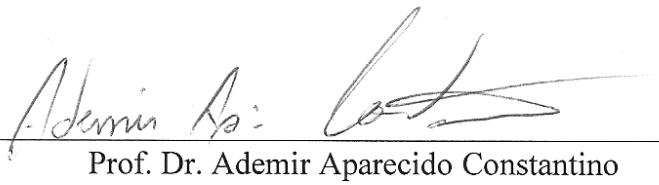
FOLHA DE APROVAÇÃO

RODRIGO LANKAITES PINHEIRO

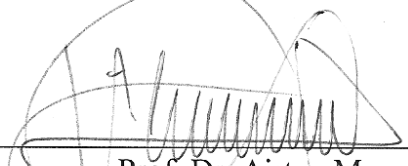
Planarização de grafos por remoção de vértices

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Comissão Julgadora composta pelos membros:

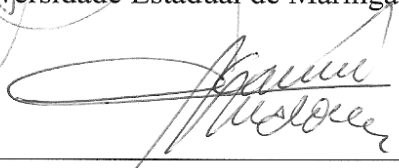
COMISSÃO JULGADORA



Prof. Dr. Ademir Aparecido Constantino
Universidade Estadual de Maringá – DIN/UEM



Prof. Dr. Airton Marco Polidorio
Universidade Estadual de Maringá – DIN/UEM



Prof. Dr. Candido Ferreira Xavier de Mendonça Neto
Universidade de São Paulo – EACH/USP

Aprovada em: 28 de agosto de 2009.
Local de defesa: DIN-UEM, Bloco 19, *campus* da Universidade Estadual de Maringá.

Dedico este trabalho a DEUS: pai e amigo de todas as horas.

AGRADECIMENTOS

Uma dissertação de mestrado nunca é um trabalho individual e ao longo de seu desenvolvimento várias pessoas de alguma forma se envolvem e contribuem para o seu progresso. É com enorme gratidão e estima que aqui expresso meu apreço por aqueles que de alguma forma me ajudaram nesta longa caminhada.

Primeiramente agradeço a DEUS, por iluminar meu caminho, me dar tranqüilidade e sabedoria quando foi preciso e simplesmente por existir, pois sem Ele eu não estaria aqui.

Agradeço a minha família, por me dar apoio e incentivo e por sempre acreditar em mim. Em especial agradeço meu pai pelo exemplo de vida, minha mãe pelos “puxões de orelha” nas horas oportunas e minha irmã por sempre estar disposta a escutar.

Agradeço o meu orientador Ademir pela ajuda, confiança e pela paciência sem fim que sempre demonstrou para comigo.

Agradeço a todos os professores que de alguma forma contribuíram para a minha formação acadêmica.

Agradeço ao Bu, que perto ou longe, junto ou não, sempre me apoiou e esteve comigo.

E finalmente, mas não menos importante, meus agradecimentos aos meus amigos Rafael, Guilherme e Henrique pela grande amizade e companheirismo e pelos bons momentos anti-estresse em nossas sessões de *RPG*.

Agradecer a todos que de alguma forma me ajudaram sem duvidas é a tarefa mais difícil desta dissertação. Tenho conhecimento de que há muitas outras pessoas que fizeram parte deste processo e me ajudaram muito, porém citar todas elas é impossível. A estas pessoas, meus profundos agradecimentos.

Planarização de grafos por remoção de vértices

RESUMO

Neste trabalho são propostos dois algoritmos que utilizam a operação de remoção de vértices para se obter um subgrafo planar. O número de remoção de vértices 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 associado é *NP*-completo, este trabalho propõe o algoritmo heurístico *VD-PLANARIZE* de complexidade $O(m + n)$ para a planarização de grafos utilizando a estrutura de dados árvores-*PQ*, a operação de remoção de vértices e *st*-numeração. Outra proposta apresentada é a do algoritmo genético *GAVD-PLANARIZE* que busca melhorar as soluções do *VD-PLANARIZE*. Este trabalho apresenta detalhes da implementação dos dois algoritmos bem como os resultados que comprovam a complexidade teórica do *VD-PLANARIZE* e os bons resultados obtidos pelo *GAVD-PLANARIZE*.

Palavras-chave: Planarização de grafos. Algoritmos em grafos. *st*-numeração. Algoritmo genético. Árvore-*PQ*.

Graph planarization by vertex deletion

ABSTRACT

This work proposes two algorithms that use the vertex deletion operation to obtain a planar subgraph. The vertex deletion number of a graph G is the lower integer $k \geq 0$ such that there is an induced planar subgraph obtained by the removal of k vertexes from G. Considering that the associated decision problem is *NP*-complete, this work proposes the $O(m + n)$ heuristic algorithm *VD-PLANARIZE* to planarize graphs using the *PQ*-tree data structure, the vertex deletion operation and *st*-numbering. Another proposal is the genetic algorithm *GAVD-PLANARIZE* that looks forward to improve the solutions obtained by the *VD-PLANARIZE*. This work presents details of the implementation of both algorithms as well as the results that aver the theoretical complexity of *VD-PLANARIZE* and show the good results obtained by the *GAVD-PLANARIZE*.

Keywords: Graph planarization. Graph algorithms. *st*-numbering. Genetic algorithm. *PQ*-tree.

Lista de Ilustrações

Lista de Tabelas

Sumário

CAPÍTULO

1

Introdução

Desde os primórdios da humanidade, desenhos e representações gráficas são utilizados para representar objetos, situações e ações. Desde as pinturas rupestres à complexos esquemas automobilísticos, uma representação gráfica não apenas facilita a compreensão, mas ocasionalmente constitui a forma mais confiável para transmitir uma idéia. Existem muitas maneiras de se representar graficamente um conjunto de dados. Uma abordagem eficiente e amplamente utilizada é o uso de grafos.

Desenho de grafos é uma área de pesquisa que investiga técnicas e métodos acerca de formas inteligíveis de representações gráficas de grafos. Considera-se um bom desenho de um grafo se ele trata certos critérios estéticos, tais como simetria, uniformidade da distribuição dos vértices e do comprimento das arestas e número mínimo de cruzamentos.

Dentre os critérios citados, o número mínimo de cruzamentos tem recebido bastante atenção e seu estudo está compreendido no campo da planarização de grafos. A área de planarização de grafos está inclusa no contexto de desenho de grafos e seu principal foco é determinar a melhor maneira de desenhar um grafo em um plano sem que haja cruzamentos de arestas.

Não obstante, em grande parte dos casos, grafos não planares são os que modelam as soluções dos problemas da vida real. Dessa forma, os métodos de planarização são

necessários para se obter uma representação adequada para uma melhor interpretação humana.

1.1. Motivação

Dado um grafo não planar, o problema estudado neste trabalho consiste em encontrar um subgrafo planar, ou seja, um subgrafo que possa ser desenhado no plano sem cruzamento de arestas. Essa atividade para obter um subgrafo planar de um grafo não planar é denominada de *planarização de grafos.* Existem várias operações que podem ser empregadas para encontrar um subgrafo planar, sendo que este trabalho estuda a operação de remoção de vértice. Este trabalho investiga um novo algoritmo heurístico para encontrar um subgrafo planar com o número mínimo de remoção de vértices e uma metaheurística para melhorar as soluções obtidas.

Planarizar um grafo utilizando o número mínimo de remoção de vértices é um problema *NP*-difícil (Lewis e Yannakakis, 1980), portanto não existe nenhum algoritmo determinístico que resolva o problema em tempo polinomial. Além disso, foi provado que não é possível a existência de um algoritmo aproximativo para o problema (Faria et al., 2006), tornando o uso de heurísticas uma opção viável.

Diversos algoritmos são encontrados na literatura a fim de planarizar grafos. Utilizando a operação de remoção de arestas, podemos citar os trabalhos de Fisher e Wing (1966), Pasedach (1976), Sadowska (1978), Chiba, Nishioka e Shirakawa (1979), Ozawa e Takahashi (1981), Jayakumar, Thulasiraman e Swamy (1989) e Vollen (1998) todo com complexidade de tempo $O(n^2)$. Eades e Mendonça (1993) propuseram um algoritmo de planarização que utiliza a operação de divisão de vértices, também de complexidade de tempo $O(n^2)$.

A operação de remoção de vértices é uma operação muito destrutiva, pois remover um vértice de um grafo elimina muita informação, no entanto não se encontra na literatura até o momento nenhum algoritmo que utiliza esta operação para planarizar um grafo, assim como não se encontra nenhum algoritmo com complexidade de tempo linear. Esses fatores constituem a motivação para a realização deste trabalho.

1.2. Objetivos

Este trabalho propõe dois algoritmos de planarização. O primeiro algoritmo, denominado *VD-*

PLANARIZE possui complexidade de tempo $O(m+n)$ e consiste numa modificação do algoritmo de Lempel Even e Cederbaum para que faça a operação de remoção de vértices na planarização de um grafo. Tendo em vista que não se encontra nenhum outro algoritmo de complexidade linear para planarizar grafos, o *VD-PLANARIZE* é até o momento o algoritmo mais rápido para resolver o problema.

O segundo algoritmo proposto, o *GAVD-PLANARIZE* é baseado em algoritmos genéticos e utiliza o *VD-PLANARIZE* para avaliar as soluções e melhorá-las. Um dos fatores que desencoraja o uso de metaheurísticas para a planarização de grafos é a falta de funções de avaliação eficientes, tornando o uso destes algoritmos lentos e, portanto inviáveis. Utilizando o próprio *VD-PLANARIZE* como função de avaliação, temos uma função linear e, portanto o uso da metaheurística passa a ser uma opção para melhorar resultados.

Esta dissertação tem como objetivos apresentar os dois algoritmos de planarização propostos, uma análise de desempenho dos mesmos e informações suficientes para possibilitar a reprodução e compreensão dos algoritmos.

1.3. Organização da Dissertação

A dissertação está dividida em cinco capítulos.

O capítulo 1 situa o leitor no assunto do trabalho, dando uma visão geral do contexto do tema; dos objetivos e da estrutura da proposta.

O capítulo 2 apresenta uma revisão bibliográfica que abrange temas como definições de grafos, invariantes de não-planaridade, *st*-numeração, árvores-*PQ*, algoritmos genéticos e trabalhos relacionados. O capítulo 3 apresenta os algoritmos propostos bem como detalhes técnicos e exemplos de execução.

O capítulo 4 contém os resultados obtidos dos testes dos algoritmos bem como uma comparação entre os métodos de *st*-numeração.

Finalmente o capítulo 5 apresenta conclusão do trabalho.

CAPÍTULO

2

Fundamentação Teórica

Esta seção apresenta conceitos básicos e definições usadas ao longo deste trabalho, tais como: invariantes de não-planaridade, *st*-numeração, árvores-*PQ*, heurísticas e metaheurísticas, algoritmos genéticos e a lista de alguns trabalhos relacionados. O leitor familiarizado com estes conceitos básicos pode saltar para o Capítulo 3.

2.1. Grafos

Um *grafo* é uma tripla ordenada $G = (V, E, \psi)$ onde:

- V é um conjunto finito não vazio de *vértices* (também denominados *pontos* ou *nós*);
- E é um conjunto finito de *arestas* (também denominadas *linhas* ou *arcos*); e
- ψ é uma função de incidência $E \to V \times V$, que associa cada aresta a um par de vértices. Usamos a notação $\psi(e) = \{u, v\}$.

Dada uma aresta $\psi(e) = \{u, v\}$, então os vértices u e v são *adjacentes* ou *vizinhos* e u e v são os *extremos* da aresta e. Dizemos ainda que u e v são *incidentes* a e.

Dado um grafo G, se a função ψ for composta de pares ordenados, dizemos que o grafo é *orientado*, logo $\psi(e) = (u, v)$. Caso contrário, o grafo é *não-orientado*.

Duas arestas distintas que compartilhem os mesmos extremos são chamadas de arestas *múltiplas* ou *paralelas*. Se a função de incidência admite arestas com extremos iguais, isto é,

$\psi(e) = \{u, u\}$, então e é um *laço*. Um grafo que admite múltiplas arestas e laços é denominado *multigrafo*. Um grafo é dito *simples* se não admitir laços ou arestas múltiplas.

Este trabalho utiliza grafos simples, não-orientados, que denominaremos simplesmente de grafos. Denotaremos um grafo fazendo uso da seguinte definição simplificada. Um *grafo* é uma dupla ordenada $G = (V, E)$ onde:

- V é um conjunto, finito, não vazio de n vértices distintos; e
- E é um conjunto finito de m pares de vértices distintos não ordenados de V.

O número de arestas incidentes a um vértice u é denominado *grau* de u, denotado por $d(u)$. O *grau mínimo* (*máximo*) de um grafo G é o menor (maior) grau dentre todos os vértices de G. Os graus mínimos e máximos de um grafo são denotados por δ e Δ, respectivamente. Se todos os vértices de um grafo tem o mesmo grau *d*, o grafo é denominado *d-regular* (ou apenas *regular*). Um grafo 3-regular é também chamado *cúbico*.

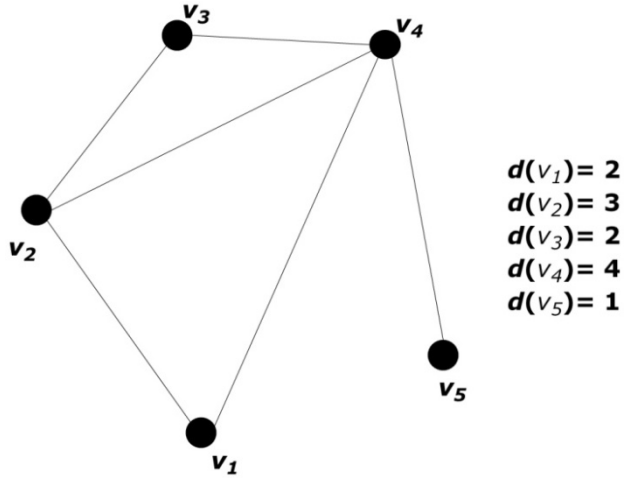


Figura 2.1. Exemplo de grafo com vértices exibindo diferentes graus

Denominamos por $N(u)$, a *vizinhança* de u, que consiste no conjunto de todos os vértices adjacentes a u. A Figura 2.2 ilustra a vizinhança do vértice u.

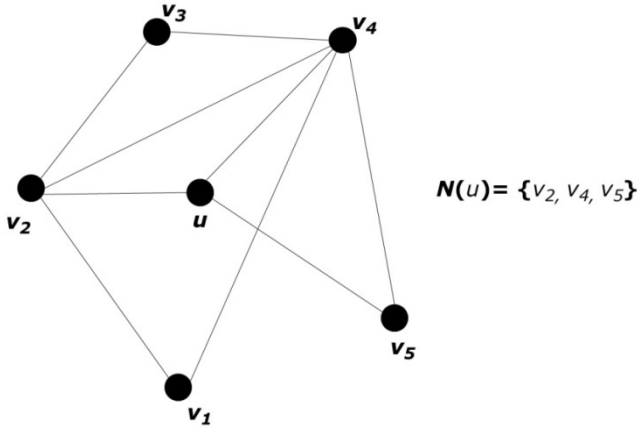


Figura 2.2. Vizinhança de um vértice

Dois grafos $G = (V_G, E_G)$ e $H = (V_H, E_H)$ são chamados *isomorfos* se existe uma bijeção $\varphi: G \to H$ tal que para $\{u, v\} \in V_G$, u e v são adjacentes em G se e somente se os vértices $\varphi(u)$ e $\varphi(v)$ são adjacentes em H.

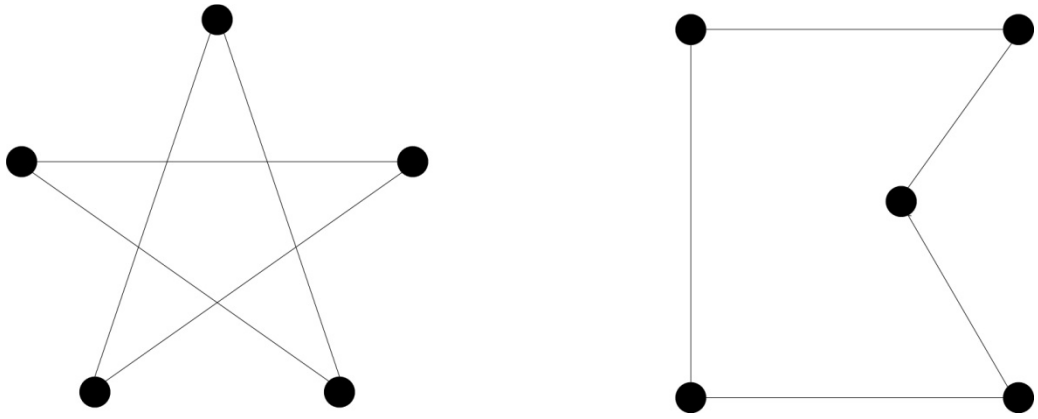


Figura 2.3. Exemplo de grafos isomorfos

Dado um grafo $G = (V, E)$, o grafo $G' = (V', E')$ é denominado *subgrafo* de G se $V' \subseteq V$ e $E' \subseteq \{\{u, v\} \mid u \in V', v \in V',\ e\ \{u, v\} \in E\}$. Além disso, se $V' = V$ então G' é um *subgrafo gerador* (*spanning subgraph*) de G. Se $V' \subset V$ ou $E' \subset E$ (ou ambos), então G' é dito ser um *subgrafo próprio* de G, denotado por $G' \subsetneq G$. Um grafo $G' = (V', E')$ é denominado *subgrafo induzido* (pelo conjunto de vértices V') de G se $V' \subseteq V$ e $E' = \{\{u, v\} \mid u \in V', v \in V'$ e $\{u, v\} \in E\}$. A Figura 2.4 apresenta em (a) um grafo G, em (b) um subgrafo de G e em (c) um subgrafo induzido de G.

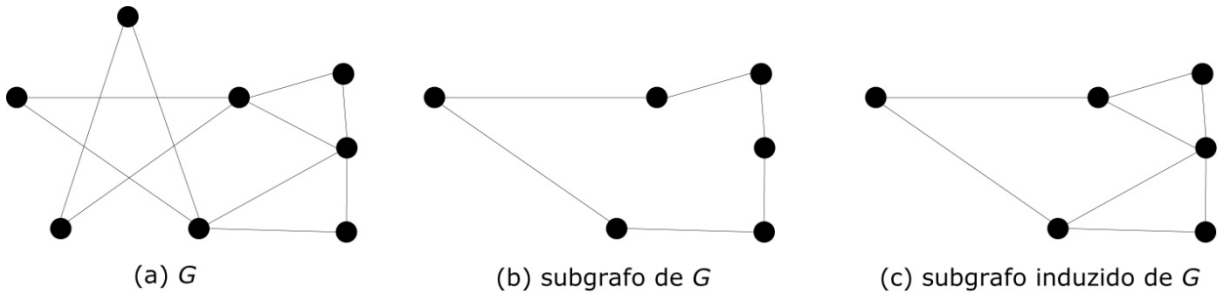


Figura 2.4. Exemplo de subgrafo e subgrafo induzido

Se $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são dois subgrafos (não necessariamente distintos) de um grafo $G = (V, E)$, então o subgrafo $G' = (V_1 \cup V_2, E_1 \cup E_2)$ do grafo G é denominado *união* de G_1 e G_2.

Dado um grafo $G = (V, E)$, um *passeio* é uma seqüência de vértices e arestas $v_0 e_1 v_1 e_2 \ldots e_k v_k$. Um passeio é denominado um *caminho* em G se os $k + 1$ vértices $v_0 \ldots v_k$ são elementos de V, se são par a par distintos, exceto possivelmente por v_0 e v_k, e se v_{i-1} e v_i são incidentes a aresta e_i, para $1 \leq i \leq k$. k é denominado *comprimento* do caminho. Dizemos também que um caminho *conecta* os vértices v_0 e v_k. Alem disso, se $v_0 = v_k$, então

o caminho é denominado um *ciclo*. O comprimento do menor ciclo em G é denominado *cintura* de G. Se um grafo não tem ciclos, ele é dito *acíclico* e sua cintura é indefinida.

Denotamos por P_n o grafo consistindo apenas no caminho de tamanho $n-1$, onde o primeiro e último vértices do caminho são distintos. P_n tem n vértices e $n-1$ arestas. C_n denota o grafo consistindo em um ciclo de comprimento n, contendo n vértices e n arestas.

Se para cada par de vértices u e v de um grafo $G = (V, E)$ existe um caminho em G conectando u a v, então o grafo G é dito *conexo*, do contrário G é *desconexo*. Se $V' \subseteq V$ é o conjunto de vértices tal que o subgrafo G' de G induzido por V' é conexo, e para cada conjunto V''onde $V' \subset V'' \subseteq V$, o subgrafo de G induzido por V'' é desconexo, então G' é um *componente conexo* de G.

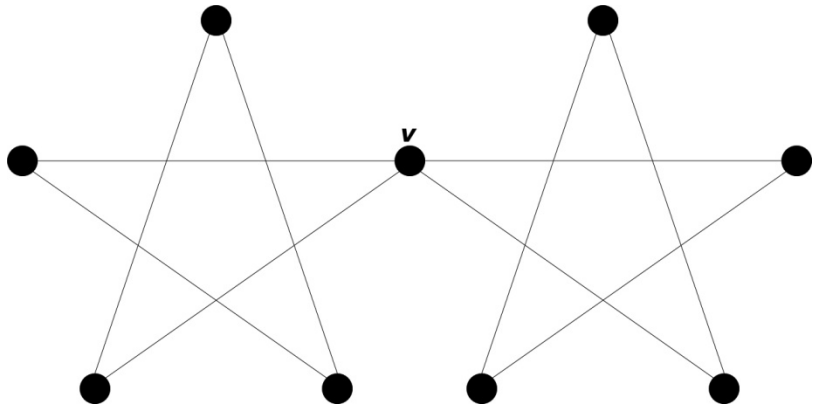


Figura 2.5. Exemplo de grafo conexo onde v representa uma articulação

Dado um grafo $G = (V, E)$, e um vértice $v \in V$, dizemos que o subgrafo G' de G induzido por $V \setminus \{v\}$ é obtido por *remover* v de G. Se G' tem mais componentes conexos do que G, então v é um *vértice de corte* (*articulação*). Se pelo menos k vértices precisam ser removidos de G antes do grafo resultante ser desconexo, ou antes do grafo resultante consistir em um único vértice, então G é dito *k-conexo*.

Seja P uma propriedade qualquer de um grafo G. Dizemos que o subgrafo $H \subseteq G$ é *maximal* (*minimal*) relativo à propriedade P se não existe um subgrafo $H' \subseteq G$ que possui a propriedade P onde $H \subsetneq H'$ ($H' \subsetneq H$). Note que H não é necessariamente o maior (menor) subgrafo com a propriedade P, no entanto quando isso for verdade dizemos que o subgrafo H é *máximo* (*mínimo*) relativo à propriedade P.

Se uma aresta $e = \{u, v\}$ de um grafo $G = (V, E)$ é substituída pelo caminho $ue'v_e e''v$ introduzindo um novo vértice $v_e \notin V$, então é dito que o grafo $G' = (V \cup \{v_e\}, (E \setminus \{e\}) \cup \{e', e''\})$ é obtido de G pela *subdivisão* da aresta e.

Figura 2.6. Exemplo de subdivisão da aresta $\{u, v\}$

Dois grafos G e G' são *homeomorfos*, se G' puder ser obtido de G por meio de uma seqüência de subdivisões de arestas. A Figura 2.7 apresenta um exemplo de dois grafos homeomorfos.

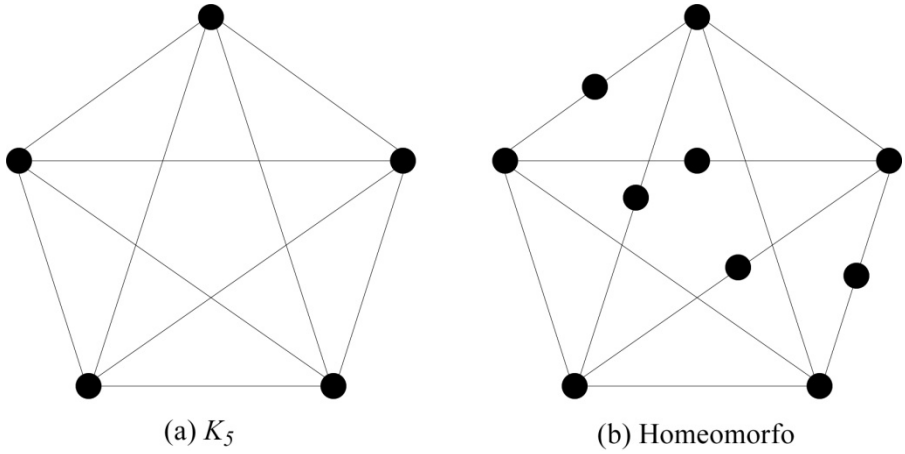


Figura 2.7. Exemplo de grafos homeomorfos

Além dos caminhos P_n e dos ciclos C_n, os seguintes grafos especiais aparecem no texto:

Para $n \geq 2$, o *grafo completo*, denotado por K_n, consiste de n vértices juntos a todas as $\binom{n}{2}$ arestas possíveis. Então para um K_n todo vértice é adjacente a todo outro vértice. Definimos o K_1 como sendo um único vértice, o K_2 como sendo uma única aresta e seus dois vértices, e o K_3 é um triangulo.

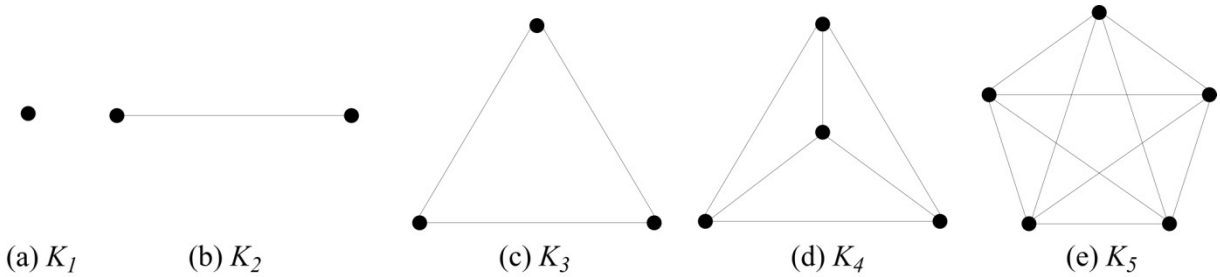


Figura 2.8. Grafo completo

O *grafo completo bipartido*, denotado por K_{n_1,n_2}(ou K_{nm}), consiste de dois conjuntos de vértices disjuntos $V = \{v_1, \dots, v_{n_1}\}$ e $W = \{w_1, \dots, w_{n_2}\}$ e o conjunto de arestas $E = \{v_i w_j \mid 1 \leq i \leq n_1 \; e \; 1 \leq j \leq n_2\}$ de todas as arestas entre os vértices de V e os vértices de W. Note que $K_{n_1,n_2} = K_{n_2,n_1}$.

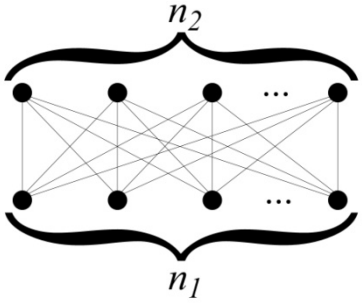
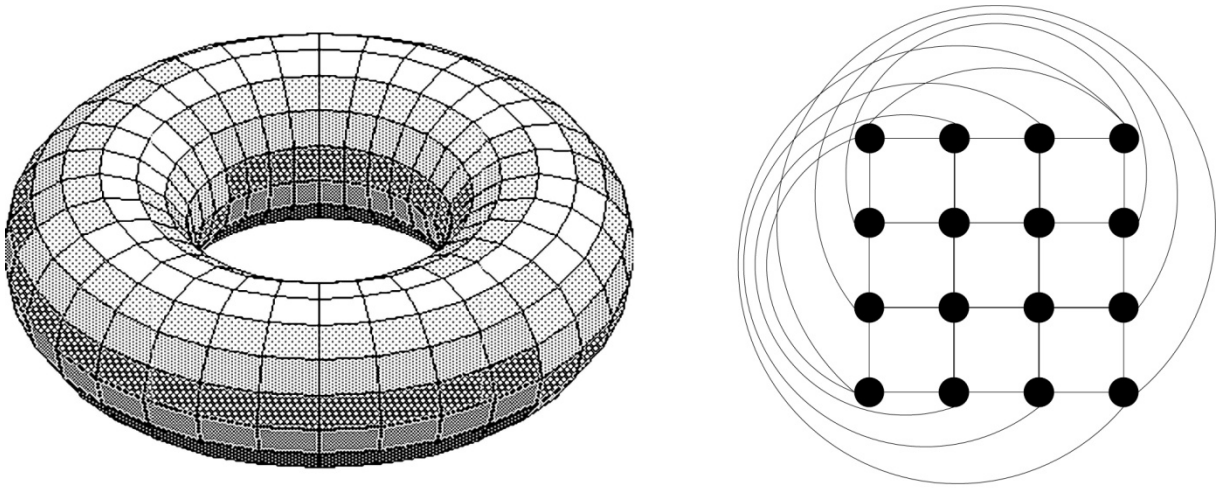


Figura 2.9. Exemplo de um grafo $K_{n1,n2}$

O *grafo produto cartesiano* $C_n \times C_m$ de dois ciclos C_n e C_m é o grafo contendo nm vértices $\{v_{i,j}\}$ e $2nm$ arestas $\{v_{i,j}, v_{(i+1) mod\, n,j}\}$ e $\{v_{i,j}, v_{i,(j+1) mod\, m}\}$, para $0 \leq i \leq n$ e $0 \leq j \leq m$. Considerando que cada vértice do $C_n \times C_m$ é representado por um ponto no plano com coordenadas (i, j), chamamos as duas famílias de arestas acima de *horizontal* e *vertical*, respectivamente. Um ciclo do grafo $C_n \times C_m$ é chamado de *meridiano* se usa apenas arestas verticais, e *paralelo* se usa apenas arestas horizontais. Assim, o grafo $C_n \times C_m$ possui n meridianos isomorfos ao C_n e m paralelos isomorfos ao C_m. Esses grafos são também conhecidos como *grafos toroidais*, pois eles podem ser desenhados no toro sem que as arestas se cruzem e sem que haja sobreposição de vértices e arestas. Um *toro* é uma superfície topologicamente similar a uma esfera com uma alça, ou igual a forma de um "pneu", como mostra a Figura 2.10.

