O PROBLEMA DO SUBGRAFO
PLANAR ÓTIMO

Renato J. S. Carmo

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: Ciência da Computação
Orientador: Prof^ª Dr^ª Yoshiko Wakabayashi

— São Paulo, maio de 1994 —

O PROBLEMA DO SUBGRAFO PLANAR ÓTIMO

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Renato J. S. Carmo e aprovada pela comissão julgadora.

São Paulo, 05 de junho de 1994

Banca examinadora:

- Profa. Dra. Yoshiko Wakabayashi (orientadora) - IME-USP
- Prof. Dr. Arnaldo Mandel - IME-USP
- Prof. Dr. Cláudio L. Lucchesi - IMECC-UNICAMP

Resumo

Este trabalho trata do problema de determinar um subgrafo planar de peso máximo de um grafo dado. Uma demonstração simplificada da $\mathcal{NP}$ -Compleitude do problema é apresentada. Descrevemos uma implementação alternativa do conhecido algoritmo de teste de planaridade de Hopcroft/Tarjan, adequada para uso como subrotina reiteradamente invocada sobre um grafo não necessariamente biconexo. Baseados nessa abordagem, propomos uma implementação não-ingênua do *método guloso* para a determinação de soluções aproximadas do problema, cujo tempo de execução é significativamente menor que o das implementações mencionadas na literatura. Resultados experimentais dessa implementação são exibidos e analisados. O *limite de desempenho absoluto* para o método guloso aplicado ao problema do subgrafo planar ótimo é determinado. Outras heurísticas encontradas na literatura são apresentadas de maneira unificada e analisadas comparativamente. Uma parte do trabalho é dedicada a uma detalhada descrição do algoritmo de teste de planaridade, com a finalidade de servir de texto de referência para algoritmos baseados na estratégia de Hopcroft/Tarjan. Nessa discussão, introduzimos a idéia de *hierarquias* como uma interessante ferramenta conceitual para a descrição formal das propriedades de uma *Busca em Profundidade* num grafo.

Abstract

In this work we are concerned with the problem of finding a maximum weight planar subgraph of a given graph. An original and simplified proof of the $\mathcal{NP}$ -Completeness of the problem is presented. We describe an alternative implementation of the well-known Hopcroft/Tarjan's planarity testing algorithm, designed to be useful as a subroutine to be run several times on non-necessarily biconnected graphs. Based on this approach, we propose a non-naïve implementation of the *greedy scheme* whose running time is significantly lower than the ones achieved by the implementations so far mentioned in the literature. Experimental results of this implementation are shown and analysed. The *absolute performance ratio* of the greedy approach to the maximum planar subgraph problem is determined. Other heuristics found in the literature are presented under a unified frame and comparatively analysed. Part of the work presents a full-detailed description of the planarity testing algorithm, with the aim of providing a reference text to algorithms based on Hopcroft/Tarjan's approach. In doing so we introduce the idea of *hierarchies* as a useful conceptual tool to describe formally *Depth-First-Search* properties of a graph.

*Aos meus pais, Paulo e Terezinha,
que me ensinaram todas as coisas
importantes que fui capaz de aprender.*

Conteúdo

Introdução	v
Preliminares	xi
Conjuntos	xi
Relações e Funções	xi
Grafos	xiv
 I Planaridade	 1
1 Definições e Resultados Preliminares	3
2 Um Algoritmo para Teste de Planaridade	11
3 Hierarquias e Busca em Profundidade	23
3.1 Definições e Resultados Preliminares	24
3.2 Hierarquias e Planaridade	38
4 Os Pingos nos 11	49
4.1 Um Exemplo de Execução	49
4.2 Implementação	54
4.3 Demonstrações dos Resultados Mencionados	63

II	O Problema do Subgrafo Planar Ótimo	71
5	Complexidade Computacional	73
6	Soluções e Heurísticas Propostas	77
6.1	Resultados Experimentais Disponíveis	84
7	O Método Guloso	89
7.1	Garantias de Otimalidade	90
7.2	Uma Alternativa para a Implementação	97
7.3	Resultados Experimentais	103
	Conclusão	109

Agradecimentos

São muitos e bem merecidos:

Antes e mais que qualquer outro, à Yoshiko, pela sua orientação, pela sua amizade, pela sua atenção, pela sua paciência, pela sua perseverança, pelo seu apoio, pelo encorajamento e por tantas outras coisas que as palavras são pouco para expressar.

Em seguida ao Fábio, meu “alter ego” no IME, capaz de quebrar uma quantidade exponencial de galhos em tempo real. Também aqui, difícil é expressar a intensidade da gratidão pela sua amizade e fidelidade.

Ao Carlinhos, capaz de ser um apoio moral e material decisivo mesmo com o Atlântico e meia Europa de permeio.

Ao Elói e ao Ueda, sempre prontos a resgatar a referência bibliográfica mais obscura, ou qualquer outra missão impossível.

Ao Flávio Keidi, pela atenção e tempo dedicados aos assuntos que discutimos, em particular no que diz respeito ao Lema 7.5.

Ao Buzzi, por toda ajuda e aconselhamento nos trâmites técnicos e aduaneiros por ocasião da aquisição do micro-computador que tornou viável a conclusão deste trabalho em tempo hábil.

Aos meus colegas do Departamento de Informática da UFPR, pelo decisivo apoio manifestado ao me conceder o período de afastamento das obrigações administrativas e docentes no período em que a parte substancial deste trabalho foi realizada. Dentre esses, uma menção muito especial à Wanda pelo interesse, apoio, amizade desde que nos conhecemos (o que inclui a amigável “cobrança” de que fui vítima nesse período); também ao André pela atenção e boa vontade com que sempre se dispôs a trocar idéias e dar sugestões em aspectos técnicos e menos técnicos; e ao Urban, pelo interesse amigo que sempre manifestou pelo trabalho e pela pessoa que o realizou.

Ao Heraldo pela sua dedicação (abnegação?) em instalar e manter “no ar” o `inf.ufpr.br` nas condições mais adversas que se possa imaginar e ao(s) responsável(is) pela manutenção do `ime.usp.br` pelo serviço inestimável.

Aos funcionários da Biblioteca Carlos Benjamin de Lyra e da Comissão de Pós-Graduação do

IME, sempre compreensivos e atenciosos nos trâmites exóticos a que algumas vezes fui obrigado a recorrer nesse período.

A todos os que, em me encontrando casualmente no período em que esse trabalho vinha sendo realizado, me perguntaram a respeito do seu andamento, pela consideração e estima manifestados.

A todos os que, em me encontrando casualmente no período em que esse trabalho vinha sendo realizado, não me perguntaram nada a respeito, pela grande consideração e estima manifestados.

A todos os que conviveram comigo nesse período e toleraram meu humor nos “dias difíceis”.

Muito Obrigado!

Introdução

A proposta original deste trabalho foi um estudo do **Problema do Subgrafo Planar Ótimo**, (mais) um problema $\mathcal{NP}$ -Completo de Otimização Combinatória que pode ser brevemente assim formulado:

Dado um grafo G com pesos nas arestas, determinar um subgrafo planar de G com peso máximo.

Nossa idéia inicial era a de fazer um estudo que envolvesse a experiência de implementação de algoritmos e análise de seu comportamento, comparação entre heurísticas relatadas na literatura e eventualmente a proposição de novas heurísticas ou variações das analisadas.

Fosse como fosse, uma coisa era clara desde o princípio: seria necessário implementar algum dos algoritmos eficientes para verificação de planaridade disponíveis na literatura. Assim, ao mesmo tempo em que iniciamos nosso levantamento bibliográfico a respeito do problema de otimização, lançamo-nos à tarefa de estudar e implementar o conhecido algoritmo de Hopcroft e Tarjan.

As conclusões a que chegamos nessa fase não foram exatamente o que esperávamos, e acabaram decidindo o rumo futuro que o trabalho como um todo iria tomar.

Por um lado, o levantamento bibliográfico em torno do Problema do Subgrafo Planar Ótimo revelou-nos uma quantidade pequena de referências, a maioria delas já um tanto antigas. Entretanto, as heurísticas relatadas nessas referências apresentavam um desempenho surpreendentemente bom e, mais ainda, dentre elas a conceitualmente mais simples, o “*Método Guloso*”, apresentava os melhores resultados. Curiosamente, nenhum resultado teórico justificando tal desempenho foi encontrado. Além disso, a demanda de tempo computacional dessa alternativa (devida aos repetidos testes de planaridade que precisam ser efetuados) de certa forma desqualificava sua viabilidade para instâncias realistas do problema como ficava evidenciado nos trabalhos publicados até então.

Por outro lado, a implementação do algoritmo para teste de planaridade revelou-nos duas lacunas interessantes:

Em primeiro lugar que, apesar de célebre e profusamente citado na literatura, não pudemos encontrar descrições do algoritmo de Hopcroft e Tarjan detalhadas ao ponto de permitir uma implementação, que dispensassem a leitura do artigo onde é originalmente proposto. Isto não chegaria a constituir um problema, não fosse pelo fato de que o referido artigo é de leitura um tanto árida. De fato, vários autores ao se referirem a esse algoritmo o reputam como complexo e de difícil implementação; mesmo após sua publicação, outros algoritmos de maior complexidade computacional continuaram a ser publicados sob o argumento de que representavam uma solução de compromisso entre a facilidade de implementação e o desempenho.

Em segundo lugar, esse algoritmo, tal como proposto originalmente, mostra-se muito pouco adequado ao uso como uma subrotina que deve ser invocada repetidas vezes recebendo como argumento um grafo não necessariamente biconexo, o que nos levou a formular uma série de adaptações na nossa implementação, e conseqüentemente a um estudo mais profundo do seu funcionamento.

Diante desse quadro concentramos nossas energias em formular uma implementação alternativa do “*Método Guloso*” que interagisse de maneira mais íntima com o algoritmo de teste de planaridade a fim de diminuir seu tempo de execução.

O resultado final dessa experiência, que apresentamos neste trabalho pode ser resumido nos seguintes itens:

- Uma reorganização das idéias por trás do algoritmo de Hopcroft e Tarjan, que conduz a uma variante de implementação de sua proposta original, esta por sua vez adequada para uso como subrotina e sem qualquer restrição quanto à estrutura do grafo recebido como argumento.

Da forma como apresentada aqui, esperamos que a exposição do algoritmo constitua uma referência didática útil a um eventual leitor interessado em compreender com profundidade e/ou implementar o algoritmo, e ao mesmo tempo seja vista como uma proposta de formalização mais rigorosa que lance maiores luzes sobre seu funcionamento.

Nessa exposição, introduzimos e fazemos uso extensivo da idéia de *Hierarquias*, uma ordem parcial com propriedades particulares que adequadamente imposta sobre os vértices de um grafo serve como correspondente formal da conhecida técnica de Busca em Profundidade. A idéia de hierarquias transposta ao nível da implementação induz uma estrutura de dados simples e ao mesmo tempo poderosa para a representação de um grafo, que cremos ser útil na implementação e justificação de outros algoritmos baseados em Busca em Profundidade, na medida em que permite uma maior proximidade entre os níveis conceitual e de implementação.

- Uma estratégia de implementação do “*Método Guloso*”, para a obtenção de soluções aproximadas do Problema do Subgrafo Planar Ótimo, que faz uso da estrutura imposta sobre o grafo pela subrotina de teste de planaridade, diminuindo significativamente o tempo de processamento necessário para se obter soluções aproximadas de boa qualidade.
- O estabelecimento do limite de desempenho absoluto para o “*Método Guloso*”.

- Uma demonstração independente e simplificada da $\mathcal{NP}$ -Compleitude do Problema do Subgrafo Planar Ótimo.
- Resultados experimentais provenientes de nossa implementação e uma análise comparativa de desempenho com outras implementações desse e outros métodos publicados.
- Uma resenha das principais heurísticas para o Problema Subgrafo Planar Ótimo, incluindo uma breve descrição de cada uma delas e uma análise comparativa dos resultados experimentais publicados.

O trabalho está dividido em duas partes, cada uma com um diferente foco de atenção:

A **Primeira Parte** do trabalho é devotada à exposição detalhada de um algoritmo para teste de planaridade, que deve ser visto como uma variante do conhecido algoritmo de Hopcroft e Tarjan, adequada para uso como subrotina e irrestritiva quanto ao grafo que lhe serve de argumento. Esta parte está dividida da seguinte forma:

O **Capítulo 1** expõe alguns resultados fundamentais bem conhecidos na literatura e que formarão o suporte teórico do material nos capítulos seguintes. Os resultados clássicos são simplesmente enunciados e alguns outros, habitualmente apresentados sem a respectiva demonstração na literatura são provados. Ao final do capítulo, alguns comentários são feitos a respeito de resultados correlatos de interesse e as respectivas referências bibliográficas são fornecidas. Esses comentários não têm a intenção de constituir uma resenha exhaustiva sobre o assunto. Pelo contrário, procuramos dar ênfase a resultados e referências menos divulgadas que possam ser de utilidade a um eventual leitor interessado em assuntos relativos a caracterizações de planaridade.

O **Capítulo 2** faz uma apresentação detalhada de um algoritmo fundamental para teste de planaridade, cuja estrutura forma a base da grande maioria dos algoritmos para teste de planaridade publicados e usualmente apresentado em textos introdutórios de Teoria dos Grafos como um algoritmo para execução manual. A justificativa quanto ao funcionamento deste algoritmo é feita em um nível de detalhe superior ao usual na literatura, por meio de uma formalização de suas idéias fundamentais, que estabelecem linguagem adequada para o conteúdo do capítulo seguinte.

Numa seção final de comentários, análoga em espírito à do Capítulo 1, apresentamos e comentamos algumas referências de diferentes algoritmos de planaridade.

O **Capítulo 3** é o capítulo central dessa parte: ali desenvolvemos passo a passo os conceitos necessários para reformular as rotinas esquematizadas no capítulo anterior de modo a obter um algoritmo eficiente para teste de planaridade.

Depois de estabelecer algumas propriedades e construções auxiliares, mostramos como e por que os algoritmos apresentados no capítulo anterior podem ser implementados eficientemente por meio de uma Busca em Profundidade particularmente escolhida.

A intenção desse capítulo é a de prover uma referência detalhada e profusa em exemplos, que sirva como “vade mecum” de estudo e implementação do algoritmo de Hopcroft e Tarjan. Entretanto, ele também pode ser visto como uma proposta de formalização das idéias centrais por trás do conceito de Busca em Profundidade, e em particular do seu uso em algoritmos que lidam com a questão de planaridade.

Pela sua própria natureza, este capítulo é o mais longo e o de leitura mais árida em todo o trabalho. Procuramos fazer comentários elucidativos a cada novo conceito introduzido bem como prover exemplos suficientes que possam servir de apoio e referência ao leitor interessado na detalhada compreensão do algoritmo apresentado. Por isso, optamos por apresentar as demonstrações de resultados auxiliares em separado, na Seção 4.3, a fim de conduzir o leitor sem distrações até o objetivo do capítulo.

Finalmente, o **Capítulo 4** tem o papel de um apêndice dessa parte do trabalho, onde o leitor encontrará o rastreamento de um exemplo completo dos algoritmos estabelecidos, a discussão dos detalhes de implementação desses algoritmos e a demonstração dos resultados utilizados no desenvolvimento do capítulo anterior.

A **Segunda Parte** trata do Problema do Subgrafo Planar Ótimo propriamente dito:

No **Capítulo 5** apresentamos uma demonstração original da $\mathcal{NP}$ -Compleitude do problema. Apesar de o resultado ser já estabelecido na literatura, nossa demonstração é consideravelmente mais simples que a referência usual para o problema, que além disso é de difícil acesso.

O **Capítulo 6** traz uma breve descrição dos diversos enfoques propostos na literatura para o problema, quer sejam métodos para a obtenção de soluções exatas para o problema, quer sejam heurísticas para a obtenção de soluções aproximadas.

Descrevemos brevemente cada um desses enfoques e recolhemos as informações disponíveis a respeito de resultados experimentais, procurando dar uma visão unificada desses resultados e tentando apresentar análises comparativas de seu desempenho sempre que possível. O capítulo não tem a pretensão de ser uma resenha completa sobre os diversos métodos, antes serve como motivação para o material apresentado nos capítulos seguintes, ao por em evidência os resultados que tiveram influência mais decisiva no desenvolvimento deste trabalho.

O **Capítulo 7** apresenta os resultados de nossa experiência com aquela dentre as heurísticas analisadas no capítulo anterior que apresenta o melhor desempenho: o “*Método Guloso*”. Como resultado inédito, estabelecemos o seu limite de desempenho absoluto e a seguir descrevemos uma implementação da heurística baseada na formalização do algoritmo de planaridade apresentado na Primeira Parte, que reduz drasticamente seu tempo de execução com relação aos trabalhos até o momento publicados, tornando-a assim uma opção viável. Tal melhoria de desempenho é atingida em nossa implementação graças à estrutura de dados utilizada para a representação do grafo em construção, projetada de maneira a

tirar proveito da estrutura imposta sobre o grafo num contexto de testes incrementais de planaridade tal como ocorrem num “contexto guloso”.

Resultados computacionais são apresentados e comparados com os do capítulo anterior e estimativas do comportamento assintótico da implementação proposta são analisadas.

Preliminares

A finalidade deste capítulo é estabelecer a notação e as convenções utilizadas neste trabalho.

Assume-se relativa familiaridade do leitor com os conceitos básicos utilizados aqui, de forma que definições precisas serão explicitadas somente nos casos onde possa haver alguma ambigüidade.

O símbolo $\equiv$ introduz uma convenção, notação ou definição.

Conjuntos

Sempre que conveniente, identificamos um conjunto unitário $X = \{x\}$ com seu único elemento x .

Se X é um conjunto, $|X|$ indica sua cardinalidade.

Se X e Y são conjuntos disjuntos então sua união é indicada por $X + Y$.

$X \oplus Y$ indica a **diferença simétrica** entre X e Y , isto é, $X \oplus Y \equiv (X \cup Y) - (X \cap Y)$.

$\mathbb{Q}$ indica o conjunto dos números racionais (inclusive o zero) e $\mathbb{R}$ o dos números reais.

Relações e Funções

Por motivos que se tornarão claros ao longo deste trabalho, será mais conveniente estabelecer a notação necessária para discorrer a respeito de relações entre conjuntos e definir função como um caso particular de relação, como usual em textos de Álgebra e Teoria dos Conjuntos.

Pelos mesmos motivos, o uso de funções parciais será muito mais adequado para nossos propósitos, de forma que prevenimos o leitor de que, salvo indicação explícita em contrário, todas as funções mencionadas são parciais.

Nas definições seguintes assumimos que A , B e C são conjuntos quaisquer.

Uma **relação** ρ entre A e B é um subconjunto de $A \times B$. Nesse caso, se $X, Y \subseteq A$ então

$$\begin{aligned} X\rho Y &\equiv (X \times Y) \cap \rho \neq \emptyset; \\ X \not\rho Y &\equiv (X \times Y) \cap \rho = \emptyset; \\ \rho(X) &\equiv \{b \in B : x\rho b \text{ para algum } x \in X\}; \\ \rho^{-1} &\equiv \{(b, a) \in B \times A : a\rho b\}. \end{aligned}$$

Uma relação $\rho \subseteq A \times B$ será representada esquematicamente por um diagrama onde os elementos de A e B serão representados por pequenos círculos e $a\rho b$ será indicado por uma seta do círculo que representa a para o que representa b , exceto no caso em que $a = b$, quando representamos a por um círculo “preenchido”.

Se $\rho \subseteq A \times B$ e $X \subseteq A$, a **restrição de ρ a X** é a relação $\rho/X \equiv \rho \cap (X \times \rho(X))$.

Se $\rho \subseteq A \times B$ e $\sigma \subseteq B \times C$, a **composição** de ρ com σ é a relação

$$\rho\sigma \equiv \{(a, c) \in A \times C : a\rho b \text{ e } b\sigma c \text{ para algum } b \in B\}.$$

No caso em que $\rho \subseteq A \times A$ dizemos que ρ é uma **relação sobre A** e definimos

$$\rho^n \equiv \begin{cases} \{(a, a) : a \in A\}, & \text{se } n = 0; \\ \rho(\rho^{n-1}), & \text{caso contrário.} \end{cases}$$

$$\begin{aligned} \rho^+ &\equiv \bigcup_{n>0} \rho^n; \\ \rho^* &\equiv \rho^0 \cup \rho^+. \end{aligned}$$

Uma relação ρ sobre A é

- reflexiva** se $\rho^0 \subseteq \rho$;
- anti-reflexiva** se $\rho^0 \cap \rho = \emptyset$;
- simétrica** se $\rho = \rho^{-1}$;
- anti-simétrica** se $\rho \cap \rho^{-1} = \emptyset$;
- transitiva** se $\rho^2 \subseteq \rho$.

Uma **função** φ de A em B , denotada por $\varphi : A \rightarrow B$, é uma relação entre A e B onde $|\varphi(a)| \leq 1$ para todo $a \in A$. Dizemos que uma tal função é **total** se $|\varphi(a)| = 1$ para todo $a \in A$.

Uma **correspondência** entre A e B é uma função $\gamma : A \rightarrow B$ cuja inversa, γ^{-1} , também é função. Denotaremos uma tal correspondência por $A \xrightarrow{\gamma} B$ e cometeremos o abuso de não distinguir γ e γ^{-1} , dizendo indistintamente que x e $\gamma(x)$ são correspondentes entre si por γ (observe que no caso particular em que $A = B$, temos efetivamente $\gamma = \gamma^{-1}$ e a convenção não é um abuso de linguagem).

A **representação** de uma correspondência $A \xrightarrow{\gamma} B$ é o conjunto

$$\{\{a, \gamma(a)\} : a \in A\}.$$

Uma **equivalência** sobre A é uma relação reflexiva, simétrica e transitiva sobre A .

Se ρ é uma equivalência sobre A , $\rho(a)$ é dita a **classe** de a por ρ . O **quociente** de A por ρ é o conjunto

$$A/\rho \equiv \{\rho(a) : a \in A\}.$$

Uma **ordem** sobre A é uma relação anti-reflexiva, anti-simétrica e transitiva sobre A .

Se $<$ é uma ordem sobre A , então $<^{-1}$ também é uma ordem sobre A que denotaremos por $>$.

A notação $a \leq b$ indica que $a < b$ ou $a = b$ e a notação $a \leqslant b$ indica que $a < b$ e $c \leq a$ para todo $c < b$

Dois elementos $a, b \in A$ são **comparáveis** por uma ordem $<$ sobre A se $a \leq b$ ou $b \leq a$.

Um conjunto $X \subseteq A$ é **totalmente ordenado** se quaisquer elementos $x, y \in X$ são comparáveis.

Se $<$ é uma ordem sobre A , a **ordem lexicográfica** induzida por $<$ sobre A^n será também denotada por $<$. Nesse caso,

$$(a_1, a_2, \dots, a_n) < (b_1, b_2, \dots, b_n)$$

significa que para algum $1 \leq k \leq n$ temos $a_k < b_k$ e $a_i = b_i$ para todo $1 \leq i < k$.

Uma ordem $<$ será representada graficamente por seu **Diagrama de Hasse**. Assim, os elementos de A são representados por pequenos círculos dispostos de tal forma que se $x < y$ o círculo que representa x é desenhado numa posição acima do círculo que representa y e cada par $x < y$ é representado por um segmento de x a y .

Grafos

Um **grafo** é um par ordenado (V, A) , onde V é um conjunto finito e $A \subseteq \{\{u, v\} : u, v \in V; u \neq v\}$. Os elementos de V são chamados **vértices** e os de A **arestas** do grafo. Em geral, se G é um grafo, o conjunto dos seus vértices será indicado por V_G e o de suas arestas por A_G .

Como mencionamos anteriormente, alguns conceitos serão apresentados mais com a finalidade de estabelecer a terminologia e a notação adotadas. Ao leitor não muito familiarizado com conceitos da Teoria dos Grafos sugerimos consultar [BM76], [Bol79] ou [L79].

Em toda esta seção u e v são vértices e G, H são grafos.

Uma aresta $\{u, v\}$ será em geral denotada simplesmente por uv .

Se $X \subseteq V_G$, $\text{Adj}(X) \equiv \{v \in V_G : uv \in A_G \text{ para algum } u \in X\}$.

A notação $H \subseteq G$ indica que H é um subgrafo de G , ou seja, que $V_H \subseteq V_G$ e $A_H \subseteq A_G$. H é dito um subgrafo **gerador** de G se $V_H = V_G$.

Além disso, convencionamos que

$$G \cup H \equiv (V_G \cup V_H, A_G \cup A_H);$$

$$G + H \equiv (V_G \cup V_H, A_G + A_H);$$

$$G - H \equiv (V_G, A_G - A_H);$$

$$G \oplus H \equiv (V_G \cup V_H, A_G \oplus A_H).$$

A notação $\uplus$ quando relativa a grafos indica a união de vários grafos dois a dois disjuntos nas arestas.

Se $X \subseteq A_G$, o **subgrafo induzido** por X é o grafo

$$G[X] \equiv \left(\bigcup_{\alpha \in X} \alpha, X \right).$$

Denotamos por K_n um grafo completo de n vértices e $K_{m,n}$ um grafo bipartido completo com $m + n$ vértices onde um dos conjuntos da bipartição tem m vértices.

Um **passeio** em G é uma seqüência $P = (v_0, v_1, \dots, v_n)$ de vértices em G , onde $v_{i-1}v_i \in A_G$ para todo $1 \leq i \leq n$. Nesse caso:

Os vértices v_0 e v_n são chamados **pontas** do passeio, enquanto que os vértices $v_1, \dots, v_{n-1}$ são chamados **internos** ao passeio, e o **tamanho** do passeio é denotado por $|P|$ e vale n .

P^{-1} indica o passeio $(v_n, v_{n-1}, \dots, v_0)$.

Se $Q = (u_0, u_1, \dots, u_n)$ com $v_n = u_0$, PQ indica o passeio

$$PQ \equiv (v_0, v_1, \dots, v_n, u_0, u_1, \dots, u_n).$$

Dois passeios são **internamente disjuntos** se os conjuntos dos vértices internos de cada um deles são disjuntos entre si.

Ocasionalmente identificamos o passeio P com o grafo

$$G[P] \equiv G[\{v_{i-1}v_i, 1 \leq i \leq n\}].$$

Um **caminho** é um passeio onde todos os vértices são distintos.

Um **circuito** é um passeio onde todos os vértices internos são distintos entre si e as pontas coincidem.

Um **triângulo** é um circuito de tamanho 3.

Uma **base de circuitos** de G é um conjunto minimal $\mathcal{B}$ de circuitos de G tal que para todo circuito C em G existe $\mathcal{X}_C \subseteq \mathcal{B}$ de forma que

$$C = \bigoplus_{\overline{C} \in \mathcal{X}_C} \overline{C}.$$

Uma **componente** de G é um subgrafo maximal conexo de G . $\|G\|$ denota o **número de componentes** de G .

Uma **floresta** de G é um subgrafo de G sem circuitos.

Dados uma floresta F e dois vértices $u, v \in V_F$ tais que $uv \notin A_F$, chamamos de **circuito fundamental** de uv com relação a F ao único circuito de $F + uv$. Este circuito será denotado por C_{uv}^F .

Um conjunto $X \subseteq V_G$ **separa** u de v em G se todo passeio envolvendo u e v envolve necessariamente algum vértice de X .

Um **separador** em G é um subconjunto de V_G que separa algum par de vértices de G .

Uma **articulação** é um separador unitário.

Um **bloco** de G é um subgrafo maximal conexo de G sem articulações.

Um bloco é dito **trivial** se tem menos de 2 arestas. Por extensão, também chamaremos de **bloco** um grafo sem articulações.

