

Marco Aurélio Lopes Barbosa

*Algoritmos para encontrar
conjuntos de retorno*

Maringá - Paraná

Novembro de 2005

Marco Aurélio Lopes Barbosa

*Algoritmos para encontrar
conjuntos de retorno*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador:

Candido Ferreira Xavier de Mendonça Neto

UNIVERSIDADE ESTADUAL DE MARINGÁ

Maringá - Paraná

Novembro de 2005

Este trabalho foi feito para honra e glória do Senhor Jesus.

“Ó profundidade da riqueza da sabedoria e do conhecimento de Deus! Quão insondáveis são os seus juízos e inescrutáveis os seus caminhos! Quem conheceu a mente do Senhor? Ou quem foi seu conselheiro? Quem primeiro lhe deu para que o recompense? Pois dele, por ele e para ele são todas as coisas. A ele seja a glória para sempre! Amém.”

Romanos 11:33-36

Agradecimentos

Ao meu orientador Xavier e sua esposa Marilice pela amizade, incentivo e confiança.

À minha noiva Maíra, companheira sempre presente, dando força e ânimo nos momentos difíceis.

Aos meus pais, Celeste e João, pelas oportunidades, incentivo e cuidado.

Aos meus irmãos, Cassiano e Bruna, pela convivência e confiança.

Ao CNPq pelo apoio financeiro e a todos os funcionários do departamento de informática.

A todos, muito obrigado.

Resumo

Problemas do conjunto de retorno (FSP) consistem em encontrar um número mínimo de vértices ou arestas em um grafo orientado, cuja remoção resulta em um subgrafo acíclico. Alguns dos problemas de decisão associados ao FSP são conhecidos NP-completos [28]. Neste trabalho apresentamos uma revisão de quatro algoritmos para resolver problemas de conjunto de retorno, bem como resultados de experimentos computacionais e comparações entre eles. Também apresentamos uma melhoria em um dos algoritmos.

Palavras chaves: conjunto de retorno, conjunto de aresta de retorno, conjunto de vértice de retorno, problemas de conjunto de retorno, grafos planares.

Abstract

The feedback set problems consists in finding the minimum cardinality set of vertices or edges whose removal yields an acyclic subgraph. Some of the decision problems associated with the FPS is known to be NP-complete [28]. In this work we present four algorithm to solve feedback set problems and results of computational experiments between them. We also presents an improvement in an algorithm.

Keywords: feedback set, feedback arc set, feedback vertice set, feedback set problems, planar graph.

Lista de Figuras

2.1	Diagramas de grafos: (a) Diagrama de um grafo orientado. (b) Diagrama de um grafo não orientado	p. 15
2.2	Uma imersão planar e suas faces	p. 18
2.3	Subdivisão da aresta uv	p. 19
2.4	O $K_{3,3}$ é não planar	p. 19
2.5	O K_5 é não planar	p. 19
2.6	Ordem cíclica induzida pelo desenho do grafo	p. 20
2.7	Ciclos faciais de um desenho planar	p. 20
2.8	Um grafo planar e seu dual. Os vértices pintados e as arestas tracejadas são do grafo dual.	p. 21
2.9	Um grafo orientado planar e seu dual. Os vértices pintados e as arestas tracejadas são do grafo dual.	p. 22
2.10	FAS \Rightarrow FVS	p. 24
2.11	Exemplo de transformação do FAS para FVS.	p. 24
2.12	FVS \Rightarrow FAS	p. 25
2.13	Exemplo de transformação do FVS para FAS.	p. 26
3.1	Um algoritmo simples para resolver o FAS	p. 29
3.2	Algoritmo proposto por Saab para resolver o FAS	p. 32

3.3	Função <code>perturb</code>	p. 33
3.4	Função <code>bisect</code> usando evolução estocástica	p. 35
3.5	Esboço geral da meta-heurística GRASP	p. 36
3.6	Função <code>ReduceInstanceSize</code>	p. 38
3.7	Exemplo da aplicação da função <code>ReduceInstanceSize</code>	p. 38
3.8	Função <code>ConstructGreedyRandomizedSolution</code>	p. 40
3.9	Função <code>LocalSearch</code>	p. 41
4.1	Algoritmo proposto por Saab para resolver o FAS	p. 51
5.1	Tamanho do conjunto de retorno para grafos com 50 vértices: <code>sse</code> e <code>ssep</code>	p. 56
5.2	Tempo médio da iteração para grafos com 50 vértices: <code>sse</code> e <code>ssep</code> . . .	p. 56
5.3	Tamanho do conjunto de retorno para grafos com 50 vértices: <code>sse</code> , <code>sse</code> e <code>pqr</code>	p. 57
5.4	Tempo médio da iteração para grafos com 50 vértices: <code>sse</code> , <code>els</code> e <code>pqr</code> .	p. 57
5.5	Tamanho do conjunto de retorno para grafos com 100 vértices: <code>sse</code> e <code>ssep</code>	p. 58
5.6	Tempo médio da iteração para grafos com 100 vértices: <code>sse</code> e <code>ssep</code> . .	p. 58
5.7	Tamanho do conjunto de retorno para grafos com 100 vértices: <code>sse</code> , <code>sse</code> e <code>pqr</code>	p. 59
5.8	Tempo médio da iteração para grafos com 100 vértices: <code>sse</code> , <code>els</code> e <code>pqr</code>	p. 59
5.9	Tamanho do conjunto de retorno para grafos com 500 vértices: <code>sse</code> e <code>ssep</code>	p. 60
5.10	Tempo médio da iteração para grafos com 500 vértices: <code>sse</code> e <code>ssep</code> . .	p. 60
5.11	Tamanho do conjunto de retorno para grafos com 500 vértices: <code>sse</code> , <code>sse</code> e <code>pqr</code>	p. 61
5.12	Tempo médio da iteração para grafos com 500 vértices: <code>sse</code> , <code>els</code> e <code>pqr</code>	p. 61

5.13	Tamanho do conjunto de retorno para grafos com 1000 vértices: sse e ssep	p.62
5.14	Tempo médio da iteração para grafos com 1000 vértices: sse e ssep . .	p.62
5.15	Tamanho do conjunto de retorno para grafos com 1000 vértices: sse , sser e pqr	p.63
5.16	Tempo médio da iteração para grafos com 1000 vértices: sse , els e pqr	p.63

Lista de Tabelas

5.1	Algoritmos testados	p. 52
5.2	Resultado para grafos com 50 vértices: sse , sser , ssep e sserp	p. 56
5.3	Resultado para grafos com 50 vértices: sse , els e pqr	p. 57
5.4	Resultado para grafos com 100 vértices: sse , sser , ssep e sserp . . .	p. 58
5.5	Resultado para grafos com 100 vértices: sse , els , pqr	p. 59
5.6	Resultado para grafos com 500 vértices: sse , sser , ssep e sserp . . .	p. 60
5.7	Resultado para grafos com 500 vértices: sse , els , pqr	p. 61
5.8	Resultado para grafos com 1000 vértices: sse , sser , ssep e sserp . . .	p. 62
5.9	Resultado para grafos com 1000 vértices: sse , els , pqr	p. 63

Sumário

1	Introdução	p. 12
1.1	Objetivos	p. 13
1.2	Justificativas	p. 13
1.3	Estrutura da Dissertação	p. 13
2	Conceitos e definição do problema	p. 15
2.1	Grafos	p. 15
2.2	Planaridade	p. 17
2.2.1	Imersão combinatorial planar	p. 19
2.2.2	Grafo dual	p. 20
2.3	Problemas do conjunto de retorno	p. 21
2.3.1	Problema do conjunto de vértices de retorno	p. 22
2.3.2	Problema do conjunto de arestas de retorno	p. 23
	Equivalência entre os problemas FVS e FAS	p. 23
3	Trabalhos relacionados	p. 27
3.1	Relação entre ordenação de vértices e conjunto de arestas de retorno	p. 28
3.2	Uma heurística simples para resolver o FAS	p. 29
3.3	Evolução estocástica aplicada ao FAS	p. 30

Evolução estocástica	p. 32
3.4 GRASP aplicado ao FVS	p. 34
Fase de construção	p. 39
Fase de busca local	p. 40
3.5 Solução do FAS para grafos planares	p. 41
3.6 Algoritmos de teste de planaridade	p. 47
4 Algoritmo proposto	p. 49
4.1 Abordagens anteriores	p. 49
5 Resultados obtidos	p. 52
5.1 Grafos com 50 vértices	p. 53
5.2 Grafos com 100 vértices	p. 54
5.3 Grafos com 500 vértices	p. 54
5.4 Grafos com 1000 vértices	p. 55
Gráficos e tabelas	p. 55
Conclusões	p. 64
Trabalhos futuros	p. 64
Referências	p. 66

1 *Introdução*

Um *conjunto de arestas de retorno* (*feedback arc set*) de um grafo é um subconjunto de arestas, cuja remoção torna o grafo acíclico. Da mesma forma, um *conjunto de vértices de retorno* (*feedback vertex set*) de um grafo é um subconjunto de vértices, cuja remoção torna o grafo acíclico. Os problemas de *conjunto de retorno* consistem em encontrar um conjunto de retorno de vértices ou arestas de custo (cardinalidade) mínimo.

Segundo Festa, Pardalos e Resende [9] “dentre os problemas clássicos de otimização combinatória NP-completos, os problemas de conjunto de retorno são os menos compreendidos. Isto fica mais evidente pela falta de resultados positivos em termos de algoritmos exatos e aproximativos”.

À parte de seu interesse teórico, os problemas de conjunto de retorno têm aplicações práticas, tais como: análise de sistemas em larga escala com retorno, desenhos de grafos, verificação de programas, inferência estatística, desenho de circuito VLSI, etc.

Uma vez que os problemas de decisão associados à encontrar conjuntos de retorno são NP-completos, é comum o estudo de algoritmos exatos polinomiais que resolvam casos específicos do problema. Citamos como exemplo o problema do conjunto de arestas de retorno para grafos planares, que pode ser resolvido em tempo polinomial [12, 3].

Recentemente, surgiram na literatura dois algoritmos heurísticos práticos [23, 27] que produzem resultados relevantes em termos de qualidade de solução versus tempo de processamento. Isto é particularmente conveniente, pois amplia as perspectivas para algoritmos melhores e mais eficientes.

1.1 Objetivos

Nosso objetivo é apresentar uma revisão de algoritmos para encontrar conjuntos de retorno. Nossos objetivos específicos são:

- propor a utilização de um algoritmo polinomial exato para resolver o problema do conjunto de arestas de retorno para grafos planares como passo intermediário de um algoritmo para resolver o problema para grafos gerais;
- apresentar um estudo comparativo entre os algoritmos existentes para resolver problemas do conjunto de retorno.

1.2 Justificativas

Apesar de existirem vários algoritmos na literatura, nenhum deles explora o fato de existir algoritmos polinomiais exatos para resolver o problema do conjunto de arestas de retorno para grafos planares.

Além disso, não foi apresentado nenhum estudo comparativo de resultados computacionais de heurísticas recentes, como a de Saab [27] e Pardalos, Qian e Resende [23].

1.3 Estrutura da Dissertação

Esta dissertação está organizada da seguinte forma: no capítulo 2 são apresentados os conceitos gerais de grafos que são utilizados ao longo deste trabalho e também a definição dos problemas de conjunto de retorno.

O capítulo 3 contém uma revisão de quatro algoritmos para encontrar conjuntos de retorno: um algoritmo simples e uma heurística dividir para conquistar para encontrar conjunto de arestas de retorno, um algoritmo baseado no GRASP para encontrar conjunto de vértices de retorno e um algoritmo polinomial exato para encontrar conjunto de arestas de retorno para grafos planares.

No capítulo 4 apresentamos uma modificação no algoritmo de Saab [27]. No capítulo 5 apresentamos os resultados computacionais obtidos, bem como as comparações entre o algoritmo proposto e os algoritmos de Saab [27] e o de Pardalos, Qian e Resende [23].

Concluimos esta dissertação no capítulo 5.4 indicando possíveis trabalhos futuros.

A quem possa interessar, este trabalho foi escrito no *Ubuntu Linux*¹ utilizando o editor *vim*², as ilustrações foram feitas com o *dia*³ e o gráficos com o *gnuplot*⁴. O estilo de formatação para o sistema \LaTeX utilizado foi o $\text{ABN}\text{\TeX}$ ⁵.

¹<http://www.ubuntulinux.org>

²<http://www.vim.org>

³<http://www.gnome.org/projects/dia/>

⁴<http://www.gnuplot.info>

⁵<http://abntex.codigolivre.org.br>

2 Conceitos e definição do problema

Neste capítulo apresentamos os conceitos básicos sobre teoria dos grafos e planaridade que são utilizados nesta dissertação. Também apresentamos a definição de alguns problemas do conjunto de retorno.

2.1 Grafos

Um *grafo* G é uma tripla ordenada (V, A, ψ) , onde V é um conjunto finito de *vértices*, A é um conjunto finito de *arestas* e $\psi : A \rightarrow V \times V$ é uma função de incidência que associa cada aresta e a um par de vértices (u, v) . Se o par de vértice (u, v) for um par ordenado, dizemos que o grafo é *orientado*. Caso contrário, dizemos que o grafo é *não orientado*.

Intuitivamente um grafo pode ser representado por diagramas como os da figura 2.1. Os vértices são representados por pequenos círculos e as arestas são setas ligando os dois círculos correspondentes aos seus extremos. Para grafos não orientados, ao invés de usarmos setas, usamos apenas uma linha.

Dado a associação $\psi(e) = (u, v)$, dizemos que a aresta e é *incidente* em u e v e que e

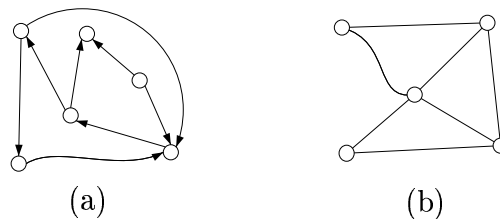


Figura 2.1: Diagramas de grafos: (a) Diagrama de um grafo orientado. (b) Diagrama de um grafo não orientado

saí de u e chega em v , neste caso u é a *origem* e v é o *destino*. Dizemos ainda que u e v são os *extremos* de e . Se dois vértices são extremos da mesma aresta eles são *adjacentes*.

Um *laço* é uma aresta, cujo os extremos são o mesmo vértice. Duas arestas e_1 e e_2 são *paralelas* se $\psi(e_1) = (u, v) = \psi(e_2)$. Um grafo que possui laços e/ou arestas paralelas é denominado *multigrafo*. Um grafo que não admite laços e arestas paralelas é denominado *grafo simples*. Para um grafo simples a função de incidência ψ torna-se bem definida e pode ser omitida. Neste trabalho consideramos apenas grafos simples e, portanto, definimos um grafo G apenas como uma dupla ordenada (V, A) . Representaremos uma aresta $\psi(e) = (u, v)$, apenas como (u, v) ou simplesmente como uv .

Um grafo $H = (V', A')$ é um *subgrafo* de um grafo $G = (V, A)$, notação $H \subseteq G$, se $V' \subseteq V$ e $A' \subseteq A$. Se $A' = \{(u, v) \in A \mid u, v \in V'\}$ dizemos que H é um *subgrafo induzido* por V' , neste caso denotamos H por $G[V']$. Seja $X \subseteq V$, denotamos o subgrafo induzido $G[V - X]$ por $G \setminus X$. Dizemos que H é *subgrafo próprio* de G , notação $H \subsetneq G$, se $|V'| < |V|$ ou $|A'| < |A|$.

Dizemos que um grafo $H = (V', A')$ é *menor* que um grafo $G = (V, A)$ se $|A'| < |A|$ e $|V'| \leq |V|$ ou $|A'| \leq |A|$ e $|V'| < |V|$. Se H é menor que G , então G é *maior* que H .

Seja \mathcal{P} um propriedade de grafos, dizemos que um subgrafo $H \subseteq G$ é *maximal* (*minimal*) com relação a propriedade \mathcal{P} se H possui a propriedade \mathcal{P} e nenhum subgrafo $H' \subseteq G$ tem a propriedade \mathcal{P} e $H \subsetneq H'$ ($H' \subsetneq H$). Dizemos também que H é subgrafo *máximo* (*mínimo*) com relação a propriedade \mathcal{P} se nenhum subgrafo $H' \subseteq G$ tem a propriedade \mathcal{P} e é maior (menor) que H .