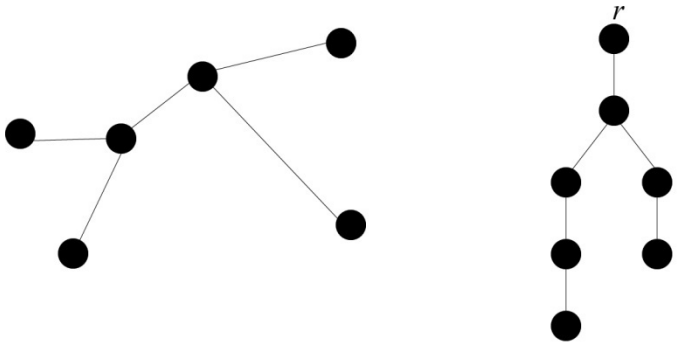


(a) Toro (b) $C_4 \times C_4$

Figura 2.10. Exemplo de toro e grafo cartesiano $C_4 \times C_4$

Um grafo conexo $G = (V, E)$ é uma *árvore*, denotada por $T = (V, E)$, se G não contém ciclos, portanto cada vértice de grau maior que um é uma articulação. Dada uma árvore T, existe apenas um caminho entre dois nós quaisquer. Uma árvore T é dita *enraizada*, denotada por (T, r), se ela possuir um nó $r \in T$ definido como *raiz*. Dada uma árvore

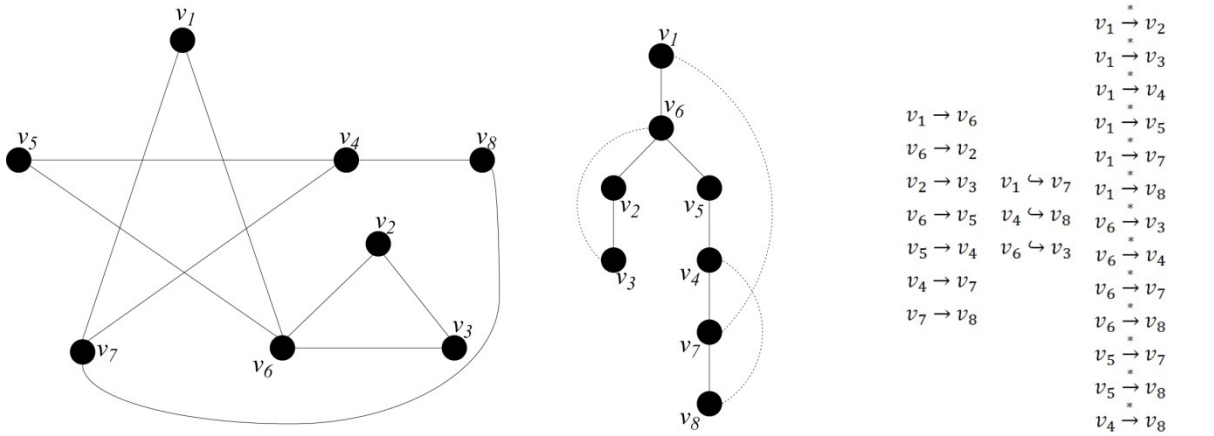
enraizada (T, r) e dois nós $\{u, v\} \in V$, u será chamado de *ancestral* de v e v será chamado de *descendente* de u se u está no caminho de r para v. Uma aresta $e = \{u, v\} \in T$ é denominada a *aresta da árvore* e denotada por $u \rightarrow v$ se u é ancestral de v e $v \rightarrow u$, se v é ancestral de u. Seja $u \rightarrow v$ uma aresta de T, dizemos que u é *pai* de v e v é *filho* de u. A Figura 2.11 apresenta um exemplo de árvore e uma árvore enraizada.



(a) árvore (b) árvore enraizada

Figura 2.11. Exemplo de árvore e árvore enraizada

Uma árvore enraizada (T, r) é uma *árvore de expansão* de um grafo G se T é um subgrafo de G e existe um vértice r em T tal que para toda aresta $e = \{u, v\}$, ou $e \in T$ ou u é ancestral de v ou v é ancestral de u em (T, r). Toda aresta $\{u, v\} \in G$ que não se encontra em (T, r) é denominada *aresta de retorno*, denotada por $u \hookrightarrow v$. Dada uma árvore (T, r), um caminho P de u para v (possivelmente vazio) onde todas as suas arestas pertençam a T é denotado por $u \overset{*}{\rightarrow} v$. A Figura 2.12 apresenta um exemplo de uma árvore de expansão e o seu grafo gerador, arestas pontilhadas na arvore representam as arestas de retorno.



(a) G (b) árvore de expansão de G com $r = v_1$ (c) arestas da árvore, arestas de retorno e caminhos da árvore de expansão de G

Figura 2.12. Exemplo de árvore de expansão

2.2. Invariantes de Não-Planaridade

A seção 2.1 apresentou algumas definições sobre grafos. Esta seção introduz as definições de desenho de grafos e algumas invariantes de não-planaridade. Para maiores informações acerca deste conteúdo, recomenda-se a leitura do trabalho de Liebers (2001).

A teoria dos grafos lida com imersão de grafos em superfícies. Uma *superfície* é um espaço topológico que é uma variedade compacta de duas dimensões. Como o plano é uma superfície de tamanho indefinido, no contexto da teoria dos grafos ele é comumente substituído pela esfera.

Uma *imersão* (*embedding*), também denominado *desenho*, de um grafo G em uma superfície S é o mapeamento dos vértices de G em pontos distintos de S, e das arestas de G em arcos abertos disjuntos de S onde:

- a representação dos vértices e das arestas são distintas; e
- a representação de uma aresta $\{u, v\}$ une a representação dos vértices u e v.

Uma *imersão planar* de um grafo G é uma imersão de G no plano onde não há cruzamentos de arestas. Um grafo G que admite uma imersão planar é denominado *grafo planar*. Claramente um grafo é planar se, e somente se, cada um de seus componentes conexos for planar.

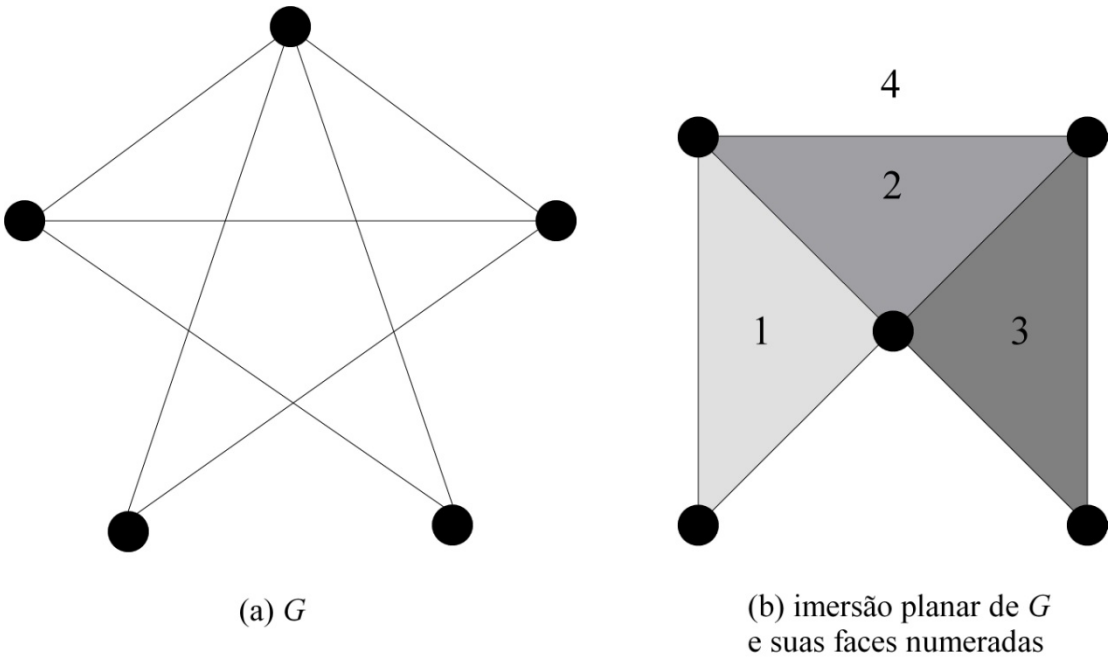


Figura 2.13. Exemplo de imersão planar e suas faces

Seja G um grafo planar e R uma imersão de G em um plano S. As linhas de R dividem S em regiões, as quais são denominadas de *faces* de R. A *face externa* é a face que não é

delimitada por vértices e arestas. As demais faces são denominadas *faces internas*. A Figura 2.13 apresenta um exemplo de grafo G planar com uma imersão não planar em (a). Em (b) é apresentado o mesmo grafo G de (a) com uma imersão planar e suas faces, numeradas, aparecem em tons de cinza, sendo que a face externa não recebe cor.

Seja G um grafo planar com n vértices e m arestas, R sua imersão planar em um plano S, e f o número de faces de R, Euler encontrou a seguinte formula:

$$n - m + f = 2 \quad \text{(Euler, 1750)} \tag{1}$$

A demonstração desta propriedade pode ser encontrada no trabalho de Nishizeki e Chiba (1988). Note que se um grafo planar com $n \geq 3$ contendo o máximo possível de arestas, então cada face é incidente a exatamente três vértices. O número de arestas em tal grafo é $m = 3n - 6$. Dessa forma a formula de Euler gera o seguinte corolário:

$$m \leq 3n - 6 \quad \text{(para } n \geq 3\text{)} \tag{2}$$

Logo, qualquer grafo com m acima deste limite não é planar.

A caracterização mais conhecida de grafos planares é provavelmente a proposta por Kuratowski (1903) que define que um grafo G é planar se, e somente se, ele não contém um subgrafo homeomorfo ao $K_{3,3}$ ou ao K_5. A Figura 2.14 apresenta um exemplo de um grafo que contém um subgrafo homeomorfo ao $K_{3,3}$. Os vértices mais claros representam os vértices do $K_{3,3}$, sendo que quadrados representam um subgrupo dos vértices e círculos representam o outro. As arestas mais claras representam as arestas utilizadas para formar o $K_{3,3}$.

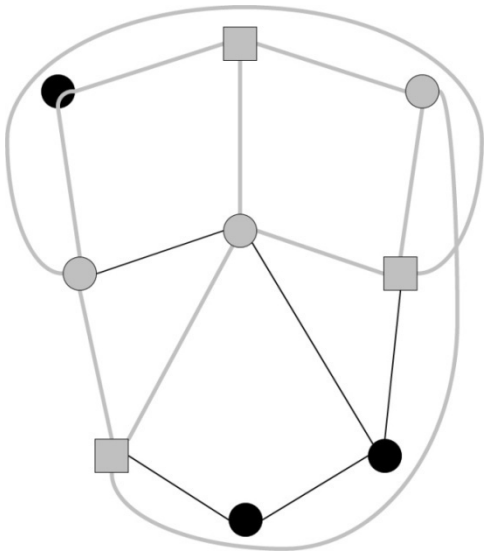


Figura 2.14. Grafo contendo um subgrafo homeomorfo ao $K_{3,3}$

Dado um grafo G que não admite uma imersão planar, é possível efetuar *operações de planarização* de maneira que o grafo resultante G' seja planar. Existem diversas operações que *planarizam* um grafo, desta forma, se aplicadas em um grafo G, produzem um grafo

resultante G' planar. Todas as operações de planarização alteram a estrutura do grafo de alguma forma, portanto é importante encontrar o menor número possível de operações a serem efetuadas no grafo.

Uma *invariante de não-planaridade* é uma medida que quantifica o quão distante de ser planar é um grafo, com relação a uma operação de planarização.

2.2.1. Cruzamento de Arestas

Em desenhos de grafos, e também em outras áreas como em projeto de circuitos *VLSI*, está-se interessado em desenhar um grafo dado com o mínimo possível de arestas que se cruzam. A invariante de não-planaridade básica no contexto de teoria dos grafos é o *número de cruzamento de arestas* de G, denotado por $cr(G)$, que é o menor número k tal que G possa ser imerso no plano com no máximo k cruzamentos de arestas. O número de cruzamentos de arestas de uma imersão de G é sempre maior ou igual a $cr(G)$, o que justifica o termo "invariante". Denominamos um *desenho ótimo* de G um desenho de G que apresente tantos cruzamentos de arestas quanto o valor de $cr(G)$. A Figura 2.15 mostra um desenho ótimo do K_5.

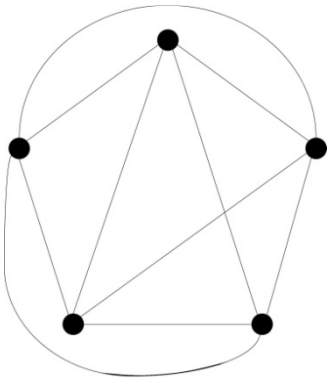
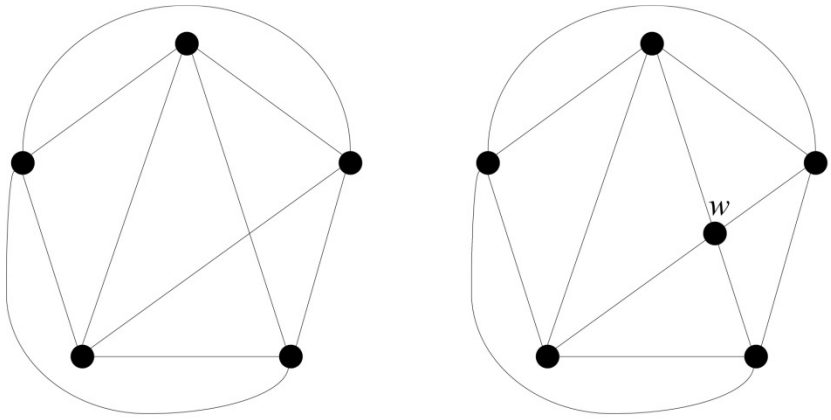


Figura 2.15. Número de cruzamentos de arestas

Claramente, o número de cruzamentos de arestas de um grafo é 0 se e somente se o grafo for planar. Um grafo G com n vértices, m arestas e com $cr(G) > 0$ (e dada uma imersão de G com $cr(G)$ cruzamentos de arestas) pode ser transformado em um grafo planar introduzindo $cr(G)$ novos vértices no exato lugar em que duas arestas se cruzam. O novo grafo G' possui $n + cr(G)$ vértices e $m + 2cr(G)$ arestas. Dizemos que G' foi obtido de G por meio de uma *inserção de vértice falso*. A Figura 2.16 ilustra a operação de inserção de vértice falso. Em (a) é apresentado um desenho do K_5 e em (b) insere-se um novo vértice *w* onde há o cruzamento de arestas.



(a) K_5 (b) inserção de um novo vértice w onde há cruzamento de aresta

Figura 2.16. Operação de inserção de vértice falso

Como destaca Liebers (2001), o problema do cruzamento de arestas pode ser definido como se segue. Dado um grafo G e um número inteiro positivo k, existe uma imersão de G no plano com k ou menos cruzamentos de arestas?

Garey e Johnson (1983) provaram que esse problema é *NP*-completo, logo o problema de se determinar o valor de cruzamentos de arestas para um determinado grafo não é passível de ser resolvido em tempo polinomial.

2.2.2. Remoção de Vértices

Dado um grafo $G = (V, E)$, pode-se obter um subgrafo planar $G' = (V', E')$, de maneira trivial, removendo todos menos quatro vértices de G e suas arestas incidentes. G' é então um K_4 ou um subgrafo do K_4 e portanto é planar. Sabendo que é possível planarizar um grafo removendo vértices, o problema passa a ser o de planarizar um grafo removendo o mínimo de vértices possíveis.

Se um grafo $G' = (V', E')$ é um subgrafo planar induzido de um grafo $G = (V, E)$ onde não exista um subgrafo planar induzido $G'' = (V'', E'')$ de G com $|V''| > |V'|$, então G' é denominado um *subgrafo planar induzido máximo*.

O problema de encontrar a menor quantidade possível de vértices a serem removidos de um grafo para que o grafo resultante seja planar significa encontrar o subgrafo planar induzido máximo. A esse problema é atribuída a invariante de não-planaridade *número de vértices removidos* (*vertex deletion number*) de um grafo G, denotada por $vd(G)$ e representa o número mínimo de vértices a serem removidos de G a fim de se obter um subgrafo planar de G.

Para um grafo G, o problema de encontrar $vd(G)$ possui um problema de decisão associado *NP*-completo, como pode ser encontrado nos trabalhos de Lewis e Yannakakis (1980) e Garey e Johnson (1979). Além disso, Faria et al. (2006) propõem que não existem algoritmos aproximativos para resolver o problema.

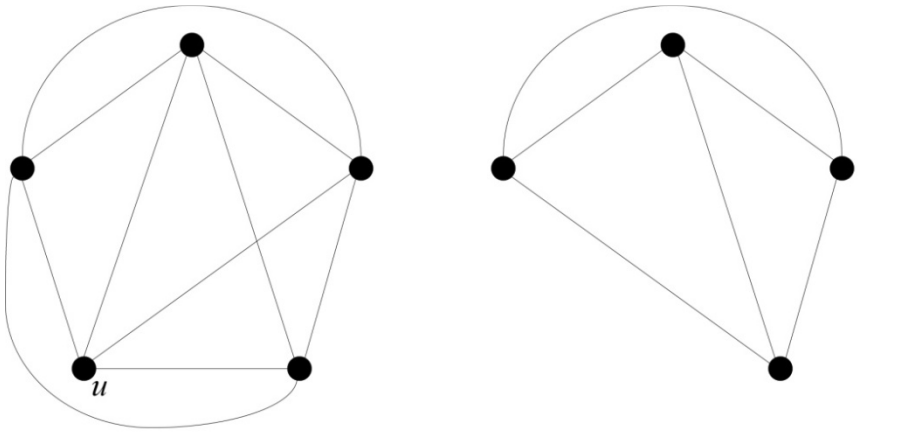


Figura 2.17. Operação de remoção de vértice

2.2.3. Remoção de Arestas

Se um grafo $G = (V, E)$ com uma aresta $e \in E$ é transformado no grafo $G' = (V, E \setminus \{e\})$ então definimos que G' foi obtido de G por meio da r*emoção da aresta* e. Trivialmente, G pode se tornar planar por meio de repetidas operações de remoção de arestas. O objetivo é encontrar um subgrafo planar G' de G com o maior número de arestas possíveis, removendo o menor número de arestas de G.

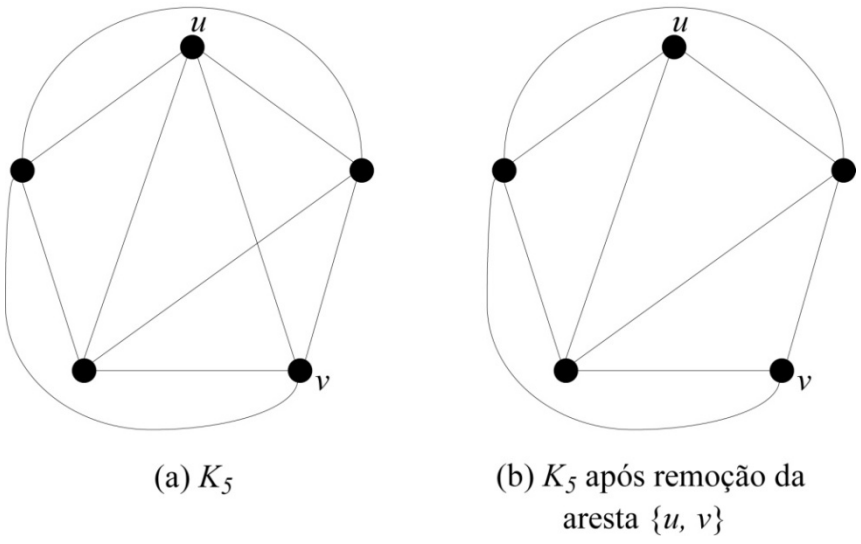


Figura 2.18. Operação de remoção de aresta

Liebers (2001) apresenta o conceito da seguinte forma. Se um grafo $G' = (V, E')$ é um subgrafo planar de um grafo $G = (V, E)$ tal que não existe um subgrafo planar $G'' = (V, E'')$ de G onde $|E''| > |E'|$, então G' é chamado de *subgrafo planar máximo* de G. Ao número correspondente $|E| - |E'|$, denomina-se a invariante de não-planaridade *número de remoção de arestas* (*skewness*), denotada por $sk(G)$.

Dado um grafo G, encontrar o valor de $sk(G)$ possui um problema de decisão associado *NP*-completo, como apresentado nos trabalhos de Yannakakis (1978), Liu e Geldmacher (1979) e Watanabe e Nakamura (1983).

Dentre os trabalhos encontrados na literatura que tratam a planarização de grafos, a operação mais utilizada é a de remover arestas do grafo, pois esta operação não elimina muita informação do grafo original se comparada às outras operações e, de certa forma, ela é bem simples. Dentre os trabalhos que abordam o número de remoção de arestas, podemos destacar os de Lempel, Even e Cederbaum (1967), Chiba, Nishioka e Shirazawa (1979), Ozawa e Takahashi (1981) e Jayakumar, Thulasiraman e Swamy (1989) que propõem algoritmos de planarização de grafos utilizando a remoção de arestas.

2.2.4. Divisão de Vértices

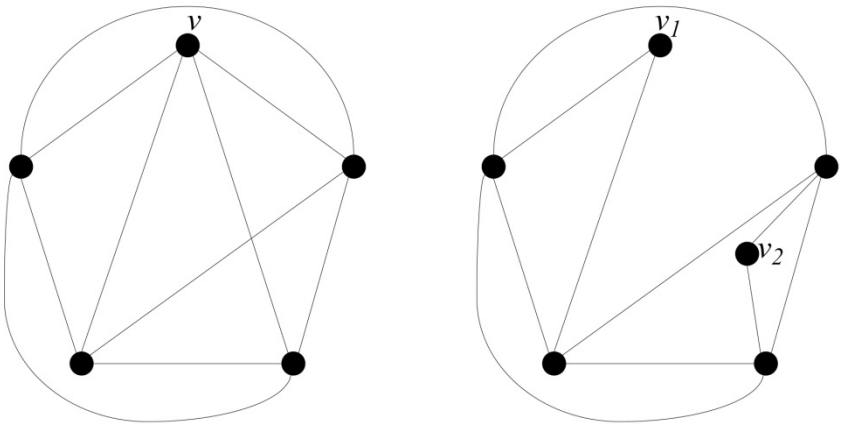
Se $G' = (V', E')$ é um grafo com dois vértices $v_1 \in V'$ e $v_2 \in V'$, e se $G = (V, E)$ é o grafo obtido de G' onde

$$V = (V' \setminus \{v_1, v_2\}) \cup \{v\} \text{ e}$$
$$E = \left(E' \setminus \{\{u, v_i\} \mid u \in V' e\ i \in \{1,2\}\ e\ \{u, v_i\} \in E'\}\right)$$
$$\cup \{\{u, v\} \mid u \in V \setminus \{v\}\ e\ (\{u, v_1\} \in E' ou\ \{u, v_2\} \in E')\},$$

então dizemos que G' foi obtido de G por meio da *divisão do vértice* v.

Uma maneira mais intuitiva de definir a operação de divisão de vértices é: dado um grafo $G = (V, E)$, um vértice $v \in V$ e um subgrafo $G' = (V', E')$ obtido de G pela remoção do vértice v, dividir v particiona o conjunto de vértices adjacentes de v em dois subconjuntos não vazios S_1 e S_2 e adiciona a G' dois novos vértices v_1 e v_2 não adjacentes, onde a vizinhança do vértice v_1 é definida por S_1 e a vizinhança do vértice v_2 é definida por S_2.

Seja G um grafo, a invariante de não-planaridade *número de divisão de vértices* de G, denotado por $sp(G)$, é o menor número inteiro positivo k, tal que um grafo planar possa ser obtido de G por k operações de divisão de vértices.



(a) K_5 (b) K_5 após a divisão do vértice v nos vértice v_1 e v_2

Figura 2.19. Operação de divisão de vértice

Faria et al. (2001) provaram que encontrar o número de divisão de vértices de um grafo possui um problema de decisão associado *NP*-completo.

Liebers (2001) apresenta que a operação de divisão de vértices está relacionada a diferentes contextos, como a decomposição de um grafo em blocos biconexos por meio de divisão dos vértices de corte do grafo. Outro contexto na qual a divisão de vértices é aplicada é que a cada triangulação de plano pode ser gerada de uma imersão planar do K_4 por meio de uma divisão de vértices, como apresenta Steinitz e Rademacher (1934). No entanto, como evidenciou Inácio (2003), a maior contribuição desta operação está na área de desenhos de grafos planares, pois já que a operação de divisão de vértices não destrói informação do grafo original, ela possui diversas aplicações práticas.

2.2.5. Particionamento Planar

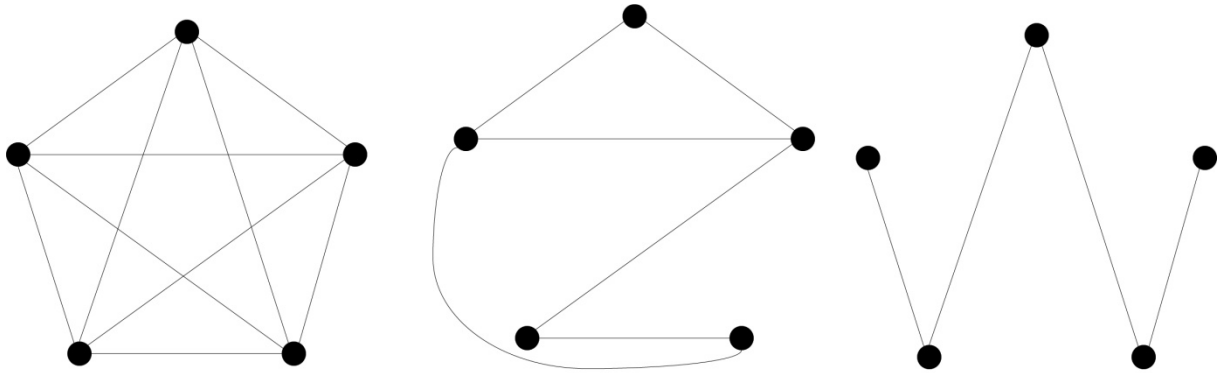
O *número de particionamentos* (*thickness*) de um grafo G, denotado por $\Theta(G)$, é o número mínimo de subgrafos planares de G, tal que a união seja G.

Claramente, se o número de particionamentos de um grafo é 1 se e somente se o grafo é planar. Tome por exemplo o grafo da Figura 2.20, onde em (a) apresenta-se um grafo K_5, e em (b) dois subgrafos planares cuja união é o próprio K_5.

Mansfield (1983) determinou que encontrar o numero de particionamentos de um grafo G é um problema *NP*-difícil reduzindo o correspondente problema de decisão a um 3-*SAT* planar.

Liebers (2001) destaca que o estudo dessa invariante tem sido amplamente estudado como parte de estudos topológicos sobre teoria dos grafos, mas poucos algoritmos foram

desenvolvidos para encontrar o número de particionamentos de um grafo. Os primeiros estudos sobre essa invariante e a introdução a esses estudos são descritos em detalhes por Hobbs (1969). Alguns *surveys* de destaque desta invariante são os apresentados por Arthur et al. (1978), Barnette e Edelson (1988) e Mutzel et al. (1998).



(a) K_5 (b) dois grafos cuja união gera o K_5

Figura 2.20. Operação de particionamento planar

2.2.6. Resultados para Classes Particulares de Grafos

Observa-se que independente da invariante de não-planaridade escolhida, encontrar o seu valor para o caso geral é um problema da classe *NP*. No entanto, para algumas classes específicas de grafos este valor pode ser determinado ou aproximado, conforme apresenta Tabela 2.1

Para o número de cruzamentos de arestas, a prova do limite superior para grafos K_n pode ser encontrada nos trabalhos de Guy (1971) e White e Beineke (1978). Posteriormente Leighton (1984) propôs um limite inferior. Zarankiewicz (1954) conjecturou o valor exato para grafos K_{nm} e posteriormente provou-se que havia erros e o valor de Zarankiewicz foi tomado como um limite superior. Kleitman (1970) provou a conjectura de Zarankiewicz para $min\{m, n\} \leq 6$ e Woodall (1993) provou para $m \leq 10,\ n \leq 8$. Para grafos $C_n \times C_m$, Harary et al. (1973) conjecturou o atual valor que foi provado por Salazar (1998).

Sobre o número de remoção de arestas, informações acerca do valor para grafos K_n e K_{nm} podem ser encontradas nos trabalhos de Vollen (1998) e Liebers (2001). Para grafos $C_n \times C_m$, Mendonça Neto et al. (2002) propuseram os valores apresentados na Tabela 2.1, O símbolo δ é o *delta de Kronecker*, onde $\delta_{i,j}$ é 1 se $i = j$ e 0 se $i \neq j$.

Tabela 2.1. Medidas de não-planaridade para classes específicas de grafos

Classe do Grafo	Medida de não-planaridade
	cr(G)
$G = K_n$	$\leq \frac{1}{4}\left\lfloor\frac{n}{2}\right\rfloor\left\lfloor\frac{n-1}{2}\right\rfloor\left\lfloor\frac{n-2}{2}\right\rfloor\left\lfloor\frac{n-3}{2}\right\rfloor$, para $n \leq 10$ $\geq \frac{1}{120}n(n-1)(n-2)(n-3)$, para $n \geq 5$
$G = K_{nm}$	$\leq \left\lfloor\frac{m}{2}\right\rfloor\left\lfloor\frac{m-1}{2}\right\rfloor\left\lfloor\frac{n}{2}\right\rfloor\left\lfloor\frac{n-2}{2}\right\rfloor$, para $min\{m,n\} \leq 6$ e $7 \leq m \leq 10, 7 \leq n \leq 8$ $9\left\lfloor\frac{m-1}{2}\right\rfloor\left\lfloor\frac{m}{2}\right\rfloor$, para $n = 7$ $\begin{cases} m \text{ ímpar}, m \geq 7 \\ m \text{ par}, \ m \geq 8 \end{cases}$
$G = C_n \times C_m$	$\leq (n-2)m$, para $3 \leq n \leq m$
	sk(G)
$G = K_n$	$= \frac{(n-3)}{2}\frac{(n-4)}{2}$, para $n \geq 4$
$G = K_{nm}$	$= (m-2)-(n-2)$, para $m, n \geq 2$
$G = C_n \times C_m$	$= min\{m,n\} - \delta_{3,n}\delta_{3,m} - \delta_{3,n}\delta_{4,m} - \delta_{4,n}\delta_{3,m}$
	vd(G)
$G = K_n$	$= (n-4)$, para $n \geq 4$
$G = K_{nm}$	$= min\{m,n\} - 2$, para $m, n \geq 3$
	tk(G)
$G = K_n$	$= \left\lfloor\frac{n+7}{6}\right\rfloor$, para $n \geq 1, n \neq 9, n \neq 10$
$G = K_{nm}$	$= \left\lceil\frac{mn}{2(m+n-2)}\right\rceil$, quando m, n são par e $n \leq m$, então $n = \left\lfloor\frac{2k(m-2)}{m-2k}\right\rfloor$

Xavier (1999) realizou um estudo acerca das invariantes de planaridade e apresentou valores para o número de remoção de vértices.

