

DEZ ALGORITMOS PARA O PROBLEMA-SANDUÍCHE DO
CONJUNTO HOMOGÊNEO

Vinícius Gusmão Pereira de Sá

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

Aprovada por:

Prof^ª. Celina Miraglia Herrera de Figueiredo, D.Sc.

Prof. Claudson Ferreira Bornstein, Ph.D.

Prof. Manoel José Machado Soares Lemos, Ph.D.

Prof. Marcus Vinicius Soledade Poggi de Aragão, Ph.D.

Prof. Nelson Maculan Filho, D.Habil.

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 2006

SÁ, VINÍCIUS GUSMÃO PEREIRA DE

Dez algoritmos para o Problema-Sanduíche do Conjunto Homogêneo [Rio de Janeiro] 2006

XIV, 101 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2006)

Tese – Universidade Federal do Rio de Janeiro, COPPE

- 1 - Teoria de Grafos
- 2 - Problemas-Sanduíche
- 3 - Conjuntos Homogêneos

I. COPPE/UFRJ II. Título (série)

Para Lander e Marcos

Agradecimentos

É evidente que um trabalho como este não poderia ter existido sem o imenso apoio recebido de tantas pessoas. Vai aqui, então, apenas uma pequena demonstração por escrito da gratidão que lhes dedico.

Muito obrigado àqueles que participaram diretamente desta pesquisa: Celina Figueiredo, minha orientadora, que soube quase que magicamente obter o equilíbrio entre as necessárias cobranças e a necessária liberdade, que soube ajustar-me às regras formais da Academia sem matar meu estilo próprio, que soube respeitar meu tempo e confiar em minha por vezes temerária capacidade de geri-lo; Guilherme Fonseca, meu colega, amigo, colaborador, co-autor, uma das melhores cabeças que já conheci e com quem aprendi demais; Claudson Bornstein e Jeremy Spinrad, também co-autores e co-responsáveis pela diversidade das idéias aqui contidas.

Muito obrigado a esses dois *gentlemen* que, se não compartilham oficialmente da paternidade dos algoritmos aqui presentes, são contudo seus padrinhos prestimosos, pessoas extremamente capazes e, sobretudo, queridas, verdadeiros modelos para mim: Nelson Maculan e Carlos Martinhon (outrora, coincidentemente ou não, também orientador e orientado em uma pesquisa de doutoramento).

Ao amigo Hugo Vidal, meu agradecimento especial por *tudo*.

A Loana Nogueira, por todas as conversas e por toda a força durante a pesquisa, obrigado de verdade. (E por ter cuidado de mim quando eu passei mal no aeroporto de Salvador!)

Obrigado, Professor Manoel Lemos, pela solicitude com que atendeu ao convite para integrar a banca avaliadora. Obrigado, Professor Marcus Poggi, não apenas por solicitude idêntica, mas também por corroborar minha teoria de que é verdadeiramente possível fazer pesquisa durante uma corrida matinal.

Professoras Márcia Cerioli e Sulamita Klein; Professores Jayme Szwarcfiter, Fábio Protti e Valmir Barbosa; Solange (vizinha!), Cláudia, Sueli, Mercedes, Sônia, todos da COPPE Sistemas, personagens de muito destaque em toda esta estória, muito obrigado a vocês.

Obrigado a todos os meus familiares, em especial a três criaturas lindas, divertidas e muito amadas que, por mais próximas, acompanharam muito bem todas as dúvidas, respostas, aborrecimentos e alegrias que são parte integrante desta empreitada: minha noiva Germana e minhas irmãs Lídia e Luciana.

E, ainda que não lhes chegue esta expressão de agradecimento, devo dizer obrigado aos seguintes nomes, pela motivação diária para o aproveitamento de cada minuto como o maior dos tesouros: Johann Sebastian Bach, Glenn Gould, Lance Armstrong e Garry Kasparov.

Por fim, e não menos importante, vai meu agradecimento amoroso àqueles dois que, por tantos anos de educação e cuidados, são os grandes arquitetos de minha jornada: minha mãe Celia e meu pai Ilydio.

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

DEZ ALGORITMOS PARA O PROBLEMA-SANDUÍCHE DO
CONJUNTO HOMOGÊNEO

Vinícius Gusmão Pereira de Sá

Março/2006

Orientador: Celina Miraglia Herrera de Figueiredo

Programa: Engenharia de Sistemas e Computação

Problemas-sanduíche são generalizações de problemas de reconhecimento em grafos, onde ao grafo de entrada — no qual investiga-se a existência de determinada propriedade — é oferecido um conjunto de arestas *opcionais*. Esta classe de problemas foi originalmente formulada com vistas a aplicações práticas em Biologia Computacional, mas sua abrangência já há muito não se encontra limitada àquele campo de pesquisa.

Muitos dos problemas-sanduíche abordados na literatura até este momento mostraram-se NP-completos. Dentre aqueles polinomiais, os mais interessantes — do ponto de vista da criação e análise de algoritmos — são os que não se sabe ainda resolver com eficiência semelhante à que se consegue para os problemas de reconhecimento clássicos que lhes são correspondentes. Pertence a este grupo o *Problema-Sanduíche do Conjunto Homogêneo*, tema desta tese, e para o qual apresentamos dez algoritmos distintos (oito de nossa autoria).

Acreditamos que este estudo venha não apenas permitir um bom entendimento do problema e suas peculiaridades, mas também fornecer um panorama amplamente ilustrativo — e, de certo modo, didático — do processo mais geral da gênese e refinamento de técnicas, na busca incessante pelo algoritmo ótimo.

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

TEN ALGORITHMS FOR THE HOMOGENEOUS SET SANDWICH
PROBLEM

Vinícius Gusmão Pereira de Sá

March/2006

Advisor: Celina Miraglia Herrera de Figueiredo

Department: Systems Engineering and Computer Science

Sandwich problems are generalizations of graph recognition problems, where the input instance graph — which will be investigated as for the existence of a certain property — is provided with a set of *optional* edges. This class of problems originally arose due to some Computational Biology practical applications, but it has long encompassed other research fields.

Many sandwich problems considered in the literature have been found to be NP-complete. Among those which are polynomial, the most interesting — from the standpoint of algorithm development and analysis — are those which are not yet solvable with an efficiency close to that achieved for their counterpart classical recognition problems. To this latter group belongs the *Homogeneous Set Sandwich Problem*, the subject of this thesis, and for which we present ten distinct algorithms (eight of them authored by us).

We believe that the text allows not only a thorough understanding of the problem and its singularities, but also provides the reader with a broad —

and quite didactic — portrayal of the more general process of creation and refinement of techniques, in the constant search for the optimum algorithm.

Conteúdo

1	Introdução	1
1.1	Definições básicas	1
1.1.1	Grafos-sanduíche e problemas-sanduíche	1
1.1.2	Arestas obrigatórias, opcionais e proibidas	3
1.1.3	Conjuntos homogêneos	4
1.1.4	Conjuntos homogêneos sanduíche e o PSCH	5
1.2	Notação e convenções	7
1.3	Histórico do problema	8
1.3.1	Pesquisa pré-existente	8
1.3.2	Material inédito	9
1.4	Organização do trabalho	10
2	Incorporação de Testemunhas	14
2.1	Testemunhas	15
2.2	Montagem do grafo-sanduíche a partir de um conjunto sem testemunhas	16
2.3	O procedimento de incorporação de testemunhas	17
2.4	Primeiro algoritmo: Incorporações Exaustivas	18
3	O Grafo-Testemunha	21

3.1	Agrupando vértices interdependentes: o grafo-testemunha	21
3.2	Segundo algoritmo: Sumidouros Fortemente Conexos	24
3.3	Subgrafos fechados-por-pares	25
3.4	Terceiro algoritmo: Duas Fases	28
3.4.1	Refinamento da análise de complexidade	30
4	Incorporação Incompleta de Testemunhas	33
4.1	Quarto algoritmo: Subconjuntos Balanceados	35
4.1.1	Prova de corretude	37
4.1.2	Análise de complexidade	37
4.2	Quinto algoritmo: Monte Carlo	38
4.2.1	Prova de corretude	43
4.2.2	Análise de complexidade	44
4.3	Sexto algoritmo: Série Harmônica	46
4.3.1	Prova de corretude	48
4.3.2	Análise de complexidade	49
5	Inimigos	50
5.1	Vértices inimigos e grafos de inimizade	50
5.2	Sétimo algoritmo: Cliques Crescentes	54
5.2.1	Prova de corretude	56
5.2.2	Análise de complexidade	58
5.3	Oitavo algoritmo: Las Vegas	59
5.3.1	Prova de corretude	62
5.3.2	Análise de complexidade	63
5.4	Nono algoritmo: Preenchimento Acelerado	64
5.4.1	Prova de corretude	68
5.4.2	Análise de complexidade	68

6	Compleição de Pares	72
6.1	Décimo algoritmo: Compleição de Pares	73
6.1.1	Prova de corretude	76
6.1.2	Análise de Complexidade	77
7	Resultados Experimentais	82
8	Conclusão	85
8.1	Quadro de algoritmos e publicações	85
8.2	Últimas considerações	87
A	Implementação eficiente da incorporação de testemunhas	89
B	Construção eficiente do grafo-testemunha	92
C	Contraexemplo interessante para o segundo algoritmo	95

Lista de Figuras

1.1	Exemplo de instância de entrada para um problema-sanduíche	2
1.2	Um grafo-sanduíche para a instância da Figura 1.1	2
1.3	Partição de um grafo em conjunto homogêneo (H), seus vizinhos (N) e não-vizinhos (\overline{N})	4
1.4	Um conjunto homogêneo	5
2.1	Vértice-testemunha	16
2.2	O procedimento de incorporação de testemunhas	18
2.3	O algoritmo das Incorporações Exaustivas [2]	19
3.1	O grafo-testemunha	22
3.2	O algoritmo dos Sumidouros Fortemente Conexos [18]	23
3.3	O algoritmo das Duas Fases	29
4.1	O procedimento de incorporação incompleta de testemunhas	34
4.2	O algoritmo Subconjuntos Balanceados para o PSCH	36
4.3	Um algoritmo Monte Carlo sim-baseado para o PSCH	42
4.4	O algoritmo Série Harmônica	48
5.1	Exemplos de grafos de inimidade durante uma execução do algoritmo Série Harmônica	52
5.2	O algoritmo das Cliques Crescentes para o PSCH	56

5.3	Grafos de inimizadas durante execução do algoritmo CC	57
5.4	Incorporação de testemunhas restrita por inimizadas	61
5.5	Algoritmo de Las Vegas para o PSCH	62
5.6	O algoritmo do Preenchimento Acelerado	71
6.1	Rotina para determinação dos SFC's atingíveis por cada nó	73
6.2	A rotina Complete_os_Pares	75
6.3	O algoritmo da Compleição de Pares	78
6.4	Execução da rotina Localize_Sumidouros_Atingíveis	80
7.1	Resultados experimentais (Instâncias Aleatórias)	83
7.2	Resultados experimentais (Ciclos)	84
8.1	Tabela de algoritmos	86
A.1	Implementação eficiente da incorporação de testemunhas	91
B.1	Construção eficiente do grafo-testemunha	94
C.1	Contraexemplo à Proposição 3.1 [18] — grafos de entrada	96
C.2	Contraexemplo à Proposição 3.1 [18] — grafo-testemunha	97
C.3	Subgrafo K associado a CHS da instância da Figura C.1	98
C.4	SFC contido propriamente no subgrafo K	98

Capítulo 1

Introdução

Problemas-sanduíche foram primeiramente definidos no contexto da Biologia Computacional e têm atraído bastante atenção desde então, surgindo, em diversas aplicações — que já há muito não estão mais circunscritas àquela área de pesquisa — como generalizações naturais de problemas de reconhecimento [4, 10, 11, 14].

A decomposição em módulos, ou *decomposição modular*, é uma forma de decomposição de grafos que foi descoberta, independentemente, por pesquisadores de teoria de grafos, teoria de jogos, redes e outras áreas, inferindo-se daí sua importância. Conjuntos homogêneos são módulos não-triviais e encontram particular interesse no estudo dos grafos perfeitos, sobretudo por embasarem procedimentos de decomposição que preservam a perfeição [15].

1.1 Definições básicas

1.1.1 Grafos-sanduíche e problemas-sanduíche

Um grafo $G_2(V, E_2)$ é *supergrafo* de $G_1(V, E_1)$ se, e somente se, $E_2 \supseteq E_1$.

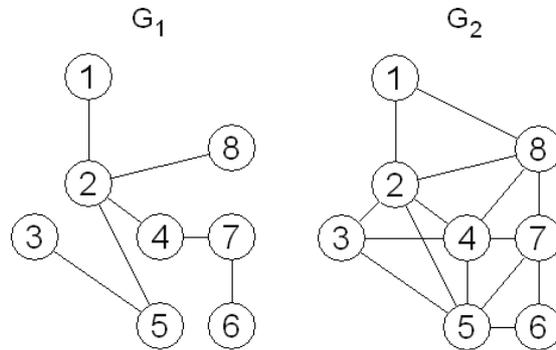


Figura 1.1: Exemplo de instância de entrada para um problema-sanduíche

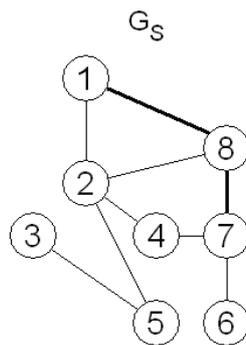


Figura 1.2: Um grafo-sanduíche para a instância da Figura 1.1

Um grafo $G_S(V, E_S)$ é dito *grafo-sanduíche* dos grafos $G_1(V, E_1)$ e $G_2(V, E_2)$ se, e somente se, $E_1 \subseteq E_S \subseteq E_2$. Isto é, um grafo-sanduíche do par (G_1, G_2) é tal que possui o mesmo conjunto V de vértices, e seu conjunto de arestas contém todo o conjunto E_1 de arestas do menor dos grafos, além de *qualquer número* de arestas que pertençam originalmente apenas a seu supergrafo, ou seja, a $E_2 \setminus E_1$.

No *problema-sanduiche* para a propriedade Π , deseja-se responder se há, para um par de grafos dado, algum grafo-sanduiche daquele par que possua a propriedade desejada Π [10].

A Figura 1.1 exemplifica um par grafo-supergrafo da entrada de um problema-sanduiche.

1.1.2 Arestas obrigatórias, opcionais e proibidas

A entrada clássica para problemas-sanduiche consiste, portanto, de dois grafos $G_1(V, E_1)$ e $G_2(V, E_2)$, ambos com o mesmo conjunto de vértices, um dos quais contendo todas as arestas do outro e, eventualmente, outras tantas mais, isto é, $E_2 \supseteq E_1$.

Em muitos casos, no entanto, é conveniente entendermos a entrada de um problema sanduiche como constituída de apenas *um* grafo completo $G(V, E)$ e uma partição de seu conjunto de arestas $E = V^2$ nos subconjuntos E_{ob} , E_{op} e E_{pr} de arestas *obrigatórias*, *opcionais* e *proibidas*, respectivamente. Justifica-se esta nomenclatura pelo fato de que todo grafo-sanduiche daquela instância deve possuir todas as arestas de E_{ob} , qualquer número de arestas de E_{op} e nenhuma aresta de E_{pr} .

A equivalência entre esses dois possíveis formatos de entrada é estabelecida por $E_{ob} = E_1$, $E_{op} = E_2 \setminus E_1$ e $E_{pr} = V^2 \setminus E_2$. Denotamos $m_{ob} = |E_{ob}|$, $m_{op} = |E_{op}|$ e $m_{pr} = |E_{pr}|$.

Um grafo-sanduiche para a instância da Figura 1.1 é apresentado na Figura 1.2, onde encontram-se destacadas as arestas opcionais daquela instância — originais, portanto, de $E_2 \setminus E_1$.

Dada a frequência com que aparecem no texto os valores $\min\{m_{ob}, m_{pr}\}$ e $\max\{m_{ob}, m_{pr}\}$, reservamos os símbolos m e M , respectivamente, para indicá-los.

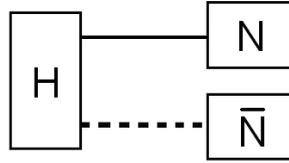


Figura 1.3: Partição de um grafo em conjunto homogêneo (H), seus vizinhos (N) e não-vizinhos (\bar{N})

1.1.3 Conjuntos homogêneos

Um *conjunto homogêneo* H para o grafo $G(V, E)$ é um subconjunto de V tal que $1 < |H| < |V|$ e todos os seus elementos têm os mesmos vizinhos fora de H , isto é, para todo $v \in V \setminus H$, vale $(v, h) \in E, \forall h \in H$ ou $(v, h) \notin E, \forall h \in H$. Conceito semelhante, portanto, ao de *módulo* de um grafo, apenas com a restrição adicional de tamanho, uma vez que conjuntos unitários de vértices, o conjunto vazio e o conjunto de todos os vértices do grafo, embora constituam módulos triviais, não são considerados conjuntos homogêneos.

Um conjunto homogêneo H particiona, pois, os demais vértices de um grafo (externos a H) em dois conjuntos, tal como indicado na Figura 1.3: o conjunto de vértices que são adjacentes a *todos* os elementos de H e o conjunto de vértices que não são adjacentes a *qualquer* vértice de H (conjuntos N e \bar{N} , respectivamente, na figura).

Na Figura 1.4, podemos ver o conjunto homogêneo $H = \{1, 8\}$ para o grafo que lá está representado. O vértice 2 é adjacente a todos os vértices de H , ao passo em que os demais vértices — 3, 4, 5, 6, 7 — não são adjacentes a qualquer vértice de H .

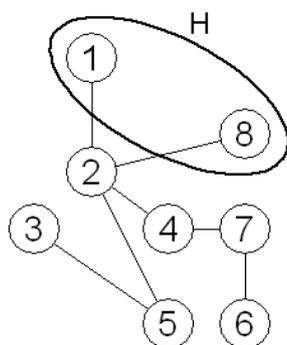


Figura 1.4: Um conjunto homogêneo

1.1.4 Conjuntos homogêneos sanduíche e o PSCH

Dados dois grafos $G_1(V, E_1)$ e $G_2(V, E_2)$, com $E_1 \subseteq E_2$, pretendemos descobrir, no *Problema-Sanduíche do Conjunto Homogêneo* (PSCH), se existe algum grafo-sanduíche $G_S(V, E_S)$ de (G_1, G_2) que contenha um conjunto homogêneo. Chamamos de *conjunto homogêneo sanduíche* (CHS) de (G_1, G_2) a um conjunto homogêneo existente em um grafo-sanduíche de (G_1, G_2) .

Uma propriedade é dita *hereditária* quando, dado que está presente num determinado grafo G , existirá também em todo subgrafo de G [13]. Como exemplos de propriedades hereditárias, temos: planaridade, total-desconectividade etc. Uma propriedade é dita *ancestral*, por outro lado, quando, dado que existe num grafo G , existirá igualmente em todo supergrafo de G . Exemplos de propriedades ancestrais são conectividade e hamiltonicidade.

Problemas-sanduíche para propriedades hereditárias ou ancestrais são solucionáveis pela simples execução de um algoritmo de reconhecimento clássico em G_1 (ou seja, o grafo de entrada sem qualquer aresta opcional) ou G_2 (o grafo de entrada com todas as arestas opcionais disponíveis), respectiva-

mente. Em se tratando de propriedades não-hereditárias e não-ancestrais, no entanto, não constitui tarefa simples transformar um algoritmo existente para um problema clássico de reconhecimento em algoritmo capaz de resolver eficientemente a versão sanduíche daquele problema. De fato, para muitas propriedades reconhecíveis polinomialmente em um grafo único (problema clássico de reconhecimento), o problema-sanduíche correspondente provou ser NP-completo [4, 7, 10, 11, 14, 20].

O fato é que um algoritmo para o problema clássico, não sanduíche, na maior parte das vezes não se mostra facilmente adaptável ao cenário com arestas opcionais. Como o número de grafos-sanduíche para o par de grafos (G_1, G_2) é exponencial no número de arestas opcionais (ou seja, em $|E_2 \setminus E_1|$), descarta-se de antemão qualquer possibilidade de se executar o algoritmo para a versão não-sanduíche em cada um dos grafos-sanduíche daquela instância. Sendo assim, novos algoritmos precisam ser considerados, mostrando-se a natureza do problema-sanduíche por vezes bastante diversa daquela de seu problema clássico correspondente. É natural, no entanto, que a existência de algoritmos eficientes para este não apenas estabeleça um limite inferior para aquele como também nos proponha, tacitamente, um desempenho a ser perseguido, como se a nos acenar com a possibilidade de melhorarmos cada vez mais os algoritmos de que dispomos.

Este é o caso do PSCH. Dada a existência de algoritmos *lineares* para sua contraparte clássica, ou seja, para o problema de reconhecimento de conjuntos homogêneos em grafos únicos [3, 16], os algoritmos até o momento conhecidos para o PSCH são relativamente pouco eficientes — fato este que constituiu a motivação inicial de nossa pesquisa.

1.2 Notação e convenções

Ao longo de todo o texto, utilizamos a seguinte notação, aqui destacada para mais fácil referência:

- (G_1, G_2) instância de entrada do PSCH
- $G_1(V, E_1)$ menor dos grafos da instância de entrada
- $G_2(V, E_2)$ maior dos grafos da instância de entrada (supergrafo de G_1)
- \overline{G}_i o grafo complementar de G_i
- V conjunto de vértices da instância de entrada
- n número de vértices da instância de entrada, ou seja, $n = |V|$
- m_{ob} número de *arestas obrigatórias*, ou seja, $m_{ob} = |E_1|$
- m_{op} número de *arestas opcionais*, ou seja, $m_{op} = |E_2 \setminus E_1|$
- m_{pr} número de *arestas proibidas*, ou seja, $m_{pr} = |V^2 \setminus E_2|$
- Δ_i grau máximo de G_i
- $\overline{\Delta}_i$ grau máximo de \overline{G}_i
- m o menor entre o número de arestas obrigatórias e proibidas, ou seja,
 $m = \min\{m_{ob}, m_{pr}\}$
- M o maior entre o número de arestas obrigatórias e proibidas, ou seja,
 $M = \max\{m_{ob}, m_{pr}\}$
- (a, b) aresta não-direcionada entre os vértices a e b
- $(a \rightarrow b)$ arco direcionado de a a b

- $N_{ob}(v)$ conjunto de *vizinhos obrigatórios* de v , ou seja,

$$N_{ob}(v) = \{x \in V \mid (x, v) \in E_1\}$$
- $N_{pr}(v)$ conjunto de *vizinhos proibidos* de v , ou seja,

$$N_{pr}(v) = \{x \in V \mid (x, v) \notin E_2\}$$

Ao longo de todo o texto, consideraremos que tanto o número m_{ob} de arestas obrigatórias quanto o número m_{pr} de arestas proibidas são limitados inferiormente por $n - 1$, uma vez que um número menor de arestas faria com que G_1 ou o complemento de G_2 , respectivamente, se tornassem desconexos — o que faria trivial a instância do problema, já que qualquer componente conexa não-unitária (quer seja de G_1 ou $\overline{G_2}$) constitui conjunto homogêneo sanduíche do par de entrada. Posto de outra forma, consideraremos sempre $m \geq n - 1$.