Uma **ponderação** ω sobre G é uma função total $\omega : A_G \rightarrow Q$.

Se ω é uma ponderação sobre G e $H \subseteq G$,

$$\omega(H) \equiv \sum_{\alpha \in A_H} \omega(\alpha).$$

Se G é um grafo e ω uma ponderação de G , dizemos que G é um grafo ω -**ponderado**.

Parte I

Planaridade

Capítulo 1

Definições e Resultados Preliminares

Para podermos tratar do assunto de maneira mais precisa, é inevitável que introduzamos algumas definições e estabeleçamos alguns resultados. É o que faremos neste capítulo.

Definição 1.1 *Uma imersão no plano de um grafo G é uma quádrupla $\overline{G} = (\overline{V}, \overline{A}, \nu, \mu)$ onde*

$\overline{V}$ é um conjunto de $|V_G|$ pontos em $\mathbb{R}^2$,

$\overline{A}$ é um conjunto de $|A_G|$ curvas contínuas em $\mathbb{R}^2$,

ν é uma correspondência total $V_G \leftrightarrow \overline{V}$ e

μ é uma correspondência total $A_G \leftrightarrow \overline{A}$

de tal forma que $\mu(uv)$ é uma curva contínua que liga $\nu(u)$ a $\nu(v)$, para toda aresta uv em A_G .

O leitor deve estar alerta para o fato de que nossa definição de *imersão no plano* difere do conceito habitualmente definido em outros textos pelo mesmo nome: uma imersão no plano (no sentido usado aqui) pode em geral apresentar “cruzamentos”, isto é, as curvas contínuas que representam diferentes arestas podem se interceptar em pontos que não sejam representações de vértices. Quando não é esse o caso, dizemos que a imersão no plano é *sem cruzamentos*, como definido mais adiante.

Em geral, dada uma imersão no plano $\overline{G}$ de um grafo G , e conjuntos $X \subseteq V_G$ e $Y \subseteq A_G$, denotaremos $\nu(X)$ por $\overline{X}$ e $\mu(Y)$ por $\overline{Y}$.

Do mesmo modo, se $H \subseteq G$ então $\overline{H}$ denota a imersão $(\overline{V_H}, \overline{A_H}, \nu/V_H, \mu/A_H)$.

Definição 1.2 *Uma imersão no plano $\overline{G}$ de um grafo G é **sem cruzamentos** se para quaisquer duas arestas α, β em G tem-se que $\overline{\alpha} \cap \overline{\beta} = \overline{\alpha \cap \beta}$.*

Definição 1.3 *Um grafo é **planar** se admite uma imersão no plano sem cruzamentos.*

Definição 1.4 *Dados dois grafos G e H , dizemos que G é **homeomorfo** a H se existe uma correspondência $V_G \leftrightarrow V_H$ tal que G admite uma decomposição $\mathcal{P}_G$ onde:*

- $\mathcal{P}_G$ é um conjunto de caminhos dois a dois internamente disjuntos.
- Existe uma correspondência total $\mathcal{P}_G \leftrightarrow A_H$ tal que os correspondentes das pontas de um caminho P são as pontas da aresta correspondente a P .

Definição 1.5 *Uma **2-base** de um grafo G é uma base de circuitos $\mathcal{B}$ de G onde cada aresta ocorre em no máximo 2 circuitos de $\mathcal{B}$.*

Apresentamos a seguir alguns resultados clássicos que serão necessários ao desenvolvimento dos capítulos seguintes. Alguns comentários a respeito de cada um deles bem como referências a trabalhos onde suas demonstrações podem ser encontradas serão dados no final do capítulo.

Teorema 1.6 *Uma curva fechada simples C divide o plano em duas regiões, chamadas **lados** de C , tais que qualquer curva contínua que tenha intersecção não vazia com ambas, também tem intersecção não vazia com C .*

Teorema 1.7 *Um grafo é planar se e somente se não tem nenhum subgrafo homeomorfo a K_5 ou $K_{3,3}$.*

Teorema 1.8 *Um grafo é planar se e somente se admite uma 2-base.*

Corolário 1.9 *Um grafo é planar se e somente se cada um de seus blocos é planar.*

Prova A necessidade da condição é evidente.

Para a suficiência, suponha que $B_1, \dots, B_k$ são os blocos de um grafo G , cada um deles planar, e sejam $B_1, \dots, B_k$ suas respectivas 2-bases. Como cada B_i é um bloco, essas 2-bases são duas a duas disjuntas nas arestas. Logo $B_1 + B_2 + \dots + B_k$ é uma 2-base de G , que portanto é planar. $\square$

Corolário 1.10 *Todo grafo sem circuitos é planar.*

Teorema 1.11 *Se $\mathcal{B}$ é uma base de circuitos de G então*

$$|\mathcal{B}| = |A_G| - |V_G| + \|G\|.$$

Fato 1.12 *Se G é um bloco não trivial e $\mathcal{B}$ é uma 2-base de G então*

$$C_{\mathcal{B}} \equiv \bigoplus_{C \in \mathcal{B}} C \text{ é um circuito de } G.$$

Nesse caso convençionamos que

$$\overline{\mathcal{B}} \equiv \mathcal{B} \cup C_{\mathcal{B}}$$

e notamos que cada aresta de G aparece em exatamente 2 circuitos de $\overline{\mathcal{B}}$ a menos que $G = C_{\mathcal{B}}$.

Corolário 1.13 *Se G é um bloco planar não trivial e o menor circuito de G tem comprimento k então*

$$|A_G| \leq \frac{k(|V_G| - 2)}{k - 2}.$$

Prova Se G é um circuito, então temos $k = |V_G| = |A_G|$ e o resultado é imediato. Caso contrário, seja $\mathcal{B}$ uma 2-base de G .

Como G é um bloco temos pelo Fato 1.12 que

$$\sum_{C \in \overline{\mathcal{B}}} |C| = 2|A_G|.$$

Por outro lado,

$$\sum_{C \in \overline{\mathcal{B}}} |C| \leq k|\overline{\mathcal{B}}| = k(|A_G| - |V_G| + 2),$$

e portanto,

$$2|A_G| = k(|A_G| - |V_G| + 2),$$

donde segue que

$$|A_G| \leq \frac{k(|V_G| - 2)}{k - 2}.$$

□

Corolário 1.14 *Se G é um bloco planar não trivial e todo circuito de G tem comprimento k então*

$$|A_G| = \frac{k(|V_G| - 2)}{k - 2}.$$

Prova Análoga à do Corolário 1.13.

□

Corolário 1.15 *Se G é um grafo planar maximal, e B é uma 2-base de G , então todo circuito em $\overline{B}$ é um triângulo.*

Prova Sejam G e B como no enunciado e considere $C \in \overline{B}$. Supondo existirem $u, v \in C$ tais que $uv \notin A_G$ temos que

$$C = (u)P(v)Q(u),$$

sendo P, Q caminhos em G . Neste caso,

$$B - C + \{(u)P(v)(v, u), (v)Q(u)(u, v)\}$$

é uma 2-base de $G + uv$. Logo, pelo Teorema 1.8 temos que $G + uv$ é planar, contrariando a hipótese de G ser maximal planar. $\square$

Corolário 1.16 *Se G é um grafo planar maximal com $|V_G| \geq 3$ temos que*

$$|A_G| = 3|V_G| - 6.$$

Além disso, se $|V_G| > 3$ toda aresta (resp. todo vértice) de G pertence a pelo menos dois triângulos.

Prova A primeira parte segue diretamente dos Corolários 1.14 e 1.15.

Quanto à segunda parte, basta notar que se G é maximal planar e B uma 2-base de G , pelo Corolário 1.15 todo circuito de $\overline{B}$ é um triângulo e pelo Fato 1.12 toda aresta de G aparece em exatamente 2 deles.

Como G é planar maximal, cada vértice é extremo de alguma aresta, e portanto comparece nos circuitos em que essa aresta comparece. $\square$

Corolário 1.17 *Se G é planar com $|V_G| \geq 3$, então*

$$|A_G| \leq 3|V_G| - 6.$$

Prova Segue imediatamente do Corolário 1.16. $\square$

Finalmente, apresentamos na Figura 1.1 o grafo que será nosso companheiro de todos os momentos ao longo deste trabalho, servindo de exemplo aos conceitos introduzidos.

Comentários

O Teorema 1.11 é um resultado fundamental da Teoria dos Grafos cuja demonstração pode ser encontrada em [BM76] ou [Ber73]. Sua versão especializada para o caso de grafos planares, onde a base de circuitos em questão é chamada de “conjunto das faces” do grafo é conhecida como Teorema de Euler, e é de menção obrigatória dos capítulos dedicados ao tópico de planaridade de textos de Teoria dos Grafos.

O Teorema 1.6 é o célebre Teorema da Curva de Jordan, um resultado de grande apelo intuitivo e cuja difícil demonstração pode ser encontrada em textos de Topologia. Deve ser notado que embora tenhamos optado por seguir o roteiro tradicional para a introdução de uma noção puramente combinatória de planaridade, definindo “imersões no plano”, “imersões sem cruzamento” e fazendo uso de linguagem própria da Topologia [Mun84], o mesmo conceito poderia ser sucintamente definido por algo como

Um grafo é planar se é o 1-esqueleto de algum CW -complexo em $\mathbb{R}^2$.

O Teorema 1.7 é o não menos célebre Teorema de Kuratowski [Kur30], a primeira e mais elegante das caracterizações combinatórias de planaridade, um ponto de referência de outras caracterizações posteriores. Os grafos K_5 e $K_{3,3}$ ou grafos homeomorfos a estes são muitas

Figura 1.1: Nosso Exemplo

vezes referidos como “*Grafos de Kuratowski*”. Sua demonstração original pode ser encontrada em textos de Teoria dos Grafos como [BM76] ou [Bol79]. Em [Tho80] o leitor encontrará uma demonstração simplificada desse resultado, também recolhida em [CN88].

O Teorema 1.8 é a caracterização de planaridade de MacLane [Mac37]. Uma demonstração relativamente simples desse resultado pode ser encontrada também em [CN88].

Depois da publicação do Teorema de Kuratowski a literatura tem registrado um grande número de resultados equivalentes. Além dos citados, os mais conhecidos são os de Whitney [Whi33] (“um grafo é planar se e somente se seu dual combinatório é planar”), e o de Tutte [Tut58], que será introduzido no próximo capítulo deste trabalho.

Dentre os menos citados, mas ainda assim com algum destaque, encontramos o de Liu Yanpei [Liu88] (que relaciona a planaridade do grafo a solução de um sistema de equações sobre $GF(2)$ construída a partir do grafo) que fornece também um algoritmo de complexidade linear para teste de planaridade.

O leitor pode ainda encontrar outras caracterizações em [Kna75], [Kel81] (cuja demonstração não se baseia em nenhuma das outras caracterizações conhecidas), [Bre86] ou [Fou74].

Alguma atenção foi dada à obtenção de provas alternativas das caracterizações mais importantes: assim o leitor pode encontrar provas das caracterizações de Whitney e MacLane como conseqüências do Teorema de Kuratowski em [Tut63] e uma prova mais simples da caracterização de MacLane em [O’N73].

O Teorema de Kuratowski pela sua elegância atraiu diversos pesquisadores: a primeira simplificação de sua demonstração foi dada por [DS54]; uma baseada na caracterização de Tutte pode ser vista em [Bur78]. Uma demonstração construtiva que sugere um algoritmo de complexidade quadrática pode ser vista em [Klo89] (o algoritmo em questão localiza um subgrafo de Kuratowski caso o grafo testado não seja planar).

Particularmente notória é a demonstração apresentada em [Tho80], recolhida com ligeiras alterações em textos como [CN88] e [GT87], pela sua organização que permite provar simultaneamente os Teoremas de Kuratowski, Fáry [Far48] e a extensão do Teorema de Fáry feita por Tutte [Tut60, Tut63]. Este artigo apresenta ainda os critérios de Whitney, MacLane, Tutte e Fournier e um outro importante resultado devido a Tutte [Tut75], e também sugere um algoritmo de complexidade polinomial para teste de planaridade, baseado no Teorema de Kuratowski.

Capítulo 2

Um Algoritmo para Teste de Planaridade

Neste capítulo desenvolveremos passo a passo as idéias fundamentais de um algoritmo de teste de planaridade. As idéias apresentadas aqui servirão de base para uma discussão mais detalhada nos capítulos seguinte, onde receberão sua forma final.

Considere um grafo G sobre o qual desejamos decidir a planaridade. De acordo com os Corolários 1.9, 1.10 e 1.17, faz sentido formular o seguinte algoritmo de teste de planaridade:

Algoritmo 2.1

Planar(G)

Entrada: Um grafo G .

Saída: sim se G é planar; caso contrário interrompe a execução.

Se $|A_G| > 3|V_G| - 6$

Interrompa;

Para cada bloco não trivial B de G

Planar'(B);

Devolva sim.

Algoritmo 2.2

Planar'(B)

Entrada: Um bloco não trivial B .

Saída: Nenhuma, se B for planar; caso contrário interrompe a execução.

Nosso primeiro objetivo de curto prazo será estabelecer o Algoritmo 2.2. Para tanto, pas-samos a apresentar alguns conceitos importantes, que serão ilustrados por meio de exemplos baseados no grafo da Figura 1.1.

A fim de simplificar a discussão subsequente, suponha que G seja um bloco não trivial e que $C \subseteq G$ seja um circuito fixo de G . O leitor deve notar que tais suposições não comprometerão o desenvolvimento a seguir, uma vez que cada resultado estende-se naturalmente para o caso geral graças ao Corolário 1.9.

Considere a relação $\sim$ sobre $A_G - A_C$ dada por $\alpha \sim \beta$ se existe um passeio P envolvendo α e β , sendo que as pontas de P estão em C mas não os seus vértices internos.

Como consequência imediata da nossa definição de imersão no plano, temos que $\overline{C}$ é uma curva fechada simples em qualquer imersão no plano sem cruzamentos de G , e daí por uma consequência também imediata do Teorema de Jordan, temos que se $\alpha \sim \beta$ então $\overline{\alpha}$ e $\overline{\beta}$ estarão contidas num mesmo lado de $\overline{C}$ em qualquer imersão no plano sem cruzamentos de G .

Por outro lado, a relação $\sim$ é claramente uma equivalência sobre $A_G - A_C$, o que nos sugere a seguinte definição.

Definição 2.1 *Um C -segmento S de G é um grafo induzido por uma classe de $(A_G - A_C)/\sim$. Os vértices em $V_S \cap V_C$ são denominados vértices de contato entre S e C .*

Denotaremos por G/C o conjunto de todos os C -segmentos de G , ou seja,

$$G/C \equiv \{G[E] : E \in (A_G - A_C)/\sim\}.$$

Fato 2.2 *Todo C -segmento S de G determina uma decomposição única de C em um conjunto de passeios internamente disjuntos onde só os extremos de cada um desses passeios são vértices de contato entre C e S . A decomposição assim obtida será denotada por C/S .*

Exemplo 2.1 *A Figura 2.2 mostra o grafo da Figura 1.1 com um circuito C em linhas destacadas.*

Para o grafo G e o circuito C indicados na Figura 2.2 as classes de $\sim$ são:

$$E_1 = \{5\ 15, 9\ 15, 11\ 14, 12\ 13, 13\ 14, 13\ 15, 14\ 15\},$$

$$E_2 = \{2\ 16, 3\ 16, 10\ 16\},$$

$$E_3 = \{6\ 8\} \text{ e}$$

$$E_4 = \{4\ 7\}.$$

Portanto,

$$G/C = \{S_1 = G[E_1], S_2 = G[E_2], S_3 = G[E_3], S_4 = G[E_4]\}.$$

A Figura 2.1 mostra os segmentos de G/C , com os vértices de contato de cada um deles.

A decomposição de C determinada pelos C -segmentos S_1 , S_2 , S_3 e S_4 é:

$$C/S_1 = \{(5, 6, 7, 8, 9), (9, 10, 11), (11, 12), (12, 1, 2, 3, 4, 5)\},$$

$$C/S_2 = \{(2, 3), (3, 4, 5, 6, 7, 8, 9, 10), (10, 11, 12, 1, 2)\},$$

$$C/S_3 = \{(6, 7, 8), (8, 9, 10, 11, 12, 1, 2, 3, 4, 5, 6)\} \text{ e}$$

$$C/S_4 = \{(4, 5, 6, 7), (7, 8, 9, 10, 11, 12, 1, 2, 3, 4)\}.$$

Consideremos agora a relação $\approx$ sobre G/C dada por

$$S \approx S' \equiv S = S' \text{ ou } V_C \cap V_S \subseteq P \text{ para algum } P \in C/S'.$$

A relação $\approx$ revela quais C -segmentos de G podem ser imersos num mesmo lado de $\overline{C}$. Mais precisamente temos o seguinte fato.

Figura 2.2: O grafo da Figura 1.1 e um circuito C

Fato 2.3 Se $S \subseteq G/C$ é tal que $S \approx S'$ quaisquer que sejam $S, S' \in S$, então $C + \left(\bigcup_{S \in S} S \right)$ admite uma imersão no plano sem cruzamentos onde todos os C -segmentos de S estão num mesmo lado de $\overline{C}$.

As observações acima motivam a seguinte definição.

Definição 2.4 Uma C -imersão de G é um subconjunto de G/C . Uma C -imersão $\mathcal{I}$ é dita **planar** se

- $S \approx S'$ para todo $S, S' \in \mathcal{I}$;
- $S \approx S'$ para todo $S, S' \in G/C - \mathcal{I}$.

Exemplo 2.2 A Figura 2.3 mostra um diagrama da relação $\approx$ para o nosso exemplo, onde podemos ver que $\{S_1, S_3\}$ é uma C -imersão planar.

A seguinte caracterização segue das definições e dos fatos anteriores.

Figura 2.3: Diagrama da relação $\approx$ sobre G/C

Fato 2.5 G é planar se, e somente se, para algum (e portanto todo) circuito $C \subseteq G$ temos que G admite uma C -imersão planar e $C + S$ é planar para todo $S \in G/C$.

Esta caracterização sugere imediatamente uma primeira formulação do Algoritmo 2.2 dada por:

Algoritmo 2.3

$\text{Planar}'(B)$

Entrada: Um bloco não trivial B .

Saída: Nenhuma se B for planar; caso contrário interrompe o processamento.

$C \leftarrow$ um circuito de B ;

$\mathcal{R} \leftarrow \emptyset$;

Para cada $S \in B/C$

$\text{Planar}'(C + S)$;

$\text{Imersão}(C, \mathcal{R}, S)$.

Algoritmo 2.4

$\text{Imersão}(C, \mathcal{R}, S)$

Entrada: Um circuito C , um par de conjuntos de C -segmentos $\mathcal{R} = \{\mathcal{I}, \mathcal{F}\}$ e um C -segmento S , $S \notin \mathcal{I} + \mathcal{F}$, tais que $\mathcal{I}$ é uma C -imersão planar de $G[C + \mathcal{I} + \mathcal{F}]$.

Saída: Um rearranjo dos C -segmentos em $\mathcal{R}$ obtendo um novo $\mathcal{R} = \{\mathcal{I}', \mathcal{F}'\}$ com $\mathcal{I}' + \mathcal{F}' = \mathcal{I} + \mathcal{F} + S$ e tal que $\mathcal{I}'$ é uma C -imersão planar de $G[C + \mathcal{I}' + \mathcal{F}']$.

Na verdade deveríamos ter certo cuidado em dizer que Planar' é um algoritmo, pois ele cairá num “loop” de recursão toda vez que a escolha do circuito C deixar um único segmento S em

B/C . Nesse caso, como $C + S = B$, a chamada recursiva terá o mesmo argumento que a anterior. Pior ainda, esse caso fatalmente acontecerá à medida em que o algoritmo se desenvolve.

Apesar disso, o leitor pode perceber que a execução manual do “algoritmo” acima é possível se os circuitos forem escolhidos convenientemente a cada passo e a cadeia recursiva for interrompida quando B for “evidentemente planar”. Preferimos apresentá-lo assim porque sua idéia geral é bastante simples e intuitiva.

Passamos então a um novo “objetivo de curto prazo”, o Algoritmo 2.4, o que nos levará a um estudo mais detalhado da relação $\approx$.

Para começar, convençionemos denotar $\napprox$ por $\times$, atendendo ao apelo intuitivo de que $S \times S'$ significa que os segmentos S e S' “cruzam-se inevitavelmente” em qualquer imersão no plano de $C + S + S'$ em que ambos sejam imersos num mesmo lado de $\overline{C}$.

Consideremos a relação $\times^2$ sobre G/C . Um momento de reflexão a seu respeito nos diz que $S \times^2 S'$ quer dizer que para algum $T \in G/C$ temos $S \times T \times S'$, e portanto S e S' devem ser imersos num mesmo lado de $\overline{C}$ em qualquer imersão no plano sem cruzamentos de G (ambos do lado “oposto” de T).

Como $\times^2$ é simétrica, temos que $(\times^2)^*$ é uma equivalência sobre G/C . Uma consideração mais detida a respeito de $(\times^2)^*$ nos leva a concluir que a classe de S por $(\times^2)^*$ é o conjunto de todos os C -segmentos de G que necessariamente *devem* ser imersos num mesmo lado de $\overline{C}$ que S em qualquer imersão no plano sem cruzamentos de G . Do mesmo modo, a imagem de S por $\times(\times^2)^*$ são todos os C -segmentos de G que *não podem* ser imersos nesse mesmo lado numa imersão no plano sem cruzamentos de G .

Essas idéias sugerem considerar a relação $\otimes$ sobre $(G/C)/(\times^2)^*$ dada por

$$S \otimes T \equiv S \times T.$$

Exemplo 2.3 Na Figura 2.4 mostramos os diagramas de $\times$, $\times^2$, $\times(\times^2)^*$ e $\otimes$. Note que

$$(G/C)/(\times^2)^* = \{\{S_1, S_3\}, \{S_2, S_4\}\}.$$

O seguintes resultados dão uma boa idéia a respeito da estrutura do quociente $(G/C)/(\times^2)^*$.

Figura 2.4: As relações $\times$, $\times^2$, $\times(\times^2)^*$ e $\otimes$

Lema 2.6 *Sejam S e T elementos de $(G/C)/(\times^2)^*$. Então $S \otimes T$ se e somente se $S \times (\times^2)^* T$ para todo $S \in \mathcal{S}$ e $T \in \mathcal{T}$.*

Prova Sejam $S, T \in (G/C)/(\times^2)^*$, $S \in \mathcal{S}$ e $T \in \mathcal{T}$. Então

$$\begin{aligned}
 S \otimes T &\Leftrightarrow ((\times^2)^*(S)) \times ((\times^2)^*(T)) \\
 &\Leftrightarrow S(\times^2)^* \times (\times^2)^* T \\
 &\Leftrightarrow S \times (\times^2)^* (\times^2)^* T \\
 &\Leftrightarrow S \times (\times^2)^* T.
 \end{aligned}$$

□

Corolário 2.7 *A relação $\otimes$ é simétrica.*

Lema 2.8 A relação $\otimes$ é uma correspondência sobre $(G/C)/(\times^2)^*$.

Prova Basta mostrar que $\otimes$ é função, o que é simples se tomamos $S \in (G/C)/(\times^2)^*$ e consideramos que

$$\otimes(S) = (\times(\times^2)^*)(S) = \times(((\times^2)^*)(S)) = \times(S).$$

Para concluir o argumento, só é necessário mostrar que

$$\times(S) \in (G/C)/(\times^2)^*,$$

ou seja, que dados $S, T \in \times(S)$ temos $S(\times^2)^*T$.

Isto também é simples pois

$$\begin{aligned} S' \in \times(S) &\Rightarrow S \times S' \text{ para algum } S' \in S, \\ T' \in \times(S) &\Rightarrow T \times T' \text{ para algum } T' \in S, \end{aligned}$$

e portanto $S'(\times^2)^*T'$.

Logo,

$$S \times S'(\times^2)^*T' \times T \Leftrightarrow S \times (\times^2)^* \times T,$$

e pela simetria de $\times$,

$$S(\times^2)^* \times \times T \Leftrightarrow S(\times^2)^*(\times^2)T \Leftrightarrow S(\times^2)^+T \Rightarrow S(\times^2)^*T,$$

o que conclui o argumento. □

Lema 2.9 Um grafo G admite uma C -imersão planar se e somente se $\otimes$ é anti-reflexiva.

Prova Pelo Lema 2.6 temos que $\otimes$ é anti-reflexiva se e somente se $\times(\times^2)^*$ é anti-reflexiva.

A partir daí temos:

($\Rightarrow$) Seja $\mathcal{I} \in G/C$ uma C -imersão planar de G e considere $S, T \in (G/C)$ tais que $S \times (\times^2)^* T$. Sem perda de generalidade, suponha que $S \in \mathcal{I}$. Então $S \times (\times^2)^k T$ para algum $k \geq 0$, ou seja,

$$S \times T_1 \times^2 T_2 \times^2 T_3 \times^2 \dots \times^2 T_{k-1} \times^2 T,$$

ou equivalentemente,

$$S \times T_1 \times S_1 \times T_2 \times S_2 \times T_3 \times S_3 \times \dots \times S_{k-2} \times T_{k-1} \times S_{k-1} \times T.$$

Daí, como $\mathcal{I}$ é uma C -imersão planar,

$$S \in \mathcal{I} \Rightarrow T_1 \notin \mathcal{I} \Rightarrow S_1 \in \mathcal{I} \Rightarrow T_2 \notin \mathcal{I} \Rightarrow \dots \Rightarrow T_{k-1} \notin \mathcal{I} \Rightarrow S_{k-1} \in \mathcal{I} \Rightarrow T \notin \mathcal{I}.$$

Logo, $S = T$ é impossível, e portanto $S \times (\times^2)^* S$ também o é, donde segue que $\times(\times^2)^*$ é anti-reflexiva.

($\Leftarrow$) Seja $\mathcal{R}$ a representação de $\otimes$. Como $\otimes$ é anti-reflexiva, temos que cada elemento de $\mathcal{R}$ é da forma $\{S, S'\}$ com $S \neq S'$ (embora eventualmente com $S = \emptyset$ ou $S' = \emptyset$).

Assim, considere

$$\mathcal{I} = \bigcup_{\{S, S'\} \in \mathcal{R}} S.$$

Pela construção de $\mathcal{I}$ temos que, se $S, T \in (G/C)$ são tais que $S \times T$, então $S \in \mathcal{I} \Leftrightarrow T \notin \mathcal{I}$, e portanto $\mathcal{I}$ é uma C -imersão planar de G .

□

Os lemas anteriores sugerem a seguinte reformulação da nossa definição de C -imersão.

Definição 2.10 A C -imersão de um grafo G é a correspondência $\otimes$ sobre $(G/C)/(\times^2)^*$.

Feito isto, podemos reformular a nossa caracterização de planaridade (Fato 2.5) do seguinte modo.

Fato 2.11 Seja G um grafo e C um circuito de G . Então G é planar se e somente se sua C -imersão é anti-reflexiva e $C + S$ é planar para todo $S \in G/C$.

E com isso temos uma primeira formulação do Algoritmo 2.4:

Algoritmo 2.5

Imersão($C, \mathcal{R}, S$)

Entrada: Um circuito C , a representação $\mathcal{R}$ da C -imersão de $G[\mathcal{R}]$ (supostamente anti-reflexiva) e um C -segmento $S \notin \mathcal{R}$.