Definimos como $\text{in}(v)$ o conjunto de arestas que chegam no vértice v . Da mesma forma, definimos como $\text{out}(v)$ o conjunto de arestas que saem do vértice v . Definimos como *grau de chegada* $d^-(v) = |\text{in}(v)|$ e como *grau de saída* $d^+(v) = |\text{out}(v)|$. O *grau* $d(v)$ de um vértice é a soma dos graus de entrada e saída deste vértice, isto é $d(v) = d^-(v) + d^+(v)$. Definimos como *fonte* um vértice com grau de entrada 0 e como *sumidouro* um vértice com grau de saída 0. O grau de um grafo é o maior valor do grau de seus vértices. Um

grafo é *regular* de grau r se todos os seus vértices tem grau r .

Um *passeio* P é um seqüência $\langle v_0, e_0, v_1, e_1, \dots, e_{n-1}, v_n \rangle$ alternada de vértices e arestas onde $e_i = (v_i, v_{i+1})$. Os vértices v_0 e v_n são os *extremos* do passeio e os vértices v_1, \dots, v_{n-1} são os vértices *internos* do passeio. Dizemos que o passeio *saí* de v_0 e *chega* em v_n . O *comprimento* do passeio P é o valor $n + 1$. Um *caminho* é um passeio sem repetição de vértices. Um *ciclo* é um passeio com os extremos iguais e sem repetição dos vértices internos. Um grafo que não contém ciclos é chamado de *acíclico*.

Um vértice v é *alcançável* a partir de um vértice u se existe um caminho que sai de u e chega em v . Um grafo não-orientado (orientado) é *conexo* (*fortemente conexo*) se para todo par de vértices u e v , v é alcançável a partir de u . Uma *componente* (*fortemente*) *conexa* de G é um subgrafo conexo de G maximal nesta propriedade.

Um grafo não orientado H é dito *subjacente* a um grafo orientado G se H é obtido a partir de G removendo a orientação das arestas de G . Chamamos uma componente \mathcal{C} de *componente fracamente conexa* se \mathcal{C} é maximal, \mathcal{C} não é uma componente fortemente conexa e o grafo subjacente a \mathcal{C} é conexo.

Dizemos que um vértice v é um *vértice de corte* de um grafo G se o grafo $G \setminus \{v\}$ contém mais componentes (fracamente) conexas que G . Um grafo conexo que não contém vértices de corte é dito *biconexo*.

Uma *árvore* é um grafo conexo sem ciclos. Uma *floresta* é um grafo onde cada componente conexa é uma árvore. Seja $T = (V, A')$ um subgrafo de um grafo $G = (V, A)$, se T é uma árvore então dizemos que T é uma *árvore geradora* de G .

2.2 Planaridade

Uma *imersão* de um grafo planar G em uma superfície S é uma representação geométrica (desenho) de G em S tal que dois vértices distintos não ocupam o mesmo lugar em S e não existe cruzamentos de arestas, a não ser, é claro, nos extremos quando duas arestas

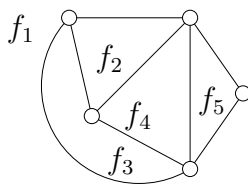


Figura 2.2: Uma imersão planar e suas faces. A imersão tem 5 faces, a face f_1 é a face externa.

são adjacentes ao mesmo vértice. Se um grafo G tem uma imersão em uma superfície S dizemos que G é *imersível* em S .

Se um grafo G é imersível no plano (\mathbb{R}^2), então dizemos que G é *planar*. Uma imersão de G no plano é denotada por \tilde{G} . As regiões limitadas pelas arestas de uma imersão planar \tilde{G} são chamadas de *faces*. Toda imersão planar contém uma face ilimitada denominada *face externa*. A face externa corresponde à região $\mathbb{R}^2 \setminus \tilde{G}$. Veja a figura 2.2.

Uma das relações mais conhecidas entre o número de faces, o número de vértices e o número de arestas de uma imersão planar é a fórmula de Euler. Esta fórmula pode ser enunciada da seguinte maneira:

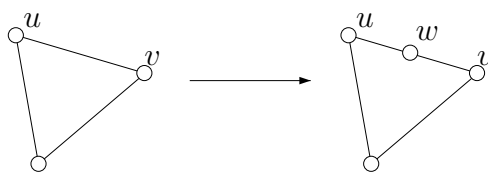
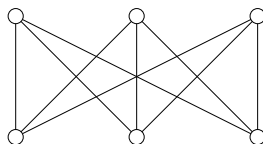
Teorema 1 (Fórmula de Euler). *Se G é um grafo planar conexo com n vértices, m arestas e f faces, então, $n - m + f = 2$.*

Corolário 2. *Se G é um grafo planar conexo com n vértices, então G tem no máximo $3n - 6$ arestas.*

A seguir, definiremos dois conceitos utilizados na caracterização de grafos planares mais utilizados, a de Kuratowski [19]. Uma *operação de subdivisão* de uma aresta $e = (u, v)$ é a substituição de e por um novo vértice w e duas novas arestas (u, w) e (w, v) (figura 2.3). Uma *subdivisão* de um grafo G é um grafo H que pode ser obtido a partir de G por uma seqüência finita de operações de subdivisão de arestas.

O teorema de Kuratowski é enunciado em termos de dois grafos não planares fundamentais, o $K_{3,3}$ (figura 2.4) e o K_5 (figura 2.5).

Teorema 3 (Kuratowski). *Um grafo G é planar se, e somente se, G não contém uma*

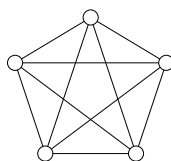
Figura 2.3: Subdivisão da aresta uv Figura 2.4: O $K_{3,3}$ é não planar

subdivisão do $K_{3,3}$ e do K_5 .

2.2.1 Imersão combinatorial planar

Todo desenho de grafo induz uma ordem cíclica das arestas adjacentes a cada vértice. Neste trabalho consideramos a ordem como sendo anti-horária, onde denotamos a ordem cíclica de um vértice v por $D(v)$. A figura 2.6 mostra um desenho de um grafo de 4 vértices, onde para cada vértice existe uma ordem cíclica induzida pelo desenho: $D(a) = \langle ad, ac, ab \rangle$, $D(b) = \langle ba, bc, bd \rangle$, $D(c) = \langle ca, cd, cb \rangle$ e $D(d) = \langle db, dc, da \rangle$. Uma vez que a ordem é cíclica, poderíamos então escrever $D(a) = \langle ac, ab, ad \rangle$ ou $D(a) = \langle ab, ad, ac \rangle$.

Existem infinitos desenhos para cada grafo. Os desenhos podem ser divididos em classes de equivalência. Dois desenhos do mesmo grafo são *equivalentes* se a ordem cíclica de cada vértice (induzida pelo desenho) é igual para os dois desenhos. Uma classe de equivalência de desenhos de um grafo é chamada de *imersão combinatorial*. Desta forma, uma imersão combinatorial é definida pela ordem cíclica das arestas de cada vértice do grafo.

Figura 2.5: O K_5 é não planar

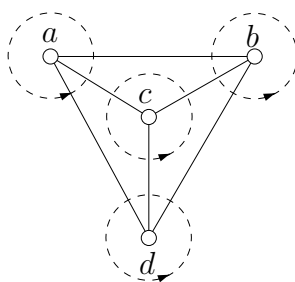


Figura 2.6: Ordem cíclica induzida pelo desenho do grafo

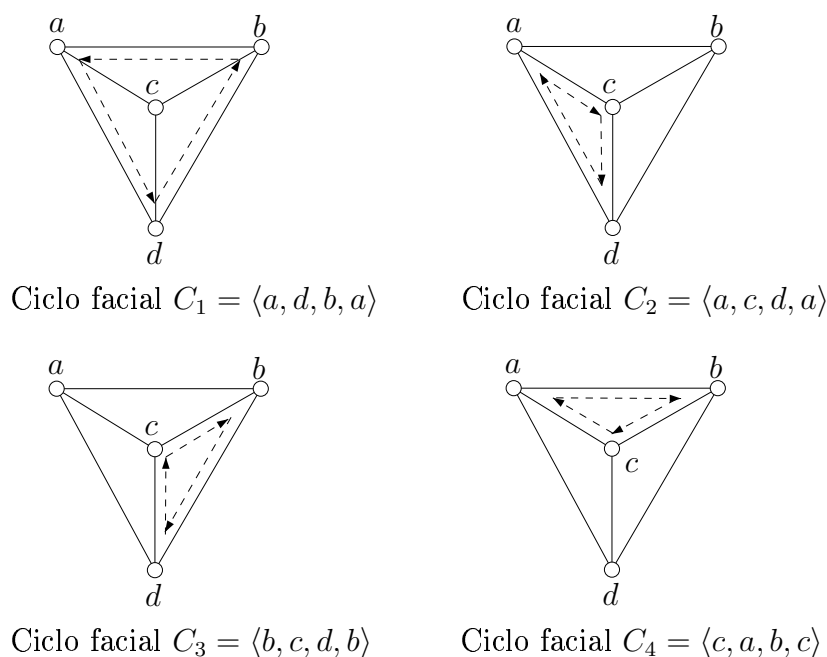


Figura 2.7: Ciclos faciais de um desenho planar

Uma imersão combinatorial de desenhos planares de um grafo é chamada de *imersão combinatorial planar*. Dado uma imersão combinatorial planar D um *ciclo facial* de D é ciclo $\langle v_0, e_0, v_1, e_1, \dots, v_{k-1} \rangle$ onde, para cada i , o sucessor cíclico da aresta $e_{(i+1) \bmod k} = (v_i \bmod k, v_{(i+1) \bmod k})$ é a aresta $e_{(i+2) \bmod k} = (v_{(i+1) \bmod k}, v_{(i+2) \bmod k})$ (figura 2.7).

2.2.2 Grafo dual

Definimos como grafo dual G^* de um grafo planar G como o grafo construído da seguinte maneira:

- para cada face de G existe um vértice em G^*

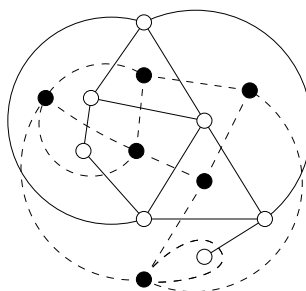


Figura 2.8: Um grafo planar e seu dual. Os vértices pintados e as arestas tracejadas são do grafo dual.

- para cada aresta de G existe uma aresta em G^* conectando os dois vértices correspondentes às faces separada por e .

Se um grafo G^* é dual de um grafo G , dizemos que G é o grafo *primal*. Um grafo dual (figura 2.8) pode conter laços e arestas múltiplas. Grafos duais estão relacionados com imersão planar, cada imersão planar de um grafo G gera um grafo dual G^* .

Podemos ampliar o conceito de grafo dual se considerarmos grafos orientados planares e definirmos uma maneira sistemática de orientar as arestas do grafo dual. Cada aresta de um grafo orientado planar G participa de dois ciclos faciais. Um dos ciclos facial tem a mesma orientação da aresta e o outro ciclo facial tem orientação contrária. Vamos usar a seguinte regra para orientar as arestas do grafo dual: uma aresta e' no grafo dual, correspondente à aresta e do grafo primal, sai do vértice correspondente ao ciclo facial com mesma orientação da aresta e e entra no vértice correspondente ao outro ciclo facial (figura 2.9).

A seguir, apresentamos uma descrição formal de dois problemas de conjunto de retorno, que são o objeto de estudo desta dissertação, e como estes se relacionam com outros problemas.

2.3 Problemas do conjunto de retorno

Intuitivamente, um *conjunto de retorno* é um conjunto de arestas ou um conjunto de vértices de um grafo G (orientado ou não), cuja remoção torna G acíclico. Os *problemas do*

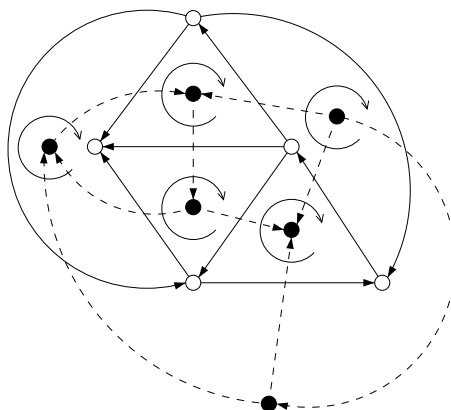


Figura 2.9: Um grafo orientado planar e seu dual. Os vértices pintados e as arestas tracejadas são do grafo dual.

conjunto de retorno consistem em encontrar um conjunto de retorno S de custo mínimo – quando as arestas ou vértices possuem pesos – ou cardinalidade mínima – quando arestas ou vértices não possuem pesos. Em outras palavras, o conjunto de retorno deve cobrir todos os ciclos de uma coleção \mathcal{C} de ciclos de um grafo $G = (V, A, w)$, onde w é uma função positiva definida sobre as arestas ou sobre os vértices.

Além disto, podemos restringir o conjunto de retorno a um subconjunto A' de arestas ou um subconjunto V' de vértices do grafo, isto é, $S \subseteq A' \subseteq A$ ou $S \subseteq V' \subseteq V$. Neste caso, temos o *problema do conjunto de retorno restrito a um subconjunto*. Outra restrição interessante pode ser feita na coleção \mathcal{C} de ciclos. Se restringirmos essa coleção aos ciclos de tamanho ímpar de um grafo não orientado G , o problema do conjunto de retorno transforma-se no *problema de bipartição de grafos*.

A seguir, apresentamos a definição formal de dois problemas, que são o alvo de estudo deste trabalho.

2.3.1 Problema do conjunto de vértices de retorno

Seja $G = (V, A)$ um grafo orientado (não-orientado) e $z : V \rightarrow \mathfrak{R}^+$ uma função positiva definida sobre V . Dizemos que um subconjunto $V' \subseteq V$ é um *conjunto de vértices de retorno* de G se V' contém pelo menos um vértice de cada ciclo de G . Em outras palavras, o subgrafo $G \setminus V'$ é acíclico. O custo de V' é denotado por $z(V') = \sum_{v \in V'} z(v)$.

O problema do conjunto de vértices de retorno valorado (WFVS) consiste em encontrar um conjunto de vértices de retorno de custo mínimo. Caso todos os vértices tenham o mesmo peso, chamamos o problema de *problema do conjunto de vértices de retorno não-valorado* (UFVS), ou simplesmente *problema do conjunto de vértices de retorno* (FVS).

O problema de decisão associado ao FVS é conhecido NP-completo tanto para grafos orientados como não-orientados [14, 28].

2.3.2 Problema do conjunto de arestas de retorno

Seja $G = (V, A)$ um grafo orientado (não-orientado) e $w : A \rightarrow \mathbb{R}^+$ uma função positiva definida sobre A . Dizemos que um subconjunto $A' \subseteq A$ é um *conjunto de arestas de retorno* de G se A' contém pelo menos um vértice de cada ciclo de G . Em outras palavras, o subgrafo $G \setminus A'$ é acíclico. O custo de A' é denotado por $w(A') = \sum_{e \in A'} w(e)$. O *problema do conjunto de arestas de retorno valorado* (WFAS) consiste em encontrar um conjunto de arestas de retorno de custo mínimo. Caso todas as arestas tenham o mesmo peso, chamamos o problema de *problema do conjunto de arestas de retorno não-valorado* (UFAS), ou simplesmente *problema do conjunto de arestas de retorno* (FAS).

Equivalência entre os problemas FVS e FAS

Even, Naor, Schieber e Sudan [7], mostraram que para grafos orientados os problemas de conjunto de retorno são equivalentes, particularmente o FAS e o FVS são equivalentes.

O algoritmo de redução do FAS para o FVS é apresentado na figura 2.10. A idéia do algoritmo é substituir cada aresta (v_0, v_2) por um novo vértice v_1 . Para cada par de arestas (v_i, v_j) e (v_j, v_k) , uma nova aresta (v_{ij}, v_{ik}) é criada, onde o vértice v_{ij} corresponde à aresta (v_i, v_j) e o vértice v_{ik} corresponde à aresta (v_j, v_k) . Cada vértice criado tem o mesmo peso da aresta correspondente. A figura 2.11 mostra a execução do algoritmo em um grafo e os conjuntos de retorno correspondentes.

