

Melhoria de um algoritmo para encontrar conjuntos de retorno

Odacil da Costa Barbosa Junior

Monografia do Projeto Final de Especialização, apresentada ao Curso de Desenvolvimento de Sistemas para WEB da Universidade Estadual de Maringá, como parte dos requisitos para obtenção do título de **Especialista em Desenvolvimento de Sistemas para WEB.**

ORIENTADOR: Msc. Marco Aurélio Barbosa

Maringá - PR
2006 - 2007

Resumo

Barbosa Junior, O.C. *Melhoria de um algoritmo para encontrar conjuntos de retorno*. Maringá/PR, 2006/2007. Monografia (Especialização) - Curso de Desenvolvimento de Sistemas para WEB - Universidade Estadual de Maringá.

Um algoritmo para encontrar conjuntos de retorno é descrito. Após testes de processamento de diferentes grafos começando com poucas arestas até muitas, é produzido um gráfico evolutivo, tendo como eixos o número de arestas em função do tamanho do conjunto de retorno (número de arestas de retorno). Este algoritmo foi desenvolvido por Eades, Lin e Smyth[6].

Feito isso, o algoritmo é modificado, sua característica original determinista é alterada, adicionando-se uma função não determinística ao algoritmo, isto foi feito utilizando o funcionamento do GRASP - *Greedy Randomized Adaptive Search Procedure* (Procedimento de Busca Adaptativo, Aleatório e Guloso).

O objetivo é obter melhoria na qualidade da solução removendo o mínimo de arestas para tornar o grafo acíclico. Por esta razão um novo mapeamento com medições do tamanho do conjunto de retorno dos grafos é efetuado, e finalmente comparado com os primeiros para a verificação na melhoria da qualidade das respostas.

Palavras-chave: *conjuntos de retorno, problemas de conjunto de retorno.*

Índice

1	Introdução	1
1.1	Objetivos	1
1.1.1	Contribuições	1
1.1.2	Justificativa e Motivação	1
1.2	Limitações	2
1.3	Metodologia	2
1.4	Organização do Texto	2
2	Fundamentação Teórica	3
2.1	Grafos e Grafos Simples	3
2.2	Conjuntos de Retorno	4
2.2.1	Problema do conjunto de retorno	4
3	Uma heurística determinista para resolver o FAS	5
3.1	Relação entre ordenação de vértices e conjunto de arestas de retorno	5
3.2	Algoritmo estudado	6
4	GRASP	7
4.1	Introdução ao GRASP	7
4.1.1	Pseudo-código do algoritmo GRASP	8
4.2	Funcionamento do algoritmo GRASP	8
4.2.1	Fase de construção	8
4.2.2	Fase de Melhoria	10
4.2.3	Características e vantagens do método	10
5	Algoritmo proposto	11
5.1	Introdução	11
5.2	Algoritmo Proposto	11
6	Resultados obtidos	13
6.1	Descrição da Metodologia dos Testes	13
6.2	Gráficos Comparativos	13
6.2.1	Análise de Grafos com 50 Vértices	14
6.2.2	Análise de Grafos com 100 Vértices	15
6.2.3	Análise de Grafos com 500 Vértices	16
6.2.4	Análise de Grafos com 1.000 Vértices	17
6.2.5	Tempo de Processamento dos Algoritmos	18
7	Conclusão	19
	Referências Bibliográficas	20

Lista de Figuras

2.1	Diagrama de Grafos: (a) Orientado (b) Não Orientado	3
3.1	Pseudocódigo do algoritmo estudado	6
4.1	Pseudocódigo do algoritmo GRASP	8
5.1	Pseudocódigo do algoritmo após modificação	12
6.1	Teste com 50 Vértices	14
6.2	Teste com 100 Vértices	15
6.3	Teste com 500 Vértices	16
6.4	Teste com 1.000 Vértices	17
6.5	Comparativo dos algoritmos: Tempo gasto com processamento	18

Capítulo 1

Introdução

Muitos dos problemas computacionais cotidianos, podem ser essencialmente reduzidos a problemas que de forma bastante natural se articulam mediante a utilização de grafos.