Saída: A representação da C -imersão de $G[\mathcal{R} + S]$; no caso de esta ser reflexiva, interrompe a execução.

```

 $Y \leftarrow S;$ 
 $Z \leftarrow \emptyset;$ 
Para cada  $\{S, S'\} \in \mathcal{R}$ 
  Se  $S \times S$ 
    Se  $S \times S'$ 
      Interrompa;
       $\mathcal{R} \leftarrow \mathcal{R} - \{S, S'\};$ 
       $Y \leftarrow Y + S';$ 
       $Z \leftarrow Z + S;$ 
    Senão
      Se  $S \times S'$ 
         $\mathcal{R} \leftarrow \mathcal{R} - \{S, S'\};$ 
         $Y \leftarrow Y + S;$ 
         $Z \leftarrow Z + S';$ 
Devolva  $\mathcal{R} + \{Y, Z\}.$ 

```

Comentários

O resultado que apresentamos como Fato 2.5 é uma formulação do conhecido critério de planaridade de Tutte [Tut58].

Os Algoritmos 2.1, 2.2 e 2.4 são uma aplicação direta deste critério, e foram publicados (aparentemente de forma independente) pela primeira vez em [AP61] e [DMP65]; [AP61] apresenta o algoritmo com um pequeno erro, que o leva a um “*loop*” de recursão, corrigido posteriormente em [Gol63].

É bastante interessante o fato de que a grande maioria dos algoritmos publicados para testar planaridade de um grafo são variações desse algoritmo. Exceções notáveis são [Klo89], um algoritmo “praticável” baseado diretamente no Teorema de Kuratowski e [FW66], um algoritmo que usa a matriz de incidência do grafo. Curioso o fato de que na sua publicação original, ambos são apresentados como algoritmos para execução manual ([AP61] chega mesmo a afirmar que uma eventual implementação do algoritmo seria muito mais interessante se feita num computador analógico).

A literatura mostra-se tão profusa quanto a diferentes algoritmos para determinação de planaridade como com relação a diferentes caracterizações de planaridade.

O mais célebre dentre estes é sem dúvida o algoritmo de Hopcroft e Tarjan descrito em [HT74], doravante referido abreviadamente por **HT**, o primeiro algoritmo de complexidade linear publicado.

Alguns trabalhos inspirados em **HT** merecem a atenção do leitor interessado nesse algoritmo: uma relação de pequenos erros no texto de [HT74] pode ser encontrada em [Deo76]; exposições das idéias básicas nas quais **HT** se baseia podem ser encontradas em [Eve79], [Gou88] ou [Wil85], embora nenhum destes seja completo ao ponto de dispensar a leitura de [HT74] no caso de o leitor desejar levar a cabo uma implementação do algoritmo; formulações e implementações alternativas de **HT** podem ser vistas em [dFR82], [Wil80], ou [Mut92]; o material exposto em [Wil80] reformulado de maneira a constituir um texto em nível de graduação explicando algumas das idéias centrais de **HT**, pode ser encontrado em [Wil85]; particularmente interessante também é a série de artigos [Vo83], [Dic84] e [VDW85], onde o tratamento formal de [Wil80] é levado adiante estabelecendo conhecidos resultados numa linguagem unificada e apropriada à exposição das idéias centrais de **HT**.

Um outro algoritmo bastante conhecido para teste de planaridade, ao qual nos referiremos abreviadamente por **LEC**, foi proposto em [LEC67]. Posteriormente uma estrutura de dados conhecida como “*Priority-Queue Tree*”, proposta em [BL76] mostrou ser possível sua implementação também em tempo linear de maneira que a partir de então, **HT** e **LEC** passaram a ser conhecidos como os dois algoritmos básicos para teste de planaridade de complexidade linear; uma exposição acessível de **LEC** (embora não completa ao ponto de permitir uma implementação eficiente) pode ser encontrada em [Eve79]; uma exposição mais completa, incluindo a discussão sobre “*P-Q Trees*” pode ser vista em [CN88].

HT e **LEC** vem sendo tradicionalmente considerados como dois enfoques diferentes para o tratamento da questão: **HT** é muitas vezes designado como o algoritmo de teste de planaridade por “sucessivos acréscimos de arestas” enquanto que **LEC** é conhecido como a estratégia de “sucessivos acréscimos de vértices”. Entretanto, um trabalho bastante interessante a esse respeito é o exposto em [CW90], que levando ainda mais adiante o tratamento formal dado em [Wil80], [Vo83], [Dic84] e [VDW85], postula a idéia de que ambos são essencialmente o mesmo algoritmo, se considerados do ponto de vista da seqüência de operações realizadas sobre o grafo.

Na seqüência apresentaremos com detalhe um algoritmo de teste de planaridade que corresponde essencialmente às idéias de **HT**, expostas e implementadas de maneira conveniente para uso como subrotina nos algoritmos expostos na segunda parte deste trabalho. Dedicamos o restante desta seção a estabelecer as relações entre **HT** e o material contido nesta parte do trabalho.

O Algoritmo de Hopcroft e Tarjan pode ser visto como uma engenhosa implementação dos algoritmos propostos em [AP61, Gol63] (correspondentes aos nossos Algoritmos 2.1, 2.2 e 2.4), que tira proveito da conhecida técnica de *Busca em Profundidade* sobre um grafo de forma a atingir desempenho linear. Entretanto, da maneira como é apresentado ao leitor em [HT74], as

propriedades fundamentais nas quais o algoritmo se baseia ficam um tanto obscurecidas em favor da ênfase dada à escolha das estruturas de dados utilizadas e das técnicas algorítmicas envolvidas, o que faz com que algumas vezes o algoritmo seja referido pela literatura como complexo, obscuro ou de difícil implementação [Eve79, FGG85, VDW85, Gou88, Wil85, CNAO85, DET93]. Muito embora o algoritmo seja de fato sofisticado, no sentido de que faz uso de agudas percepções a respeito da estrutura imposta sobre a Busca em Profundidade e sua utilização na determinação de planaridade, veremos que boa parte de sua “dificuldade” desvanece no momento em que as idéias fundamentais em que se baseia são sistematicamente apresentadas dentro da linguagem usual da Teoria dos Grafos.

Como um exemplo palpável do que acabamos de afirmar, chamamos a atenção do leitor para o fato de que o desenvolvimento do estudo da relação $(\times^2)^*$ acima tenta explicitar uma das idéias chave de HT, implícita na manipulação das estruturas de dados apresentadas em [HT74]. O que apresentamos como “classes de $(G/C)/(\times^2)^*$ ” são o que [HT74] denomina “*blocks*”, apresentados ali como uma das estruturas de dados utilizadas pelo algoritmo o que justifica o desenvolvimento da segunda metade deste capítulo e a forma final do Algoritmo 2.5.

Nosso intuito na seqüência deste texto é trazer à luz as idéias centrais nas quais o algoritmo se baseia, deixando em segundo plano as questões concernentes à implementação, trazendo a primeiro plano os conceitos centrais por trás do algoritmo formulados na linguagem usual da Teoria dos Grafos.

O leitor familiarizado com [HT74] sem dúvida reconhecerá no grafo da Figura 1.1 o mesmo grafo que acompanha o texto do artigo. A escolha deste grafo foi feita justamente com a intenção de prover um ponto de referência entre nossa apresentação e a de [HT74].

Capítulo 3

Hierarquias e Busca em Profundidade

Neste capítulo apresentaremos de maneira formal um algoritmo para teste de planaridade de um grafo que corresponde essencialmente ao conhecido algoritmo de Hopcroft e Tarjan.

Para tanto, introduziremos na Seção 3.1 um conceito que retrata mais formalmente as características de uma *Busca em Profundidade* (doravante abreviadamente referida por **BP**), e uma notação adequada para manipulá-lo: o conceito de *hierarquias*. O leitor deve percorrer esta seção ciente de que o que será definido como *hierarquia sobre um grafo* representa uma formalização do conceito de *Busca em Profundidade sobre um grafo* pretendendo capturar suas propriedades essenciais. Veremos que uma hierarquia nada mais é do que o estabelecimento explícito da (algumas vezes assim chamada) relação de “*ancestralidade*” induzida sobre os vértices de um grafo numa **BP**.

Faremos a seguir uma série de construções auxiliares a fim de estabelecer uma decomposição conveniente de um grafo em circuitos e seus respectivos segmentos, procurando dessa forma tornar eficiente a aplicação dos algoritmos do capítulo anterior. O objetivo perseguido aí é a definição de uma ordem, que denotaremos por $<'$, que estabelece uma particular seqüência em que uma base de circuitos do grafo deve ser examinada a fim de se obter um bom desempenho na execução dos algoritmos esquematizados no capítulo anterior. Prevenimos o leitor de que esta seção é talvez a de leitura mais árida em todo o texto, uma vez que trata-se de estabelecer uma notação rigorosa para a descrição de um algoritmo sofisticado. Uma sugestão de leitura desse material poderia ser percorrê-lo uma primeira vez de maneira superficial de forma a adquirir uma idéia geral do seu conteúdo e do objetivo a ser atingido, e mais tarde proceder a uma leitura cuidadosa das definições e resultados apresentados. Sempre que conveniente, cada novo conceito introduzido será acompanhado de um exemplo e de comentários apontando os paralelos com seu correspondente na já consagrada terminologia usada no estudo tradicional de **BP**.

Na Seção 3.2 reescrevemos os principais resultados e algoritmos do capítulo anterior, utilizando-

nos das construções feitas na Seção 3.1. Esta é, portanto, a seção de interesse central do capítulo, onde uma versão já completa do algoritmo é apresentada.

Um exemplo completo da execução dos algoritmos estabelecidos bem como as demonstrações dos resultados mencionados, mas não provados neste capítulo, são apresentados na Seção 4.3, que tem o papel de um “apêndice final” da primeira parte do trabalho. Optamos por deixar essas demonstrações isoladas, de modo que ao percorrer as duas primeiras seções, o leitor não tivesse sua atenção desviada por resultados auxiliares (e suas respectivas provas) que não fossem essenciais para a compreensão do algoritmo de acordo com o formalismo que pretendemos desenvolver.

3.1 Definições e Resultados Preliminares

Definição 3.1 *Uma hierarquia sobre um conjunto finito A é uma ordem $<$ sobre A onde para cada $a, b, c \in A$ tais que $b < a$ e $c < a$ temos que b e c são comparáveis. A notação $a \wedge b$ indica $\max(\{c \in A : c < a \text{ e } c < b\})$.*

Definição 3.2 *Uma hierarquia sobre um grafo G é uma hierarquia $<$ sobre V_G onde u e v são comparáveis sempre que $uv \in A_G$, e $uv \in A_G$ para todo $u < v$.*

Exemplo 3.1 *A Figura 3.1 mostra um exemplo de uma hierarquia sobre o conjunto dos vértices do grafo da Figura 1.1.*

Pelo texto a seguir, A indica um conjunto finito e $<$ uma hierarquia sobre A .

Por consequência imediata da definição de hierarquia, temos que para cada $a \in A$, existe no máximo um b tal que $b < a$. Esse elemento será denotado por $\top(a)$. Note que, segundo as convenções estabelecidas para esse texto, $\top$ é uma função parcial $A \rightarrow A$. Em particular, $\top(r) = \emptyset$ para todo $r \in \min(A)$ e para cada $a \in A$ existe um único tal r com $r < a$. Tal r será denotado por $\rho(a)$ e o único valor d tal que $\top^d(a) = r$ será denotado por $\delta(a)$. Estes conceitos são vistos mais detidamente no início da Seção 4.3.

Informalmente podemos caracterizar uma hierarquia $<$ sobre um conjunto A como sendo uma ordem sobre A cujo Diagrama de Hasse é uma floresta. O leitor é estimulado a manter essa

Figura 3.1: Uma hierarquia sobre o conjunto dos vértices do grafo da Figura 1.1

caracterização em mente ao longo do desenvolvimento deste capítulo. Então, para cada $a \in A$, $\top(a)$ é o “pai” de a no Diagrama de Hasse, $\rho(a)$ é a “raiz” da árvore do diagrama à qual a pertence.

Definição 3.3 Se $<$ é uma hierarquia sobre G , para todo $v \notin \min(V_G)$ denotamos por $\bar{v}$ a aresta $\{v, \top(v)\}$ e por F o conjunto das arestas $\bar{v}$ de G . Ou seja,

$$\begin{aligned}\bar{v} &\equiv \{v, \top(v)\} \text{ e} \\ F &\equiv \{\bar{v} : v \notin \min(V_G)\}.\end{aligned}$$

Uma hierarquia $<$ sobre um grafo G será representada de maneira ligeiramente diferente de hierarquias ordinárias: desenha-se o Diagrama de Hasse correspondente à relação $<$ sobre V_G (o que equivale a desenhar as arestas de F) e desenha-se as outras arestas de G em linhas curvas, como mostrado na Figura 3.2.

Como mencionado anteriormente, o conceito de hierarquia sobre um grafo é uma tentativa de capturar as propriedades essenciais de uma BP, a fim de dar um tratamento mais rigoroso a algoritmos baseados nessa técnica. É bastante comum que textos que tratam de BP introduzam informalmente o conceito de “ancestralidade” relativa a essa BP, dizendo que um vértice v é **ancestral** do vértice u quando v é “visitado” antes de u nessa BP. Nesse contexto, podemos dizer que uma hierarquia $<$ sobre um grafo representa explicitamente a ordem implícita no conceito de ancestralidade, de forma que quando temos $u < v$ estamos denotando de maneira formal que o vértice u é “ancestral” do vértice v ; o vértice $\top(v)$ corresponde ao que nesse contexto se chamaria o “pai” ou “ancestral imediato” de v e o vértice $\rho(v)$ é o “ancestral mais remoto” ou “patriarca” de v .

Figura 3.2: Uma hierarquia sobre o grafo da Figura 1.1

A floresta F vem a ser a “Floresta Induzida pela BP” sobre o grafo, ou a “Floresta em Profundidade”, ou seja, a floresta formada pelas arestas de G que ligam um vértice “já visitado” a um outro “ainda não visitado” conforme a busca vai sendo realizada.

Convém notar que tradicionalmente fala-se de diferentes BP com uma mesma Floresta Induzida, em função da ordem em que “vértices de mesmo nível” são “visitados”. Quando isso se fizer necessário na nossa apresentação, introduziremos explicitamente essa ordem, de forma que, de maneira rigorosa, devemos dizer que uma hierarquia sobre um grafo G representa todas as BP sobre G cuja Floresta Induzida pela busca é F .

Pelo resto desta seção, assumimos um grafo fixo G e uma hierarquia $<$ sobre G ; para cada aresta $\alpha \in A_G - F$ denotaremos por C_α o circuito fundamental de α com relação a F .

Teorema 3.4 *Toda componente de G é da forma $G[\rho^{-1}(v)]$ para algum $v \in \min(V_G)$, e para esse v , $F \cap G[\rho^{-1}(v)]$ é uma árvore geradora dessa componente.*

Prova Segue como consequência do Lema 4.9. □

Teorema 3.5 *Todo grafo admite uma hierarquia.*

Prova Segue facilmente por indução sobre V_G , utilizando o Lema 4.11. □

O Lema 4.11, que serve como base do resultado acima, foi elaborado de forma a justificar o seguinte algoritmo para a construção de uma hierarquia sobre um grafo dado.

Algoritmo 3.1

Hierarquia(G)

Entrada: Um grafo G .

Saída: A função T de uma hierarquia sobre G .

$S \leftarrow T \leftarrow \emptyset;$

Enquanto $S \neq V_G$

$s \leftarrow \max(\text{Adj}(V_G - S));$

$v \leftarrow \text{um elemento de } \text{Adj}(s) - S;$

```

Se  $v = \emptyset$ 
   $v \leftarrow$  um elemento de  $V_G - S$ ;
 $S \leftarrow S + v$ ;
 $T \leftarrow T + (v, s)$ ;
Devolva  $T$ ;

```

O leitor familiarizado com a **BP**, sem dúvida reconhecerá no Algoritmo 3.1 uma variante do algoritmo básico de Busca em Profundidade sobre um grafo.

No trecho a seguir faremos as definições de conceitos comuns no contexto de **BP**: começamos por definir a relação $\uparrow$ que corresponde à formalização do conceito de “ancestrais atingíveis a partir da árvore de **BP** usando uma única aresta fora da árvore”, freqüentemente apresentado em textos tratando de **BP**; em seguida, definimos as funções λ e λ' , nosso correspondente ao conceito de “lowpoints”, que permitirão estabelecer uma segunda hierarquia, esta sobre A_G , que também denotaremos por $<$. É interessante observar que esta nova hierarquia sobre A_G é induzida pela hierarquia já estabelecida sobre V_G , de maneira que ambas mantêm um grande paralelo entre si. Em particular, o Teorema 3.9, análogo ao Teorema 3.4, mostra-nos como caracterizar e manipular os blocos do grafo G por meio de hierarquias.

Definição 3.6 *Seja $\uparrow$ a relação de A_G em V_G dada por*

$$\alpha \uparrow v \equiv v = \min(\beta) \text{ para alguma aresta } \beta \in A_G - F, \text{ tal que } \alpha \in C_\beta.$$

Observando que

$$\alpha \uparrow v \Rightarrow v \leq \min(\alpha),$$

temos imediatamente pela definição de hierarquia que

Fato 3.7 *Para todo $\alpha \in A_G$ o conjunto $\uparrow(\alpha)$ é totalmente ordenado.*

Tendo em vista o fato acima, as seguintes funções estão bem definidas.

Definição 3.8 Para toda aresta $\alpha \in A_G$ definimos:

$$\lambda(\alpha) \equiv \min(\uparrow(\alpha));$$

$$\lambda'(\alpha) \equiv \begin{cases} \lambda(\alpha), & \text{se } |\uparrow(\alpha)| \leq 1; \\ \min(\uparrow(\alpha) - \lambda(\alpha)), & \text{caso contrário.} \end{cases}$$

A função λ é uma adaptação do conceito de “*lowpoint*”, freqüentemente encontrado na literatura por ser de grande importância em algoritmos eficientes para a determinação de biconexidade e conexidade forte de grafos[AHU74, Ter91, Szw84, HT74].

Usualmente o “*lowpoint*” é apresentado como uma função que associa a cada vértice v do grafo o seu “*ancestral mais remoto*” atingível por meio de um caminho a partir de v com no máximo uma aresta fora da árvore de BP, sendo que nesse caminho, arestas da árvore devem ser percorridas no sentido da ponta “*menos profunda*” para a “*mais profunda*” e a aresta fora da árvore no sentido inverso.

Na nossa versão, podemos entender $\lambda(\alpha)$ como o “*menor vértice*” de G que pertence a algum circuito fundamental do qual α faça parte.

A função λ' , analogamente a λ , é uma adaptação do conceito de “*segundo lowpoint*” de um vértice v , que vem a ser nada mais que o segundo “*ancestral*” de v com a mesma propriedade, desconsiderado o “*primeiro lowpoint*”. Nos casos em que não existe um “*segundo lowpoint*” ambos assumem o mesmo valor, por conveniência. O conceito de “*segundo lowpoint*” é introduzido em [HT74].

Exemplo 3.2 A Tabela 3.1 lista os valores de $\uparrow(\alpha)$, $\lambda(\alpha)$ e $\lambda'(\alpha)$ para todas as arestas $\alpha \in A_G$ na hierarquia da Figura 3.3.

Considere agora a função $\pi : A_G \rightarrow F$ dada por

$$\pi(\alpha) \equiv \begin{cases} \overline{\max \alpha}, & \text{se } \alpha \notin F; \\ \overline{\min \alpha}, & \text{se } \alpha \in F \text{ e } \lambda(\alpha) < \min(\alpha); \\ \emptyset, & \text{caso contrário.} \end{cases}$$

Como atesta o Lema 4.12, a relação π^+ é uma hierarquia sobre A_G que denotaremos por $<$, sem perigo de confusão com a hierarquia fixada sobre G , uma vez que o contexto sempre deixará claro de qual delas se trata.

Tabela 3.1: Os valores de $\uparrow(\alpha)$, $\lambda(\alpha)$ e $\lambda'(\alpha)$ para $\alpha \in A_G$ referentes ao grafo da Figura 3.2.

α	$\uparrow(\alpha)$	$\lambda(\alpha)$	$\lambda'(\alpha)$
1 2	{1}	1	1
1 12	{1}	1	1
2 3	{1, 2}	1	2
2 16	{2}	2	2
3 4	{1, 2, 3}	1	2
3 16	{3}	3	3
4 5	{1, 2, 3, 4}	1	2
4 7	{4}	4	4
5 6	{1, 2, 3, 4, 5}	1	2
5 15	{5}	5	5
6 7	{1, 2, 3, 4, 5, 6}	1	2
6 8	{6}	6	6
7 8	{1, 2, 3, 4, 5, 6, 7}	1	2
8 9	{1, 2, 3, 5}	1	2
9 10	{1, 2, 3, 5, 9}	1	2
9 15	{9}	9	9
10 11	{1, 5, 9}	1	5
10 16	{2, 3}	2	3
11 12	{1, 5, 9, 11}	1	5
11 14	{11}	11	11
12 13	{5, 9, 11}	5	9
13 14	{5, 9, 11, 13}	5	9
13 15	{13}	13	13
14 15	{5, 9, 13}	5	9

Um resultado análogo ao Teorema 3.4 pode ser formulado com relação a essa hierarquia, resultado esse que constitui o princípio de funcionamento do mais popular algoritmo para determinação de blocos e componentes biconexos de um grafo através de uma BP [AHU74, Szw84, Ter91].

Teorema 3.9 *Todo bloco de G com pelo menos uma aresta é da forma $G[\rho^{-1}(\alpha)]$ para algum $\alpha \in \min(A_G)$.*

Prova Se α é um bloco trivial de G o resultado é imediato; caso contrário o resultado segue imediatamente do Corolário 4.16. $\square$

Exemplo 3.3 *A Figura 3.3 mostra a hierarquia $<$ sobre A_G correspondente ao exemplo da Figura 3.2*

Como veremos a seguir, procederemos ao exame de cada bloco do grafo G examinando cada um de seus circuitos fundamentais, e a ordem em que esses circuitos deve ser examinada será de importância central no algoritmo.

As próximas definições têm o objetivo de estabelecer uma ordem conveniente entre os circuitos fundamentais do grafo, o que será feito estabelecendo-se uma ordem total entre as arestas de $A_G - F$. Essa ordem, que denotaremos por $<_\lambda$, induzirá uma ordem parcial sobre $A_G - F$ que denotaremos por $<'$ e que denotará formalmente quais circuitos devem envolver quais outros numa imersão no plano sem cruzamentos de G .

Assim, se α e β são arestas em $A_G - F$, então $\alpha <' \beta$ significará que o circuito C_α envolverá o circuito C_β na imersão de G no plano construída pelo algoritmo. Intuitivamente, é imediata a percepção de que essa “relação de envolvimento” $<'$ é uma hierarquia sobre os circuitos fundamentais de G se consideramos uma imersão no plano sem cruzamentos de G , onde as arestas de F são desenhadas como no diagrama da hierarquia $<$ (Figura 3.2).

Para definir a relação $<'$, atribuiremos a cada aresta α de G uma aresta de $A_G - F$ que denotaremos por $\varepsilon(\alpha)$, que indica qual o circuito fundamental “mais externo” a que α pertence, e a partir daí estabeleceremos a “hierarquia de envoltórias” $<'$ sobre $A_G - F$.

Para determinar o valor de $\varepsilon(\alpha)$ para cada aresta α de A_G será necessário estabelecer antes uma ordem total entre os circuitos fundamentais (arestas em $A_G - F$) de G que evite “cruzamentos desnecessários” entre os circuitos. Essa idéia será formalmente estabelecida pelo conceito de λ -compatibilidade de uma ordem sobre A_G com uma hierarquia sobre G : informalmente podemos dizer que uma ordenação total sobre os circuitos fundamentais de G é λ -compatível se a tentativa de construção de uma imersão no plano de G que obedeça essa ordem (com relação aos

Figura 3.3: A hierarquia $<$ sobre A_G

“envolvimentos mútuos” entre os circuitos) não cria outros cruzamentos além dos “inevitáveis”. Aqui, entram em jogo as funções λ e λ' definidas acima, pois identificaremos os circuitos “mais externos” com aqueles que contêm os “menores” vértices (segundo a hierarquia $<$).

Definição 3.10 Para cada par de arestas incomparáveis $\alpha, \beta \in A_G$ definimos α/β como a única aresta γ de A_G tal que $\alpha \wedge \beta \leq \gamma \leq \alpha$.

Note que $\alpha/\beta = \emptyset$ se e somente se α e β estão em blocos diferentes de G .

Exemplo 3.4 A Figura 3.4 mostra esquematicamente duas arestas incomparáveis α e β no grafo, bem como as arestas $\alpha \wedge \beta$, α/β e β/α . Informalmente podemos dizer que os valores de λ e λ' para α/β e β/α nos indicam qual delas será a “mais externa” na imersão de G no plano.

Figura 3.4: α/β e β/α

A Tabela 3.2 mostra os valores de α/β e para cada par de arestas $\alpha, \beta \notin F$ do grafo da Figura 3.2.

Tabela 3.2: α/β

$\downarrow \alpha \xrightarrow{\beta}$	1 12	2 16	3 16	4 7	5 15	6 8	9 15	11 14	13 15
1 12	—	10 11	10 11	7 8	1 12	8 9	1 12	1 12	1 12
2 16	10 16	—	2 16	7 8	10 16	8 9	10 16	10 16	10 16
3 16	10 16	3 16	—	7 8	10 16	8 9	10 16	10 16	10 16
4 7	4 7	4 7	4 7	—	4 7	7 8	4 7	4 7	4 7
5 15	12 13	10 11	10 11	7 8	—	8 9	5 15	14 15	5 15
6 8	6 8	6 8	6 8	7 8	6 8	—	6 8	6 8	6 8
9 15	12 13	10 11	10 11	7 8	9 15	8 9	—	14 15	9 15
11 14	12 13	10 11	10 11	7 8	11 14	8 9	11 14	—	11 14
13 15	12 13	10 11	10 11	7 8	14 15	8 9	13 15	14 15	—

Lema 3.11 *Para todo par de arestas incomparáveis α, β num mesmo bloco de G temos que $\lambda(\alpha/\beta), \lambda(\beta/\alpha), \lambda'(\alpha/\beta)$ e $\lambda'(\beta/\alpha)$ são todos dois a dois comparáveis.*

Prova Se α e β estão num mesmo bloco de G temos pelo Corolário 4.16 que $\rho(\alpha) = \rho(\beta)$ e portanto, pelo Lema 4.6, $\alpha \wedge \beta \neq \emptyset$.

Nesse caso, é imediato da definição de α/β , que $\alpha \wedge \beta < \alpha/\beta$ e daí

$$\lambda(\alpha/\beta) \leq \min(\alpha/\beta) = \max(\alpha \wedge \beta),$$

e portanto

$$\lambda(\alpha/\beta) \leq \max(\alpha \wedge \beta).$$

O resultado segue então da definição de hierarquia. $\square$

Com este resultado podemos formular a seguinte

Definição 3.12 *Uma ordem $\prec$ sobre $X \subseteq A_G$ é λ -compatível se para todo par de arestas incomparáveis α, β num mesmo bloco de G temos*

$$\alpha \prec \beta \Rightarrow (\lambda(\alpha/\beta), \lambda'(\alpha/\beta)) \leq (\lambda(\beta/\alpha), \lambda'(\beta/\alpha)).$$

De acordo com o Lema 3.11, temos que sempre é possível estabelecer alguma ordem λ -compatível total sobre $A_G - F$. Daqui por diante fixamos uma tal ordem que será denotada por $<_\lambda$.

Exemplo 3.5 O conceito de λ -compatibilidade terá importância decisiva na obtenção de uma decomposição adequada do grafo em circuitos e segmentos que possibilite um teste eficiente de planaridade.