Para a invariante número de particionamento planar, informações acerca do valor para os grafos K_n podem ser encontradas nos trabalhos de Battle et al. (1962), Tutte (1963), White et al. (1978) e Wessel (1985). Para os grafos K_{mn}, informações podem ser obtidas nos

trabalhos de Beineke et al. (1964) e Beineke (1967).

O algoritmo *VD-PLANARIZE*, estudado neste trabalho, utiliza a operação de remoção de vértices para planarizar um grafo. Conforme apresentado na seção 2.2.2, encontrar o valor de $vd(G)$ é um problema da classe *NP*, portanto não pode ser resolvido otimamente em tempo polinomial, além disso, não é possível desenvolver um algoritmo aproximativo para resolver o problema, desta forma justificando o uso de soluções heurísticas.

O algoritmo *VD-PLANARIZE* requer que uma ordem respeitando certos parâmetros seja estabelecida nos vértices do grafo a ser planarizado, esta ordem é adquirida por meio de um processo de aquisição denominado *st*-numeração.

2.3. *st*-Numeração

Seja $G = (V, E)$ um grafo biconexo, $s, t \in V$ e $\{s, t\} \in E$. Uma *st-numeração* de G consiste em rotular o vértice s, denominado *fonte* (*source*), com o valor de rótulo 1, o vértice t, denominado *sumidouro* (*sink*), com o valor de rótulo $|V|$, e cada vértice $v \in V \setminus \{s, t\}$, a v é atribuído um valor de rótulo tal que v seja adjacente a pelo menos um vértice de valor de rótulo maior e a pelo menos um vértice de valor de rótulo menor. O rótulo atribuído a um vértice dessa maneira é denominado *st-número*. Chamamos de *st-grafo* um grafo com uma *st*-numeração atribuída. A Figura 2.21 apresenta um exemplo de um *st*-grafo onde a *st*-numeração é apresentada próxima a cada vértice.

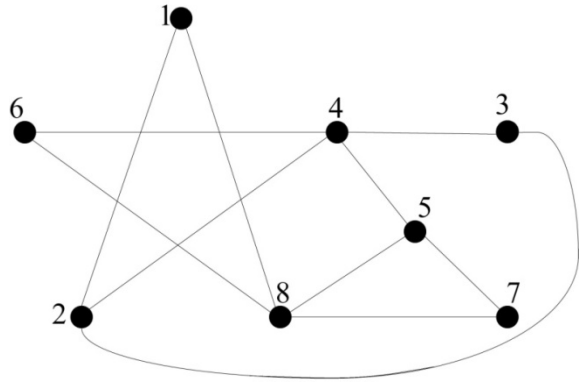


Figura 2.21. Exemplo de um st-grafo

Lempel, Even e Cederbaum (1967) demonstraram que cada grafo biconexo tem pelo menos uma *st*-numeração. Além disso, cada grafo possui mais de uma st-numeração (Bueno, 2005).

A *st*-numeração constitui um passo fundamental para testes de planaridade, planarização de grafos e desenhos de grafos. Ela foi proposta por Lempel, Even e Cederbaum

(1967) como parte integrante de um trabalho sobre um algoritmo de planarização de grafos. Posteriormente outros métodos foram propostos.

O primeiro algoritmo proposto por Lempel, Even e Cederbaum (1967) possui complexidade de tempo $O(mn)$. Neste trabalho são apresentados detalhes de dois algoritmos: Even-Tarjan e de Tarjan, pois ambos apresentam complexidade de tempo $O(m+n)$.

2.3.1. Algoritmo de Even-Tarjan

O primeiro algoritmo de complexidade de tempo linear capaz de efetuar uma *st*-numeração em um grafo foi proposto por Even e Tarjan (1976). Esse algoritmo é dividido em duas fases: a fase de montar a árvore de expansão baseada em uma busca em profundidade e; a fase que decompõe em caminhos e realiza a *st*-numeração dos vértices.

Dado um grafo biconexo *G* e uma aresta $\{s,t\}$, o algoritmo de busca em profundidade gera a árvore de expansão (T,t) onde a primeira aresta de (T,t) é a aresta fornecida $\{s,t\}$. A Figura 2.22 apresenta esse algoritmo de busca em profundidade.

```
    DFS
    Entrada: grafo biconexo G e uma aresta {s,t}
    Saída: árvore T e valores de low, pai e pre
    Pré-processamento: inicializar v para {s,t}, pre(v) := 0 ∀v ∈ V e cont: = 1
1:  Início:
2:      pre(v) := cont;
3:      cont := cont + 1;
4:      low(v) := pre(v);
5:      para cada w ∈ todos_vertices_adjacentes_ao_vertice(v) faça
6:          se pre(v) = 0 então
7:              T := T ∪ {v → w};
8:              pai(w) := v;
9:              DFS(v);
10:             low(v) := min{low(v), low(w)};
11:         senão se w ≠ pai(v) então
12:             low(v) := min{low(v), pre(w)};
13: Fim;
```

Figura 2.22. Pseudocódigo da primeira fase do algoritmo Even-Tarjan

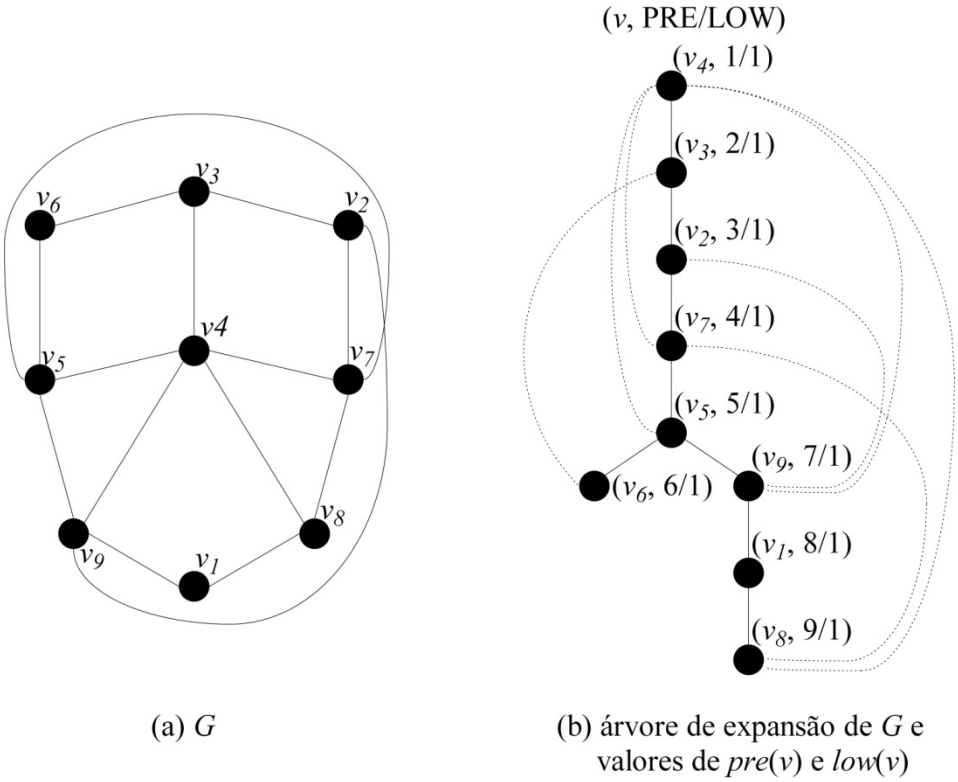
O procedimento *DFS* (*Depth First Search*), apresentado na Figura 2.22, recebe como entrada um grafo biconexo G e um vértice v, produz como saída uma árvore de expansão (T,t) e os valores dos vetores *pre*, *low* e *pai*. Um pré-processamento precisa ser efetuado antes da execução do *DFS*, dada uma aresta $\{s,t\}$ inicial, este pré-processamento precisa

garantir que, ao inicializar o procedimento com o vértice s, o primeiro vértice consultado na linha 5 deve ser necessariamente o vértice t para que a primeira aresta da árvore gerada seja a aresta $\{s, t\}$.

Além de prover a árvore (T, t), o algoritmo *DFS* ainda gera valores para três vetores utilizados nas próximas fases do algoritmo de *st*-numeração, sendo estes vetores:

- $pre(v)$ – vetor que armazena a ordem em que os vértices são visitados em pré-ordem;
- $pai(v)$ – vetor que armazena o valor pre do pai de cada vértice. Então, $pai(v) = pre(w)$ onde w é o vértice pai de v em T; e
- $low(v)$ – considerando todos os caminhos que partem de v (incluindo arestas da árvore e arestas de retorno) onde o vértice $pai(v)$ não aparece ou aparece apenas na extremidade, $low(v)$ é o valor do menor pre dentre todos os vértices de todos esses possíveis caminhos.

A Figura 2.23 apresenta um exemplo de um grafo G, em (a), e em (b) um possível resultado que o algoritmo *DFS* pode produzir ao tomar como aresta $\{s, t\}$ a aresta $\{v_3, v_4\}$. Outro detalhe é que para uma fácil compreensão, a ordem de visita dos vértices no exemplo está do vértice v_n de menor valor n para o de maior valor.



(a) *G*

(b) árvore de expansão de *G* e valores de *pre*(*v*) e *low*(*v*)

Figura 2.23. Exemplo de uma árvore de expansão gerada pelo DFS

Após a execução do procedimento *DFS*, o algoritmo de *st*-numeração proposto por

Even-Tarjan executa dois outros procedimentos, o procedimento principal *ETST-NUMBERING* e o procedimento *PATHFINDER*.

O procedimento *PATHFINDER* encontra, se existir, um caminho de arestas não visitadas até um vértice distinto w na árvore (T, t) a partir de um vértice $v \in (T, t)$. A Figura 2.24 apresenta o pseudocódigo do procedimento *PATHFINDER*.

```
   PATHFINDER
   Entrada: um vértice v
   Saída: caminho ainda não percorrido com início em v
   Pré-processamento: uma árvore T e valores de pre, low e pai, obtidos pelo DFS
 1: Início:
 2:     para todo w ∈ todos_vertices_adjacentes_ao_vertice(v) faça:
 3:         se new(v ↪ w) e w →* w então:
 4:             marque v ↪ w como old;
 5:             path := {v, w};
 6:         senão se new(v → w) então:
 7:             marque v → w como old;
 8:             path := {v, w};
 9:             enquanto new(w) faça:
10:                 procure (new(w ↪ x) e x = low(w)) ou
11:                 procure (new(w → x) e low(x) = low(w));
12:                 marque w e w → x como old;
13:                 path := path ∪ {w, x};
14:                 w := x;
15:         senão se new(v ↪ w) e v →* w então:
16:             marque v ↪ w como old;
17:             path := {v, w};
18:             enquanto new(w) faça:
19:                 procure (new(x → w));
20:                 marque w e x → w como old;
21:                 path := path ∪ {w, x};
22:                 w := x;
23:         senão path = ∅;
24:         se path ≠ ∅ então retorna path;
25:     retorna path;
26: Fim;
```

Figura 2.24. Pseudocódigo do procedimento PATHFINDER

A segunda fase do algoritmo de Even-Tarjan é responsável por *st*-numerar os vértices. Seja $G = (V, E)$ o grafo de entrada, $\{s, t\} \in E$ a aresta inicial e $P = \{\emptyset\}$ uma pilha de

vértices, o procedimento *ETST-NUMBERING* ao iniciar empilha em P os vértices t e s respectivamente, e então entra em um processo iterativo que se repete enquanto a pilha P não for vazia e que em cada iteração:

1. desempilha do topo de P um vértice v;
2. chama o procedimento *PATHFINDER* para o vértice v e atribui o resultado retornado à variável $path$;
3. se $path$ não for vazio, então os vértices que compõem o caminho são empilhados em P na ordem inversa, de maneira que o último vértice a ser empilhado seja o próprio v; ou
4. se $path$ for vazio, então a *st*-numeração é atribuída a v de acordo com um contador inicializado em 1 e incrementado sempre que um *st*-número é atribuído.

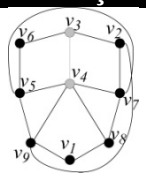
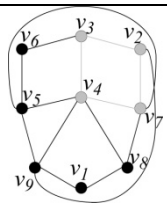
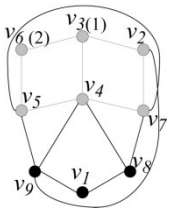
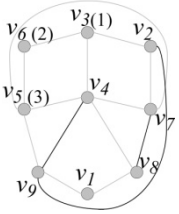
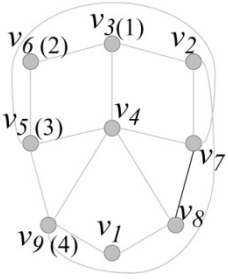
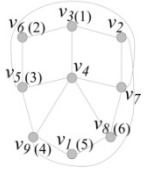
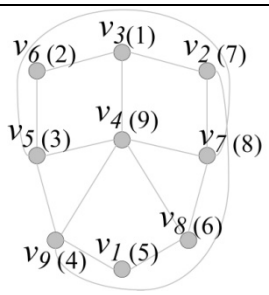
Dessa forma, o primeiro vértice a ser removido é sempre o vértice s, o qual é atribuído o *st*-número 1, e o último vértice a ser removido é o vértice t, que recebe o *st*-numero $|V|$, pois todos os outros vértices são empilhados e desempilhados antes de t. A Figura 2.25 apresenta o pseudocódigo do algoritmo de *st*-numeração proposto por Even e Tarjan.

A corretude do algoritmo pode ser encontrada nos trabalhos de Even e Tarjan (1976, 1977).

```
   ETST-NUMBERING
   Entrada: grafo biconexo G e aresta {s, t}
   Saída: vetor stnumber(v) que contém a st-numeração de cada vértice v ∈ G
   Pré-processamento: executar o procedimento DFS para G e {t, s}, marcar s, t e {s, t} como old
1: Início:
2:     P := empilha(t);
3:     P := empilha(s);
4:     stcounter := 0;
5:     enquanto P ≠ ∅ faça:
6:         v := desempilha(P);
7:         path := PATHFINDER(v);
8:         se path ≠ ∅ então:
9:             para j := k − 1 até 1 faça:
10:                P := empilha(path(v_j));
11:            senão:
12:                stcounter := stcounter + 1;
13:                stnumber(v) := stcounter;
14:    retorna stnumber;
15: Fim;
```

Figura 2.25. Procedimento de st-numeração por Even-Tarjan

*Tabela 2.2. Teste de execução para o algoritmo de st-numeração Even-*Tarjan *usando o grafo e árvore da Figura 2.23*

it	*P*	Path	Alterações	Marcação
pré	v_3, v_4	-	$old := \{v_3, v_4\}, v_3, v_4$	
1	v_3, v_4	v_3, v_2, v_7, v_4	$old := \{v_3, v_2\}, \{v_2, v_7\}, \{v_7, v_4\}, v_2, v_7$	
2	$v_3, v_2, v_7, v_4,$	v_3, v_6, v_5, v_4	$old := \{v_3, v_6\}, \{v_6, v_5\}, \{v_5, v_4\}, v_6, v_5$	
3	~~v_3~~, v_6, v_5, v_2 v_7, v_4	Ø	$old := \emptyset$ $stnumber(v_3) := 1$	
4	~~v_6~~, v_5, v_2, v_7 v_4	Ø	$old := \emptyset$ $stnumber(v_6) := 2$	
5	v_5, v_2, v_7, v_4	v_5, v_7	$old := \{v_5, v_7\}$	
6	v_5, v_2, v_7, v_4	v_5, v_9, v_1, v_8, v_4	$old := \{v_5, v_9\}, \{v_9, v_1\},$ $\{v_1, v_8\}, \{v_8, v_4\}, v_6, v_5$	
7	~~v_5~~, $v_9, v_1, v_8,$ v_2, v_7, v_4	Ø	$old := \emptyset$ $stnumber(v_5) := 3$	
8	$v_9, v_1, v_8,$ v_2, v_7, v_4	v_9, v_2	$old := \{v_9, v_2\}$	
9	$v_9, v_1, v_8,$ v_2, v_7, v_4	v_9, v_4	$old := \{v_9, v_4\}$	
10	~~v_9~~, $v_1, v_8,$ v_2, v_7, v_4	Ø	$old := \emptyset$ $stnumber(v_9) := 4$	
11	~~v_1~~, v_8, v_2, v_7 v_4	Ø	$old := \emptyset$ $stnumber(v_1) := 5$	
12	v_8, v_2, v_7 v_4	v_8, v_7	$old := \{v_8, v_7\}$	
13	~~v_8~~, v_2, v_7, v_4	Ø	$old := \emptyset$ $stnumber(v_8) := 6$	
14	~~v_2~~, v_7, v_4	Ø	$old := \emptyset$ $stnumber(v_2) := 7$	
15	~~v_7~~, v_4	Ø	$old := \emptyset$ $stnumber(v_7) := 8$	
16	~~v_4~~	Ø	$old := \emptyset$ $stnumber(v_4) := 9$	

A Tabela 2.2 ilustra a execução do algoritmo de Even-Tarjan para o grafo da Figura 2.23 (a) e a aresta $\{s, t\} = \{v_3, v_4\}$. O exemplo considera que o procedimento *DFS* já foi executado e que o resultado é o mesmo encontrado na Figura 2.23 (b). A coluna "*it*" representa a iteração atual do *loop* da linha cinco do pseudocódigo apresentado na Figura 2.25, a coluna *"P"* representa os vértices contidos na pilha onde o primeiro elemento está disposto no topo. Os vértices escritos em estilo itálico são aqueles que foram desempilhados e submetidos ao procedimento *PATHFINDER* e os vértices riscados são aqueles que foram desempilhados e tiveram sua *st*-numeração atribuída. A coluna "Path" apresenta o valor produzido pela chamada do procedimento *PATHFINDER* na *it*-ésima iteração. A coluna "Alterações" apresenta as alterações obtidas nas marcações de old e do vetor $stnumber$ ocorridas na *it*-ésima iteração, e finalmente a coluna "Marcação" apresenta uma representação do grafo com as arestas old em cor clara e as new em cor escura, alem disso ela exibe, quando disponível, o valor do *st*-número de cada vértice (entre parênteses).

2.3.2. Algoritmo de Tarjan

Ebert (1983) propôs um algoritmo para gerar uma *st*-numeração, e baseado neste algoritmo Tarjan (1986) projetou melhorias e criou um novo algoritmo. O algoritmo proposto por Tarjan possui complexidade de tempo linear, no entanto sua concepção é mais simples que o algoritmo de Even-Tarjan, por não necessitar do resultado da decomposição em caminhos efetuado pelo procedimento *PATHFINDER*.

O algoritmo de Tarjan também é executado em dois passos, sendo que o primeiro é a aplicação do mesmo procedimento *DFS* que gera a árvore de expansão e calcula os valores de pre, pai e low usado no algoritmo de Even-Tarjan.

No entanto as semelhanças de ambos os algoritmos se restringem a essa primeira fase, pois a segunda fase do algoritmo de Tarjan é diferente por construir uma lista L de modo que, ao final do processamento, L contém os vértices na exata ordem da *st*-numeração.

O procedimento rotula alternadamente cada vértice com um sinal (positivo ou negativo) começando com a raiz da árvore com sinal negativo. Os vértices da árvore são percorridos em pré-ordem, e para cada vértice *v* visitado se o sinal de $low(v)$ for positivo então *v* é adicionado em L depois de seu pai, o qual tem seu sinal marcado como negativo. Quando o sinal de $low(v)$ for negativo, v é adicionado em L antes de seu pai, o qual tem seu sinal marcado como positivo. A Figura 2.26 apresenta o pseudocódigo da segunda fase do algoritmo de Tarjan, denominada TARJAN*ST-NUMBERING*.

TARJAN*ST-NUMBERING*
Entrada: grafo biconexo G e aresta $\{s, t\}$
Saída: vetor $stnumber(v)$ que contém a *st*-numeração de cada vértice $v \in G$
Pré-processamento: $pre(s) := 1, count := 0$, executar o procedimento *DFS* para G e $\{t, s\}$

1: **Início:**
2: $L := s$;
3: $L := L \cup t$;
4: $sinal(s) := negativo$;
5: **para cada** $v \in preordem(T)$ **faça**:
6: **se** $sinal(low(v))$ for negativo **então**:
7: adicione v antes de $pai(v)$em L;
8: $sinal(pai(v)) := positivo$;
9: **senão se** $sinal(low(v))$ for positivo **então**:
10: adicione v depois de $pai(v) em$ L;
11: $sinal(pai(v)) := negativo$;
12: $count := 0$;
13: **para cada** $v \in L$ **faça**:
14: $count := count + 1$;
15: $stnumber(v) := count$;
16: **retorna** $stnumber$;
18: **Fim;**

Figura 2.26. Pseudocódigo do algoritmo de Tarjan para st-numeração

A Tabela 2.3 apresenta um exemplo de execução do algoritmo de Tarjan para o mesmo exemplo usado para ilustrar a execução (Tabela 2.2) do algoritmo de Even-Tarjan, com a diferença que a aresta $\{s, t\}$ usada é a aresta $\{v_4, v_3\}$. O grafo tratado e a árvore gerada pelo *DFS* são os mesmos da Figura 2.23. A coluna "*it*" representa a iteração atual do *loop* da linha 5 do procedimento TARJAN*ST-NUMBERING* apresentado na Figura 2.26, onde o valor "pré" é referente à pré-execução do *loop*. A coluna "v" representa o vértice atual da *it*-ésima iteração, enquanto a coluna L representa o conteúdo atual da lista L e a coluna "*low*(*v*)" mostra o vértice atribuído ao $low(v)$, e não a seu valor em si. Finalmente a coluna "Vetor sinal" apresenta o sinal de cada vértice. A Figura 2.27 apresenta o resultado final com a *st*-numeração obtida entre parênteses.

A prova da corretude do algoritmo pode ser encontrada no próprio trabalho de Tarjan (1986). Alguns outros métodos foram propostos posteriormente para se *st*-numerar um grafo, para maiores informações recomendamos os trabalhos de Papamanthou (2005) e Papamanthou e Tollis (2008).

Tabela 2.3. Teste de execução do método de Tarjan para st-numeração usando grafo e árvore da Figura 2.23

it	*v*	*L*	*low*(*v*)	Vetor sinal
pré	-	$v_4 v_3$	-	$(v_1)(v_2)(v_3 -)(v_4)(v_5)(v_6)(v_7)(v_8)(v_9)$
1	v_2	$v_4\ v_2\ v_3$	v_3	$(v_1)(v_2)(v_3 -)(v_4 +)(v_5)(v_6)(v_7)(v_8)(v_9)$
2	v_7	$v_4 v_7 v_2 v_3$	v_3	$(v_1)(v_2 +)(v_3 -)(v_4 +)(v_5)(v_6)(v_7)(v_8)(v_9)$
3	v_5	$v_4 v_5 v_7 v_2 v_3$	v_3	$(v_1)(v_2 +)(v_3 -)(v_4 +)(v_5)(v_6)(v_7 +)(v_8)(v_9)$
4	v_6	$v_4 v_5 v_6 v_7 v_2 v_3$	v_4	$(v_1)(v_2 +)(v_3 -)(v_4 +)(v_5 -)(v_6)(v_7 +)(v_8)(v_9)$
5	v_9	$v_4 v_9 v_5 v_6 v_7 v_2 v_3$	v_3	$(v_1)(v_2 +)(v_3 -)(v_4 +)(v_5 +)(v_6)(v_7 +)(v_8)(v_9)$
6	v_1	$v_4 v_1 v_9 v_5 v_6 v_7 v_2 v_3$	v_3	$(v_1)(v_2 +)(v_3 -)(v_4 +)(v_5 +)(v_6)(v_7 +)(v_8)(v_9 +)$
7	v_8	$v_4 v_8 v_1 v_9 v_5 v_6 v_7 v_2 v_3$	v_3	$(v_1 +)(v_2 +)(v_3 -)(v_4 +)(v_5 +)(v_6)(v_7 +)(v_8)(v_9 +)$

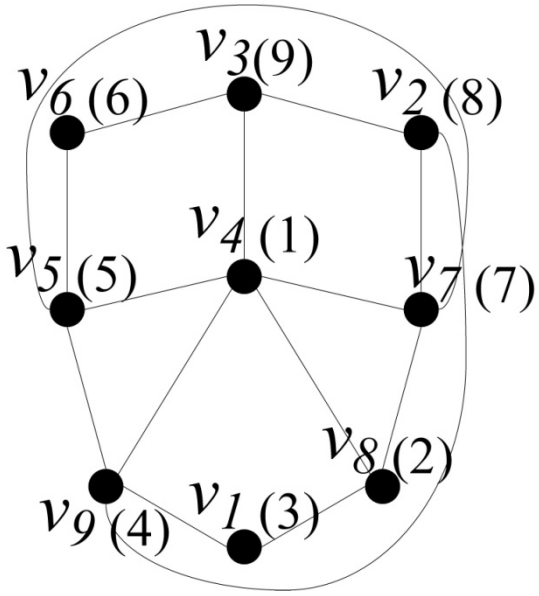


Figura 2.27. Grafo do exemplo com sua st-numeração atribuída (entre parênteses) pelo método de Tarjan

O algoritmo estudado neste trabalho, o *VD-PLANARIZE* utiliza uma *st*-numeração para planarizar um grafo. Esse algoritmo é baseado no algoritmo de Lempel, Even e Cederbaum (1967), o qual utiliza uma estrutura de dados árvore-*PQ*.

2.4. Árvores-*PQ*

Uma árvore-*PQ* (do inglês *Priority Queue Tree)* é uma estrutura de dados que representa os possíveis conjuntos de permutações em um conjunto *S* qualquer. Concebida por Booth e

Lueker (1976), elas são usadas para se resolver problemas onde o objetivo é encontrar uma ordenação que satisfaça varias restrições. Nesses problemas, restrições na ordenação são inclusas uma de cada vez modificando a estrutura da árvore-*PQ* de tal maneira ela represente somente ordenações que satisfaçam as condições. Dentre suas aplicações, destacam-se o teste de matrizes em busca de propriedades consecutivas, reconhecimento de grafos intervalos e testes de planaridade de grafos.

Dada uma árvore-*PQ* T formada pelos elementos de S, seus nós podem ser:

- folhas, representando os elementos de S;
- nós P, são nós onde os filhos podem ser livremente permutados, convencionalmente representados por círculos; e
- nós Q, são nós onde os filhos podem ter sua ordem apenas invertida, convencionalmente representado por retângulos. Em testes de planaridade nós Q representam uma componente biconexa planar do grafo.

A Figura 2.28 ilustra uma árvore-*PQ* derivada de um conjunto S que contêm cinco elementos.

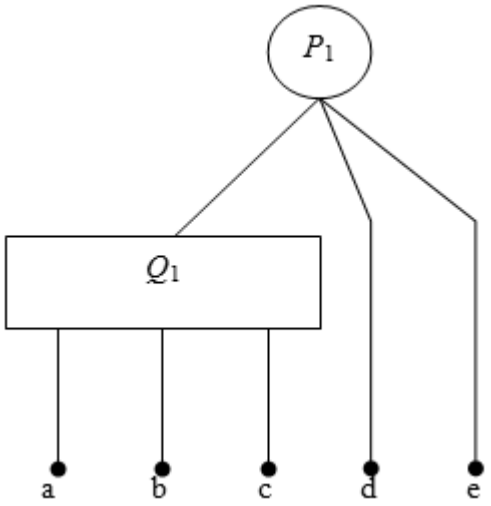


Figura 2.28. Exemplo de árvore-PQ

Seja T uma árvore-*PQ*. Define-se como uma *fronteira* de T o acesso seqüencial de suas folhas, para uma permutação qualquer dos seus nós P e Q. A Figura 2.29 mostra todas as fronteiras possíveis da árvore ilustrada na Figura 2.28, bem como as combinações existentes para diferentes arranjos dos nós P_1 e Q_1 do exemplo.

Uma árvore-*PQ* admite uma operação especial que manipula seus nós de tal maneira que a árvore aceite, quando possível, uma nova restrição. Esta operação é denominada *redução*. Seja T uma árvore-*PQ* formada por elementos de um conjunto S, a operação de

redução recebe como entrada um subconjunto $S' \subseteq S$ e rearranja os nós de T, possivelmente adicionando novos nós P ou Q, de tal forma que os elementos de S' apareçam consecutivamente em todas as fronteiras de T. Os elementos do conjunto S' são denominados *pertinentes*.

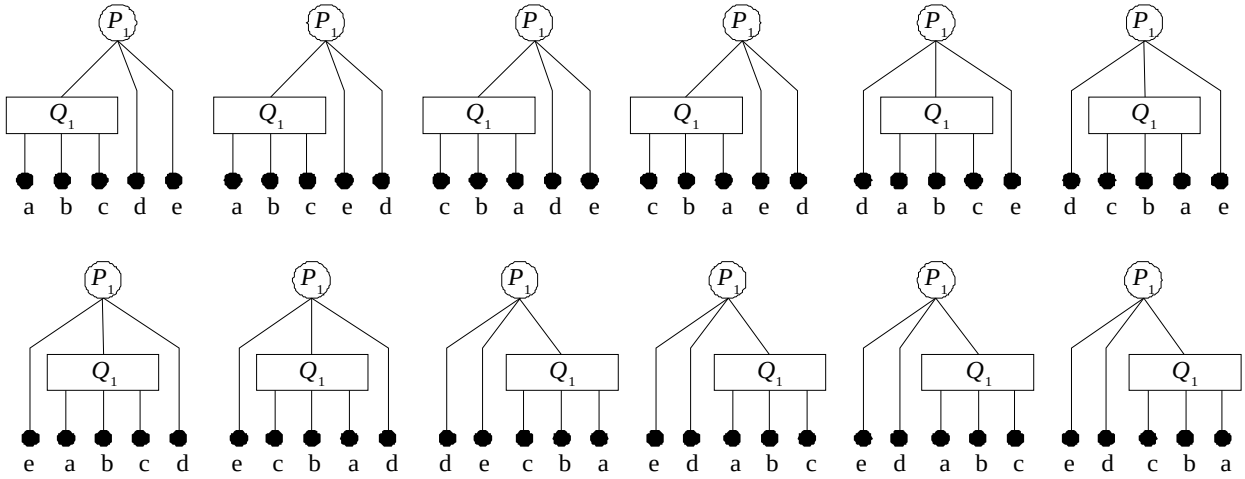


Figura 2.29. Fronteiras possíveis da árvore-PQ da Figura 2.28

2.4.1. Operação de Redução

Seja T uma árvore-*PQ* formada com elementos do conjunto S e $S' \subseteq S$ um subconjunto de elementos pertinentes. A operação de redução efetua uma série de padrões de troca de nós – *template matchings* – em todos os elementos de S' e seus pais até o nó ancestral comum de nível mais baixo possível. Esse ancestral é chamado de *raiz da subárvore pertinente*. Nós-*PQ* pertinentes são denominados *cheios*, quando todos os seus filhos são pertinentes ou *parciais* quando ele possui pelo menos um filho pertinente e um não pertinente. Nós-*PQ* não pertinentes são denominados *vazios*.