1.3 Histórico do problema

1.3.1 Pesquisa pré-existente

O primeiro algoritmo polinomial para o PSCH foi apresentado em 1998 por Cerioli *et al.* [2], estabelecendo um limite superior para o problema em $O(n^4)$. Alguns anos mais tarde, Tang *et al.* [18] publicaram um algoritmo bastante engenhoso, que, supostamente, teria baixado em muito aquele limite, servindo de base para um artigo de Habib *et al.* [12] sobre a versão de enumeração do PSCH. De fato, a complexidade de $O(n^2\Delta_2)$ do algoritmo de Tang *et al.* se manteve como o limite superior aceito para o problema até 2003, quando foi demonstrada sua incorreção [8, 17]. Àquela época, então, o melhor algoritmo para o problema teria voltado a ser o algoritmo $O(n^4)$ original de Cerioli *et al.*, mas um novo algoritmo — a que chamamos

algoritmo *Duas Fases* — foi proposto naquela mesma ocasião, melhorando ligeiramente o limite superior do problema para $O(n^2m)$.

1.3.2 Material inédito

Uma série de algoritmos cada vez mais eficientes surgiram desde então (em ordem cronológica de descoberta):

- um algoritmo $O(n^{3,5})$, baseado na técnica de balanceamento, a que chamamos algoritmo dos *Subconjuntos Balanceados*;
- um algoritmo randomizado de Monte Carlo, que resolve o PSCH em tempo $O(n^3)$, com erro conhecido ϵ ;
- dois algoritmos, apelidados *Série Harmônica* e *Cliques Crescentes*, que conseguem, de maneiras distintas, resolver o problema deterministicamente em tempo $O(n^3 \log n)$;
- outro algoritmo randomizado, desta vez um algoritmo de Las Vegas, que resolve o problema em tempo esperado $O(n^3)$ — e com taxa de acerto de 100%;
- uma melhoria do algoritmo das Cliques Crescentes (uma espécie de híbrido deste com o algoritmo das Duas Fases), a que nos referimos como algoritmo do *Preenchimento Acelerado*, e cuja complexidade de tempo é $O(n^3 \log \frac{m}{n})$;
- o algoritmo determinístico mais eficiente até o momento, chamado *Compleição de Pares*, de tempo $O(nm \log n)$.¹

¹Na verdade, o algoritmo do Preenchimento Acelerado é ainda o mais eficiente para um número pequeno de entradas, a saber, aquelas em que $M = \Omega(n \log n)$.

Uma vez que muitos dos algoritmos que apresentamos baseiam-se numa variação do procedimento auxiliar de *incorporação de testemunhas*, criado por Cerioli *et al.*, propomos também uma nova implementação daquele procedimento, que permite baixar sua complexidade de tempo de $O(n^2)$ para $O(M)$, otimizando, por conseguinte, os algoritmos que a empregam. Um efeito colateral desta melhoria é a obtenção da expressão mais justa $O(mM)$ para a complexidade do algoritmo Duas Fases apresentado em [17].

Da mesma forma, o digrafo auxiliar idealizado por Tang *et al.*, o *grafo-testemunha*, também é utilizado em alguns dos algoritmos propostos, de modo que apresentamos uma implementação mais eficiente para o procedimento que o obtém, reduzindo seu custo temporal de $O(n^2\Delta_2)$ para $O(nm)$.

Ainda que os algoritmos e a maioria dos conceitos teóricos aqui apresentados tenham sido criados especificamente para a solução do PSCH, em particular, não é menos verdade que boa parte das *técnicas* aqui empregadas são aplicáveis a outros problemas, investindo-as de caráter didático. Assinalamos, especialmente, a utilização da técnica do balanceamento e os algoritmos randomizados de Monte Carlo e de Las Vegas, a beleza teórica — e o poder prático — deste tipo de abordagem.

1.4 Organização do trabalho

Nesta tese, estudamos cada um dos algoritmos propostos numa seqüência cronológica, isto é, respeitando a ordem em que foram idealizados. Os dois últimos a serem aqui apresentados são, portanto, os que determinam o limite superior atual para o problema (trata-se de um daqueles casos assaz recorrentes em que um ou outro será o mais eficiente dependendo da esparsidade dos grafos de entrada). Acreditamos que essa forma de apresentação auxilie

no entendimento do arcabouço teórico que foi sendo aprendido e acumulado, ao longo das sucessivas descobertas, além de favorecer a percepção do inter-relacionamento de idéias que propicia o refinamento contínuo, tão presente em pesquisas deste teor.

O Capítulo 1 contém a formulação do problema e algumas definições básicas que lhe são peculiares.

O Capítulo 2 introduz o primeiro conceito específico para o PSCH, o de vértices *testemunhas*, e o procedimento básico de *incorporação de testemunhas*, que aparecerá em diversos momentos, ao longo do texto. Sua Seção 2.4 revisita o algoritmo de Cerioli *et al.* [2], importante para a compreensão do material seguinte.

No Capítulo 3, abordamos o *grafo-testemunha*, importante estrutura criada por Tang *et al.* [18], e que será utilizada em dois de nossos novos algoritmos. Em sua Seção 3.2, rerepresentamos o algoritmo original de Tang *et al.*, baseado no grafo-testemunha, e damos uma versão mais econômica de um contraexemplo apresentado em [17]. Mais uma vez, a exposição de material pré-existente se dá aqui, não apenas a título de completude, mas principalmente pela necessidade de se estabelecer claramente os alicerces teóricos para os capítulos vindouros. Há ainda uma seção devotada ao algoritmo *Duas Fases* (de nossa autoria, mas cuja criação é também anterior a este trabalho), o primeiro a utilizar com algum sucesso a idéia do grafo-testemunha [17].

A seguir, no Capítulo 4, é apresentada uma versão modificada do procedimento de incorporação de testemunhas, a que chamamos incorporação *incompleta* de testemunhas. Esta versão é utilizada em vários dos algoritmos que serão adiante estudados, a começar dos três que constam daquele mesmo capítulo: o algoritmo dos *Subconjuntos Balanceados*, da Seção 4.1, que utiliza a técnica do balanceamento; o primeiro algoritmo randomizado para o

problema, que é o algoritmo de Monte Carlo de erro unilateral baseado no sim, visto na Seção 4.2; e o algoritmo da *Série Harmônica* para o PSCH, de que trata a Seção 4.3.

Introduzimos, no Capítulo 5, um novo conceito: o de vértices *inimigos*. Esta idéia permitirá não apenas o nascimento do algoritmo das *Cliques Crescentes*, estudado na Seção 5.2, como também embasará o novo procedimento de incorporação de testemunhas *restrita por inimizadas*, peça principal do segundo algoritmo randomizado para o PSCH, agora um algoritmo do tipo de Las Vegas, ao qual a Seção 5.3 está dedicada. Um híbrido do algoritmo das Cliques Crescentes e do algoritmo Duas Fases é o algoritmo chamado *Preenchimento Acelerado*, que é abordado na Seção 5.4. Este algoritmo permaneceu, por bastante tempo ao longo de nossa pesquisa, como o algoritmo determinístico mais eficiente para o problema.

Finalmente, no Capítulo 6, vemos o algoritmo da Compleição de Pares, que traduz as melhores idéias para a solução do PSCH até o momento, e que sequer utiliza qualquer variação do procedimento de incorporação de testemunhas; por outro lado, emprega com sucesso o grafo-testemunha *estendido*, uma variação do digrafo idealizado por Tang *et al.* Este algoritmo veio substituir o algoritmo Preenchimento Acelerado como o mais eficiente para o PSCH para a maior parte das entradas, isto é, para todas aquelas em que haja da ordem $n \log n$ arestas obrigatórias *ou* proibidas, pelo menos — isto é, $M = \Omega(n \log n)$. Para as demais entradas, aquelas com um número tão grande de arestas opcionais que ambos m_{ob} e m_{pr} são $O(n \log n)$, o algoritmo Preenchimento Acelerado permanece como sendo o mais rápido que se conhece.

Como todos os algoritmos aqui apresentados foram de fato implementados no computador, pudemos verificar, na prática, seus comportamentos

distintos, para diferentes tipos de entrada. Os resultados obtidos figuram no Capítulo 7.

Encerrando esta tese, encontra-se o Capítulo 8 com nossa Conclusão e, por fim, os Apêndices, com algumas figuras extras e detalhes de implementação das rotinas auxiliares para as quais apresentamos melhorias.

Capítulo 2

Incorporação de Testemunhas

Sejam $G_1(V, E_1)$ e $G_2(V, E_2)$, com $E_1 \subseteq E_2$, os grafos de entrada do PSCH.

Nosso objetivo é muito simples: descobrir se, acrescentando a G_1 uma certa quantidade de arestas opcionais (isto é, de $E_2 \setminus E_1$), conseguimos um grafo-sanduíche $G_S(V, E_S)$, com $E_1 \subseteq E_S \subseteq E_2$, que possua algum conjunto homogêneo. Em outras palavras, queremos descobrir se (G_1, G_2) admite algum conjunto homogêneo sanduíche (CHS).

Começemos, portanto, com a mais simples das perguntas: dado um único grafo $G(V, E)$, o que faz com que um conjunto $F \subset V$, com $1 < |F| < |V|$, *não seja* um conjunto homogêneo do par de entrada? É evidente que a resposta é a existência de algum vértice b , não pertencente a F , tal que b é adjacente a algum vértice de F e, também, não-adjacente a algum outro vértice de F .

Agora, dado o par de entrada do PSCH (G_1, G_2) , perguntamos: o que faz com que um conjunto $F \subset V$, com $1 < |F| < |V|$, *não seja* um CHS do par de entrada? Em outras palavras, o que faz com que não exista, para aquele par, *grafo-sanduíche algum* do qual seja F um conjunto homogêneo? A resposta

é, também aqui, facilmente encontrada: a existência de algum vértice b , não pertencente a F , tal que b é *obrigatoriamente* adjacente a algum dos vértices de F (isto é, será adjacente a algum vértice de F em *todo* grafo-sanduíche daquele par) e, ao mesmo tempo, obrigatoriamente não-adjacente (ou *proibitivamente* adjacente) a algum outro vértice de F (isto é, será não-adjacente a algum vértice de F em todo grafo-sanduíche). Tais vértices b são chamados de *testemunhas* do conjunto F .

2.1 Testemunhas

Formalmente, definimos *vértice-testemunha* (ou apenas *testemunha*) de um conjunto $F \subset V$ como sendo todo vértice $b \notin F$ tal que existem dois vértices $v_i, v_j \in F$ para os quais $(b, v_i) \in E_1$ e $(b, v_j) \notin E_2$ (isto é, existe pelo menos uma aresta obrigatória e pelo menos uma aresta proibida entre b e F). Ao conjunto $B(F)$ que contém todos os vértices-testemunha de F dá-se o nome de *conjunto-testemunha* de F .

Como exemplo, podemos ver, na Figura 2.1, que o vértice 4 é testemunha do conjunto $F = \{2, 3\}$, dado que a aresta $(4, 2)$ é obrigatória (existe em G_1) e a aresta $(4, 3)$ é proibida (não existe em G_2). Os demais vértices não são testemunhas de F , pela ausência de arestas obrigatórias até F (como é o caso do vértice 6) ou proibidas (como é o caso dos vértices 1 e 5).

O Teorema 2.1, a seguir, caracteriza os CHS's e está presente, direta ou indiretamente, nas provas de corretude de todos os algoritmos conhecidos para o PSCH.

Teorema 2.1. [2] *O conjunto $H \subset V$, com $|H| \geq 2$, é um conjunto homogêneo sanduíche do par (G_1, G_2) se, e somente se, $B(H)$ é vazio.*

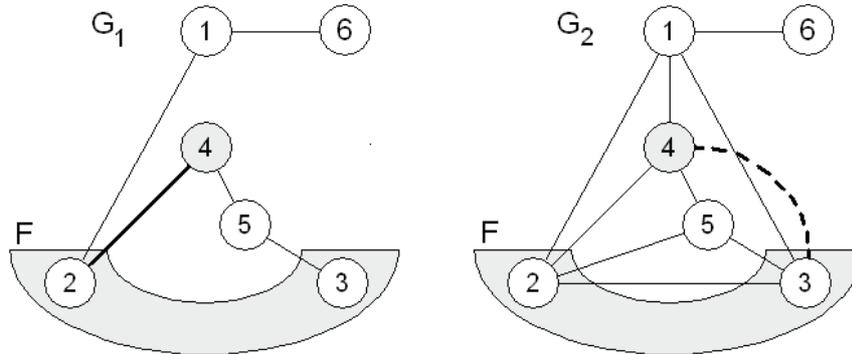


Figura 2.1: Vértice-testemunha

Demonstração. Suponhamos $B(H) \neq \emptyset$ e seja G_S um grafo sanduíche de (G_1, G_2) . Assim, qualquer vértice $b \in B(H)$ será ao mesmo tempo adjacente a algum vértice $v_i \in H$ e não-adjacente a algum vértice $v_j \in H$. Isto claramente impede que H seja um conjunto homogêneo de G_S . Como este raciocínio é válido para *todo* grafo-sanduíche de (G_1, G_2) , H jamais será um CHS daquele par.

Se supusermos, por outro lado, que $B(H) = \emptyset$, então é sempre possível construirmos um grafo-sanduíche de modo que H seja um seu conjunto homogêneo. Isto é conseguido da forma apresentada na Seção 2.2 a seguir. \square

2.2 Montagem do grafo-sanduíche a partir de um conjunto sem testemunhas

Uma vez que dispomos de um conjunto H cujo conjunto-testemunha é vazio, a obtenção de um grafo-sanduíche $G_S(V, E_S)$ da instância de entrada

(G_1, G_2) do PSCH para o qual H é de fato um conjunto homogêneo acontece da seguinte forma:

Parte-se de um conjunto de arestas E_S contendo todas as arestas obrigatórias, e apenas estas; em seguida, para todo vértice $x \in V \setminus H$ tal que (x, v_i) é aresta obrigatória para algum $v_i \in H$, adiciona-se a E_S as arestas (x, v) que ligam x a cada um dos vértices v pertencentes a H .

Dessa forma, cada vértice externo a H que possui algum vizinho obrigatório em H será *adjacente a todos* os vértices de H , no grafo sanduíche G_S ; e cada vértice externo a H que não possui vizinho obrigatório algum em H será *não-adjacente a todos* os vértices de H . É H , portanto, conjunto homogêneo de G_S .

Ressaltamos que o grafo obtido por tal procedimento é realmente um grafo-sanduíche válido do par de entrada. Isto é garantido pelo fato de que uma aresta não-obrigatória adicionada a E_S nunca haverá de ser do tipo *proibida* (pois, do contrário, o vértice $x \notin H$ incidente àquela aresta seria uma testemunha de H , contradizendo o fato de que seu conjunto-testemunha é vazio).

2.3 O procedimento de incorporação de testemunhas

Resulta diretamente do Teorema 2.1 que todo CHS contendo $H \subset V$ deve conter igualmente $B(H)$. Esta exigência deu origem, em [2], ao procedimento que chamamos de *incorporação de testemunhas*.

O objetivo do procedimento de incorporação de testemunhas é descobrir se a instância de entrada do PSCH possui algum CHS *que contenha um dado conjunto de vértices*. Partindo do conjunto dado (ou *candidato* a CHS)

Procedimento: Inc_Test ($G_1(V, E_1), G_2(V, E_2), H_1$)

1. $H \leftarrow H_1$
 2. **enquanto** $|H| < |V|$ **faça**
 - 2.1. **se** $B(H) = \emptyset$
retorne *sim* // um CHS foi localizado: H
 - 2.2. **senão**
 $H \leftarrow H \cup B(H)$
 3. **retorne** *não* // não há CHS's contendo H_1
-

Figura 2.2: O procedimento de incorporação de testemunhas

$H_1 \subset V$, serão obtidos, sucessivamente, $H_q = H_{q-1} \cup B(H_{q-1})$, até que (i) $B(H_q) = \emptyset$, caso este em que H_q é um CHS e o procedimento retorna *sim* (dizemos que a incorporação foi *bem-sucedida*), ou (ii) $H_q \cup B(H_q) = V$, quando então uma resposta *não* é retornada, significando a inexistência de qualquer CHS contendo o candidato inicial H_1 .

A Figura 2.2 apresenta o pseudo-código para o procedimento de incorporação de testemunhas.

2.4 Primeiro algoritmo: Incorporações Exaustivas

O procedimento de incorporação de testemunhas, por si só, é suficiente para mostrar que o PSCH é polinomialmente decidível. De fato, a idéia do algoritmo original de Cerioli *et al.* [2] é, simplesmente, a de executar

Algoritmo 1: Incorporações Exaustivas ($G_1(V, E_1), G_2(V, E_2)$)

1. **para cada** par de vértices $\{x, y\} \subset V$ **faça**
 - 1.1. **se** $\text{Inc_Test}(G_1, G_2, \{x, y\}) = \text{sim}$
retorne sim
 2. **retorne não**
-

Figura 2.3: O algoritmo das Incorporações Exaustivas [2]

uma incorporação de testemunhas para cada par de vértices da entrada do problema, como ilustrado na Figura 2.3. Como, por definição, não é possível existir CHS's com menos de dois vértices, essa estratégia claramente funciona. Referimo-nos a esse algoritmo como o algoritmo das *Incorporações Exaustivas* (IE).

Juntamente à determinação da polinomialidade do PSCH, a contribuição mais significativa encontrada em [2] é a maneira pela qual é conseguido um limite de tempo quadrático (no número de vértices da entrada) para cada execução da incorporação de testemunhas. Como a determinação do conjunto-testemunha $B(H_q)$ demanda tempo $O(n^2)$, e porque é possível que o tamanho do conjunto candidato cresça lentamente (mesmo à base de incrementos unitários), uma incorporação de testemunhas completa consumiria tempo $O(n^3)$. No entanto, uma vez que sejam utilizados alguns conjuntos auxiliares mantidos dinamicamente (como descrito em [2]), cada um dos conjuntos-testemunha $B(H_q)$ será obtido a partir da atualização de $B(H_{q-1})$ por meio de um número constante de uniões, diferenças e interseções em tempo linear. Dessa forma, toda a execução de uma incorporação de teste-

munhas ocorre em tempo $O(n^2)$, fornecendo-nos a complexidade global de $O(n^2 \cdot n^2) = O(n^4)$ para o algoritmo IE.

A bem da verdade, mostramos que há uma maneira ainda mais rápida de se executar a incorporação de testemunhas. A complexidade de tempo $O(M)$ que é, então, conseguida não apenas tem papel importante na análise de alguns dos algoritmos que serão aqui apresentados, como também nos dá o limite superior mais justo de $O(Mn^2)$ para o próprio algoritmo das Incorporações Exaustivas. Os detalhes técnicos encontram-se no Apêndice A.

Capítulo 3

O Grafo-Testemunha

3.1 Agrupando vértices interdependentes: o grafo-testemunha

Em 2001, Tang *et al.* [18] vislumbraram uma maneira deveras interessante de representar, de uma só vez, todas as relações existentes entre pares de vértices da instância do PSCH e seus vértices testemunhas: o *grafo-testemunha*. O que se busca conseguir, com o emprego do grafo-testemunha, é um rápido agrupamento de vértices interdependentes, ou seja, o particionamento dos vértices da entrada em conjuntos que não possam conter propriamente qualquer CHS daquela entrada.

O grafo-testemunha para o par $G_1(V, E_1), G_2(V, E_2)$ é um digrafo $G_T(V_T, E_T)$, onde $V_T = \{[x, y] \mid x, y \in V, x \neq y\}$, isto é, há um nó em V_T para cada par de vértices em V . Para cada um de seus nós $[x, y]$, haverá dois arcos direcionados de $[x, y]$ a $[x, b]$ e $[y, b]$ se, e somente se, o vértice b é um vértice-testemunha do conjunto $\{x, y\} \subset V$. Note que $[x, y]$ e $[y, x]$, no

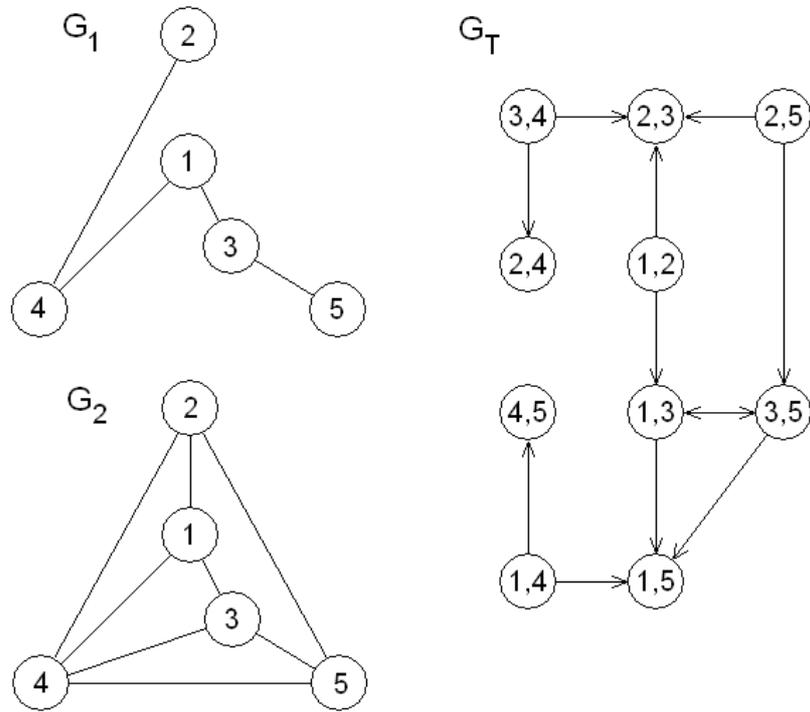


Figura 3.1: O grafo-testemunha

grafo-testemunha, são o mesmo nó. A Figura 3.1 dá o grafo-testemunha G_T para o par de grafos (G_1, G_2) que também lá se encontram.