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 relação 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.

Conjuntos de retorno podem ser utilizados para análise de sistemas em larga escala, desenho de grafos, inferência estatística entre outras utilidades.

Como problemas dessa natureza exigem um processamento dispendioso, a busca por algoritmos mais eficientes é um meio na redução do tempo gasto para encontrar a resposta desejada. Muitas vezes resultados exatos são inviáveis em virtude do tempo exorbitante gasto para chegar a esta solução, portanto, algoritmos que produzem resultados não exatos, mas dentro de uma margem de erro conhecida também são utilizados como alternativa para melhorar o tempo de obtenção de uma resposta satisfatória.

1.1 Objetivos

O objetivo geral desse trabalho é a comparação de algoritmos para encontrar conjuntos de retorno.

Como finalidade específica dessa revisão, o foco é para os seguintes objetivos:

- Propor a modificação de um algoritmo;
- Implementar o algoritmo original e o modificado e realizar comparações entre as respostas do algoritmo original e do modificado.

1.1.1 Contribuições

As contribuições esperadas com o trabalho são:

- Melhoria de um algoritmo para encontrar conjuntos de retorno.

1.1.2 Justificativa e Motivação

Esta pesquisa é importante para ampliar as fontes acerca de algoritmos para encontrar conjuntos de retorno.

A motivação está principalmente na possibilidade de melhoria e implementação de um algoritmo para encontrar conjuntos de retorno, e sua descrição.

A sobrecarga no processamento computacional, pode ser resolvida, dentre outras formas, com a melhoria dos algoritmos utilizados.

Sendo assim, é necessária uma revisão nos algoritmos existentes na tentativa de melhorá-los tornando-os mais eficientes com melhoria na qualidade dos resultados e redução no tempo de processamento.

Este trabalho poderá ser utilizado para aperfeiçoamentos e desenvolvimentos posteriores.

1.2 Limitações

As ocorrências desta pesquisa será a revisão de um algoritmos para encontrar conjunto de retorno, sendo efetuado a alteração no mesmo.

Será apresentado o algoritmo em sua forma original, e após modificado.

Também será efetuado a implementação, para medição do tempo de processamento. Os resultados serão apresentados em forma de gráficos.

A revisão de grafos será apenas de grafos e grafos simples planares.

1.3 Metodologia

A metodologia utilizada envolve:

- Revisão bibliográfica de grafos: Apresentação dos conceitos e grafos utilizados para demonstrar os algoritmos para encontrar conjuntos de retorno.
- Revisão bibliográfica de algoritmos: Estudo de um algoritmo que resolve o problema em foco.
- Implementação: Será utilizado java para implementação do algoritmo.
- Testes e comparação dos resultados: Verificação do desempenho do algoritmo original, e do algoritmo modificado, através da implementação de ambos e execução para medição comparativa da resposta apresentada.

1.4 Organização do Texto

A estrutura deste trabalho será o seguinte: inicialmente na Seção 2 será feito uma revisão teórica com definições básicas sobre grafos, além de conceitos do problema de conjuntos de retorno, na sequência será abordado as características do GRASP. Abaixo segue enumerado as seções deste trabalho:

- Apresentação do algoritmo ELSFAS
- Apresentação do GRASP
- Apresentação do ELSFAS alterado para usar função do GRASP
- Gráficos do teste usando ELSFAS e ELSFASGRASP
- Conclusão

Capítulo 2

Fundamentação Teórica

Muitas situações reais podem ser descritas convenientemente como um diagrama consistindo em pontos e linhas unindo certos pontos. A partir do aperfeiçoamento desse tipo de representação surgiram os grafos.

2.1 Grafos e Grafos Simples

Nesta seção será apresentada a definição e conceitos de grafos[3].