A operação de redução é dividida em duas fases, a fase *bubble* e a fase *reduce*. Na fase *bubble*, é identificada a subárvore pertinente na qual a operação de redução será processada. Na fase *reduce* uma série de operações de padrões de trocas de nós são efetuadas sobre os nós *P's* e *Q's* de maneira que o resultado final produzido contemple todas as folhas pertinentes arranjadas consecutivamente em sua fronteira.

Cada *padrão de troca* de nós promove um conjunto de permutações de nós na árvore de tal forma que os respectivos nós pertinentes sejam arranjados seqüencialmente na fronteira. Os padrões para as folhas são simples, no entanto os padrões para os nós P e Q são bem mais complexos e numerosos.

Para mais informações sobre a operação de redução e os padrões de troca, recomenda-

se leitura dos trabalhos de Vollen (1998) e Harris (2002).

O algoritmo de Ozawa e Takahashi realiza o teste de planaridade de um grafo utilizando *árvores-PQ*. O algoritmo *VD-PLANARIZE* utiliza esse teste para decidir quais são os nós que devem ser removidos do grafo para torná-lo planar.

2.5. Heurísticas e Metaheurísticas

Para se resolver um problema computacional normalmente existe duas preocupações:

- elaborar um algoritmo de resolução que tenha um tempo computacional aceitável; e
- elaborar um algoritmo que forneça a melhor solução possível para o problema.

No entanto, para problemas complexos do mundo real nem sempre é possível atingir essas duas metas e se faz necessário optar por uma delas.

Propor algoritmos que forneçam a melhor solução possível para um problema é o objetivo de qualquer projetista de algoritmos, no entanto prover a solução ótima nem sempre é a melhor opção. Dentro da teoria da computação, existe a subárea *teoria da complexidade computacional* que foca em classificar problemas de acordo com as suas dificuldades inerentes. Além disso, a teoria da complexidade computacional trata de problemas de decisão, entretanto é trivial demonstrar que todo problema que não é de decisão pode ter um problema de decisão associado. Seja X um problema qualquer, e A uma solução válida para X, podemos associar a X o seguinte problema de decisão $Y =$ "A é a melhor solução possível para o problema X?".

Os problemas de decisão associados podem ser classificados de duas maneiras:

- problemas da classe *P* (*deterministic Polynomial*) - podem ser resolvidos por uma máquina determinística em tempo polinomial;
- problemas da classe *NP* (*nondeterministic Polynomial*) – podem ser resolvidos em tempo polinomial apenas por uma máquina de Turing não determinística. Máquinas determinísticas (como os computadores) não podem resolver esses problemas em tempo polinomial.

Para mais informações acerca da teoria da *NP*-completude e problemas pertencentes à classe *NP*, recomenda-se a leitura do trabalho de Garey e Johnson (1979).

Os *algoritmos aproximativos* constituem uma abordagem adequada para problemas da classe *NP* por fornecerem soluções dentro de um limite de qualidade absoluto ou assintótico,

assim como um limite assintótico de complexidade polinomial para o pior caso comprovado matematicamente, dessa forma, conforme o tempo de execução de um método aproximativo cresce, as soluções geradas tendem a se aproximarem de uma solução ótima.

Quando um problema da classe *NP* precisa ser resolvido em tempo restrito, e esse não admite um algoritmo aproximativo, comumente é proposta uma heurística. Um algoritmo *heurístico* (do grego *heuriskein*, descobrir) é uma regra, simplificação ou aproximação que reduz ou limita a busca em domínios de soluções muito extensos e inviáveis de serem integralmente investigados. De modo simples, uma heurística é um método de busca restrita no espaço de soluções que procura por uma boa solução, mas não garante que a solução encontrada seja ótima. Dependendo de quão restrito é o espaço vasculhado, os métodos heurísticos são mais ou menos rápidos.

Uma *metaheurística* é um modelo heurístico genérico para resolver problemas de otimização. Elas se destacam por apresentarem uma metodologia de busca eficiente para diferentes espaços de soluções. Dentre as metaheurísticas mais comuns podemos destacar os algoritmos genéticos, *simulated annealing*, *GRASP*, *VNS*, *VND*, busca tabu e *ant system*.

Para maiores informações acerca de metaheurísticas recomendamos o trabalho de Glover e Kochenberger (2003).

Este trabalho propõe o uso de dois algoritmos heurísticos para solucionar o problema de planarização de grafos utilizando a operação de remoção de vértices. Como visto na seção 2.2.2, encontrar o valor do número de remoção de vértices de um grafo G possui um problema de decisão associado *NP*-completo. Portanto esse problema não pode ser resolvido em tempo polinomial. Além disso, foi provado (Faria et al, 2006) que o problema tratado não admite soluções por meio de um algoritmo aproximativo, dessa forma justificando o uso de heurísticas.

2.6. Algoritmos Genéticos

Um algoritmo genético (*GA – Genetic Algorithm*) é uma metaheurística utilizada para resolver problemas de otimização. Algoritmos genéticos são pertencentes a uma classe particular de algoritmos denominados *algoritmos evolutivos* que usam técnicas inspiradas pela biologia evolutiva (hereditariedade, mutação, seleção natural e recombinação).

Os algoritmos genéticos apresentam algumas vantagens em relação a outras técnicas tradicionais de otimização. A primeira vantagem notável é lidar com um conjunto de soluções e não apenas com uma única solução, por ter a capacidade de cobrir diferentes áreas do

espaço de soluções simultaneamente. Outra vantagem é que usam transições probabilísticas e não determinísticas, desta forma para diferentes execuções pode-se obter diferentes resultados. Algoritmos genéticos não necessitam de conhecimento derivado do problema, mas apenas uma forma de avaliar os resultados.

Um algoritmo genético é constituído basicamente de quatro elementos:

- o *indivíduo*, elemento que contém o *código-genético*. Incluso no indivíduo está uma possível solução do problema de otimização. O *cromossomo* é a parte específica do indivíduo que representa computacionalmente uma solução do problema;
- a *função-objetivo* utilizada, objeto da otimização. Essa função deve avaliar uma solução proposta em função das entradas e fornecer um parâmetro de comparação que permita decidir se a solução proposta é boa ou ruim;
- o processo de *seleção*, o qual define como a população é organizada e atualizada, como os indivíduos são selecionados para cruzamento e se existe ou não alguma escolha elitista; e
- a *reprodução*, que constitui a fase que define como, uma vez selecionados dois indivíduos, o cruzamento entre eles vai gerar novos indivíduos. Durante a reprodução podem ocorrer também mutações.

2.6.1. Indivíduo

O indivíduo é a unidade fundamental de um algoritmo genético por conter as possíveis soluções do problema. É por meio dos indivíduos que as soluções são otimizadas. Um indivíduo possui um cromossomo, estrutura que normalmente armazena uma solução do problema de otimização. Cada elemento desse cromossomo é denominado gene. Define-se de genótipo como sendo um conjunto de genes. O genótipo é a informação interpretada pela máquina. Define-se fenótipo como sendo o significado interpretado do conjunto de genes por humanos.

A Tabela 2.4 apresenta exemplos de diferentes cromossomos para diferentes problemas, incluindo o genótipo e o fenótipo.

Tabela 2.4. Exemplo de cromossomos para diferentes problemas

Genótipo	Fenótipo	Problema
1101010101101100	54636	otimização numérica
BACDEGF	comece pela cidade B, depois passe pelas cidades A, C, D, E, G e termine em F	caixeiro viajante
C1R3C5R2	Se condição 1 execute regra 3, se condição 5 execute regra 2	regras para aprendizagem de agentes

2.6.2. População

A *população* de um algoritmo genético é o conjunto de indivíduos que existem durante uma iteração do algoritmo. O tamanho da população é normalmente fixo e os indivíduos normalmente substituídos a cada iteração.

A *função-objetivo* $f_O(x)$ é a função que recebe um indivíduo x e retorna um valor correspondente à aptidão de x. Comumente a função-objetivo é uma mesura do problema a ser resolvido, por exemplo, em um problema de otimização numérica onde o objetivo é encontrar o menor valor, dado um indivíduo x, $f_O(x)$ retorna o valor correspondente a x.

2.6.3. Seleção

A *seleção* de um algoritmo genético consiste um processo onde certos critérios são satisfeitos para selecionar os pais que gerarão a próxima população. A priori se faz necessário definir quantos indivíduos são gerados pelo cruzamento de dois outros quaisquer, e com base nisso definir quantos pares de indivíduos serão selecionados para a fase de cruzamento. Normalmente o cruzamento entre dois indivíduos geram outros dois indivíduos.

O meio mais simples de se compor cada par de indivíduos para o cruzamento é por meio de sorteio aleatório. No entanto este método de seleção não simula a seleção natural, onde o individuo mais apto tende a sobreviver e o menos apto a morrer, além disso, selecionar ao acaso é uma abordagem que raramente é usada por não gerar bons resultados. Baseado no método de sorteio aleatório desenvolveu-se sistemas probabilísticos que consistem em utilizar a informação provida pela função-objetivo $f_O(x)$ para determinar quão apto é um individuo no sistema, portanto quanto melhor a avaliação da função-objetivo $f_O(x)$, mais apto o indivíduo é para ser escolhido.

Goldberg (1989) sugere o uso de uma técnica denominada *scaling* para aprimorar o

método de avaliação da adequação de cada individuo. Sem *scaling*, o valor bruto da função-objetivo é usado como parâmetro de probabilidade de seleção, no entanto nem sempre proporciona bons resultados. Dada uma função objetivo $f_O(x)$, uma função *scaling* $f_{scaling}(f_O(x))$ pode ser do tipo:

- **linear** - o valor da função-objetivo $f_O(x)$ recebe a seguinte alteração:

$$f_{scaling}(f_O(x)) = f_O(x) \times a + b,$$

onde

$$\begin{cases} a = (c-1) \times \dfrac{med}{\Delta} \text{ e } \quad b = med \times \dfrac{max - c \times med}{\Delta}, se\ \Delta \neq 0 \\ a = 1, b = 0, \text{caso contrário.} \end{cases}$$

A variável max representa o maior valor de adaptação encontrado, med a média de todas as soluções da população atual, c é uma constante real maior que 1 denominada *fator escala* (*scale factor*) e Δ é o maior valor de adaptação encontrado na população atual menos o valor $f_O(x)$.

- **truncamento de sigma** – um valor múltiplo do desvio médio d_m é subtraído dos valores obtidos pela função-objetivo e os valores negativos são ajustados para zero. A fórmula resultante é a seguinte:

$$f_{scaling}(f_O(x)) = f_O(x) - med - c \times d_m,$$

onde c é uma constante real maior que 1.

- **potência** – o valor da função-objetivo é elevado a uma constante pré-definida k:

$$f_{scaling}(f_O(x)) = f_O(x)^k$$

Em sistemas onde o *scaling* é adotado, a função f_O é substituída pela função $f_{scaling}$ para cálculo de aptidão dos indivíduos.

Depois de decidido como a função-objetivo qualifica cada individuo é que se define como o processo de seleção deve ser feito. Existem cinco métodos comumente utilizados para fazer a seleção que são:

- **ranking** – os indivíduos da população são ordenados de acordo com seu valor de adequação e então sua probabilidade de seleção é atribuída conforme a posição que ele ocupa no ranking;
- **roleta** –cada indivíduo recebe uma probabilidade e um valor aleatório é sorteado. Dada uma população composta de n indivíduos $I_1, I_2, \ldots, I_n$. Seja

$$S(k) = \sum_{j=1}^{k} f_O(I_j),$$

o somatório dos valores de adequação dos indivíduos da população e r um valor sorteado aleatoriamente entre 0 e $S(n)$. Seja i um indivíduo da população, i será selecionado pelo processo da roleta se e somente se $S(i-1) \leq r \leq S(i)$;

- **torneio** – grupos de soluções são arbitrariamente escolhidos e os indivíduos mais adaptados entre eles são selecionados para o cruzamento; e
- **seleção estocástica do remanescente** – dada a formula $s(x) = \frac{fo(x)}{med}$, onde med é a média do valor de adaptação da população, assume-se que a parte inteira da função determina quantas cópias do indivíduo são selecionadas diretamente. Os outros indivíduos são escolhidos aplicando outro método de seleção sobre a parte decimal do valor $s(x)$ de cada indivíduo.

2.6.4. Cruzamento

Depois de selecionados os pares, o *cruzamento* (*crossover*) tem início. Nesta fase os genes dos indivíduos *pais* são combinados para gerarem o(s) *filho*(*s*).

A fase de cruzamento é totalmente dependente da representação de solução escolhida para o problema. Uma má representação pode fazer com que o algoritmo percorra um espaço muito limitado de soluções, conseqüentemente não gerando boas melhorias.

Dado dois indivíduos pais I_p^1 e I_p^2 os quais tem cromossomos de comprimento fixo l, que gerarão dois filhos I_f^1 e I_f^2, os principais tipos de cruzamento são:

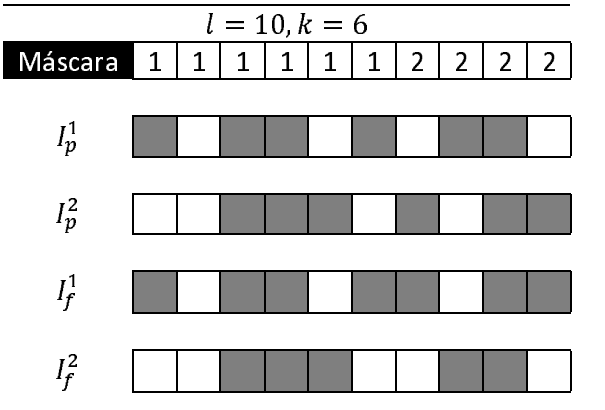


Figura 2.30. Exemplo de cruzamento de um ponto

- **um ponto** – nesse cruzamento, divide-se em dois os genes dos pais e os filhos recebem uma parte de cada pai. O processo sorteia um número k entre 1 e l (*ponto de corte*) onde I_f^1 recebe os genes de 1 a k de I_p^1 e $k+1$ a l de I_p^2 enquanto I_f^2 recebe os genes de 1 a k de I_p^2 e $k+1$ a l de I_p^1. A Figura 2.30 exemplifica a

situação de cruzamento de um ponto para o valor sorteado $k = 6$. Para melhor compreensão, a figura utiliza uma *máscara* que contém valores 1 ou 2 que indicam se o indivíduo I_f^1 herdará aquele gene de I_p^1 ou I_p^2 respectivamente. A máscara do indivíduo I_f^2 é a máscara complementar à exibida;

- **multiponto** – cruzamento baseado no cruzamento de um ponto, mas neste modelo um número n de pontos de cortes é pré-definido. Os locais dos cortes continuam sendo sorteados aleatoriamente, assim como no cruzamento de um ponto;

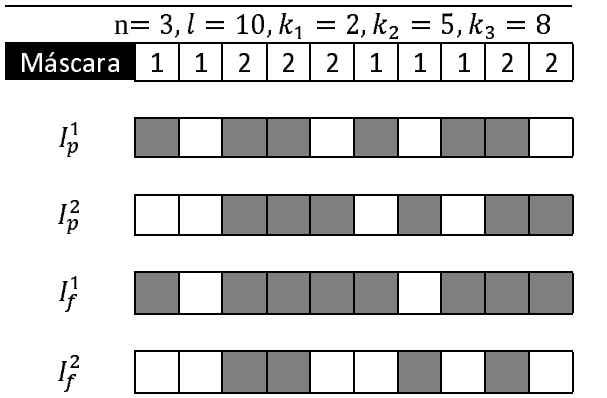


Figura 2.31.Exemplo de cruzamento multiponto

- **segmentado** – cruzamento semelhante ao multiponto, no entanto, cada vez que dois pais se cruzam é sorteado um novo valor para n;

Figura 2.32. Exemplo de cruzamento uniforme

- **uniforme** – para cada gene de I_f^k, o método do cruzamento uniforme consiste em sortear o pai que transmitirá o gene; e
- **combinação parcial** – sorteia-se dois pontos de corte. O indivíduo I_f^1 recebe todos os genes de I_p^1 para o espaço entre os pontos de corte, o filho I_f^2 recebe todos os

genes de I_p^2 para o mesmo espaço. O restante dos genes é preenchido com os valores considerados mais apropriados para cada I_f^k.

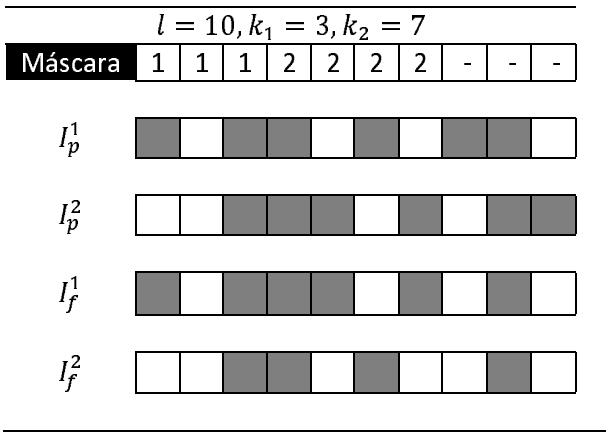


Figura 2.33. Exemplo de cruzamento por combinação parcial

Depois de gerado cada novo indivíduo, existe uma pequena chance de ele sofrer uma *mutação*, alguma alteração fora dos padrões em seus genes. O algoritmo genético pode ser pré-configurado para possibilitar a mutação, que normalmente não supera 5% dos indivíduos gerados. Se o indivíduo for selecionado para mutação, ela ocorrerá normalmente por um dos seguintes mecanismos:

- ***flip*** – cada gene a ser mutado tem seu valor alterado para qualquer valor possível do alfabeto valido;
- **troca** – dois genes são sorteados e tem seus valores trocados entre si; e
- ***creep*** – um valor aleatório é somado ou subtraído do gene.

Ao fim do processo de cruzamento, os novos indivíduos substituem os antigos na população e o processo reinicia. Algumas abordagens distintas são utilizadas nos processos de substituição da população das quais se destacam as abordagens:

- **clássica** – nessa abordagem todos os indivíduos de uma população são substituídos pelos novos indivíduos gerados pelo processo de seleção;
- **mais apto** – neste método, os indivíduos gerados só substituem os geradores se estes forem mais aptos; e
- **elitismo** – um conjunto das *n* melhores soluções de uma geração é sempre mantido, enquanto o restante da população é substituída segundo algum outro critério.

Para cada nova população gerada, o processo de cruzamento se repete até que o algoritmo atinja uma condição de parada. Essa condição pode ser uma quantidade fixa de iterações, um tempo fixo de processamento ou por *estagnação* (*starvation*), onde após n

iterações (ou período de tempo) sem se obter uma melhora na solução a execução do algoritmo é encerrada.

A Figura 2.34 apresenta o pseudocódigo de um algoritmo genético genérico. Os algoritmos genéticos são largamente utilizados pela sua simplicidade de implementação e sua reconhecida capacidade e explorar diversas regiões distintas do espaço de soluções gerando para muitos problemas bons resultados. Para mais informações acerca de algoritmos genéticos recomendamos a leitura dos trabalhos de Mitchell e Forrest (1995) e Mitchell (1998).

```
ALGORITMO GENÉTICO
Entrada: população, função-objetivo f_o
Saída: indivíduo
Pré-processamento: gerar uma população válida
1: Início:
2:     enquanto a condição de parada não for satisfeita:
3:         melhor indivíduo := identifica melhor individuo(população);
4:         lista de pais := seleção(população, f_o);
5:         população := reprodução(lista de pais);
6:     retorna melhor indivíduo
7: Fim;
```

Figura 2.34. Pseudocódigo de um algoritmo genético

2.7. Trabalhos Relacionados

Encontramos na literatura diversos trabalhos que tratam do problema de planarização de grafos e invariantes de planaridade. Destacam-se os trabalhos de Jayakumar, Thulasiraman e Swamy (1989) que desenvolveram o algoritmo *PLANARIZE* que utiliza a operação de remoção de arestas para planarizar um grafo. Outro trabalho considerado foi desenvolvido por Eades e Mendonça (1993), que utiliza o algoritmo *PLANARIZE* adaptado para utilizar a operação de divisão de vértices. Ambos possuem complexidade de tempo $O(n^2)$.

Outros trabalhos destacados são: Ozawa e Takahashi (1981) que, por remoção de arestas, desenvolveram um algoritmo de complexidade de tempo $O(n^{1,5})$ e Kant (1992) que propôs outro algoritmo de planarização também $O(n^2)$. Outros algoritmos que utilizam a remoção de arestas para planarizar um grafo foram propostos por: Fisher e Wing (1966), Pasedach (1976), Sadowska (1978) e Chiba, Nishioka e Shirizawa (1979).

CAPÍTULO

3

Algoritmos Propostos

Este trabalho propõe dois algoritmos heurísticos para resolver o problema de planarização de grafos utilizando a operação de remoção de vértices. O primeiro algoritmo, denominado *VD-PLANARIZE* é uma heurística que foi primeiro descrita na tese de doutorado de Mendonça (1994). O segundo algoritmo proposto, o *GAVD-PLANARIZE* utiliza uma metaheurística genética para melhorar as soluções obtidas pelo *VD-PLANARIZE*.

3.1. *VD-PLANARIZE*

Jayakumar, Thulasiraman e Swamy (1989) propuseram um algoritmo para planarizar grafos que utiliza a operação de remoção de arestas denominado *PLANARIZE*. Eades e Mendonça (1993) adaptaram o *PLANARIZE* para utilizar a operação de divisão de vértices e assim criaram o *SPLIT-PLANARIZE*. Ambos os algoritmos possuem complexidade de tempo $O(n^2)$ considerando *n* o tamanho do conjunto de vértices do grafo de entrada.

O algoritmo *PLANARIZE* se baseia no teste de planaridade de Lempel, Even e Cederbaum (1967) que utiliza árvores-*PQ* (Booth e Lueker, 1976) na sua implementação. Este trabalho adapta o algoritmo *PLANARIZE* para a operação de remoção de vértices, denominando este novo algoritmo de *VD-PLANARIZE* e fazendo as devidas modificações para que este possuísse complexidade de tempo linear.

O algoritmo de Lempel, Even e Cederbaum (algoritmo de *LEC*) realiza um teste num

grafo para determinar se ele é ou não planar. Esse algoritmo trata somente de grafos biconexos. Não obstante, é possível separar em tempo linear um grafo conexo e não biconexo em componentes biconexos, como mostra Gibbons (1994). Desta forma o algoritmo *VD-PLANARIZE* pode ser aplicado em qualquer grafo conexo, no entanto neste trabalho utilizaremos apenas grafos biconexos a fim de evitar a etapa de divisão em componentes biconexos. O algoritmo de *LEC* requer um *st*-grafo, um grafo que possui uma *st*-numeração.

Dado um *st*-grafo G, seja G_k, para $1 \leq k \leq n$, um subgrafo induzido pelo conjunto de vértices $V_k = \{1, 2, \ldots, k\}$, seja B_k um grafo isomorfo ao subgrafo G_k mais as arestas de G que são incidentes ao conjunto de vértices V_k e não estão em G_k, denominadas *arestas virtuais*, onde cada aresta virtual é adjacente a um vértice de V_k e a um novo *vértice virtual* que é rotulado com o seu equivalente em G. Denominamos o desenho de B_k por *arbusto*. Em um arbusto, os vértices de menor rótulo aparecem em um nível mais elevado e todos os vértices virtuais aparecem no último nível. Além disso, é comum que existam vários vértices virtuais com o mesmo rótulo. A Figura 3.1 apresenta em (a) um grafo $K_{6,3}$ com uma *st*-numeração e em (b) um arbusto B_5 deste grafo.

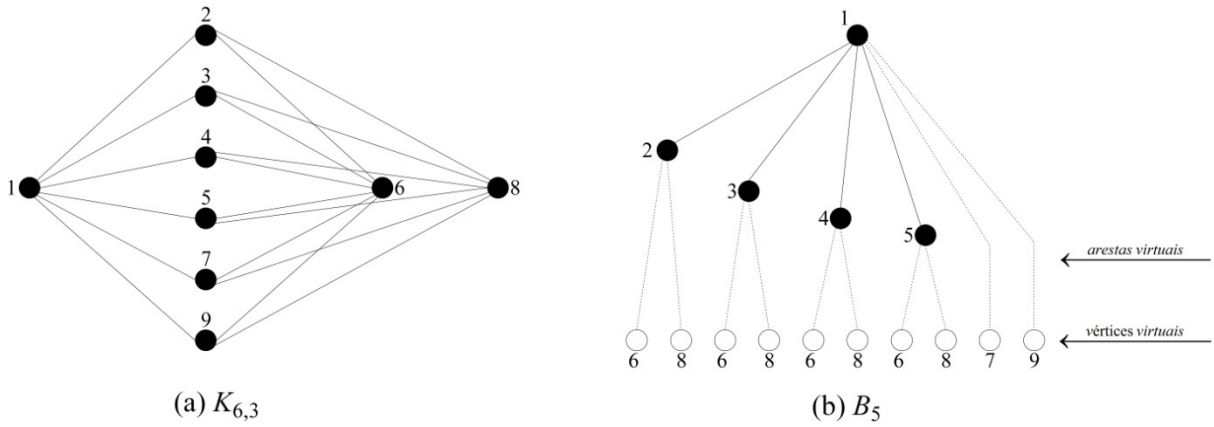


Figura 3.1. Um st-grafo e um arbusto B_5

Um *st*-grafo é planar se, e somente se, para cada arbusto B_k, $2 \leq k \leq n-2$ existir um grafo B'_k isomorfo a B_k tal que todo os vértices virtuais rotulados $k+1$ apareçam consecutivamente (Even e Tarjan, 1976).

O algoritmo de *LEC* se baseia nessa propriedade para averiguar a planaridade de um grafo. Ele inicia seu processamento com o arbusto B_2 e para cada arbusto B_k, é verificada a possibilidade de arranjar seus vértices virtuais de maneira consecutiva. Caso seja possível, ele avança para a próxima iteração com o arbusto B_{k+1}, caso não seja possível ele acusa que o grafo não é planar. A fim de realizar com eficiência essas verificações, o algoritmo de *LEC* utiliza árvores-*PQ* para representar os arbustos. As propriedades da árvore-*PQ* garantem a

validade dos arbustos e a operação de redução garante a verificação da seqüência de vértices virtuais (Lempel, Even e Cederbaum, 1967).

O algoritmo de *LEC* inicia em B_2 pois B_1 é trivialmente redutível, visto que dentre seus vértices virtuais não haverá vértices com rótulos iguais. O algoritmo termina com a iteração do arbusto B_{n-2} pois é também trivial provar que o arbusto B_{n-1} é facilmente redutível para o conjunto de vértices virtuais de rótulos iguais a n, pois todas as suas folhas possuem esse rótulo.

O algoritmo *PLANARIZE* utiliza o algoritmo de *LEC* para planarizar um grafo removendo arestas. Assim como o algoritmo de *LEC*, ele monta os arbustos e quando a não-planaridade é detectada, ele escolhe uma ou mais arestas a serem removidas, mantendo a propriedade de planaridade citada anteriormente. No entanto, o algoritmo de *LEC* detecta que uma redução não é possível durante o próprio processo de redução, dessa forma quando ele acusa a não-planaridade a estrutura final da árvore-*PQ* está alterada de forma que ela não mais representa o arbusto B_k. Reverter esse processo de redução é muito custoso, assim como criar uma cópia da árvore a cada iteração, por isso o *PLANARIZE* utiliza um teste que detecta se a árvore-*PQ* é redutível sem alterar sua estrutura antes de efetuar a redução.

O teste que detecta se uma árvore-*PQ* é passível de redução sem alterar sua estrutura foi proposto por Ozawa e Takahashi (1981). Esse teste toma uma árvore-*PQ* T_k e um conjunto de vértices virtuais s e classifica os nós da subárvore pertinente a fim de averiguar se uma redução é ou não possível. Um nó X *P* ou *Q* qualquer pode ser classificado como:

- **tipo *A*:** um nó X é do tipo *A* se a subárvore enraizada até X puder ser rearranjada tal que todas as folhas pertinentes descendentes de X apareçam consecutivamente no meio da fronteira com pelo menos uma folha não pertinente de cada ponta da fronteira;
- **tipo *B*:** um nó X é do tipo *B* se a fronteira da subárvore enraizada até X consistir somente de folhas pertinentes. Em outras palavras, X é um nó cheio;
- **tipo *H*:** um nó X é do tipo *H* se a subárvore enraizada até X puder ser rearranjada tal que todas as folhas pertinentes descendentes de X apareçam consecutivamente em uma das extremidades da fronteira;
- **tipo *V*:** um nó X é do tipo *V* se a subárvore enraizada até X pode ser rearranjada tal que todas as folhas não pertinentes descendentes de X apareçam consecutivamente no meio da fronteira com pelo menos uma folha pertinente aparecendo em cada extremidade da fronteira; e

- **tipo *W*:** um nó X é dito ser do tipo W se a fronteira da subárvore enraizada até X consistir somente de folhas não pertinentes. Em outras palavras, X é um nó vazio.

A Figura 3.2 apresenta um exemplo de uma árvore-*PQ* com seus nós classificados de acordo com o teste de Ozawa e Takahashi. Os nós virtuais que compõem o conjunto S' de nós a serem reduzidos são os nós pretos, as letras ao redor de cada nó P e Q apresentam as classificações dos nós. Perceba que as classes de nós não são mutuamente exclusivas, é possível que um nó seja de mais de um tipo.

Classificados os nós, o algoritmo verifica se o tipo da raiz da subárvore pertinente é do tipo *B*, *H* ou *A*, indicando que a redução é possível, caso contrário a redução não é possível. Observe na Figura 3.2 que a raiz da subárvore pertinente, o nó P_2, é do tipo *H* portanto a árvore é redutível, facilmente constatado se invertermos as folhas *9* e *8* do nó P_2.