Como apontado em [18], a obtenção do grafo-testemunha requer tempo $O(n^2\Delta_2)$. Este limite — que domina a complexidade de tempo de todo o algoritmo por eles proposto — não é, entretanto, justo. Apresentamos, no Apêndice B, um método de construção do grafo-testemunha em tempo $O(nm)$.

Embora nó e vértice sejam sinônimos em nosso contexto, note que estamos usando o termo *nó* para designar aqueles do grafo-testemunha e *vértices* para os demais. Tal distinção se mostrará útil para a fluidez da leitura

Algoritmo 2: Sumidouros Fortemente Conexos $(G_1(V, E_1), G_2(V, E_2))$

1. construa o grafo-testemunha G_T de (G_1, G_2)
 2. particione G_T em seus componentes fortemente conexos (CFC)
 3. encontre um CFC S de G_T que seja um sumidouro
 4. **se** $L(S) \neq V$
 retorne *sim* // $L(S)$ é um CHS
 5. **retorne** *não*
-

Figura 3.2: O algoritmo dos Sumidouros Fortemente Conexos [18]

durante a exposição de alguns dos algoritmos vindouros, em cujas análises estaremos nos referindo, constante e alternadamente, a ambos os conjuntos: o de *vértices* da entrada do problema, ou V ; e o de *nós* do grafo-testemunha, ou V_T . Aplicaremos recurso ortográfico semelhante para distinguir as *arestas* dos grafos de entrada dos *arcos* — direcionados — do grafo-testemunha.

Escreveremos $L(X)$ para designar o subconjunto de vértices $v \in V$ que aparecem no rótulo de algum nó do subgrafo $X \subseteq G_T$, referindo-nos a ele como o *conjunto rotulador* de X e a seus elementos como *vértices rotuladores* de X . Mais formalmente, $L(X) = \{v \in V \mid [v, z] \in X, \text{ para algum } z \in V\}$.

Infelizmente, o algoritmo proposto por Tang *et al.* em [18] era falho [8, 17]. Contudo, a engenhosidade de sua idéia central, o grafo-testemunha, veio a ser, por fim, parte integrante de alguns dos melhores algoritmos aqui apresentados.

3.2 Segundo algoritmo: Sumidouros Fortemente Conexos

Este a que nos referiremos como algoritmo dos *Sumidouros Fortemente Conexos* é o que foi proposto por Tang *et al.* no artigo em que o grafo-testemunha foi originalmente apresentado [18]. Ali, o grafo-testemunha foi empregado da seguinte forma: uma vez construído, passa-se à execução do método de Tarjan [19] para encontrar todos os seus componentes fortemente conexos (CFC) e, a seguir, determina-se, dentre eles, os *sumidouros fortemente conexos* (SFC) daquele digrafo, ou seja, os CFC's que não apresentam arcos de saída (para outros CFC's). O algoritmo escolhe, assim, um SFC S (sempre há pelo menos um, ainda que seja o digrafo inteiro), e então, caso os vértices do conjunto rotulador $L(S)$ do SFC escolhido não compreendam todos os vértices da instância do problema, o algoritmo responde *sim* (evidentemente, à pergunta do problema: “existe CHS?”) e retorna $L(S) \subset V$ como sendo um conjunto homogêneo sanduíche. Caso contrário, o algoritmo retorna *não*.

Os passos resumidos do algoritmo dos Sumidouros Fortemente Conexos são mostrados na Figura 3.2. Reescrevemos, a seguir, a proposição equivocada que o motivou.

Proposição 3.1. [18] *Um conjunto $H \subset V$, $|H| \geq 2$, é um conjunto homogêneo sanduíche de (G_1, G_2) se, e somente se, H é o conjunto rotulador de algum sumidouro fortemente conexo propriamente contido no grafo-testemunha G_T daquela instância.*

A proposição acima é a própria essência do algoritmo dos Sumidouros Fortemente Conexos.

Ainda que seja verdade o fato de que (G_1, G_2) não podem apresentar qualquer conjunto homogêneo sanduíche caso G_T não possua um SFC próprio, não é certo assumir que *todo* SFC de G_T esteja associado a um conjunto homogêneo sanduíche daquela instância. De fato, pode ocorrer de um SFC não conter *todos* os pares formados por seus vértices rotuladores, isto é, há a possibilidade da existência de um *par faltante* $\{x, y\} \subset L(S)$ tal que $[x, y] \in G_T \setminus S$. Caso o par $\{x, y\}$ apresente um vértice-testemunha b que não esteja contido em $L(S)$, b será testemunha também de $L(S) \supset \{x, y\}$, e assim $L(S)$ não será, evidentemente, um conjunto homogêneo sanduíche — embora seja S um SFC próprio de G_T . Apresentamos, em [17], um contraexemplo mínimo para a Proposição 3.1, onde o conjunto rotulador de um SFC próprio não é CHS do par de entrada.

Ainda assim, mostramos, no Apêndice C, outro contraexemplo. Trata-se de instância muito interessante, para a qual o algoritmo não apenas encontra um conjunto homogêneo sanduíche *falso*, como também é incapaz de encontrar o (único) CHS que *de fato existe!* (É evidente que, em se tratando de um problema de decisão, o algoritmo, nesse caso — e por mero acaso — daria a resposta certa *sim*).

3.3 Subgrafos fechados-por-pares

Seja $G_T(V_T, E_T)$ o grafo-testemunha dos grafos $G_1(V, E_1), G_2(V, E_2)$ da entrada do problema. Um subgrafo $K \subseteq G_T$ é dito *fechado-por-pares* se, e somente se, $x, y \in L(K)$ implica $[x, y] \in K$. Como exemplo, o subgrafo cujo conjunto de vértices é $Q = \{[1, 2], [1, 3], [2, 3]\}$ é fechado-por-pares, enquanto aquele cujo conjunto de vértices é $Q' = \{[1, 2], [1, 3], [1, 4], [2, 3], [2, 4]\}$ não

é, já que os vértices 3 e 4 estão entre os vértices rotuladores de Q' mas $[3, 4] \notin Q'$.

Denotaremos $G_T\langle H \rangle$ para nos referirmos ao subgrafo induzido de G_T pelo conjunto que contém todos os nós que são rotulados por vértices de H . Ou seja, $G_T\langle H \rangle$ é, por definição, fechado-por-pares e seu conjunto rotulador é $L(G_T\langle H \rangle) = H$.

O Teorema 3.2 a seguir emprega o grafo-testemunha numa correta caracterização de conjunto homogêneo sanduíche, embasando, além do algoritmo *Duas Fases* que ora apresentamos, também os algoritmos *Preenchimento Acelerado* e *Compleição de Pares*, estudados mais adiante.

Teorema 3.2. *Um conjunto $H \subset V$, $|H| \geq 2$, é um conjunto homogêneo sanduíche de (G_1, G_2) se, e somente se, o subgrafo fechado-por-pares $G_T\langle H \rangle$ é sumidouro propriamente contido no grafo-testemunha G_T daquela instância.*

Demonstração. Seja $K = G_T\langle H \rangle$ o subgrafo fechado-por-pares que contém todos os nós $[v, w]$ do grafo-testemunha tais que $v, w \in H$, e apenas estes.

Partamos da hipótese de que K é um sumidouro em G_T . Suponhamos agora, por contradição, que $H = L(K)$ não é um CHS de (G_1, G_2) . Então, H possui um vértice-testemunha $b \in V \setminus H$, o que significa haver uma aresta obrigatória entre b e algum vértice $h_1 \in H$ e também uma aresta proibida entre b e algum outro vértice $h_2 \in H$, de forma que b seja testemunha de $\{h_1, h_2\} \subseteq H$. Mas isto implica que, no grafo-testemunha, o nó $[h_1, h_2] \in K$ possui arcos de saída para os nós $[h_1, b]$ e $[h_2, b]$, que não estão em K (pois K é fechado-por-pares e b não é vértice rotulador de K). Isto é uma contradição, pois não há arcos de um nó em K para um nó que não esteja em K (K é sumidouro, por hipótese).

Por outro lado, assumamos a hipótese de que H é um CHS. Tomemos, então, o subgrafo fechado-por-pares K tal que $L(K) = H$ (ou seja, $K = G_T\langle H \rangle$) e suponhamos, por absurdo, que K não é um sumidouro, isto é, que existe um arco de um nó $[h_1, h_2]$ em K para um nó $[h_1, b]$ que não esteja em K . A existência deste arco nos diz que b é testemunha de $\{h_1, h_2\}$. Como $h_1 \in L(K)$ e $[h_1, b]$ não está em K , então b não pode estar em $L(K) = H$ (K é fechado-por-pares). Assim sendo, b é testemunha de um par de vértices de H e b não está em H , o que faz de b uma testemunha do próprio H . Mas isto é uma contradição, pois H é, por hipótese, um CHS — donde K não pode ter arcos de saída, isto é, K é de fato um *sumidouro* fechado-por-pares. \square

Ressaltem-se, aqui, as duas sutis, porém cruciais, diferenças entre o Teorema 3.2 e a Proposição 3.1: o sumidouro ao qual está associado um conjunto homogêneo sanduíche *não é necessariamente fortemente conexo*; precisa, sim, ser *fechado-por-pares*.

Como não é conhecido método rápido para localizar sumidouros fechados-por-pares, o Teorema 3.2 parece não nos apontar nenhum caminho muito promissor. No entanto, o Corolário 3.3 abaixo traz a necessária inspiração para o algoritmo seguinte.

Corolário 3.3. *Se $H \subset V$ é um CHS dos grafos G_1, G_2 , então o subgrafo fechado-por-pares $G_T\langle H \rangle \subset G_T$ contém (propriamente ou não) um sumidouro fortemente conexo do grafo-testemunha G_T .*

Demonstração. Pelo Teorema 3.2, sabemos que $K = G_T\langle H \rangle$ é sumidouro de G_T . Se K é fortemente conexo, então o enunciado vale. Se não é, então há dois nós $\alpha, \beta \in K$ tais que não há caminho de α até β . O conjunto $R(\alpha)$ de todos os nós que são *atingíveis* a partir de α induz um sumidouro pro-

priamente contido em K . Assim, o fato de que K não é fortemente conexo implica que K contém, propriamente, um sumidouro qualquer K' . Este sumidouro $K' \subset K$, por sua vez, ou é ele mesmo fortemente conexo ou então, pelo mesmo raciocínio, contém propriamente um sumidouro K'' , e assim por diante. Como K é finito, é preciso que essa seqüência de sumidouros propriamente contidos pare, e teremos então achado um sumidouro fortemente conexo contido em K . \square

3.4 Terceiro algoritmo: Duas Fases

O algoritmo a que chamamos *Duas Fases* (2F) é bastante simples e pode ser entendido de duas formas:

- uma melhoria em relação ao algoritmo das Incorporações Exaustivas, onde, em lugar de submetermos ao procedimento de incorporação de testemunhas todos os $O(n^2)$ pares de vértices da entrada, submetemos apenas um número menor — porém suficiente — de conjuntos candidatos;
- a adição de uma segunda etapa ao algoritmo dos Sumidouros Fortemente Conexos, onde, após termos determinado os SFC's do grafo-testemunha, não nos apressaremos em retornar o conjunto rotulador de qualquer deles à guisa de CHS, mas sim submeteremos seus conjuntos rotuladores à verificação quanto a estarem ou não contidos em algum CHS daquele par de entrada.

A Figura 3.3 ilustra a mecânica do algoritmo das Duas Fases. Baseados no fato de que o procedimento de incorporação de testemunhas corretamente responde se um conjunto inicial está ou não contido em algum CHS, vemos

Algoritmo 3: Duas Fases ($G_1(V, E_1), G_2(V, E_2)$)

1. construa o grafo-testemunha G_T de (G_1, G_2)
 2. particione G_T em seus componentes fortemente conexos
 3. localize todos os SFC's S_i de G_T
 4. **para cada** S_i **faça**
 - 4.1. $H_i \leftarrow L(S_i)$
 - 4.2. **se** $\text{Inc_Test}(G_1, G_2, H_i) = \text{sim}$
retorne sim
 5. **retorne não**
-

Figura 3.3: O algoritmo das Duas Fases

facilmente que toda resposta *sim* dada pelo algoritmo é correta. Por outro lado, o Corolário 3.3 nos garante que, em existindo um sumidouro fechado-por-pares (e, portanto, um CHS), haverá algum sumidouro fortemente conexo nele contido, donde a suficiência de investigarmos, como candidatos iniciais, apenas os conjuntos rotuladores dos SFC's do grafo-testemunha.

A prova de corretude e a análise de complexidade do algoritmo Duas Fases encontram-se mais detalhadas em [17]. A rerepresentação deste algoritmo no presente trabalho visa não apenas permitir uma melhor compreensão do algoritmo de *Compleição de Pares*, exposto mais adiante, como também propor o refinamento de sua análise de complexidade, uma vez que tanto a complexidade do procedimento de incorporação de testemunhas quanto a da construção do grafo-testemunha foram por nós redefinidas através da descoberta de métodos mais eficientes para conseguí-los (vide Apêndices A e B).

3.4.1 Refinamento da análise de complexidade

A complexidade de tempo de todos os passos da primeira fase do algoritmo das Duas Fases (linhas 1 a 3, na Figura 3.3) é linear no número de nós mais arcos do grafo-testemunha, ou seja, pode ser expressa como $O(n^2 + nm)$. (Novamente, referimos o leitor ao Apêndice B para detalhes sobre a cardinalidade em arcos do grafo-testemunha e sua construção.) Como já o adiantáramos na Introdução, $m = \Omega(n)$ para que o problema não seja trivial, o que nos permite escrever $O(nm)$ como a complexidade de tempo desta primeira fase.

A segunda fase (linha 4 em diante) faz diversas chamadas ao procedimento de incorporação de testemunhas. O tempo exigido por cada execução daquele procedimento é $O(m_{ob} + m_{pr}) = O(M)$, como pode ser visto no Apêndice A. Assim, a questão é: quantas vezes, no pior caso, será a incorporação de testemunhas executada? Ora, dado que o algoritmo pára quando um CHS é encontrado, podemos reformular a pergunta anterior como: qual o maior tamanho possível de uma seqüência de SFC's cujos conjuntos rotuladores *não sejam* CHS's da instância de entrada?

Segundo o Teorema 3.2, todo sumidouro próprio do grafo-testemunha que é fechado-por-pares corresponde a um CHS. Dessa forma, o tamanho máximo de nossa seqüência de SFC's que falham em apontar CHS's é igual ao maior número possível de SFC's que *não são fechados-por-pares*. O Lema 3.4, a seguir, estabelece um limite superior para este número.

Lema 3.4. *Em um grafo-testemunha, há no máximo $O(m)$ sumidouros fortemente conexos que não são fechados-por-pares.*

Demonstração. Seja $G_T(V_T, E_T)$ o grafo-testemunha dos grafos $G_1(V, E_1)$, $G_2(V, E_2)$ e seja $P \subset V_T$ um SFC de G_T que *não é* fechado-por-pares. Seja também $[x, y]$ um nó pertencente a P . O nó $[x, y]$ apresenta, necessariamente, um arco de saída, pois P , não sendo fechado-por-pares, não pode ser unitário e, além disso, P é fortemente conexo, donde nenhum de seus nós pode ser um sumidouro. Seja, ainda, $([x, y] \rightarrow [x, t])$ um arco de saída de $[x, y]$. Ora, o vértice $t \in V$ é, por construção, um vértice-testemunha de $\{x, y\}$, o que implica a existência de uma aresta obrigatória e uma proibida entre t e os vértices daquele par. Sem perda de generalidade, seja (x, t) a aresta obrigatória e (y, t) a aresta proibida.

Definamos uma função $rotula_{ob}(P)$ que associa o SFC P a uma tal aresta obrigatória (x, t) daquela instância.¹

Note que tal função do conjunto de SFC's de G_T que não são fechados-por-pares sobre contra-domínio E_T é injetora, isto é, não pode haver dois SFC's distintos P_i e P_j tais que $rotula_{ob}(P_i) = rotula_{ob}(P_j)$, uma vez que a aresta retornada pela função é sempre aresta obrigatória existente entre dois vértices de V que rotulam um dos nós de um componente fortemente conexo (em particular, um SFC) e, como se sabe, componentes fortemente conexos particionam o conjunto de nós de um grafo. Sendo assim, o número de SFC's que não são fechados-por-pares é, no máximo, igual ao número de arestas obrigatórias, ou $O(m_{ob})$.

É evidente que função análoga $rotula_{pr}(P)$ pode ser definida para retornar a aresta *proibida* associada a um arco interno de P , donde se infere um limite

¹Para que tal função seja determinística, podemos, por exemplo, estabelecer uma ordem total qualquer para os nós do grafo-testemunha e tomarmos $[x, y]$ e $[x, t]$ como os dois nós de P mais à frente, segundo aquela ordenação.

superior $O(m_{pr})$ e, conseqüentemente, $O(m)$, para o número de SFC's que não são fechados-por-pares. \square

O tempo computacional que demanda a segunda fase do algoritmo das Duas Fases é, portanto, $O(m)$ vezes o tempo $O(M)$ de cada execução da incorporação de testemunhas, ou seja, sua complexidade é $O(mM)$.

Dominada pela de sua segunda fase, a complexidade temporal de todo o algoritmo Duas Fases é, portanto, $O(mM)$.

Capítulo 4

Incorporação Incompleta de Testemunhas

Antes de passarmos aos próximos algoritmos para o PSCH, apresentaremos uma variação do procedimento de incorporação de testemunhas. Nela estará baseada a maior parte dos algoritmos vindouros. Nós a chamamos incorporação *incompleta* de testemunhas.

A entrada do procedimento de incorporação incompleta de testemunhas não é composta apenas de um par de grafos $G_1(V, E_1), G_2(V, E_2)$ e de um candidato a conjunto homogêneo $H_1 \subset V$, mas também de um *parâmetro de parada* $k \leq n$. A única diferença entre a versão original (completa) e a versão incompleta daquele procedimento é a de que, nesta última, quando o tamanho $|H_q|$ do candidato corrente se tornar maior do que k , o procedimento será prematuramente interrompido retornando *não*. Ressaltamos que uma resposta *não* obtida pelo procedimento de incorporação incompleta de testemunhas com parâmetro de parada k significa que H_1 não está contido em qualquer CHS com k vértices ou menos.

Procedimento: Inc_Incompleta_Test ($G_1(V, E_1), G_2(V, E_2), H_1, k$)

1. $H \leftarrow H_1$
 2. **enquanto** $|H| \leq \min\{k, |V| - 1\}$ **faça**
 - 2.1. **se** $B(H) = \emptyset$
retorne sim // *um CHS foi encontrado: H*
 - 2.2. **senão**
 $H \leftarrow H \cup B(H)$
 3. **retorne não** // *não há CHS's com k vértices*
// *ou menos que contenham H₁*
-

Figura 4.1: O procedimento de incorporação incompleta de testemunhas

A versão incompleta da incorporação de testemunhas claramente generaliza sua versão completa, uma vez que esta equivale àquela com parâmetro de entrada $k = n$.

O pseudo-código para a incorporação incompleta de testemunhas encontra-se na Figura 4.1.

Com a utilização das mesmas estruturas de dados definidas em [2], a incorporação incompleta de testemunhas roda em tempo $O(nk)$. Se, no entanto, o método otimizado para a incorporação de testemunhas apresentado no Apêndice 3 for adaptado para permitir a incorporação incompleta, podemos melhorar ligeiramente o limite de tempo para $O(\min\{M, k\Delta_1 + k\bar{\Delta}_2\})$. No entanto, este limite é de difícil aplicação na análise dos algoritmos que empregam a incorporação incompleta de testemunhas. O limite mais simples, embora menos justo, de $O(nk)$, é suficiente para nossos propósitos e será, então, por nós doravante empregado.

4.1 Quarto algoritmo: Subconjuntos Balanceados

O algoritmo que propomos nesta seção, que será referido como algoritmo dos *Subconjuntos Balanceados* (*SB*, de forma abreviada), é bastante similar ao primeiro algoritmo IE, no sentido de que, neste, também será submetido à incorporação de testemunhas cada um dos pares de vértices da instância de entrada do problema. A diferença é que, aqui, o algoritmo proposto estabelece certa ordem particular de submissão dos pares àquele procedimento, de tal forma que ele possa se beneficiar, num dado momento, dos resultados obtidos de incorporações de testemunhas *que já tenham acontecido até ali*. Após algumas incorporações de testemunhas mal-sucedidas terem ocorrido, terão sido revelados alguns pares de vértices que sabidamente não estão contidos em quaisquer CHS's daquela entrada. Este conhecimento será então utilizado pelo algoritmo, que poderá economizar tempo interrompendo mais cedo futuras incorporações, sem qualquer detrimento de completude.

A Figura 4.2 ilustra o pseudo-código do algoritmo Subconjuntos Balanceados.