O algoritmo de redução do FVS para o FAS é apresentado na figura 2.12. A idéia

Função FAS2FVS**Entrada:** Um grafo $G = (V, A)$ **Saída:** Um grafo $G' = (V', A')$ tal que para cada conjunto de arestas de retorno de G existe um conjunto de vértices de retorno equivalente (com mesmo custo) em G'

- 1: **para** cada aresta $e \in A$ **faça**
- 2: criar um vértice v_e em V'
- 3: **fim para**
- 4: **para** cada aresta $e_i = (v_i, v_j) \in A$ **faça**
- 5: **se** existe uma aresta $e_j = (v_j, v_k) \in A$ **então**
- 6: $E' = E' \cup (e_i, e_j)$
- 7: **fim se**
- 8: **fim para**
- 9: **retornar** o grafo $G' = (V', A')$

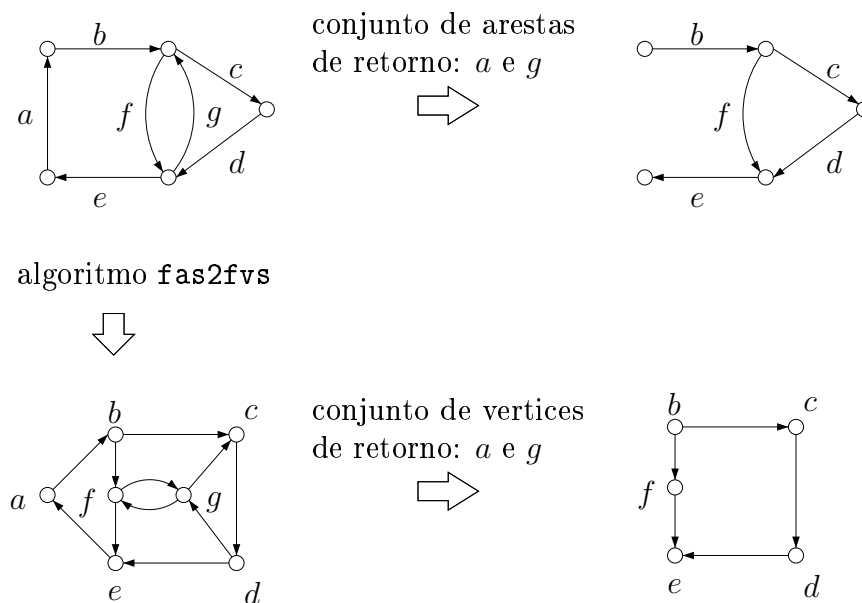
Figura 2.10: FAS \Rightarrow FVS

Figura 2.11: Exemplo de transformação do FAS para FVS.

Função FVS2FAS**Entrada:** Um grafo $G = (V, A)$ **Saída:** Um grafo $G' = (V', A')$ tal que para cada conjunto de vértices de retorno de G existe um conjunto de arestas de retorno equivalente (com mesmo custo) em G'

- 1: **para** cada vértice $v \in V$ **faça**
- 2: criar dois vértices v_1, v_2 em V'
- 3: criar uma aresta (v_1, v_2) em A' com o mesmo peso de v
- 4: $\text{map1}[v] \leftarrow v_1$ {map1 e map2 são vetores associativos $V \rightarrow V'$ }
- 5: $\text{map2}[v] \leftarrow v_2$
- 6: **fim para**
- 7: **para** cada aresta $(u, v) \in A$ **faça**
- 8: criar a aresta $(\text{map2}[u], \text{map1}[v])$ em A' com peso infinito
- 9: **fim para**
- 10: **retornar** o grafo $G' = (V', A')$

Figura 2.12: FVS \Rightarrow FAS

do algoritmo é substituir cada vértice v_0 por uma nova aresta (v_1, v_2) e dois vértices v_1 e v_2 . As arestas que entram em v_0 passam a ser as arestas que entram em v_1 , enquanto as arestas que saem de v_0 passam a ser as que saem de v_2 . O peso associado as novas arestas são os pesos dos vértices correspondentes. As arestas antigas têm peso infinito. A figura 2.13 mostra a execução do algoritmo em um grafo e os conjuntos de retorno correspondentes.

Da mesma forma que para FVS o problema de decisão associado ao FAS é NP-completo. Assim, tanto para o FAS como para o FVS evitamos algoritmos exatos, dando preferência à algoritmos aproximativos ou heurísticas práticas. No capítulo seguinte apresentamos quatro algoritmos para encontrar conjuntos de retorno.

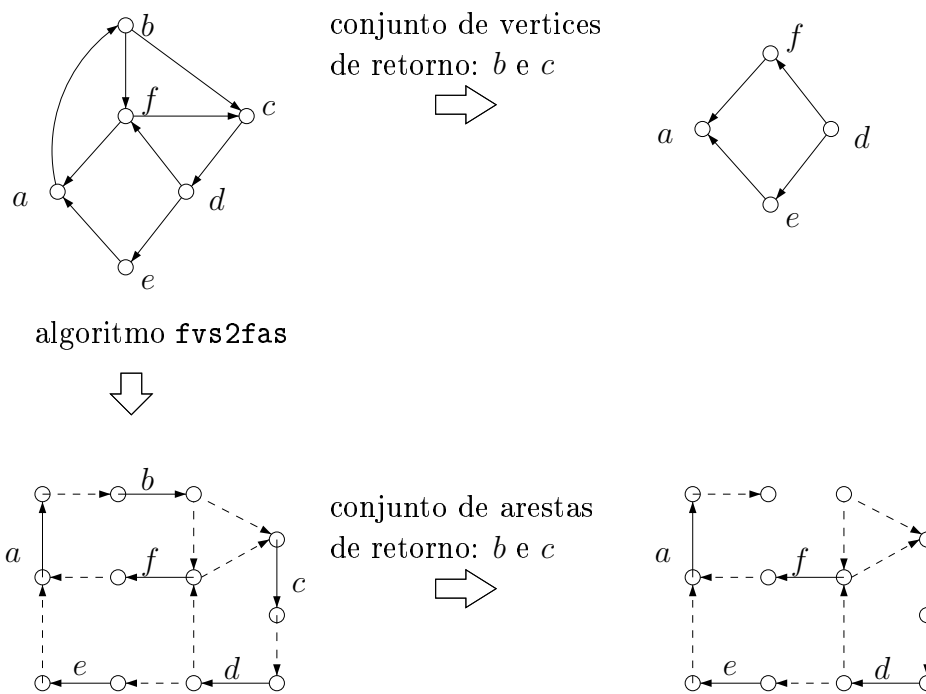


Figura 2.13: Exemplo de transformação do FVS para FAS.

3 *Trabalhos relacionados*

Segundo Festa, Pardalos e Resende em [9], apesar dos algoritmos aproximativos garantirem uma certa qualidade de solução, para muitos problemas práticos do mundo real, métodos heurísticos podem levar à soluções melhores em quantidade razoável de tempo e CPU. Nesta dissertação nos concentraremos em algoritmos heurísticos. Neste mesmo trabalho (de Festa, Pardalos e Resende [9]) é apresentada também uma revisão de algoritmos aproximativos para problemas do conjunto de retorno, que no melhor do nosso conhecimento é bastante completa. Por esta razão, não a reproduzimos aqui.

Neste capítulo apresentamos três algoritmos heurísticos para resolver problemas de conjunto de retorno. O primeiro é uma heurística simples para resolver o FAS [5]. Este é um dos primeiros algoritmos encontrado na literatura. Em seguida, apresentamos a aplicação da meta-heurística *evolução estocástica* para resolver o FAS [27]. Este algoritmo é baseado em uma estratégia dividir para conquistar. Por último, apresentamos a aplicação da meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedures*) para resolver o problema FVS. Também apresentamos um algoritmo polinomial exato para resolver o FAS restrito a grafos planares.

Os dois algoritmos para resolver o FAS utilizam a relação entre ordenação de vértices e conjunto de arestas de retorno. Esta relação é descrita a seguir.

3.1 Relação entre ordenação de vértices e conjunto de arestas de retorno

Seja $l : V \rightarrow N$ uma ordenação qualquer dos vértices de um grafo $G = (V, A)$. Uma aresta (u, v) nessa ordenação é dita estar *para frente* se $l(u) < l(v)$, caso contrário ($l(u) > l(v)$) esta aresta é dita estar *para trás*. Uma ordenação qualquer dos vértices determina dois conjunto de arestas de retorno: o conjunto de arestas para frente e o conjunto de arestas para trás, e, portanto, todo grafo admite pelo menos um conjunto de arestas de retorno com no máximo $\frac{1}{2}|A|$.

O primeiro a estudar a relação entre ordenação de vértices e conjunto de arestas de retorno foi Younger [29] que estabeleceu o seguinte:

1. Existe um ordenação de vértices tal que o conjunto de arestas para trás forma um conjunto de arestas de retorno de custo mínimo. Tal ordenação é chamada de *ordenação ótima*.
2. Um *subgrafo consecutivo* de um grafo $G = (V, A)$ com respeito a uma ordenação de vértices é um subgrafo induzido por um subconjunto de vértices consecutivos nesta ordenação. Dado dois subgrafos G_1 e G_2 de um grafo $G = (V, A)$, definimos como $N(G_1, G_2)$ o número de arestas que saem de um vértice em G_1 e chegam em um vértice em G_2 . Se G_1 e G_2 são dois subgrafos consecutivos de G com respeito a uma ordenação ótima de tal maneira que o último vértice de G_1 precede o primeiro vértice de G_2 , então $N(G_1, G_2) \geq N(G_2, G_1)$.
3. O conjunto de todas as arestas para trás com respeito a uma ordenação ótima de um grafo $G(V, A)$ deve ser minimal.
4. Se H é um subgrafo consecutivo com respeito a uma ordenação ótima de um grafo G , então a restrição da ordenação ótima aos vértices de H é uma ordenação ótima para H .

Função ELSFAS**Entrada:** um grafo orientado $G = (V, A)$ **Saída:** um conjunto de arestas de retorno de G 1: $S_1 = \emptyset$ 2: $S_2 = \emptyset$ 3: **enquanto** ($G \neq \emptyset$) **faça**4: **se** existe sumidouro u em G **então**5: colocar u no início de S_2 6: $v = u$ 7: **senão se** existe fonte u em G **então**8: colocar u no final de S_1 9: $v = u$ 10: **senão**11: escolher um vértice u tal que $d^+(u) - d^-(u)$ é máximo12: colocar u no começo de S_1 13: $v = u$ 14: **fim se**15: remover v e todas suas arestas incidentes de G 16: **fim enquanto**17: concatenar S_1 e S_2 em uma lista S 18: **retornar** o conjunto de arestas para trás da ordenação S

Figura 3.1: Um algoritmo simples para resolver o FAS

A seguir, apresentamos dois algoritmos heurísticos que utilizam esta relação para encontrar um conjunto de arestas de retorno.

3.2 Uma heurística simples para resolver o FAS

Eades, Lin e Smyth [5] desenvolveram um algoritmo heurístico linear “guloso” que explora a relação entre ordenação de vértices e o conjunto de arestas de retorno. Intuitivamente o algoritmo irá ordenar os vértices pelo grau de saída procurando orientar o máximo possível as arestas para frente, após este passo o algoritmo toma as arestas para trás como conjunto de retorno.

A figura 3.1 mostra um pseudocódigo do algoritmo. Para diminuir a quantidade de arestas para trás, coloca-se os vértices que têm mais arestas saindo (possivelmente uma fonte) no final da lista S_1 e os vértices com mais arestas entrando (possivelmente um

sumidouro) no início da lista S_2 . No final do algoritmo a lista S_2 é concatenada com a lista S_1 . Esta ordenação intuitivamente fornece um “bom” conjunto de arestas de retorno.

O tamanho do conjunto de retorno encontrado pelo algoritmo é limitado por $\frac{|A|}{2} - \frac{|V|}{6}$. O critério guloso empregado na linha 11 foi melhorado posteriormente em [4]. Com esta melhoria o algoritmo encontra um conjunto de arestas de retorno com tamanho máximo $\frac{1}{4}|A|$ para grafos cúbicos.

3.3 Evolução estocástica aplicada ao FAS

Outro autor a explorar a relação entre ordenação de vértices e conjunto de arestas de retorno é Saab em [27]. O algoritmo dividir para conquistar proposto neste trabalho está fundamentado no resultado obtido por Younger [29]: Se H é um subgrafo consecutivo com respeito a uma ordenação ótima de um grafo G , então a restrição da ordenação ótima aos vértices de H é uma ordenação ótima para H .

Para construir uma estratégia dividir para conquistar para resolver o FAS, Saab preocupa-se em responder a seguinte questão: Quais arestas farão parte do conjunto de retorno após o passo da divisão? Para obter um bom algoritmo, em cada passo de divisão a quantidade de aresta incluídas no conjunto de retorno deve ser minimizada e, além disso, esta divisão deve ser feita de maneira global, considerando o grafo como um todo e não apenas alguns vértices e arestas em particular (evitando mínimos locais).

Levando em consideração o resultado de Younger e procurando construir um algoritmo que tenha as características descritas anteriormente, vamos supor a existência de um algoritmo que particione o conjunto de vértices V , de um grafo $G = (V, A)$, em dois subconjuntos V_1 e V_2 tal que exista uma ordenação ótima de G com todos os vértices de V_1 à esquerda dos vértices de V_2 . Então, podemos construir um conjunto de arestas de retorno mínimo de G pela união do conjunto de arestas de retorno mínimo de $G[V_1]$, do conjunto de arestas de retorno mínimo de $G[V_2]$ e $\{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}$. Acredita-se que tal algoritmo não exista, dado que o FAS é NP-completo. No entanto, Saab considera

um objetivo mais simples: procurar uma partição (V_1, V_2) tal que o cardinalidade dos conjuntos V_1 e V_2 sejam próximas e a cardinalidade do conjunto $\{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}$ seja minimizada (que são exatamente as características desejadas na etapa de divisão). Este problema é conhecido como biseção de grafos. Para formalizar esta idéia, vamos fazer algumas definições.

Uma partição de um grafo $G = (V, A)$ é um par ordenado (V_1, V_2) onde $V_1 \cup V_2 = V$ e $V_1 \cap V_2 = \emptyset$. O custo da partição (V_1, V_2) é definido como $\text{cost}(V_1, V_2) = |\{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}|$. Uma *biseção* de um grafo G é uma partição (V_1, V_2) , tal que $|V_1| \leq \alpha|V_2|$ e $|V_2| \leq \alpha|V_1|$, onde $1/2 \leq \alpha < 1$.

O *problema de biseção de grafo* (GB) consiste em, dado um grafo orientado $G = (V, A)$, encontrar um biseção de custo mínimo. Apesar deste problema ser NP-completo [27], Saab desenvolveu nos últimos anos algoritmos rápidos e eficientes para otimizá-lo.

Sabendo da existência de algoritmos eficientes para otimizar o GB e observando que uma aresta só pode fazer parte de um ciclo se a origem e o destino da aresta fazem parte da mesma componente fortemente conexa, Saab propôs o algoritmo apresentado na figura 3.2. Onde $\text{scc}(G)$ é uma função que recebe um grafo $G = (V, A)$ como entrada e retorna uma partição $P = \{S : S \subseteq V \text{ e } S \text{ induz uma componente fortemente conexa de } G\}$, e $\text{bisect}(G)$ é uma função que recebe um grafo $G = (V, A)$ como entrada e retorna um biseção (V_1, V_2) de G com pequeno custo.

O algoritmo **SaabFAS** retorna um conjunto de arestas de retorno F do grafo $G = (V, A)$. Se G é fortemente conexo, então F é computado como a união dos conjuntos de retorno de cada subgrafo induzido $G[S]$ onde $S \in P$. Se G não é fortemente conexo, então, a função **bisect** é usada para decompor o conjunto de vértices V em dois subconjuntos V_1 e V_2 de tamanhos semelhantes. O conjunto F é então computado como a união dos conjuntos de retorno F_1 do grafo $G[V_1]$, F_2 do grafo $G[V_2]$ e do conjunto $L = \{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}$. O algoritmo **SaabFAS** é eficiente porque o grafo de entrada é rapidamente decomposto em subgrafos menores, seja pela função **scc** ou pela função

```

Função SAABFAS
Entrada: Um grafo  $G = (V, A)$ 
Saída: Um conjunto de arestas de retorno de  $G$ 
1:  $P \leftarrow \text{scc}(G)$ 
2: se  $P$  tem apenas um elemento  $\{G \text{ é fortemente conexo}\}$  então
3:   se  $|V| \leq 1$  então
4:     retornar  $\emptyset$ 
5:   senão
6:      $(V_1, V_2) \leftarrow \text{bisect}(G)$ 
7:      $F \leftarrow \text{SaabFas}(G[V_1]) \cup \text{SaabFas}(G[V_2]) \cup \{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}$ 
8:   fim se
9: senão
10:   $F \leftarrow \emptyset$ 
11:  para cada  $S \in P$  faça
12:     $F \leftarrow F \cup \text{SaabFAS}(G[S])$ 
13:  fim para
14: fim se
15: retornar  $F$ 

```

Figura 3.2: Algoritmo proposto por Saab para resolver o FAS

`bisect`.