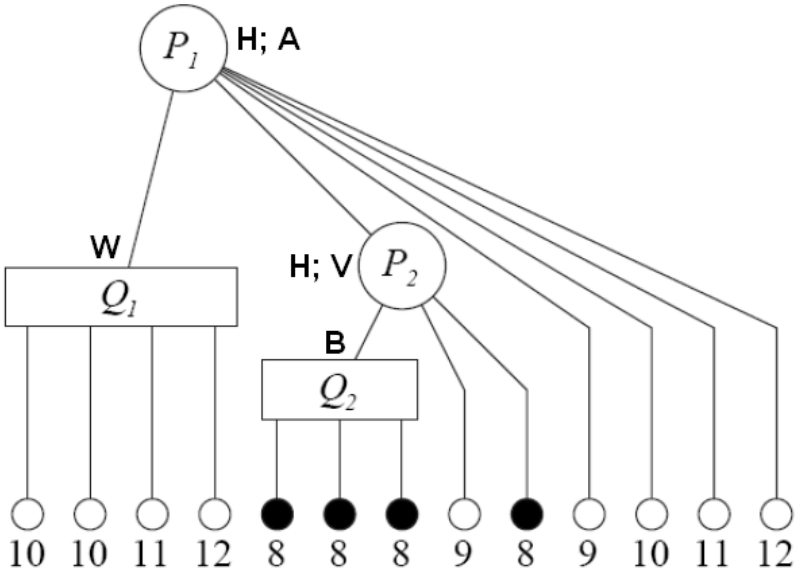


Figura 3.2. Exemplo de classificação de nós segundo teste de Ozawa e Takahashi

Percebe-se que nem todo nó pertence a uma dessas classificações. Para uma árvore-*PQ* que representa uma forma arbusto B_k, a classificação com relação ao conjunto de vértices virtuais de rótulo $k+1$ cobre todos os vértices. Quando um nó não for classificado com um desses tipos, a árvore é alterada de acordo com o algoritmo de planarização, por exemplo, pela remoção de arestas.

O algoritmo *VD-PLANARIZE* proposto se baseia no algoritmo de planarização *PLANARIZE* desenvolvido por Jayakumar, Thulasiraman e Swamy, mas a nossa proposta utiliza a operação de remoção de vértices. Tendo em vista que na fase de remoção, dado o arbusto B_k, ao contrário do processo de remoção de arestas onde o *PLANARIZE* tem que realizar testes para descobrir quais arestas devem ser removidas, o *VD-PLANARIZE* apenas

remove o vértice $k+1$ com uma operação de tempo constante. Essa operação garante ao algoritmo uma complexidade de tempo $O(n+m)$ para o *VD-PLANARIZE* contra $O(n^2)$ para o *PLANARIZE*, onde n é o número de vértices do grafo de entrada e m seu número de arestas.

De maneira geral, o algoritmo *VD-PLANARIZE* iterage com os vértices do grafo seguindo uma ordem definida por uma *st*-numeração, e tenta inserir cada vértice em uma árvore-*PQ* auxiliar. Antes de realizar uma inserção, o algoritmo verifica se é possível reduzir a árvore para o conjunto de nós virtuais de rótulo igual ao do vértice da iteração atual. Caso não seja possível reduzir, o vértice atual é removido do grafo e a árvore é re-arranjada para o próximo vértice. Caso seja possível, essa redução é feita e a nova forma arbusto do vértice seguinte é construída. Por se tratar essencialmente do algoritmo de *LEC*, ao final da execução o grafo resultante é planar.

```
   VD-PLANARIZE
   Entrada: grafo G' isomorfo a 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'
1: Início:
2:     construir a árvore inicial T_1;
3:     para k := 2 até n − 2 faça: {seguindo a st − numeração}
4:         se T_{k−1} for redutível {teste de Ozawa e Takahashi} então:
5:             aplica a redução {algoritmo de Booth e Lueker};
6:         senão
7:             UPDATE(v_k);
8:         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;
9:     retorna G;
10: Fim;
```

Figura 3.3. Pseudocódigo do algoritmo estudado VD-PLANARIZE

A Figura 3.3 apresenta o pseudocódigo do algoritmo estudado *VD-PLANARIZE*. O algoritmo recebe um *st*-grafo G' isomorfo a G e iterage para os vértices $k=2$ até $k=n-2$ (assim como no algoritmo de *LEC*).

Seja v_k o vértice da iteração k e T_{k-1} a árvore-*PQ* que representa o arbusto B_{k-1}, para cada iteração, inicialmente T_{k-1} é verificada pelo teste de Ozawa e Takahashi para certificar sua redutibilidade. Caso a redução seja possível, então o algoritmo de Booth e Lueker é aplicado para derivar T^*_{k-1} de T_{k-1}, em outras palavras a redução é aplicada em T_{k-1}. Caso o teste de Ozawa e Takahashi acuse que a redução é impossível, o método *UPDATE* é chamado

para o vértice v_k.

A Figura 3.4 apresenta o pseudocódigo da função *UPDATE*. Esta função recebe um grafo G', uma árvore T_{k-1}, um vértice v_k e remove este vértice de G' garantindo que a árvore T_{k-1} contenha em sua fronteira o vértice v_{k+1} para que o *VD-PLANARIZE* possa prosseguir. Para garantir essa propriedade sem afetar no desempenho do algoritmo, a função *UPDATE* utiliza uma lista denominada $small(u)$ que armazena o total de vértices adjacentes a u que tem rótulos menores do que o rótulo de u, de acordo com a *st*-numeração de G'. Quando o valor de $small$ se torna zero para algum vértice v, uma nova aresta $\{s, v\}$ e vértice v virtuais são inseridos na árvore T_{k-1}.

```
UPDATE
Entrada: grafo G, árvore T_k e vértice v_k
Saída: G − v_k e T_{k−1} rearranjada de modo que a st-numeração seja respeitada
Pré-processamento: lista small atualizada
1: Início:
2:     remove o vértice v_k de G e todas suas arestas incidentes;
3:     remove todos os nós virtuais de rótulo igual a v_k de T_{k−1};
4:     para cada u adjacente a v_k faça:
5:         small(u) = small(u) − 1;
6:         se small(u) = 0 então:
7:             adicione um nó virtual u e uma aresta virtual {s, u} em T_{k−1};
8:     retorna G e T_{k−1};
9: Fim;
```

Figura 3.4. Pseudocódigo da função UPDATE

Após a redução ou execução da função *UPDATE*, a árvore T_k é obtida de T^*_{k-1} removendo-se todas as folhas com rótulo v_k e inserindo no lugar um novo nó P de rótulo P_k e para cada vértice v_i adjacente a v_k, onde $i > k$, inserir um novo vértice virtual de rótulo v_i e uma aresta virtual $\{P_k, v_i\}$.

Esse processo é repetido até que $k = n - 2$. Quando isso acontecer, $G' = (V', E')$ será o subgrafo planar induzido de $G = (V, E)$ por V'.

O algoritmo de redução proposto por Booth e Lueker, utilizado no *VD-PLANARIZE* para reduzir todas as árvores-*PQ* redutíveis T_k tem seu tempo total executado com complexidade de tempo de $O(n + m)$ (Jayakumar, Thulasiraman e Swamy, 1989). Quando uma árvore-*PQ* T_k não é redutível, executa-se a operação *UPDATE* que fará a remoção dos vértices pertinentes. Supomos o pior caso com o máximo de vértices removidos (note que isto é verdadeiro para o grafo K_n, pois o grafo resultante não será planar até que tenha $n - 4$ vértices removidos). Para cada vértice v removido, o algoritmo inspeciona os rótulos de cada

vértice u adjacente a v. Se o rótulo de u for maior que o rótulo de v, o valor de $small(u)$ é reduzido uma unidade. Portanto, a operação *UPDATE* inspeciona cada vértice e seus adjacentes. Logo, no pior caso, o tempo total dessa operação é $O(n + m)$. A adição de arestas virtuais ao vértice s (raiz) é feita, no pior caso, n vezes. Portanto a complexidade de tempo do *VD-PLANARIZE* é $O(m + n)$.

A Figura 3.5 apresenta um exemplo de execução do *VD-PLANARIZE* para o grafo *st*-numerado da Figura 2.27. Esse exemplo usa uma *st*-numeração obtida pelo algoritmo de Tarjan e exibe todo o procedimento de planarização.

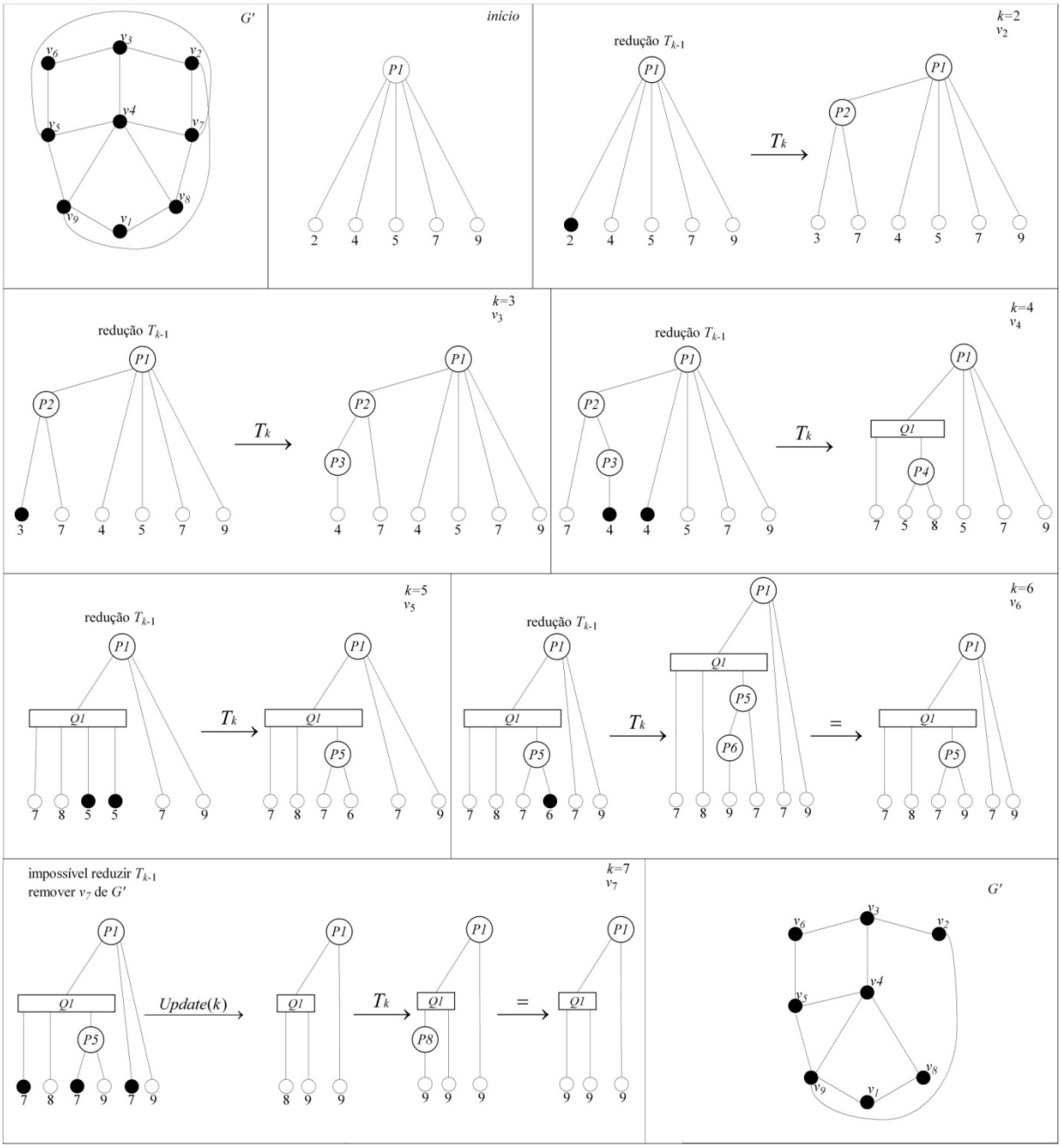


Figura 3.5. Exemplo de execução do VD-PLANARIZE para o st-grafo da Figura 2.27

A Figura 3.6 apresenta um exemplo da execução do algoritmo *VD-PLANARIZE* para um grafo $C_4 \times C_4$ com uma *st*-numeração que faça com que o algoritmo chegue ao resultado ótimo, isto é, que remova o mínimo de vértices possível. Diferente do exemplo da Figura 3.5, o exemplo da Figura 3.6 mostra que cada vértice vai sendo imerso em um novo grafo até obter-se o resultado final para que se possa visualizar a relação do arbusto com a geração do grafo planar.

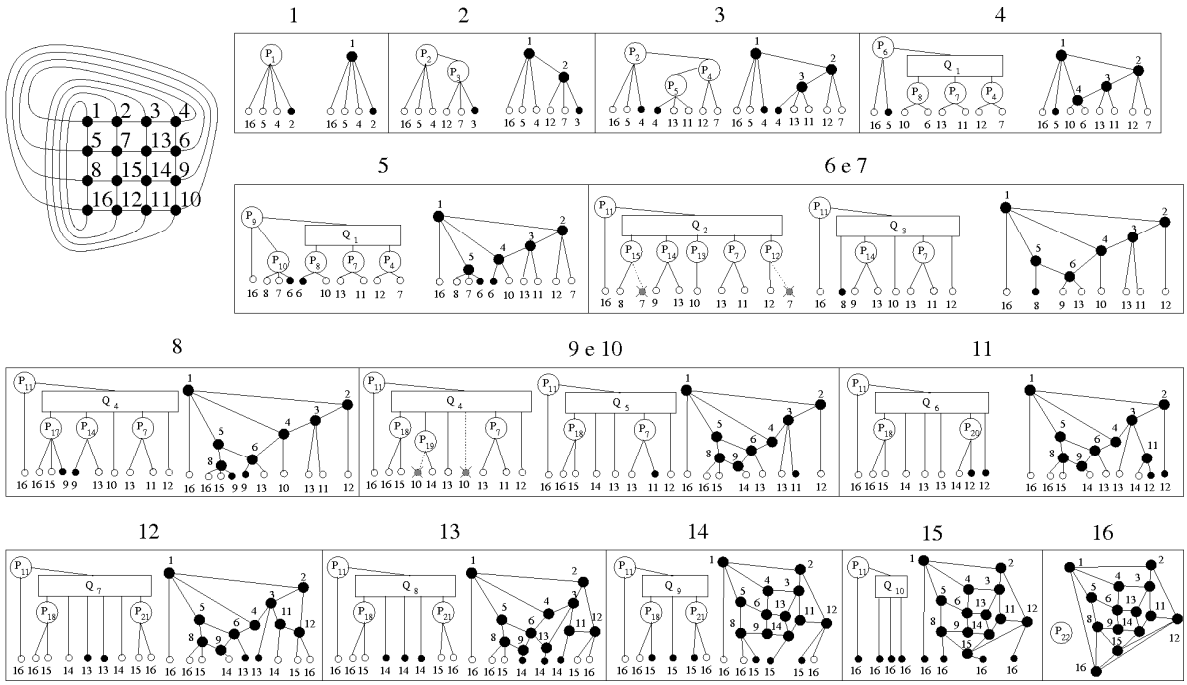


Figura 3.6. Exemplo de execução do VD-PLANARIZE para um grafo $C_4 \times C_4$

3.1.1. Qualidade das Soluções

A qualidade das soluções do algoritmo *VD-PLANARIZE* é totalmente dependente da *st*-numeração, pois dada diferentes numerações, as árvores T_k serão montadas de maneiras distintas onde o resultado final pode ser completamente diferente. A Figura 3.7 apresenta um exemplo de um grafo bipartido K_{nm} onde a *st*-numeração apresentada leva o *VD-PLANARIZE* a planarizar o grafo de maneira ótima de acordo com o valor teórico de $\min\{m, n\} - 2$.

Em contrapartida, a Figura 3.8 apresenta um exemplo, também para grafos bipartidos K_{nm}, onde a *st*-numeração apresentada leva o grafo a ser planarizado da pior forma possível ao eliminar $n - 5$ vértices. Fica evidente que a qualidade do resultado produzido pelo *VD-PLANARIZE* depende de se obter uma *st*-numeração adequada.

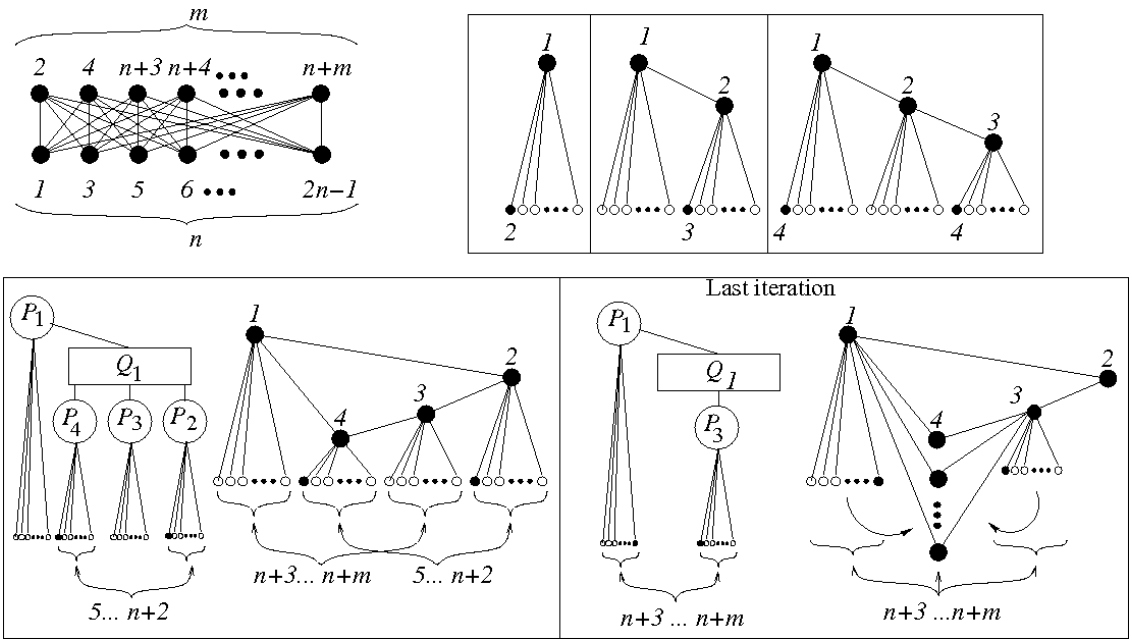


Figura 3.7. Exemplo de Melhor Caso de Execução do Algoritmo para o K_{mn}

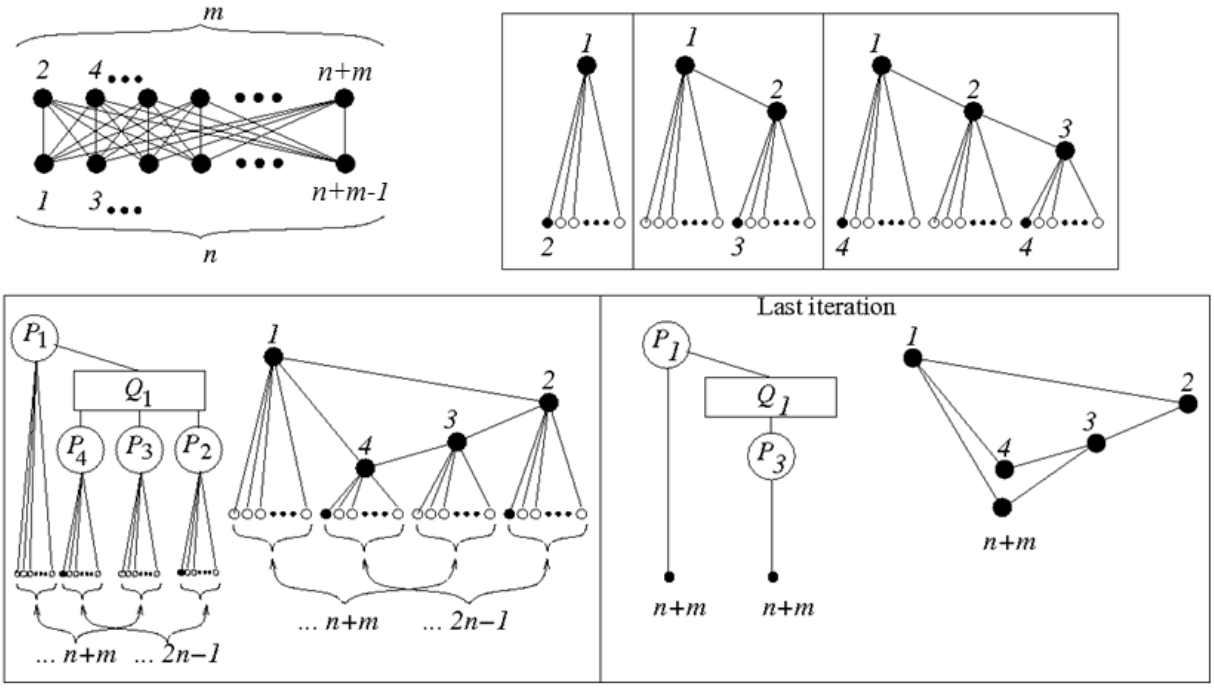


Figura 3.8. Exemplo de Pior Caso de Execução do Algoritmo para o K_{mn}

Os dois algoritmos de st-numeração apresentados, Even-Tarjan e Tarjan, produzem resultados que são influenciados pela escolha da aresta $\{s,t\}$ e ordem de visita dos vértices adjacentes de cada vértice. Portanto gerar apenas duas *st*-numerações para cada par de vértices – para a aresta $\{u,v\}$ pode-se partir de u para v ou de v para u - não cobre todas as possibilidades de numerações possíveis. O espaço total de combinações possíveis para a entrada dos métodos de *st*-numeração é dado por duas vezes o número de arestas do grafo vezes o número de combinações possíveis da ordem de visita das adjacências de cada vértice.

Dado um grafo G, os limites do número de combinações c são definidos pela Equação (3).

$$2m[g_h!\, g_l!^{n-1}] \leq c \leq 2m g_h!^n \quad (3),$$

onde n é o número de vértices, m é o número de arestas, g_h é o maior grau entre todos os vértices do grafo, e g_l é o menor grau entre todos os vértices do grafo.

Apesar das inúmeras possibilidades de entradas para os métodos de *st*-numeração, não há garantias que eles cubram todo o espaço de possibilidade de numerações do grafo.

Outro fator acerca da qualidade das soluções do *VD-PLANARIZE* é que não há garantias da existência de uma *st*-numeração que leve o algoritmo a encontrar a solução planar ótima de um grafo. O algoritmo tem seu desempenho dependente da *st*-numeração dos vértices do grafo, e não há garantias de que as possibilidades de diferentes *st*-numerações – inclusive as que não são possíveis de serem geradas pelos métodos de *st*-numerações - cubram todo o espaço de soluções possíveis de planarização.

Conclui-se que mesmo sem essas garantias, pode-se observar pelos exemplos dos grafos K_{mn} que a variedade de soluções possíveis de serem alcançadas pelo *VD-PLANARIZE* é muito grande, o que torna o algoritmo de planarização eficiente se uma boa técnica de *st*-numeração for associada. Essa variedade de soluções foi comprovada experimentalmente com detalhes da implementação como se segue.

3.1.2. Detalhes de Implementação

Esta seção apresenta alguns pontos importantes acerca da implementação dos algoritmos de *st*-numeração e do *VD-PLANARIZE*. Inicialmente é abordada a estrutura de dados dos grafos e posteriormente das outras estruturas e dos algoritmos envolvidos neste trabalho.

Os algoritmos de *st*-numeração e de planarização necessitam usar operações que forneçam conhecimento sobre a vizinhança de um vértice e percorrer vértices e arestas. Optou-se por utilizar uma estrutura de dados que armazena as relações de adjacências dos vértices de um grafo, uma vez que o uso dessa estrutura se mostra adequado para realizar essas operações. Outra razão para o uso de uma estrutura de adjacências, é que a ordem de visita dos vértices não é fixa e para gerar diferentes *st*-numerações baseado em ordem de visitas distintas não é necessário usar estruturas auxiliares.

A Figura 3.9 apresenta o exemplo de duas estruturas de adjacências diferentes que representam o mesmo grafo. Considerando que os algoritmos de *st*-numeração para a fase de construção da árvore de expansão seguem a ordem de visita fornecida das adjacências de um

vértice, o uso dessas estruturas torna possível que os algoritmos de *st*-numeração produzam diferentes resultados, pois a ordem dos vértices visitados utilizada é aquela que está na lista de cada vértice.

Quando há a necessidade de um grafo receber várias *st*-numerações distintas, cópias do grafo são geradas e as várias configurações diferentes da ordem de visita dos vértices são representadas nas listas de adjacências de cada vértice.

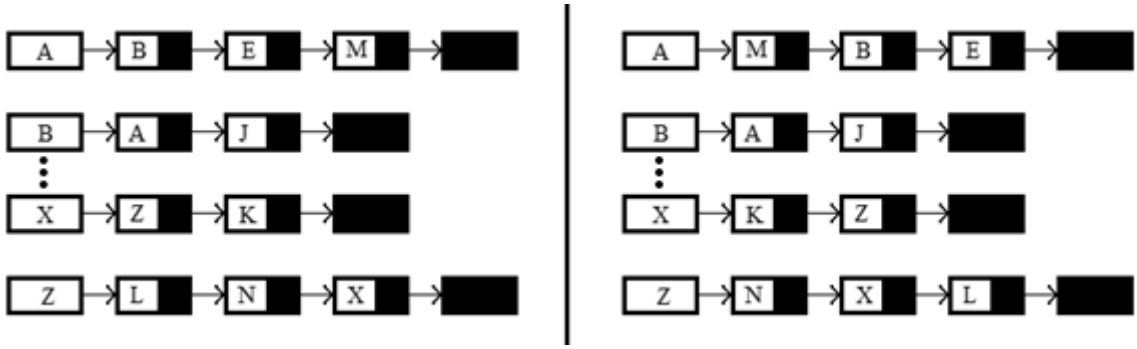


Figura 3.9. Exemplo de duas configurações distintas para o mesmo grafo

Para diferentes *st*-numerações foram utilizados vetores que armazenam o *st*-número dos vértices. As funções básicas de grafos, como percorrer vértices e arestas foram implementadas levando em consideração esse vetor e não a ordem dos vértices na estrutura. Dessa forma não ocorre nenhuma alteração no comportamento da função de tempo dos algoritmos.

Este trabalho implementa a estrutura de árvores-*PQ* proposta por Harris (2002). Ela utiliza diferentes vetores para armazenar os diferentes tipos de nós da árvore e apresenta uma implementação eficiente da operação de redução utilizando o algoritmo de Booth e Lueker. Todos os procedimentos e funções que manipulam árvores-*PQ* neste trabalho, tais como o teste de Ozawa e Takahashi e a função *UPDATE*, estão orientados para manipular essas estruturas.

Os algoritmos abordados neste trabalho foram implementados na linguagem *Java*. Para efetuar as medições de tempo, tomou-se o cuidado de realizar chamadas regulares ao *Garbage Collector* para desalocar objetos não mais referenciados na memória, para que não houvesse comprometimento de performance na realização dos testes.

3.2. GAVD-PLANARIZE

Após o estudo acerca da qualidade das soluções do *VD-PLANARIZE* apresentado na seção 3.1.1, onde se concluiu que o espaço de possíveis *st*-numerações passíveis de serem geradas

pelos algoritmos de *st*-numeração é muito grande, optou-se por estudar alguma técnica que explorasse esse espaço em busca de boas planarizações. Considera-se então o uso de metaheurísticas que comprovaram ser eficientes na busca de boas soluções quando se tem um espaço de possibilidades muito grande.

Uma metaheurística procura melhorar as soluções obtidas para um problema. No contexto da planarização de grafos por remoção de vértices, o objetivo é encontrar um subgrafo planar com um maior número de vértices possível, desta forma removendo a menor quantidade de vértices do grafo original.

Como o algoritmo proposto *VD-PLANARIZE* possui complexidade de tempo linear, seu uso para avaliar diversas soluções se mostra eficiente. Os algoritmos apresentados de *st*-numeração também apresentam complexidade de tempo linear, dessa forma a metaheurística que combinar esses algoritmos terá uma complexidade de tempo eficiente. Dessa maneira propomos o algoritmo *GAVD-PLANARIZE*.

O *GAVD-PLANARIZE* é definido pela a estrutura básica de um algoritmo genético, apresentada na Figura 2.34. Seguindo os conceitos de algoritmos genéticos apresentados, dividimos esta seção em quatro partes abordando os diferentes componentes de uma metaheurística genética *GAVD-PLANARIZE*. Essas seções abordam detalhes de implementação dos indivíduos, da população, da seleção e do cruzamento.

3.2.1. Indivíduos

O *GAVD-PLANARIZE* implementa os indivíduos como sendo estruturas que contêm uma cópia da lista de adjacências do grafo a ser planarizado, uma aresta $\{s, t\}$ para uso do método de *st*-numeração, um valor de aptidão e um vetor contendo uma *st*-numeração. A Figura 3.10 mostra uma representação gráfica da organização de um indivíduo.

O cromossomo de um indivíduo tratado no *GAVD-PLANARIZE* como sendo uma cópia da lista de adjacências do grafo a ser planarizado. Dado um grafo G e um cromossomo c_i, esse cromossomo é constituído de n genes, onde n representa o número de vértices de G. Cada gene g_k é composto pela lista de adjacências do vértice v_k.

Sempre que um indivíduo é gerado, seja na população inicial ou após um cruzamento, o algoritmo *TARJANST-NUMBERING* é aplicado sobre a sua estrutura de adjacências utilizando a aresta $\{s, t\}$ do próprio indivíduo para obter uma *st*-numeração, o qual é armazenada no vetor st. Depois de obtida uma *st*-numeração, o valor de aptidão do indivíduo é calculado e armazenado. Somente após, esse indivíduo está pronto para ser submetido aos

processos de seleção e cruzamento.

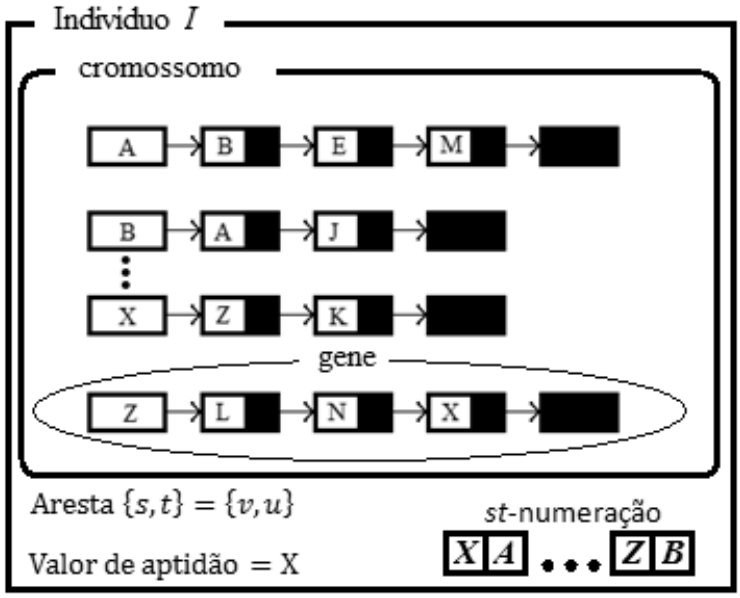


Figura 3.10. Exemplo de um indivíduo do GAVD-PLANARIZE

3.2.2. População

O *GAVD-PLANARIZE* foi projetado para utilizar uma população de tamanho fixo. No caso geral, a geração dos indivíduos da população é feita aleatoriamente, copiando a lista de adjacências do grafo original e embaralhando os elementos.