Quando o algoritmo começa, os n vértices dos grafos da entrada do problema são particionados em $\lceil \sqrt{n} \rceil$ subconjuntos disjuntos C_i , cada qual com tamanho $O(\sqrt{n})$. Em seguida, todos os pares de vértices serão submetidos à incorporação de testemunhas em duas fases distintas: na primeira fase (linhas 4 e 4.1, na Figura 4.2), todos os pares de vértices *de um mesmo subconjunto* C_i (e apenas esses) serão submetidos; na segunda fase (linhas 5 e 5.1), serão submetidos todos os demais pares (ou seja, aqueles formados por vértices que *não* pertencem a um mesmo subconjunto C_i). Dessa forma, todos os possíveis pares de vértices da entrada terão sido investigados quanto a pertencerem

Algoritmo 4: Subconjuntos Balanceados ($G_1(V, E_1), G_2(V, E_2)$)

1. rotule todos os vértices em V de v_1 a v_n
 2. crie $\lceil \sqrt{n} \rceil$ conjuntos C_i , inicialmente vazios
 3. **para cada** vértice $v_j \in V$ **faça**
 - 3.1. $C_{j \bmod \lceil \sqrt{n} \rceil} \leftarrow C_{j \bmod \lceil \sqrt{n} \rceil} \cup \{v_j\}$
 4. **para cada** par de vértices $\{x, y\}$ do mesmo conjunto C_i **faça**
 - 4.1. **se** $\text{Inc_Test}(G_1, G_2, \{x, y\}) = \text{sim}$
retorne sim
 5. **para cada** par de vértices $\{x, y\}$ que não estejam num mesmo C_i **faça**
 - 5.1. **se** $\text{Inc_Incompleta_Test}(G_1, G_2, \{x, y\}, \lceil \sqrt{n} \rceil) = \text{sim}$
retorne sim
 6. **retorne não**
-

Figura 4.2: O algoritmo Subconjuntos Balanceados para o PSCH

ou não a algum CHS daquela instância do problema, tal como no algoritmo IE. Eis o ponto-chave: se nenhuma incorporação de testemunhas da primeira fase encontrou um CHS, então é fato que aquela instância não possui nenhum CHS que contenha dois vértices de um mesmo subconjunto C_i . Dessa forma, o tamanho máximo de qualquer possível CHS daquela instância será $\lceil \sqrt{n} \rceil$ (o número de subconjuntos nos quais foram dispersados os vértices da entrada), garantindo que todos os procedimentos de incorporação de testemunhas a partir de então não precisarão procurar por CHSs de maior tamanho. Por esta razão, incorporações *incompletas* de testemunhas com parâmetro de parada $k = \lceil \sqrt{n} \rceil$ poderão ser seguramente utilizadas.

4.1.1 Prova de corretude

Teorema 4.1. *O algoritmo SB reconhece corretamente se o par de entrada admite um CHS.*

Demonstração. Se o algoritmo retorna *sim*, então ele encontrou um conjunto $H \subset V$, com $|H| \geq 2$, tal que o conjunto testemunha de H é vazio. Dessa forma, H é de fato um CHS válido.

Agora, suponhamos que a entrada possua um CHS H . Se $|H| > \lceil \sqrt{n} \rceil$ então há mais vértices em H do que subconjuntos entre os quais os vértices de entrada foram inicialmente distribuídos (linha 3.1). Assim, pelo princípio da casa do pombo, é preciso haver dois vértices $x, y \in H$ pertencentes ao mesmo subconjunto C_i . Dessa forma, quando for o momento em que $\{x, y\}$ seja submetido à incorporação (completa) de testemunhas (linha 4.1), o algoritmo irá encontrar um CHS. Por outro lado, se $|H| \leq \lceil \sqrt{n} \rceil$, então é possível que H não contenha dois vértices quaisquer de um mesmo subconjunto C_i , o que faria com que todas as incorporações de testemunhas da primeira fase do algoritmo falhassem. Nesse caso, entretanto, quando o primeiro par de vértices contido em H for submetido à incorporação incompleta de testemunhas (linha 5), um CHS será certamente encontrado, uma vez que o tamanho de H é, por hipótese, menor ou igual ao parâmetro de parada $k = \lceil \sqrt{n} \rceil$. \square

4.1.2 Análise de complexidade

Como cada subconjunto C_i possui $O(n)$ pares de vértices e há um número $O(\sqrt{n})$ de tais subconjuntos, a quantidade de pares que serão submetidos à incorporação de testemunhas durante a primeira fase do algoritmo é $O(n\sqrt{n})$. Todos os procedimentos de incorporação, nessa fase, são completos e de-

mandam, portanto, tempo $O(n^2)$ para serem executados, o que acarreta um subtotal de $O(n^{3.5})$ para o tempo de toda a primeira fase.

O número de pares que são submetidos à incorporação de testemunhas durante a segunda fase do algoritmo SB é $O(n^2) - O(n\sqrt{n}) = O(n^2)$ pares. Cada incorporação de testemunhas, nessa fase, é do tipo incompleta com parâmetro de parada $k = \lceil \sqrt{n} \rceil$. Como a complexidade de tempo de cada incorporação incompleta de testemunhas com parâmetro de parada k é $O(nk)$, o tempo total gasto por toda a segunda fase do algoritmo é $O(n^3k) = O(n^{3.5})$.

Assim, a complexidade de tempo global para o algoritmo Subconjuntos Balanceados é $O(n^{3.5}) + O(n^{3.5}) = O(n^{3.5})$.

4.2 Quinto algoritmo: Monte Carlo

Algoritmos randomizados de Monte Carlo *baseados no sim*, para problemas de decisão, são tais que sempre estarão retornando a resposta correta quando responderem *sim* (um certificado para o *sim* é apresentado), ao passo que respostas *não*, emitidas por tais algoritmos, podem não corresponder sempre à verdade. Sua probabilidade de erro é, no entanto, limitada por alguma constante ϵ conhecida. Em outras palavras, são algoritmos que sempre responderão *não* se a resposta correta para uma determinada entrada for *não* (já que são incapazes de inventar falsos certificados para o *sim*), e que responderão *sim* com probabilidade $p \geq 1 - \epsilon$ para entradas cuja resposta correta é *sim*.

De forma a adquirirmos alguma intuição a respeito do próximo algoritmo que apresentaremos, suponhamos que uma instância para o PSCH possui um CHS H com h vértices ou mais.

Qual seria, nesse caso, a probabilidade \bar{p}_1 de que um par de vértices $\{x, y\} \in V$, escolhido randomicamente, *não* estivesse contido em H ? Claramente,

$$\bar{p}_1 \leq 1 - \frac{h(h-1)}{n(n-1)}.$$

E o que dizer da probabilidade \bar{p}_t de que, dentre t pares de vértices randomicamente escolhidos, nenhum deles esteja contido em H ? É fácil ver que

$$\bar{p}_t \leq \left(1 - \frac{h(h-1)}{n(n-1)}\right)^t.$$

Ainda sob a hipótese de que exista um CHS H com h ou mais vértices, qual seria, agora, a probabilidade p_t de que, após t procedimentos de incorporação de testemunhas tenham sido executados (a partir de t pares de vértices escolhidos de forma randômica), algum CHS daquela instância de entrada tenha sido encontrado? Novamente, não é difícil chegarmos à seguinte expressão:

$$p_t \geq 1 - \left(1 - \frac{h(h-1)}{n(n-1)}\right)^t. \quad (4.1)$$

Se, ao invés de obtermos a probabilidade p_t a partir da expressão acima, nós *fixássemos* p_t em algum valor desejado $p = 1 - \epsilon$, seríamos capazes de calcular o menor valor inteiro h_t (denotando, por simplicidade, h em função de t) que satisfaz a desigualdade 4.1. Este valor h_t é tal que a execução de t incorporações de testemunhas independentes (em t pares aleatórios) é *suficiente para localizar um CHS da instância de entrada do PSCH, com probabilidade maior ou igual a p , dado que aquela instância possui algum CHS com h_t vértices ou mais* (veja equação 4.2):

$$h_t = \left\lceil \frac{1 + \sqrt{1 + 4(n^2 - n)(1 - (1 - p)^{1/t})}}{2} \right\rceil. \quad (4.2)$$

Todavia, tendo em vista a obtenção de um algoritmo Monte Carlo baseado no sim para o PSCH, desejamos ser capazes de encontrar um CHS com probabilidade p *caso a entrada possua qualquer CHS, não importando seu tamanho*. Como h_t decresce com o aumento do número t de pares, surge a seguinte questão: quantos pares aleatórios de vértices precisamos submeter à incorporação de testemunhas de forma a conseguir a desejada probabilidade de sucesso? A resposta é bastante simples: o menor inteiro t' tal que $h_{t'} = 2$, uma vez que 2 é o menor tamanho possível para um CHS!

A determinação de t' advém diretamente da equação 4.2 (os cálculos encontram-se em maior detalhe na Subseção 4.2.2:

$$t' = \frac{\ln(1-p)}{\ln\left(1 - \frac{2}{n(n-1)}\right)} = \Theta(n^2). \quad (4.3)$$

Como o número t' de incorporações de testemunhas que precisam ser realizadas a partir de pares de vértices escolhidos randomicamente é $\Theta(n^2)$, e sendo $O(n^2)$ a complexidade de cada uma das execuções do procedimento de incorporação, parece estarmos lutando por um algoritmo randomizado de tempo $O(n^4)$. Tal algoritmo seria absolutamente indesejável, dado ser possível resolver o problema de forma *determinística* em menos tempo (vide, por exemplo, a Seção 4.1 anterior)!

Agora, porém, chegamos ao ponto em que a versão *incompleta* do procedimento de incorporação de testemunhas desempenhará, mais uma vez, importante papel no que concerne à economia de tempo computacional. Mostraremos que, ao tempo da t -ésima execução daquele procedimento (isto é, quando for sorteado o t -ésimo par aleatório de vértices), sua versão incompleta com parâmetro de parada $k = h_{t-1}$ terá exatamente a mesma serventia que sua versão completa teria.

Lema 4.2. *Com o objetivo de encontrar um CHS com probabilidade p , caso exista algum com h_t ou mais vértices, o t -ésimo procedimento de incorporação de testemunhas pode ser interrompido quando o tamanho do conjunto candidato exceder h_{t-1} .*

Demonstração. Duas são as possibilidades (mutuamente exclusivas) com relação a uma dada instância de entrada do PSCH: (i) ela admite algum CHS com mais do que h_{t-1} vértices; ou (ii) ela não admite CHS algum com mais do que h_{t-1} vértices.

Se (i) é verdade, então mais do que $t - 1$ incorporações de testemunhas não precisam sequer ser executadas para garantir a probabilidade desejada p (pela definição de h_t) e, sendo assim, a t -ésima execução da incorporação de testemunhas pode parar a *qualquer* momento.

Se (ii) é verdade, então uma incorporação incompleta de testemunhas com parâmetro de parada $k = h_{t-1}$ retornará exatamente a mesma resposta que seria retornada por uma incorporação *completa*, já que não há CHS's com mais de h_{t-1} vértices (por hipótese).

Qualquer que seja o caso, portanto, o procedimento *incompleto* é perfeitamente suficiente. □

Nesse ponto, podemos escrever um algoritmo Monte Carlo eficiente para o PSCH, que o responde corretamente com probabilidade maior ou igual a p .

A idéia do algoritmo que estamos propondo é a de rodar diversas incorporações de testemunhas a partir de conjuntos candidatos (pares de vértices) escolhidos randomicamente. A cada iteração t do algoritmo será executada uma incorporação incompleta com parâmetro de parada $k = h_{t-1}$, durante a qual ou bem encontraremos com sucesso um CHS (e o algoritmo, nesse caso, retorna *sim*), ou então a incorporação corrente (a t -ésima) será abortada

Algoritmo 5: Monte Carlo $(G_1(V, E_1), G_2(V, E_2), p)$

1. $h \leftarrow |V| - 1$
 2. $t \leftarrow 1$
 3. **enquanto** $h \geq 2$
 - 3.1. seja (v_1, v_2) um par de vértices de V randomicamente escolhidos
 - 3.2. **se** $\text{Inc_Incompleta_Test}(G_1, G_2, \{v_1, v_2\}, h) = \text{sim}$
retorne *sim*
 - 3.3. $h \leftarrow \lfloor (1 + \sqrt{1 + 4(|V|^2 - |V|)(1 - (1 - p)^{1/t})})/2 \rfloor$
 - 3.4. $t \leftarrow t + 1$
 4. **retorne** *não*
-

Figura 4.3: Um algoritmo Monte Carlo sim-baseado para o PSCH

quando o número de vértices no conjunto candidato exceder o limite de h_{t-1} (com base no Lema 4.2). Para a primeira iteração, o parâmetro de parada k é fixado em $h_0 = n$, de forma que corresponderá a uma incorporação de testemunhas *completa*. Ao término de cada iteração, o valor h_t é atualizado (vide equação 4.2). Isto faz com ele decresça continuamente ao longo das sucessivas iterações até que atinja o valor 2 (o tamanho mínimo permitido para um conjunto homogêneo), o que necessariamente acontecerá após $\Theta(n^2)$ iterações (vide equação 4.3).

O pseudo-código para o algoritmo Monte Carlo para o PSCH encontra-se na Figura 4.3.

4.2.1 Prova de corretude

Teorema 4.3. *O algoritmo apresentado na Figura 4.3 é um algoritmo de Monte Carlo que responde corretamente se existe algum CHS para os grafos da entrada do PSCH com probabilidade maior ou igual a p .*

Demonstração. Se a resposta correta é *não*, o algoritmo responde corretamente *não* com probabilidade 1, pois não é possível que dê *erradamente* uma resposta positiva, uma vez que todo *sim* decorre de ter sido encontrado um conjunto $H \subset V$, com $|H| \geq 2$, tal que o conjunto testemunha de H é vazio.

Se, por outro lado, a resposta correta é *sim*, queremos mostrar que o algoritmo responde corretamente *sim* com probabilidade pelo menos p . Seja h^* o tamanho do maior CHS do par de grafos da entrada do problema. Como $h_0 = n - 1$ e o algoritmo apenas responde *não* após h_t ter decrescido até atingir o valor 2, é forçoso existir um índice d tal que $h_d \leq h^* \leq h_{d-1}$. Pela definição de h_t sabemos que, dada a hipótese de que a entrada possui um CHS com h_t ou mais vértices, t incorporações de testemunhas são suficientes para encontrar um CHS com probabilidade pelo menos p . Como, por hipótese, *existe* um CHS com $h^* \geq h_d$ vértices, é certo que d incorporações independentes de testemunhas (cujos parâmetros de parada sejam maiores ou iguais a h^*) são suficientes para encontrar um CHS com probabilidade p . Essa cota necessária de incorporações é garantida exatamente pelas d primeiras incorporações, todas com parâmetro de parada maior ou igual àquele da d -ésima incorporação, que é $h_{d-1} \geq h^*$. \square

4.2.2 Análise de complexidade

A t -ésima iteração do laço principal do algoritmo de Monte Carlo que propomos consome tempo $O(nh_t)$. Para chegarmos a sua complexidade de tempo global, precisamos calcular

$$\sum_{t=1}^{t'} O(nh_{t-1}),$$

onde t' é o número de iterações, no pior caso.

O valor de h_t , obtido ao final da t -ésima iteração, é dado pela equação 4.2.

Para calcular t' , substituímos $h_{t'}$ por 2, o que nos leva a

$$\left(1 - \frac{2}{n(n-1)}\right)^{t'} = 1 - p, \text{ chegando finalmente a}$$

$$t' = \frac{\ln(1-p)}{\ln\left(1 - \frac{2}{n(n-1)}\right)}.$$

Para $0 < x < 1$, é sabido que

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \dots$$

Conseqüentemente,

$$t' = \frac{\ln(1-p)}{-\frac{2}{n(n-1)} - \frac{1}{\Theta(n^4)}} = \Theta(n^2).$$

Agora, mostraremos que $h(h-1)/n(n-1) \geq h^2/2n^2$. Este resultado será útil para simplificar alguns cálculos vindouros. Temos

$$\frac{n}{n-1} \cdot \frac{h-1}{h} \cdot \frac{h^2}{n^2} = \frac{h(h-1)}{n(n-1)}, \text{ e}$$

$$\frac{h-1}{h} \cdot \frac{h^2}{n^2} \leq \frac{h(h-1)}{n(n-1)}.$$

Uma vez que $h \geq 2$,

$$\frac{h^2}{2n^2} \leq \frac{h(h-1)}{n(n-1)}.$$

Para que possamos calcular a complexidade de tempo total, substituímos $h(h-1)/n(n-1)$ por $h^2/2n^2$ e p_t pelo valor fixo p na equação 4.1, chegando a

$$\left(1 - \frac{h^2}{2n^2}\right)^t \geq 1 - p,$$

$$\frac{h^2}{2n^2} \leq 1 - (1 - p)^{1/t}, \text{ e}$$

$$h \leq \Theta(n) \sqrt{1 - (1 - p)^{1/t}}.$$

Sabe-se que

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots.$$

Em conseqüência, para $x > 1$,

$$e^{1/x} = 1 + 1/\Theta(x).$$

Usando essa aproximação, teremos

$$h \leq \Theta(n) \sqrt{1 - (1 + 1/\Theta(t))} = \Theta(n)/\Theta(\sqrt{t}).$$

A complexidade total do algoritmo é

$$\sum_{t=1}^{\Theta(n^2)} O(nh_t) = \sum_{t=1}^{\Theta(n^2)} \frac{O(n^2)}{O(\sqrt{t})} = O(n^2) \sum_{t=1}^{\Theta(n^2)} 1/O(\sqrt{t}).$$

Usando cálculo elementar, temos

$$O(n^2) \sum_{t=1}^{\Theta(n^2)} 1/O(\sqrt{t}) = O(n^3),$$

que vem a ser finalmente a complexidade temporal deste algoritmo de Monte Carlo para o PSCH.

4.3 Sexto algoritmo: Série Harmônica

De volta a algoritmos determinísticos, apresentamos nesta seção o algoritmo da *Série Harmônica* (vide Figura 4.4), assim chamado em função de sua análise de complexidade peculiar.

A idéia central do algoritmo que será aqui apresentado é basicamente a mesma dos algoritmos IE da Seção 2.4 e SB da Seção 4.1: submeter cada um dos pares de vértices $\{x, y\} \subset V$ da entrada do problema à incorporação de testemunhas de forma a descobrir se estão ou não contidos em algum CHS daquele par de grafos.

Outra característica comum entre o algoritmo dos Subconjuntos Balanceados e o da Série Harmônica que se segue é a capacidade que ambos têm de se beneficiar, num dado momento, de incorporações de testemunhas mal-sucedidas que já tenham acontecido anteriormente, diferentemente do que acontecia com algoritmos anteriores como o Incorporações Exaustivas e o Duas Fases, onde as incorporações eram totalmente independentes umas das outras.

Assim é que o algoritmo da Série Harmônica estabelece, também, uma ordem especial para a investigação dos pares de vértices, de forma que o conhecimento acumulado ao longo das incorporações de testemunhas passadas pode ser usado para acelerar incorporações futuras.

Definimos a *distância n -circular* $dc_n(x, y)$ entre dois números naturais $x, y \in \{1, \dots, n\}$ como o número de arestas no caminho mínimo do vértice v_x ao vértice v_y em um ciclo ordenado $v_1, v_2, \dots, v_n, v_1$ de tamanho n . (Por exemplo, $dc_8(1, 4) = 3$, $dc_8(1, 7) = 2$). A distância n -circular entre x e y pode ser obtida facilmente pela expressão abaixo:

$$dc_n(x, y) = \min\{|x - y|, n - |x - y|\}. \quad (4.4)$$

Rotulando todos os n vértices de um conjunto como v_i , com i variando de 1 a n , diremos que dois vértices v_i e v_j são k -índice-separados se, e somente se, $dc_n(i, j) = k$. Claramente, dois vértices quaisquer podem ser no máximo $\lfloor n/2 \rfloor$ -índice-separados.

O algoritmo Série Harmônica procede da seguinte maneira: os $n(n-1)/2$ pares de vértices da entrada serão submetidos à incorporação de testemunhas em $\lfloor n/2 \rfloor$ turnos com n pares cada. No primeiro turno, são submetidos todos os pares de vértices 1-índice-separados ($\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_n, v_1\}$); no segundo, terão vez os pares 2-índice-separados ($\{v_1, v_3\}, \{v_2, v_4\}, \dots, \{v_n, v_2\}$); e assim por diante, com cada t -ésimo turno sempre submetendo à incorporação de testemunhas todos os n pares de vértices que são t -índice-separados.

Ocorre que, quando as n execuções do procedimento de incorporação de testemunhas do t -ésimo turno forem acontecer, já se terá tido conhecimento de que a instância de entrada não admite quaisquer CHSs que contenham algum par de vértices $\{v_i, v_j\}$ tal que $dc_n(i, j) < t$ (uma vez que tais pares já terão todos sido submetidos a incorporação de testemunhas em turnos anteriores). Essa informação estabelece um limite para o tamanho de possíveis CHS's daquela instância em $\lfloor n/t \rfloor$. Em consequência, todas as incorporações de testemunhas deste t -ésimo turno não precisarão continuar

Algoritmo 6: Série Harmônica $(G_1(V, E_1), G_2(V, E_2))$

1. **para** $t = 1$ **até** $\lfloor n/2 \rfloor$ **faça**
 - 1.1. **para cada** $\{v_i, v_j\} \subset V$ tal que $dc_n(i, j) = t$ **faça**
 - 1.1.1. **se** `Inc_Incompleta_Test` $(G_1, G_2, \{v_i, v_j\}, \lfloor n/t \rfloor) = \text{sim}$
retorne sim
 2. **retorne não**
-

Figura 4.4: O algoritmo Série Harmônica

após o tamanho do conjunto candidato ter excedido este limite de $\lfloor n/t \rfloor$ vértices, donde a versão incompleta do procedimento de incorporação será utilizada, com parâmetro de parada $k = \lfloor n/t \rfloor$.

4.3.1 Prova de corretude

Teorema 4.4. *O algoritmo Série Harmônica resolve corretamente o PSCH.*

Demonstração. Uma vez mais, temos que, se o algoritmo retorna *sim*, então foi encontrado um conjunto $H \subset V$, com $2 \leq |H| < |V|$, tal que o conjunto testemunha de H é vazio. Isto implica ser H um CHS da entrada do problema.