Saab utiliza duas heurísticas para implementar a função `bisect`. Uma baseada em agrupamento dinâmico (*Dynamic Clustering* – DC) e outra baseada em evolução estocástica (*Stochastic Evolution* – SE). Nos resultados computacionais apresentados por Saab, a heurística DC apresentou resultados um pouco melhores que a heurística SE. No entanto, a heurística DC é muito mais complicada e difícil de ser implementada. Apresentamos neste trabalho apenas a heurística SE.

Evolução estocástica

Evolução Estocástica (SE) é uma metodologia geral para otimização combinatória (meta-heurística). Esta metodologia melhora iterativamente uma solução inicial através de perturbações locais chamadas de *movimentos*.

Seja (V_1, V_2) uma partição do conjunto de vértices de um grafo $G = (V, A)$. Esta partição pode ser modificada movendo um vértice de um subconjunto para outro. Vamos

```

Função PERTURB
Entrada:  $V, V_1, V_2, p$ 
Saída: modificação da partição  $(V_1, V_2)$ 

1:  $S_1 \leftarrow \emptyset$ 
2:  $S_2 \leftarrow \emptyset$ 
3: para cada  $v \in V$  faça
4:     se  $\text{gain}(v) > \text{randint}(p, 0)$  então
5:          $\text{move}(i)$ 
6:         se  $i \in V_1$  então
7:             adicionar  $i$  em  $S_1$ 
8:         senão
9:             adicionar  $i$  em  $S_2$ 
10:        fim se
11:    fim se
12: fim para
13: se  $|V_1| > |V_2|$  então
14:      $j \leftarrow 1$ 
15: senão
16:      $j \leftarrow 2$ 
17: fim se
18: enquanto  $|V_j| > \alpha|V|$  faça
19:     remover o vértice  $i$  de  $S_j$ 
20:      $\text{move}(i)$ 
21: fim enquanto

```

Figura 3.3: Função perturb

definir $\text{move}(i)$ como a função que move o vértice $i \in V$ de seu subconjunto corrente para o subconjunto complementar na partição (V_1, V_2) . A alteração no custo de (V_1, V_2) depois que $\text{move}(i)$ é executada é dada pela função $\text{gain}(i)$ que é definida como:

$$\text{gain}(i) = \begin{cases} \text{cost}(V_1, V_2) - \text{cost}(V_1 \setminus \{i\}, V_2 \cup \{i\}) & \text{se } i \in V_1 \\ \text{cost}(V_1, V_2) - \text{cost}(V_1 \cup \{i\}, V_2 \setminus \{i\}) & \text{se } i \in V_2 \end{cases}$$

Definimos $\text{randint}(l, h)$ como uma função que retorna um número aleatório no intervalo $[l, h]$. A figura 3.3 mostra a função perturb que “tenta” melhorar um biseção inicial (V_1, V_2) realizando movimentos através da função move. A função perturb usa um parâmetro $p \leq 0$ e duas pilhas S_1 e S_2 para armazenar os vértices movidos de V_2 para V_1 e de V_1 para V_2 , respectivamente.

Observe que como $p \leq 0$, a função `move` é sempre executada quando $\text{gain}(i) > 0$, ou seja, se o movimento diminui o custo da partição, ele é realizado. Por outro lado, quando $\text{gain}(i) \leq 0$, a execução da função `move` depende do número gerado por `randint` e do valor de p , ou seja, mesmo que o custo da partição aumente, o movimento pode ser realizado. Isto é interessante para evitar mínimos locais e procurar por novas partições. Quando menor o valor de p maior será a região explorada. Por fim, possíveis correções no balanceamento do conjunto V_1 e V_2 são realizadas retornando os últimos vértices movidos para o conjunto menor.

Agora podemos descrever o algoritmo proposto por Saab para resolver o GB utilizando evolução estocástica. O algoritmo é apresentado na figura 3.4.

O algoritmo `bisect` requer três parâmetros: R , p_0 , δ . O parâmetro R controla a quantidade de interações do algoritmo. Em cada interação, se a bisecção é melhorada o contador é diminuído de R , permitindo a realização de mais interações. Caso contrário o contador é aumentado de 1. O parâmetro p_0 inicializa a variável p que controla a realização dos movimentos na função `perturb`. O parâmetro δ é utilizado para diminuir o valor de p . Quando não houver melhora na bisecção, o valor de p é diminuído de δ para permitir que mais movimentos sejam realizados na função `perturb`. Quando a função `perturb` melhora o valor da bisecção, o valor p_0 é atribuído à variável p (volta para o valor inicial). No final do algoritmo a melhor bisecção encontrada é retornada.

A seguir, apresentamos um algoritmo para resolver o FVS que utiliza uma outra abordagem.

3.4 GRASP aplicado ao FVS

Greedy Randomized Adaptive Search Procedure (GRASP) é uma meta-heurística (multi início) que vem sendo utilizada com sucesso para resolver vários problemas de otimização combinatória [8, 24, 25, 26]. GRASP é constituído de duas fases: fase de construção e fase de busca local. Na fase de construção, uma solução viável aproximada para o problema

Função BISECT**Entrada:** um grafo $G = (V, A)$, três parâmetros de controle: R , p_0 e δ **Saída:** uma biseção (V_1, V_2) de baixo custo

```

1:  $(V_1, V_2) \leftarrow$  uma biseção inicial aleatória de  $G$ 
2:  $(B_1, B_2) \leftarrow (V_1, V_2)$  {salva a melhor biseção}
3:  $p \leftarrow p_0$  {valor inicial para o parâmetro p}
4: definir o valor do parâmetro  $R$ 
5:  $counter \leftarrow 0$ 
6: repetir
7:    $C_{pre} \leftarrow \text{cost}(V_1, V_2)$ 
8:   perturb $(V, V_1, V_2, p)$ 
9:    $C_{pos} \leftarrow \text{cost}(V_1, V_2)$ 
10:  se  $C_{pos} < C_{pre}$  então
11:     $(B_1, B_2) \leftarrow (V_1, V_2)$  {salva a melhor biseção}
12:     $counter \leftarrow counter + R$  {permite mais iterações}
13:  senão
14:     $counter \leftarrow counter + 1$ 
15:  fim se
16:  se  $C_{pos} = C_{pre}$  então
17:     $p \leftarrow p - \delta$  {diminui  $p$  para permitir mais movimentos de vértices na
    função perturb}
18:  senão
19:     $p \leftarrow p_0$  {restaura o valor inicial de  $p$ }
20:  fim se
21: até que  $counter > R$ 
22: retornar  $(B_1, B_2)$ 

```

Figura 3.4: Função bisect usando evolução estocástica

Função GRASP**Entrada:** uma instância de um problema**Saída:** uma solução viável aproximada para o problema

```

1:  $BestSolution \leftarrow \emptyset$ 
2: para  $k \leftarrow 1$  até MaxIter faça
3:    $x \leftarrow \text{ConstructGreedyRandomizedSolution}()$ 
4:    $x \leftarrow \text{LocalSearch}(x)$ 
5:    $\text{UpdateSolution}(BestSolution, x)$ 
6: fim para
7: retornar  $BestSolution$ 

```

Figura 3.5: Esboço geral da meta-heurística GRASP

é construída interativamente. Um elemento por vez é escolhido aleatoriamente de uma lista restrita de candidatos (candidatos - *restrict candidate list*), cujos elementos estão ordenados segundo um critério guloso. A solução construída geralmente não é ótima em relação a uma vizinhança adotada. A fase de busca local tenta melhorar a solução construída e encontrar um ótimo local (em relação à vizinhança adotada). Estas duas fases são executadas diversas vezes e a melhor solução gerada é tida como uma boa aproximação. Veja o esboço geral do GRASP na figura 3.5.

Para aplicar o GRASP na resolução de um problema, deve-se especificar a fase de construção, fase de busca local e qualquer alteração no procedimento geral da meta-heurística. Pardalos, Qian e Resende [23] utilizaram o GRASP para resolver o FVS. Para melhorar a eficiência do algoritmo os autores utilizam técnicas de redução de problema, tanto na fase de construção, como na busca local. No contexto do FVS, uma *redução* de um grafo G é a remoção de vértices e/ou arestas de G tal que, o grafo original e o novo grafo têm o mesmo conjunto de vértices de retorno. Dado um grafo $G = (V, A)$, quatro reduções são definidas:

1. **out0, in0.** Se $\text{out}(v) = 0$ ou $\text{in}(v) = 0$ para algum $v \in V$ ¹, então

- (a) $V = V \setminus \{v\}$

¹No trabalho de Pardalos, Qian e Resende [23] existe um erro na descrição da redução 1: os autores dizem que o vértice v faz parte do conjunto de retorno.

- (b) $A = A \setminus \{(x, y) \mid x = v \text{ ou } y = v\}$
2. **loop**. Se $(v, v) \in A$ para algum $v \in V$, então
- (a) v faz parte do conjunto de retorno
- (b) $V = V \setminus \{v\}$
- (c) $A = A \setminus \{(v, u) \text{ ou } (u, v), \text{ para } \forall u \in V\}$
3. **in1**. Se $\text{in}(v) = 1$ e $(u, v) \in A$, então
- (a) $V = V \setminus \{v\}$
- (b) $\text{out}(u) = \text{out}(u) \cup \text{out}(v)$
- (c) $A = A \cup \{(u, w) \mid w \in \text{out}(v)\} \setminus \{(v, w) \mid w \in \text{out}(v)\}$
4. **out1**. Se $\text{out}(v) = 1$ e $(v, u) \in A$, então
- (a) $V = V \setminus \{v\}$
- (b) $\text{in}(v) = \text{in}(v) \cup \text{in}(u)$
- (c) $A = A \cup \{(w, u) \mid w \in \text{in}(v)\} \setminus \{(w, v) \mid w \in \text{in}(v)\}$

A redução 1 consiste em remover do grafo os vértices que não têm arestas saindo ou arestas entrando, ou seja, remover do grafo aqueles vértices que não podem fazer parte de nenhum ciclo. A redução 2, por outro lado, consiste em remover do grafo os vértices que contém laços, isto é, que fazem parte de um ciclo. As reduções 3 e 4 têm um aspecto um pouco diferente: a redução 3 é aplicada nos vértices que têm apenas uma aresta entrando e a redução 4 em vértices que têm apenas uma aresta saindo. Em ambos os casos, o vértice é removido do grafo e as arestas são organizadas de forma a manter as informações dos ciclos. Note que as reduções podem ser aplicadas em qualquer ordem e repetidamente sobre um determinado grafo.

A figura 3.6 mostra a função `ReduceInstanceSize` que aplica as reduções interativamente em um dado grafo. A função retorna o grafo reduzido e um conjunto S de vértices

Função REDUCEINSTANCE SIZE**Entrada:** um grafo G **Saída:** um grafo reduzido G' e um conjunto S de vértices

- 1: aplicar redução 1
- 2: aplicar redução 2
- 3: aplicar redução 3
- 4: aplicar redução 4
- 5: **enquanto** pelo menos uma redução for aplicada com sucesso **faça**
- 6: aplicar redução 1
- 7: aplicar redução 2
- 8: aplicar redução 3
- 9: aplicar redução 4
- 10: **fim enquanto**

Figura 3.6: Função ReduceInstanceSize

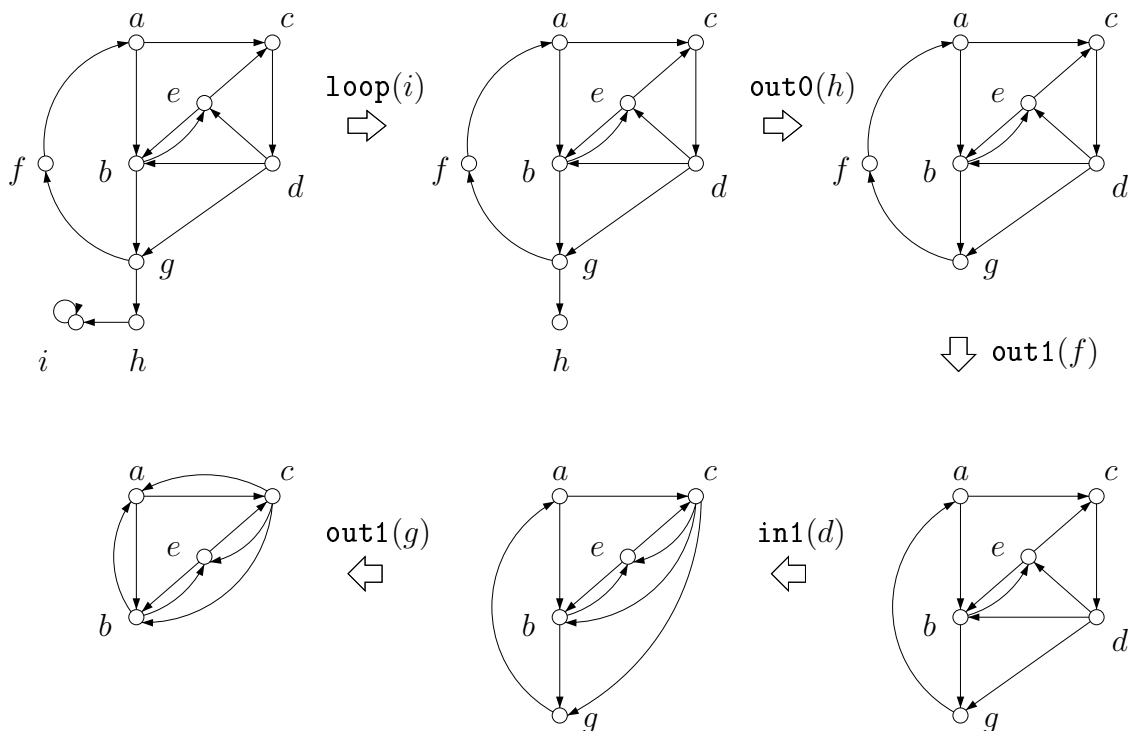


Figura 3.7: Exemplo da aplicação da função ReduceInstanceSize.

que continham laços em algum momento durante a execução da função, isto é, os vértices removidos devido à redução 2. A figura 3.7 mostra um exemplo da execução da função `ReduceInstanceSize`.

Fase de construção

Para especificar a fase de construção é preciso uma função gulosa adaptativa, um mecanismo para construir a lista RCL e um procedimento probabilístico de seleção.

A função gulosa deve estar associada a quanto um determinado vértice contribui para a função objetivo. No caso do FVS a seleção de um vértice deve reduzir os ciclos do grafo. Intuitivamente, quando maior o grau de um vértice, maior a probabilidade de que a remoção deste vértice diminua mais ciclos do grafo. Pardalos, Qian e Resende propõem três função que expressam esta noção:

1. $G_A(v) = |\text{in}(v)| + |\text{out}(v)|$
2. $G_B(v) = |\text{in}(v)| * |\text{out}(v)|$
3. $G_C(v) = \max(|\text{in}(v)|, |\text{out}(v)|)$

A função G_A considera pesos iguais para o grau de saída e o grau de entrada, a função G_B favorece o balanceamento dos graus de entrada e saída e a função G_C considera apenas o maior valor entre os dois graus. Pardalos, Qian e Resende em experimentos computacionais observaram que a função G_B produz melhores resultados. Neste trabalho chamamos a função G_B de **greedy**.