Um grafo G é uma tripla ordenada $(V(G), E(G), \psi_G)$ que consiste em um conjunto não vazio de vértices $V(G)$, um conjunto $E(G)$, desmembrando de $V(G)$, de arestas, e uma função de incidência ψ_G que associa para cada aresta de G um par ordenado (não necessariamente distinto) de vértices de G . Se e é uma aresta e u e v são vértices sendo $\psi_G(e) = uv$, então e é o caminho de ligação de u e v ; os vértices u e v são chamados de *terminais* de e . Os vértices também são chamados de *nós* e arestas chamadas de *arcos*.

Um grafo é não orientado quando as arestas não possuem uma orientação (direção) específica. Uma aresta orientada (u,v) vai de u para v , sendo que u é a *origem* e v o *destino*[2].

A figura Figura 2.1(a) representa um grafo orientado, e a Figura 2.1(b) um grafo não-orientado. Os vértices são representados por pequenos círculos e as arestas são setas ligando os dois círculos correspondentes aos seus extremos. No caso de grafos não orientados utiliza-se apenas linhas unindo os vértices.

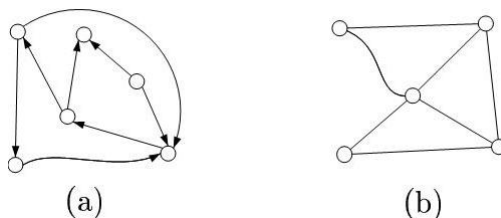


Figura 2.1: Diagrama de Grafos: (a) Orientado (b) Não Orientado

Quaisquer dois vértices conectados por uma aresta ou quaisquer duas arestas conectadas por um vértice, são chamados *adjacentes*. O *grau de um vértice* em um grafo não orientado é o número de arcos incidentes nele. Um *caminho* em um grafo não orientado é uma sequência de arestas, onde o vértice final de uma aresta é o vértice inicial da próxima (sequência de vértices adjacentes)[2].

Em um grafo orientado, os caminhos também são orientados, com arestas adjacentes simultaneamente chegando e saindo dos vértices.

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 é definido apenas os conceitos de grafos simples, sendo assim, um grafo G é definido apenas como uma dupla ordenada (V, A) . Uma aresta será representada como sendo $\psi(e) = (u, v)$, ou apenas como (u, v) , ou ainda uv .

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 [1].

Um *passeio* P é uma 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 *sai* 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*[1].

2.2 Conjuntos de Retorno

2.2.1 Problema do conjunto de retorno

Um *conjunto de retorno* pode ser formado pelo conjunto de arestas ou vértices que quando removidos tornam um grafo acíclico, independente do grafo ser orientado ou não orientado [1].

Quando há um caminho possível para percorrer todos os vértices do grafo caminhando pelas arestas, temos um grafo conectado [3].

Inúmeros algoritmos foram desenvolvidos para encontrar conjuntos de retorno. Existem alguns algoritmos exatos para encontrar conjunto de retorno em grafos planares em tempo polinomial.

A dificuldade está em obter o conjunto de retorno de custo mínimo, para arestas ou vértices com pesos, e o conjunto de retorno de cardinalidade mínima, para arestas ou vértices sem pesos [1].

Formalmente, Festa, Pardalos e Resende[7] definiram o problema do conjunto de vértices de retorno da seguinte forma:

Assuma $G = (V, E)$ como sendo um grafo e seja $w : V(G) \rightarrow \mathbb{R}^+$ uma função positiva definida sobre os vértices de G . Um *conjunto de vértices de retorno* de G é um subconjunto de vértices $V' \subseteq V(G)$ tal que cada ciclo em G contenha pelo menos um vértice em V' , ou seja, um conjunto de vértices de retorno de V' é um conjunto de vértices de G de tal forma que removendo V' de G juntamente com todas as arestas incidentes em V' , resulta em uma floresta.

A cardinalidade $d_G(v)$ de um vértice v em G é o número de arestas de G incidentes em v . Denota-se $\delta(G)$ e $\Delta(G)$, a cardinalidade mínima e máxima respectivamente [3].