Suponhamos, agora, que a entrada possua de fato um CHS H . Seja $d = \min\{cd_n(i, j) \mid v_i, v_j \in H\}$ a menor dentre as distâncias n -circulares que há entre dois vértices quaisquer de H e sejam $v_r, v_s \in H$ dois vértices d -índice-separados. Suponhamos ainda que H possui mais do que $\lfloor n/d \rfloor$ vértices. Então, pelo princípio da casa do pombo, deve haver dois vértices v_x e v_y tais que $cd_n(x, y) < d$. Isto é uma contradição, pois d é mínimo. Daí ser

a cardinalidade de H limitada superiormente por $\lfloor n/d \rfloor$. Se o algoritmo ainda não houver localizado algum CHS e portanto parado com uma resposta *sim*, então no momento em que o par $\{v_r, v_s\}$ for tomado como candidato inicial do procedimento de incorporação de testemunhas (o que irá forçosamente acontecer durante o d -ésimo turno de incorporações), esta incorporação (que será incompleta com parâmetro de parada $k = \lfloor n/d \rfloor$) estará destinada a encontrar um CHS contendo v_r e v_s (pois, por hipótese, existe um CHS, H , que contém k vértices ou menos). E o algoritmo responderia, portanto, o esperado *sim*. \square

4.3.2 Análise de complexidade

No pior caso, o algoritmo terá $\lfloor n/2 \rfloor$ turnos, cada qual com n incorporações de testemunhas. Durante o t -ésimo turno, o parâmetro de parada k de cada incorporação é $\lfloor n/t \rfloor$. Como a complexidade de tempo de uma incorporação incompleta com parâmetro de parada k é $O(nk)$, a complexidade de todo o t -ésimo turno será $n \cdot O(n \lfloor n/t \rfloor) = O(n^3/t)$.

Assim, a complexidade global do algoritmo da Série Harmônica pode ser formulada como se segue (justificando seu nome):

$$\sum_{t=1}^{\lfloor n/2 \rfloor} O\left(\frac{n^3}{t}\right) = O\left(n^3 \sum_{t=1}^{O(n)} O\left(\frac{1}{t}\right)\right) = O(n^3 \log n).$$

Capítulo 5

Inimigos

5.1 Vértices inimigos e grafos de inimizade

O próximo algoritmo aqui proposto, que será apresentado na Seção 5.2, é mais engenhoso, por assim dizer, do que o algoritmo Série Harmônica (e, na prática, algo mais rápido, ainda que ambos compartilhem a mesma complexidade de tempo assintótica).

Antes de dirigirmos a ele nossa atenção, introduziremos a definição de *vértices inimigos* (ou simplesmente *inimigos*), conceito este que servirá de base para o algoritmo que se segue.

Sejam $G_1(V, E_1), G_2(V, E_2)$ os grafos da entrada do PSCH. Dois vértices x, y são ditos *inimigos* um do outro, com relação a (G_1, G_2) , se, e somente se, nenhum CHS existir, para aquela entrada, contendo ambos x e y .

Definimos os *grafos de inimizade* de $G_1(V, E_1), G_2(V, E_2)$ como grafos $G_N(V, E_N)$ que possuem o mesmo conjunto de vértices que G_1 e G_2 e nos quais uma aresta (x, y) existe *somente se* x e y são inimigos com relação àquele par de grafos.

Um grafo de inimizadas para um par (G_1, G_2) é dito *trivial* quando ele é vazio, ou seja, sem arestas (nesse caso, nada revela sobre a existência de vértices inimigos para aquele par de grafos). É dito, por outro lado, *final*, se apresenta aresta (x, y) entre *todos os pares* de vértices inimigos $x, y \in V$. Um grafo de inimizadas final G_N claramente resolve o PSCH, uma vez que quaisquer dois vértices que sejam *não*-adjacentes em G_N não são inimigos, ou seja, há pelo menos um CHS de (G_1, G_2) que contém a ambos. Em outras palavras, o PSCH terá resposta *não* se, e somente se, o grafo de inimizadas final para sua instância de entrada for *completo* (i.e. cada vértice é adjacente a todos os outros).

Grafos de inimizado para uma instância do PSCH são, portanto, quaisquer grafos-sanduíche dos grafos de inimizado trivial e final para aquela instância, ou seja, são grafos que descrevem *subconjuntos* das relações de inimizado entre os pares de vértice de uma instância do PSCH. O Lema 5.1, dado a seguir, destaca a utilidade dos grafos de inimizado para a solução do PSCH.

(Lembramos que, em um grafo $G(V, E)$, um *conjunto independente* é um subconjunto $S \subseteq V$ que não contém qualquer par de vértices adjacentes em G .)

Lema 5.1. *Dado um grafo de inimizadas G_N de (G_1, G_2) , a cardinalidade de qualquer CHS de (G_1, G_2) é no máximo igual à cardinalidade do maior conjunto independente de G_N .*

Demonstração. Seja d o tamanho do conjunto independente máximo do grafo de inimizado G_N dado. Suponhamos, por absurdo, que o par (G_1, G_2) admita um CHS H que contenha mais do que d vértices. Sendo assim, o subgrafo de G_N induzido por H não pode ser totalmente desconexo (ou os vértices de H constituiriam um conjunto independente de G_N com tamanho superior a

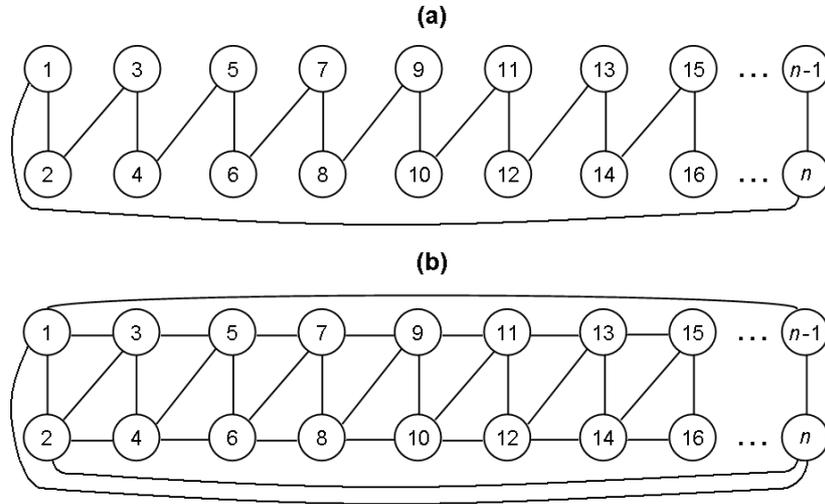


Figura 5.1: Exemplos de grafos de inimidade durante uma execução do algoritmo Série Harmônica

d), de forma que há pelo menos dois vértices $x, y \in H$ que são conectados por uma aresta em G_N . Mas isso é possível somente se x e y são inimigos, o que é uma contradição, uma vez que há um CHS $— H$, por hipótese — que contém ambos os vértices x e y . \square

Para ilustrar a aplicabilidade dos grafos de inimidade, voltaremos rapidamente ao algoritmo da Série Harmônica, recém-estudado na Seção 4.3. Pode-se considerar como sendo a principal diferença entre o algoritmo Série Harmônica e o algoritmo das Incorporações Exaustivas (o primeiro algoritmo publicado para o PSCH) o fato de que, ao passo em que o IE negligencia a inimidade entre dois vértices x e y que é trazida à tona pela incorporação mal-sucedida do par $\{x, y\}$, o SH usa esse conhecimento para restringir a extensão de incorporações futuras.

Em outras palavras, o que, implicitamente, todos os algoritmos determinísticos baseados no procedimento de incorporação de testemunhas fazem é, a partir de um grafo de inimizades trivial, adicionar-lhe novas arestas (entre vértices que se revelem inimigos) até que: (i) algum CHS seja localizado com sucesso (respondendo *sim*); ou (ii) um grafo de inimizades completo for enfim obtido (respondendo *não*). O algoritmo IE ignora os grafos de inimizade. O SH, embora não disponha realmente de uma estrutura de dados para armazenamento de grafos de inimizade, utiliza aquele mesmo conhecimento que lhes é subjacente para economizar precioso tempo.

Na Figura 5.1, (a) e (b) mostram os grafos de inimizade que se teria obtido ao final, respectivamente, do primeiro e do segundo turno do algoritmo da Série Harmônica para uma entrada qualquer (sob a hipótese, evidentemente, de que o algoritmo não tenha encontrado, até então, CHS algum).

No algoritmo da Série Harmônica, n relações de inimizade são descobertas a cada turno de incorporações de testemunhas (como se foram n novas arestas adicionadas a G_N). Dessa forma, o número de vértices contidos em algum possível CHS daquela instância será a cada turno limitado pelo tamanho do conjunto independente máximo no grafo de inimizades G_N a que se chegou ao fim do turno anterior — limite este que, como vimos, é fácil de ser calculado, *nesse caso particular*, dada a ordem em que as arestas são adicionadas a G_N .

O algoritmo Cliques Crescentes, apresentado na Seção 5.2 a seguir, difere do algoritmo Série Harmônica por ser capaz de aplicar não apenas uma ordem de submissão dos pares de vértices que o torne capaz de determinar continuamente um limite superior razoavelmente justo para o tamanho de possíveis CHSs, mas sim a *melhor* ordem de submissão de pares, isto é, aquela que minimiza o tamanho dos conjuntos independentes máximos de G_N ao longo das sucessivas adições de arestas.

5.2 Sétimo algoritmo: Cliques Crescentes

Em todos os algoritmos para o PSCH que se baseiam na incorporação de testemunhas, quanto mais justo for o limite superior conhecido para o tamanho de possíveis CHSs da instância de entrada, num dado momento, mais rapidamente será permitida a parada dos procedimentos de incorporação que forem executados dali em diante. Com vistas à otimização de tais algoritmos, é preciso responder a seguinte pergunta: *Qual a melhor ordem de submissão dos pares de vértices ao procedimento de incorporação de testemunhas?* Dado que incorporações de testemunhas mal-sucedidas revelam relações de inimidade (ou, em outras palavras, adicionam uma aresta ao grafo de inimidades subjacente G_N — ainda que este não esteja sendo de fato armazenado), e que tais relações, segundo o Lema 5.1, prestam-se à reduzir o limite superior para a cardinalidade de CHS's que se precisa buscar, a questão proposta acima se lê: *Qual a ordem de submissão de pares que permite que, a cada momento, o tamanho do conjunto independente máximo de G_N seja o menor possível?*

A resposta é simples e nos leva diretamente ao algoritmo ilustrado na Figura 5.2: aquela em que são construídas cliques disjuntas de tamanho crescente, de forma que o tamanho da *cobertura de cliques mínima* de G_N é o menor possível. (Lembramos que uma *cobertura de cliques* de um grafo é uma coleção de cliques tal que cada vértice do grafo pertence a exatamente uma das cliques da coleção. É claro que o tamanho da cobertura de cliques mínima é um limite superior para o tamanho do conjunto independente máximo.

O algoritmo das *Cliques Crescentes* (CC), tanto quanto o algoritmo SH já apresentado, submete à incorporação de testemunhas cada um dos pares de vértices da entrada do problema. Neste novo algoritmo, os procedimentos de incorporação são ainda organizados em turnos — mas de modo diverso.

O primeiro turno de incorporações de testemunhas do algoritmo CC, que parte de um grafo de inimizadas trival (vazio) G_N , encarrega-se de juntar, duas a duas, as cliques de G_N (que têm, àquele momento, apenas 1 vértice cada), de forma a constituir $\lfloor n/2 \rfloor$ novas cliques de tamanho 2. (Possivelmente, restará uma clique de tamanho 1 — caso n seja ímpar.) É fácil ver que o tamanho do conjunto independente máximo de G_N é, agora, $\lfloor n/2 \rfloor$. A Figura 5.3(a) ilustra o grafo de inimizadas G_N após todas as incorporações de testemunhas do primeiro turno do algoritmo CC terem sido executadas.

O segundo turno, por sua vez, executará incorporações de testemunhas de forma a unir as cliques correntes de tamanho 2, duas a duas, formando assim novas cliques de tamanho 4. (É evidente que, se o número n de vértices não for múltiplo de 4, haverá uma — e apenas uma — clique de tamanho menor que 4.)

Os turnos seguintes procederão da mesma forma, realizando incorporações de testemunhas entre pares de vértices de cliques distintas de tal modo que as k cliques de tamanho n/k provenientes do turno anterior sejam transformadas em $\lfloor k/2 \rfloor$ cliques de tamanho $2n/k$ e no máximo uma clique de tamanho inferior.

As Figuras 5.3(b) e 5.3(c) mostram, respectivamente, o grafo de inimizadas G_N ao final do segundo e terceiro turnos de incorporações de testemunhas realizados pelo algoritmo das Cliques Crescentes.

Esse processo de construção de cliques continua até que, durante uma incorporação de testemunhas, o algoritmo encontre algum CHS da instância de entrada (retornando *sim*), ou o grafo de inimizadas passe a ser completo, ou seja, constituído por uma única clique contendo todos os seus vértices (retornando *não*).

Algoritmo 7: Cliques Crescentes $(G_1(V, E_1), G_2(V, E_2))$

```
1.       $C \leftarrow \{\{v_i\} \mid v_i \in V\}$  // inicializa a cobertura com singletons
2.      enquanto  $|C| > 1$  faça
2.1.    indexe todas as cliques em  $C$  de 1 a  $|C|$ 
2.2.    para cada clique  $C_j \in C$  tal que  $j$  é par faça
2.2.1.  para cada  $\{x, y\}$  tal que  $x \in C_j, y \in C_{j-1}$  faça
2.2.1.1. se  $\text{Inc\_Incompleta\_Test}(G_1, G_2, \{x, y\}, |C|) = \text{sim}$ 
          retorne sim
2.2.2.   $C \leftarrow (C \cup \{C_j \cup C_{j-1}\}) \setminus \{C_j, C_{j-1}\}$  // une  $C_j$  e  $C_{j-1}$ 
3.      retorne não
```

Figura 5.2: O algoritmo das Cliques Crescentes para o PSCH

O fato crucial, aqui, é que as incorporações de testemunhas do t -ésimo turno não precisarão ir adiante para candidatos com mais do que $\lceil n/2^{t-1} \rceil$ vértices, pois este, sendo o tamanho da cobertura de cliques mínima, é um limite superior para o tamanho de conjuntos independentes em G_N , o que limita, por conseguinte, o tamanho máximo de quaisquer CHSs daquela instância de entrada.

5.2.1 Prova de corretude

Teorema 5.2. *O algoritmo das Cliques Crescentes resolve corretamente o PSCH.*

Demonstração. Como nos algoritmos anteriores, uma resposta *sim* é somente dada se um CHS válido foi de fato encontrado.

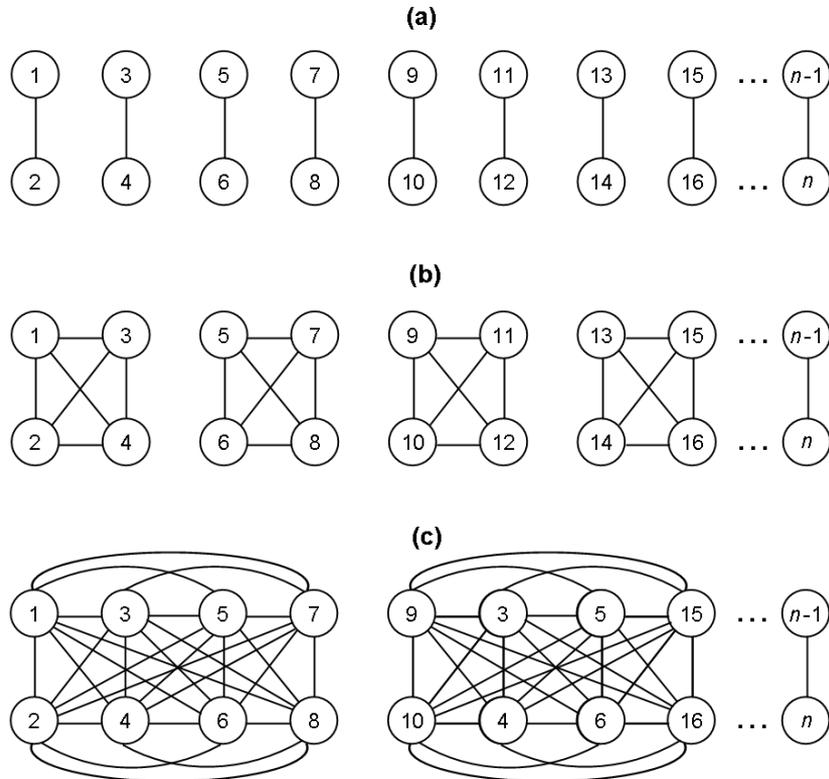


Figura 5.3: Grafos de inimizados durante execução do algoritmo CC

Suponhamos, agora, que a instância de entrada possui um CHS H com tamanho h . Para mostrar que, neste caso, o algoritmo responderia corretamente *sim*, é suficiente provarmos que: (i) o algoritmo não pode responder *não* antes que cada um dos pares de vértices da entrada do problema tenham sido submetidos à incorporação de testemunhas; e (ii) quando um par qualquer $\{x, y\} \subseteq H$ for submetido à incorporação (incompleta) de testemunhas, aquele procedimento não parará sem que tenha encontrado com sucesso um CHS.

Para provarmos (i), lembramos que o algoritmo só pode parar com resposta *não* caso a condição de seu laço principal (linha 2, na Figura 5.2) não seja mais atendida, o que significa que uma única clique existe, cobrindo todos os vértices de G_N . Isto significa que arestas foram adicionadas entre todos os pares de vértices, e sabemos que uma adição de aresta advém exclusivamente da descoberta de inimidade, isto é, de uma incorporação mal-sucedida.

Para provarmos (ii), iremos supor, por absurdo, que a incorporação incompleta de testemunhas que começou com candidato inicial $\{x, y\} \subseteq H$ parou sem ter encontrado CHS algum. Como ao seu parâmetro de parada fora atribuído (linha 2.2.1.1) a cardinalidade da cobertura de cliques corrente C (ou seja, o número de cliques, maximais por construção, no grafo de inimidades corrente) e H , que contém h vértices, não foi localizado, infere-se que o número de cliques maximais em G_N àquele momento era menor do que h . Mas isto é uma contradição, pois o Lema 5.1 garante que o tamanho do conjunto independente máximo de um grafo de inimidades (que não pode ser maior do que o número de cliques numa cobertura) é um limite superior para o tamanho de qualquer CHS daquela instância. \square

5.2.2 Análise de complexidade

Como cada turno de incorporações de testemunhas divide o tamanho da cobertura de cliques C por 2, o algoritmo das Cliques Crescentes pára, no pior caso, após $O(\log n)$ turnos. Este é, portanto, o número máximo de iterações do laço principal do algoritmo (linha 2).

A complexidade de tempo de cada turno (linhas 2.1 a 2.2.2) é dada pelo número de incorporações de testemunhas que são executadas durante aquele turno multiplicado pela complexidade de cada uma das incorporações daquele turno. No t -ésimo turno, o número de incorporações é $O(2^{t-2}n)$ e

o parâmetro de parada é $n/2^{t-1}$. Como a complexidade de tempo das incorporações incompletas de testemunhas com parâmetro de parada k são $O(nk)$, temos que cada incorporação do t -ésimo turno executa em tempo $O(n \cdot n/2^{t-1}) = O(n^2/2^{t-1})$. Assim, a complexidade de tempo de todo o t -ésimo turno é $O(2^{t-2}n) \cdot O(n^2/2^{t-1}) = O(n^3/2) = O(n^3)$, para todo t .

Chega-se, dessa forma, à complexidade global do algoritmo Cliques Crescentes:

$$\sum_{t=1}^{O(\log n)} O(n^3) = O(n^3 \log n).$$

Outra maneira de utilizarmos proveitosamente os grafos de inimizados é dada na Seção 5.3 seguinte, onde será apresentado um algoritmo randomizado de Las Vegas que resolve o PSCH em tempo esperado $O(n^3)$ com o emprego de uma estrutura de dados que realmente constrói, dinamicamente, o grafo de inimizados associado à entrada em questão.

5.3 Oitavo algoritmo: Las Vegas

Algoritmos randomizados de Las Vegas são tais que dão a resposta correta em cem por cento dos casos, mas cujo tempo de execução é uma variável aleatória, de alguma forma dependente de escolhas randômicas. A eficiência de tais algoritmos é, portanto, avaliada em termos do *valor esperado* de sua complexidade assintótica.

O algoritmo de Las Vegas que ora apresentamos pertence também à família de algoritmos para o PSCH baseados no procedimento de incorporação de testemunhas. Aqui, mais uma vez, teremos todos os pares de vértices da instância de entrada submetidos àquele procedimento, e novamente as relações de inimizado descobertas por incorporações mal-sucedidas terão papel fundamental.

Ao abordarmos os algoritmos Série Harmônica e Cliques Crescentes, nas Seções 4.3 e 5.2, vimos que a performance daqueles algoritmos era absolutamente dependente da ordem adotada para a submissão dos pares de vértices da entrada ao procedimento de incorporação de testemunhas. Ainda que a melhor das seqüências seja seguida, no entanto, vimos que, no pior caso, o tempo gasto por tais algoritmos seria ainda $O(n^3 \log n)$. Veremos agora ser possível baixar o tempo esperado de execução para uma entrada qualquer, com o emprego de uma ordem randomizada de submissão dos pares.

Nos algoritmos SH e CC, o grafo de inimizadas subjacente apenas existia conceitualmente (nenhuma aresta ou qualquer informação sobre as relações de inimizado era armazenada em lugar algum), como um meio de permitir o cálculo de limites para a extensão de incorporações incompletas. No algoritmo de Las Vegas que vamos propor, o grafo de inimizadas *de fato existe*. Ao invés de apenas obter limites superiores decrescentes para o tamanho de CHS's cuja existência se pretende investigar (constituindo parâmetros de parada adequados para procedimentos de incorporação incompleta de testemunhas), o grafo de inimizadas G_N será agora realmente construído (isto é, sua matriz de adjacências será armazenada e atualizada a cada nova aresta inserida), oferecendo igualmente meios de se interromper antecipadamente incorporações de testemunhas que não tenham mais chance de encontrar qualquer CHS. Essa construção é tal que a escolha de cada nova aresta a ser inserida em G_N (isto é, do próximo par de vértices que será submetido à incorporação de testemunhas) será sempre feita de forma randômica, como logo veremos.