Para especificar o mecanismo de construção da lista RCL, vamos fazer duas definições. Dado um grafo $G = (V, A)$, definimos $gmin = \min_{v \in V} \text{greedy}(v)$ e $gmax = \max_{v \in V} \text{greedy}(v)$. A lista restrita de candidados é o conjunto de vértices $\text{RCL} = \{v \in V \mid \text{greedy}(v) \geq gmin + \alpha \cdot (gmax - gmin)\}$, onde $0 \leq \alpha \leq 1$.

A fase de construção é definida no algoritmo da figura 3.8. O laço principal do algoritmo, definido entre as linhas 4 e 11, é executado enquanto há vértices no grafo. Neste

```

Função CONSTRUCTGREEDYRANDOMIZEDSOLUTION
Entrada: uma grafo  $G = (V, A)$ ,  $\alpha$ 
Saída: um conjunto de vértices de retorno de  $G$ 

1:  $S \leftarrow \emptyset$ 
2:  $(G', S') = \text{ReduceInstanceSize}(G)$ 
3:  $S \leftarrow S \cup S'$ 

4: enquanto  $G' \neq \emptyset$  faça
5:    $rcl \leftarrow \text{MakeRCL}(G', \alpha)$ 
6:    $v \leftarrow \text{SelectVertex}(rcl)$ 
7:    $S \leftarrow S \cup \{v\}$ 
8:    $G' \leftarrow G' \setminus v$  {remover  $v$  e todas as arestas adjacentes a  $v$  em  $G'$ }
9:    $(G', S') = \text{ReduceInstanceSize}(G)$ 
10:   $S \leftarrow S \cup S'$ 
11: fim enquanto
12: retornar  $S$ 

```

Figura 3.8: Função ConstructGreedyRandomizedSolution

laço, uma lista RCL é criada, em seguida um vértice é selecionado aleatoriamente desta lista e então adicionado ao conjunto S , que representa o conjunto de retorno. Em seguida o grafo é atualizado, isto é, o vértice selecionado é removido do grafo, e a função `ReduceInstanceSize` é chamada. A função `ReduceInstanceSize` é utilizada para diminuir o tamanho do grafo, acelerando assim o processo.

Fase de busca local

Ao contrário da simplicidade da fase de construção, a busca local empregada por Pardalos, Qian e Resende é um pouco complicada. A dificuldade associada à busca local é verificar se um dado grafo é acíclico ou não. Os autores restringem a busca local a uma tentativa de eliminar vértices redundantes, ou seja, encontrar um conjunto de retorno minimal. A figura 3.9 mostra o pseudocódigo da busca local, onde S representa o conjunto de vértices de retorno e s_i representa o i -ésimo elemento de S .

A função `LocalSearch` verifica se cada vértice do conjunto de retorno é redundante. Isto é feito removendo um vértice $s_i \in S$ e aplicando o algoritmo de redução (função `ReduceInstanceSize`) no grafo induzido $G[(V \setminus S) \cup s_i]$ (linhas 6 á 9). Se o grafo reduzido

```

Função LOCALSEARCH
Entrada: uma grafo  $G = (V, A)$ , e um conjunto de vértices de retorno  $S \in V$ 
Saída: um conjunto de vértices de retorno minimal de  $G$ 

1:  $flag \leftarrow 1$ 
2: enquanto  $flag = 1$  faça
3:    $flag \leftarrow 0$ 
4:    $i \leftarrow 1$ 

5:   enquanto  $i \leq |S|$  e  $flag = 0$  faça
6:      $V' \leftarrow (V \setminus S) \cup s_i$ 
7:      $A' \leftarrow \{(v, w) \mid v, w \in V'\}$ 
8:      $G' \leftarrow (V', A')$ 
9:      $(G', S') \leftarrow \text{ReduceInstanceSize}(G')$ 
10:    se  $V' = \emptyset$  e  $S' = \emptyset$  então
11:       $S \leftarrow S \setminus s_i$ 
12:       $flag \leftarrow 1$ 
13:    fim se
14:  fim enquanto
15: fim enquanto
16: retornar  $S$ 

```

Figura 3.9: Função LocalSearch

for vazio, então ele não continha ciclos e o vértice s_i é redundante. Caso contrário, o grafo contém ciclos e o vértice s_i não é redundante.

Na seção seguinte, apresentamos um algoritmo exato polinomial para resolver o FAS para grafos planares.

3.5 Solução do FAS para grafos planares

Dado um grafo orientado $G = (V, A)$ e uma função inteira não negativa $d(d : A \rightarrow \mathbb{Z}^+)$ sobre o conjunto de arestas, dizemos que G é *fortemente conexo* se para quaisquer vértices x e y de V , existe um caminho orientado de x para y . Dizemos que um subconjunto $X \subseteq V$ é um *núcleo* (*kernel*) se não existe aresta saindo de X . O conjunto não-vazio $D(X)$ de arestas entrando em X é chamado de *corte direcionado* ou *dicut*. É sabido que um grafo é fortemente conexo se, e somente se, não contém núcleos.

Um conjunto de arestas $B \subseteq A$ é chamado de *cobertura* se cada dicut contém pelo

menos um elemento de B . As arestas de B são chamadas de arestas *azuis*, e as demais arestas do grafo de *brancas*. O custo de B é dado por $d(B) = \sum(d(e) : e \in B)$. Note que B é uma cobertura se, e somente se, a contração dos elementos de B torna o grafo fortemente conexo. O *problema da cobertura* consiste em encontrar uma cobertura de custo mínimo.

Existe uma relação entre coberturas e conjuntos de arestas de retorno. Esta relação é expressa na proposição 1.

Proposição 1. *Se B é uma cobertura de custo mínimo de um grafo orientado D , dual de um grafo orientado planar G , o conjunto F de arestas de G relacionadas com as arestas de B , forma um conjunto de arestas de retorno de custo mínimo do grafo G .*

Demonstração. Circuitos em desenhos de grafos planares orientados estão relacionados com dicuts no grafo dual correspondente. Para mais informações sobre grafos duais e suas propriedades veja [15]. □

O primeiro a resolver o problema de cobertura para grafos não-valorados (e por consequência o FAS para grafos planares) foi Lucchesi e Younger [21] em 1978. O resultado pode ser enunciado da seguinte forma:

Teorema 4. *A cardinalidade mínima de uma cobertura é igual ao número máximo de cortes direcionados disjuntos.*

O resultado para casos gerais (grafos valorados) foi obtido por Edmonds e Giles em [6]. Para enunciar o resultado precisamos de uma definição. Dado um grafo orientado $G = (V, A)$ e uma função não negativa d , definida sobre A , uma família \mathcal{K} de dicuts (não necessariamente distintos) é chamada de *d -independente* se nenhuma aresta e ocorre em mais que $d(e)$ membros de \mathcal{K} .

Teorema 5. *O custo mínimo τ_d de uma cobertura é igual a máxima cardinalidade ν_d de uma família d -independente de dicuts.*

Podemos expressar os teoremas 4 e 5 de outra maneira (útil para elaborar um algoritmo). Se uma cobertura B e uma família \mathcal{K} de núcleos satisfazem as condições:

- Toda aresta azul está em exatamente um dicut $D(X)$, $X \in \mathcal{K}$.
- Toda aresta branca está no máximo em um dicut $D(X)$, $X \in \mathcal{K}$.
- Existe exatamente uma aresta azul entrando em X para cada $X \in \mathcal{K}$.

então, B é uma cobertura de custo mínimo.

Frank em [12], apresenta uma prova algorítmica para os teoremas 4 e 5. Antes de apresentar o algoritmo de Frank, vamos fazer algumas definições.

Um núcleo X é dito ser *estrito* com respeito a um subconjunto $B \subseteq A$ se para cada aresta de B entrando em X , existe uma componente fracamente conexa em $V \setminus X$, ou seja, a aresta deixa a componente $V \setminus X$ e entra em X . Um núcleo X é dito ser *1-estrito* se X é estrito com respeito a um subconjunto B e apenas uma aresta de B entra em X . Definimos $R(x)$ como a intersecção de todos os núcleos estritos contendo o vértice x . Um potencial é uma função inteira definida sobre o conjunto de vértices de um grafo. Definimos como *diferencial* a função $\bar{d}(x, y) = d(x, y) - p(y) + p(x)$.

Frank mostrou que uma família \mathcal{K} e a cobertura B , que satisfazem o teorema 5, podem ser facilmente produzidas se uma cobertura B e um potencial p forem encontrados tal que o seguinte critério de otimalidade seja verdadeiro:

- a Para cada aresta azul (x, y) , $\bar{d}(x, y) \leq 0$.
- b Para cada aresta branca (x, y) , $\bar{d}(x, y) \geq 0$.
- c Para cada vértice y em $R(x)$, $p(y) \geq p(x)$.

Baseado neste critério de otimalidade, vamos supor a existência do seguinte algoritmo:

Entrada: Uma cobertura B , um potencial p , uma aresta azul (a, b) tal que os itens (b) e (c) do critério de otimalidade são satisfeitos e o item (a) não.

Saída: Uma cobertura B' e um potencial p' tal que (b) e (c) do critério de otimalidade seja satisfeito novamente e (a, b) não viola (a) e todas as arestas que violam (a) com respeito a B' e p' também violam (a) com respeito a B e p .

Se tal algoritmo existir e for repetido sucessivamente até que não exista mais arestas azuis violando (a), então, depois de no máximo $|B| \leq |V| - 1$ aplicações deste algoritmo, chegaremos a uma cobertura B e um potencial p que satisfazem os três critérios de otimalidade.

Para efetivamente descrever este algoritmo precisamos definir um grafo auxiliar $H = (V, A)$, onde A é constituído de três partes A_B , A_W e A_R , não necessariamente distintos. Note que H depende de G , B e p , e ainda, H pode conter múltiplas arestas.

$$A = \begin{cases} A_B & = (x, y) : (x, y) \in B, d(x, y) \geq 0 \\ A_W & = (y, x) : (y, x) \in E \setminus B, d(x, y) \leq 0 \\ A_R & = (x, y) : y \in R(x), p(y) = p(x) \end{cases}$$

Vamos tentar encontrar um caminho de b até a em H para um arco (a, b) que não satisfaça (a) do critério de otimalidade. Existem dois casos.

Caso 1. Não existe caminho de b até a em H . Neste caso, vamos definir $T : \{x : x \text{ pode ser alcançado a partir de } b \text{ em } H\}$ um conjunto de vértices. Se a não está em T , então mudar p da seguinte maneira:

$$p'(x) = \begin{cases} p(x) & \text{se } x \notin T \\ p(x) + \delta & \text{se } x \in T \end{cases}$$

Onde

$$\delta = \begin{cases} \delta_e & = d(a, b) \\ \delta_B & = \min\{-d(x, y) : (x, y) \in B, (x, y) \text{ entra em } T\} \\ \delta_W & = \min\{d(x, y) : (x, y) \in E \setminus B, (x, y) \text{ sai de } T\} \\ \delta_R & = \min\{p(y) - p(x) : x \in T, y \in R(x) \setminus T\} \end{cases}$$

Pela definição de δ obtemos que o novo $\bar{d}' = d(x, y) - p'(y) + p'(x)$ é:

$$\bar{d}'(x, y) = \begin{cases} d(x, y) - \delta & \text{se } (x, y) \text{ entra em } T \\ d(x, y) + \delta & \text{se } (x, y) \text{ sai de } T \\ d(x, y) & \text{caso contrário} \end{cases}$$

Temos agora que depois de no máximo $n - 1$ iterações, ou $\delta = \delta_e$ ou $a \in T$ (caso 2). Se esta interação for necessária, observamos que a definição de δ garante que H' contém pelo menos uma aresta deixando T (que está em A_B , A_W ou em A_R , de acordo com δ). Conseqüentemente o conjunto de vértices de T que podem ser alcançados por um caminho direcionado de b em H' inclui T .

Caso 2. Existe um caminho de b até a em H . Seja $F^* = \{(x, y) : (y, x) \in F\}$.

Seja U o menor caminho de b até a em H . Como $(a, b) \in A_B$, U e (a, b) formam um circuito direcionado C em H . Seja C_B e C_W o conjunto de arestas azuis e brancas de C respectivamente. Seja $B' = B \setminus C_B \cup C_W^*$

O algoritmo de Frank tem complexidade $O(n^5)$. Para grafo planares a complexidade é $O(n^4)$. Para prova de corretude do algoritmo veja [12].

Uma descrição formal do algoritmo é dada a seguir.

Passo 0

0.1 (Início) A cobertura B é o conjunto de arcos de uma árvore geradora e $p \equiv 0$.

0.2 Determinar $R(b)$ para todas as arestas e azuis que não satisfaçam (a) do critério de otimalidade.

0.3 Mudar a cor de cada aresta azul boa para branca.

Passo 1

1.1 Determinar $R(x)$ para todos $x \in V$.

1.2 Se cada aresta azul satisfaz (a) do critério de otimalidade: Parar. A cobertura corrente é ótima.

1.3 Selecionar uma aresta e violando (a) do critério de otimalidade.

1.4 Construir o grafo auxiliar H e tentar achar um caminho de b para a em H utilizando a técnica de rotulação de Ford e Fulkerson em [11]. No novo grafo auxiliar H' o conjunto de arestas expandidas por T é o mesmo de H . Além disso, a definição de δ garante que H' contém pelo menos uma aresta saindo de T (que esta em A_B , A_W ou A_R dependendo se δ é igual a δ_B , δ_W ou δ_R). Com isso, o conjunto T de vértices que podem ser alcançados por um caminho orientado de b em H' corretamente inclui T . (Note que neste caso, o novo grafo auxiliar resulta do grafo antigo adicionando algumas arestas saindo de T e removendo algumas arestas entrando em T). Assim, se $\delta < \delta_e$ ocorre (neste caso a partir de 2.2) nós podemos usar os rótulos calculados, mas não ainda removidos. Se este caminho U existir então ir para o passo 3.

Passo 2 (mudança do potencial). Seja T o conjunto de vértices rotulados. Calcular δ e fazer $p(x) = p(x) + \delta$ para $x \in T$.

2.1 se $\delta = \delta_e$ remover todos os rótulos e ir para 1.2

2.2 Ir para 1.4.

Passo 3 (mudança da cobertura). Seja $C = U + e$, C_B e C_w o conjunto de arestas azuis e brancas de C , respectivamente, fazer $B = B \setminus C_B \cup C_w^*$. Ir para 1.1.

Para que este algoritmo possa ser usado para resolver o FAS para grafos planares é necessário um procedimento para encontrar o grafo dual. Para construir um grafo dual é necessário uma imersão planar combinatorial. Por fim, para construir uma imersão combinatorial planar é necessário um algoritmo de teste de planaridade.

3.6 Algoritmos de teste de planaridade

O primeiro algoritmo linear de teste de planaridade é o de Hopcroft e Tarjan [16]. O algoritmo primeiramente faz a imersão de um ciclo C e então “quebra” o restante do grafo em uma seqüência de caminhos que podem ser adicionados do lado de dentro ou do lado de fora do ciclo inicial. Esta abordagem é conhecida como adição de caminhos, visto que em cada interação o algoritmo “tenta” fazer a imersão de um caminho na parte do grafo já processada.

O segundo método linear para teste de planaridade é o de Lempel, Even e Cederbaum [20]. O algoritmo inicialmente cria uma s, t -numeração para um grafo biconexo. Uma propriedade da s, t -numeração é que existe um caminho de vértices com numeração maior que leva todos os vértices ao vértice t , que tem a maior numeração. Assim, existe uma imersão \tilde{G}_k dos primeiros k vértices tal que, os vértices restantes ($k + 1$ até t) podem ser imersos em uma única face de \tilde{G}_k . Para alcançar complexidade linear, este algoritmo utiliza uma estrutura de dados chamada árvores-PQ, desenvolvida por Booth e Lueker [1]. Esta abordagem é conhecida como adição de vértices.

Estes dois algoritmos foram desenvolvidos inicialmente apenas para fazer o teste de planaridade. Posteriormente, outros autores modificaram os algoritmos para gerar uma imersão combinatorial planar. No entanto, os procedimentos são complicados.

Recentemente, um novo método baseado em adição de arestas foi proposto por Boyer e Myrvold [2]. Ao contrário das outras abordagens que utilizam estruturas de dados complicadas, este método utiliza apenas propriedades dos vértices e uma numeração de busca primeiro em profundidade. Além de realizar o teste de planaridade o algoritmo

também gera uma imersão combinatorial planar do grafo.

Pela sua simplicidade em relação às outras abordagens, o algoritmo implementado para executar os experimentos nesta dissertação foi o de Boyer e Myrvold [2].