A fim de dar ao leitor uma idéia mais concreta do seu significado, a Figura 3.5 mostra esquematicamente a situação em que duas arestas α e β de um grafo são tais que $\alpha \prec \beta$ necessariamente, se $\prec$ é uma ordem λ -compatível.

Outra observação ilustrativa do conceito consiste em notar que no diagrama da Figura 3.3 tomamos o cuidado de dispor as arestas em cada nível de forma que, quando lidas da esquerda para a direita, tal ordem é λ -compatível. Assim, uma ordenação λ -compatível completa de $A_G - F$ é obtida coletando-se (na ordem) as “folhas” do diagrama ao percorrê-lo em pré-ordem.

Definição 3.13 Para cada $\alpha \in A_G$ definimos:

$$\varepsilon(\alpha) \equiv \min_{\lambda}(\{\beta : \alpha \in C_{\beta}\}).$$

Exemplo 3.6 De acordo com as observações feitas no Exemplo 3.5, uma maneira simples de determinar o valor da função ε para cada aresta α é observar que para cada $\alpha \in A_G$, no exemplo da Figura 3.3, $\varepsilon(\alpha)$ corresponde à primeira “folha” encontrada ao percorrer a sub-árvore do diagrama que começa em α em pré-ordem.

Observando os valores das Tabelas 3.1 e 3.2 o leitor chegará à conclusão de que, no nosso exemplo, a única ordem total λ -compatível sobre $A_G - F$ é:

$$1 \ 12 <_{\lambda} 5 \ 15 <_{\lambda} 9 \ 15 <_{\lambda} 13 \ 15 <_{\lambda} 11 \ 14 <_{\lambda} 2 \ 16 <_{\lambda} 3 \ 16 <_{\lambda} 6 \ 8 <_{\lambda} 4 \ 7.$$

Alguns momentos de reflexão em torno da definição da função ε nos levam à conclusão de que $\varepsilon(\alpha) = \emptyset$ se e somente se α não está envolvida em nenhum circuito fundamental (e portanto em nenhum circuito), ou seja, se e somente se α é uma aresta de corte. Por outro lado, também é evidente dessa definição que $\alpha \in A_{C_{\varepsilon(\alpha)}}$, e portanto $G[\varepsilon^{-1}(\beta)] \subseteq C_{\beta}$ para todo $\beta \in A_G - F$. Finalmente, como para todo $\alpha < \beta$, temos necessariamente $\varepsilon(\alpha) \leq_{\lambda} \varepsilon(\beta)$ temos que $G[\varepsilon^{-1}(\beta)]$ é conexo, e portanto é um caminho em C_{β} ou é o próprio C_{β} , o que sugere as seguintes definições:

Figura 3.5: O caso geral $\alpha <_{\lambda} \beta$

Definição 3.14 Para cada $\alpha \in A_G - F$ definimos

$$\begin{aligned} P_\alpha &\equiv G[\varepsilon^{-1}(\alpha)]; \\ \pi'(\alpha) &\equiv \varepsilon(\max(A_{C_\alpha - P_\alpha})). \end{aligned}$$

De acordo com o Lema 4.17 temos que π'^+ é uma hierarquia sobre $A_G - F$, que passamos a denotar por $<'$.

Definição 3.15 Para cada $\alpha \in A_G - F$ definimos

$$\begin{aligned} S_\alpha &\equiv \biguplus_{\alpha \leq' \beta} P_\beta; \\ G_\alpha &\equiv C_\alpha \cup S_\alpha. \end{aligned}$$

Exemplo 3.7 A Figura 3.6 mostra o diagrama da hierarquia $<'$ para o nosso exemplo.

A Figura 3.7 mostra S_α para cada aresta $\alpha \in A_G - F$. P_α é mostrado em linhas destacadas e os vértices (mas não as arestas) de C_α são também mostrados, de forma que o leitor possa visualizar G_α , bastando para isso acrescentar mentalmente o caminho $C_\alpha - P_\alpha$ a S_α em cada um dos desenhos.

Veremos mais adiante, com o Teorema 3.18, que para cada par de arestas α, β tais que $\alpha <' \beta$, S_β é um C_α -segmento de G_α de forma que esta relação fornece a decomposição do grafo em circuitos e segmentos que utilizaremos no algoritmo. A disposição dos segmentos na página procura imitar a hierarquia $<'$, de forma a deixar evidente essa decomposição de G . O leitor deve notar que os segmentos em cada um dos níveis da Figura são segmentos de (algum dos) grafos no nível imediatamente acima.

Figura 3.6: A hierarquia $<'$ sobre $A_G - F$

Teorema 3.16 *Cada bloco não trivial de G é da forma $G[\rho'^{-1}(\alpha)]$ para algum $\alpha \in \min'(A_G - F)$.*

Prova Da definição de $<'$ temos que $\alpha \wedge' \beta \neq \emptyset$ se e somente se $\alpha \wedge \beta \neq \emptyset$. Daí o resultado segue aplicando-se o Lema 4.6 e o Teorema 3.9. $\square$

3.2 Hierarquias e Planaridade

A partir deste ponto, podemos começar a adaptar os resultados do capítulo anterior de forma a tirar proveito da estrutura imposta sobre o grafo pela hierarquia $<'$. Em primeiro lugar reformulamos o Corolário 1.9 da seguinte maneira

Teorema 3.17 *Um grafo G é planar se e somente se G_α é planar para cada $\alpha \in A_G - F$.*

Feito isso podemos reformular o Algoritmo 2.1 como a seguir.

Figura 3.7: G_α , P_α e S_α

Algoritmo 3.2Planar(G)**Entrada:** Um grafo G .**Saída:** sim se G é planar; caso contrário interrompe a execução.Se $|A_G| > 3|V_G| - 6$

Interrompa.

Hierarquiza(G);Para cada $\alpha \in \min'(A_G - F)$ Planar'(α);

Devolva sim.

Algoritmo 3.3Hierarquiza(G)**Entrada:** Um grafo G .**Saída:** Uma hierarquia $<'$ sobre $A_G - F$ tal como descrita acima.**Algoritmo 3.4**Planar'(α)**Entrada:** Uma aresta $\alpha \in A_G - F$.**Saída:** A representação da C_α -imersão de G_α ; interrompe a execução se a imersão não for anti-reflexiva.

Antes de poder reescrever uma nova versão de Planar' será necessário estabelecer alguns resultados que formalizam nossas observações no Exemplo 3.7. Os resultados a seguir ligam as construções auxiliares deste capítulo às noções estabelecidas no capítulo anterior, mostrando

como a hierarquia $<'$ e a decomposição dada pela Definição 3.15 nos permitem fazer um teste de planaridade tal como descrito nos Algoritmos 2.1, 2.2 e 2.4 de maneira eficiente.

Teorema 3.18 *Para toda aresta $\alpha \in A_G - F$ temos*

$$G_\alpha / C_\alpha = \{S_\beta : \alpha <' \beta\}.$$

Prova Em primeiro lugar observamos que se $\alpha, \beta, \beta' \in A_G - F$ são tais que $\alpha <' \beta$ e $\alpha <' \beta'$, então β e β' são incomparáveis por $<'$ e daí, pela definição de S_β e $S_{\beta'}$ temos que $A_{S_\beta} \cap A_{S_{\beta'}} = \emptyset$.

Resta então mostrar que

$$G_\alpha = C_\alpha + \left(\biguplus_{\alpha <' \beta} S_\beta \right),$$

o que é simples pois

$$\left(\biguplus_{\alpha <' \beta} S_\beta \right) = \left(\biguplus_{\alpha <' \beta} \left(\biguplus_{\beta <' \gamma} P_\gamma \right) \right) = \left(\biguplus_{\alpha <' \beta} P_\beta \right) = S_\alpha - P_\beta,$$

e então

$$C_\alpha + \left(\biguplus_{\alpha <' \beta} S_\beta \right) = C_\alpha + (S_\alpha - P_\alpha) = C_\alpha \cup S_\alpha = G_\alpha.$$

□

Teorema 3.19 *Para cada $\alpha \in A_G - F$, G_α é planar se e somente se sua C_α -imersão é anti-reflexiva e $C_\alpha \cup G_\beta$ é planar para todo $\alpha <' \beta$.*

Prova Se em vez de $C_\alpha \cup G_\beta$ tivéssemos $C_\alpha + S_\beta$ no enunciado, teríamos nesse resultado simplesmente uma reformulação do critério de planaridade de Tutte (Fato 2.5).

Basta então mostrar que

$$C_\alpha + S_\beta = C_\alpha \cup G_\beta.$$

Para isso note que pela definição de G_β , temos

$$G_\beta = S_\beta \cup C_\beta;$$

e portanto

$$S_\beta = G_\beta - (C_\beta - S_\beta) = G_\beta - (C_\beta - P_\beta),$$

e como $C_\beta - P_\beta \subseteq C_\alpha$, temos

$$C_\alpha + S_\beta = C_\alpha + (G_\beta - (C_\beta - P_\beta)) = C_\alpha \cup G_\beta.$$

□

Teorema 3.20 *Para cada $\alpha \leq' \beta$ temos que*

$$C_\alpha + S_\beta = G_\beta + (C_\alpha - C_\beta).$$

Além disso, $C_\alpha - C_\beta$ é um C_β -segmento de $C_\alpha + S_\beta = G_\beta$.

Prova Basta notar que

$$C_\alpha + S_\beta = ((C_\alpha - C_\beta) + C_\beta) + S_\beta = (C_\alpha - C_\beta) + (C_\beta \cup S_\beta) = G_\beta + (C_\alpha - C_\beta).$$

□

Esses resultados permitem reformular o Algoritmo 2.3 da seguinte maneira.

Algoritmo 3.5Planar'(α)**Entrada:** Uma aresta $\alpha \in A_G - F$.**Saída:** A representação da C_α -imersão de G_α ; interrompe a execução se a imersão não for anti-reflexiva. $\mathcal{I} \leftarrow \emptyset$;Para cada β tal que $\alpha <' \beta$ $\mathcal{I}' \leftarrow \text{Planar}'(\beta)$;Para cada $\{S, S'\} \in \mathcal{I}'$ Se $(C_\alpha - C_\beta) \times S$ e $(C_\alpha - C_\beta) \times S'$

Interrompa.

 $\mathcal{I} \leftarrow \text{Imersão}(C_\alpha, \mathcal{I}, S_\beta)$;Devolva $\mathcal{I}$.

Finalmente, resta mostrar a maneira de determinar quando ocorre “cruzamento entre segmentos” utilizando a decomposição dada pela hierarquia $<'$, o que faremos mediante os próximos resultados.

Teorema 3.21 *Se $\alpha, \beta, \beta' \in A_G - F$ são tais que $\alpha <' \beta$ e $\alpha <' \beta'$ com $\beta <_\lambda \beta'$, então $S_\beta \times S_{\beta'}$ se e somente se $\min(\beta') < v$ para algum $v \in V_{C_\alpha} \cap V_{C_\beta} \cap V_{C_{\beta'}}$.*

Prova Fazendo

$$u = \max(V_{S_\beta} \cap V_{C_\alpha});$$

$$u' = \max(V_{S_{\beta'}} \cap V_{C_\alpha}),$$

temos necessariamente $u' \leq u$ já que a ordem $<'$ foi definida de forma a garantir que nas condições do enunciado isto ocorre.

($\Rightarrow$) Sem perda de generalidade, suponha $u' \leq u$ e considere:

$$v = \max(\{s \in V_{S_\beta} \cap V_{C_\alpha} : s < u'\});$$

$$w = \min(\{s \in V_{S_\beta} \cap V_{C_\alpha} : u' \leq s\}).$$

Evidentemente

$$\min(\beta) \leq v < u' \leq w \leq u,$$

e então $P_{wv} \in C_\alpha / S_\beta$.

Como $S_{\beta'} \times S_{\beta}$, temos algum $v' \in P$ onde $P \in C_{\alpha}/S_{\beta} - P_{uv}$. Então necessariamente

$$\min(\beta') \leq \min(\beta) \leq v' < v.$$

($\Leftarrow$) Sejam

$$\begin{aligned} w &= \min(V_{S_{\beta}}) \cap V_{C_{\alpha}}, \\ w' &= \min(V_{S_{\beta'}}) \cap V_{C_{\alpha}}. \end{aligned}$$

Sem perda de generalidade podemos tomar

$$v = \max(\{s \in V_{S_{\beta}} \cap V_{C_{\alpha}} : s < u'\}),$$

e daí temos

$$w' < v < u' \leq u.$$

Neste caso, $P_{uv} \in C_{\alpha}/S_{\beta}$ e evidentemente $w' \notin P_{uv}$ e portanto $S_{\beta} \times S_{\beta'}$.

□

Estes teoremas poderão ser reformulados muito mais convenientemente se considerarmos as seguintes definições.

Definição 3.22 Se $\alpha, \beta \in A_G - F$ são tais que $\alpha \leq' \beta$ então definimos

$$\begin{aligned} \mu(\beta) &= \max(V_{C_{\alpha}} \cap V_{S_{\beta}}); \\ \overline{G}_{\beta} &= C_{\alpha} + \left(\bigcup_{\alpha \leq' \beta' \leq \lambda \beta} S_{\beta'} \right). \end{aligned}$$

Nessas condições, definimos a β -**imersão** de G como sendo a C_{α} -imersão de $\overline{G}_{\beta}$.

Finalmente, definimos a β -**imersão normalizada** de G pelo conjunto

$$\mathcal{I}_{\beta} \equiv \{(\overline{S}, \overline{S}') : (S, S') \in \mathcal{I} \text{ e } \min(\{v \in V_S \cap V_{C_{\alpha}} : S \in S + S'\}) < \mu(\beta)\},$$

onde $\mathcal{I}$ é a representação da β -imersão de G e cada par $(\overline{S}, \overline{S}')$ satisfaz

$$\{\overline{S}, \overline{S}'\} = \{\{v \in V_S \cap V_{C_{\alpha}} : S \in \mathcal{S}, v < \mu(\beta)\}, \{v \in V_{S'} \cap V_{C_{\alpha}} : S' \in \mathcal{S}', v < \mu(\beta)\}\},$$

com $\overline{S} \neq \emptyset$, e $\max(\overline{S}) > \max(\overline{S}')$ se $\overline{S}' \neq \emptyset$.

Essas definições nos permitem formular de maneira concisa os seguintes resultados.

Lema 3.23 *Se α , β e β' são arestas tais que $\beta <_{\lambda} \beta'$ e $\alpha <'_1 \beta$ e $\alpha <'_1 \beta'$ e $\mathcal{I}$ é uma β -imersão normalizada de G , então $\overline{G_{\beta}} + S_{\beta'}$ não é planar se e somente se para algum $(S, S') \in \mathcal{I}$ temos que $\min(\beta') < \max(\overline{S})$ e $\min(\beta') < \max(\overline{S'})$, onde $\overline{X}$ denota $X' - \{v \in X' : \mu(\beta') < v\}$.*

Prova Segue como corolário do Teorema 3.21, reformulado segundo as definições acima. $\square$

Teorema 3.24 *Se α e β são arestas tais que $\alpha <'_1 \beta$ e G_{β} é planar, então $C_{\alpha} + S_{\beta}$ é planar se e somente se $S' = \emptyset$ para cada (S, S') da β -imersão normalizada de G .*

Prova De acordo com o Teorema 3.20 temos que $C_{\alpha} - C_{\beta}$ é um C_{β} -segmento de G_{β} . Por outro lado, evidentemente, $C_{\beta}/(C_{\alpha} - C_{\beta}) = \{C_{\alpha} - C_{\beta}, C_{\alpha} \cap C_{\beta}\}$ e portanto um C_{β} -segmento S de G_{β} cruza com $C_{\alpha} - C_{\beta}$ se e somente se

$$\max(V_{C_{\alpha}} \cap V_{C_{\beta}}) < v < \max(V_{C_{\alpha}} \cap V_{C_{\beta}}),$$

o que é equivalente à proposição do enunciado. $\square$

Com esses resultados temos em mãos todos os elementos necessários para executar cada passo dos algoritmos do capítulo anterior, e podemos dar-lhes a seguinte formulação, de acordo com o desenvolvimento feito até aqui.

Algoritmo 3.6

Planar(G)

Entrada: Um grafo G .

Saída: sim se G é planar; caso contrário interrompe a execução.

Se $|A_G| > 3|V_G| - 6$;

Interrompa;

Hierarquiza(G);

Para cada $\alpha \in \min'(A_G - F)$

Planar'(α);

Devolva sim.

Algoritmo 3.7

Planar'(α)

Entrada: Uma aresta $\alpha \in A_G - F$.

Saída: A β -imersão normalizada de G_α , onde $\beta = \max_\lambda(\{\gamma \in A_G - F : \alpha \leq' \gamma\})$; interrompe a execução se G_α não for planar.

$\mathcal{I} \leftarrow \emptyset$;

Para cada β tal que $\alpha \leq' \beta$, em ordem $<_\lambda$ -crescente

$K \leftarrow \text{Planar}''(\beta)$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta)$;

Devolva $\mathcal{I}$.

Algoritmo 3.8

Planar''(β)

Entrada: Uma aresta $\beta \in A_G - F$.

Saída: O conjunto dos vértices de contato entre S_β e C_α , onde $\alpha \leq' \beta$; interrompe a execução se $C_\alpha + S_\beta$ não for planar.

$\mathcal{I} \leftarrow \text{Planar}'(\beta)$;

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta))$;

Se $S' \neq \emptyset$ para algum $(S, S') \in \mathcal{I}$

Interrompa;

Devolva $\min(\beta) \cup \left(\bigcup_{(S, S') \in \mathcal{I}} S \right)$.

Algoritmo 3.9Imersão($\mathcal{I}, K, \beta$)

Entrada: Uma aresta β , a β' -imersão normalizada $\mathcal{I}$ de G onde $\beta' <_{\Lambda} \beta$ e o conjunto K dos vértices de contato entre S_{β} e C_{α} , sendo $\alpha <' \beta$ e $\alpha <' \beta'$.

Saída: A β -imersão normalizada de G ; interrompe a execução se $\overline{G_{\beta'}} + S_{\beta}$ não for planar.

```

 $Y \leftarrow K$ ;
 $Z \leftarrow \emptyset$ ;
 $\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta))$ ;
Enquanto  $\mathcal{I} \neq \emptyset$ 
     $(S, S') \leftarrow (\mathcal{X}, \mathcal{X}') \in \mathcal{I}$  tal que  $\max(\mathcal{X})$  máximo;
    Se  $\max(S) \leq \min(Y)$ 
        Abandone;
    Se  $S' \neq \emptyset$  e  $\min(Y) \leq \max(S)$ 
        Interrompa.
     $\mathcal{I} \leftarrow \mathcal{I} - (S, S')$ 
     $Y \leftarrow Y \cup S'$ 
     $Z \leftarrow Z \cup S$ 
Se  $\max(Y) < \max(Z)$ 
     $Y \leftarrow Z$ ;
Devolva  $\mathcal{I} + (Y, Z)$ .

```

Algoritmo 3.10Normaliza($\mathcal{I}, v$)

Entrada: Uma α -imersão normalizada $\mathcal{I}$ de G e um vértice $v \in C_{T(\alpha)}$.

Saída: O conjunto $\mathcal{I}$, extraídos os vértices u tais que $v < u$ de cada S, S' em cada par $(S, S') \in \mathcal{I}$.

```

Enquanto  $\mathcal{I} \neq \emptyset$ 
     $(S, S') \leftarrow (\mathcal{X}, \mathcal{X}') \in \mathcal{I}$  tal que  $\max(\mathcal{X})$  é máximo;
    Se  $(\min(S) < v)$  ou  $(S' \neq \emptyset \text{ e } \min(S) < v)$ 
        Abandone;
     $\mathcal{I} \leftarrow \mathcal{I} - (S, S')$ ;
Se  $\mathcal{I} \neq \emptyset$ 
    Enquanto  $v < \max(S)$ 
         $S \leftarrow S - v$ ;

```

```
    Enquanto  $v < \max(\mathcal{S}')$   
         $\mathcal{S}' \leftarrow \mathcal{S}' - v$ ;  
Se  $\mathcal{S} = \emptyset$  ou ( $\mathcal{S}' \neq \emptyset$  e  $\max(\mathcal{S}) < \max(\mathcal{S}')$ )  
     $\mathcal{S} \leftrightarrow \mathcal{S}'$ ;  
Devolva  $\mathcal{I}$ .
```

Capítulo 4

Os Pingos nos 11

Este capítulo tem o papel de um apêndice desta primeira parte do trabalho, trazendo material complementar à leitura dos capítulos anteriores. Na Seção 4.1 simulamos detalhadamente a execução dos algoritmos expostos no capítulo anterior, usando como entrada o grafo da Figura 1.1; a seguir, na Seção 4.2, discutimos detalhadamente os aspectos de implementação destes algoritmos, sugerindo algumas estratégias e estruturas de dados concretas para a representação de hierarquias e das demais estruturas necessárias. Finalmente, a Seção 4.3 traz as demonstrações dos resultados mencionados e não provados no capítulo anterior sob a forma de texto corrido. O leitor deve usar a Seção 4.3 como uma seção de referência, que acompanha o texto do Capítulo 3.

4.1 Um Exemplo de Execução

Para melhor orientação do leitor, apresentamos nesta seção um “compacto dos melhores lances” da simulação dos algoritmos elaborados na seção anterior sobre o grafo da Figura 1.1, considerando as hierarquias construídas de acordo com os exemplos desenvolvidos no capítulo anterior.

Na Figura 4.1 mostramos o rastreamento dos Algoritmos invocados na execução de **Planar** sobre o grafo da Figura 1.1. As execuções das instâncias assinaladas são detalhadas mais adiante (na mesma ordem em que aparecem na figura) a fim de dar ao leitor uma idéia concreta do desenvolvimento do algoritmo (as instâncias não detalhadas correspondem a execuções “triviais” dos algoritmos, que o leitor pode acompanhar sem maior dificuldade).

Ao detalhar a execução de **Planar'** mostramos, para cada iteração do laço principal os valores da aresta β e os valores de retorno de cada algoritmo invocado; no detalhamento de **Imersão** mostramos, para cada iteração do laço principal, o valor de $\mathcal{I}$, $\mathcal{S}$, $\mathcal{S}'$ e os valores de Y e Z no início e no fim; no detalhamento de **Planar''** os valores de retorno de cada algoritmo invocado são mostrados. Em todos os detalhamentos, os valores dos parâmetros de entrada e valores de retorno são também mostrados.

Figura 4.1: Rastreamento dos algoritmos

```

Planar( $G$ );
  Hierarquiza( $G$ );
  — Planar'(1 12);
  — Planar''(5 15);
    Planar'(5 15);
    Planar''(9 15);
    Planar'(9 15);
    Normaliza(...);
    Imersão(...);
    Normaliza(...);
    Planar''(11 14);
    Planar'(11 14);
    Normaliza(...);
  — Imersão(...);
    Normaliza(...);
    Planar''(13 15);
    Planar'(13 15);
    Normaliza(...);
  — Imersão(...);
    Normaliza(...);
    Normaliza(...);
    Imersão(...);
    Normaliza(...);
  — Planar''(2 16);
  — Planar'(2 16);
    Planar''(3 16);
    Planar'(3 16);
    Normaliza(...);
    Imersão(...);
    Normaliza(...);
    Normaliza(...);
  — Imersão(...);
    Normaliza(...);
    Planar''(6 8);
    Planar'(6 8);
    Normaliza(...);
  — Imersão(...);
    Normaliza(...);
    Planar''(4 7);
    Planar'(4 7);
    Normaliza(...);
  — Imersão(...);
    Normaliza(...);

```

$\text{Planar}'(\alpha = 1\ 12)$

$\beta = 5\ 15:$

$K \leftarrow \text{Planar}''(\beta) = \{11, 9, 5\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{11, 9, 5\}, \{\})\}$

$\beta = 2\ 16:$

$K \leftarrow \text{Planar}''(\beta) = \{3, 2\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{9, 5\}, \{3, 2\})\}$

$\beta = 6\ 8:$

$K \leftarrow \text{Planar}''(\beta) = \{6\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{6\}, \{\}), (\{5\}, \{3, 2\})\}$

$\beta = 4\ 7:$

$K \leftarrow \text{Planar}''(\beta) = \{4\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{6, 5\}, \{4, 3, 2\})\}$

Devolva $\{(\{6, 5\}, \{4, 3, 2\})\}$

$\text{Planar}''(\alpha = 5\ 15)$

$\mathcal{I} \leftarrow \text{Planar}'(\alpha) = \{(\{13\}, \{11\}), (\{9\}, \{\})\}$

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{11\}, \{\}), (\{9\}, \{\})\}$

Devolva $\{11, 9, 5\}$

$\text{Planar}'(\alpha = 5\ 15)$

$\beta = 9\ 15:$

$K \leftarrow \text{Planar}''(\beta) = \{3\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{9\}, \{\})\}$

$\beta = 11\ 14:$

$K \leftarrow \text{Planar}''(\beta) = \{11\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{11\}, \{\}), (\{9\}, \{\})\}$

$\beta = 13\ 15:$

$K \leftarrow \text{Planar}''(\beta) = \{13\}$

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{13\}, \{11\}), (\{9\}, \{\})\}$

Devolva $\{(\{13\}, \{11\}), (\{9\}, \{\})\}$

$\text{Imersão}(\mathcal{I} = \{(\{9\}, \{\})\}, K = \{11\}, \beta = 11 \ 14)$

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{9\}, \{\})\}$

$\mathcal{I} = \{(\{9\}, \{\})\}$:
 $S = \{9\} \ S' = \{\}$
 $Y = \{11\} \rightarrow \{11\}$
 $Z = \{\} \rightarrow \{\}$

Devolva $\{(\{11\}, \{\}), (\{9\}, \{\})\}$

$\text{Imersão}(\mathcal{I} = \{(\{11\}, \{\}), (\{9\}, \{\})\}, K = \{13\}, \beta = 13 \ 15)$

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{11\}, \{\}), (\{9\}, \{\})\}$

$\mathcal{I} = \{(\{11\}, \{\}), (\{9\}, \{\})\}$:
 $S = \{11\} \ S' = \{\}$
 $Y = \{13\} \rightarrow \{13\}$
 $Z = \{\} \rightarrow \{11\}$

$\mathcal{I} = \{(\{9\}, \{\})\}$:
 $S = \{11\} \ S' = \{\}$
 $Y = \{13\} \rightarrow \{13\}$
 $Z = \{11\} \rightarrow \{11\}$

Devolva $\{(\{13\}, \{11\}), (\{9\}, \{\})\}$

$\text{Planar}''(\alpha = 2 \ 16)$

$\mathcal{I} \leftarrow \text{Planar}'(\alpha) = \{(\{3\}, \{\})\}$
 $\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{3\}, \{\})\}$

Devolva $\{3, 2\}$

$\text{Planar}'(\alpha = 2 \ 16)$

$\beta = 3 \ 16$:
 $K \leftarrow \text{Planar}''(\beta) = \{3\}$
 $\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta) = \{(\{3\}, \{\})\}$

Devolva $\{(\{3\}, \{\})\}$

Imersão($\mathcal{I} = \{(\{9, 5\}, \{\})\}, K = \{3, 2\}, \beta = 2 \ 16$)

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{9, 5\}, \{\})\}$

$\mathcal{I} = \{(\{9, 5\}, \{\})\}$:

$S = \{9, 5\} \ S' = \{\}$

$Y = \{3, 2\} \rightarrow \{3, 2\}$

$Z = \{\} \rightarrow \{9, 5\}$

Devolva $\{(\{9, 5\}, \{3, 2\})\}$

Imersão($\mathcal{I} = \{(\{5\}, \{3, 2\})\}, K = \{6\}, \beta = 6 \ 8$)

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{5\}, \{3, 2\})\}$

$\mathcal{I} = \{(\{5\}, \{3, 2\})\}$:

$S = \{5\} \ S' = \{3, 2\}$

$Y = \{6\} \rightarrow \{6\}$

$Z = \{\} \rightarrow \{\}$

Devolva $\{(\{6\}, \{\}), (\{5\}, \{3, 2\})\}$

Imersão($\mathcal{I} = \{(\{6\}, \{\}), (\{5\}, \{3, 2\})\}, K = \{4\}, \beta = 4 \ 7$)

$\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\beta)) = \{(\{6\}, \{\}), (\{5\}, \{3, 2\})\}$

$\mathcal{I} = \{(\{6\}, \{\}), (\{5\}, \{3, 2\})\}$:

$S = \{6\} \ S' = \{\}$

$Y = \{4\} \rightarrow \{4\}$

$Z = \{\} \rightarrow \{6\}$

$\mathcal{I} = \{(\{5\}, \{3, 2\})\}$:

$S = \{5\} \ S' = \{3, 2\}$

$Y = \{4\} \rightarrow \{4, 3, 2\}$

$Z = \{6\} \rightarrow \{6, 5\}$

Devolva $\{(\{6, 5\}, \{4, 3, 2\})\}$

4.2 Implementação

Nesta seção comentamos os detalhes de implementação do algoritmo descrito no capítulo anterior. Faremos uma breve exposição geral das estruturas de dados utilizadas e da sua forma de manipulação, e a seguir apresentaremos a descrição detalhada de cada algoritmo de forma a permitir tradução imediata para uma linguagem de programação convencional. A descrição dada aqui corresponde em linhas gerais à implementação referida no Capítulo 7.