Uma variação do procedimento de incorporação incompleta de testemunhas, a que nos referimos como incorporação de testemunhas *restrita por inimizadas*, é apresentada na Figura 5.4.

Procedimento: Inc_Test_Rest_Inimizades $(G_1(V, E_1), G_2(V, E_2), H_1, G_N(V, E_N))$

1. $H \leftarrow H_1$
 2. **enquanto** $|H| < |V|$ **faça**
 - 2.1. **se** $B(H) = \emptyset$
retorne *sim*
 - 2.2. **senão**
 - para cada** par $\{x, y \mid x \in B(H), y \in H \cup B(H)\}$ **faça**
 - 2.2.1 **se** $(x, y) \in E_N$
retorne *não*
 - 2.2.2. $H \leftarrow H \cup B(H)$
 3. **retorne** *não*
-

Figura 5.4: Incorporação de testemunhas restrita por inimizades

A incorporação de testemunhas restrita por inimizades é também incompleta, no sentido de que pode, da mesma forma, dar resposta *não* muito antes de o candidato ter englobado todos os vértices dos grafos da entrada do PSCH. Ao invés de ser informado um parâmetro de parada k que limitaria o tamanho do candidato, a incorporação restrita será interrompida tão logo o candidato contenha dois vértices que sejam sabidamente inimigos. (Note que, agora, as relações de inimizade de que se tem conhecimento são explicitamente representadas pelas arestas do grafo G_N , que é passado como parâmetro.)

O algoritmo de Las Vegas que apresentamos (vide Figura 5.5) pode ser visto como similar ao algoritmo IE, com a única diferença no fato de que

Algoritmo 8: Las Vegas ($G_1(V, E_1), G_2(V, E_2)$)

1. inicialize $G_N(V, E_N)$ com um conjunto de arestas vazio $E_N = \emptyset$
 2. escolha uma ordenação randômica Γ dos pares $\{x, y\} \subset V$
 3. **para cada** par de vértices $\{x, y\}$ em Γ **faça**
 - 3.1. **se** $\text{Inc_Test_Restr_Inimizades}(G_1, G_2, \{x, y\}, G_N) = \text{sim}$
retorne sim
 - 3.2. **senão** $E_N \leftarrow E_N \cup \{(x, y)\}$
 4. **retorne não**
-

Figura 5.5: Algoritmo de Las Vegas para o PSCH

incorporações de testemunhas *restritas por inimizades* substituirão as incorporações completas daquele algoritmo. Mostraremos que, fazendo-se com que a ordem de submissão dos pares à incorporação restrita seja obtida randomicamente, consegue-se uma performance esperada eficiente para qualquer entrada.

5.3.1 Prova de corretude

Teorema 5.3. *O algoritmo apresentado na Figure 5.5 é um algoritmo de Las Vegas que resolve corretamente o PSCH.*

Demonstração. Por ser baseado no procedimento de incorporação de testemunhas, é redundante reiterarmos que uma resposta *sim* é apenas possível se um CHS de fato existe. Por outro lado, se o algoritmo responde *não*, isto se deve ao fato de ter submetido todos os pares de vértices a incorporações de testemunhas (restritas por inimizades), nenhuma das quais tendo encontrado

qualquer CHS. Cada uma das incorporações restritas, por sua vez, apenas retorna *não* após um par de vértices inimigos estar contido no candidato corrente, o que, pela definição de inimigos, atesta a não-existência de CHS's contendo aquele candidato. \square

5.3.2 Análise de complexidade

O número de incorporações de testemunhas executadas pelo algoritmo de Las Vegas é, no pior caso, $n(n-1)/2 = O(n^2)$.

A complexidade de tempo do t -ésimo procedimento de incorporação de testemunhas restrita por inimigades é $O(nr_t)$, onde r_t é o número de vértices no conjunto candidato H no momento em que a incorporação é interrompida, ou seja, no momento em que aparecem pela primeira vez dois inimigos entre seus elementos. Queremos obter uma expressão para o valor esperado para r_t .

No momento em que se inicia a t -ésima execução do procedimento de incorporação de testemunhas restrito, $t-1$ pares de vértices inimigos já terão sido revelados pelas incorporações anteriores, o que significa que o grafo de inimigades $G_N(V, E_N)$, àquele momento, possuirá exatamente $t-1$ arestas. Como aquelas $t-1$ arestas foram adicionadas entre pares de vértices *escolhidos de maneira randômica*, uma aresta entre dois vértices x, y que façam parte do candidato corrente H existirá, então, em E_N com probabilidade $p_t = 2(t-1)/n(n-1)$. O número q_t de *pares* de vértices que estarão contidos no candidato H , no momento em que o primeiro par de inimigos dele fizer parte, será, portanto, uma variável aleatória geométrica com probabilidade de sucesso $p_t = O(t/n^2)$. É bem sabido que tais variáveis têm como valor esperado o inverso de sua probabilidade de sucesso, donde $E[q_t] = 1/p_t = O(n^2/t)$.

Como o número r_t de *vértices* em H que constituem os q_t *pares de vértices* em H é $r_t = O(\sqrt{q_t})$, o valor esperado que desejamos é $E[r_t] = O(\sqrt{E[q_t]}) = O(n/\sqrt{t})$.

O tempo esperado para a execução do t -ésimo procedimento de incorporação restrita é, assim, $O(nr_t) = O(n^2/\sqrt{t})$, o que nos dá a seguinte expressão para a complexidade esperada global do algoritmo:

$$\sum_{t=1}^{O(n^2)} O\left(\frac{n^2}{\sqrt{t}}\right) = O(n^2) \sum_{t=1}^{O(n^2)} O\left(\frac{1}{\sqrt{t}}\right) = O(n^3).$$

5.4 Nono algoritmo: Preenchimento Acelerado

O algoritmo que apresentaremos agora, a que nos referimos como *Preenchimento Acelerado*, consegue agregar três importantes resultados já implementados por algoritmos anteriores: (i) o contínuo decréscimo na extensão de cada um dos sucessivos procedimentos de incorporação de testemunhas (como no algoritmo das Cliques Crescentes); (ii) um número reduzido de candidatos que precisam ser submetidos a tais procedimentos (como no algoritmo das Duas Fases); e (iii) a capacidade de interromper prematuramente uma incorporação tão logo um par de inimigos seja adicionado ao conjunto candidato (como no algoritmo de Las Vegas), e não apenas quando o tamanho do candidato houver ultrapassado um determinado limite teórico.

A justificativa para o algoritmo Preenchimento Acelerado encontra-se no seguinte lema:

Lema 5.4. *Sejam $G_1(V, E_1), G_2(V, E_2)$ os grafos de entrada para o PSCH, e sejam $u, v \in V$ dois vértices inimigos. Para todo $\{x, y\} \subseteq V$, se existe um caminho, no grafo-testemunha $G_T(V_T, E_T)$, do nó $[x, y]$ ao nó $[u, v]$, então x e y são também inimigos entre si.*

Demonstração. Dito de outra forma, o Lema 5.4 proclama que, se não há CHS algum que contenha $\{u, v\}$, e do nó $[x, y] \in V_T$ pode-se atingir o nó $[u, v] \in V_T$, então não haverá igualmente qualquer CHS que contenha $\{x, y\}$. Pela construção do grafo-testemunha, sabe-se que todos os arcos em E_T são da forma $([x, y] \rightarrow [x, b])$ ¹, onde b é uma testemunha de $\{x, y\}$. Vem do Teorema 2.1 o fato de que todos os CHSs que contenham $\{x, y\}$ devem também conter b . Daí, se não há CHS algum contendo $\{x, b\}$, não pode haver nenhum que contenha $\{x, y\}$. Em outras palavras, se os vértices rotuladores do nó-destino são inimigos, também o serão aqueles que rotulam o nó-origem.

É fácil ver que esse raciocínio se aplica não apenas a arcos, mas a *caminhos* (no grafo-testemunha) de qualquer tamanho. Seja $\alpha_1, \alpha_2, \dots, \alpha_k$ um caminho em G_T , onde α_i representa um nó $[u_i, v_i] \in V_T$. Se não há qualquer CHS contendo $\{u_k, v_k\}$ (isto é, os vértices rotuladores de α_k), então o raciocínio acima exposto mostra que não pode existir CHS algum contendo $\{u_{k-1}, v_{k-1}\}$, o que atesta, por sua vez, a inexistência de CHSs contendo $\{u_{k-2}, v_{k-2}\}$, e assim por diante. Isto é suficiente para mostrar os vértices rotuladores u_i, v_i de $\alpha_i (1 \leq i \leq k)$ são de fato inimigos. \square

Visando o entendimento do algoritmo Preenchimento Acelerado, voltaremos nossa atenção ao algoritmo de Las Vegas estudado na Seção 5.3.

Há um grafo de inimizades G_N , inicialmente vazio, ao qual uma aresta é adicionada cada vez que uma incorporação é executada sem sucesso (reve-

¹ou, similarmente, $([x, y] \rightarrow [y, b])$.

lando uma relação de inimizade entre dois dos vértices da entrada do problema). O armazenamento da informação a respeito dessas inimizades permite que as incorporações de testemunhas restritas sejam interrompidas tão logo um par de inimigos venha a fazer parte do candidato corrente. Assim, quanto mais arestas tenham sido adicionadas ao grafo de inimizades, tão mais prematuramente serão interrompidas futuras incorporações.

O melhoramento tornado possível pelo Lema 5.4 é a possibilidade de adicionarmos ao grafo de inimizades *mais de uma aresta* por cada incorporação mal-sucedida. Na verdade, após um par $\{u, v\} \subset V$ ter sido revelado como não-pertencente a nenhum CHS daquela instância, uma aresta pode ser adicionada a G_N *entre cada par* de vértices x, y tal que $[x, y]$ atinja $[u, v]$ no grafo-testemunha associado àquela entrada.

Começa o algoritmo com a geração do grafo-testemunha $G_T(V_T, E_T)$ de (G_1, G_2) e, em seguida, tal como no algoritmo dos Sumidouros Fortemente Conexos formulado por Tang *et al.*, são localizados seus sumidouros fortemente conexos. A cada SFC $P \subseteq G_T$ é, então, atribuído um *representante*, que pode ser qualquer nó $[u, v] \in P$. Claramente, cada um dos nós do grafo-testemunha atinge a pelo menos *um* representante de SFC.

Determina-se, então, para cada vértice $[x, y] \in V_T$, o que será chamado seu *representante atingível* $ra(x, y) = [u, v] \in V_T$, o que nada mais é que o representante de um dos SFC atingidos por $[x, y]$ (seja, digamos, $[u, v]$). A determinação dos sumidouros atingíveis é feita através de uma simples busca em profundidade feita numa cópia de G_T com todas as arestas invertidas. As raízes são os representantes de SFCs. O sumidouro atingível de cada nó será, enfim, a raiz da árvore que o contiver, na floresta de profundidade obtida.

Logo a seguir, um grafo de inimizades trivial $G_N(V, E_N = \emptyset)$ é criado.

A idéia, a partir daí, é a de submeter os pares de vértices da entrada do problema à incorporação de testemunhas, respeitando exatamente a mesma ordem de submissão que é seguida pelo algoritmo das Cliques Crescentes, de forma que sua extensão possa ser igualmente minimizada ao longo das sucessivas execuções. A única diferença é a de que, aqui, quando for o momento — segundo aquela ordem particular de submissões — de ser o par $\{x, y\} \subset V$ submetido à incorporação (com vistas à adição da aresta (x, y) a E_N), não será realmente $\{x, y\}$ o par a ser submetido à incorporação de testemunhas; em seu lugar, será feita a incorporação de testemunhas a partir de seu representante atingível $ra(x, y) = [u, v] \in G_T$. Como resultado, o algoritmo terá parado com uma resposta *sim* ou terá adicionado a E_N a aresta (u, v) e também a aresta desejada (x, y) . Se for o caso em que já exista tal aresta (u, v) , no grafo de inimizados, no momento em que o par $\{x, y\}$ (para o qual $rsa(x, y) = [u, v]$) deve ser submetido à incorporação, então a aresta (x, y) será imediatamente adicionada a E_N , sem que sequer uma incorporação tenha de ser executada — daí o nome do algoritmo.

Não há dúvidas de que uma adição de aresta em tempo $O(1)$ signifique, na prática, uma benvinda economia de tempo. É fácil ver que também a própria complexidade de tempo assintótica será beneficiada. Ora, se apenas *representantes de SFC's* são de fato submetidos à incorporação de testemunhas, então o número de incorporações será, no pior caso, $O(m)$, que é número máximo de SFCs que não são fechados por pares.

Em resumo, o algoritmo Preenchimento Acelerado pode ser entendido como: (i) um algoritmo Cliques Crescentes que execute no máximo $O(m)$ incorporações, ao invés das $O(n^2)$ que são executadas por aquele algoritmo; ou (ii) um algoritmo Duas Fases que use incorporações incompletas (restritas por inimizados).

A Figura 5.6 torna mais claro o algoritmo Preenchimento Acelerado através da exibição de seu pseudo-código.

5.4.1 Prova de corretude

Teorema 5.5. *O algoritmo Preenchimento Acelerado resolve corretamente o PSCH.*

Demonstração. A prova é idêntica à do algoritmo Cliques Crescentes (Seção 5.2), adicionado o fato, respaldado no Lema 5.4, de que dois vértices $x, y \in V$ não precisarão ser submetidos ao procedimento de incorporação (para que a inimizade entre eles seja revelada), se é verdade que já se conhece um par de inimigos $u, v \in V$ tal que $[x, y]$ atinge $[u, v]$. \square

5.4.2 Análise de complexidade

Cada incorporação incompleta de testemunhas, no algoritmo Cliques Crescentes, recebe um parâmetro de parada cujo valor é igual a um certo limite superior conhecido para o tamanho do conjunto independente máximo do grafo de inimizades subjacente, durante um determinado *turno* de incorporações.

No algoritmo Preenchimento Acelerado, incorporações restritas por inimizades são utilizadas, tal como no algoritmo Las Vegas (vide Seção 5.3), como um modo de o algoritmo se beneficiar tanto quanto possível do conhecimento que foi já adquirido, por meio das incorporações anteriores, a respeito de algumas relações de inimizades existentes entre os vértices da entrada do problema. Sua análise de complexidade, porém, é similar àquela do algoritmo CC, uma vez que, *no pior caso*, cada uma das incorporações que serão executadas pelo algoritmo PA será apenas interrompida quando

o tamanho do candidato corrente atingir o mesmo limite teórico que seria passado como parâmetro de parada para a incorporação correspondente executada pelo algoritmo CC. Portanto, dado que, no pior caso (para o algoritmo PA), cada uma das incorporações executadas por ambos os algoritmos CC e PA terá a mesma extensão, a complexidade de tempo de cada incorporação, *individualmente*, será idêntica para os dois algoritmos.

Em ambos os algoritmos (CC e PA), as incorporações de testemunhas são divididas em turnos, cada um dos quais responsável por unir cliques do grafo de inimizades, duas a duas, em cliques com tamanho dobrado. Além disso, foi visto que no t -ésimo turno, o número de incorporações é $O(2^{t-2}n)$. O que distingue os dois algoritmos é de fato o *número de turnos* que serão, no pior caso, processados.

No algoritmo CC, o número de turnos precisa ser suficiente para que, no pior caso, todos os $O(n^2)$ pares de vértices da entrada sejam submetidos à incorporação de testemunhas, o que nos estabelece o limite de $O(\log n)$ turnos. No caso do algoritmo PA, apenas $O(m)$ representantes de SFC's precisam ser submetidos à incorporação de testemunhas. Assim, podemos obter um limite superior para o número t' de turnos de incorporações:

$$\begin{aligned} \sum_{t=1}^{t'} O(2^{t-2}n) &= O(m) \\ t' &= O\left(\log \frac{m}{n}\right) = \\ &= O\left(\log \frac{m}{n}\right), \end{aligned}$$

Tal como no algoritmo CC, a complexidade de tempo do t -ésimo turno é $O(2^{t-2}n) \cdot O(n^2/2^{t-1}) = O(n^3/2) = O(n^3)$, para todo t .

Assim, a complexidade global do algoritmo Preenchimento Acelerado é:

$$\sum_{t=1}^{t'} O(n^3) = \sum_{t=1}^{O(\log \frac{m}{n})} O(n^3) = O\left(n^3 \log \frac{m}{n}\right).$$

É digno de nota o fato de que o algoritmo Preenchimento Acelerado não pode nunca demandar mais tempo computacional que o algoritmo Duas Fases (Seção 3.4), uma vez que ambos executam exatamente o mesmo número de incorporações de testemunhas, com a diferença de que as incorporações são todas completas no Duas Fases e incompletas no Preenchimento Acelerado. Um limite superior mais acurado seria, portanto, dado por $O(\min\{mM, n^3 \log \frac{m}{n}\})$, que é, no entanto, menos legível.

Algoritmo 9: Preenchimento Acelerado ($G_1(V, E_1), G_2(V, E_2)$)

1. construa o grafo-testemunha G_T de (G_1, G_2)
 2. seja G'_T uma cópia G_T com as arestas invertidas
 3. obtenha uma floresta de profundidade F para G'_T ,
com suas raízes em nós de SFC's distintos
 4. **para cada** nó $[x, y] \in G_T$ **faça**
 - 4.1. $ra(x, y) \leftarrow [u, v]$, onde $[u, v]$ é a raiz da árvore
à qual pertence $[x, y]$, em F
 5. inicialize $G_N(V, E_N)$ com um conjunto de arestas vazias $E_N = \emptyset$
 6. $C \leftarrow \{\{v_i\} \mid v_i \in V\}$ // inicializa a cobertura de cliques
 7. **enquanto** $|C| > 1$
 - 7.1. indexe todas as cliques em C de 1 a $|C|$
 - 7.2. **para cada** clique $C_j \in C$ tal que j é par **faça**
 - 7.2.1. **para cada** par $\{x, y\}$ tal que $x \in C_j, y \in C_{j-1}$ **faça**
 - 7.2.1.1. seja $[u, v]$ o representante atingível $ra(x, y)$
 - 7.2.1.2. **se** $(u, v) \in E_N$
vá para 7.2.1.5. // preenchimento acelerado
 - 7.2.1.3. **se** $\text{Inc_Test_Restr_Inimizades}(G_1, G_2, \{u, v\}, G_N) = \text{sim}$
retorne sim
 - 7.2.1.4. $E_N \leftarrow E_N \cup \{(u, v)\}$
 - 7.2.1.5. $E_N \leftarrow E_N \cup \{(x, y)\}$
 - 7.2.2. $C \leftarrow (C \cup \{C_j \cup C_{j-1}\}) \setminus \{C_j, C_{j-1}\}$ // une C_j a C_{j-1}
 8. **retorne não**
-

Figura 5.6: O algoritmo do Preenchimento Acelerado

Capítulo 6

Compleição de Pares

Ao longo do texto, até aqui, a idéia do grafo-testemunha, criada por Tang *et al.* [18] e apresentada no Capítulo 3, havia sido aplicada três vezes:

- sem sucesso, no algoritmo Sumidouros Fortemente Conexos;
- com algum sucesso, no algoritmo Duas Fases, embora a complexidade de tempo $O(mM)$ daquele algoritmo signifique o mesmo limite $O(n^4)$ do algoritmo primeiro, de Cerioli *et al.*, se expressarmos o tempo como função apenas do número de vértices da entrada;
- com sucesso também discreto, ao ser utilizada para transformar o algoritmo Clique Crescentes, de tempo $O(n^3 \log n)$, no algoritmo Preenchimento Acelerado, de tempo $O(n^3 \log \frac{m}{n})$, ligeiramente melhor.

Neste capítulo, finalmente, o grafo-testemunha será utilizado com ganho significativo de eficiência na solução de nosso problema. Apresentamos o algoritmo determinístico mais rápido conhecido, até o momento, para o PSCH. Chama-se algoritmo da *Compleição de Pares*.

Localize_Sumidouros_Atingíveis ($G_T(V_T, E_T)$)

1. **para cada** nó $[x, y] \in V_T$ **faça**
 - 1.1. $sfc(x, y) \leftarrow \text{indefinido}$
 2. **para cada** SFC S **faça**
 - 2.1. **seja** $[u, v]$ um nó qualquer de S
 - 2.2. $sfc(u, v) \leftarrow S$
 - 2.3. **seja** R uma lista contendo inicialmente apenas $[u, v]$
 - 2.4. **enquanto** R é não-vazia **faça**
 - 2.4.1. **seja** $[c, d]$ o primeiro elemento de R
 - 2.4.2. **para cada** arco $([x, y] \rightarrow [c, d]) \in E_T$ **faça**
 - 2.4.2.1. **se** $sfc(x, y) = \text{indefinido}$
 - 2.4.2.1.1. $\text{sumidouro}(x, y) \leftarrow S$; coloque $[x, y]$ em R
 - 2.4.2.2. **senão se** $sfc(x, y) \neq S$ e $sfc(x, y) \neq \text{vários}$
 - 2.4.2.2.1. $sfc(x, y) \leftarrow \text{vários}$; coloque $[x, y]$ em R
 - 2.4.3. remova $[c, d]$ de R

Figura 6.1: Rotina para determinação dos SFC's atingíveis por cada nó

6.1 Décimo algoritmo: Compleição de Pares

Vimos, nos Capítulo 3, o Teorema 3.2, sugerindo a possibilidade de transformarmos o PSCH no problema de se encontrar um sumidouro fechado-por-pares no grafo-testemunha da instância dada. Por existir um número exponencial tanto de sumidouros quanto de subgrafos fechados-por-pares (e, possivelmente, mesmo de sumidouros fechados por pares [12]), descarta-se sumariamente o emprego da força bruta.

O algoritmo da *Compleição de Pares* (CP) começa pela construção do grafo-testemunha e pela determinação de seus SFC's. (Se não houver nenhum SFC próprio, então o algoritmo responde *não* imediatamente.) Então, como medida preparatória da estratégia de busca de SFC's fechados-por-pares que se seguirá, é executada a rotina *Localize_Sumidouros_Atingíveis* (LSA) dada na Figura 6.1, que coleta informação sobre o conjunto de SFC's que podem ser atingidos por cada nó do grafo-testemunha. (Seus detalhes serão focalizados mais adiante.)