Para os casos onde o grafo a ser planarizado é um grafo $C_n \times C_m$ a população é gerada de forma diferente. Como o grau de cada vértice é quatro, o número de combinações possíveis para um gene g_k é $4! = 24$, portanto são gerados no mínimo vinte e quatro indivíduos. Dado uma população P, para todo indivíduo $i_k^1 \in P$, onde $|P| \geq 24$ e para $1 \leq k \leq 24$, se i_k^1 contém um gene g qualquer, não existe um individuo i_j^2 onde $1 \leq j \leq 24$ e $j \neq k$ que contém o gene g.

Essa abordagem para grafos da classe $C_n \times C_m$ garante que a população inicial conterá todas as possíveis variedades de genes. Caso a população seja maior que 24, os outros indivíduos têm seus genes sorteados aleatoriamente.

A Figura 3.11 apresenta uma representação gráfica dos cromossomos da população inicial mínima gerada para um grafo $C_3 \times C_3$. São vinte e quatro indivíduos onde nenhum gene se repete, garantindo que diferentes áreas do espaço de soluções sejam visitadas.

Durante a seleção e renovação da população, optou-se por utilizar um sistema de elitismo. O elitismo consiste em manter parte da população e renovar a contraparte. O *GAVD-*

PLANARIZE mantém 10% da população e o restante passa pelo processo de seleção. A parte mantida é denominada *elite*.

Na geração da população, assim que um indivíduo é aleatoriamente gerado uma aresta $\{s, t\}$ aleatória é atribuída a esse individuo.

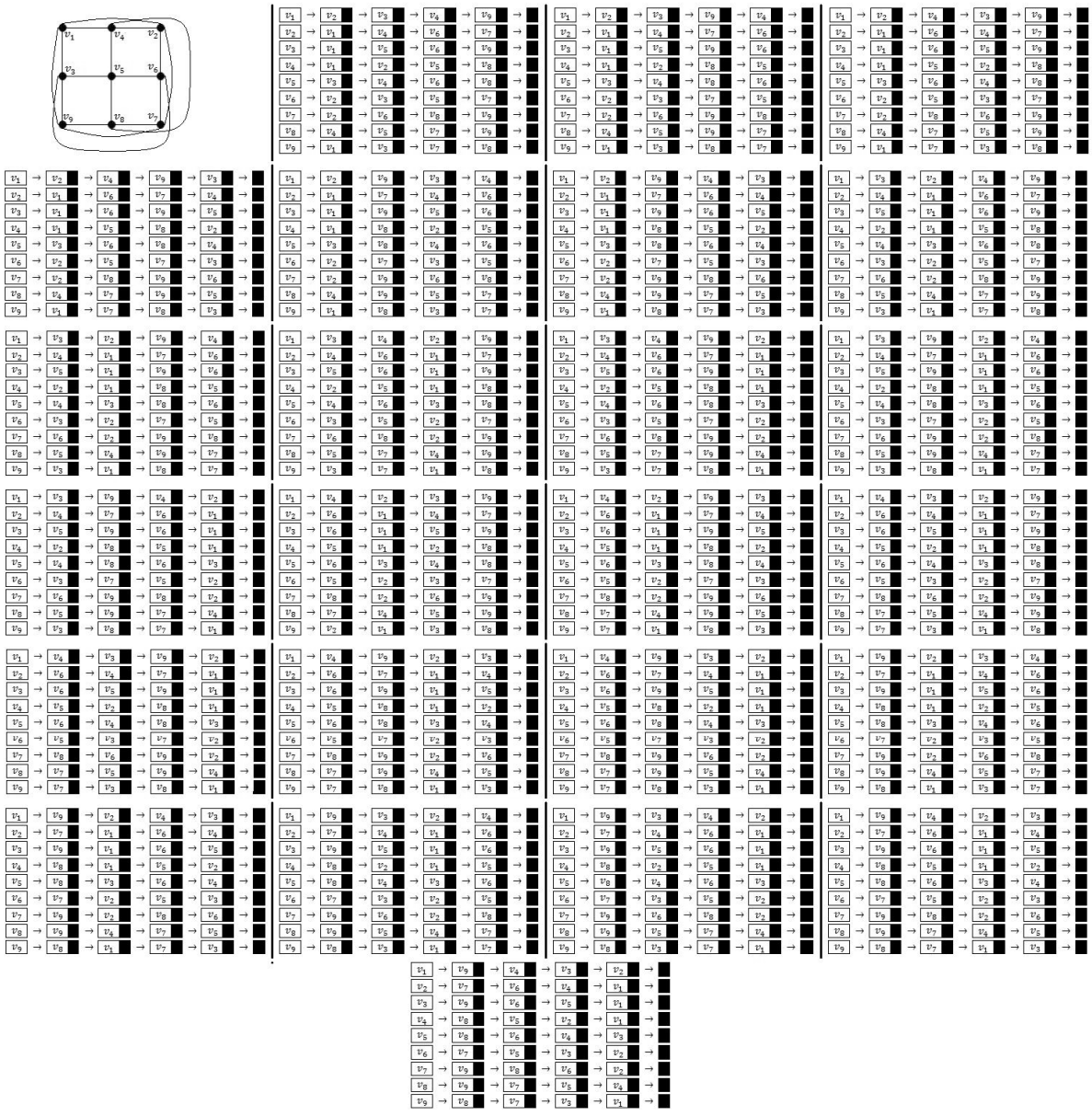


Figura 3.11. Combinações de cromossomos do GAVD-PLANARIZE para um grafo $C_3 \times C_3$

3.2.3. Seleção

A metaheurística proposta utiliza *scaling* linear para determinar a aptidão de um indivíduo. Dessa forma temos que a função objetivo $f_{scaling}(f_O(x))$ para um indivíduo x qualquer é determinada pela equação:

$$f_{scaling}\left(f_O(x)\right) = f_O(x) \times a + b,$$

onde

$$\begin{cases} a = (c-1) \times \frac{med}{\Delta} \text{ e } \quad b = med \times \frac{max - c \times med}{\Delta}, se\ \Delta \neq 0 \\ a = 1, b = 0, \text{caso contrário.} \end{cases},$$

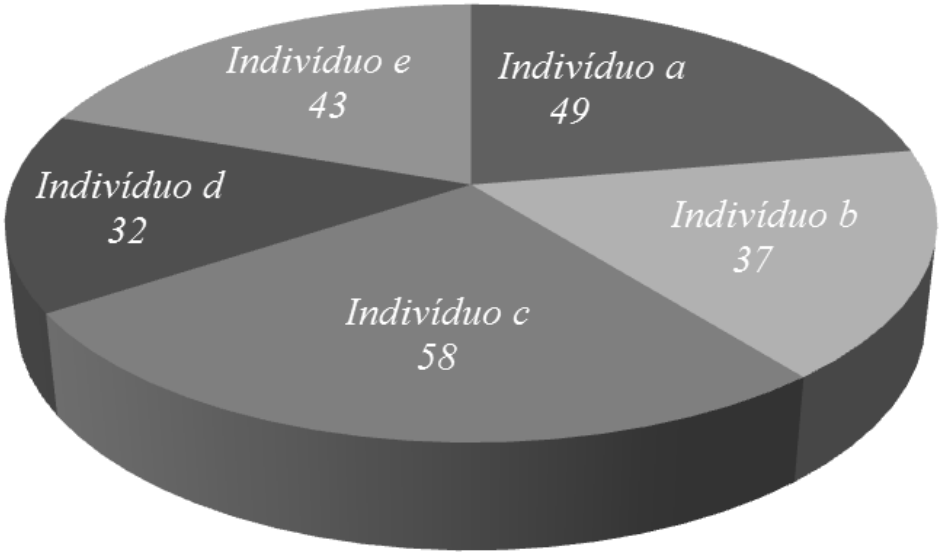
$f_O(x)$ é o número de vértices do grafo planar do indivíduo x, max é o número de vértices do maior grafo planar encontrado, med é a média de todas as soluções, Δ é o número de vértices do maior grafo planar encontrado naquela população menos o valor de $f_O(x)$ e o fator escala foi definido $c = 1{,}5$ para que indivíduos menos aptos tenham um pouco a mais de chances de serem sorteados.

O *GAVD-PLANARIZE* utiliza um sistema de elitismo que mantém sempre 10% dos indivíduos mais aptos da população atual para compor a próxima geração. O critério de seleção para manter a elite é avaliar cada indivíduo de uma geração e manter os k indivíduos mais aptos na geração seguinte, onde k representa a décima parte do número de indivíduos da população.

Após definir a elite, o processo de seleção escolhe, pela aplicação do método da roleta, pares de indivíduos para que se reproduzam até que uma nova geração de indivíduos seja obtida. O processo de seleção pelo método da roleta é definido em função do valor de aptidão t_k de cada indivíduo, e o valor total $t_s = \sum_{k=1}^{n} t_k$, onde n é o número de indivíduos da população. Sorteia-se um valor aleatório r onde $1 \leq r \leq t_s$ e seleciona-se os indivíduos que pertencem às faixas dos somatórios dos números sorteados.

A Figura 3.12 apresenta um exemplo de como a seleção pelo método da roleta funciona. A população do exemplo é composta por cinco indivíduos e os valores de aptidão de cada um são exibidos no gráfico. A tabela inclusa na figura mostra o valor da margem de r, sendo que para cada valor possível de r, um indivíduo é selecionado. A tabela ainda apresenta a probabilidade de sorteio de cada indivíduo.

O *GAVD-PLANARIZE* assume que o tamanho da população é relativamente grande e no caso de haver o sorteio de pares compostos por indivíduos idênticos, o algoritmo repete o sorteio do último indivíduo do par até que seja obtido um indivíduo diferente.



$t_s = 219$

Indivíduo	a	b	c	d	e
k	1	2	3	4	5
t_k	49	37	58	32	43
Margem de r	1-49	50-86	87-144	145-176	177-219
Porcentagem	22%	16%	27%	15%	20%

Figura 3.12. Exemplo do sistema de seleção utilizado no GAVD-PLANARIZE

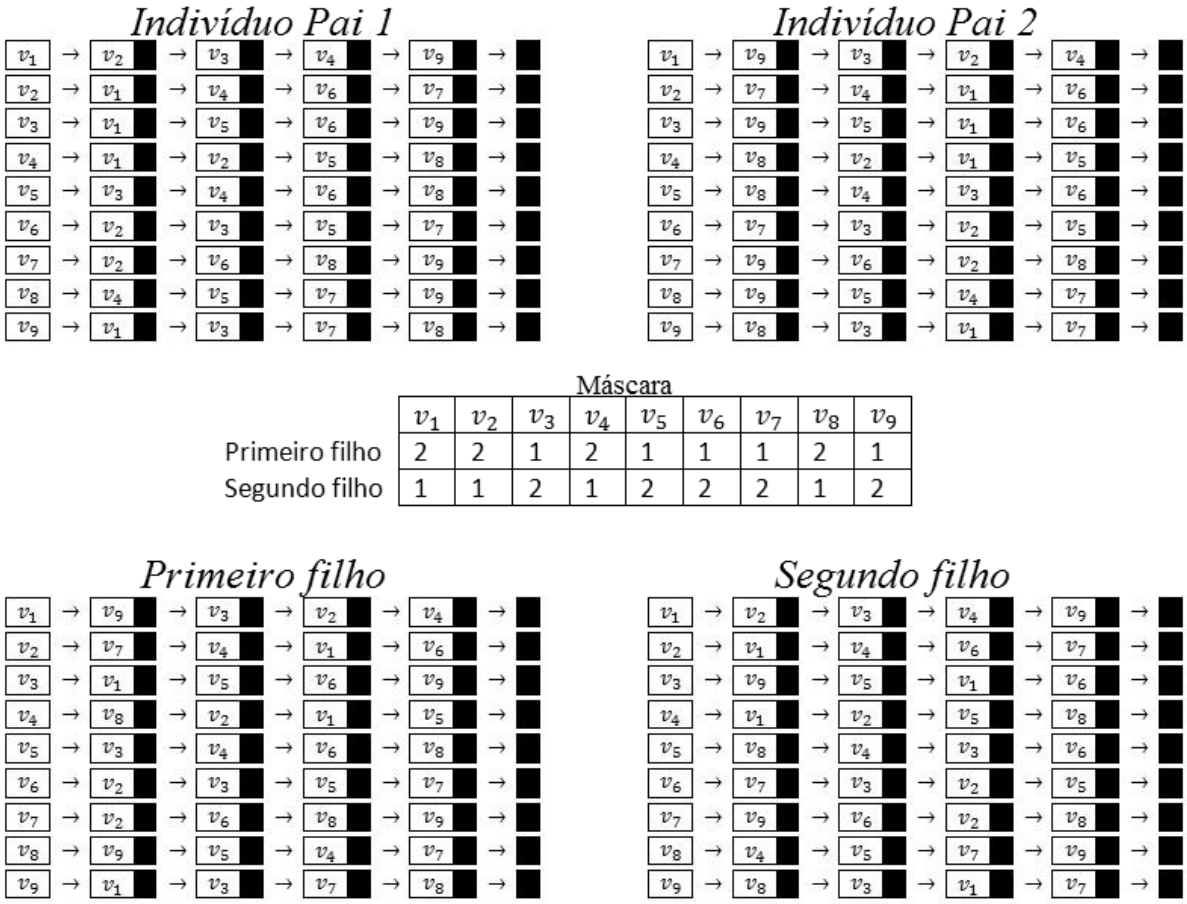
3.2.4. Cruzamento

O mecanismo de reprodução proposto para o *GAVD-PLANARIZE* é composto de duas fases. A primeira realiza a operação de *crossover* e a segunda efetua uma busca local gulosa para encontrar a melhor aresta $\{s, t\}$ para ser utilizada pelo método de *st*-numeração.

A operação de *crossover* é efetuada pela aplicação do método uniforme. Para cada cruzamento, os genes que o primeiro filho herda são sorteados para determinar se esses genes devem ser herdados do primeiro ou do segundo pai. O segundo filho recebe os genes complementares em relação aos herdados pelo primeiro filho.

A Figura 3.13 apresenta um exemplo de cruzamento, de dois pais selecionados da Figura 3.11, para o grafo $C_3 \times C_3$, mostrando os cromossomos dos pais e filhos bem como a máscara usada para gerar os filhos.

Depois de gerados os cromossomos e antes da definição do valor de aptidão de cada filho, todo indivíduo gerado é submetido a uma pequena chance α de sofrer uma mutação. O processo de mutação é realizado utilizando a técnica *flip* que no caso escolhe um gene ao acaso e embaralha toda a ordem dos vértices deste gene.



Máscara	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
Primeiro filho	2	2	1	2	1	1	1	2	1
Segundo filho	1	1	2	1	2	2	2	1	2

Figura 3.13. Exemplo de reprodução para pais tirados da Figura 3.11

Ao gerar os cromossomos dos filhos, o *GAVD-PLANARIZE* executa um algoritmo não bio-baseado chamado *GREEDYST-SEARCH* que faz uma busca local gulosa na vizinhança das arestas $\{s, t\}$ para encontrar a melhor aresta para aquela configuração de ordem de visita.

O algoritmo *GREEDYST-SEARCH* inicia sua busca pela melhor aresta dos seus pais. A escolha dessa aresta é feita no momento do cruzamento e consiste em gerar diferentes *st*-numerações dadas as arestas $\{s, t\}$ e $\{t, s\}$ de cada pai e a estrutura de adjacências do filho. Após gerar as *st*-numerações, para cada *st*-numeração o grafo do indivíduo filho é planarizado com o algoritmo *VD-PLANARIZE*, e a aresta $\{s, t\}$ da *st*-numeração associada ao grafo planar de maior número de vértices é escolhida como ponto de partida do algoritmo *GREEDYST-SEARCH*.

A Figura 3.14 apresenta o pseudocódigo do algoritmo *GREEDYST-SEARCH*. O algoritmo inicia a busca na aresta $\{s, t\}$ inicial gerando uma *st*-numeração para essa aresta e planarizando o grafo com o algoritmo *VD-PLANARIZE*, e para cada vértice v adjacente a s o algoritmo *GREEDYST-SEARCH* gera uma *st*-numeração para a aresta $\{s, v\}$ e então planariza o grafo com o algoritmo *VD-PLANARIZE*. Se o resultado for um subgrafo planar de número

de vértices maior do que o subgrafo planar obtido pela aresta $\{s,t\}$ inicial, então v substitui t. O algoritmo *GREEDYST-SEARCH* realiza o mesmo processo para a aresta $\{v,s\}$ e caso o grafo planar obtido tenha mais vértices que o melhor obtido até então, a aresta $\{s,t\} \coloneqq \{v,s\}$ e o algoritmo *GREEDYST-SEARCH* retorna uma chamada recursiva sobre a nova aresta $\{s,t\}$. O algoritmo encerra quando não houver melhora nas soluções.

```
GREEDYST-SEARCH
Entrada: um indivíduo i e uma aresta {s, t}
Saída: uma aresta {s, t}
Pré-processamento: determinar a melhor aresta {s, t} dentre as arestas dos dois pais, incluindo as arestas reversas {t, s}
1:  Início:
2:      valor_m := |VD-PLANARIZE(TARJANST-NUMBERING(i.cromossomo), {s, t})|;
3:      para v_k adjacente a s faça:
4:          valor_a := |VD-PLANARIZE(TARJANST-NUMBERING(i.cromossomo), {s, v_k})|;
5:          se valor_a > valor_m então:
6:              valor_m := valor_a;
7:              t := v_k
8:          valor_a := |VD-PLANARIZE(TARJANST-NUMBERING(i.cromossomo), {v_k, s})|;
9:          se valor_a > valor_m então:
10:             retorna GREEDYST-SEARCH(i, {v_k, s});
11:     retorna {s, t};
12: Fim;
```

Figura 3.14. Pseudocódigo do método GREEDYST-SEARCH

Propomos esta abordagem devido ao modelo de cromossomo utilizado não alterar a aresta $\{s,t\}$ que também tem uma importância decisiva para a qualidade das soluções, já que a escolha de diferentes arestas $\{s,t\}$ pode definir diferentes *st*-numerações. Não obstante, tendo em vista que o número de combinações de arestas $\{s,t\}$ não ultrapassa duas vezes o número de arestas do grafo, o que é pouco, se comparado ao número de combinações possíveis que um cromossomo pode ter, a heurística proposta se mostra uma boa opção para o tratamento do problema por explorar o espaço de possíveis soluções em busca da melhor solução.

Com isso, o *GAVD-PLANARIZE* apresenta um meio de busca no espaço de soluções possíveis do *VD-PLANARIZE* e uma forma de refinar soluções baseado nas *st*-numerações.

CAPÍTULO

4

Resultados Obtidos

Este capítulo apresenta os resultados obtidos nos experimentos realizados. Os algoritmos foram implementados em *Java 6 Update 14* e os testes de medição de tempo foram executados em um *PC AMD Athlon XP 2600+* com *1GB* de memória na plataforma *Windows XP*.

4.1. VD-PLANARIZE

O primeiro algoritmo experimentado foi o *VD-PLANARIZE*, pois ele forma a base do *GAVD-PLANARIZE*. Inicialmente foram realizados testes relativos ao tempo de execução e posteriormente avaliado as soluções e o impacto de diferentes métodos de st-numeração na qualidade das soluções.

4.1.1. Tempo

No Capítulo 3 foi mostrado que o *VD-PLANARIZE* apresenta uma complexidade de tempo teórica de $O(m+n)$. É apresentado agora se a complexidade experimental corresponde com a teórica.

Para realizar o teste de tempo, foi gerada uma seqüencia de pares de grafos quaisquer idênticos de tamanhos $m+n$ crescentes. Para cada par de grafos gerados foi gerada uma *st-*

numeração e feita a planarização desses grafos utilizando o *VD-PLANARIZE*, e foi tomado como tempo de planarização o menor valor obtido para cada par. A decisão de gerar pares foi tomada a fim de minimizar possíveis anomalias como algum processamento externo realizado pelo sistema operacional interferindo no desempenho da planarização.

Tendo em vista a necessidade de se controlar o tamanho do grafo, o meio mais simples encontrado para gerar os grafos consiste em um processo iterativo onde, a partir de um grafo G completo de quatro vértices, a cada iteração se a soma dos vértices e arestas $m + n$ de G é inferior ao tamanho desejado, se G for completo remove-se duas arestas e se insere um novo vértice adjacente aos vértices 1 e n (aumentando o seu tamanho de entrada $m + n$ em uma unidade). Caso G não é completo e não tenha alcançado o valor mínimo desejado de $m + n$ vértices e arestas, uma nova aresta é inserida (novamente aumentando o tamanho de vértices e arestas do grafo em uma unidade). Desta forma estabeleceu-se um mecanismo capaz de gerar grafos de tamanho $m + n$ crescentes de forma controlada.

O teste de tempo foi realizado com um grafo de tamanho mínimo de $m + n$ de 9500 e crescendo em intervalos regulares de 2000.

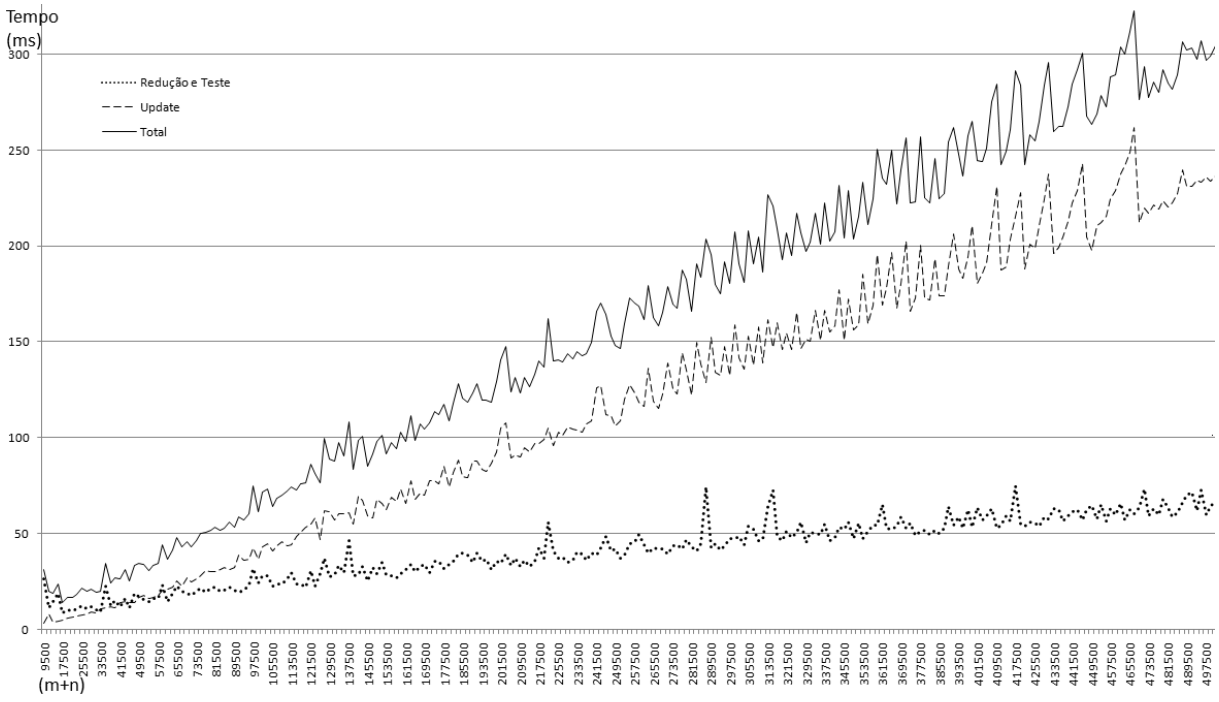


Figura 4.1. Gráfico com a curva de tempo de execução do algoritmo VD-PLANARIZE

A Figura 4.1 apresenta o gráfico da curva do tempo de execução. O eixo x do gráfico representa o tamanho do problema em valores de $m + n$ e o eixo y do gráfico representa o tempo necessário para o algoritmo planarizar o grafo. O gráfico apresenta três informações. A linha contínua representa o tempo total gasto pela execução do algoritmo, a linha tracejada

representa o tempo gasto pelas operações de *UPDATE* utilizadas na execução do algoritmo e a linha pontilhada representa o tempo gasto pelas operações de redução e no teste de possibilidade de redução.

A *média móvel simples* (ou somente *média móvel*) representa o valor médio em um intervalo de tempo. Seja V_i o valor de um gráfico no instante i, e N o intervalo de tempo do cálculo da média móvel, a média móvel no instante i é definida por $VM_i = \frac{V_i+V_{i+1}+\cdots+V_{i+N}}{N}$.

A Figura 4.2 mostra o gráfico das médias móveis intervalo de cinqüenta testes. O eixo x representa os $m+n$ vértices e arestas usados no cálculo de tempo médio. O eixo y representa o tempo médio consumido em milissegundos.

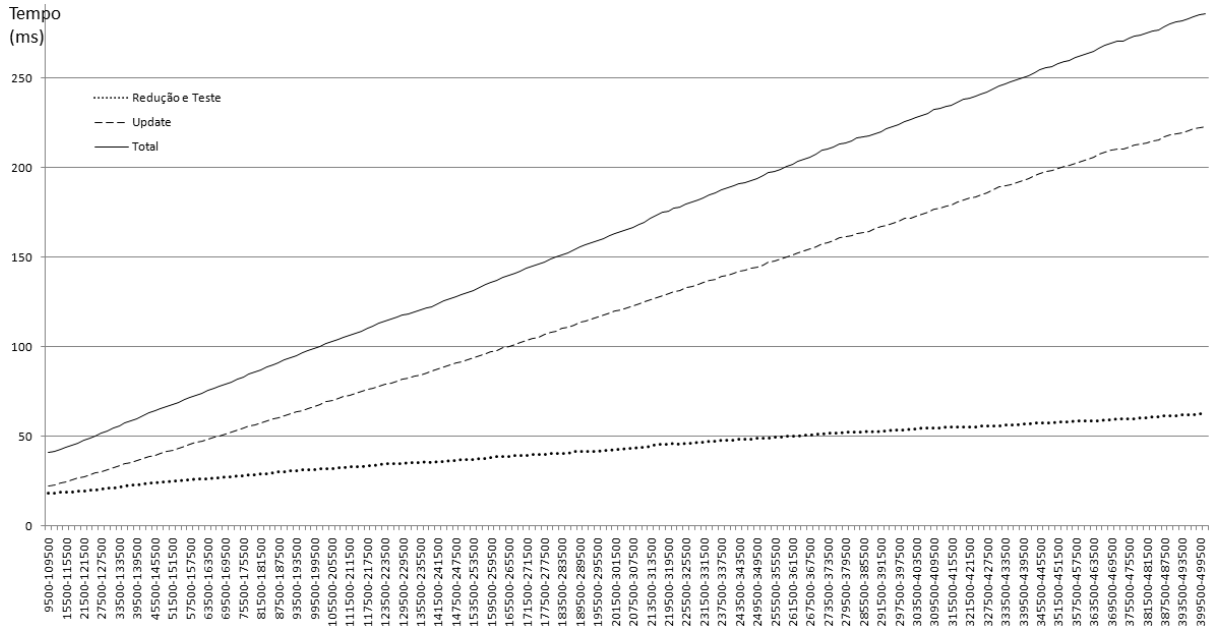


Figura 4.2. Gráfico com a curva das médias móveis para 50 testes do VD-PLANARIZE

A Figura 4.3 apresenta o último gráfico dos testes de tempo. Este gráfico mostra a diferença entre o tempo consumido por um teste *k* e o teste *k-1*. Valores positivos da linha clara significam que o teste *k* foi executado em mais tempo que o teste *k-1*, valores negativos representam o contrário. A linha tracejada representa a reta constante com valor de 1.115274 que é a média dentre todas as diferenças computadas.

Conforme demonstrado no Capitulo 3, o algoritmo *VD-PLANARIZE* possui complexidade de tempo de *O(m+n)*, o que pôde ser constatado nos gráficos da Figura 4.1 e da Figura 4.2. Como a Figura 4.2 nos mostra a tendência do gráfico, é visível que a curva total do tempo de execução do *VD-PLANARIZE* é linear.

O primeiro e o terceiro gráfico ainda nos mostram que existe uma oscilação relevante

no tempo de processamento, fazendo com que problemas maiores sejam resolvidos mais rapidamente que problemas menores. A oscilação do tempo total não ocorre exclusivamente em uma fase da execução, já que ela acompanha as oscilações tanto da função *UPDATE* quanto da função de redução e teste de redução, no entanto a maior oscilação ocorre durante a função *UPDATE*, o que tem uma explicação.

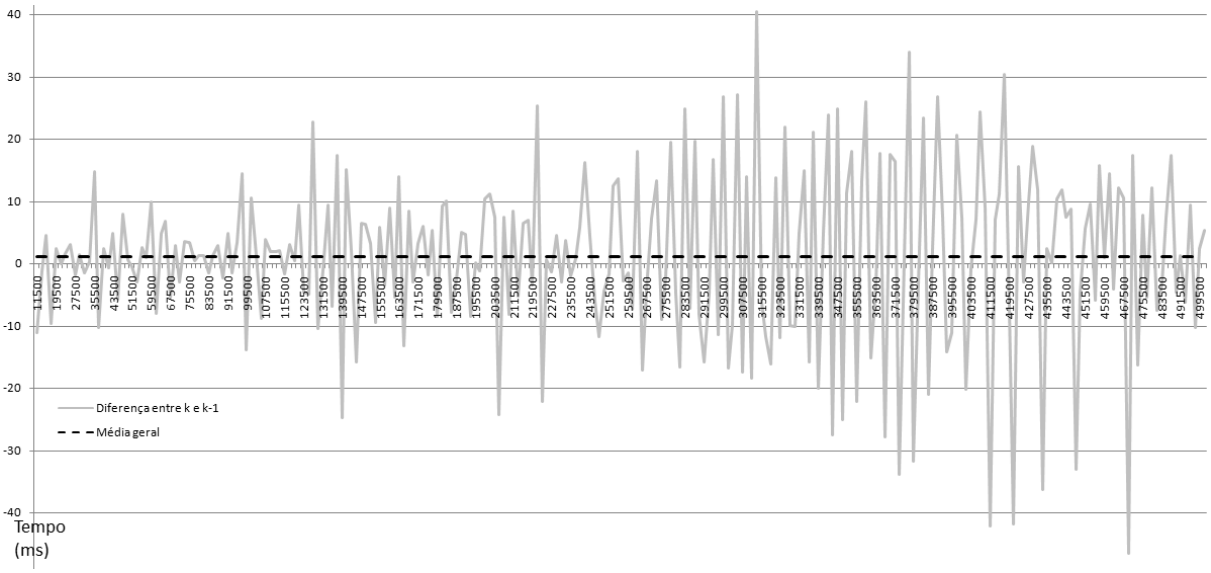


Figura 4.3. Diferença de tempo local de processamento entre o grafo $k-1$ e k.