Apesar do assunto de algoritmos de teste de planaridade ser vasto, não cabe a esta dissertação expô-los. Para uma revisão em português sobre teste de planaridade o leitor é convidado a ver [17, 22].

4 *Algoritmo proposto*

Neste capítulo propomos uma modificação no algoritmo de Saab [27].

4.1 **Abordagens anteriores**

No capítulo anterior apresentamos quatro algoritmos para resolver problemas de conjunto de retorno: uma heurística simples e um algoritmo dividir para conquistar para resolver o FAS para grafos gerais, um algoritmo polinomial exato para resolver o FAS restrito a grafos planares e a meta-heurística GRASP para resolver o FVS.

Apesar da abordagem GRASP (Pardalos, Qian e Resende [23]), descrita na seção 3.4, ter sido concebida inicialmente para resolver o FVS, a mesma também pode ser utilizada para resolver o FAS. Como apresentado na seção 2.3.2 o FAS e o FVS são equivalentes. Festa, Pardalos e Resende [10] utilizam o algoritmo da figura 2.12 e apresentam sub-rotinas em FORTRAN para resolver o FVS e o FAS.

Um ponto interessante do algoritmo de Pardalos, Qian e Resende é a utilização de técnicas de redução de problema durante as iterações do algoritmo. A idéia é remover vértices e/ou arestas do grafo que não podem influenciar na composição do conjunto de retorno. Esta idéia mostrou-se válida, tanto em relação ao tempo de processamento, quanto à qualidade de soluções. O que caracteriza a aplicação das técnicas de redução é o fato do algoritmo ser iterativo e a cada iteração um subgrafo ser processado, podendo desta forma, conter vértices e/ou arestas com propriedades diferentes da interação anterior.

O algoritmo de Eades, Lin e Smith [5] apresentado na seção 3.2, também utiliza,

de certa forma, operações de redução. A cada iteração o algoritmo dá prioridade na seleção de sumidouros e fontes, vértices cujas arestas adjacentes não podem fazer parte do conjunto de retorno.

O algoritmo desenvolvido por Saab [27], apresentado na seção 3.3, utiliza uma abordagem dividir para conquistar, onde cada passo de divisão é tratado como um problema de biseção. Uma das características deste algoritmo é a rápida redução do problema original em problemas menores. As divisões são realizadas recursivamente até subgrafos de 1 vértice, situação em que o problema de biseção é trivial.

Uma questão interessante a ser analisada em relação ao passo de divisão do algoritmo de Saab é a seguinte: O caso base do algoritmo pode influenciar a qualidade das soluções geradas?

As divisões são realizadas por duas funções: `scc` e `bisect`. A função `bisect` é heurística, e portanto, pode gerar soluções que não são ótimas. Se pudermos substituir a execução da função `bisect`, em algum nível da recursão, por uma função exata, intuitivamente poderíamos gerar soluções globais melhores, ou seja, o caso base do algoritmo pode, intuitivamente, influenciar a qualidade das soluções.

Para explorar esta observação propomos uma modificação no algoritmo de Saab: utilizar o algoritmo polinomial exato de Frank [12] (descrito na seção 3.5) para resolver o FAS para grafos planares, como caso base do passo de divisão.

Na figura 4.1 apresentamos o algoritmo de Saab modificado. Além do algoritmo de Frank, também utilizamos técnicas de redução de problema. Neste caso, a redução que pode ser aplicada é a remoção de vértices com grau de entrada ou grau de saída igual a zero, ou seja, vértices cujas arestas adjacentes não podem fazer parte do conjunto de retorno.

Para que o algoritmo de Frank possa ser aplicado o grafo deve ser planar. A função `planarity`, chamada na linha 10, testa se o grafo G é planar ou não. Se o grafo for planar, a função `PlanarFAS` é chamada, caso contrário, o algoritmo realiza mais uma etapa de

```

Função MSAABFAS
Entrada: Um grafo  $G = (V, A)$ 
Saída: Um conjunto de arestas de retorno de  $G$ 

1: enquanto existe vértice  $v \in V$  tal que  $|\text{in}(v) = 0|$  ou  $|\text{out}(v)| = 0$  faça
2:    $V \leftarrow V \setminus \{v\}$  {remover  $v$  e as arestas adjacentes}
3:    $A \leftarrow A \setminus \{e \mid e \in \text{in}(v) \text{ ou } e \in \text{out}(v)\}$ 
4: fim enquanto
5:  $P \leftarrow \text{scc}(G)$ 
6: se  $P$  tem apenas um elemento  $\{G \text{ é fortemente conexo}\}$  então
7:   se  $|V| \leq 1$  então
8:     retornar  $\emptyset$ 
9:   senão
10:     $\text{planar} \leftarrow \text{planarity}(G)$ 
11:    se  $\text{planar}$  então
12:       $F \leftarrow \text{PlanarFAS}(G)$ 
13:    senão
14:       $(V_1, V_2) \leftarrow \text{bisect}(G)$ 
15:       $F \leftarrow \text{MSaabFas}(G[V_1]) \cup \text{MSaabFas}(G[V_2]) \cup \{(i, j) \mid i \in V_2 \text{ e } j \in V_1\}$ 
16:    fim se
17:  fim se
18: senão
19:    $F \leftarrow \emptyset$ 
20:  para cada  $S \in P$  faça
21:     $F \leftarrow F \cup \text{MSaabFAS}(G[S])$ 
22:  fim para
23: fim se
24: retornar  $F$ 

```

Figura 4.1: Algoritmo proposto por Saab para resolver o FAS

divisão através da função `bisect`.

A função `PlanarFAS` encontrar um conjunto de retorno ótimo para o grafo planar de entrada. Para tanto, são necessários a construção do grafo dual, a execução do algoritmo de Frank, e por fim, o mapeamento da cobertura encontrada do grafo dual para o conjunto de arestas de retorno do grafo primal.

No próximo capítulo, apresentamos os resultados de experimentos computacionais entre os diversos algoritmos estudados e o algoritmo de Saab modificado.

5 *Resultados obtidos*

Neste capítulo apresentamos os resultados computacionais da execução dos algoritmos estudados nesta dissertação. Todos os algoritmos foram utilizados para resolver o FAS. A tabela 5.1 apresenta os algoritmos e configurações testados.

Nome	Descrição	Parâmetros
<code>sse</code>	Algoritmo de Saab, descrito na seção 3.3	$\alpha = 0.6$, $R = 10$, $p_0 = -1$ e $\delta = 2$
<code>sser</code>	Algoritmo de Saab com redução de problema, descrito no capítulo 4	Igual a <code>sse</code>
<code>ssep</code>	Algoritmo de Saab com algoritmo de Frank, descrito no capítulo 4	Igual a <code>sse</code>
<code>sserp</code>	Algoritmo de Saab com redução de problema e com algoritmo de Frank, descrito no capítulo 4	Igual a <code>sse</code>
<code>els</code>	Algoritmo de Eades, Lin e Smyth [5], descrito na seção 3.2	
<code>pqr</code>	Algoritmo de Pardalos, Qian e Resende [23], descrito na seção 3.4	$\alpha = 0.9$

Tabela 5.1: Algoritmos testados

Os algoritmos foram implementados usando a linguagem C++ e o compilador g++ com o parâmetro `-O2`. A biblioteca GTL (*Graph Template Library*¹) foi utilizada como suporte para a estrutura de dados de grafos. Os experimentos foram realizados em um PC Pentium III 750 Mhz com 512 MB de memória e sistema operacional Ubuntu Linux.

O conjunto de testes utilizado é o mesmo do trabalho de Pardalos, Qian e Resende [23]². O conjunto é constituído de 40 grafos, com número de vértices variando de 50 a 1000. Cada algoritmo, com exceção do `els`, foi executado 50 vezes.

¹<http://infosun.fmi.uni-passau.de/GTL/>

²Disponível em <http://www.research.att.com/~mgcr/data/gfsp-data.tar.gz>

Para cada conjunto de grafos com o mesmo número de vértices os seguintes resultados são apresentados:

- Tabela que apresenta a quantidade de arestas do grafo e o melhor resultado obtido, isto é, o menor conjunto de retorno. Para cada algoritmo, a tabela apresenta ainda a melhor solução alcançada pelo algoritmo, a iteração em que a solução foi alcançada e o tempo em segundos necessário para alcançá-la, respectivamente. Os valores em negrito destacados com * indicam que o algoritmo alcançou o melhor resultado dentre todos os algoritmos testados;
- Gráfico que apresenta o tempo médio da iteração de cada algoritmo;
- Gráfico que apresenta o tamanho do melhor conjunto de retorno encontrado por cada algoritmo.

Para facilitar a apresentação dos resultados agrupamos os algoritmos em dois grupos:

- grupo 1: algoritmo **sse** de Saab e as modificações propostas **sser**, **ssep** e **sserp**;
- grupo 2: algoritmos existentes na literatura **sse**, **els** e **pqr**.

A seguir apresentamos os resultados para cada conjunto de grafos com o mesmo número de vértices.

5.1 Grafos com 50 vértices

Na tabela 5.2 apresentamos os resultados para os algoritmos do grupo 1 e grafos de 50 vértices. O algoritmo **ssep** obteve 7 dos 10 melhores resultados enquanto os algoritmos **sse** e **sserp** alcançaram apenas 4. O tamanho dos conjuntos de retorno encontrados pelos algoritmos **sse** e **ssep** é apresentado graficamente na figura 5.1. O tempo de execução dos algoritmos **sse** e **ssep** é apresentado na figura 5.2. O algoritmo **sse** é ligeiramente mais rápido que o algoritmo **ssep**.

Na tabela 5.3 apresentamos os resultados para os algoritmos do grupo 2 e grafos de 50 vértices. A figura 5.3 mostra o melhor resultado para cada algoritmo do grupo 2 e a figura 5.4 o tempo médio das iterações. Para grafos com até 300 arestas o algoritmo `pqr` é mais rápido que o algoritmo `sse`, no entanto, a curva assintótica de tempo do algoritmo `pqr` cresce mais rápido que a do algoritmo `sse`. Em termos de qualidade das soluções, o algoritmo `sse` alcançou 4 dos melhores resultados enquanto o algoritmo `pqr` obteve 3. Apesar do algoritmo `els` ser o mais rápido, este algoritmo alcançou apenas uma melhor solução.

5.2 Grafos com 100 vértices

Na tabela 5.4 apresentamos os resultados para os algoritmos do grupo 1 e grafos de 100 vértices. O algoritmo `ssep` obteve 6 dos 10 melhores resultados enquanto o algoritmo `sse` alcançou apenas 3. O tamanho dos conjuntos de retorno encontrados pelos algoritmos `sse` e `ssep` é apresentado graficamente na figura 5.5 e o tempo médio das iterações é apresentado na figura 5.6. A diferença de tempo de execução dos algoritmos `sse` e `ssep` é pequena.

Na tabela 5.5 apresentamos os resultados para os algoritmos do grupo 2 e grafos de 100 vértices. O algoritmo `pqr` alcançou 4 das melhores soluções, enquanto o algoritmo `sse` alcançou 3. Na figura 5.7 é apresentado o gráfico das melhores soluções encontradas pelos algoritmos do grupo 2. O tempo médio das iterações é apresentado na figura 5.8. O algoritmo `els` é o mais rápido, porém, não alcança boas soluções. O algoritmo `pqr` é muito lento para grafos densos.

5.3 Grafos com 500 vértices

Na tabela 5.6 apresentamos os resultados para os algoritmos do grupo 1 e grafos de 500 vértices. Ao contrário dos resultados para grafos de 50 e 100 vértices, para grafos de 500 vértices o algoritmo `sse` obteve melhores soluções que o algoritmo `ssep`. O tempo

médio de execução dos algoritmos **sse** e **ssep** (figura 5.10) é semelhante.

Na tabela 5.7 apresentamos os resultados para os algoritmos do grupo 2 e grafos de 500 vértices. O algoritmo **sse** obteve melhores soluções que o algoritmo **pqr** (figura 5.11), além disso, o tempo de execução do algoritmo **sse** é muito menor (figura 5.12).

5.4 Grafos com 1000 vértices

Na tabela 5.8 apresentamos os resultados para os algoritmos do grupo 1 e grafos de 1000 vértices. O algoritmo **sse** alcançou 6 dos melhores resultados enquanto o algoritmo **ssep** alcançou 5, no entanto, a soma dos resultados do algoritmo **ssep** é melhor que a do algoritmo **sse**. Tanto em relação à qualidade das soluções quanto ao tempo de execução não é possível afirmar que um algoritmo foi melhor que outro (figuras 5.13 e 5.14).

Na tabela 5.9 apresentamos os resultados para os algoritmos do grupo 2 e grafos de 1000 vértices. Novamente as soluções e o tempo de execução do algoritmo **sse** são melhores que os do algoritmo **pqr** (figuras 5.15 e 5.16).

A	M	sse			ssep			ssep			ssep		
		M	IT	T	M	IT	T	M	IT	T	M	IT	T
100	6	6*	13	0.34	6*	4	0.04	6*	8	0.59	6*	8	0.14
150	19	19*	28	0.92	19*	6	0.11	19*	30	2.21	19*	3	0.11
200	35	36	22	0.90	37	17	0.49	35*	9	0.59	36	41	2.21
250	45	48	35	1.56	48	7	0.25	45*	22	1.59	46	14	0.84
300	62	65	13	0.66	64	3	0.13	62*	9	0.72	63	44	3.51
500	149	151	15	1.33	150	50	4.08	149*	32	5.05	149*	14	1.92
600	200	208	13	1.50	208	30	3.20	206	43	6.97	202	28	4.68
700	236	239	34	4.77	238	11	1.48	236*	49	9.87	237	37	7.58
800	288	288*	32	5.60	295	18	3.05	289	47	10.86	288*	32	7.57
900	315	315*	48	10.30	321	33	6.82	318	34	9.25	321	15	4.34
soma		1375	253	27.89	1386	179	19.66	1365	283	47.71	1367	236	32.90

Tabela 5.2: Resultado para grafos com 50 vértices: sse, ssep, ssep e ssep

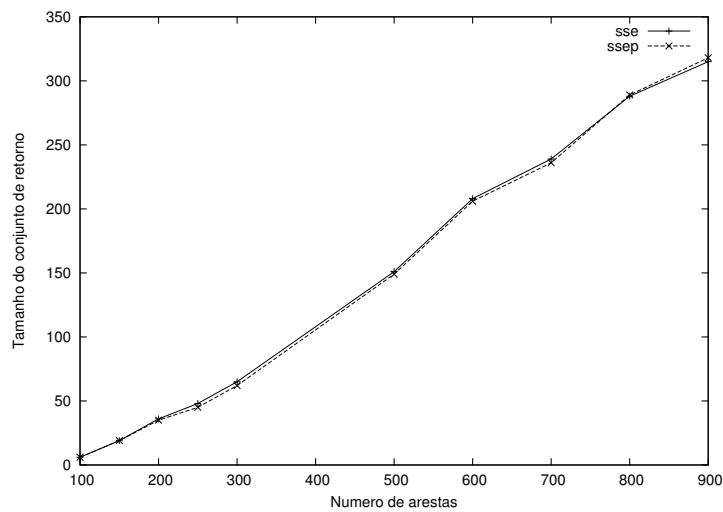


Figura 5.1: Tamanho do conjunto de retorno para grafos com 50 vértices: sse e ssep