Um grafo G será representado por um vetor G de $|V_G|$ listas. Para cada $v \in \{1, \dots, |V_G|\}$, $G[v]$ é uma lista contendo os elementos de $Adj(v)$. Essa representação é usualmente conhecida como a **representação em listas de adjacência** de G .

A hierarquia $<$ será representada pelos vetores τ e δ indexados de 1 a $|V_G|$, que correspondem de maneira óbvia às funções homônimas relacionadas a $<$. Note que, dessa forma, se u e v são vértices comparáveis por $<$, então temos que $u < v \Leftrightarrow \delta[u] < \delta[v]$.

Uma constante especial, que denotaremos por $\emptyset$ servirá para indicar valores indefinidos.

Assim, para cada $1 \leq \alpha \leq |V_G|$, se $\tau[\alpha] \neq \emptyset$ então α representa a aresta $\bar{\alpha} \in F$; um vetor de pares de inteiros $B[|V_G|+1..|A_G|]$ guardará os valores das pontas de cada aresta em $A_G - F$. Cada uma dessas arestas será representada por um inteiro $\alpha \in \{|V_G|+1, \dots, |A_G|\}$. Cada posição α desse vetor é composta de dois inteiros que denotaremos $B[\alpha].u$ e $B[\alpha].v$, representando respectivamente $\max(\alpha)$ e $\min(\alpha)$. Note que utilizando tal convenção para a representação de arestas, temos que, dado um inteiro α representando uma aresta de G , então $\alpha \in F \Leftrightarrow \alpha \leq |V_G|$.

Dois vetores de inteiros $\lambda[1..|V_G|]$ e $\lambda'[1..|V_G|]$, representarão as funções homônimas relacionadas a $<$. Note que só nos é necessário armazenar os valores de λ, λ' para as arestas em F , pois para as arestas em $A_G - F$ o valor é óbvio. Por isso, os vetores λ, λ' serão indiretamente consultados por meio das seguintes rotinas:

Algoritmo 4.1

$\lambda(\alpha)$

Entrada: Um inteiro α representando uma aresta.

Saída: O valor do inteiro representando o vértice $\lambda(\alpha)$.

Se $\alpha \in F$

 Devolva $B[\alpha].v$.

Devolva $\lambda[\alpha]$.

Algoritmo 4.2 $\lambda'(\alpha)$ **Entrada:** Um inteiro α representando uma aresta.**Saída:** O valor do inteiro representando o vértice $\lambda(\alpha)$.Se $\alpha \in F$ Devolva $B[\alpha].v$.Devolva $\lambda'[\alpha]$.

A partir daqui temos já implícita uma representação da hierarquia $<$ sobre A_G , dada pela seguinte rotina.

Algoritmo 4.3 $T(\alpha)$ **Entrada:** Um inteiro α representando uma aresta.**Saída:** O valor do inteiro representando $T(\alpha)$.Se $\alpha \in F$ Devolva $B[\alpha].u$ Se $T[\alpha] = \emptyset$ ou $\lambda(\alpha) = T[\alpha]$; Devolva $\emptyset$.Devolva $T[\alpha]$.

O leitor deve notar que, de acordo com nossa notação, há uma diferença importante entre $T[]$ e $T()$, $\lambda[]$ e $\lambda()$, etc.¹.

A partir desse ponto da descrição, passamos a não mais distinguir entre os conceitos abstratos de grafo, aresta, etc. e suas representações tal como descritas acima. Assim, por exemplo, cada vez que nos referimos a um vértice v estaremos nos referindo a um inteiro $v \in \{1, \dots, |V_G|\}$; “uma aresta $\alpha \in A_G - F$ ” será nossa maneira de nos referirmos ao inteiro $\alpha \in \{|V_G| + 1, \dots, |A_G|\}$ que a representa, e assim por diante.

Um vetor de listas $\perp[0..|V_G|]$ manterá em cada posição $\perp[\alpha]$ uma lista das arestas $\{\beta \in A_G : \alpha < \beta\}$. Cada uma dessas listas será mantida lexicograficamente ordenada de acordo com o valor do par

¹O leitor familiarizado com a linguagem de programação C++, utilizada nas nossas implementações certamente notará sua influência sobre esta e outras convenções implícitas na nossa apresentação de algoritmos ao longo do trabalho.

$(\lambda(\alpha), \lambda'(\alpha))$. A posição $\perp[0]$ terá uma lista das arestas $\alpha \in A_G$ tais que $T(\alpha) = \emptyset$. Dessa forma é facilmente visível que para cada $\alpha \in \perp[0]$, temos que $\perp[\alpha]$ é o início de uma árvore (no sentido da estrutura de dados) em que estão representadas todas e cada uma das arestas do bloco de α em G . Do mesmo modo, também é evidente que cada aresta em $\perp[0]$ é a “representante” de um bloco de G .

De acordo com nossos comentários na Seção 3.1 temos também de forma evidente que, para cada aresta $\alpha \in A_G$, o valor de $\varepsilon(\alpha)$ é dado pela primeira aresta $\beta \in A_G - F$ encontrada ao se percorrer a árvore a partir de $\perp[\alpha]$ em pre-ordem. Para cada $\alpha \in F$ esse valor é armazenado no vetor $\varepsilon[1..|V_G|]$ que será consultado indiretamente pela rotina descrita abaixo.

Algoritmo 4.4

$\varepsilon(\alpha)$

Entrada: Um inteiro α representando uma aresta.

Saída: O valor do inteiro representando $\varepsilon(\alpha)$.

Se $\alpha \in F$

 Devolva $\varepsilon[\alpha]$

Devolva α .

Com essa representação dos dados, é simples verificar que para cada aresta $\alpha \in A_G - F$ o valor $T'(\alpha)$ é o primeiro β na cadeia $T(\alpha), T(T(\alpha)), \dots$, tal que $\varepsilon(\beta) \neq \alpha$. Além disso nessas condições temos que, $\mu(\alpha) = \min(\beta)$. Esses valores serão armazenados nos vetores $T'[|V_G| + 1..|A_G|]$ e $\mu[|V_G| + 1..|A_G|]$.

Analogamente à representação da hierarquia $<$ sobre A_G , teremos também um vetor de listas $\perp'[|V_G|..|A_G|]$ onde cada posição $\perp'[\alpha]$ manterá uma lista ordenada segundo $<_\lambda$ das arestas $\{\beta \in A_G - F : \alpha <' \beta\}$. Em $\perp'[|V_G|]$ temos a lista dos elementos de $\{\alpha \in A_G - F : T'[\alpha] = \emptyset\}$.

Finalmente, uma imersão normalizada será representada por uma lista $\mathcal{I}$ de pares de listas. Cada posição dessa lista será denotada por $\mathcal{I}_S$ e $\mathcal{I}'_S$, cada uma delas por sua vez será uma lista de vértices em G mantidos em ordem não crescente segundo $<$.

Observe que dessa forma uma consulta do tipo $\max(\mathcal{I}_S)$ significa simplesmente a consulta ao primeiro elemento da lista. Do mesmo modo, todas as operações de união de conjuntos necessários em *Imersão e Planar''* são efetuadas por meio de simples concatenações de listas, graças às garantias de ordenação dos elementos desses conjuntos dada pelas hierarquias construídas. Evidentemente os conjuntos de vértices de contato devolvidos por *Planar''* são também eles listas construídas da mesma maneira.

Mais abaixo, descrevemos os algoritmos elaborados no Capítulo 3 levando em conta os comentários acima. Adotamos para a representação de listas a seguinte notação: se l e l' são listas,

então $\bar{l}$ e $\underline{l}$ indicam respectivamente seu primeiro e último elementos, $x \rightarrow l$ indica a remoção do primeiro elemento de l acompanhado do seu armazenamento em x , enquanto que $l \rightarrow x$ e $x \hookrightarrow l$ o armazenamento do valor de x no início ou no fim de l respectivamente; $l \ll l'$ indica a concatenação das listas l e l' de tal maneira que após efetuada a operação, a lista l contém o resultado da concatenação (seu primeiro elemento permanece inalterado) e a lista l' fica vazia.

Algoritmo 4.5

Planar(G)

Entrada: A representação G de um grafo.

Saída: sim se o grafo representado por G é planar; caso contrário interrompe a execução.

Se $|A_G| > 3|V_G| - 6$

Interrompa.

Hierarquia'(G);

Percorra $\perp'[0]$ fazendo para cada α

Planar'(α);

Devolva sim.

Algoritmo 4.6

Planar'(α)

Entrada: Uma aresta $\alpha \in A_G - F$.

Saída: Uma lista de pares de listas representando a imersão normalizada de G_α ; interrompe a execução se G_α não for planar.

$\mathcal{I} \leftarrow \emptyset$;

Percorra $\perp'[\alpha]$ fazendo para cada β

$K \leftarrow \text{Planar}''(\beta)$;

$\mathcal{I} \leftarrow \text{Imersão}(\mathcal{I}, K, \beta)$;

Devolva $\mathcal{I}$.

Algoritmo 4.7**Planar''**(α)**Entrada:** Uma aresta $\alpha \in A_G - F$.**Saída:** Uma lista ordenada dos vértices de contato entre S_α e $C_{T(\alpha)}$; interrompe a execução se G_α não for planar. $\mathcal{I} \leftarrow \text{Planar}'(\alpha);$ $\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu(\alpha));$ $K \leftarrow \emptyset;$ **Enquanto** $\mathcal{I} \neq \emptyset$ $(S, S') \leftarrow \mathcal{I};$ **Se** $S' \neq \emptyset$ **Interrompa.** $K \ll S;$ $K \leftarrow B[\alpha].v;$ **Devolva** K .**Algoritmo 4.8****Normaliza**($\mathcal{I}, v$)**Entrada:** Uma lista de pares de listas de inteiros representando uma imersão normalizada e um vértice v .**Saída:** A mesma lista, eliminados os seus elementos com distância maior ou igual à de v .**Enquanto** $\mathcal{I} \neq \emptyset$ $(S, S') \leftarrow \mathcal{I};$ **Enquanto** $S \neq \emptyset$ e $\overline{S} < v$ $x \leftarrow S;$ **Enquanto** $S' \neq \emptyset$ e $\overline{S'} < v$ $x \leftarrow S';$ **Se** $S \neq \emptyset$ ou $S' \neq \emptyset$ **Se** $S \neq \emptyset$ ou $\overline{S} < \overline{S'}$; $S \leftarrow S';$ $(S, S') \leftarrow \mathcal{I};$ **Abandone;****Devolva** $\mathcal{I}$.

Algoritmo 4.9

Imersão($\mathcal{I}, v, \alpha$)

Entrada: A representação de uma imersão normalizada.

Saída: A representação da α -imersão normalizada de G ; interrompe a execução se $\overline{G_\alpha}$ não for planar.

```

 $Z \leftarrow \emptyset$ ;
 $\mathcal{I} \leftarrow \text{Normaliza}(\mathcal{I}, \mu[\alpha])$ ;
Enquanto  $\mathcal{I} \neq \emptyset$ 
     $(S, S') \leftarrow \mathcal{I}$ ;
    Se  $\overline{S} \leq \underline{K}$ 
         $(S, S') \leftarrow \mathcal{I}$ ;
        Abandone;
    Se  $S' \neq \emptyset$  e  $\overline{S'} > \underline{K}$ ;
        Interrompa.
     $K \ll S'$ ;
     $Z \ll S$ ;
    Se  $\overline{K} < \overline{Z}$ 
         $K \leftarrow Z$ ;
     $(Y, Z) \leftarrow \mathcal{I}$ ;
    Devolva  $\mathcal{I}$ .
```

Na seqüência mostramos os algoritmos para a construção das estruturas de dados descritas acima, de acordo com a notação convencionada. Por simplicidade de exposição, convencionamos que todas as variáveis utilizadas nesses algoritmos são compartilhadas por todos eles, de forma que para cada limitamo-nos a descrever seu propósito, omitindo nossa convencional descrição de “Entrada” e “Saída”.

Algoritmo 4.10

Hierarquia'()

Propósito: Constrói a estrutura de dados descrita acima a partir da representação de um grafo por meio de suas listas de adjacência em G .

```

Hierarquia; Preencha cada posição de  $\perp'$  e  $\varepsilon$  com  $\emptyset$ ;
Percorra  $\perp[0]$  fazendo para cada  $\alpha$ 
    Se  $\perp[\alpha] \neq \emptyset$ 
         $f \leftarrow \text{Folhas}(\alpha)$ ;
        Enquanto  $f \neq \emptyset$ 
             $\beta \leftarrow f$ ;
            Marca( $\beta, \varepsilon, \perp'$ ); .
```

Algoritmo 4.11

Hierarquia()

Propósito: Constrói a representação da hierarquia $<$ sobre A_G .

```

V--Hierarquia;
Lambdas;
 $l \leftarrow \emptyset$ ;
Para cada  $\alpha$  de 1 até  $|A_G|$ 
     $l \leftarrow \alpha$ ;
 $l \leftarrow \text{Orderna}(l, \lambda')$ ;
 $l \leftarrow \text{Orderna}(l, \lambda)$ ;
Enquanto  $l \neq \emptyset$ 
     $\alpha \leftarrow l$ ;
     $\perp[\top(\alpha)] \leftarrow \alpha$ ;

```

Algoritmo 4.12

V--Hierarquia()

Propósito: Constrói a representação da hierarquia $<$ sobre G . Essencialmente faz uma busca em profundidade sobre G .

```

Preencha cada posição de  $\delta$  com  $\emptyset$ ;
 $n \leftarrow G$ ;
 $b \leftarrow k \leftarrow 0$ ;
Para cada  $v$  de 1 até  $|V_G|$ 
    Se  $\delta[v] = \emptyset$ 
        Hier( $G, v, 0$ );
         $k \leftarrow k + 1$ ;

```

Algoritmo 4.13

$\text{Hier}(G, v, d)$

Entrada: A representação G de um grafo sob a forma de listas de adjacência, um vértice v de G e o valor d da distância desse vértice na hierarquia em construção.

Saída: Nenhuma propriamente; simplesmente examina uma aresta do grafo fazendo as alterações convenientes em m , b , $\top$, δ e B de forma a construir a hierarquia. Corresponde a um passo de uma Busca em Profundidade.

```

 $\delta[v] \leftarrow d;$ 
Percorra  $G[v]$  fazendo para cada  $u$ 
  Se  $\delta[u] = \emptyset$ 
     $\top[u] \leftarrow v;$ 
     $m \leftarrow m + 1;$ 
     $\text{Hier}(G, u, d + 1);$ 
  Senão
    Se  $\delta[u] < d$ 
       $b \leftarrow b + 1;$ 
       $m \leftarrow m + 1;$ 
       $B[b] \leftarrow (u, v)$ 

```

Algoritmo 4.14

$\text{Marca}(\alpha)$

Propósito: Determina os valores de $\top'(\alpha)$, $\varepsilon^{-1}(\alpha)$ e $\mu(\alpha)$ preenchendo os respectivos valores nos vetores convenientes.

```

 $\beta \leftarrow \alpha;$ 
Repita
   $\varepsilon[\beta] \leftarrow \alpha;$ 
  Se  $\top(\beta) = 0$  ou  $\varepsilon[\top[\beta]] \neq \emptyset;$ 
    Abandone;
   $\beta \leftarrow \top(\beta);$ 
 $\top'[\alpha] \leftarrow \varepsilon[\top[\beta]];$ 
 $\perp'[\top'[\alpha]] \leftarrow \alpha;$ 
 $\mu[\beta] \leftarrow \top[\beta];$ 

```

Algoritmo 4.15

Lambdas()

Proposito: Determina os valores de λ e λ' para cada aresta preenchendo os respectivos valores nos vetores convenientes.

```

 $l \leftarrow \emptyset;$ 
Para  $\alpha$  de  $n + 1$  até  $m$ 
     $l \leftarrow \alpha;$ 
 $l \leftarrow \text{Ordena}(l, \lambda);$ 
Enquanto  $l \neq \emptyset$ 
     $\alpha \leftarrow l;$ 
     $\beta \leftarrow \top[\beta[\alpha].u];$ 
    Enquanto  $\beta \neq B[\alpha].v$  e  $\lambda[\beta] = \lambda[\beta']$ 
         $\lambda'[\beta] \leftarrow B[\alpha].v;$ 
        Se  $\lambda(\beta) = \emptyset$ 
             $\lambda[\beta] \leftarrow \lambda'[\beta];$ 
         $\beta \leftarrow \top[\beta];$ 

```

Algoritmo 4.16

Ordena(l, f)

Entrada: Uma lista l e uma rotina f tal que para cada elemento x de l , $f(x)$ devolve um valor inteiro entre 0 e $|A_G|$.

Saída: A lista l ordenada de acordo com os valores devolvidos por l . A ordenação é feita pelo método conhecido por “*Radix Sort*” [AHU74, Ter91] e é supostamente estável.

4.3 Demonstrações dos Resultados Mencionados

Lema 4.1 *Se π é uma relação sobre A , então π^+ é uma ordem se e somente se $a \not\sim^n a$ para todo $n > 0$ e todo $a \in A$.*

Prova A necessidade da condição é evidente. Para a suficiência, basta notar que se $a \not\sim^n a$ para todo $n > 0$, temos que π^+ é anti-reflexiva e anti-simétrica. Como π^+ é transitiva temos que é ordem. $\square$

Corolário 4.2 *No caso particular do Lema 4.1 em que π é uma função temos que a ordem π^+ é uma hierarquia, e além disso $\top = \pi$.*

Prova Se π é uma função e π^+ é uma ordem, evidentemente $\pi(a) \leq a$ para todo a e $\pi = \top$ é uma identidade. $\square$

Definição 4.3 *Para cada $a \in A$, definimos:*

$$\delta(a) \equiv \begin{cases} 0, & \text{se } a \in \min(A); \\ \delta(\top(a)) + 1, & \text{caso contrário.} \end{cases}$$

$$\rho(a) \equiv \top^{\delta(a)}(a).$$

Lema 4.4 *Para todo $a, b \in A$ temos que $a < b$ se e somente se $a = \top^{\delta(b)-\delta(a)}(b)$.*

Prova Por indução sobre $\delta(b) - \delta(a)$. $\square$

Corolário 4.5 *Se a e b são comparáveis, então $\rho(a) = \rho(b)$.*

Prova Sejam a e b comparáveis e sem perda de generalidade suponha. $a < b$. Então

$$\rho(b) = \top^{\delta(b)}(b) = \top^{\delta(b)}(\top^{\delta(b)-\delta(a)}(a)) = \top^{\delta(a)}(a) = \rho(a)$$

□

Lema 4.6 *Para todo $a, b \in A$ temos que $a \wedge b \neq \emptyset$ se e somente se $\rho(a) = \rho(b)$.*

Prova Se $a \wedge b \neq \emptyset$ temos $a \wedge b$ comparável simultaneamente a a e b e, pelo Corolário 4.5 temos $\rho(a) = \rho(b)$.

Se, por outro lado $\rho(a) = r = \rho(b)$ então temos $r < a$ e $r < b$ e evidentemente $a \wedge b \neq \emptyset$. □

Fato 4.7 *Se $v \in V_G$ então a seqüência*

$$(v, \top(v), \top^2(v), \dots, \top^k(v))$$

é um caminho de v a $\top^k(v)$ em F para todo $k \leq \delta(v)$,

Definição 4.8 *Se u e v são vértices de G então P_{uv} denota o caminho assim definido*

$$P_{uv} \equiv \begin{cases} (u, \top(u), \top^2(u), \dots, \top^{\delta(u)-\delta(v)}(u) = v), & \text{se } v < u, \\ P_{vu}^{-1}, & \text{se } u < v, \\ P_{u,u \wedge v} P_{u \wedge v, v}, & \text{se } u \wedge v \neq \emptyset. \end{cases}$$

Lema 4.9 *Para todo par de vértices u, v de G , existe um caminho de u a v em F se e somente se $\rho(u) = \rho(v)$. Além disso, F é uma floresta geradora de G .*

Prova Sejam $u, v \in V_G$. Se $u \wedge v \neq \emptyset$ (e portanto $\rho(u) = \rho(v)$, pelo Lema 4.6) evidentemente P_{uv} tal como definido acima é um caminho de u a v sobre F .

Por outro lado, se P é um caminho de u a v em G , as pontas de cada aresta de P são comparáveis entre si, e pelo Corolário 4.5, todos os vértices de P tem o mesmo valor de ρ . $\square$

Fato 4.10 *Se $\alpha = uv \in A_G - F$ é tal que $u < v$, então o circuito fundamental de uv com relação a F é*

$$C - \alpha = C_\alpha^F = (u, v)P_{vu}.$$

Conseqüentemente o conjunto dos vértices de C_α é totalmente ordenado. Isso vale para toda $\alpha \in A_G - F$.

Lema 4.11 *Seja G um grafo, $S \subset V_G$ e $\pi : S \rightarrow S$ tais que*

1. π^+ é uma hierarquia sobre $G[S]$.
2. $\text{Adj}(V_G - S) \cap S$ é totalmente ordenado por π^+ .

Seja então $s = \max(\text{Adj}(V_G - S) \cap S)$ e tome $v \in (V_G - S) \cap \text{Adj}(s)$ (no caso em que $s = \emptyset$ tome v qualquer em $V_G - S$). Nessas circunstâncias tomando-se $S + v$ no lugar de S e $\pi + (v, s)$ no lugar de π as condições 1 e 2 estão satisfeitas.

Prova Evidentemente $S + v \subseteq V_G$ e $\pi + (v, s)$ é uma função $S + v \rightarrow S + v$.

Se denotamos $(\pi + (s, v))^+$ por $<$ temos que mostrar que:

1. $<$ é uma hierarquia sobre $G[S + v]$.
Basta mostrar que para toda $uv \in G[S + v]$ temos $u < v$. Isso é simples, pois se $uv \in G[S + v]$ temos necessariamente $u \in S$.
No caso em que $s \neq \emptyset$ temos que $u < s < v$ pela escolha de v e s .
O caso contrário implicaria em $\text{Adj}(V_G - S) \cap S = \emptyset$ e portanto $G[S]$ seria uma componente de G , de forma que $v \notin G[S]$ e tal u não poderia existir.
2. $\text{Adj}(S + v) \cap (S + v)$ é totalmente ordenado por $<$.
Temos novamente dois casos a considerar.

No primeiro, temos $\text{Adj}(v) \subseteq S$ o que levaria em $\text{Adj}(V_G - (S + v)) \cap (S + v) = \text{Adj}(V_G - S) \cap S$, onde não há nada a fazer.

No caso contrário temos $\text{Adj}(V_G - (S + v)) \cap (S + v) = (\text{Adj}(V_G - S) \cap S) + v$, que certamente é totalmente ordenado por $<$ já que $\text{Adj}(V_G - S) \cap S$ o é, e $v \in \max(S + v)$.

□

Lema 4.12 *A relação π^+ é uma hierarquia sobre A_G .*

Prova De acordo com o Lema 4.2 basta verificar que $\pi^n(\alpha) \neq \alpha$ para todo $n > 0$, $\alpha \in A_G$. Isto é imediato se observarmos que $\max(\pi(\alpha)) < \max(\alpha)$ para toda $\alpha \in F$ e que $\pi(A_G) \subseteq F$. □

Fato 4.13 *A_{C_α} é totalmente ordenado por $<$ para qualquer aresta $\alpha \in A_G - F$.*

Lema 4.14 *Para todo par de arestas comparáveis $\alpha, \beta \in A_G$, existe um circuito que envolve α e β .*

Prova De acordo com o Lema 4.4, basta mostrar que para cada aresta $\alpha \in A_G$ existe um circuito envolvendo α e $\top^k(\alpha)$, para todo $0 < k < \delta(\alpha)$.

A prova será feita por indução sobre k :

$k = 1$:

Considere $v = \lambda(\alpha)$ e seja $uv \notin F$ uma aresta tal que $\alpha \in C_{uv}$.

Como $\top(\alpha) \neq \emptyset$ temos que $v < \min(\alpha)$, e então

$$C_{uv} = (u, \top(u), \dots, \max(\alpha), \min(\alpha), \top(\min(\alpha)), \dots, v)$$

e evidentemente $\top(\alpha) = \{\min(\alpha), \max(\alpha)\} \in C_{uv}$.

$k > 1$:

Considere α e $\top^k(\alpha) \neq \emptyset$:

Pela hipótese de indução temos que existe um circuito, digamos C' , envolvendo $\top^k(\alpha)$ e $\top(\alpha)$, uma vez que $\top^k(\alpha) = \top^{k-1}(\top(\alpha))$.

Por outro lado, no caso anterior mostramos que $\alpha, \top(\alpha) \in C_\beta$ para alguma $\beta \in A_G - F$ sempre que $\top(\alpha) \neq \emptyset$.

Se $C = C_\beta$ não há mais nada a fazer; caso contrário, $C \oplus C_\beta$ é um circuito envolvendo α e $\top^k(\alpha)$.

□

Corolário 4.15 *Para toda aresta $\alpha \in A_G - F$ e toda aresta $\beta \in A_{C_\alpha}$ temos $\rho(\alpha) = \rho(\beta)$.*

Prova Isso é uma consequência da composição do Lema 4.14 com o Fato 4.13. □

Corolário 4.16 *Existe um circuito envolvendo as arestas α e β se e somente se $\rho(\alpha) = \rho(\beta)$.*

Prova Sejam $\alpha, \beta \in A_G$ com $\rho(\alpha) = \rho(\beta)$.

De acordo com o Lema 4.6 temos que $\alpha \wedge \beta \neq \emptyset$, e assim, como $\alpha \wedge \beta$ é comparável tanto com α como com β , pelo Lema 4.14 temos que existem circuitos C e C' envolvendo respectivamente $\alpha, \alpha \wedge \beta$ e $\beta, \alpha \wedge \beta$. Se $C = C'$, então C é um circuito envolvendo α e β , caso contrário, $C \oplus C'$ é o circuito desejado.

Por outro lado, considere um circuito C envolvendo α e β .