O *Garbage Collector* do *Java* permite limpar referências perdidas na memória. Como os testes de tempo geram e se desfazem de muitos objetos (os próprios grafos, por exemplo), é comum que o *Garbage Collector* entre em atividade varias vezes para recuperar a memória da máquina. As oscilações observadas no gráfico da Figura 4.3 ocorrem devido a essa atividade do *Garbage Collector*, pois o programador não tem controle do momento em que ele é ativado. Observa-se no gráfico da Figura 4.1 que a maior necessidade de tempo de processamento é exigido pela função *UPDATE*, fato que a torna mais suscetível à interrupção em sua execução para a ativação do *Garbage Collector* para limpar a memória. No entanto, essa interferência não compromete o comportamento da função de crescimento de tempo, como mostra o gráfico da Figura 4.2.

4.1.2. Qualidade das Soluções

Depois de constatado experimentalmente que o algoritmo é executado em tempo linear, realizou-se testes para verificar a qualidade das soluções obtidas pelo *VD-PLANARIZE*. O primeiro teste avaliou os resultados da planarização com a aplicação dos dois métodos de *st-*

numeração estudados, primeiro com a classe de grafos $C_n \times C_m$ e posteriormente com grafos gerados randomicamente.

Para a classe $C_m \times C_n$ foram criados exemplares de grafos que variam de $C_3 \times C_3$ ao $C_{25} \times C_{25}$, contemplando todas as possíveis combinações intermediárias. A estrutura de dados escolhida para representar esses grafos é a lista de adjacências onde os elementos da lista apresentam uma ordem arbitrária de seus elementos. Dessa forma, a escolha de diferentes arestas gera diferentes *st*-numerações.

Para esse primeiro teste, os grafos $C_n \times C_m$ foram gerados e para cada aresta $\{u, v\}$ duas *st*-numerações foram estabelecidas para cada algoritmo de *st*-numeração (Tarjan e Even-Tarjan) – uma *st*-numeração com $s = u$ e $t = v$ e outra st-numeração com $s = v$ e $t = u$. As Figura 4.4 e Figura 4.5 apresentam os resultados obtidos no teste de performance do *VD-PLANARIZE* aplicado para as diferentes *st*-numerações obtidas.

A Figura 4.4 apresenta os resultados das melhores soluções obtidas pelo *VD-PLANARIZE* em conjunto com cada método de *st*-numeração. O eixo x representa os casos experimentados, e o eixo y representa o número de vértices removidos. O gráfico apresenta três linhas. A linha pontilhada mostra a evolução das soluções obtidas pelo *VD-PLANARIZE* e obtidas pelo algoritmo de Even-Tarjan. A linha contínua escura representa as soluções para o algoritmo de Tarjan e a linha contínua clara representa o valor ótimo teórico que é de $min(m, n) - 2$ para grafos $C_n \times C_m$.

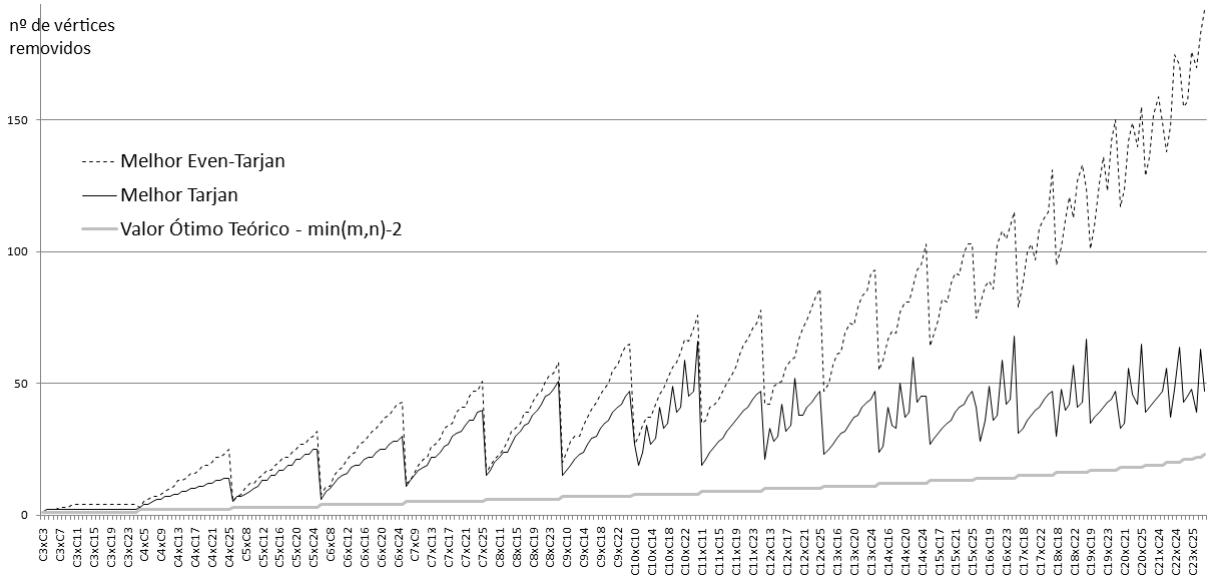


Figura 4.4. Gráfico com as curvas das melhores soluções encontradas para grafos $C_n \times C_m$

Observa-se na Figura 4.4 que o algoritmo de Tarjan gera *st*-numerações que proporcionam uma melhor planarização e que, quanto maior o grafo, a planarização obtida pelo uso das *st*-numerações geradas pelo algoritmo de Even-Tarjan encontram soluções inferiores em comparação com o algoritmo de Tarjan.

Entretanto o desempenho da planarização fica muito aquém do valor ótimo teórico para ambas as *st*-numerações. Este fato se deve aos dois fatores discutidos no Capitulo 3 – a gama de variações das *st*-numerações desconsideradas e a falta de garantias de que uma *st*-numeração é capaz de alcançar uma planarização ótima.

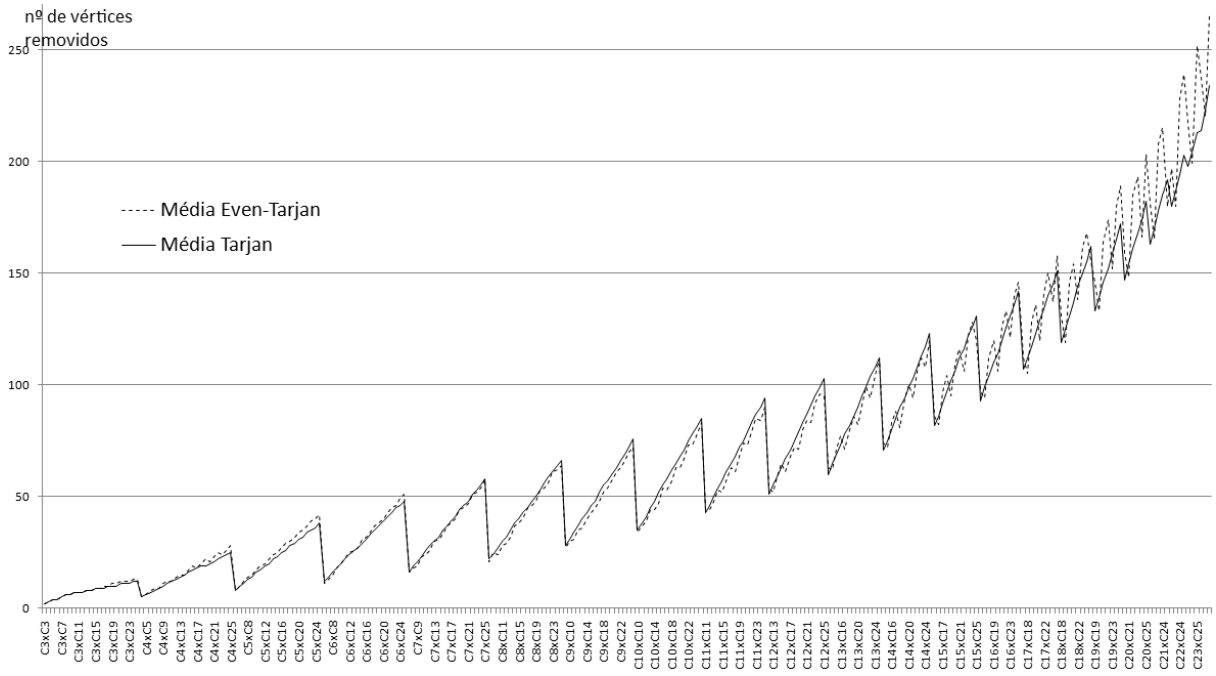


Figura 4.5. Gráfico com as curvas das soluções médias encontradas para grafos $C_n \times C_m$

A Figura 4.5 apresenta as curvas das soluções médias obtidas nos testes dos algoritmos de Tarjan e Even-Tarjan. Para gerar o gráfico, tomaram-se todas as soluções obtidas e calculou-se a média aritmética. Como se pode observar, os resultados médios de ambos os algoritmos foram muito similares, no entanto com o crescimento dos problemas, as soluções médias do algoritmo de Tarjan apresentaram uma pequena vantagem.

A Figura 4.6 apresenta um gráfico com um resumo dos testes, que exibe a soma de todos os resultados obtidos. O eixo x representa o conjunto de dados enquanto o eixo y representa a freqüência de número de vértices removidos nas respectivas soluções. As três colunas à esquerda representam a somatória dos resultados da melhor solução para ambos os métodos de *st*-numeração e para a solução ótima. As colunas centrais comparam as soluções médias de ambos os métodos de *st*-numeração e as colunas à direita comparam as piores

soluções encontradas pela planarização em conjunto com cada método de *st*-numeração.

Observa-se que apesar de ambos os algoritmos gerarem soluções médias parecidas, o algoritmo de Tarjan gera soluções melhores e piores do que o algoritmo de Even-Tarjan. Conclui-se que o método de Tarjan gera uma variedade maior de numerações possíveis para a classe de grafos $C_n \times C_m$.

O segundo conjunto de testes foi realizado com grafos randômicos. A geração desses grafos foi feita como segue. Para um grafo de n vértices, foi gerado o grafo ciclo C_n. Após a criação do grafo ciclo, para cada par de vértices u e v não adjacentes foi sorteado se a aresta $\{u, v\}$ deveria ser, ou não, adicionada ao grafo.

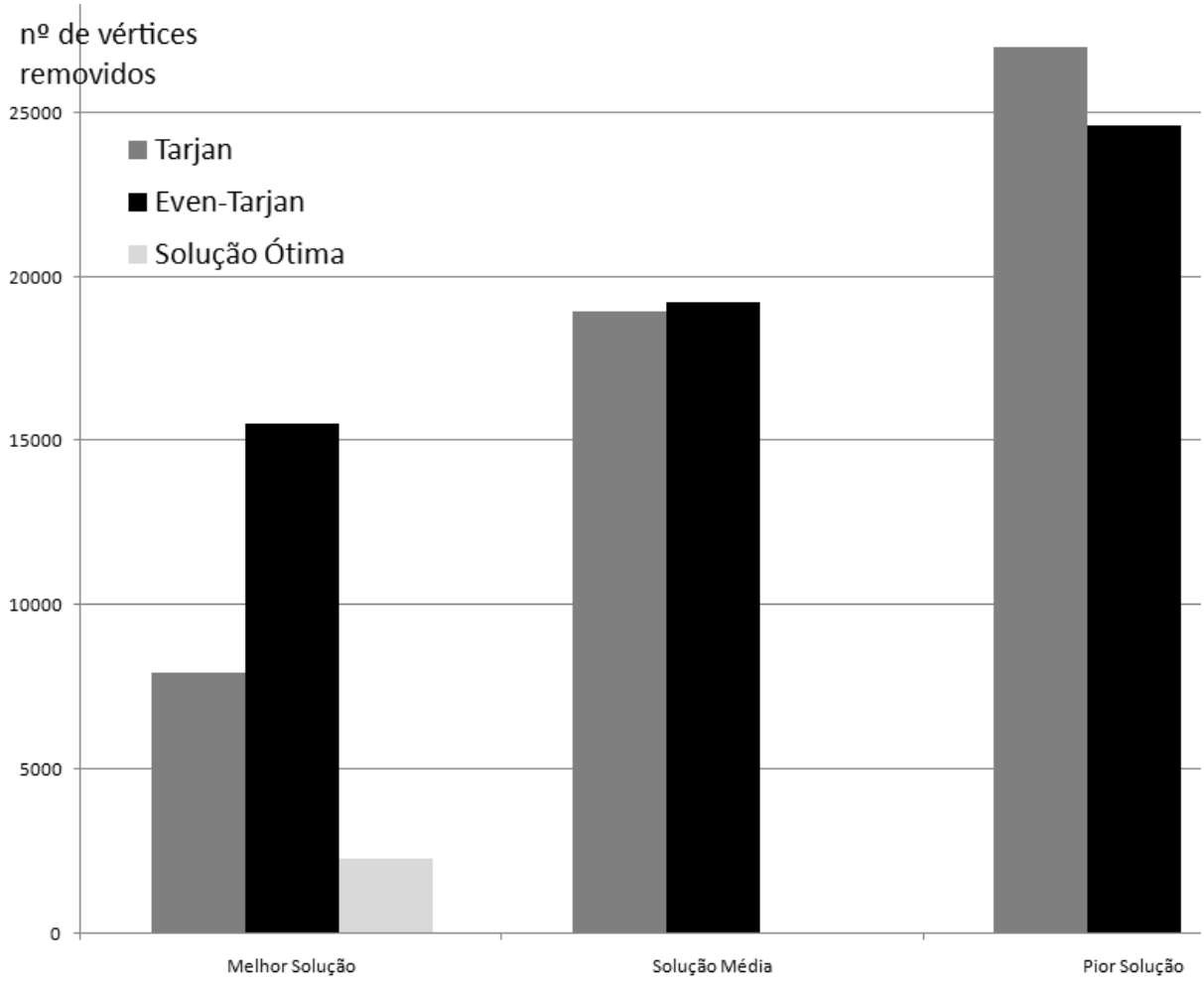


Figura 4.6. Gráfico do acumulado das soluções geradas pelo VD-PLANARIZE para grafos $C_n \times C_m$

Para realização dos testes foram gerados duzentos grafos de tamanhos aleatórios. Para cada grafo gerado foram obtidas, usando cada um dos algoritmos, *st*-numerações que totalizam duas vezes o número de arestas do grafo. E para cada *st*-numeração calculada o grafo foi planarizado pelo *VD-PLANARIZE* e o resultado catalogado.

A Figura 4.7 apresenta o gráfico da tabulação das melhores soluções obtidas pelo *VD-PLANARIZE* utilizando ambas as *st*-numerações. O eixo x representa o tamanho de $m + n$ dos grafos de teste enquanto o eixo y representa o número de vértices removidos. Observa-se que ambos os métodos tem desempenho muito parecidos, mas o algoritmo de Tarjan, representado pela linha contínua consegue se sobressair e apresentar uma pequena vantagem sobre o algoritmo de Even-Tarjan, representado pela linha pontilhada.

A Figura 4.8 apresenta o gráfico para as soluções médias obtidas. Analogamente ao gráfico anterior, o eixo *x* representa os valores de $m + n$ dos grafos de teste enquanto o eixo y representa o número de vértices removidos. Ao contrário dos exemplos anteriores, nesses testes o algoritmo de Even-Tarjan mostrou ter uma pequena vantagem sobre o algoritmo de Tarjan.

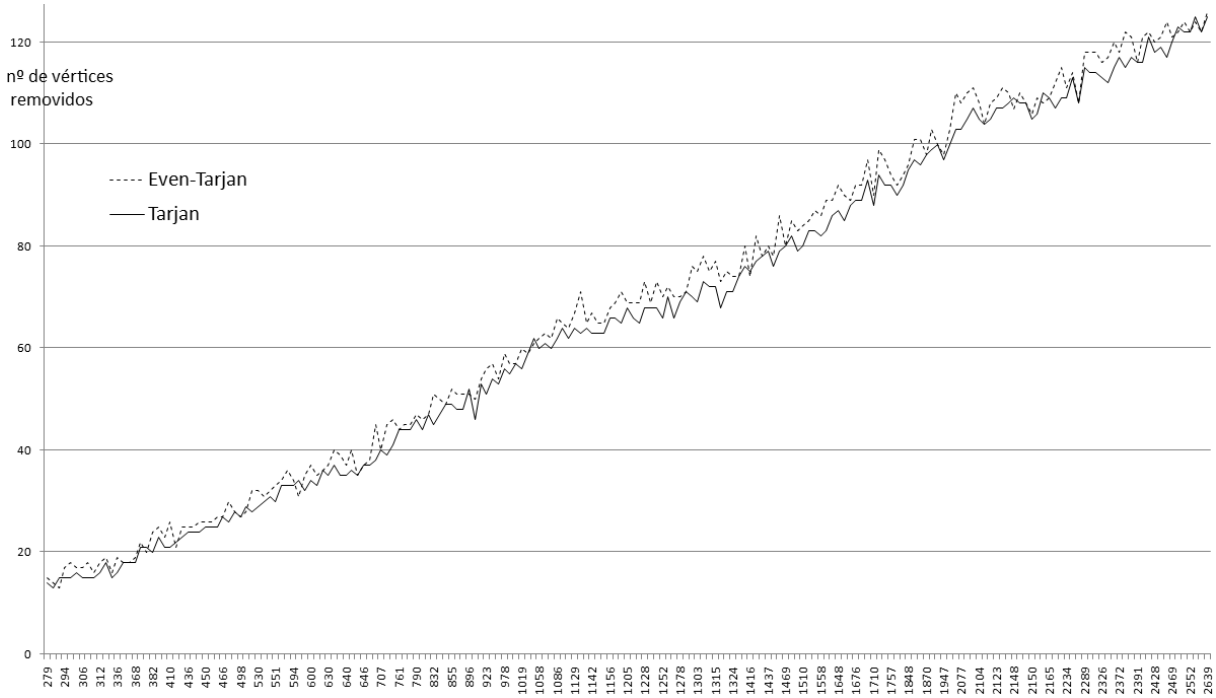


Figura 4.7. Gráfico com as curvas das melhores soluções encontradas para grafos randômicos

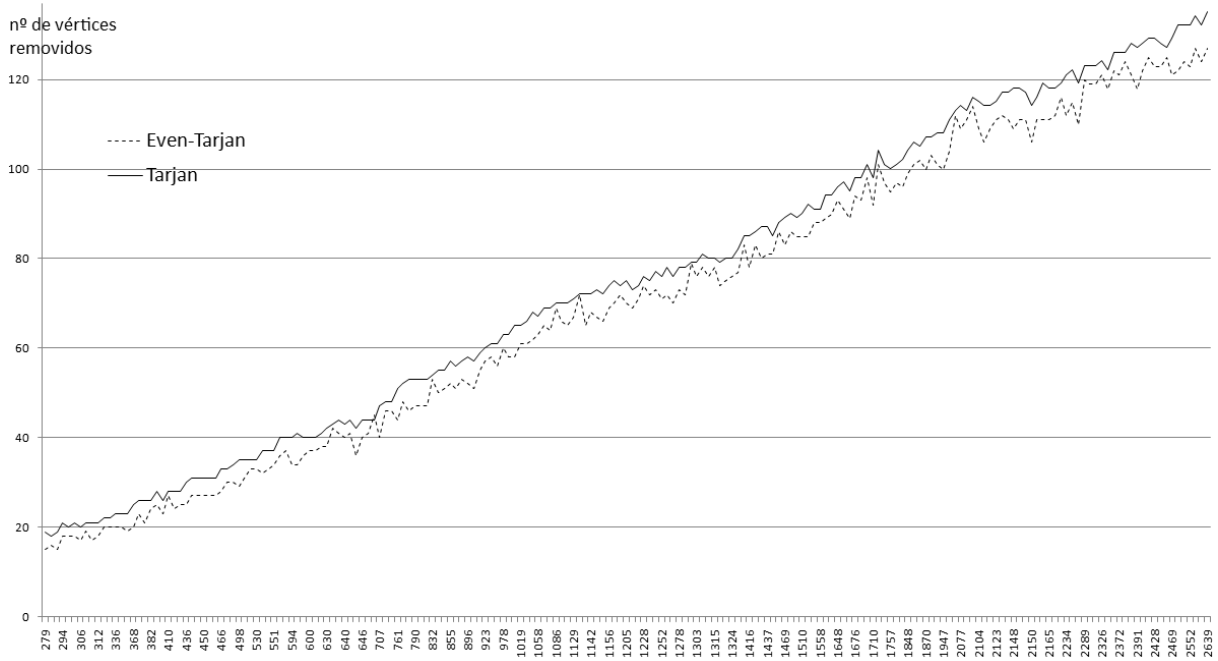


Figura 4.8. Gráfico com as curvas das soluções médias encontradas para grafos randômicos

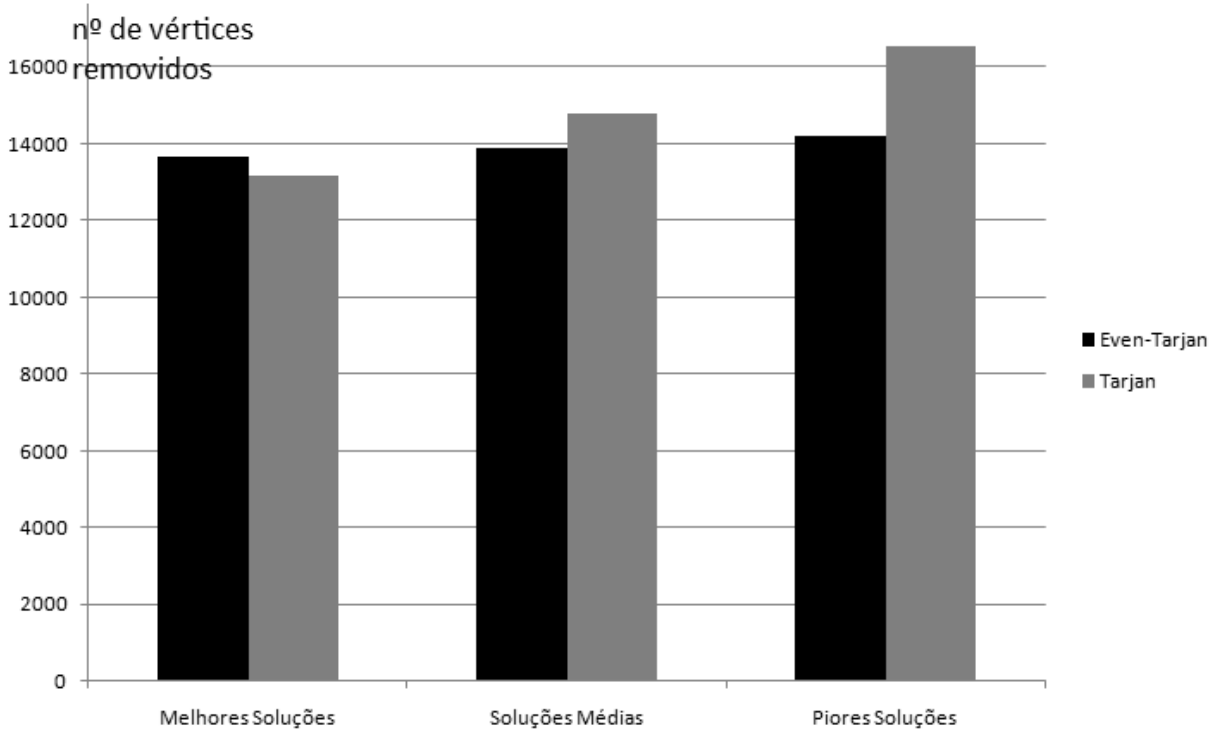


Figura 4.9. Gráfico do acumulado das soluções geradas pelo VD-PLANARIZE para grafos randômicos

A Figura 4.9 apresenta uma compilação dos resultados exibindo a soma das soluções de cada método para cada conjunto de dados. O eixo x representa os conjuntos de dados e é dividido entre os conjuntos das melhores soluções, das soluções médias e também das piores

soluções. O eixo y representa o número de vértices removidos pelo *VD-PLANARIZE* para cada método de *st*-numeração.

O gráfico mostra que o algoritmo de Tarjan, no geral encontra as melhores soluções, mas a média de soluções por ele encontrada é pior que o algoritmo de Even-Tarjan. O algoritmo de Tarjan também é responsável por encontrar as piores soluções.

O gráfico ainda mostra que a variação de soluções do algoritmo de Even-Tarjan é bem inferior à apresentada pelo algoritmo de Tarjan. Esta foi a razão pela qual o algoritmo proposto *GAVD-PLANARIZE* utilizou o algoritmo de Tarjan para realizar a *st*-numeração – a variedade de soluções é muito maior.

4.2. GAVD-PLANARIZE

Depois de concluídos os testes do algoritmo *VD-PLANARIZE*, foi notado que o grande espaço de soluções geradas por diferentes *st*-numerações, aliado a um baixo tempo de execução do algoritmo, propiciava o mesmo à aplicação de uma meta-heurística a fim de procurar por melhores soluções. Desta forma surgiu o algoritmo *GAVD-PLANARIZE*.

Os testes para a meta-heurística foram efetuados de maneira similar aos testes do *VD*-PLANARIZE - inicialmente realizou-se a medição do tempo e posteriormente avaliou-se a qualidade das soluções.

4.2.1. Tempo

Para determinar com precisão o tempo de execução do algoritmo *GAVD-PLANARIZE*, é necessário realizar duas medições. A primeira diz respeito ao aumento do tamanho do problema, pois ao aumentarmos o tamanho do grafo de entrada, aumentamos o tempo consumido pelo *VD-PLANARIZE* e, conseqüentemente, o tempo consumido pela meta-heurística. A segunda medição diz respeito ao tamanho da população escolhida.

Inicialmente tomaram-se grafos de tamanho crescente em intervalos de vinte e cinco unidades de $m+n$, determinou-se uma população de tamanho fixo em dez e condição de parada por estagnação em 30 e mediu-se o tempo de cada execução do algoritmo.

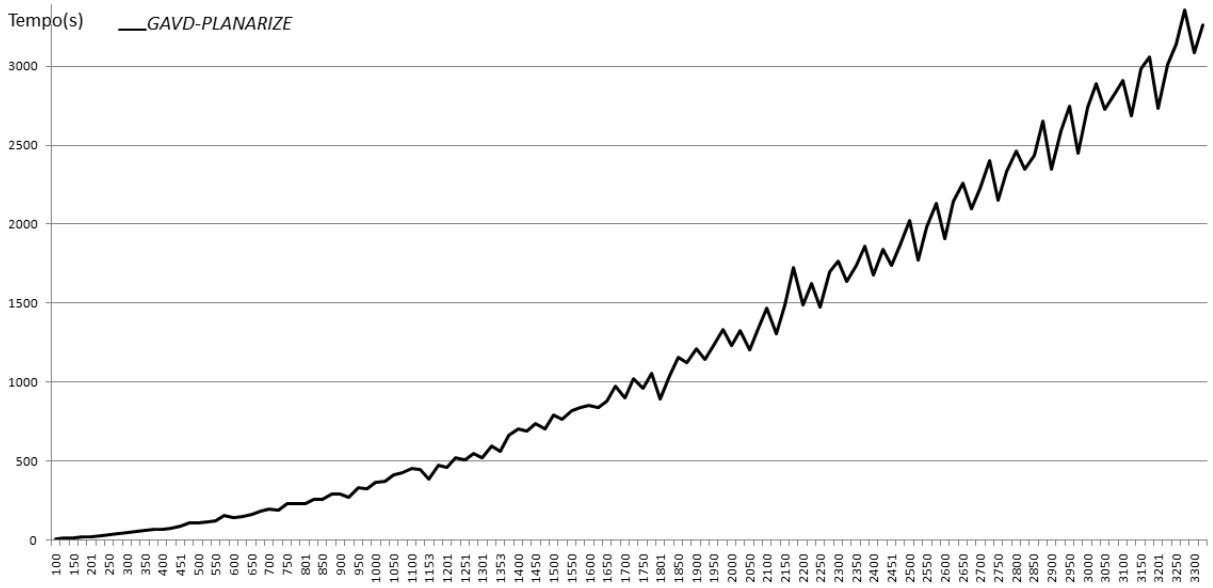


Figura 4.10. Curva do tempo de execução do algoritmo GAVD-PLANARIZE

A Figura 4.10 apresenta o resultado obtido do teste. O eixo x representa o tamanho do problema em unidades de $m + n$ enquanto o eixo y representa o tempo, em segundos, consumido pela execução do algoritmo. O gráfico mostra que a curva de execução de tempo do *GAVD-PLANARIZE* segue uma tendência polinomial, sendo que da metade em diante, apesar das oscilações, a tendência é quase linear.