Segue o algoritmo com a seleção de um dos SFC's, digamos S , e sua submissão à rotina principal, que destacamos na Figura 6.2, e à qual chamamos *Complete_os_Pares* (CoP).

A rotina CoP reúne os vértices rotuladores do SFC $S \subset G_T$ no conjunto L (que é visto como um candidato a CHS). Então, para cada par $\{x, y\}$ em L , verifica-se se o nó $[x, y]$ atinge, em G_T , qualquer SFC *que não seja o próprio* S . Se isto acontecer, (ou seja, se é o caso em que qualquer SFC diferente de S é atingível por $[x, y]$), então é adicionado um arco saindo de S (isto é, de qualquer de seus nós) e chegando a $[x, y]$ — de forma que S deixa de ser um SFC — e o algoritmo abandona S e parte para nova rotina CoP (no próximo SFC de G_T). Se, por outro lado, $[x, y]$ atinge apenas um SFC e este é o próprio S , então não haverá adição de arcos, e o algoritmo então apenas acrescentará ao candidato L os vértices rotuladores dos nós que são *vizinhos de saída* de $[x, y]$ em G_T — caso eles não se encontrem ainda em L . (A necessidade desta medida será evidenciada logo adiante, durante a prova de corretude do algoritmo.)

Se todos os pares de vértices em L forem investigados sem terem ensejado a adição de nenhum novo arco ao grafo-testemunha, então o algoritmo terá

Complete_os_Pares ($G_T(V_T, E_T), S$)

1. **seja** $L = \{v \in V \mid [u, v] \in S \text{ — ou } [v, u] \in S \text{ — para algum } u\}$
2. **seja** P uma lista contendo todo $[x, y] \in V_T$ tal que $\{x, y\} \subset L$
3. **enquanto** P é não-vazio **faça**
 - 3.1. **seja** $[x, y]$ o primeiro elemento em P
 - 3.2. **se** $sfc(x, y) \neq S$ // *obtido pela rotina LSA*
 - 3.2.1. **adicione** o arco $([u, v] \rightarrow [x, y])$ a E_T , para algum $[u, v] \in S$
 - 3.2.2. $P \leftarrow \emptyset; L \leftarrow \emptyset$
 - 3.3. **senão**
 - 3.3.1. **para cada** nó $[z, w]$ tal que $([x, y] \rightarrow [z, w]) \in E_T$ **faça**
 - 3.3.1.1. **se** $z \notin L$
 - 3.3.1.1.1. **para cada** elemento $h \in L$ **faça** coloque $[z, h]$ em P
 - 3.3.1.1.2. $L \leftarrow L \cup \{z\}$
 - 3.3.1.2. **se** $w \notin L$
 - 3.3.1.2.1. **para cada** elemento $h \in L$ **faça** coloque $[w, h]$ em P
 - 3.3.1.2.2. $L \leftarrow L \cup \{w\}$
 - 3.4. **remova** $[x, y]$ de P
4. **se** $1 < |L| < |V|$
 - retorne** *sim*
5. **retorne** *não*

Figura 6.2: A rotina Complete_os_Pares

encontrado o sumidouro fechado-por-pares $S' = \{[u, v] \in V_T \mid u, v \in L\} = G_T\langle L \rangle$ e parará, assim, respondendo *sim*.

No momento em que todos os SFC's originais do grafo-testemunha tiverem sido submetidos à rotina CoP — sem que qualquer CHS tenha sido localizado — o algoritmo não poderá ainda responder *não*, já que é possível que as adições de arcos tenham propiciado a formação de *novos* SFC's. Por esta razão, todo o processo de localização de SFC's, execução da rotina LSA e de uma rotina CoP para cada SFC deve ser recommçado. (Encontra-se na Figura 6.3 o pseudo-código do algoritmo da Compleição de Pares.)

O algoritmo prossegue, então, dessa forma (com os sucessivos *turnos* de execuções da rotina CoP, que é como nos referiremos às iterações do laço principal do algoritmo — linha 2, na Figura 6.3), até que tenha encontrado algum CHS (respondendo *sim*) ou até que o grafo-testemunha corrente (o grafo-testemunha original acrescido dos arcos que tenham sido acrescentados até aquele momento) tenha se tornado fortemente conexo, não mais contendo SFC's próprios (respondendo *não*).

6.1.1 Prova de corretude

Chamaremos de grafo-testemunha *estendido* ao grafo-testemunha ao qual tenha sido adicionado qualquer número de arcos durante a execução do algoritmo CP.

Interessa-nos mostrar que: (i) se o algoritmo retorna *sim*, então a instância possui de fato um CHS; e (ii) se a instância de entrada possui algum CHS, então o algoritmo é capaz de encontrar algum CHS.

(i) Respostas *sim* são sempre resultantes de uma execução bem sucedida da rotina CoP. Mas aquela rotina apenas dá uma resposta positiva se foi localizado um conjunto $L \subset V$ tal que, em algum grafo-testemunha estendido $G'_T(V_T, E'_T)$ de (G_1, G_2) , *todos* os nós rotulados por dois vértices de L apenas apresentam vizinhos de saída *que também são rotulados por pares de vértices*

que pertencem a L . Dessa forma, L é o conjunto rotulador de um sumidouro fechado-por-pares, em G'_T . Ocorre que os nós de qualquer sumidouro de G'_T induzem um sumidouro também no grafo-testemunha original G_T , uma vez que $E'_T \supseteq E_T$. Conseqüentemente, pelo Teorema 3.2, L é um CHS.

(ii) Suponhamos que a instância (G_1, G_2) possua CHS L . Pelo Teorema 3.2, seu grafo-testemunha $G_T(V_T, E_T)$ contém um sumidouro fechado-por-pares $P = G_T \langle L \rangle$. Mostraremos não ser possível que a resposta dada seja *não*. Para que o algoritmo reponda *não*, é preciso que um número suficiente de arcos tenha sido adicionado de forma a tornar G_T fortemente conexo. Isto significa que, em particular, P precisa ter deixado de ser um sumidouro. Ocorre que todo arco adicional é tal que liga um SFC S a um de seus pares faltantes, isto é, a um nó $[x, y] \notin S$ tal que $x, y \in L(S)$. Assim, para que seja um arco de saída de P , o arco adicionado precisa ligar um SFC $S \subseteq P$ a um dos pares faltantes de S que *não pertença a P* . Isto é impossível, porque P é fechado-por-pares em G_T , e permanece fechado-por-pares em qualquer grafo-testemunha estendido $G'_T(V_T, E'_T)$, já que arco adicional algum em $E'_T \setminus E_T$ pode mudar este fato.

6.1.2 Análise de Complexidade

O algoritmo da Compleição de Pares compreende um passo em tempo $O(nm)$ para a construção do grafo-testemunha, mais alguns turnos de compleição de pares que consistem em: (i) particionar o grafo-testemunha (estendido) corrente em seus CFC's e localizar, dentre eles, os que são SFC's; (ii) determinar os sumidouros atingíveis de todos os nós; (iii) visitar os nós correspondentes aos pares (faltantes) de cada SFC até que um arco (por SFC) seja adicionado.

Algoritmo 10: Compleição de Pares ($G_1(V, E_1), G_2(V, E_2)$)

1. construa o grafo-testemunha $G_T(V_T, E_T)$ de (G_1, G_2)
2. **repita**
 - 2.1. particione G_T em seus componentes fortemente conexos
 - 2.2. encontre os sumidouros fortemente conexos próprios de G_T
 - 2.3. se não há nenhum SFC próprio **retorne não**
 - 2.4. Localize_Sumidouros_Atingíveis(G_T)
 - 2.5. **para cada SFC S faça**
 - 2.5.1. se Complete_os_Pares(G_T, S) retornar *sim*
 - 2.5.1.1. **retorne sim** // do contrário, um arco terá sido adicionado

Figura 6.3: O algoritmo da Compleição de Pares

O passo (i) faz uma chamada ao método de Tarjan [19] que, como já foi visto, demanda tempo linear no número de arcos do grafo-testemunha. Uma vez que o grafo-testemunha tem originalmente $O(nm)$ arcos, este passo levará tempo $O(nm + d(k))$ durante o k -ésimo turno, onde $d(k)$ é o número de arcos adicionados pelo algoritmo ao grafo-testemunha antes do começo do k -ésimo turno. Pelo Lema 3.4, o número de SFC's que não são fechados-por-pares é $O(m)$, portanto $d(k)$ é claramente limitado por $O(tm)$, onde t é o maior número possível de turnos de compleição de pares durante uma execução do algoritmo. Logo mostraremos que t é $O(\log n)$, o que nos dará um limite em $O(nm)$ para o tempo do passo (i).

A complexidade do passo (ii) é a da rotina LSA, cujo funcionamento é o seguinte: um atributo $sfc(\alpha)$, inicialmente *indefinido*, é associado a cada nó $\alpha \in V_T$. Então, a partir de um nó qualquer $[x, y]$ de cada SFC S do grafo-

testemunha G_T , a rotina percorrerá os arcos de G_T em sentido contrário, como se estivesse executando uma simples busca em largura com raiz em $[x, y]$ em um digrafo com os nós de G_T , mas com todos os seus arcos invertidos com relação aos de G_T . Conforme cada nó $\alpha \in V_T$ é visitado durante essa busca, a $sfc(\alpha)$ é atribuído S , se este é o primeiro SFC atingível a partir de α a ser descoberto, ou a indicação *vários*, se $sfc(\alpha)$ já não mais possui valor *indefinido*. Tal indicação é, certamente, suficiente, uma vez que tudo o que o algoritmo precisa conhecer é se determinado SFC vem a ser o *único* que é atingível por certo nó. O que permite-nos significativa economia de tempo, aqui, é o fato de que cada uma das buscas pode ser descontinuada em nós previamente marcados como *vários*, pois os nós que os atingem, no grafo-testemunha (e que seriam seus descendentes na árvore de largura da busca corrente) decerto já terão sido igualmente marcados como *vários* durante alguma busca anterior.

A Figura 6.4 ilustra uma execução da rotina LSA. As elipses rotuladas A , B e C , na parte de baixo da figura, representam SFC's do grafo-testemunha em determinado momento. O sumidouro atingível $sfc(\alpha)$ encontra-se indicado ao lado de cada nó α . Sinais de mais (+) indicam *vários*.

Arcos que partem de nós α que atingem um único SFC ($sfc(\alpha) \neq \textit{vários}$) foram visitados apenas *uma* vez. Por outro lado, arcos que partem de nós β que atingem mais de um SFC ($sfc(\beta) = \textit{vários}$) foram visitados exatamente duas vezes, pois buscas posteriores à segunda que o visitou (e que atribuiu indicador *vários* a seu atributo sfc) teriam sido descontinuadas em β . Já que todos os arcos são visitados um número constante de vezes, a complexidade de uma execução da rotina LSA é $O(|V_T| + |E_T|) = O(n^2 + nm) = O(nm)$.

Finalmente, o passo (iii) gasta tempo constante em cada nó visitado. É fácil ver que o número máximo de nós visitados, em cada turno, é limitado

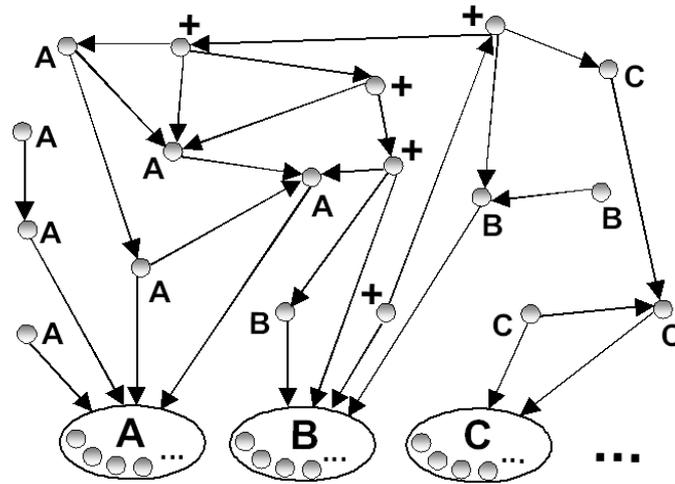


Figura 6.4: Execução da rotina Localize_Sumidouros_Atingíveis

em $O(n^2)$, pois nós para os quais nenhum arco *de entrada* foi adicionado pelo algoritmo foram visitados durante, no máximo, *uma* execução da rotina CoP, ao passo em que nós que receberam arcos de entrada adicionais totalizam não mais que um por SFC.

A respeito do número de turnos de compleição de pares, no pior caso, seja $S(k) = \{S_i : i = 1, \dots, s\}$ o conjunto de SFC's no começo do k -ésimo turno. Sabemos que, ao final desta k -ésima iteração do laço principal do algoritmo, todos aqueles s SFC's S_i terão deixado de ser SFC's. Estamos interessados em saber o número máximo de *novos* SFC's que podem ter sido formados. Ora, se um novo SFC S' foi constituído, é forçoso que ele contenha pelo menos um subgrafo de $S(k)$, que agora não mais é um sumidouro. Isto é verdade porque apenas arcos com origem em nós internos a ex-sumidouros de S_i foram adicionados durante o k -ésimo turno — de forma que nenhum sumidouro pode ter sido formado contendo apenas vértices que não pertençam

a qualquer S_i . No entanto, sabemos que um arco de saída adicionado a um nó de $S_p \in S(k)$ durante o k -ésimo turno lhe forneceu um caminho para um outro elemento de $S(k)$, digamos, S_q . Conseqüentemente, só será possível a existência de um sumidouro $S' \supset S_p$ se S' contiver também S_q . Assim, o número mínimo de elementos de $S(k)$ contidos em qualquer novo SFC que se tenha constituído é, na verdade, *dois*. Como SFC's são obviamente disjuntos, o número de SFC's no começo do turno $(k + 1)$ será, no máximo, $|S(k)|/2$, e assim, $O(\log n)$ turnos são suficientes para fortemente-conectar G_T .

A complexidade de tempo global do algoritmo da Compleição de Pares é, portanto, $O(\log n) \cdot O(nm) = O(nm \log n)$.

Capítulo 7

Resultados Experimentais

Todos os algoritmos apresentados nesta tese foram implementados numa máquina com processador Pentium 4 de 3,2 GHz com 2 GB de memória RAM.

As instâncias $R_{n,ob,pr}$ foram obtidas randomicamente, a partir do número de vértices n e das porcentagens de arestas obrigatórias (ob) e proibidas (pr).

Ao algoritmo de Monte Carlo foi requerida uma probabilidade de acerto mínima de 95%.

Os resultados obtidos mostram claramente que:

- os comportamentos de todos os algoritmos estão de acordo com as previsões da análise teórica;
- para instâncias pequenas, os algoritmos que se utilizam de uma grande parcela inicial de tempo para a montagem do grafo-testemunhas são compreensivelmente mais lentos;
- ainda que a complexidade de pior caso do algoritmo Duas Fases não seja melhor do que a do algoritmo Cliques Crescentes para entradas com

	$R_{100,10,10}$	$R_{200,10,10}$	$R_{400,10,10}$	$R_{100,10,70}$	$R_{200,10,70}$	$R_{400,10,70}$
IE	1"620	18"580	5'56"670	1"109	18"407	6'06"921
2F	0"265	0"875	3"813	0"609	5"470	out mem
SB	0"234	2"985	38"703	0"234	2"844	37"531
MC	0"094	0"828	6"437	0"094	0"750	6"687
SH	0"125	1"219	12"610	0"141	1"219	12"719
CC	0"110	0"859	8"563	0"125	0"859	9"297
LV	0"032	0"343	3"160	0"062	0"375	2"985
PA	0"268	0"888	4"105	0"620	6"212	out mem
CP	0"280	0"900	3"995	0"646	5"533	out mem

Figura 7.1: Resultados experimentais (Instâncias Aleatórias)

muitas arestas obrigatórias e proibidas, entradas para as quais o CC apresenta desempenho superior ao 2F são raras e de difícil obtenção, uma vez que seu grafo-testemunha precisa possuir um grande número de estruturas bastante peculiares que são os SFC's que não são fechados-por-pares.

- a mesma dificuldade é encontrada quando se tenta observar instâncias de entrada para as quais o algoritmo Preenchimento Acelerado é mais rápido do que o algoritmo Duas Fases, embora o PA certamente não seja jamais menos eficiente que o 2F.
- uma grande desvantagem dos algoritmos baseados no grafo-testemunha, cuja matriz de adjacências é biquadrática no número de vértices da entrada, é sua grande demanda de espaço.

	C ₁₀₀	C ₂₀₀	C ₄₀₀	C ₈₀₀	C ₁₆₀₀
IE	1"470	19"468	6'18"631	1h45'39"121	28h02'22"424
2F	0"141	0"500	2"330	8"423	35"532
SB	0"203	2"627	36"227	7'12"790	1h22'25"139
MC	0"063	0"563	6"399	45"107	5'45"349
SH	0"094	1"320	11"970	1'43"499	15'16"156
CC	0"062	0"688	7"148	1'04"659	9'27"998
LV	0"031	0"297	2"700	19"735	2'37"126
PA	0"130	0"510	2"344	8"820	34"294
CP	0"144	0"571	2"388	8"420	36"444

Figura 7.2: Resultados experimentais (Ciclos)

- o algoritmo randomizado de Las Vegas é, na prática, a melhor opção (muito simples e rápido) para a solução do PSCH — embora o algoritmo Compleição de Pares seja o melhor algoritmo determinístico para o problema.

Capítulo 8

Conclusão

8.1 Quadro de algoritmos e publicações

O algoritmo Incorporação Exaustiva foi publicado por Cerioli *et al.* em 1998 [2].

O algoritmo Sumidouros Fortemente Conexos foi publicado por Tang *et al.* em 2001 [18].

Os contraexemplos para o algoritmo SFC e o algoritmo Duas Fases são originais de minha dissertação de mestrado, defendida em 2003 [17]. Esse material foi em seguida apresentado no *Workshop on Combinatorics, Algorithms and Applications* (<http://www.ime.usp.br/~yoshi/pronex/Workshop/>). A contraprova para o SFC foi publicada na *Information Processing Letters* [8].

Os algoritmos Subconjuntos Balanceados e Monte Carlo foram apresentados no *III Workshop on Efficient and Experimental Algorithms* (<http://wea2004.inf.puc-rio.br/>) e apareceram em um volume do *Lecture Notes in Computer Science* [6], tendo sido o último apresentado também no *Mathematical Programming in Rio: A Conference in Honour of Nelson Maculan* [5].

Algoritmo	Complexidade de Tempo	Referências
Incorporação Exaustiva	$O(n^4)$	[2]
Sumidouros Fortemente Conexos	$O(n^2 \Delta_2)$ [incorreto]	[18, 8]
Duas Fases	$O(mM)$	[17, 9]
Subconjuntos Balanceados	$O(n^{3,5})$	[6, 9]
Monte Carlo	$O(n^3)$ [randomizado]	[5, 6, 9]
Série Harmônica	$O(n^3 \log n)$	[6, 9]
Cliques Crescentes	$O(n^3 \log n)$	[9]
Las Vegas	$O(n^3)$ [randomizado]	[9]
Preenchimento Acelerado	$O(n^3 \log \frac{m}{n})$	[9]
Compleição de Pares	$O(nm \log n)$	[1]

Figura 8.1: Tabela de algoritmos

Todos os algoritmos de nossa autoria, desde o Duas Fases até o Preenchimento Acelerado (com a exceção, portanto, do Compleição de Pares, mais recente) constam de um artigo recém-aceito para publicação na revista *Algorithmica* [9].

Finalmente, nosso artigo contendo o algoritmo Compleição de Pares foi aceito para publicação na *Information Processing Letters* e pode ser encontrado em [1].

A Figura 8.1 contém um resumo dos algoritmos conhecidos para o PSCH.

8.2 Últimas considerações

O conjunto de idéias que apresentamos nesta tese, mais do que oferecer soluções para um problema, permite a observação do processo pelo qual são progressivamente desvendados seus segredos e aprimorados os métodos para melhor resolvê-lo. Este mecanismo, um misto de tentativa e erro, criatividade e análise, é aproximadamente o mesmo, qualquer que seja o problema em foco.

Muito se cresce com uma pesquisa deste porte. Além da destreza técnica que se exercita e do ferramental algorítmico com que se trava contato, aprende-se, entre outras coisas: a confiar na intuição; a atentar para a comprovação cuidadosa de detalhes que pareceriam, de outra forma, auto-evidentes; a não desprezar concepções que tenham parecido, à primeira vista, fracassadas; a entender que, após um pensamento frutífero, alta é a probabilidade de que o tempo empregado em refiná-lo converta-se em outros ainda melhores, ou que cheguem mesmo a invalidá-lo; que algumas coisas ficam muito mais claras quando nos dispomos a escrevê-las; que muitas coisas apenas ficam completamente claras quando nos dispomos a compartilhá-las.

O Problema-Sanduíche do Conjunto Homogêneo foi um bom companheiro. Da idéia inicial que identificou o problema como polinomialmente computável ao algoritmo mais rápido que hoje conhecemos, longo foi o caminho. Cada pequena descoberta, embora trouxesse por si só a recompensa de um pequeno fim atingido, atuou sobretudo como *meio* de se prosseguir na busca, reabastecendo a imaginação e descortinando novos caminhos e suas promessas. E o problema-companheiro nunca perdeu sua graça, fonte inesgotável de surpresas.

A pesquisa para uma tese de doutorado preenche um tempo todo especial em nossas vidas, divisor de águas do qual se guarda, imagino, as mais ternas recordações. E, embora felizes por atingirmos o fim almejado de tão importante etapa, fica no ar um quê de saudade. Chegado, pois, o momento de oferecermos a este trabalho suas linhas derradeiras, não furta-se a mente ao pensamento final: e agora?

Apêndice A

Implementação eficiente da incorporação de testemunhas

O pseudo-código da Figura A.1 mostra uma maneira simples e eficiente de se implementar o procedimento de incorporação de testemunhas, com que lidamos ao longo de praticamente todo o texto. Esta implementação econômica tem papel importante na análise de alguns dos algoritmos apresentados.