Capítulo 3

Uma heurística determinista para resolver o FAS

Nesta seção será feita uma revisão do algoritmo desenvolvido por Eades, Lin e Smyth[6], que estabelece uma heurística determinística para encontrar um conjunto de retorno. Como o algoritmo utiliza ordenação de vértices como processo para obter o conjunto de arestas de retorno, uma breve descrição é feita.

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

Para um grafo $G = (V, A)$ assumimos uma ordenação de vértices qualquer tal que $l : V \rightarrow N$. Uma aresta (u, v) é definida como *para frente* se $l(u) < l(v)$, senão a aresta será definida como *para trás* ($l(u) > l(v)$).

As definições que relaciona a ordenação de vértices com o conjunto de arestas de retorno foram inicialmente descrita por Younger[8]:

1. Há uma ordenação de vértices de tal forma que o conjunto de arestas para trás define um conjunto de arestas de retorno de custo mínimo. Essa ordenação é chamada de *ordenação ótima*.
2. Um *subgrafo consecutivo* de um grafo $G = (V, A)$ com relação a uma ordenação de vértices é um subgrafo induzido por um subconjunto de vértices consecutivos nesta ordenação. Sendo dois subgrafos G_1 e G_2 de um grafo $G = (V, A)$, define-se como $N(G_1, G_2)$ o número de arestas que saem de um vértice 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 representado por todas as arestas para trás em 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 máximo
12. 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: Pseudocódigo do algoritmo estudado

3.2 Algoritmo estudado

O algoritmo demonstrado na Figura 3.1, constrói uma lista com os vértices, organizando-os segundo um critério determinístico baseado na cardinalidade do vértice, ou seja, a diferença entre as arestas que chegam no vértice pelas arestas que saem.

Após a ordenação considera-se as arestas que apontam para trás da ordenação como arestas de retorno.

Para realizar a ordenação foi usada duas listas, denominadas S_1 e S_2 . O primeiro passo do algoritmo é localizar os sumidouros e armazená-los no início da lista 2. Não havendo sumidouro o algoritmo procura por fontes, e então adiciona no final da lista 1. Não havendo sumidouro nem fonte no grafo, o algoritmo seleciona o vértice com maior valor resultante da diferença entre o número de arestas que chegam no vértice pelas arestas que saem.

Este procedimento, descreve uma ordenação onde os primeiros são os vértices que possuem maior número de arestas saindo, provavelmente fontes, e na sequência, a ordenação dos vértices que possuem menor número de arestas chegando. Esta ordenação diminui a quantidade de arestas que apontam para trás, e intuitivamente, fornece um bom conjunto de arestas de retorno.

Cada vértice adicionado a lista 1 e lista 2 é removido do grafo, e as listas 1 e 2 são concatenadas. As arestas que apontam para trás na ordenação obtida pela concatenação das duas listas, definem o conjunto de arestas de retorno.

Capítulo 4

GRASP

Nesta seção é apresentado as características e funcionalidades do GRASP, com detalhes de execução e anotação do pseudo-código. Algumas comparações com outros métodos foram estabelecidos para uma melhor avaliação do procedimento. As bases utilizadas neste tópico estão em Pitsoulis[4], Pardalos[5].

4.1 Introdução ao GRASP

Um procedimento GRASP - *Greedy Randomized Adaptive Search Procedure* (Procedimento de Busca Adaptativo, Aleatório e Guloso) - é uma metaheurística híbrida de construção seguida de melhoramento para Problemas de Otimização Combinatorial (POC). O princípio é análogo ao chamado método de múltiplas partidas aleatórias (*random multi-start*): repete-se várias vezes a aplicação de busca local em soluções iniciais diferentes geradas aleatoriamente.