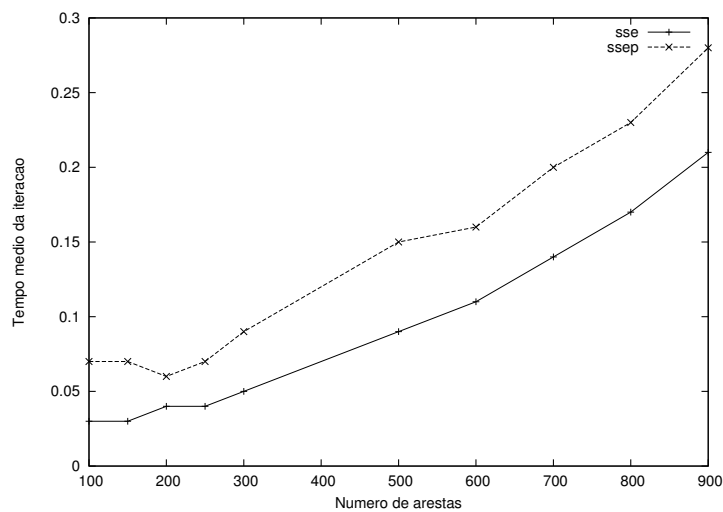


Figura 5.2: Tempo médio da iteração para grafos com 50 vértices: sse e ssep

$ A $	M	sse			els			pqr		
		M	IT	T	M	IT	T	M	IT	T
100	6	6*	13	0.34	6*	1	0.00	6*	1	0.00
150	19	19*	28	0.92	26	1	0.01	20	1	0.01
200	35	36	22	0.90	41	1	0.01	35*	18	0.24
250	45	48	35	1.56	55	1	0.01	46	6	0.13
300	62	65	13	0.66	74	1	0.01	64	14	0.46
500	149	151	15	1.33	160	1	0.03	151	6	0.63
600	200	208	13	1.50	214	1	0.05	200*	9	1.54
700	236	239	34	4.77	257	1	0.07	239	48	12.28
800	288	288*	32	5.60	305	1	0.09	299	1	0.39
900	315	315*	48	10.30	339	1	0.12	321	12	6.76
soma		1375	253	27.89	1477	10	0.40	1381	116	22.44

Tabela 5.3: Resultado para grafos com 50 vértices: sse, els e pqr

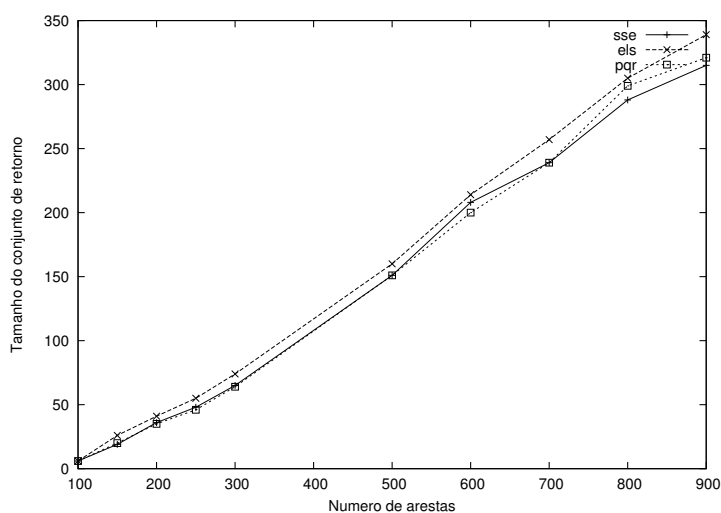


Figura 5.3: Tamanho do conjunto de retorno para grafos com 50 vértices: sse, sser e pqr

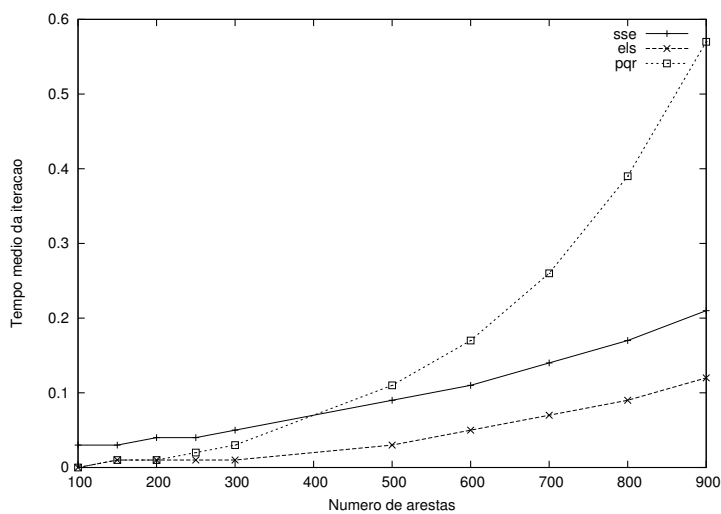


Figura 5.4: Tempo médio da iteração para grafos com 50 vértices: sse, els e pqr

A	M	sse			sser			ssep			sserp		
		M	IT	T	M	IT	T	M	IT	T	M	IT	T
200	14	14*	24	1.63	14*	3	0.08	14*	6	1.09	14*	3	0.16
300	35	37	41	3.60	38	31	1.64	38	7	1.06	38	40	3.33
400	59	59*	33	3.54	63	2	0.18	59*	41	7.19	62	10	1.22
500	91	91*	6	0.77	95	43	4.86	91*	46	9.43	94	11	1.74
600	122	129	29	4.97	123	28	3.82	122*	11	2.72	131	9	1.88
1000	287	289	44	16.31	295	10	3.59	287*	26	11.74	288	40	16.72
1100	324	330	7	3.08	336	37	15.78	324*	19	9.97	330	4	1.97
1200	378	382	37	19.37	387	8	4.20	382	35	22.21	382	24	15.15
1300	418	426	48	29.35	419	12	6.99	429	20	15.19	421	24	17.22
1400	456	466	27	18.76	466	37	25.27	456*	46	38.37	457	33	26.41
soma		2223	296	101.39	2236	211	66.40	2202	257	118.96	2217	198	85.80

Tabela 5.4: Resultado para grafos com 100 vértices: sse, sser, ssep e sserp

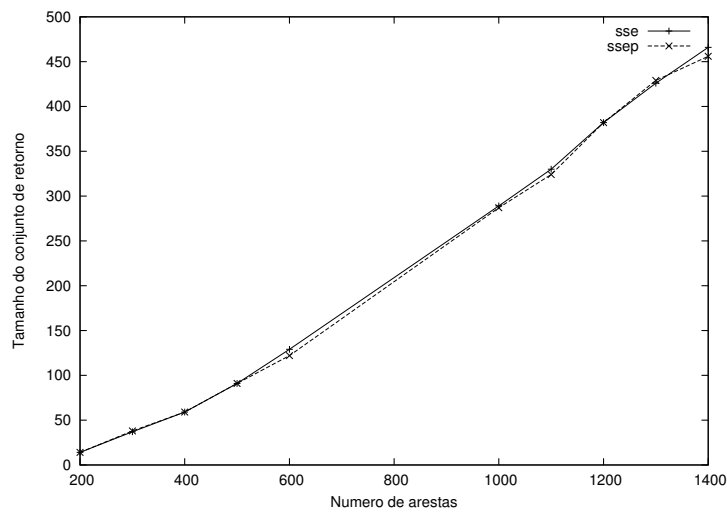


Figura 5.5: Tamanho do conjunto de retorno para grafos com 100 vértices: sse e ssep

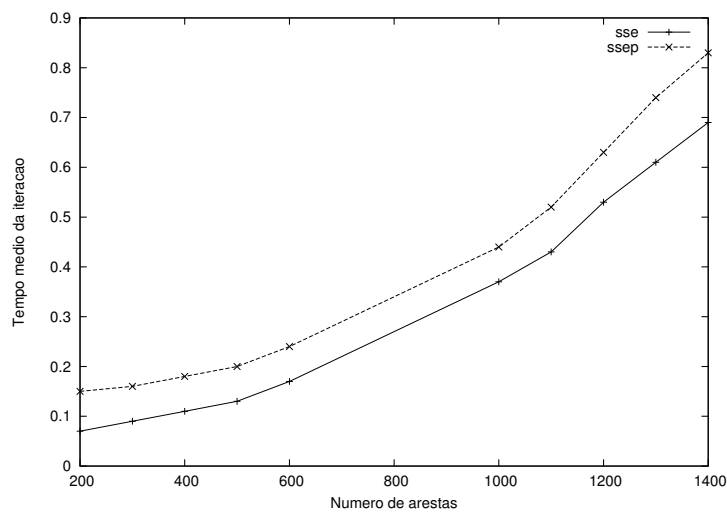


Figura 5.6: Tempo médio da iteração para grafos com 100 vértices: sse e ssep

A	M	sse			els			pqr		
		M	IT	T	M	IT	T	M	IT	T
200	14	14*	24	1.63	16	1	0.02	15	1	0.01
300	35	37	41	3.60	44	1	0.02	35*	15	0.34
400	59	59*	33	3.54	66	1	0.03	62	31	1.40
500	91	91*	6	0.77	104	1	0.05	93	27	2.11
600	122	129	29	4.97	132	1	0.07	122*	44	5.41
1000	287	289	44	16.31	319	1	0.21	289	11	5.04
1100	324	330	7	3.08	350	1	0.26	335	15	9.28
1200	378	382	37	19.37	392	1	0.33	378*	29	23.53
1300	418	426	48	29.35	443	1	0.39	418*	46	47.29
1400	456	466	27	18.76	480	1	0.48	467	49	64.84
soma		2223	296	101.39	2346	10	1.84	2214	268	159.24

Tabela 5.5: Resultado para grafos com 100 vértices: sse, els, pqr

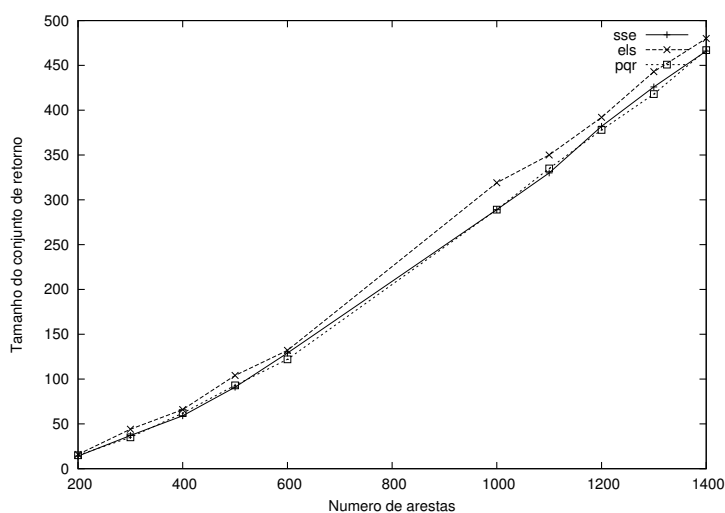


Figura 5.7: Tamanho do conjunto de retorno para grafos com 100 vértices: sse, sser e pqr

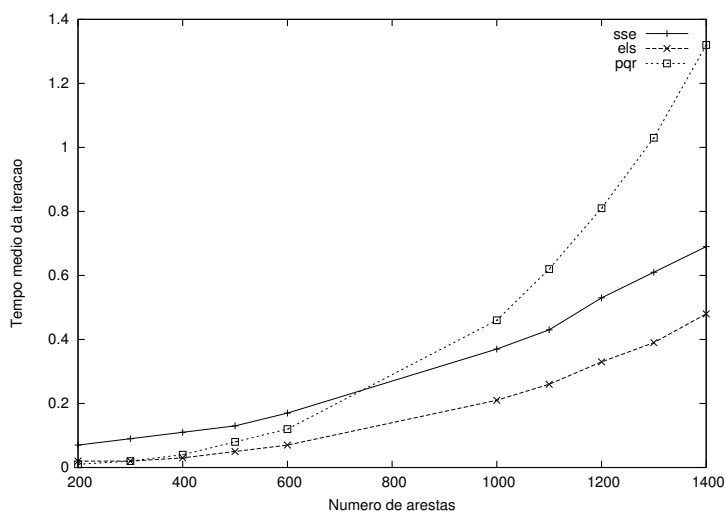


Figura 5.8: Tempo médio da iteração para grafos com 100 vértices: sse, els e pqr

A	M	sse			sser			ssep			sserp		
		M	IT	T	M	IT	T	M	IT	T	M	IT	T
1000	51	57	26	31.39	56	46	18.65	56	40	107.21	57	10	6.22
1500	147	149	16	28.33	152	50	58.37	150	49	119.82	150	5	6.56
2000	270	270*	47	113.56	282	37	75.64	275	50	151.85	286	30	65.45
2500	416	416*	33	108.91	437	19	60.54	419	12	46.35	427	30	98.68
3000	593	596	24	110.39	615	30	131.02	593*	37	185.27	613	20	90.48
5000	1309	1316	14	163.34	1323	30	332.66	1309*	21	252.03	1339	38	445.46
5500	1514	1514*	24	328.99	1542	50	694.65	1553	7	110.38	1528	33	470.54
6000	1712	1712*	37	605.50	1762	49	800.22	1736	16	268.62	1749	20	331.00
6500	1907	1907*	23	425.93	1920	41	760.88	1913	9	176.75	1941	17	359.35
7000	2103	2103*	39	931.72	2126	48	1165.24	2114	13	344.64	2153	47	1162.45
soma	10040	283	2848.07	10215	400	4097.88	10118	254	1762.93	10243	250	3036.18	

Tabela 5.6: Resultado para grafos com 500 vértices: sse, sser, ssep e sserp

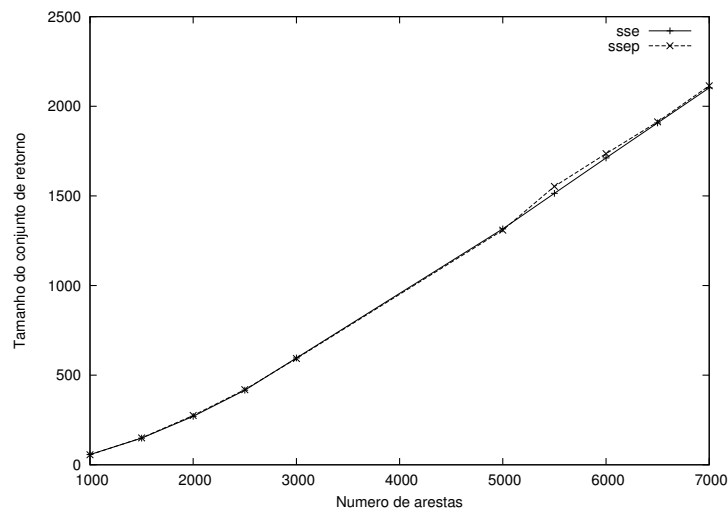


Figura 5.9: Tamanho do conjunto de retorno para grafos com 500 vértices: sse e ssep

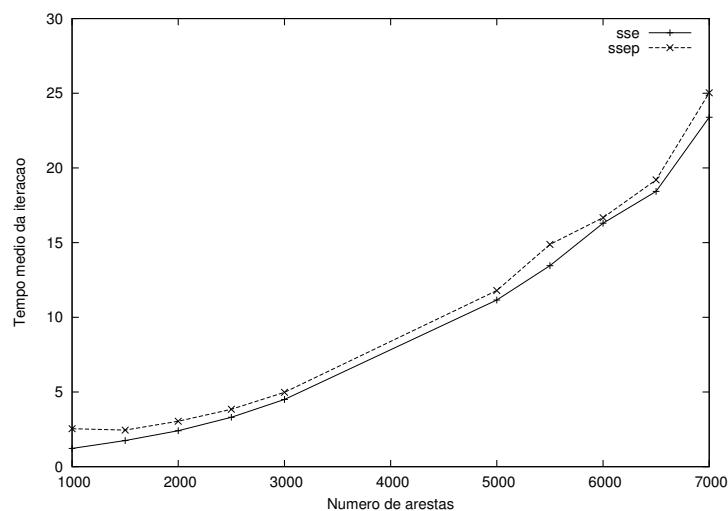


Figura 5.10: Tempo médio da iteração para grafos com 500 vértices: sse e ssep

$ A $	M	sse			els			pqr		
		M	IT	T	M	IT	T	M	IT	T
1000	51	57	26	31.39	69	1	0.79	51*	40	6.82
1500	147	149	16	28.33	171	1	1.32	147*	35	22.43
2000	270	270*	47	113.56	335	1	1.98	276	39	58.62
2500	416	416*	33	108.91	490	1	2.90	424	1	2.80
3000	593	596	24	110.39	667	1	3.94	616	35	184.15
5000	1309	1316	14	163.34	1427	1	11.03	1367	14	280.41
5500	1514	1514*	24	328.99	1632	1	13.30	1550	13	325.18
6000	1712	1712*	37	605.50	1862	1	14.79	1776	21	626.10
6500	1907	1907*	23	425.93	2046	1	17.46	1963	1	36.95
7000	2103	2103*	39	931.72	2262	1	20.72	2157	1	50.62
soma		10040	283	2848.07	10961	10	88.22	10327	200	1594.07