Como $\{C_\alpha : \alpha \in A_G - F\}$ é uma base de circuitos de G temos que, para algum $X \subseteq A_G - F$ podemos escrever

$$C = \bigoplus_{\gamma \in X} C_\gamma.$$

temos então que todas as arestas em C_γ de para cada $\gamma \in X$ tem o mesmo valor de ρ , em particular α e β , o que conclui o resultado. □

Lema 4.17 *A relação π'^+ é uma hierarquia sobre $A_G - F$.*

Prova Por consequência imediata da definição de π' temos que

$$\pi'(\alpha) <_\lambda \alpha,$$

para toda $\alpha \in A_G - F$ e daí, pelo Corolário 4.2 temos que π'^+ é uma hierarquia. □

Comentários

Como mencionado anteriormente, o algoritmo desenvolvido e apresentado nessa parte do trabalho é essencialmente o célebre algoritmo de Hopcroft e Tarjan.

A nossa apresentação do algoritmo tem um propósito triplo:

Em primeiro lugar apresentamos uma estrutura de dados que torna conveniente o uso de HT como subrotina para teste de planaridade sem a restrição operacional de que o grafo de entrada seja um bloco, ou por outra, uma estrutura de dados que mantém informação suficiente para selecionar prontamente um (ou vários) blocos do grafo representado e aplicar somente sobre estes o teste de planaridade. Veremos na Segunda Parte do trabalho que a habilidade de efetuar esse tipo de operação eficientemente pode reduzir de maneira significativa o tempo de execução de algoritmos que fazem uso intensivo de testes de planaridade.

Em segundo lugar, a formalização apresentada para o algoritmo procura trazer à luz as idéias centrais por trás de HT, deixando em segundo plano os detalhes mais diretamente referentes à implementação. Nesse aspecto, procuramos caracterizar a idéia de Busca em Profundidade por meio de suas propriedades essenciais, ao invés de apresentá-la como uma estratégia genérica de varredura de uma estrutura de dados formada por listas ligadas encadeadas. Com tal enfoque ganhamos uma proximidade entre a implementação e o nível conceitual que nos permite justificativas bastante precisas do funcionamento dos algoritmos apresentados. cremos que tratamento semelhante dado a outros algoritmos baseados na mesma técnica podem vir a oferecer (como foi o caso no presente trabalho) “*insights*” valiosos e alternativas interessantes de implementação para algoritmos já consagrados².

Finalmente, reconhecendo que pela nossa própria intenção de apresentar justificativas rigorosas aos algoritmos e conceitos introduzidos não pudemos evitar uma certa aridez no desenvolvimento do texto, procuramos compensar o fato com profusão de comentários e exemplos que possam servir como referência a um eventual leitor interessado na compreensão do algoritmo em linhas gerais e/ou em sua implementação. Procuramos desse modo suprir uma lacuna observada por nós na literatura a respeito do algoritmo, em especial no que diz respeito à literatura da área em língua portuguesa.

Justamente por causa de nosso propósito de manter o texto acessível ao leitor primariamente interessado na implementação do algoritmo, as sugestões de implementação apresentadas na Seção 4.2 não correspondem à maneira mais eficiente possível de implementação do algoritmo. De fato, o algoritmo tal como descrito em [HT74] é implementado por meio de uma sequência de duas BPs cuidadosamente escolhidas. Se o propósito da implementação é um programa ou rotina para testes isolados de planaridade, essa alternativa é certamente mais interessante. Ainda mais, nossa implementação faz uso explícito e intensivo de recursão nos algoritmos, enquanto

²Uma proposta assim não pode ser chamada propriamente de original: podemos encontrar enfoques semelhantes em [dFR82], [Wil80], embora não deixe de ser significativo que tais descrições acabem sendo citadas na literatura como “diferentes algoritmos” para teste de planaridade em vez de simples “variantes” ou “enfoques alternativos” de HT.

deve ser notado que a implementação proposta em [HT74] implementa a recursão estrutural do problema por meio de artifícios da estrutura de dados. Novamente nossa escolha pendeu pela clareza e precisão de exposição em detrimento de mais refinada eficiência. Não obstante, o leitor que tenha acompanhado nossa exposição não terá maior dificuldade em implementar tais recursos se convenientes, bem como localizar outras passagens onde a implementação pode sofrer pequenas mudanças que possam resultar em significativa melhora quanto ao tempo de execução.

Parte II

O Problema do Subgrafo Planar Ótimo

Capítulo 5

Complexidade Computacional

O objetivo deste capítulo é demonstrar que o Problema do Subgrafo Planar Ótimo é $\mathcal{NP}$ -Difícil. Isso será feito da maneira tradicional, mostrando que um problema $\mathcal{NP}$ -Completo é polinomialmente redutível a uma “versão em problema de decisão” do **PSPO**.

Assumimos aqui que o leitor tenha alguma familiaridade com conceitos da Teoria de Complexidade Computacional, em especial no referente às classes de problemas comumente designadas por $\mathcal{P}$ e $\mathcal{NP}$, bem como com os conceitos de “redução polinomial entre problemas” e $\mathcal{NP}$ -Completeness. A terminologia aqui adotada está de acordo com a apresentada no texto de Garey/Johnson [GJ79], que indicamos ao leitor como referência dos conceitos mencionados.

Por todo este capítulo, G indica um grafo e as abreviações, **SP**, **CHB**, **SPP** e **PSPO** indicam respectivamente os seguintes problemas de decisão:

SP: Problema do Subgrafo Planar (problema [GT 27] em [GJ79])

Instância: Um grafo G e um inteiro positivo $k \leq |A_G|$.

Pergunta: G tem um subgrafo planar H com $|A_H| \geq k$?

CHB: Problema do Circuito Hamiltoniano Bipartido (variante (2) do problema [GT 37] em [GJ79]).

Instância: Um grafo bipartido G .

Pergunta: G tem um Circuito Hamiltoniano?

SPP: Problema do Subgrafo Ponderado Planar.

Instância: Um grafo ω -ponderado G e um inteiro positivo k .

Pergunta: G tem um subgrafo planar H com $\omega(H) \geq k$?

PSPO : Problema do Subgrafo Planar Ótimo:

Instância: Um grafo ω -ponderado G .

Resposta: Um subgrafo planar H de G tal que $\omega(H') \leq \omega(H)$ para todo subgrafo planar H' de G .

Trata-se de um resultado conhecido o fato de que **CHB** é $\mathcal{NP}$ -Completo [Kri75]. Este resultado será usado para provar que **SP** é $\mathcal{NP}$ -Completo em conjunto com o seguinte Lema.

Lema 5.1 *Seja G um grafo bipartido, $u, v \notin V_G$, $u \neq v$ e $G' = G + G_u + G_v$, onde*

$$V_{G_u} = V_G + u,$$

$$A_{G_u} = \{uw : w \in V_G\};$$

$$V_{G_v} = V_G + v,$$

$$A_{G_v} = \{vw : w \in V_G\}.$$

Então G tem um circuito hamiltoniano se e somente se G' tem um subgrafo planar com pelo menos $3|V_G|$ arestas.

Prova

($\Rightarrow$) Seja C um circuito hamiltoniano de G , e considere

$$H = C + G_u + G_v.$$

H é planar pois $H/C = \{G_u, G_v\}$ e evidentemente $C + G_u$ e $C + G_v$ são planares. Além disso,

$$|A_H| = |C| + |A_{G_u}| + |A_{G_v}| = 3|V_G|.$$

($\Leftarrow$) Seja $H \subseteq G'$ planar com $|A_H| \geq 3|V_G|$. Inicialmente notamos que

$$3|V_G| \leq |A_H| \leq 3|V_{G'}| - 6 = 3(|V_G| + 2) - 6 = 3|V_G|,$$

e portanto H é planar maximal com

$$|A_H| = 3|V_G|.$$

Por outro lado, pela construção de G' temos que

$$|A_{H \cap G}| = |A_H| - (|A_{G_u \cap H}| + |A_{G_v \cap H}|) \geq 3|V_G| - 2|V_G| = |V_G|,$$

e portanto $H \cap G$ tem um circuito.

Seja então C um circuito $H \cap G$. Suponha que C não é hamiltoniano e tome $w \in V_{H \cap G} - V_C$.

Como G é bipartido temos que os únicos triângulos possíveis em H envolvem necessariamente u ou v .

Como H é planar maximal, pelo Corolário 1.16 temos que w pertence a um triângulo, digamos (u, w, w', u) . Como a aresta ww' pertence a dois triângulos (também pelo Corolário 1.16) segue que $vw \in A_H$.

Considere $ab, ac \in A_C$. As mesmas observações em relação a ab e ac nos levam a concluir que $ua, ub, uc, va, vb, vc \in A_H$.

Portanto,

$$K = C + ua + ub + uc + va + vb + vc + uw + vw \subseteq H.$$

Mas K é claramente homeomorfo a K_5 como mostra a Figura 5.1, o que contradiz a escolha de H .

Logo, C é um circuito hamiltoniano em $H \cap G$, donde segue que G é hamiltoniano o que completa o Lema.

□

Figura 5.1: O subgrafo K , homeomorfo a K_5

Teorema 5.2 *O SP é $\mathcal{NP}$ -Completo.*

Prova Claramente $\text{SP} \in \mathcal{NP}$. Além disso, dada uma instância G para o **CHB**, é fácil ver que a construção de G' a partir de G (mencionada no Lema 5.1) pode ser feita em tempo polinomial em função do tamanho de G . Daí o resultado segue por consequência imediata do Lema 5.1 e do fato de **CHB** ser $\mathcal{NP}$ -Completo. □

Corolário 5.3 *O SPP é $\mathcal{NP}$ -Completo.*

Prova Basta notar que **SP** é o caso particular de **SPP** onde a ponderação ω é tal que $\omega(\alpha) = 1$ para toda $\alpha \in A_G$. $\square$

É imediato que o **SPP** é polinomialmente redutível ao **PSPO**, e com isso temos o resultado que nos propusemos ao início do capítulo:

Corolário 5.4 *O PSPO é $\mathcal{NP}$ -difícil.*

A $\mathcal{NP}$ -Compleitude do **SP** é um fato já estabelecido na literatura, como o atesta [GJ79]¹.

Além de tornar o resultado mais acessível do ponto de vista bibliográfico, a demonstração aqui apresentada é bastante mais simples e direta que a ali relatada (que envolve duas reduções intermediárias antes de chegar ao resultado final).

¹Fazemos notar entretanto que a referência dada aí ao leitor em busca de uma demonstração do resultado aponta

LIU, P.C., AND R.C., GELDMACHER [1978] , "On the deletion of nonplanar edges of a graph," *SIAM J. Comput.* (to appear).

e infelizmente, este trabalho não chegou a ser efetivamente publicado neste periódico por algum motivo, estando disponível apenas em [LG79], uma referência cujo acesso se nos mostrou um tanto difícil.

Capítulo 6

Soluções e Heurísticas Propostas

Neste capítulo apresentaremos sucintamente um apanhado das diversas soluções e heurísticas propostas para o PSPO que encontramos na literatura.

Descreveremos aqui as seguintes abordagens:

BB: um esquema de “*Branch and Bound*” proposto em [FR76].

BC: um esquema de “*Branch and Cut*” proposto em [JM93].

DH, DH', IDH e IDHL: as heurísticas deltaédricas¹ inspiradas nos resultados expostos em [FR79, FR78] e descritas em [FGG85].

WE: a heurística da roda expandida² proposta em [EFG82].

WE': a variante de WE descrita em [FGG85].

GDY: a implementação do esquema guloso³ tal como apresentada em [FGG85].

GRD: a variante do esquema guloso, proposta em [GF84].

DS: a heurística da decomposição simplicial⁴ proposta em [Leu92].

Em vez de dedicar uma seção a cada uma delas procuraremos apresentar um quadro geral dessas abordagens. O leitor deve ter em mente que o objetivo deste capítulo é o de dar uma idéia dos resultados que mais fortemente influenciaram nosso trabalho, e mesmo definiram sua linha fundamental, e a partir disso apresentar a motivação para os resultados apresentados no capítulo seguinte.

¹ “*deltahedron heuristics*”

² “*wheel expansion*”

³ “*greedy scheme*”

⁴ “*simplicial decomposition*”

Como todas estas abordagens estão bastante relacionadas entre si, optamos por fazer uma apresentação geral e tão breve quanto possível dos pontos essenciais de cada uma delas. Posteriormente apresentamos resultados computacionais relativos a cada uma delas.

BB é a primeira tentativa de solução exata do problema. Trata-se de uma aplicação do conhecido esquema de “enumeração inteligente das soluções”, denominado *Branch and Bound* [SP82, GM84, Fou84]. Sua descrição em [FR76] apresenta-o como um algoritmo para execução manual e menciona que sua aplicação a um grafo de 12 vértices (não menciona o número de arestas) tomou aos autores algo em torno de 4 horas de “processamento”. Os autores indicam seu uso para grafos “pequenos e esparsos”.

Posteriormente o algoritmo foi implementado para processamento em um computador digital e [EFG82] menciona que o processamento de um grafo completo de 10 vértices toma mais de 5 minutos de execução. O mesmo artigo reputa como impraticável a aplicação do algoritmo a instâncias maiores que 10 vértices.

Para um grafo de n vértices, o algoritmo usa a soma dos pesos das $3n - 6$ arestas de maior peso como limite superior⁵ da possível solução nas etapas de “*bounding*” e propõe uma série de refinamentos a cada grafo a ser testado para planaridade a fim de simplificar essa etapa da execução. Sendo proposto para execução manual somente, os autores se abstêm de indicar qualquer algoritmo de decisão de planaridade em particular, uma vez que quando o problema se apresenta numa instância de tamanho hábil ao “processamento manual”, o problema de decisão de planaridade pode ser resolvido facilmente da mesma forma.

Muito embora tenha suas limitações e inconvenientes, é interessante ressaltar que **BB** foi até o aparecimento de [JM93] o *único* algoritmo disponível para determinação de soluções exatas para o **PSPO**.

BC é uma tentativa (ao que tudo indica promissora) de tratamento do **PSPO** usando ferramentas de combinatória poliédrica⁶, desenvolvido usando um esquema de *Branch and Cut* para a determinação de uma solução exata para o **PSPO**.

Este esquema combina a estratégia de “*Branch and Bound*” com a chamada técnica de Planos de Corte⁷[Sch86]. Basicamente, o **PSPO** é formulado como um Problema de Programação Linear sobre um poliedro cujos vértices são as soluções do problema (estas representadas por vetores de tamanho A_G sobre $\{0, 1\}$, de forma que cada vetor representa as arestas de um subgrafo de G).

A estratégia de “*Branch and Bound*” é usada para fazer uma enumeração dos vetores “candidatos a solução”, sendo que “versões relaxadas” do Problema de Programação Linear original são usadas para se obter “*upper bounds*” para as soluções do problema. Heurísticas apropriadas também são utilizadas para a determinação de “*lower bounds*” do problema. Uma descrição parcial do poliedro é determinada por meio de conjuntos de inequações-facetadas, sendo que estas são

⁵ “*upper bound*”

⁶ “*polyhedral combinatorics*”

⁷ “*Cutting Planes*”

progressivamente acrescentadas ao “problema relaxado” à medida que soluções não viáveis são encontradas e facetas violadas por tais soluções vão sendo detectadas. Trata-se de um método exato cuja viabilidade prática só pode ser atestada conduzindo experimentos computacionais sobre grande massa de dados.

Os resultados relativos a BC apresentados em [JM93] ainda são preliminares. Neste trabalho uma implementação variante de HT (descrita em [Mut92]) é usada como subrotina para a decisão de planaridade (viabilidade) dos “candidatos à solução” e algumas heurísticas ingênuas são usadas para a obtenção de “lower bounds”.

GDY e GRD são implementações do conhecido esquema guloso⁸, doravante abreviadamente referido por GUL que pode ser brevemente descrito da seguinte forma:

- Disponha as arestas do grafo de entrada de forma não-crescente com relação aos pesos.
- Começando com um grafo vazio H , examine cada uma das arestas assim ordenadas, acrescentando a aresta examinada α a H se $H + \alpha$ permanece planar.

GDY é uma implementação direta do algoritmo acima, onde o papel de Planar é desempenhado por HT tal como descrito em [HT74]. Os autores de [FGG85] notam que, apesar do reconhecidamente bom desempenho de HT, sua execução a cada passo é o ponto crítico da implementação, respondendo pela parte mais substancial do tempo de execução. Em vista disso, os mesmos autores reputam como impraticável a sua utilização para instâncias “grandes” do problema⁹. Por outro lado, como veremos mais adiante (ao apresentar os resultados experimentais), a qualidade das soluções fornecidas pelo algoritmo é bastante boa, não obstante a simplicidade conceitual da heurística.

GRD é uma implementação alternativa de GUL que tenta contornar os inconvenientes operacionais de GDY, optando por um algoritmo diferente para o papel de Planar. Os autores de [GF84] sustentam que HT ou qualquer outro algoritmo de desempenho linear para teste de planaridade conhecido (tais como o proposto em [LEC67] na implementação proposta por [BL76] ou o proposto em [dFR82]) são *inadequados para testes incrementais de planaridade* tais como os requeridos numa heurística como GUL (onde execuções sucessivas são realizadas para testar se um grafo $H + \alpha$ é planar já cientes de antemão que H é um grafo planar¹⁰). Por essa razão defendem a utilização do algoritmo proposto em [FW66] (um algoritmo de desempenho pessimista quadrático) para os testes de planaridade.

⁸ “greedy scheme”

⁹ As maiores instâncias para as quais [FGG85] apresenta resultados são grafos completos de 40 vértices.

¹⁰ Esses inconvenientes serão ampla e detalhadamente discutidos no capítulo subsequente.

Duas versões de GUL são propostas em [GF84]. Na primeira assume-se a hipótese simplificada de que a cada teste de planaridade sobre $H + \alpha$, H tem um circuito hamiltoniano conhecido. Essa hipótese simplifica sobremaneira os cálculos¹¹. A segunda versão trata do caso geral, sem assumir essa hipótese, mas, antes de cada novo teste de planaridade, procura-se detectar um “circuito longo” no grafo a ser testado a fim de se obter alguma simplificação nos cálculos. Alguns resultados são apresentados sobre a estrutura de grafos planares e o acréscimo de arestas de forma a induzir o uso da estrutura de dados proposta para essa segunda versão.

Em [GF84] há uma extensiva descrição de cada uma das versões sob a forma de “pseudo-código” (Pidgin Pascal). Só a primeira delas (a que trabalha sob a hipótese de existência de um circuito hamiltoniano conhecido) foi implementada, e os seus tempos de execução comparados aos de GDY; trazemos esses valores mais adiante, onde expomos um quadro geral das heurísticas discutidas nesse capítulo. A versão para o caso geral do problema não havia sido ainda implementada quando da publicação de [GF84].

WE, WE', DH, DH', IDH, IDH' e DS são todas elas heurísticas que contornam o problema de testes sucessivos de planaridade: cada uma delas parte de um subgrafo completo de 4 vértices do grafo G original e acrescenta sucessivamente vértices e arestas a este subgrafo de forma a mantê-lo sempre maximal planar. Todas elas partem da hipótese (irrestritiva quanto a generalidade do problema) de que o grafo G original é completo (eventualmente com arestas de peso 0).

WE e WE', DH e DH', IDH e IDH' diferem entre si respectivamente pela estratégia de escolha do subgrafo completo de 4 vértices inicial. Essas estratégias podem ser descritas esquematicamente da seguinte forma (os nomes são os mesmos dados em [FGG85]):

Algoritmo 6.1

AVARICE(G, ω)

Entrada: Um grafo completo G ω -ponderado.

Saída: Um subgrafo completo de quatro vértices de G .

Comentário: Processo de escolha do subgrafo inicial em WE, DH, IDH.

$ab \leftarrow$ a aresta de maior peso de G ;

$c \leftarrow$ um vértice de $V_G - \{a, b\}$ tal que $G[\{a, b, c\}]$ tem o maior peso possível;

$d \leftarrow$ um vértice de $V_G - \{a, b, c\}$ tal que $G[\{a, b, c, d\}]$ tem o maior peso possível;

Devolva $G[\{a, b, c, d\}]$.

¹¹Note o leitor, entretanto, que sob a mesma hipótese o desempenho de HT seria notavelmente superior, como fica claro a partir da nossa exposição na Primeira Parte deste trabalho.

Algoritmo 6.2SELECT(G, ω)**Entrada e Saída:** Idênticos aos do Algoritmo 6.1.**Comentário:** Processo de escolha do subgrafo inicial em WE', DH', IDH' e DS.
$$l \leftarrow \text{a lista } (v_1, v_2, \dots, v_{|V_G|}) \text{ dos vértices de } G \text{ tal que } \sum_{v_i \in \alpha} \omega(\alpha) \geq \sum_{v_{i+1} \in \alpha} \omega(\alpha);$$
Devolva $G[\{v_1, v_2, v_3, v_4\}]$.

Claramente temos que os Algoritmos 6.1 e 6.2 podem ser implementados com desempenho pessimista $O(n^2)$.

As heurísticas WE e WE', uma vez escolhido o subgrafo inicial H , adicionam a cada passo um novo vértice em $G - H$ realizando a operação reversa de uma *contração de aresta* do grafo, isto é, um vértice v em $G - H$ é escolhido, um vértice u adjacente a v em H é escolhido e a aresta uv é acrescentada a H . Então algumas arestas do tipo uw em H são convenientemente substituídas por arestas do tipo vw , de forma a manter o grafo H maximal planar. As estratégias de escolha dos vértices v, u e das arestas uw é essencialmente uma estratégia *gulosa*, no sentido em que se procura tais vértices de forma a maximizar o peso do grafo resultante da transformação.

A Figura 6.1 mostra esquematicamente um passo do algoritmo, antes e depois da inclusão do vértice v e respectivas arestas, destacando as arestas retiradas e as novas arestas colocadas em H . O desenho mostra em detalhe $H[u + \text{Adj}(u)]$ e $H[u + v + \text{Adj}(\{u, v\})]$.

Figura 6.1: Um Passo da Construção em WE

Em [EFG82] são apresentados resultados mostrando que qualquer grafo maximal planar pode ser obtido a partir de uma seqüência dessas operações convenientemente escolhida, de forma que uma modificação da heurística que a transformasse num esquema de *backtracking* seria um algoritmo para determinar a solução exata do problema (entretanto essa implementação é simplesmente sugerida e reputada impraticável pela estimativa de seu tempo de execução). Uma estrutura de dados simples (uma variante do esquema de representação do grafo por meio de *listas de adjacência*, onde uma particular ordem dos vértices deve ser mantida a cada passo do algoritmo) é apresentada para tornar eficiente o processamento das operações necessárias ao algoritmo.

Uma análise da complexidade do algoritmo (excluindo a etapa de inicialização) é apresentada concluindo com o resultado $O(n^4)$ para o caso pessimista. Um comentário é feito de que a experiência computacional de execução da implementação sobre uma série de grafos aleatoriamente gerados dá um desempenho $O(n^3)$ e comenta que esse será o caso quando se tem uma garantia de que há um limite conhecido para o grau máximo dos vértices. Os resultados obtidos a partir dos testes da implementação são apresentados e comparados com o valor da solução exata (para instâncias de 10 vértices) ou com o valor da soma dos pesos das $3n - 6$ arestas de maior peso (para instâncias de 20, 30 e 40 vértices).

DH, DH', IDH e IDH' partem também de um subgrafo completo de quatro vértices H de G escolhido tal como em WE e WE'. Ao longo da execução são mantidas uma lista l dos vértices de $G - H$ tal como a obtida no Algoritmo 6.2 e uma representação de uma 2-base B de H (um conjunto de triângulos) de H . A cada passo da execução, o primeiro vértice v de l é retirado e um triângulo $\{a, b, c\}$ de B é escolhido de forma a obter o maior valor possível para $\omega(\{av, bv, cv\})$. Então v , av , bv e cv são acrescentados a H e B é convenientemente ajustado.

A Figura 6.2 mostra esquematicamente um passo do algoritmo, antes e depois da inclusão do vértice v e respectivas arestas, destacando as novas arestas colocadas em H . O desenho mostra em detalhe um triângulo abc antes e depois do acréscimo do vértice v .

Figura 6.2: Um Passo da Construção em DH

DH foi proposto pela primeira vez em [FR78]. Um estudo das propriedades de grafos construídos segundo esse processo é apresentado em [FR79], dando as justificativas teóricas necessárias para a elaboração de um algoritmo. Em [FGG85] as heurísticas DH', IDH e IDH' são propostas como alternativas de implementação de DH.

O desempenho dessa heurística (excluindo a etapa de inicialização) é $O(n^2)$. Resultados experimentais são apresentados mais abaixo.

IDH e IDH' representam o mesmo algoritmo de DH e DH' respectivamente, acrescido de uma etapa posterior onde se tenta melhorar a qualidade da solução fornecida por essas últimas. Duas estratégias são utilizadas para obter essas melhorias:

- Para cada circuito (a, b, c, d) tal que $ac \in H$, $bd \notin H$ e $\omega(ac) < \omega(bd)$, substitui-se ac por bd .
- Cada vértice v de grau 3 é removido e recolocado em H segundo a mesma estratégia descrita acima. Em alguns casos essa operação pode levar a um grafo de peso maior que o original.

Justificativas para essas estratégias de melhoria da solução final também aparecem em [FR79]. A complexidade da etapa de melhoria das soluções é $O(n^3)$.

DS pode ser vista como uma extensão da idéia básica de DH', onde a cada passo pode-se escolher não somente um, mas também três novos vértices que são acrescentados ao grafo em construção juntamente com as convenientes arestas de forma a manter o grafo maximal planar. O critério de escolha do(s) novo(s) vértice(s), do triângulo envolvido e do tipo de acréscimo (um ou três novos vértices) é a maximização do aumento do peso do grafo obtido depois de cada alteração.

A Figura 6.3 mostra esquematicamente um passo do algoritmo no caso da inclusão de três novos vértices (o caso em que um único vértice é incluído é idêntico ao caso geral de DH, tal como ilustrado na Figura 6.2). O desenho mostra em detalhe um triângulo abc antes e depois do acréscimo dos novos vértices

Uma análise do desempenho pessimista de **DS** é feita em [Leu92], concluindo que a heurística tem complexidade $O(n^4 \log n)$. Entretanto essa análise é feita sob a suposição de que os triângulos do grafo estão representados com o auxílio de *Priority-Queue Trees* [BL76], exigindo, portanto a implementação de uma estrutura de dados mais sofisticada do que as outras alternativas.

6.1 Resultados Experimentais Disponíveis

Nesta seção apresentamos um resumo dos resultados experimentais disponíveis a respeito das heurísticas comentadas na seção anterior. Nossa principal intenção é estabelecer termos de comparação entre os seus desempenhos, tanto no que diz respeito à qualidade da solução fornecida por cada uma delas como no seu tempo de execução observado.

Nas tabelas a seguir, apresentamos um extrato dos resultados experimentais apresentados nos trabalhos mencionados acima. Em geral procuramos extrair e/ou elaborar os resultados apresentados nesses trabalhos somente na medida em que fossem significativos para uma análise comparativa do desempenho das heurísticas em foco.

Os dados das tabelas abaixo são provenientes das seguintes fontes:

BC: Os dados são apresentados em [JM93]. As instâncias são grafos completos ponderados e a implementação foi feita numa SUN SPARCStation 10 Model 20 utilizando uma biblioteca genérica de rotinas para esquemas de *Branch and Cut* escrita em C.