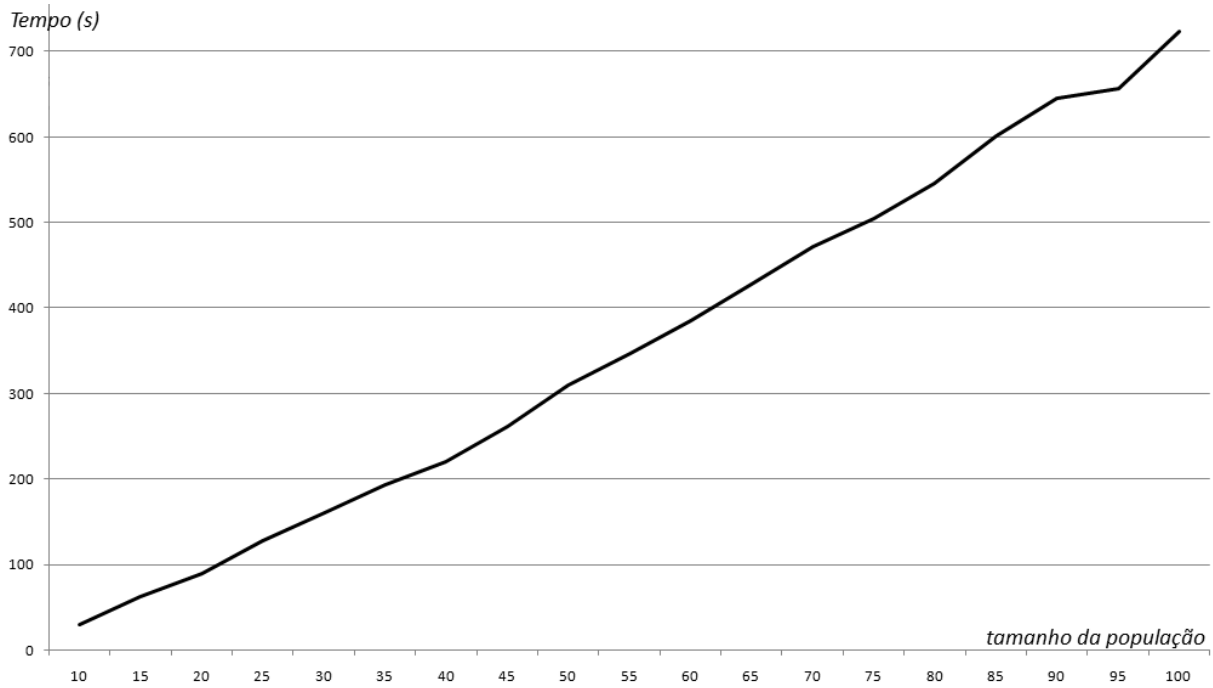


Figura 4.11. Gráfico com a curva de tempo da execução do GAVD-PLANARIZE para diferentes tamanhos de população

A Figura 4.11 apresenta os resultados obtidos do segundo teste de tempo realizado. Para este teste foi fixado um grafo de duzentos vértices e aumentou-se o tamanho da

população para cada execução do algoritmo, começando em dez e terminando em cem.

O eixo x representa o tamanho da população que cresce em incrementos de cinco e o eixo y representa o tempo em segundos. O gráfico apresenta uma curva de tempo muito próxima a linear.

4.2.2. Qualidade das Soluções

A medição da qualidade do *GAVD-PLANARIZE* se deu de maneira similar à do *VD-PLANARIZE*. Inicialmente testou-se o algoritmo com um conjunto de grafos $C_n \times C_m$ e posteriormente testou-se com grafos aleatórios. Em ambos os casos tomou-se o resultado dos testes do *VD-PLANARIZE* para o comparativo.

A Figura 4.12 apresenta os resultados obtidos do primeiro teste. O eixo *x* representa os grafos $C_n \times C_m$ e o eixo *y* representa o número de vértices removidos. O teste considerou grafos que variam do $C_3 \times C_3$ ao $C_{10} \times C_{25}$. A linha escura mostra a média obtida pelo *VD-PLANARIZE*, sendo este executado duas vezes para cara aresta e tendo a média dos resultados calculada. A linha clara representa as soluções obtidas pelo algoritmo *GAVD-PLANARIZE*.

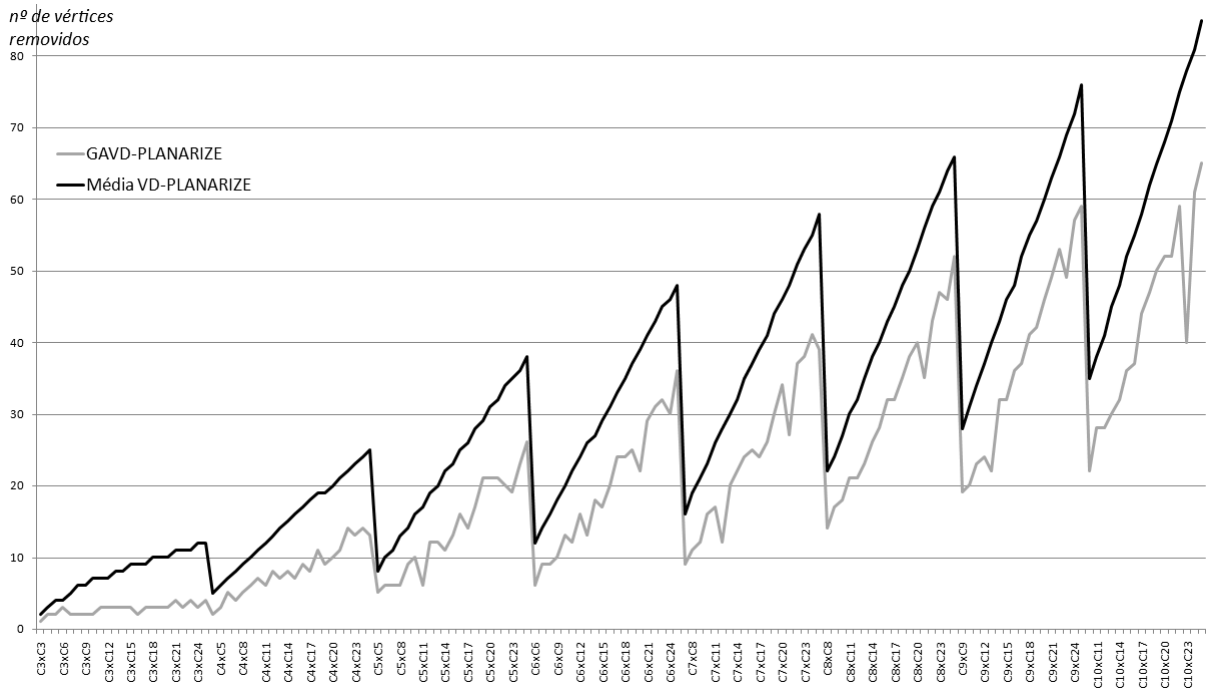


Figura 4.12 Gráfico com a curva das soluções encontradas pelo GAVD-PLANARIZE e das soluções médias do VD-PLANARIZE para grafos da classe $C_n \times C_m$

A meta-heurística proposta apresentou um desempenho superior ao valor médio do *VD-PLANARIZE* para grafos $C_n \times C_m$, como era de se esperar, no entanto o desempenho não se mostrou tão bom se tomado o melhor caso encontrado pela execução em "força-bruta" do

VD-PLANARIZE, como mostra a Figura 4.13.

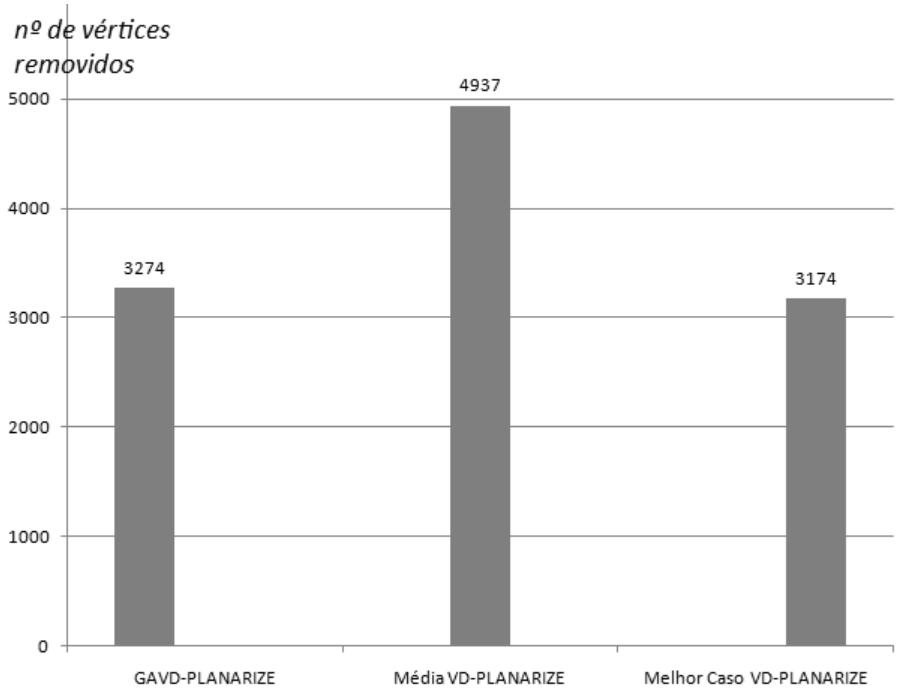


Figura 4.13. Gráfico do acumulado das soluções para grafos $C_n \times C_m$

A Figura 4.13 apresenta os resultados obtidos do acumulado dos testes anteriores. O eixo x representa os algoritmos enquanto o eixo y representa o número de vértices removidos. O gráfico foi gerado somando-se todos os resultados das planarizações dos grafos $C_n \times C_m$. O gráfico nos mostra que oi algoritmo proposto *GAVD-PLANARIZE* obteve um desempenho ligeiramente inferior à utilização da "força bruta" com o *VD-PLANARIZE*.

O segundo teste do *GAVD-PLANARIZE* foi realizado sobre grafos gerados randomicamente, como nos testes do *VD-PLANARIZE*. Os grafos foram obtidos da mesma forma. O estudo compreendeu duzentos grafos de tamanhos variando entre trinta e setenta e cinco vértices com valores aleatórios para α que variaram de vinte a cinqüenta por cento sorteados ao acaso.

A Figura 4.14 apresenta os resultados obtidos. O eixo *x* representa o tamanho do grafo em teste (número de vértices) e o eixo y representa o número de vértices removidos pela execução das planarizações. A linha escura representa a média obtida pela execução do *VD-PLANARIZE* tomando duas *st*-numerações por aresta enquanto a linha clara representa o resultado obtido pela meta-heurística proposta.

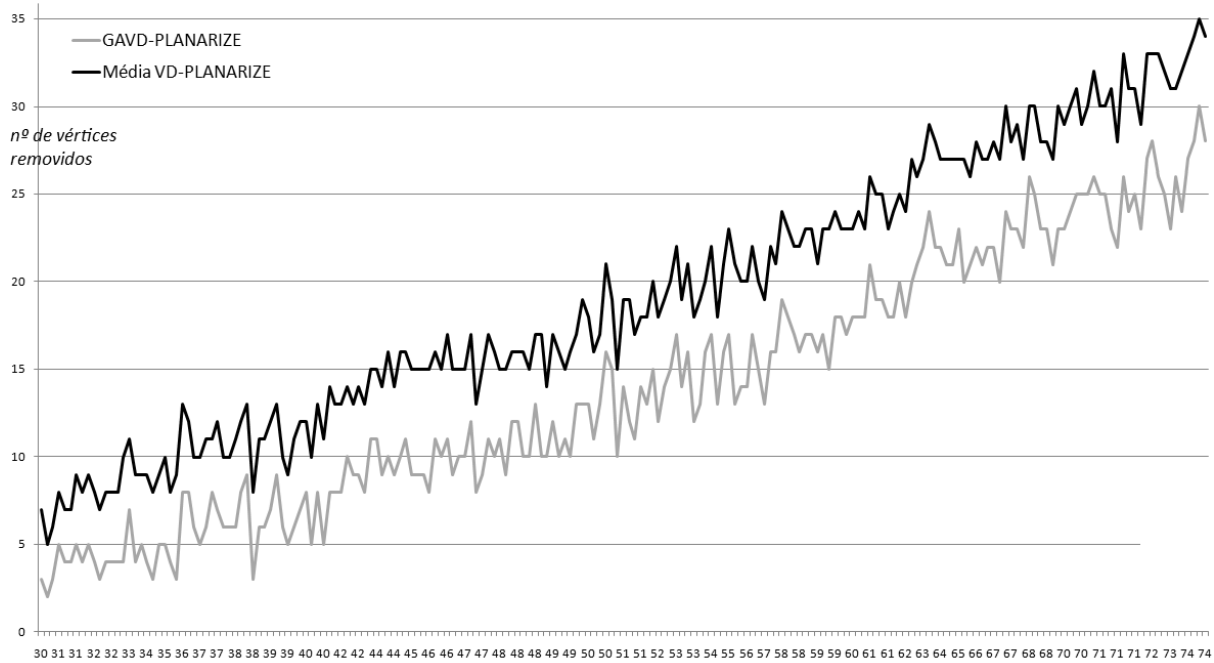


Figura 4.14. Gráfico com a curva das soluções encontradas pelo GAVD-PLANARIZE e das soluções médias do VD-PLANARIZE

A Figura 4.15 mostra o acumulado dos resultados obtidos, incluindo o acumulado das melhores soluções obtidas no *VD-PLANARIZE* com uso da "força bruta". As barras representam a soma do número de vértices removidos em todos os grafos. Observamos que a meta-heurística proposta apresenta resultados não apenas melhores que a média, mas também melhores que o uso da "força bruta" pelo *VD-PLANARIZE*.

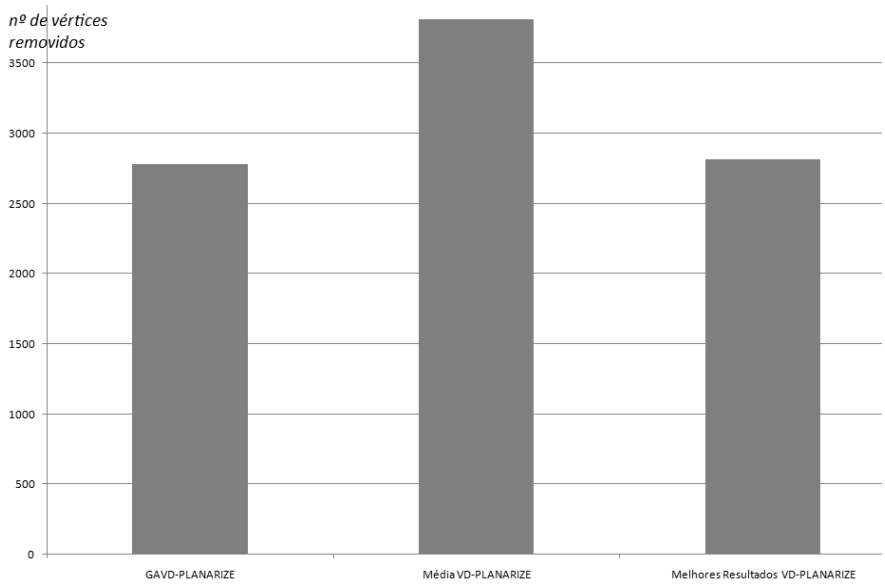


Figura 4.15. Gráfico do acumulado das soluções para grafos randômicos

Observamos dos testes que a meta-heurística proposta não teve um excelente desempenho para grafos da classe $C_n \times C_m$, no entanto conseguiu obter melhores resultados

com grafos randômicos. Isto se deve pela natureza do problema.

Para grafos da classe $C_n \times C_m$, uma peculiaridade ocorre. Sem perda de generalidade, para $n = m$, não importa qual aresta o algoritmo de *st*-numeração tome por aresta inicial, pois devido a simetria entre todos os vértices e arestas, todas as possibilidades de *st*-numerações podem ser alcançadas a partir de uma única aresta.

A Figura 4.16 exemplifica esse caso. Observa-se que a aresta $\{v_5, v_6\}$ em destaque em (a) pode corresponder à aresta $\{v_3, v_2\}$ do grafo de (b) se rotularmos os vértices do grafo de (b) como mostrado em (c), dessa forma obtendo-se a mesma *st*-numeração, pois todos os vértices são adjacentes aos mesmos vértices nos dois grafos.

Analogamente, para grafos $C_n \times C_m$ onde $m \neq n$, existem duas opções de arestas, as arestas de um ciclo C_n ou as arestas de um ciclo C_m.

Nos grafos da classe $C_n \times C_m$, o número de *st*-numerações produzidas por ambos os algoritmos estudados é bem reduzido, visto que não depende da escolha da aresta a se tomar como base, mas apenas da ordem de visita dos vértices o que influencia na formação da árvore de busca.

Quando as *st*-numerações são geradas, duas para cada aresta, a variação nas soluções geradas pelo *VD-PLANARIZE* não se dá devido às diferentes arestas, mas como a lista de adjacências não está ordenada, a *st*-numeração em diferentes arestas acaba formando árvores de buscas diferentes devido a variação na ordem de visita dos vértices, portanto o algoritmo acaba percorrendo diversas áreas do espaço de soluções de um espaço muito menor, enquanto a meta-heurística por se mais genérica, perde tempo procurando a melhor aresta.

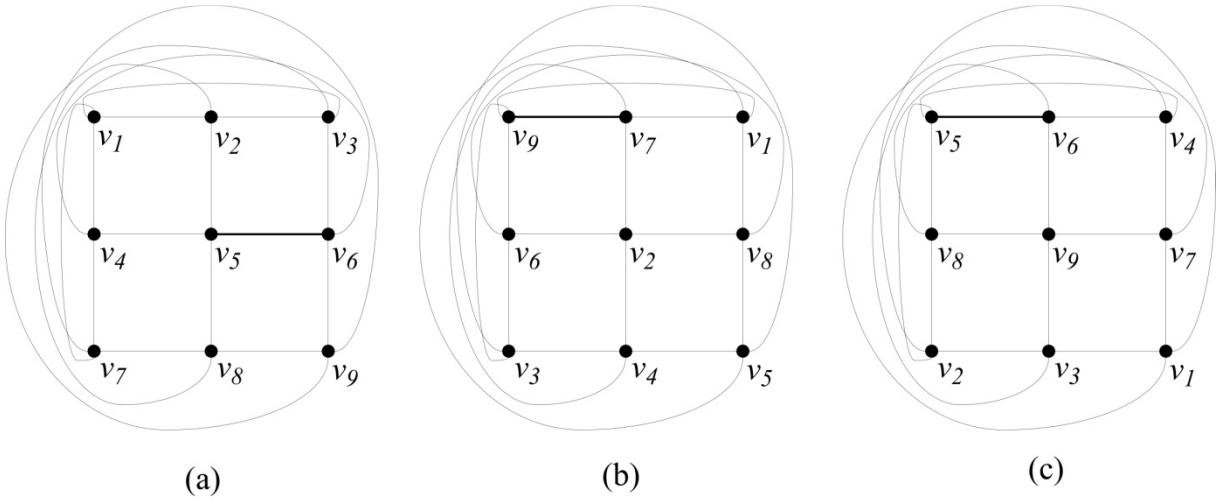


Figura 4.16. Exemplo uma mesma st-numeração partindo de arestas diferentes

Já nos grafos aleatórios onde não existe simetria alguma, o espaço de soluções acaba sendo bem maior e a meta-heurística consegue tirar proveito disso não apenas procurando

pela melhor aresta para se realizar a *st*-numeração, mas também procurando a melhor combinação da ordem de visita dos vértices para a geração da árvore.

CAPÍTULO

5

Conclusão

A área de planarização de grafos é uma área amplamente estudada devida as suas aplicações práticas e a sua complexa teoria. A maioria dos grafos que modelam problemas da vida real é do tipo não planar, portanto estudos acerca da planaridade de grafos e formas de melhor representá-los são cada vez mais comuns.

Mas para derivar um grafo G, não planar, em um grafo correspondente G', planar, tem-se que contornar os seguintes problemas:

- só é possível planarizar um grafo alterando sua estrutura; e
- para cada operação que altera a estrutura de um grafo que torna possível a planarização, existe uma invariante de não-planaridade associada. Planarizar um grafo dado uma operação é um problema *NP*-completo.

Este trabalho apresenta dois algoritmos para resolver o problema de planarização de grafos utilizando a operação de remoção de vértices. O primeiro é uma heurística que possui complexidade de tempo linear e o segundo é um algoritmo genético que busca otimizar as soluções obtidas pelo primeiro algoritmo. Na literatura não se tem conhecimento de outros trabalhos que utilizam a operação de remoção de vértices para planarizar um grafo, tampouco trabalhos que possuam complexidade de tempo linear para esse problema.

5.1. Contribuições

Utiliza-se neste trabalho a *st*-numeração para determinar a ordem de visita dos vértices do grafo no processo de planarização e sugere-se o uso de dois algoritmos propostos por Even e Tarjan (1976) e por Tarjan (1986). Ambos possuem complexidade de tempo linear. Apresenta-se resultados de planarizações realizadas a partir de *st*-numerações geradas pelos dois algoritmos. Os resultados são comparados e conclui-se que o algoritmo de Tarjan gera uma maior variedade de *st*-numerações.

Propõe-se o algoritmo *VD-PLANARIZE*, uma heurística baseada no teste de planaridade de Lempel, Even e Cederbaum (1967) que utiliza árvores-*PQ* para verificar a planaridade de um grafo. Ressalta-se que esse algoritmo possui complexidade de tempo linear e até o momento não é conhecido nenhum outro algoritmo de planarização que apresente complexidade igual ou inferior. Além disso, o *VD-PLANARIZE* é o único algoritmo de planarização até o momento que utiliza a operação de remoção de vértices.

Este trabalho também propõe uma metaheurística genética, o *GAVD-PLANARIZE*, para explorar diferentes *st*-numerações a fim de otimizar as soluções encontradas pelo *VD-PLANARIZE*. Para isso ele testa diferentes ordens de visita dos vértices na geração da *st*-numeração.

Outra importante contribuição deste trabalho é a implementação de ambos os algoritmos e a realização de testes para diferentes classes de grafos com o intuito de constatar a complexidade de tempo e a qualidade das soluções.

Finalmente, este trabalho contribui com os resultados dos testes realizados. Esses testes comprovam experimentalmente a complexidade teórica de $O(m+n)$ do *VD-PLANARIZE* e também o impacto que diferentes *st*-numerações de um mesmo grafo podem ter sobre a planarização. Os testes ainda comprovam que o algoritmo *GAVD-PLANARIZE* é capaz de explorar o espaço de soluções do *VD-PLANARIZE* a ponto de apresentar resultados muito bons e um pequeno tempo de execução.

5.2. Trabalhos Futuros

Alguns aspectos do algoritmo genético proposto podem ainda ser investigados. Por exemplo, as configurações da taxa de mutação e testes detalhados acerca do tamanho das populações. Outras abordagens metaheuristicas ainda podem ser abordadas a fim de se comparar os resultados, como *simulated annealing* e *GRASP*.

Referências Bibliográficas

BARNETTE, D. W.; EDELSON, A. L. All Orientable 2-Manifolds Have Finitely Many Minimal Triangulations. *Israel J. od Mathematics*, vol. 62, p. 90-98, 1988.

BONDY, J. A.; MURTY, U. S. R. *Graph Theory with Applications*. New York. North Holland, 1976 p. 12–21.

BOOTH, K. S.; LUEKER, G. S. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-TREE algorithms. *Journal of Computer Science and System Sciences*, vol. 13, p. 335-379, 1976.

BUENO, L. R. *Sobre Redução de Cruzamento de Arestas em Desenho Linear*. 2005. 89 f. Dissertação de Mestrado, Universidade Estadual de Maringá, Maringá, 2005.

CHIBA T; NISHIOKA I.; SHIRAZAWA I.An Algorithm of Maximal Planarization of Graphs. *Proc. 1979 IEEE Int. Symp. on Circuits and Systems*, p. 648–652, 1979.

EADES, P.; MENDONÇA, C. F. X. Heuristics for planarization by vertex splitting. *Proc. ALCOM Int. Workshop on Graph Drawing*, GD'93, p. 83–85, 1993.

EBERT, J. st-ordering the vertices of biconnected graphs. *Computing*, vol. 30, p. 19-33, 1983.

EVEN, S.; TARJAN, R. E. Computing and st-numbering. *Theoretical Computer Science*, vol. 2, p. 339-344, 1976.

EVEN, S.; TARJAN, R. E. Corrigendum: Computing and st-numbering. *Theoretical Computer Science*, vol. 4, p. 123, 1977.

FARIA, L.; DE FIGUEIREDO, C. M. H.; DE MENDONÇA NETO, C. F. X. Splitting number is NP-Complete. *Discrete Applied Mathematics*, vol. 108, p. 65-83, 2001.

FISHER G.J.; WING O. Computer Recognition and Extraction of Planar Graphs from the Incidence Matrix. *IEEE Trans. Circuit Theory*, CT-13, p. 154–163, Jun. 1966.

GAREY, M.R.; JOHNSON, D. S. *Computers and Intractability,* San Francisco. W.H. Freeman, 1979. 340 p.

GAREY, M. R.; JOHNSON, D. S. Crossing number is NP-Complete. *SIAM Journal on Algebric and Discrete Methods* vol. 4, issue 3, p. 312-316, set. 1983.

GIBBONS, A. *Algorithmic Graph Theory*. Cambridge, Cambridge University Press, 1984. 272 p.

GLOVER, F.; KOCHENBERGER, G. A. *Handbook of metaheurístics.* Springer, 2003. 570 p.

GOLDBERG, D. E. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989. 432 p.

GUY, R. K. Latest results on crossing numbers. *Lecture Notes in Mathematics,* vol. 186, p. 143-156, 1971.

HARARY, F. *Graph Theory*. Reading, MA. Addison-Wesley, 1994. p. 22.

HARARY, F.; KAINEN, P. C.; SCHWENK, A. J. Toroidal graphs with arbitrarily high crossing number. *Nanta Mathematics*, vol. 6, p. 58-67, 1973.

HARRIS J. A Graphical Java Implementation of PQ-Trees. *Advanced Topics in Data Structures* vol. *95.590X.* Abr. 2002.

HOBBS, A. M.. A survey of thickness. *3rd Waterloo Conference on Combinatorics*, p. 255-264, mai. 1969.

JAYAKUMAR, R.; THULASIRAMAN, K.; SWAMY, M.N.S. O(n2) algorithms for graph planarization. *IEE Transaction on Computer-Aided Design*, vol. 8, issue 3, p. 257-267, mar. 1989.

KANT, G. An $O(n^2)$ maximal planarization algorithm based on PQ-trees. *Technical Report RUU-CS-92-0*, Department of Information and Computing Sciences, Utrecht University, 1992.

KLEITMAN, D. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory*, vol. 9, p. 315-323, 1970.

KURATOWSKI, K. Sur le probleme dès courbes gauches en topologie, *Fundamenta Mathematicae*, vol. 15, p. 271-283, 1930.

LEIGHTON, F. T. New lower bound techniques for VLSI. *Math. Systems Theory*, vol. 17, p. 47-70, 1984.

LEMPEL, A.; EVEN, S.; CERDERBAUM, I. An Algorithm for Planarity Testing of Graphs, *Theory of Graphs: International Symposium (Rome 1966)* (New York, 1967), Gordon and Breach, p. 215-232, 1966.

LEWIS, J. M.; YANNAKAKIS, M. The node-deletion problem for hereditary properties is

NP-Complete. *Journal of Computer and System Sciences,* vol. 20, p. 219-230, 1980.

LIEBERS, A. Planarizing graphs - a survey and annotatted bibliography. *Journal of Graph Algorithms and Applications,* vol. 5, issue 1, p. 1-74, 2001.

LIU, P. C.; GELDMACHER, R. C. On the deletion of nonplanar edges of a graph. *10th South East Conference on Combinatorics, Graph Theory, and Computing*, pp. 727-738, 1979.

MANSFIELD, A. Determining the thickness of graphs is NP-Hard. *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 93, issue 1, p. 9, 1983.

MEHLHORN, K.; MUTZEL, P. On the embedding phase of the Hopcraft and Tarjan planarity testing algorithm. *Algorithmica*, vol. 16, issue 2, p. 233-242, ago. 1996.

DE MENDONÇA NETO, C. F. X. *A Layout System for Information System Diagrams*. Tese de Doutorado, University of Queensland, Department of Computer Science, 1994.

DE MENDONÇA NETO, C. F. X.; SCHAFFER, K.; XAVIER, E. F.; STOLFI, J.; FARIA, L.; DE FIGUEIREDO, C. M. H. The splitting number and skewness of $C_n \times C_m$. *ARS Combinatoria*, vol. 63, p. 193-205, 2002.

MITCHELL, M. *An Introduction to Genetic Algorithms*. Cambridge, MA. MIT PRESS, 1998. 221 p.

MITCHELL, M; FORREST, S. Genetic algorithms and artificial life. *Artificial Life,* vol. 1, no. 3, p. 267-289, 1995.

MUTZEL, P; ODENTHAL, T.; SCHARBRODT, M. The Thickness of Graphs: A Survey.. *Graphs and Combinatorics*, vol. 14, p. 59-73, 1988.

NISHIZEKI, T; CHIBA, N. Planar Graphs: Theory and Algorithms. *Discrete Mathematics*, vol. 32, North Holland Mathematical Studies, 1988.

OZAWA T; TAKAHASHI H. A Graph-Planarization Algorithm and its Applications to Random Graphs. *Graph Theory and Algorithms, Lectures Notes in Computer Science*, Springer-Verlag, vol. 108, p. 95–107, 1981.

PAPAMANTHOU, C. *Computing longest path parameterized st-orientations of graphs: algorithms and applications.* Master's thesis, University of Crete, Heraklion, Greece, Jul. 2005.

PAPAMANTHOU, C. TOLLIS, I. G. Algorithms for computing a parametrized st-orientation. *Theor. Comput. Sci.*, vol. 408, p. 224-240, 2008.

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

SADOWSKA M. M. Planarization Algorithm for Integrated Circuits Engineering. *Proc. 1978 IEEE Int. Symp. on Circuits and Systems,* vol. 1, p. 919–923, 1978.

SALAZAR, G. On the crossing number of $C_n \times C_m$. *Journal of Graph Theory*, vol. 28, p. 163-170, 1998.

STEINITZ, E.; RADEMACHER, H. Vorlesungen uber die Theorie der Polyeder Einschlu der Elemente der Topologie, *Die Grundlehren der Mathematischen*, vol. 41, 1934.

TARJAN, R. E. Two streamlined depth-first search algorithms. *Fundamenta Informaticae*, vol. 9, p. 85-94, 1986.

VOLLEN, G. *PQ-Trees and Maximal Planarization: an approach to skewness*. Tese de Doutorado, University of Solo, Department of Informatics, February 1998. 168 p.

WATANABE, T.; AE, T.; NAKAMURA, A. On the NP-Hardness of edge-deletion and contraction problems. *Discrete Applied Math.* vol. 6, p. 63-78, 1983.

WHITE, A. T.; BEINEKE, W. Topological Graph Theory. *Selected Topics in Graph Theory*. p. 15-49, 1978.

WOODALL, D. R. Cyclic-order graphs and Zarankiewics's crossing number conjecture. *Journal of Graph Theory*, vol. 17, p. 657-671, 1993.

XAVIER, E. F. *Invariantes de planaridade*. Dissertação de Mestrado, Universidade Estadual de Campinas, Instituto de Computação, dez. 1999.

YANNAKAKIS, M. Node- and edge-deletion NP-Complete problems. *10th Annual ACM Symposium on Theory of Computing*, STOC-78, p. 253-264, 1978.

ZARANKIEWICZ, K. On a problem of p. turán concerning graphs. *Fundamenta Informaticae*, vol. 41, p. 137-145, 1954.