A diferença crucial entre GRASP e Busca Local com Múltiplas Partidas Aleatórias (BLMP) é que no primeiro método a aleatoriedade é utilizada em conjunto com informação heurística que leva em conta critérios de otimização relativos à solução parcial; na segunda, a aleatoriedade é usada somente como critério direto de geração de combinações, arranjos ou seqüências iniciais sem nenhuma preocupação com o valor de função objetivo.

Na prática, um procedimento GRASP tende a ser melhor do que BLMP tanto em termos de tempo e muito provavelmente em termos de qualidade de soluções. Isto se explica pelo seguinte motivo: uma BLMP realiza amostragem de soluções iniciais em todo o espaço de soluções, enquanto que um GRASP faz a amostragem em pontos do espaço de soluções limitados por um critério de otimização (função gulosa), tendo assim menor variância e melhor média no valor da função objetivo e por conseqüência, menor distância média entre as soluções iniciais construídas e os seus respectivos ótimos locais alcançados.

Dentro do mesmo intervalo de tempo, enquanto um procedimento de BLMP despende esforço computacional perturbando muitas vezes a solução corrente, o procedimento GRASP tenta várias alternativas de solução inicial. A estratégia do método é portanto diminuir a distância percorrida entre soluções iniciais e ótimos locais para que se possa descobrir uma maior quantidade de ótimos locais dentro de toda a vizinhança explorável.

A idéia de iteração do GRASP, na sua forma básica, difere bastante das outras metaheurísticas, pois a solução da iteração corrente não é resultado das anteriores, sendo fruto unicamente da semente aleatória da iteração.

No GRASP básico, não existe deterioração controlada da solução corrente como mecanismo de fuga do ótimo local, como ocorre por exemplo no *Simulated Annealing* (deterioração controlada estocasticamente) ou Busca Tabu (deterioração controlada por estruturas de memória). Neste métodos, a idéia é começar de uma solução inicial e percorrer um único caminho dentro da vizinhança explorável (eventualmente deteriorando a solução cor-

rente, *Simulated Annealing*, ou o ótimo local alcançado, Busca Tabu) até que se pare em alguma solução, que será a final.

De acordo com a exploração do espaço de soluções temos: Metaheurísticas com movimento de piora controlada (Busca Tabu, *Simulated Annealing*, e Metaheurísticas de múltiplas partidas (GRASP).

Cada iteração do GRASP é dividida em duas fases importantes e distintas:

- fase de construção, onde uma solução viável é construída elemento por elemento através de um mecanismo guloso e aleatório;
- fase de melhoria, um procedimento de busca local que toma como solução inicial a solução resultante da fase anterior, explorando sua vizinhança até que um ótimo local seja alcançado.

4.1.1 Pseudo-código do algoritmo GRASP

Pseudo código do algoritmo GRASP de acordo com Resende e Ribeiro:

```

procedure GRASP(Max.Iterations, Seed)
1  Read_Input();
2  for k=1,...,Max.Iterations do
3      Solution ← Greedy_Randomized_Construction(Seed);
4      Solution ← Local_Search(Solution);
5      Update_Solution(Solution, Best_Solution);
6  end;
7  return Best_Solution;
end GRASP.
```

Figura 4.1: Pseudocódigo do algoritmo GRASP

4.2 Funcionamento do algoritmo GRASP

4.2.1 Fase de construção

A fase de construção, é baseada no algoritmo de Hart e Shogan, onde a idéia é inserir um "grão de sal" ao algoritmo obtido do método guloso. Em um algoritmo guloso normal, os seguintes passos ocorrem:

- calcula-se para cada elemento passível de inserção o valor de uma função utilizada como critério de otimização, chamada função gulosa;
- ordena-se os elementos (ordem crescente se $meta = min$, decrescente se $meta = max$) de acordo com seu valor de função gulosa;
- insere-se sucessivamente os elementos, seguindo a ordenação do passo anterior, até que se tenha uma solução viável.