GRD e GDY*: Dados de [EFG82]. Não há informação quanto ao tipo de instâncias utilizadas. A implementação foi feita num Burroughs 6930. Não há indicação da linguagem utilizada, mas o contexto do artigo leva a crer numa implementação em Pascal. Também não são apresentados os valores das soluções obtidas; somente é feito o comentário de que as soluções obtidas a partir de **GRD** tem qualidade ligeiramente inferior à das obtidas por **GDY*** que por sua vez podemos supor da mesma qualidade de **GDY**.

BB e WE*: Dados de [EFG82]. As instâncias são grafos completos com os pesos das arestas calculados segundo uma distribuição *normal* de média e variância controladas, pelo método descrito em [BM76]. A implementação foi feita num Burroughs 6700 em Pascal.

WE, WE', DH, DH', IDH, IDH' e GDY: Dados de [FGG85]. As instâncias são grafos completos com os pesos das arestas calculados segundo uma distribuição *normal* de média e variância controladas, pelo método descrito em [BM76]. A implementação foi feita num Burroughs 6903, sob Mark 3.0. **DH, DH', IDH e IDH'** foram escritos em Burroughs Algol e as demais em Pascal.

DS: Dados de [Leu92]. As instâncias são grafos completos com os pesos das arestas calculados segundo uma distribuição *normal* de média e variância controladas, pelo método descrito em [BM76]. A implementação foi feita numa SUNSparcStation. Não há indicação quanto à linguagem utilizada.

A Tabela 6.1 mostra para algumas das heurísticas o valor da razão

$$\frac{\text{valor da solução obtida}}{\text{valor da solução exata}}$$

em termos percentuais. Cada linha diz respeito a um conjunto de instâncias de 10 vértices construídas tal como descrito acima, com o *desvio padrão* sobre a distribuição dos pesos das arestas indicado na coluna σ . Cada posição da tabela indica uma média calculada a partir dos valores observados para cinco instâncias aleatoriamente geradas. Deve ser notado que, exceto para **WE*** as instâncias são exatamente as mesmas em cada teste. O valor da solução exata em cada teste foi determinado por **BB**. A última linha da tabela mostra a média dos valores da coluna, o que pode ser interpretado como um índice do desempenho médio da respectiva heurística.

Tabela 6.1: Valores de Soluções Aproximadas sobre Grafos Completos de 10 vértices (% sobre o valor do limite superior)

σ	GDY	WE*	WE	WE'	DH	DH'	IDH	IDH'
5	99,734	99,400	98,567	99,517	99,461	99,710	99,661	99,855
10	99,808	99,160	98,612	98,536	98,870	98,979	99,342	99,269
15	99,716	98,040	98,615	98,602	98,730	98,976	99,692	99,236
20	99,346	97,760	98,292	97,905	98,156	99,113	98,925	99,451
25	99,415	96,720	97,552	97,009	97,510	98,956	98,748	99,187
30	99,508	—	97,675	97,037	96,894	98,808	98,238	99,530
Média	99,588	98,216	98,219	98,101	98,270	99,090	99,101	99,421

A Tabelas 6.2, 6.3 e 6.4 mostram o valor da razão

$$\frac{\text{valor da solução obtida}}{\text{soma das } 3n - 6 \text{ arestas de maior peso}}$$

em termos percentuais, para $n = 20, 30, 40$ respectivamente. Cada linha diz respeito a um conjunto de instâncias de n vértices com a mesma interpretação que na Tabela 6.1. Deve ser notado que, exceto para **WE*** e **DS**, as instâncias são exatamente as mesmas em cada teste, e que no caso $n = 40$ cada média é dada sobre um conjunto de 2 (e não 5) instâncias diferentes, exceto nessas mesmas colunas.

A Tabela 6.5 mostra um interessante indicativo do desempenho global comparativo das heurísticas estudadas em [FGG85], onde cada uma delas é testada com o mesmo lote de 102 instâncias geradas conforme o procedimento descrito acima. Em cada posição da tabela, anotamos a porcentagem de vezes em que cada heurística teve o melhor dos desempenhos para uma mesma instância. A última linha da tabela mostra a média dos valores da coluna, como um indicativo do desempenho geral da heurística.

Tabela 6.2: Valores de Soluções Aproximadas sobre Grafos Completos de 20 vértices (% sobre o valor do limite superior)

σ	GDY	WE*	WE	WE'	DH	DH'	IDH	IDH'	DS
5	98,884	97,720	98,172	98,082	98,140	98,531	98,545	98,744	99,80
10	98,104	95,540	96,953	97,019	97,065	97,084	97,792	98,057	97,50
15	97,035	93,880	95,510	95,269	95,070	95,621	96,059	96,172	96,19
20	92,514	92,560	93,891	93,569	94,336	94,957	95,036	95,977	95,69
25	94,914	90,880	92,520	92,698	92,415	93,182	93,689	94,799	94,39
30	94,801	—	92,276	91,718	90,616	92,882	93,584	93,712	93,19
Média	96,042	94,116	94,887	94,726	94,607	95,376	95,784	96,244	96,423

Tabela 6.3: Valores de Soluções Aproximadas sobre Grafos Completos de 30 vértices (% sobre o valor do limite superior)

σ	GDY	WE*	WE	WE'	DH	DH'	IDH	IDH'	DS
5	97,324	97,460	97,604	97,744	97,804	97,960	98,024	98,413	98,24
10	96,824	95,480	96,080	95,793	95,980	96,341	96,508	96,827	96,66
15	95,410	93,260	95,670	93,450	94,388	94,695	94,609	95,298	95,43
20	94,878	91,400	92,404	92,686	92,730	94,121	94,003	94,901	93,95
25	94,184	89,860	91,435	91,565	91,856	92,139	95,081	93,785	93,20
30	94,890	—	92,723	91,471	91,398	92,965	93,677	94,456	92,04
Média	95,585	93,492	93,986	93,785	94,025	94,703	94,984	95,613	95,507

Tabela 6.4: Valores de Soluções Aproximadas sobre Grafos Completos de 40 vértices (% sobre o valor do limite superior)

σ	GDY	WE	WE'	DH	DH'	IDH	IDH'	DS
5	97,786	97,514	97,514	97,656	97,798	98,103	98,050	98,04
10	96,537	95,081	94,771	95,781	96,047	96,576	96,459	96,38
15	94,774	92,962	92,435	93,265	93,403	94,133	94,515	94,69
20	94,338	90,735	91,039	92,098	92,102	93,148	93,338	93,62
25	93,417	89,644	90,249	91,660	91,752	92,834	92,388	92,38
30	91,529	88,816	88,314	89,216	90,299	91,235	91,066	91,45
Média	94,730	92,459	92,387	93,280	93,567	94,338	94,303	94,985

Tabela 6.5: Porcentagem de Instâncias com o Melhor Desempenho Comparativo

n	GDY	WE	WE'	DH	DH'	IDH	IDH'
10	46,67	0,00	0,00	3,33	6,66	26,67	36,67
20	53,33	0,00	0,00	0,00	3,33	13,33	36,67
30	50,00	0,00	0,00	0,00	0,00	10,00	40,00
40	66,67	0,00	0,00	0,00	0,00	25,00	8,33
	52,94	0,00	0,00	0,98	2,94	17,65	34,31

Um outro indicativo interessante é observar o pior desempenho numa instância particular, dentre as mencionadas nestes resultados. A Tabela 6.6 mostra esses valores (também sob forma percentual) para cada uma das heurísticas:

Tabela 6.6: Pior Desempenho Observado (% sobre o valor do limite superior)

GDY	91,060
WE	91,002
WE'	89,481
DH	87,335
DH'	90,894
IDH	89,740
IDH'	90,977

Finalmente, na Tabela 6.7 apresentamos os tempos de execução observados nas execuções mencionadas. Apesar de se referirem a implementações diferentes, feitas por pessoas diferentes em ambientes operacionais diferentes, a comparação ainda assim é significativa se observamos a razão de crescimento desses tempos em função do tamanho da instância processada. Os tempos apresentados são as médias dos tempos de execução observadas no lote de instâncias comentadas acima. Tempos de execução reportados sobre instâncias que não fossem grafos completos e ponderados não são mencionados (exceto, possivelmente no caso de GRD e GDY* onde não há informação sobre o tipo da instância utilizada).

Deve-se notar também que [FGG85] não deixa claro ao reportar os tempos de execução de WE, DH e IDH, a qual das alternativas de inicialização (SELECT ou AVARICE) esses tempos se referem, nem se os tempos de execução dessa etapa de inicialização estão arrolados nos valores apresentados. Pelo contexto, entretanto, induzimos que os tempos de inicialização *não* estão computados nesses valores.

Tabela 6.7: Tempos de Execução (em segundos de CPU)

n	BB	BC	GDY	GDY*	GRD	WE*	WE(')	DH(')	IDH(')	DS
10	300	∞	2,0	2,0	0,07	0,49	0,27	0,15	0,17	—
20	—	∞	20,2	20,4	0,28	7,06	3,67	0,38	1,82	0,90
30	—	—	71,3	70,5	0,72	29,38	13,41	0,72	7,37	7,60
40	—	—	165,8	166,3	1,61	70,61	39,49	1,13	19,29	37,05
50	—	—	—	—	—	159,02	—	—	—	—

Comentários

Observando-se os dados acima, um fato que chama a atenção é a boa qualidade das soluções aproximadas obtidas, inesperadas tanto pela complexidade computacional do problema, como pela relativa simplicidade conceitual das heurísticas.

Dentre as soluções aproximadas estudadas, podemos ver claramente que **GUL** e **IDH'** apresentam os melhores desempenhos, e aqui chegamos a um impasse: se por um lado **GUL** parece ter um desempenho mais estável, por outro exige um tempo de processamento significativamente maior, além de exigir a implementação de um algoritmo de desempenho ótimo para a decisão de planaridade. O estudo comparativo apresentado em [FGG85] conclui sugerindo **IDH'** como uma solução de compromisso entre a qualidade da solução obtida contra a demanda de recursos computacionais.

Outra observação que deve ser feita é que nenhum resultado é mencionado para instâncias maiores que um grafo completo de 50 vértices, o que pode ter duas explicações: os recursos computacionais (tempo de processamento/memória) exigidos e o tipo de problema prático que motivou a publicação de parte dos resultados apresentados, onde as maiores instâncias esperadas não excedem esse tamanho (cf. [EFG82]).

O texto de [JM93] é bastante profuso em dados experimentais do desempenho de **BC**, ilustrando o comportamento geral de sua implementação sobre diversas classes de grafos, além dos que nos interessavam para efeitos de comparação com as demais soluções propostas. Suas conclusões podem ser resumidas nos seguintes pontos:

- A implementação tem um desempenho satisfatório sobre grafos esparsos e não ponderados (ou seja, com pesos exclusivamente 0 ou 1). Os resultados apresentados sobre grafos densos referem-se a instâncias de no máximo 20 vértices.
- No caso de grafos densos e ponderados, a implementação ainda não obtém um desempenho satisfatório: os testes foram elaborados de forma a terminar a execução depois de 2 minutos de processamento: em todas as instâncias densas e ponderadas apresentadas o programa encerra a execução sem encontrar a solução do problema¹².
- Apesar de ressaltar o fato de que [JM93] apresenta *pela primeira vez* a solução de algumas instâncias constantes na literatura, os autores conjecturam que uma melhora significativa no desempenho do programa adviria do uso de melhores heurísticas na fase de determinação dos limites inferiores das soluções parciais, a apontam este como o próximo passo no sentido de obter “a useful algorithm”.

¹²o artigo afirma que, de acordo com sua experiência, um maior limite no tempo máximo de execução não traz melhoras significativas à solução encontrada.

Capítulo 7

O Método Guloso

Como visto no capítulo anterior, o *Método Guloso* (doravante abreviado por **GUL**), apesar de sua simplicidade conceitual, tem um desempenho inusitadamente bom para o tratamento de um problema desse tipo. Esse desempenho foi o que mais nos despertou a atenção quando pela primeira vez tomamos contato com os resultados experimentais publicados em torno do **PSPO** e sua qualidade foi decisiva no rumo tomado por este trabalho.

Uma análise comparativa dos resultados das diferentes abordagens para o **PSPO** indica que, se por um lado a tentativa de uma solução exata para o problema é muito dispendiosa, métodos heurísticos encontram soluções aproximadas bastante satisfatórias. Dentre as heurísticas utilizadas para a obtenção de soluções aproximadas, **GUL** parece ter um desempenho francamente superior em termos da qualidade da solução encontrada e uniformidade de comportamento. Esta heurística, porém, apresenta dois inconvenientes operacionais:

- Exige a implementação de uma subrotina para teste de planaridade, o que torna a implementação trabalhosa devido à sofisticação dos algoritmos eficientes de teste de planaridade.
- Tem complexidade computacional $O(|V_G|^3)$ no caso geral (grafos densos), o que a torna impraticável para instâncias muito grandes do problema.

A julgar pelos trabalhos publicados, esses inconvenientes parecem ter constituído razão suficiente para se abrir mão do bom desempenho de **GUL** em favor de outras heurísticas mais fáceis de serem implementadas e com tempo de resposta menor.

Em vista disso, procuramos contribuir no sentido de diminuir o impacto desses inconvenientes, tornando defensável — e até mesmo interessante — o uso de **GUL** na busca de soluções aproximadas do problema.

Essas melhorias foram feitas a partir de nossa experiência com a implementação de **HT**, que nos levou a reformular o processo de testes sucessivos de planaridade fazendo uso de uma estrutura de dados baseada na nossa exposição na primeira parte deste trabalho. A descrição desses

algoritmos aparece na Seção 7.2 e depende fundamentalmente da compreensão do algoritmo de teste de planaridade exposto na Primeira Parte. Os resultados experimentais que obtivemos são apresentados na Seção 7.3.

Outro aspecto que despertou nossa atenção no decorrer deste trabalho foi o fato de que, embora as heurísticas até aqui propostas tenham apresentado um bom desempenho quanto à qualidade das soluções encontradas, não encontramos na literatura qualquer pesquisa mais profunda no sentido de justificar tal desempenho. Trabalhando nesse sentido, a partir da experiência adquirida com os testes experimentais das sucessivas versões de **GUL** implementadas, chegamos a estabelecer um *limite inferior* (*lower bound*) quanto à qualidade da solução encontrada por **GUL** e mostramos que tal limite é o mais acurado possível, constituindo portanto o limite de desempenho absoluto¹ de **GUL** para o **PSPO**. Esses resultados são apresentados na Seção 7.1.

7.1 Garantias de Otimalidade

Nesta seção mostraremos que um subgrafo planar H de um grafo G obtido por **GUL** não pode ter peso aquém de um certo limite, e ainda, que este limite é o mais acurado possível.

Para simplificar a discussão, convencionemos chamar de **subgrafo guloso** a um subgrafo obtido na execução de **GUL** e de **floresta ótima** a uma floresta geradora de peso máximo.

Lema 7.1 *Se G é um grafo planar, F uma floresta maximal de G e u, v são dois vértices de G tais que $F + uv$ é ainda uma floresta, então $G + uv$ é planar.*

Prova Se $F + uv$ é uma floresta, então u e v estão em diferentes componentes de F , e como F é uma floresta maximal de G podemos concluir que u e v estão também em diferentes componentes de G . Logo, $G + uv$ é planar, uma vez que G é planar. $\square$

Com isso em mente, podemos postular:

Lema 7.2 *Seja G um grafo ω -ponderado. Então toda floresta ótima de um subgrafo guloso de G tem o mesmo peso de uma floresta ótima de G .*

Prova Seja G um grafo ω -ponderado e seja $A_G = \{\alpha_1, \alpha_2, \dots, \alpha_{|A_G|}\}$ tal que $\omega(\alpha_i) \geq \omega(\alpha_{i+1})$ para todo $1 \leq i < |A_G|$.

¹ “absolute performance ratio”

Seja $F_0 = H_0 = (V_G, \emptyset)$ e para cada $1 \leq i < |A_G|$

$$F_i \equiv \begin{cases} F_{i-1} + \alpha_i, & \text{se } F_{i-1} + \alpha_i \text{ é uma floresta;} \\ F_{i-1}, & \text{caso contrário.} \end{cases}$$

$$H_i \equiv \begin{cases} H_{i-1} + \alpha_i, & \text{se } H_{i-1} + \alpha_i \text{ é planar;} \\ H_{i-1}, & \text{caso contrário.} \end{cases}$$

Sem dificuldade alguma reconhecemos $F_{|A_G|}$ como sendo uma floresta ótima e $H_{|A_G|}$ como um subgrafo guloso de G . Além disso, pelo Lema 7.1 temos que se $\alpha_i \in F_i$ então $\alpha_i \in H_i$, ou seja, $F_{|A_G|} \subseteq H_{|A_G|}$. $\square$

A fim de simplificar o enunciado dos resultados seguintes, considere a seguinte definição.

Definição 7.3 Se G é um grafo e F uma floresta de G , então $\mathcal{C}_F^G$ denota o conjunto das arestas de $G - F$ que ao serem acrescentadas a F formam um circuito, ou seja:

$$\mathcal{C}_F^G \equiv \{\alpha \in G - F : \|F + \alpha\| = \|F\|\}.$$

Então temos os seguintes resultados.

Lema 7.4 Se G é um grafo planar e F uma floresta de G com pelo menos duas arestas, então

$$|\mathcal{C}_F^G| \leq 2|A_F| - 3\|F\|.$$

Prova Note que se F tem alguma componente F' com menos de duas arestas, temos que $|\mathcal{C}_{F'}^{G'}| = 0$. Portanto, para provar o enunciado, basta considerar o caso em que cada componente de F tem pelo menos duas arestas.

A prova será feita por indução sobre $\|F\|$.

$$\|F\| = 1$$

Nesse caso

$$\begin{aligned} |C_F^G| &= |C_F^{G[F]}| = |A_{G[F]}| - |A_F| \\ &= \leq 3|V_{G[F]}| - 6 - |A_F| = 3(|A_F| + 1) - 6 - |A_F| \\ &= 2|A_F| - 3\|F\|. \end{aligned}$$

$$\|F\| > 1$$

Seja T um componente de F . Então

$$\begin{aligned} |C_F^G| &= |C_{F-T}^G| + |C_T^G| \\ &\leq 2|A_{F-T}| - 3\|F - T\| + 2|A_T| - 3 \\ &= 2(|A_F| - |A_T|) - 3(\|F\| - 1) + 2|A_T| - 3 \\ &= 2|A_F| - 3\|F\|. \end{aligned}$$

□

Lema 7.5 *Se G é um grafo ω -ponderado planar e F é uma floresta ótima de G , então*

$$\omega(F) > \frac{1}{3}\omega(G).$$

Prova Equivalentemente provaremos que

$$\omega(G - F) < 2\omega(F).$$

Suponha $A_G = \{\alpha_1, \alpha_2, \dots, \alpha_{|A_G|}\}$ onde, $\omega(\alpha_1) \geq \omega(\alpha_2), \dots, \omega(\alpha_{|A_G|})$ e seja F uma floresta ótima de G construída como no Lema 7.2 (Algoritmo de Kruskal).

Sejam então

$$\begin{aligned} A_F &= \{\phi_1, \phi_2, \dots, \phi_{|F|}\}, \\ A_{G-F} &= \{\gamma_1, \gamma_2, \dots, \gamma_{|A_G-F|}\}, \end{aligned}$$

tais que para cada $1 \leq i < j \leq |A_G|$, se $\alpha_i = \phi_{i'}$ e $\alpha_j = \phi_{j'}$ (respectivamente $\alpha_i = \gamma_{i'}$ e $\alpha_j = \gamma_{j'}$) então $i' < j'$. Vamos mostrar, por indução sobre k que

$$\sum_{i=1}^{2k} \omega(\gamma_i) < 2 \sum_{i=1}^k \omega(\phi_i).$$

O caso $k = 1$ é evidente. Suponha então $k > 1$.

Seja p tal que $\alpha_p = \phi_k$ e considere $S = \{\alpha_1, \alpha_2, \dots, \alpha_{p-1}\}$.

Observe que

$$\begin{aligned} S \cap F &= \{\phi_1, \phi_2, \dots, \phi_{k-1}\} \text{ e} \\ S - F &\subseteq \mathcal{C}_F^G. \end{aligned}$$

Então, pelo Lema 7.4 segue que

$$|S - F| \leq 2|S \cap F| - 3 < 2|S \cap F| = 2(k - 1),$$

e portanto

$$S - F \subseteq \{\gamma_1, \dots, \gamma_{2(k-1)}\}.$$

Logo, $\gamma_{2k-1} \notin \{\alpha_1, \alpha_2, \dots, \alpha_p = \phi_k\}$ e assim,

$$\omega(\gamma_{2k-1}) \leq \omega(\phi_k),$$

donde segue que

$$\omega(\gamma_{2k}) + \omega(\gamma_{2k-1}) \leq 2\omega(\phi_k).$$

Somando esta última desigualdade com a desigualdade

$$\sum_{i=1}^{2(k-1)} \omega(\gamma_i) \leq 2 \sum_{i=1}^{k-1} \omega(\phi_i),$$

válida pela hipótese de indução, obtemos a desigualdade desejada. $\square$

Teorema 7.6 *Seja G um grafo ω -ponderado, H^* um subgrafo planar ótimo de G e H um subgrafo guloso de G . Então*

$$\omega(H) > \frac{1}{3}\omega(H^*).$$

Prova Sejam F_G , F^* e F florestas ótimas de G , H^* e H respectivamente.

Pelo Lema 7.2 temos que

$$\omega(H) \geq \omega(F) = \omega(F_G).$$

Por outro lado, pelo Lema 7.5 temos que

$$\omega(H^*) < 3\omega(F^*) \leq 3\omega(F_G).$$

Logo

$$\frac{\omega(H)}{\omega(H^*)} > \frac{\omega(F_G)}{3\omega(F_G)} = \frac{1}{3}.$$

□

O Teorema 7.6 nos garante que o método guloso fornece uma solução cujo peso nunca fica aquém de $\frac{1}{3}$ da solução ótima. Este é um resultado bastante surpreendente, pois em se tratando de uma heurística tão simples, poder-se-ia pensar que seria possível encontrar instâncias onde o desempenho de tal heurística pudesse ser arbitrariamente ruim.

Por outro lado, a garantia de qualidade de solução dado pelo Teorema 7.6 pode parecer um tanto decepcionante em vista dos resultados experimentais apresentados no Capítulo 6. Surge então a pergunta natural sobre a possibilidade de se melhorar tal garantia. O resultado a seguir mostra que tal limite é o melhor possível.

Teorema 7.7 Para todo $\varepsilon > 0$, arbitrariamente pequeno, existe um grafo ω -ponderado G tal que

$$\omega(H) < \left(\frac{1}{3} + \varepsilon\right) \omega(H^*),$$

onde H^* é um subgrafo planar ótimo de G e H um subgrafo guloso de G .

Prova Considere a família de grafos G_k ilustrada na Figura 7.1, para valores de k inteiros e maiores ou iguais a 4, definida por:

$$\begin{aligned} V_{G_k} &= \{1, 2, \dots, 2k\}, \\ A_{G_k} &= C_k + D_k + E_k + F_k, \end{aligned}$$

onde

$$\begin{aligned} C_k &= \{\{i, i+1 \bmod 2k+1\}, 1 \leq i \leq 2k\}, \\ D_k &= \{\{i, k-1\}, k+2 \leq i \leq 2k-1\}, \\ D'_k &= \{\{i, k+2\}, 3 \leq i \leq k-1\}, \\ E_k &= \{\{i, 2k-1\}, 2 \leq i \leq k-2\}, \\ E'_k &= \{\{i, 2\}, k+2 \leq i \leq 2k-2\}, \\ F_k &= \{\{1, k+1\}, \{k, 2k\}\}; \end{aligned}$$

e a ponderação ω de G_k é dada por

$$\omega(\alpha) \equiv \begin{cases} k+1, & \text{se } \alpha \in C_k + F_k; \\ k, & \text{caso contrário.} \end{cases}$$

Da construção de G_k temos imediatamente:

$$H^* = G[C_k + D_k + D'_k + E_k + E'_k]$$

e

$$H = G[C_k + F_k].$$

Logo,

$$\begin{aligned} \omega(H^*) &= \omega(C_k + D_k + E_k + D'_k + E'_k) \\ &= (k+1)|C_k| + k|D_k| + k|E_k| + k|D'_k| + k|E'_k| \\ &= (k+1)(2k) + k((k-1) + (k-3) + (k-3) + (k-4)) \\ &= 2k^2 + 2k + 4k^2 - 11k \\ &= 6k^2 - 9k, \end{aligned}$$

e

$$\omega(H) = \omega(C_k + F_k) = (k+1)(|C_k| + |F_k|) = (k+1)(2k+2) = 2(k+1)^2,$$

e daí

$$\begin{aligned}
\frac{\omega(H)}{\omega(H^*)} - \frac{1}{3} &= \frac{2(k+1)^2}{3k(2k-3)} - \frac{1}{3} \\
&= \frac{1}{3k} \left(\frac{2(k+1)^2}{2k-3} - k \right) \\
&= \frac{1}{3k} \left(\frac{7k+2}{2k-3} \right) \\
&\leq \frac{16}{3k}.
\end{aligned}$$

Então, para $k > \frac{16}{3\varepsilon}$ temos

$$\frac{\omega(H)}{\omega(H^*)} - \frac{1}{3} < \frac{16}{3k} < \frac{16}{3(\frac{16}{3\varepsilon})} = \varepsilon.$$

e portanto

$$\omega(H) < \left(\frac{1}{3} + \varepsilon \right) \omega(H^*).$$

□

As conclusões acima podem ser resumidas no seguinte enunciado:

Teorema 7.8 *O limite justo de desempenho absoluto do método guloso quando aplicado ao PSPO é $\frac{1}{3}$.*

Noutras palavras, a afirmação

$$\omega(H) \geq \alpha \omega(H^*),$$

onde H e H^* são como no Teorema 7.7 é verdadeira para $\alpha = \frac{1}{3}$ e falsa para $\alpha = \frac{1}{3} + \varepsilon$ para todo $\varepsilon > 0$.

Figura 7.1: O Grafo G_k

7.2 Uma Alternativa para a Implementação

Nesta seção, apresentamos uma alternativa para a implementação de GUL que visa reduzir significativamente seu tempo de execução. Nossa proposta é baseada na apresentação do algoritmo de teste de planaridade feita na primeira parte do trabalho, e fundamentalmente consiste em manter atualizada uma estrutura de dados que descreve uma hierarquia sobre o subgrafo planar em construção, a fim de evitar repetidas BPs e reordenações de arestas.

Uma implementação direta de GUL poderia ser esquematizada da seguinte forma:

Algoritmo 7.1

Guloso(A)

Entrada: Uma lista das arestas de um grafo ponderado G em ordem não crescente de peso.

Saída: Um subgrafo planar guloso de G .

$H \leftarrow \emptyset$;

Enquanto $A \neq \emptyset$

$\alpha \leftarrow$ primeiro elemento (retirado) de A ;

```

    Ainda_Planar( $H, \alpha$ );
    Se  $|A_H| = 3|V_H| - 6$ 
        Devolva  $H$ .
    Devolva  $H$ .

```

Algoritmo 7.2

```
Ainda_Planar( $G, \alpha$ )
```

Entrada: Um grafo planar G e uma aresta $\alpha \notin A_G$.

Saída: Acrescenta α a G se $G + \alpha$ é planar; caso contrário não faz nada.

A fim de que Guloso tenha bom desempenho o algoritmo `Ainda_Planar` deve ser o mais eficiente possível, e portanto, baseado em algum algoritmo de desempenho linear, tal como **HT**.