A idéia é destinar, a cada vértice $v \in V$, dois *flags* booleanos $ob[v]$ e $pr[v]$ que sinalizarão, respectivamente, se há alguma aresta obrigatória ou proibida entre v e qualquer dos vértices presentes no conjunto candidato H corrente. Consegue-se, assim, acelerar a incorporação, uma vez que as operações de conjuntos em tempo $O(n)$ por vértice incorporado, da versão original de Cerioli *et al.* [2], são substituídas por operações de leitura e escrita de *flags* booleanos em tempo constante para cada um dos vizinhos obrigatórios e proibidos do vértice incorporado.

Há uma etapa de inicialização (linhas 3 a 4.1.2) que seta os *flags* de todos os vértices que não estão contidos no candidato inicial H_1 , em função de

serem ou não vizinhos obrigatórios e/ou proibidos de algum vértice $x \in H_1$. Vértices $b \in V \setminus H_1$ que tenham ambos os indicadores booleanos em *verdadeiro* são colocados no conjunto-testemunha inicial B .

Começa, então, o laço principal, que é encerrado somente quando B torna-se vazio. A cada iteração, um vértice b é “incorporado”, ou seja, movido de B para H . Nesse momento, os únicos vértices que precisam ser considerados (de modo a terem seus *flags* atualizados, se necessário, e poderem ser eventualmente adicionados a B) são os vizinhos obrigatórios e proibidos de b que estejam em $V \setminus (H \cup B)$.

A corretude dessa implementação é facilmente verificada. Mostraremos que: (i) cada vértice em $H_{k+1} \setminus H_k$ é de fato um vértice-testemunha de H_k ; e (ii) cada vértice-testemunha de H_k irá necessariamente pertencer a H_{k+t} , para algum t positivo, antes que o procedimento pare.

O fato de que um vértice apenas passa a fazer parte do conjunto candidato H se ambos seus booleanos (*ob* e *pr*) tiverem valor *verdadeiro* garante (i).

A respeito de (ii), temos que o procedimento não pára sem que B esteja vazio (linha 5), e que vértices só saem de B para serem adicionados a H ; basta mostrarmos que as testemunhas de H_k *entram* em B . Ora, cada testemunha b de H_k ou é também testemunha do candidato inicial H_1 , e foi portanto adicionada a B durante a etapa de inicialização, ou teve seus *flags* atualizados, a cada iteração, de forma que pôde ser adicionado a B (e necessariamente o foi) no momento em que o primeiro par {vizinho obrigatório de b , vizinho proibido de b } passou a fazer parte de H (linhas 5.3.2 e 5.4.2).

O tempo de execução desta incorporação acelerada é claramente $O(m_{ob} + m_{pr}) = O(M)$, já que cada aresta não-opcional — seja ela obrigatória ou proibida — dispara uma operação em tempo constante por um número também constante de vezes.

Inc_Test_Acelerada $(G_1(V, E_1), G_2(V, E_2), H_1)$

1. $H \leftarrow H_1; B \leftarrow \emptyset$
 2. **para cada** vértice $v \in V$ **faça**
 - 2.1. $ob[v] \leftarrow falso; pr[v] \leftarrow falso$
 3. **para cada** vértice $v \in H$ **faça**
 - 3.1. **para cada** vértice $x \in N_{ob}(v)$ **faça**
 - 3.1.1. $ob[x] \leftarrow verdadeiro$
 4. **para cada** vértice $v \in H$ **faça**
 - 4.1. **para cada** vértice $x \in N_{pr}(v)$ **faça**
 - 4.1.1. $pr[x] = verdadeiro$
 - 4.1.2. **if** $ob[x] = verdadeiro$ **e** $x \notin H$
 $B \leftarrow B \cup \{x\}$
 5. **enquanto** $|B| > 0$ **faça**
 - 5.1. $b \leftarrow$ um elemento de B
 - 5.2. $B \leftarrow B \setminus \{b\}; H \leftarrow H \cup \{b\}$
 - 5.3. **para cada** vértice $x \in N_{ob}(b)$ **faça**
 - 5.3.1. $ob[x] \leftarrow verdadeiro$
 - 5.3.2. **se** $pr[x] = verdadeiro$ **e** $x \notin H$
 $B \leftarrow B \cup \{x\}$
 - 5.4. **para cada** vértice $x \in N_{pr}(b)$ **faça**
 - 5.4.1. $pr[x] \leftarrow verdadeiro$
 - 5.4.2. **se** $ob[x] = verdadeiro$ **e** $x \notin H$
 $B \leftarrow B \cup \{x\}$
 6. **se** $H \neq V$ **retorne** *sim*
 7. **retorne** *não*
-

Figura A.1: Implementação eficiente da incorporação de testemunhas

Apêndice B

Construção eficiente do grafo-testemunha

De acordo com [18], a montagem do grafo-testemunha $G_T(V_T, E_T)$ de uma dada instância (G_1, G_2) do PSCH requerer tempo $O(n^2\Delta_2)$. Em realidade, sua construção pode ser conseguida em tempo $O(nm)$, uma vez que se utilize uma estrutura de dados adequada, tal qual a seguinte:

- matrizes de adjacências para G_1 e G_2 (ou apenas uma única matriz, onde cada célula indica um de três tipos de aresta: obrigatória, opcional ou proibida), de forma a permitir verificação em tempo constante do tipo de adjacência existente entre dois vértices quaisquer;
- duas listas de adjacências para cada vértice $v \in V$, permitindo acessar em tempo $O(|N_1(v)|)$ e $O(|\bar{N}_2(v)|)$, respectivamente, todos os vizinhos obrigatórios e proibidos de v — em lugar do tempo $O(n)$ que seria necessário para se percorrer toda uma linha da matriz de adjacências.

A Figura B.1 mostra, em pseudo-código, uma maneira simples de se conseguir esta obtenção acelerada do grafo-testemunha.

A inserção, no grafo-testemunha, de todas as arestas que *saem* de cada nó $[x, y] \in V_T$ requer a determinação do conjunto-testemunha de $\{x, y\}$ com respeito a (G_1, G_2) . Ocorre que os vértices de testemunha de $\{x, y\}$ são aqueles que são ao mesmo tempo vizinhos obrigatórios de x e proibidos de y , ou vice-versa. Com vistas a se determinar, de forma eficiente, a interseção, digamos, do conjunto de vizinhos obrigatórios de x com o de vizinhos proibidos de y , percorremos a *menor* dentre as listas de adjacências que lhes correspondem, checando, para cada um de seus elementos, se se trata de elemento também da lista mais longa (mas esta checagem *não* é feita em tempo linear no tamanho da lista mais longa, mas sim em tempo constante, por meio de um simples acesso à posição correspondente na matriz de adjacências).

Suponhamos que, ao invés de optar por percorrer a menor das listas, nosso procedimento sempre percorresse a lista de vizinhos *obrigatórios*. (Isto, certamente, não o tornaria mais rápido.) Fica claro, nessa maneira piorada (ou pelo menos não-melhorada) de se realizar a verificação de elementos comuns a ambas as listas, que é gasto tempo $\Theta(nm_{ob})$ para se obter todo o conjunto de arestas E_T , uma vez que, para todo $x \in V$, cada vizinho obrigatório de x seria acessado $\Theta(n)$ vezes (uma vez para cada $[x, w] \in V_T$), ensejando um acesso à matriz de adjacências e, possivelmente, a inclusão de uma aresta a E_T , ambos em tempo $O(1)$.

De modo análogo, um limite $\Theta(nm_{pr})$ seria encontrado, caso nosso método sempre optasse pelo percurso na lista de vizinhos *proibidos* (ao invés da menor das listas).

Dessa forma, o procedimento apresentado na Figura B.1, que certamente não é mais lento do que essas versões modificadas, tem seu tempo computacional limitado em $O(\min\{nm_{ob}, nm_{pr}\}) = O(nm)$.

Grafo-Testemunha_Acelerado $(G_1(V, E_1), G_2(V, E_2))$

1. $V_T \leftarrow \{[x, y] \mid x, y \in V, x \neq y\}$
 2. $E_T \leftarrow \emptyset$
 3. **para cada** nó $[x, y] \in V_T$ **faça**
// obrigatório p/ x, proibido p/ y
 - 3.1. **se** $|N_{ob}(x)| \leq |N_{pr}(y)|$
 - 3.1.1. **para cada** nó $w \in N_1(x)$ **faça**
 - 3.1.1.1 **se** (y, w) é aresta proibida
 $E_T \leftarrow E_T \cup \{([x, y] \rightarrow [x, w]), ([x, y] \rightarrow [y, w])\}$
 - 3.2 **senão**
 - 3.2.1. **para cada** nó $w \in N_{pr}(y)$ **faça**
 - 3.2.1.1 **se** (x, w) é aresta obrigatória
 $E_T \leftarrow E_T \cup \{([x, y] \rightarrow [x, w]), ([x, y] \rightarrow [y, w])\}$
// obrigatório p/ y, proibido p/ x
 - 3.3. **se** $|N_{ob}(y)| \leq |N_{pr}(x)|$
 - 3.3.1. **para cada** nó $w \in N_1(y)$ **faça**
 - 3.3.1.1 **se** (x, w) é aresta proibida
 $E_T \leftarrow E_T \cup \{([x, y] \rightarrow [x, w]), ([x, y] \rightarrow [y, w])\}$
 - 3.4 **senão**
 - 3.4.1. **para cada** nó $w \in N_{pr}(x)$ **faça**
 - 3.4.1.1 **se** (y, w) é aresta obrigatória
 $E_T \leftarrow E_T \cup \{([x, y] \rightarrow [x, w]), ([x, y] \rightarrow [y, w])\}$
 - 4 **retorne** $G_T(V_T, E_T)$.
-

Figura B.1: Construção eficiente do grafo-testemunha

Apêndice C

Contraexemplo interessante para o segundo algoritmo

Instância semelhante à que apresentamos aqui foi originalmente apresentada em [17]. A que aparece na Figura C.1 constitui, no entanto, uma melhoria em relação àquela, conseguindo mostrar de forma mais econômica a situação almejada, qual seja a de enganar duplamente o algoritmo dos Sumidouros Fortemente Conexos de Tang *et al.* [18]: seu grafo-testemunha apresenta dois SFC's (e apenas dois), nenhum dos quais associado a conjunto homogêneo sanduíche daquela instância; e, ainda assim, tal instância *possui* CHS, não podendo este ser, todavia, localizado pelo algoritmo, uma vez que seus vértices não constituem conjunto rotulador de qualquer SFC daquele grafo-testemunha.

Na Figura C.2 vê-se, de forma esquematizada, o grafo-testemunha que se obtém para a instância da Figura C.1. Note que, por economia de espaço, não colocamos vírgulas separando os vértices rotuladores de cada um dos nós. Há três “caixas”, em cor escura, cada qual contendo 18 nós. Arcos que chegam a uma dessas caixas indicam a existência de 18 arcos de origem comum e

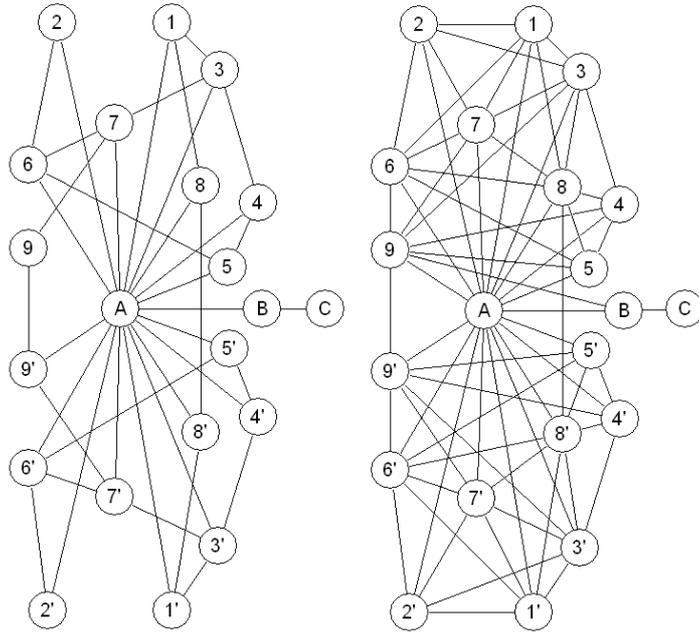


Figura C.1: Contraexemplo à Proposição 3.1 [18] — grafos de entrada

que incidem em cada um dos 18 nós daquele grupo. Da mesma forma, arcos que saem de uma caixa são indicadores de 18 arcos com destino comum, cada qual sendo originado em um dos nós daquele grupo. O nó rotulado K comporta todo o subgrafo apresentado na Figura C.3. Nós que chegam a K são, na verdade, incidentes a *algum* vértice do subgrafo representado por K . Importa aqui apenas o fato de que não há qualquer arco saindo de K , o que faz dele um sumidouro (que não é, contudo, fortemente conexo).

A respeito da Figura C.3, é necessário ressaltarmos apenas que os nós S e S' indicam subgrafos isomorfos ao que se vê à esquerda da linha tracejada na Figura C.4, onde se pode perceber, pela existência dos arcos em negrito perfazendo três ciclos intersectantes, que tais subgrafos são fortemente conexos. Ressalte-se, também, ainda na Figura C.3, que há apenas arcos *chegando* a

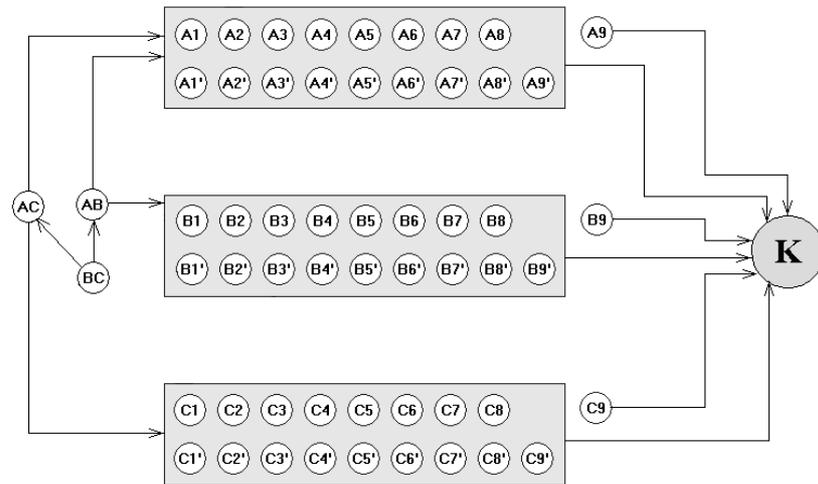


Figura C.2: Contraexemplo à Proposição 3.1 [18] — grafo-testemunha

S e S' (o que está simbolizado pelas pontas de seta grandes, em destaque), o que mostra serem aqueles subgrafos SFC's do grafo-testemunha em questão.

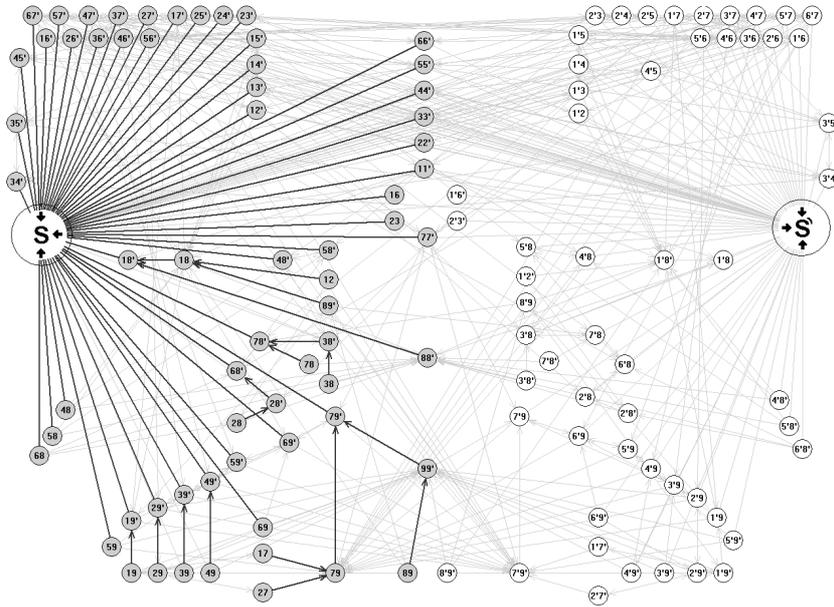


Figura C.3: Subgrafo K associado a CHS da instância da Figura C.1

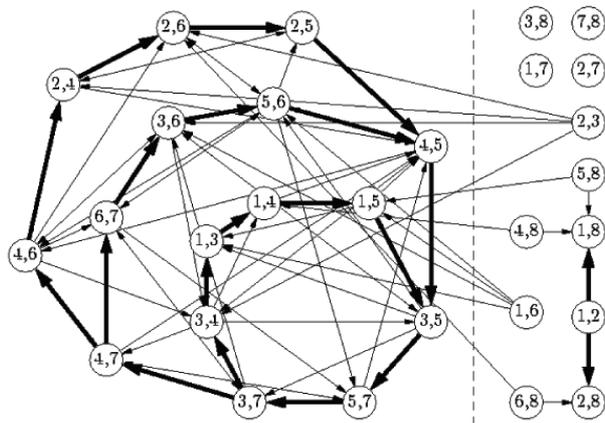


Figura C.4: SFC contido propriamente no subgrafo K

Bibliografia

- [1] BORNSTEIN, C. F., FIGUEIREDO, C. M. H., SÁ, V. G. P., “The Pair Completion algorithm for the Homogeneous Set Sandwich Problem”, Tech. Rep. ES-685/05, COPPE/Sistemas, Universidade Federal do Rio de Janeiro, 2005. disponível em <http://cronos.cos.ufrj.br/publicacoes/reltec/es68505.pdf>, aceito para publicação em *Information Processing Letters*.
- [2] CERIOLI, M. R., EVERETT, H., FIGUEIREDO, C. M. H., *et al.*, “The Homogeneous Set Sandwich Problem”, *Information Processing Letters*, v. 67, pp. 31–35, 1998.
- [3] DAHLHAUS, E., GUSTEDT, J., MCCONNELL, R. M., “Efficient and practical algorithms for sequential modular decomposition”, *Journal of Algorithms*, v. 41, pp. 360–387, 2001.
- [4] DANTAS, S., FARIA, L., FIGUEIREDO, C. M. H., “On decision and optimization (k,l) -graph sandwich problems”, *Discrete Appl. Math.*, v. 143, pp. 155–165, 2004.
- [5] FIGUEIREDO, C. M. H., FONSECA, G. D., Sá, V. G. P., “A fast monte carlo algorithm for the homogeneous set sandwich problem”, In:

- Proc. of Mathematical Programming in Rio: a Conference in Honour of Nelson Maculan*, pp. 35–39, 2003.
- [6] FIGUEIREDO, C. M. H., FONSECA, G. D., SÁ, V. G. P., *et al.*, “Faster deterministic and randomized algorithms on the homogeneous set sandwich problem”, In: *3rd Workshop on Efficient and Experimental Algorithms*, v. 3059 of *Lecture Notes Comput. Sci.*, pp. 243–252, Springer-Verlag, 2004. disponível em <http://dx.doi.org/10.1007/b97914>.
- [7] FIGUEIREDO, C. M. H., KLEIN, S., VUSKOVIC, K., “The graph sandwich problem for 1-join composition is NP-complete”, *Disc. Appl. Math.*, v. 121, pp. 73–82, 2002.
- [8] FIGUEIREDO, C. M. H., SÁ, V. G. P., “A note on the Homogeneous Set Sandwich Problem”, *Information Processing Letters*, v. 93, pp. 75–81, 2005.
- [9] FONSECA, G. D., SÁ, V. G. P., FIGUEIREDO, C. M. H., *et al.*, “The Homogeneous Set Sandwich Problem”, Tech. Rep. ES-680/05, COPPE/Sistemas, Universidade Federal do Rio de Janeiro, 2005. disponível em <http://www.cos.ufrj.br/publicacoes/reltec/es68005.pdf>, aceito para publicação em *Algorithmica*.
- [10] GOLUMBIC, M. C., KAPLAN, H., SHAMIR, R., “Graph sandwich problems”, *Journal of Algorithms*, v. 19, pp. 449–473, 1995.
- [11] GOLUMBIC, M. C., WASSERMANN, A., “Complexity and algorithms for graph and hypergraph sandwich problems”, *Graphs Combin.*, v. 14, pp. 223–239, 1998.

- [12] HABIB, M., LEBHAR, E., PAUL, C., “A note on finding all homogeneous set sandwiches”, *Information Processing Letters*, v. 87, pp. 147–151, 2003.
- [13] HEDETNIEMI, S. T., “On hereditary properties of graphs”, *J. Combin. Theory (B)*, n. 14, pp. 94–99, 1973.
- [14] KAPLAN, H., SHAMIR, R., “Bounded degree interval sandwich problems”, *Algorithmica*, v. 24, pp. 96–104, 1999.
- [15] LOVÁSZ, L., “Normal hypergraphs and the perfect graph conjecture”, *Discrete Math.*, n. 2, pp. 253–267, 1972.
- [16] MCCONNELL, R. M., SPINRAD, J., “Linear-time modular decomposition and efficient transitive orientation of comparability graphs”, In: *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 536–545, 1994.
- [17] SÁ, V. G. P., *O Problema-Sanduíche para Conjuntos Homogêneos em Grafos*, COPPE / Universidade Federal do Rio de Janeiro, May 2003.
- [18] TANG, S., YEH, F., WANG, Y., “An efficient algorithm for solving the Homogeneous Set Sandwich Problem”, *Information Processing Letters*, v. 77, pp. 17–22, 2001.
- [19] TARJAN, R. E., “Depth-first search and linear graph algorithms”, *SIAM J. Comput.*, v. 1, n. 2, pp. 146–160, 1972.
- [20] TEIXEIRA, R. B., FIGUEIREDO, C. M. H., “The sandwich problem for cutsets”, In: *Proceedings of LACGA 2004*, v. 18 of *Electronic Notes in Discrete Mathematics*, pp. 219–225, Elsevier, 2004.