Tabela 5.7: Resultado para grafos com 500 vértices: sse, els, pqr

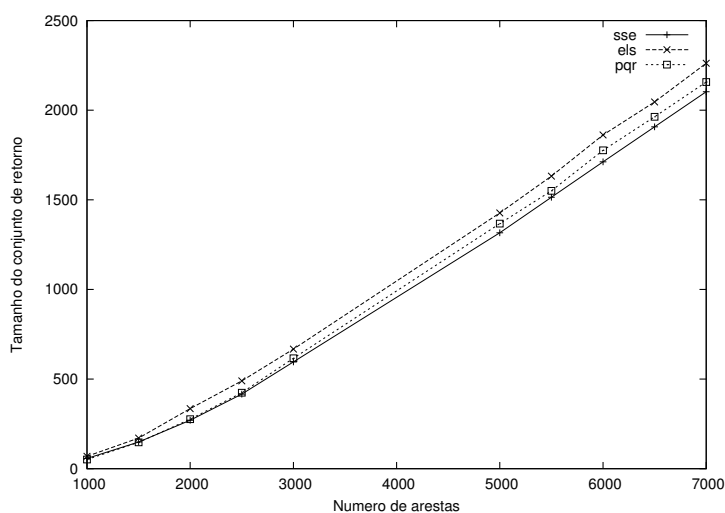


Figura 5.11: Tamanho do conjunto de retorno para grafos com 500 vértices: sse, sser e pqr

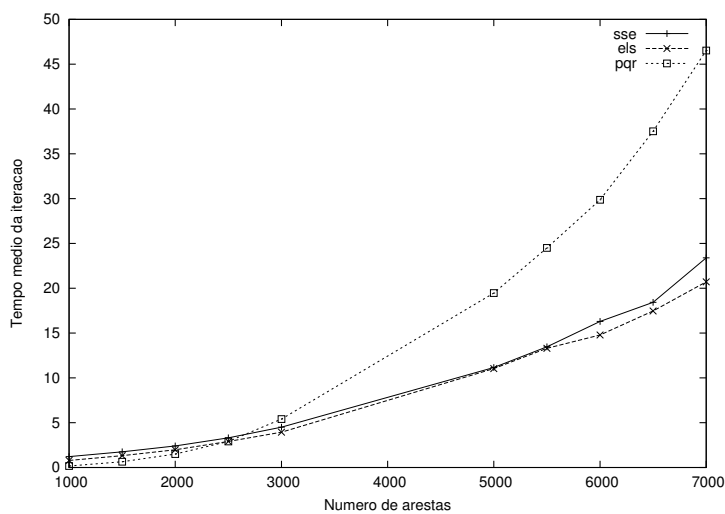


Figura 5.12: Tempo médio da iteração para grafos com 500 vértices: sse, els e pqr

A	M	sse			ssep			ssep			sserp		
		M	IT	T	M	IT	T	M	IT	T	M	IT	T
3000	287	301	38	309.25	305	21	107.39	295	42	402.45	308	13	65.59
3500	405	405*	34	296.11	416	20	131.40	405*	29	332.14	422	50	344.85
4000	521	521*	36	362.70	558	7	58.14	522	3	35.71	548	39	343.68
4500	670	670*	15	199.97	696	1	13.00	670*	5	66.66	717	42	441.41
5000	803	807	29	387.06	856	26	315.71	803*	23	336.33	847	23	289.70
10000	2625	2625*	35	1574.86	2766	46	2064.89	2641	45	2063.87	2680	43	1946.79
15000	4539	4539*	34	3310.81	4669	18	1762.66	4563	24	2393.72	4599	44	4307.16
20000	6704	6724	33	6029.28	6819	38	6903.95	6704*	46	8424.50	6791	49	9177.34
25000	8843	8843*	26	7685.56	8943	36	10644.91	8848	48	15595.30	8933	49	14854.00
30000	11136	11194	48	21547.73	11275	43	19122.33	11136*	50	22303.83	11277	22	9904.67
soma		36629	328	41703.33	37303	256	41124.37	36587	315	51954.52	37122	374	41675.20

Tabela 5.8: Resultado para grafos com 1000 vértices: sse, ssep, ssep e sserp

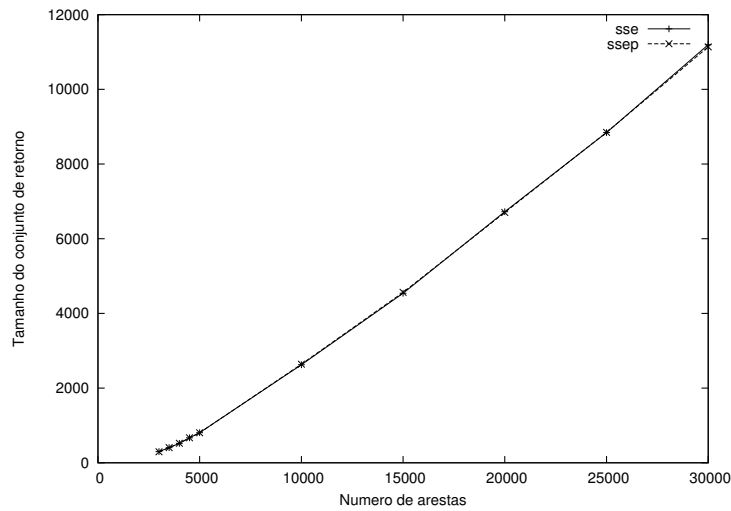


Figura 5.13: Tamanho do conjunto de retorno para grafos com 1000 vértices: sse e ssep

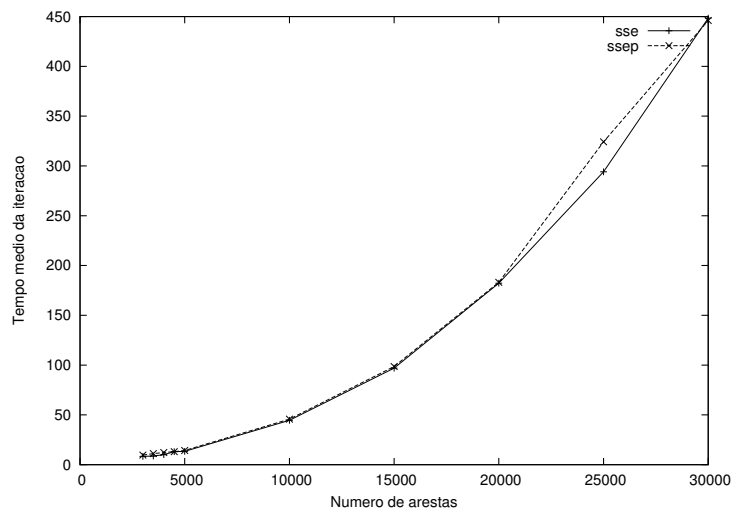


Figura 5.14: Tempo médio da iteração para grafos com 1000 vértices: sse e ssep

A	M	sse			els			pqr		
		M	IT	T	M	IT	T	M	IT	T
3000	287	301	38	309.25	365	1	6.84	287*	43	135.68
3500	405	405*	34	296.11	506	1	8.41	411	30	163.62
4000	521	521*	36	362.70	637	1	14.18	539	4	36.64
4500	670	670*	15	199.97	774	1	10.93	675	36	355.33
5000	803	807	29	387.06	907	1	12.99	839	49	662.98
10000	2625	2625*	35	1574.86	2900	1	43.83	2759	11	868.53
15000	4539	4539*	34	3310.81	4855	1	98.64	4736	45	9649.52
20000	6704	6724	33	6029.28	7044	1	181.72	7024	10	4679.55
25000	8843	8843*	26	7685.56	9321	1	296.94	9299	3	2490.40
30000	11136	11194	48	21547.73	11617	1	480.51	11702	34	49877.48
soma		36629	328	41703.33	38926	10	1154.99	38271	265	68919.73

Tabela 5.9: Resultado para grafos com 1000 vértices: sse, els, pqr

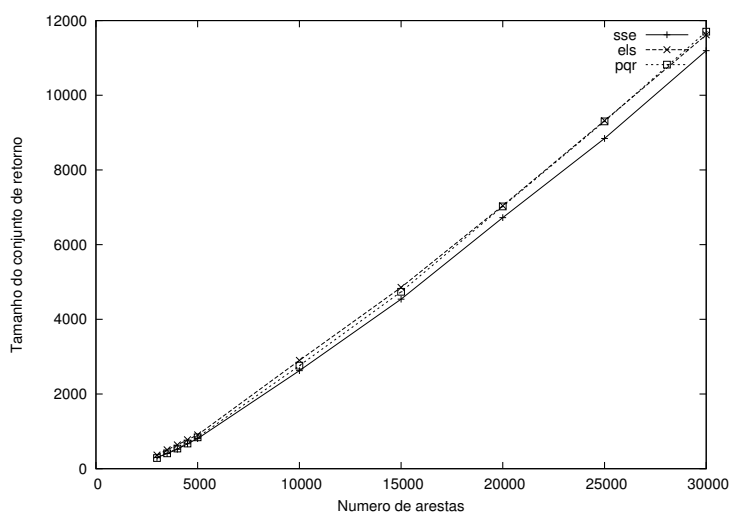


Figura 5.15: Tamanho do conjunto de retorno para grafos com 1000 vértices: sse, sser e pqr

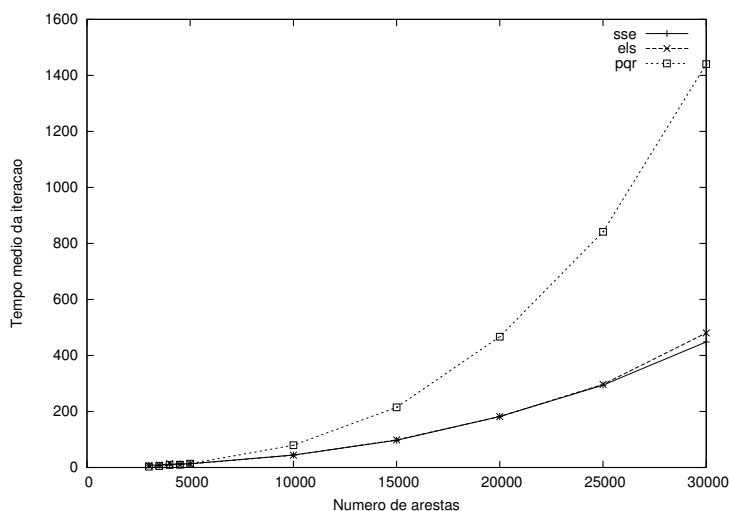


Figura 5.16: Tempo médio da iteração para grafos com 1000 vértices: sse, els e pqr

Conclusões

Neste trabalho apresentamos quatro algoritmos existentes na literatura para resolver problemas de conjunto de retorno: uma heurística simples e um algoritmo “dividir-para-conquistar” para resolver o FAS; a aplicação da meta-heurística GRASP para resolver o FVS e um algoritmo polinomial exato para resolver o FAS restrito a grafos planares.

Apresentamos também a equivalência entre o problema do conjunto de arestas de retorno e o problema do conjunto de vértices de retorno, bem como a relação entre ordenação de vértices e conjunto de arestas de retorno.

A principal contribuição desta dissertação foi a modificação do algoritmo de Saab [27]. Acrescentamos técnicas de redução do problema e modificamos o caso base do algoritmo, utilizando um algoritmo polinomial exato para resolver o FAS para grafos planares. A segunda modificação mostrou-se eficiente e gerou as melhores soluções para grafos com 50 e 100 vértices. O acréscimo no tempo de execução não é significativo.

Outra contribuição foi o estudo comparativo de dois algoritmos heurísticos: o algoritmo de Saab [27] e o de Pardalos, Qian e Resende [23]. A qualidade das soluções encontradas pelas duas heurísticas são semelhantes. No entanto, o algoritmo de Saab é um pouco melhor em relação à qualidade de solução e o tempo de execução para grafos densos é muito menor.

Trabalhos futuros

No final desta dissertação os autores tiveram conhecimento de outros dois algoritmos para resolver o FAS para grafos planares, a saber o de Gabow [13] e o de Iwata e Kobayashi

[18]. Estes algoritmos têm complexidade $O(n^3)$ e podem melhorar o desempenho do nosso algoritmo (o algoritmo de Frank tem complexidade $O(n^4)$).

Outra possibilidade é explorar novas funções gulosas para o algoritmo GRASP de Pardalos, Qian e Resende [23]. Um função interessante que dá uma visão “global” da participação dos vértices nos ciclos de um grafo é o tamanho do fecho transitivo direto e fecho transitivo indireto.

Ainda é possível explorar outras meta-heurísticas para resolver o FAS. Apesar de ser difícil definir um método de busca local para conjuntos de retorno, é mais provável construir uma busca local vendo o conjunto de retorno como uma ordenação de vértices. Poderemos pensar, por exemplo, em trocar dois vértices de posição na ordenação e verificar se a nova ordenação gera um conjunto de retorno melhor que a anterior.

Referências

- [1] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
- [2] John M. Boyer and Wendy Myrvold. On the cutting edge: Simplified $O(n)$ planarity by edge addition. Aceito para *Journal of Graph Algorithms and Applications*, 07 2005.
- [3] Candido Ferreira Xavier de Mendonça Neto and Peter Eades. An improvement for an algorithm for finding a minimum feedback arc set for planar graphs. *Acta Scientiarum*, 21(4):841–845, 1999.
- [4] P. Eades and X. LIN. A heuristic for the feedback arc set problem. *The Australasian Journal of Combinatorics*, 12:15–26, 1995.
- [5] Peter Eades, Xuemin Lin, and W. F. Smyth. A fast and effective heuristic for the feedback arc set problem. *Inf. Process. Lett.*, 47(6):319–323, 1993.
- [6] J. Edmonds and R. Giles. A min-max relation for submodular functions on graphs. *Annals of Discrete Mathematics*, 1:185–204, 1977.
- [7] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. volume 20, pages 151–174, 1998.
- [8] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [9] P. Festa, P. Pardalos, and M. Resende. Feedback set problems, 1999.
- [10] Paola Festa, Panos M. Pardalos, and Mauricio G. C. Resende. Algorithm 815: Fortran subroutines for computing approximate solutions of feedback set problems using grasp. *ACM Trans. Math. Softw.*, 27(4):456–464, 2001.
- [11] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, 1963.
- [12] Andràs Frank. How to make a digraph strongly connected. *Combinatorica*, 1(2):145–153, 1981.
- [13] Harold N. Gabow. A representation for crossing set families with applications to submodular flow problems. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 202–211, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

-
- [14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [15] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, Cambridge, U.K., 1985.
- [16] John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of ACM*, 21(4):549–568, 1974.
- [17] Edna Ayako Hoshino. Algoritmos lineares para teste de planaridade em grafos. Master's thesis, Departamento de Computação e Estatística – Universidade Federal de Mato Grosso do Sul, 2002.
- [18] Satoru Iwata and Yusuke Kobayashi. An algorithm for minimum cost arc-connectivity orientations. Technical report, Department of Mathematical Informatics – Graduate School of Information Science and Technology – University of Tokyo, julho 2005.
- [19] C. Kuratowski. Sur le probleme des corbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.
- [20] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing in graphs. In Gordon and Breach, editors, *Theory of Graphs: International Symposium*, pages 215–232, New York, 1967.
- [21] C.L. Lucchesi and D.H. Younger. A minimax relation for directed graphs. *J. London Mathematical Society*, 17:369–374, 1978.
- [22] Alexandre Noma. Análise experimental de algoritmos de planaridade. Master's thesis, Instituto de Matemática e Estatística da Universidade de São Paulo, 2003.
- [23] P. Pardalos, T. Qian, and M. Resende. A greedy randomized adaptive search procedure for feedback vertex set, 1999.
- [24] L. Pitsoulis and M. Resende. Greedy randomized adaptive search procedures, 2001.
- [25] L.S. Pitsoulis and M.G.C. Resende. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 178–183. Oxford University Press, 2002.
- [26] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.
- [27] Youssef Saab. A fast and effective algorithm for the feedback arc set problem. *J. Heuristics*, 7(3):235–250, 2001.
- [28] Mihalis Yannakakis. Node-and edge-deletion np-complete problems. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, New York, NY, USA, 1978. ACM Press.
- [29] D. H. Younger. Minimum feedback arc sets for a directed graph. *IEEE Transactions on Circuits and Systems*, 10(2):238–245, 6 1963.