Entretanto, imagine o leitor que, no algoritmo acima, `Ainda_Planar` seja baseado numa implementação direta de **HT** tal como apresentado em [HT74]. Teríamos:

Algoritmo 7.3

```
Ainda_Planar( $G, \alpha$ )
```

Entrada e Saída: Como no Algoritmo 7.2.

```

 $G \leftarrow G + \alpha$ ;
Para cada bloco não trivial  $B \subseteq G$ 
    Se HT( $B$ ) = não
         $G \leftarrow G - \alpha$ ;
    Retorne.

```

Salta aos olhos a inconveniência dessa alternativa:

Em primeiro lugar, devemos localizar e isolar todos os blocos não triviais de $G + \alpha$ a cada passo. Algoritmos eficientes para tal são bem conhecidos [AHU74, Ter91, Szw84], sendo o

mais popular deles baseado em uma **BP** exaustiva sobre o grafo. Entretanto, a utilização desse tipo de algoritmo exige que o grafo $G + \alpha$ seja representado em forma de listas de adjacência (tal como necessário para a execução de **HT**), o que nos obriga a criar uma réplica de cada bloco não trivial de $G + \alpha$ para submissão a **HT**, com evidente desperdício de memória.

Além disso, a cada passo temos o importante fato de estar testando a planaridade de um grafo $G + \alpha$ onde G é sabidamente planar, e não fazemos qualquer uso dessa informação.

Finalmente, recordando que cada execução de **HT** sobre um bloco não trivial B de $G + \alpha$ envolve a execução de duas **BP** sobre B e uma ordenação completa de todas as arestas em B , vemos que implementação tal como esboçada acima, os resultados de todo esse processamento são perdidos e refeitos a cada ciclo, causando uma grande redundância nos cálculos.

Nosso objetivo nessa seção é apresentar uma maneira de contornar estes inconvenientes, apresentando uma maneira de reaproveitar os cálculos feitos a cada ciclo da execução, diminuindo o impacto dos repetidos testes de planaridade.

Essa estrutura de dados vem a ser nada mais que a representação explícita das hierarquias tal como exposto na primeira parte do trabalho.

Deve ser notado que, muito embora a complexidade do algoritmo permaneça $O(n^3)$ numa análise pessimista, o ganho em tempo de execução e requisitos de memória ainda assim é significativo, como indicam os resultados experimentais apresentados na Seção 7.3.

O seguinte teorema estabelece as condições necessárias e suficientes para que um grafo permaneça planar ao se lhe acrescentar uma aresta.

Teorema 7.9 *Seja G um grafo planar e $u, v \in V_G$. Sejam $s_1, s_2, \dots, s_n$ os vértices em G que separam u de v , tomados na ordem em que aparecem em qualquer caminho de u a v em G . Nessas circunstâncias, $G + uv$ é planar se e somente se $G + us_1 \in G + s_nv \in G + s_is_{i+1}$, $1 \leq i \leq n$, são todos eles planares.*

Prova Inicialmente façamos $s_0 = u$ e $s_{n+1} = v$. Dessa forma, podemos reformular o enunciado assim:

$G + s_0s_{n+1}$ não é planar se e somente se $G + s_is_{i+1}$ não é planar para algum $0 \leq i \leq n$.

Primeiramente suponhamos que $G + ab$ não é planar, onde

$$\begin{aligned} a &= s_p, \\ b &= s_{(p+1) \bmod (n+2)} \end{aligned}$$

para algum $0 \leq p \leq n+1$.

Então temos algum $H \subseteq G + ab$ homeomorfo a $K \in \{K_{3,3}, K_5\}$.

Sejam $v_1, v_2, \dots, v_{|V_K|}$ os vértices de H correspondentes aos vértices de K por tal homeomorfismo e seja P_{ij} , $1 \leq i, j \leq |V_K|$, o caminho de v_i a v_j em H identificado com uma única aresta de K pelo homeomorfismo.

Se v_1v_2 é uma aresta de K , então $K - v_1v_2$ é um bloco. Logo, $H - P_{12}$ também o é e nenhum dos separadores s_i pode estar em $H - P_{12}$. Note que como G é planar e $G + ab$ não é planar então H contém a aresta ab e portanto P_{12} deve conter algum vértice separador.

Seja s_l o primeiro deles em P_{12} (Se s_{l+1} ocorre antes de s_l considere P_{21} em vez de P_{12}). Então, s_{l+1} tem de ser o último deles em P_{12} .

Com isso (e tendo a Figura 7.2 em mente), o resultado segue facilmente. De fato, no caso da implicação ($\Rightarrow$) temos $a = u$, $b = v$. Neste caso, claramente $H + s_ls_{l+1}$ não é planar, e portanto $G + s_ls_{l+1}$ também não é.

Para a implicação ($\Leftarrow$) considere $a = s_i$, $b = s_{i+1}$, $0 \leq i \leq n$. Claramente $H + uv$ não é planar e daí $G + uv$ também não é. $\square$

Assim, poderíamos esquematizar o Algoritmo `Ainda_Planar` da seguinte forma:

Algoritmo 7.4

`Ainda_Planar`(G, α)

Entrada e Saída: Idênticas às do Algoritmo 7.2.

Se u e v estão na mesma componente de G

Se u e v são incomparáveis

Reconstrua a hierarquia sobre essa componente de G de forma a fazê-los comparáveis

Para cada segmento maximal Q de P_{uv} inteiramente contido num bloco de G

Se `Continua_Planar`(G, Q) = não

Retorne.

$G \leftarrow G + \alpha$.

Algoritmo 7.5

`Continua_Planar`(G, P)

Entrada: Um grafo G e um caminho P sobre G com ambas as pontas comparáveis e num mesmo bloco de G .

Saída: sim ou não conforme $G + uv$ seja ou não planar, onde u e v são as pontas de P .

A vantagem em estabelecer **GUL** dessa forma é que evitamos alterações estruturais no grafo sendo construído enquanto não temos certeza de que a alteração é realmente necessária.

Assim, se mantemos informação atualizada a respeito dos blocos de G a cada passo, evitamos que a tentativa de inclusão de uma nova aresta destrua essa informação (ao ligar dois blocos diferentes) e depois nos obrigue a recalculá-la, no caso de essa nova aresta não ser aceita.

Embora a um exame superficial, pareça estarmos trocando um único teste de planaridade por vários (no caso geral), e portanto perdendo em desempenho, não é esse o caso: na verdade estamos evitando o recálculo dos blocos de G a cada passo (indispensável para o teste de planaridade), só o fazendo na medida estritamente necessária. Mais ainda, deve-se lembrar que no caso

geral tentaremos incorporar $O(|V_G|^2)$ arestas, mas só $O(|V_G|)$ serão efetivamente incorporadas, o que significa que teremos $O(|V_G|^2)$ tentativas frustradas. Esse é o caso que consome a parte mais substancial do tempo de execução da heurística.

Voltemos então nossa atenção para a implementação do algoritmo de teste de planaridade da primeira parte.

Como vimos ali, representando explicitamente as hierarquias (que representam uma BP), temos informação suficiente para manter controle dos blocos de G a cada passo, bem como para fazer um teste de planaridade tal como o necessário no Algoritmo 7.5, evitando as dispendiosas BPs da proposta original de HT, bem como a ordenação total de suas arestas a cada nova tentativa. Mais ainda, no caso de fazermos um teste de planaridade de $G + \alpha$ e a resposta ser negativa, reestabelecer a hierarquia original sobre G é uma tarefa simples, se garantimos que α é uma aresta fora da árvore de BP, ou seja, se garantimos que as pontas de α são comparáveis pela hierarquia.

Assim, poderíamos esquematizar `Continua_Planar` por:

Algoritmo 7.6

`Continua_Planar( $G, P$ )`

Entrada e Saída: Como no Algoritmo 7.5.

```

 $G \leftarrow G + uv$ 
 $resposta \leftarrow \text{Planar}'(\rho(uv));$ 
 $G \leftarrow G - uv;$ 
Devolva  $resposta$ .
```

O leitor que tenha acompanhado a detalhada descrição da Seção 4.2 onde sugerimos uma maneira adequada de implementação de uma estrutura de dados que representa uma hierarquia sobre um grafo de forma a permitir testes de planaridade em diferentes blocos do grafo de forma independente, bem como manter controle sobre esses blocos não terá dificuldade em perceber como cada passo dos algoritmos descritos acima pode ser implementado.

7.3 Resultados Experimentais

Nesta seção apresentamos os resultados experimentais obtidos com relação ao desempenho de `GUL`, de acordo com a implementação² de um algoritmo baseado nas idéias expostas na Seção

²Essa implementação foi programada em C++ compilado com BorlandC 2.0 num computador padrão IBM-PC baseado em processador Intel 80486 de 33MHz, sob MS-DOS versão 5.0.

4.2.

Duas séries de testes são apresentadas a seguir: uma sobre *instâncias aleatórias* e outra sobre *instâncias pessimistas*.

Cada **instância aleatória** de tamanho n consistiu de um grafo completo de n vértices dado por uma lista de suas arestas em ordem aleatória³.

Cada **instância pessimista** de tamanho n consistiu de uma instância de tamanho n onde as $3n - 5$ primeiras e a penúltima arestas são aceitas, ou seja, uma instância com o maior número possível de tentativas frustradas de acréscimo de arestas, onde cada acréscimo é feito no maior grafo planar não maximal possível⁴.

A Figura 7.3 mostra um gráfico dos tempos médios de execução t_n para instâncias aleatórias de tamanho n onde $3 \leq n \leq 200$. Cada tempo t_n indica a média de tempos sobre uma série de instâncias aleatórias de tamanho n . O número de execuções para instâncias de um mesmo tamanho também é aleatório. Para melhor visualização, os eixos do gráfico são apresentados em escalas diferentes.

A Figura 7.4 mostra um gráfico dos tempos de execução T_n para instâncias pessimistas⁵ de tamanho n onde $3 \leq n \leq 67$. Nesse gráfico, ambos os eixos são mostrados na mesma escala.

Para efeito de comparação, apresentamos lado a lado na Tabela 7.1, os dados disponíveis referentes aos tempos de execução de cada uma das soluções aproximadas discutidas no Capítulo 6. Nessa tabela, as duas colunas referentes a GUL mostram (à esquerda e à direita respectivamente) os tempos t_n e T_n apresentados nos gráficos das Figuras 7.3 e 7.4.

Tabela 7.1: Tempos de Execução Comparados (em segundos de CPU)

$ V_G $	$ A_G $	GUL		GDY	IDH	DH	WE	DS
10	45	0.06	0.06	2.0	0.17	0.15	0.27	—
20	190	0.66	0.99	20.2	1.82	0.38	3.67	0.90
30	435	2.53	3.68	71.3	7.37	0.72	13.41	7.60
40	780	6.65	9.23	165.8	19.29	1.13	39.49	37.05

Muito embora a comparação entre implementações feitas em ambientes operacionais diferentes seja sempre uma questão delicada, esses dados nos parecem ainda assim significativos pela

³Não foi incluída qualquer informação a respeito dos pesos das arestas, uma vez que o objetivo destes testes é simplesmente a aferição de tempos de execução da implementação: supõe-se que a lista de arestas em ordem aleatória representa a lista de arestas do grafo em ordem não crescente de peso numa “instância completa” do PSPO.

⁴Como esperado, verificamos experimentalmente que essa classe de instâncias é a que demanda maior tempo de execução.

⁵O processamento de uma instância pessimista demanda um consumo de memória maior do que instâncias “usuais”, razão por que não pudemos efetuar testes sobre instâncias pessimistas de maiores tamanhos dentro das restrições de memória impostas pelo ambiente computacional utilizado.

Figura 7.3: Tempos de Execução sobre Instâncias Aleatórias

Figura 7.4: Tempos de Execução sobre Instâncias Pessimistas

sua desproporção.

Para uma estimativa mais global do comportamento assintótico dessas implementações, vamos definir, a partir desses dados, os polinômios

P_{GUL} : O polinômio de grau 3 em n que interpola os valores de t_n (tempos médios de instâncias aleatórias) referentes a **GUL** apresentados na Tabela 7.1.

$P_{\text{GUL}\bullet}$: Idem, para os valores de T_n (tempos de instâncias pessimistas).

P_{GDY} : Idem, referente aos valores de **GDY** na Tabela 7.1.

P_{IDH} : Idem, para os valores referentes a **IDH**.

P_{WE} : Idem, para os valores referentes a **WE**.

e

$\widehat{P}_{\text{GUL}}$: O polinômio da forma kn^3 , onde k é o menor valor (com duas casas de precisão em notação de ponto flutuante) tal que $t_n \leq kn^3$ para todos os valores de t_n do gráfico na Figura 7.3.

$\widehat{P}_{\text{GUL}\bullet}$: Idem, para os valores do gráfico na Figura 7.4.

$\widehat{P}_{\text{GDY}}$: Idem, referente aos valores de **GDY** na Tabela 7.1.

$\widehat{P}_{\text{IDH}}$: Idem, para os valores referentes a **IDH**.

$\widehat{P}_{\text{WE}}$: Idem, para os valores referentes a **WE**.

$\widehat{P}_{\text{DS}}$: O polinômio da forma kn^4 , onde k é o menor valor (com duas casas de precisão em notação de ponto flutuante) tal que $t_n \leq kn^4$ para todos os valores referentes a **DS** mostrados na Tabela 7.1.

A Tabela 7.2 mostra dos polinômios P_{H} e $\widehat{P}_{\text{H}}$ para cada uma das heurísticas **H** arroladas acima.

Tabela 7.2: Estimativas dos Polinômios

H	P_H	$\widehat{P}_H$
GUL	$(1,63 \times 10^{-4})n^3 - (3,45 \times 10^{-3})n^2 + (4,92 \times 10^{-2})n - (2,50 \times 10^{-1})$	$(1,04 \times 10^{-4})n^3$
GUL*	$(1,83 \times 10^{-4})n^3 - (2,20 \times 10^{-3})n^2 + (3,11 \times 10^{-2})n - (2,20 \times 10^{-1})$	$(1,45 \times 10^{-4})n^3$
GDY	$(2,35 \times 10^{-3})n^3 + (1,75 \times 10^{-2})n^2 + (3,50 \times 10^{-1})n + (1,40 \times 10^0)$	$(2,56 \times 10^{-3})n^3$
IDH	$(4,12 \times 10^{-4})n^3 - (5,20 \times 10^{-3})n^2 + (3,28 \times 10^{-2})n - (5,00 \times 10^{-2})$	$(3,02 \times 10^{-4})n^3$
WE	$(1,67 \times 10^{-3})n^3 - (6,83 \times 10^{-2})n^2 + (1,22 \times 10^0)n - (6,79 \times 10^0)$	$(6,18 \times 10^{-4})n^3$
DS	—	$(1,45 \times 10^{-4})n^4$

Conclusão

Diversos pontos ao longo do desenvolvimento deste trabalho seriam dignos de nota como marcantes na nossa formação:

A experiência de lidar com um problema de otimização $\mathcal{NP}$ -Completo, atacando-o tanto pelo flanco teórico como pelo prático foi uma das mais notáveis. De fato, vista com perspectiva, a vivência da interação entre teoria e prática; a maneira como resultados teóricos influenciaram as opções de implementação, e, por outro lado, como a implementação de algoritmos conhecidos e a análise de seus resultados influenciaram as indagações e soluções formais é talvez a marca mais profunda deixada por todo o processo, do ponto de vista técnico.

Embora o texto não deixe transparecer, o material aqui apresentado é o resultado de uma dinâmica difícil de descrever com detalhe: tanto a exposição do algoritmo da Primeira Parte quanto os resultados da implementação descrita na Segunda Parte passaram por uma série de versões intermediárias; cada uma destas, uma vez consumada, nos levou a novos enfoques e conseqüentes refinamentos que acabavam por se mostrar significativamente diferentes da anterior.

Na realidade, nossa impressão é de que este processo ainda não alcançou a exaustão: pretendemos dar prosseguimento a este trabalho num futuro breve, explorando algumas linhas que fomos obrigados a deixar de lado a fim de não nos desviarmos do nosso objetivo principal. Dentre estes poderíamos apontar a idéia de aproveitar o algoritmo de teste de planaridade da Primeira Parte para obter uma representação explícita de uma imersão do grafo, a análise das possibilidades de mesclar a estratégia gulosa com outras heurísticas a fim de obter melhores soluções aproximadas e o refinamento da representação interna do grafo em construção a fim de poupar mais tempo de execução na busca do “subgrafo guloso”. Ficamos bastante estimulados pela recente publicação de resultados correlatos como [Tam88], [CHT93] e pelos comentários em [DET93], que indicam a atualidade do problema bem como a relevância dos resultados atingidos.

Dentre as surpresas agradáveis, talvez a mais notável tenha ocorrido quando fomos abordados por um dos autores de [JM93] interessado na utilização da nossa implementação em seu projeto, o que por sua vez nos levou à também interessante experiência de redigir e publicar de forma preliminar parte dos resultados deste trabalho [Car93]⁶.

Finalmente, não posso deixar de mencionar o quão marcante quanto foi, do ponto de vista

⁶É nossa intenção publicar uma versão revisada deste trabalho num futuro próximo.

pessoal, o relacionamento com a Orientadora, transcorrido sempre da maneira mais amigável possível (apesar de todos os meus “esforços” em contrário). Assim como na seção de agradecimentos, poupo ao leitor maiores comentários além de fazer constância do meu reconhecimento pela sua atenção que generosamente passou muito além da simples “orientação técnica” do trabalho em si^{7 8}.

⁷Conste entretanto que, ainda que tivesse se resumido somente a isso, já haveria aqui motivo suficiente para uma menção mais que passageira.

⁸Num momento de rara lucidez, o leitor fica também poupado da transcrição do “Mar Portuguez” [Pes86] nesse encerramento ...

Bibliografia

- [AHU74] A. E. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1974.
- [AP61] L. Auslander and S. V. Parter. On embedding graphs in the plane. *J. Math. and Mech.*, 10(3):517–513, May 1961.
- [Ber73] C. Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, Amsterdam, 1973.
- [BL76] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq -tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
- [BM76] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.
- [Bol79] B. Bollobás. *Graph Theory, an Introductory Course*. Graduate Texts in Mathematics. Springer Verlag, New York, 1979.
- [Bre86] J. R. Breitenbach. A criterion for the planarity of a graph. *Journal of Graph Theory*, 10:529–532, 1986.
- [Bur78] M. Burstein. Kuratowski–Pontrjagin theorem on planar graphs. *Journal of Combinatorial Theory*, 24:228–231, 1978.
- [Car93] R. Carmo. Incremental planarity testing. Technical Report RT—DI/ET001, Universidade Federal do Paraná, 1993. Preliminary Version.
- [CHT93] Jiazhen Cai, Xiaofeng Han, and R. E. Tarjan. An $O(m \log n)$ -time algorithm for the maximal planar subgraph problem. *SIAM J. Comput.*, 23(6):1142–1162, December 1993.
- [CN88] N. Chiba and T. Nishizeki. *Planar Graphs: Theory and Algorithms*, volume 32 of *Annals of Discrete Mathematics*. North-Holland Publishing Company, Amsterdam, 1988.

- [CNAO85] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using pq -trees. *Journal of Computer and System Sciences*, 30:54–76, 1985.
- [CW90] E. R. Canfield and S. G. Williamson. The two basic linear time planarity algorithms: Are they the same? *Linear and Multilinear Algebra*, 26:243–265, 1990.
- [Deo76] N. Deo. Note on Hopcroft and Tarjan's planarity algorithm. *Journal of the Association for Computing Machinery*, 23(1):74–75, January 1976.
- [DET93] G. Di Battista, P. Eades, and R. Tamassia. Algorithms for drawing graphs: an annotated bibliography. Available via anonymous ftp from wilma.cs.brown.edu (128.148.33.66), files /pub/gdbiblio.tex.Z and /pub/gdbiblio.ps.Z, March 1993.
- [dFR82] H. de Fraysseix and P. Rosenstiehl. Embedding graphs in the plane — algorithmic aspects. In *Annals of Discrete Mathematics*, pages 75–80. North-Holland Publishing Company, 1982.
- [Dic84] W. E. Dick. Discrete characterizations of planarity part I: The classical viewpoint. *Linear and Multilinear Algebra*, 16:39–65, 1984.
- [DMP65] G. Demoucron, Y. Malgrange, and R. Pertisuet. Graphes planaires: Reconnaissance et construction des représentations planaires topologiques. *Revue Française Recherche Operationelle*, 8:34–47, 1965.
- [DS54] G. A. Dirac and S. Schuster. A theorem of Kuratowski. *Indag. Math.*, 16:343–348, 1954.
- [EFG82] P. Eades, L. R. Foulds, and J. W. Giffin. An efficient heuristic for identifying a maximum weight planar graph. In *Lecture Notes in Mathematics*. Springer Verlag, Berlin, 1982.
- [Eve79] S. Even. *Graph Algorithms*. Computer Software Engineering Series. Computer Science Press Inc., Potomac, Maryland, 1979.
- [Far48] I. Fáry. On straight line representations of planar graphs. *Acta Sci. Math. Szeged*, 11:229–233, 1948.
- [FGG85] L. R. Foulds, P. B. Gibbons, and J. W. Giffin. Facilities layout adjacency determination: An experimental comparison of three graph-theoretic heuristics. *Operations Research*, 33(5):1091–1106, September/October 1985.
- [Fou74] J. C. Fournier. Une relation de separation entre cocircuits d'un matroïd. *Journal of Combinatorial Theory*, 16:181–190, 1974.
- [Fou84] L. R. Foulds. *Combinatorial Optimization for Undergraduates*. Undergraduate Texts in Mathematics. Springer Verlag, 1984.

- [FR76] L. R. Foulds and D. F. Robinson. A strategy for solving the plant layout problem. *Operational Research Quarterly*, 27(4 i):845–855, 1976.
- [FR78] L. R. Foulds and D. F. Robinson. Graph theoretic heuristics for the plant layout problem. *Int. J. Prod. Res.*, 16:27–37, 1978.
- [FR79] L. R. Foulds and D. F. Robinson. Construction properties of combinatorial deltahedra. In *Discrete Applied Mathematics*, volume 1, pages 75–87. North-Holland Publishing Company, 1979.
- [FW66] G. J. Fisher and D. Wing. Computer recognition and extraction of planar graphs from the incidence matrix. *IEEE Transactions on Circuit Theory*, 13(2):154–163, June 1966.
- [GF84] J. W. Giffin and L. R. Foulds. Efficient graph planarity updating tests. *Ars Combinatoria*, 17 A:185–202, 1984.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.
- [GM79] M. Gondrand and M. Minoux. *Graphes et Algorithmes*. Editions Eyrolles, Paris, 1979.
- [GM84] M. Gondran and M. Minoux. *Graphs and Algorithms*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., New York, 1984. Translation of [GM79].
- [Gol63] A. J. Goldstein. An efficient and constructive algorithm for testing whether a graph can be embedded in a plane. In *Graph and Combinatorics Conference*, pages 16–18. Office of Naval Research Logistics Proj., Dep. of Mathematics, Princeton University, May 1963.
- [Gou88] R. Gould. *Graph Theory*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1988.
- [GT87] J. L. Gross and T. W. Tucker. *Topological Graph Theory*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., New York, 1987.
- [HT74] J. E. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the Association for Computing Machinery*, 21(4):549–568, July 1974.
- [JM93] M. Jünger and P. Mutzel. Solving the maximum weight planar subgraph. In G. Rinaldi and L. Wolsey, editors, *Proceedings of the 3rd IPCO Conference*, 1993.
- [Kel81] A. K. Kel'mans. A new planarity criterion for 3-connected graphs. *Journal of Graph Theory*, 5:259–267, 1981.
- [Klo89] W. Klotz. A constructive proof of Kuratowski's theorem. *Ars Combinatoria*, 28:51–54, 1989.

- [Kna75] B. Knauer. A simple planarity criterion. *Journal of the Association for Computing Machinery*, 2:226–230, April 1975.
- [Kri75] M. S. Krishnamoorthy. An $\mathcal{NP}$ -hard problem in bipartite graphs. *SIGACT News*, 7:1–26, 1975.
- [Kur30] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicæ*, 15:271–283, July 1930.
- [LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs: International Symposium*, pages 215–232, New York, NY, 1967. Gordon and Breach.
- [Leu92] J. Leung. A new graph-theoretic heuristic for facility layout. *Management Science*, pages 594–605, 1992.
- [LG79] P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proceedings of the Tenth Southeastern Conference on Combinatorics, Graph Theory and Computing (Florida Atlantic University, Boca Raton, Fla)*, volume XXIV of *Congressus Numerantium*, pages 727–738. Utilitas Math., 1979.
- [Liu88] Y. Liu. A new approach to the linearity of testing planarity of graphs. *Acta Mathematicæ Applicatæ Sinica*, 4(3):257–265, August 1988.
- [L79] C. L. Lucchesi. *Introdução à Teoria dos Grafos*. 12^o Colóquio Brasileiro de Matemática, Poços de Caldas — MG, July 1979. IMPA, Rio de Janeiro.
- [Mac37] S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicæ*, 28:22–32, 1937.
- [Mun84] J. Munkres. *Elements of Algebraic Topology*. Addison-Wesley Publishing Company, New York, 1984.
- [Mut92] P. Mutzel. A fast linear time embedding algorithm based on Hopcroft-Tarjan planarity test. Technical Report 92.107, Angewandte Mathematik und Informatik, Universität zu Köln, 1992.
- [O’N73] P. V. O’Neil. A short proof of MacLane’s planarity theorem. *Proceedings of the American Mathematical Society*, 37(2), February 1973.
- [Pes86] F. Pessoa. Mensagem. In *Fernando Pessoa: Obra Poética*, page 16. Editora Nova Aguilar, Rio de Janeiro, 1986. Parte II, Canto X (“Mar Português”).
- [Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., New York, 1986.
- [SP82] K. Steiglitz and C. H. Papadimitriou. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall Inc., Englewood Cliffs, NJ, 1982.

- [Szw84] J. L. Szwarcfiter. *Grafos e Algoritmos Computacionais*. Editora Campus, Rio de Janeiro, 1984.
- [Tam88] R. Tamassia. A dynamic data structure for planar graph embedding. *ICALP*, pages 576–590, 1988. Extended Abstract.
- [Ter91] R. Terada. *Desenvolvimento de Algoritmo e Estruturas de Dados*. Makron Books — McGraw Hill, São Paulo, 1991.
- [Tho80] C. Thomassen. Planarity and duality of finite and infinite graphs. *Journal of Combinatorial Theory*, 29:244–271, 1980.
- [Tut58] W. T. Tutte. Matroids and graphs. *Transactions of the American Mathematical Society*, 88:144–174, 1958.
- [Tut60] W. T. Tutte. Convex representations of graphs. *Proceedings of the London Mathematical Society*, 10:304–320, 1960.
- [Tut63] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, 13:743–768, 1963.
- [Tut75] W. T. Tutte. Separation of vertices by a circuit. *Discrete Math.*, 12:173–184, 1975.
- [VDW85] Kiem-Phong Vo, W. E. Dick, and S. G. Williamson. Ranking and unranking planar embeddings. *Linear and Multilinear Algebra*, 18:35–65, 1985.
- [Vo83] Kiem-Phong Vo. Segment graphs, depth-first cycle basis, 3-connectivity and planarity of graphs. *Linear and Multilinear Algebra*, 13:119–141, 1983.
- [Whi33] H. Whitney. 2-isomorphic graphs. *Amer. J. Math.*, 55:245–254, 1933.
- [Wil80] S. G. Williamson. Embedding graphs in the plane — algorithmic aspects. In *Annals of Discrete Mathematics*, pages 349–384. North-Holland Publishing Company, 1980.
- [Wil85] S. G. Williamson. *Combinatorics for Computer Science*. Computers and Math Series. Computer Science Press Inc., Rockville, Maryland, 1985.