O grão de sal a ser inserido no algoritmo guloso se refere ao terceiro passo: troca-se o determinismo da escolha ordenada por uma aleatoriedade controlada. Para que isto ocorra, define-se uma lista com os elementos com melhores valores, sendo que o tamanho da lista e os valores de função gulosa dos elementos pertencentes à esta são controlados por parâmetros passados para o algoritmo. A cada passo de inserção, o elemento a ser inserido é escolhido aleatoriamente dentro desta lista restrita de candidatos (LRC), que é reconstruída a cada inserção.

A fase de construção se dá em três passos, que se repetem até que se tenha uma solução inicial viável:

1. Calcular e adaptar os valores da função gulosa para os elementos de E ;
2. Construir a LRC;
3. Escolher aleatoriamente de LRC e inserir na solução.

Para as definições que seguem, será considerado que: C é o conjunto de elementos candidatos a serem inseridos na solução; $L_C = \langle e_1, \dots, e_{|C|} \rangle$ é uma lista ordenada de C , tal que:

$$(\forall i, |C| > i \geq 1) (((meta = min) \rightarrow fg(e_i) \leq fg(e_{i+1})) \wedge ((meta = max) \rightarrow fg(e_i) \geq fg(e_{i+1})))$$

Cálculo e adaptação da função gulosa

O termo "adaptativo" na denominação do método GRASP se refere ao cálculo da função gulosa para os elementos de C . Na maioria dos algoritmos gulosos, a ordenação dos elementos de C é feita uma única vez, e as inserções dos elementos só levam em consideração o valor desta função.

Em procedimentos GRASP, a idéia é utilizar um mecanismo mais inteligente, onde se leve em consideração a solução parcial construída. A função gulosa tem como parâmetros o elemento a ser inserido e a solução parcial já construída, alterando portanto, a noção de método "miope". A função é na verdade semi-gulosa.

Obviamente, a adaptatividade em alguns casos pode não ser possível, devido a estrutura do problema e em alguns casos pode ser desnecessária. Certamente esta é a parte crítica da fase de construção e a de maior esforço computacional. A escolha da estrutura de dados utilizada para armazenar a solução parcial e o conjunto C é que irá determinar a eficiência do algoritmo.

Construção da Lista Restrita de Candidatos

Para a construção da lista restrita de candidatos (LRC), o GRASP básico utiliza as seguintes formas de restrição aos elementos de C :

- restrição por cardinalidade, onde dado um parâmetro de entrada β , $LRC = \{e_i \in L_C | 1 \leq i \leq \beta\}$
- restrição por valor, onde dado um parâmetro de entrada α ,
 - $(meta = min) \rightarrow LRC = \{e \in C | f(e) \leq f(e_1) + \alpha(f(e_n) - f(e_1))\}$
 - $(meta = max) \rightarrow LRC = \{e \in C | f(e) \geq f(e_1) - \alpha(f(e_1) - f(e_n))\}$
- restrição por ambas estratégias.

Apesar do próprio algoritmo de Hart e Shogan utilizar a restrição por cardinalidade para LRC esta estratégia pode não ser muito adequada.

Restringir por cardinalidade para determinados problemas implicará em ordenação total dos elementos não incluídos na solução parcial a cada passo de inserção da fase de construção, aumentando consideravelmente a complexidade desta fase. Além disso, restringir por cardinalidade incorre em um problema durante as inserções: podem ocorrer classes de equivalência entre os elementos, ou seja, empates no valor de fg e desta forma a ordenação dos elementos na lista influenciará a diversidade das soluções construídas. Se ocorrer o empate e não houver um mecanismo tal como embaralhamento do vetor com os elementos pertencentes a C , estará sendo inserido no algoritmo um fator contribuinte para prisão em ótimos locais, pois haverá menor diversidade nas soluções iniciais.

A restrição por valor é mais sutil e eficiente pois verificar o menor e maior valor da função gulosa e selecionar os elementos dentro do intervalo definido pelo parâmetro requer menos esforço ($O(n)$) que ordenação total dos elementos ($O(n \cdot \log n)$), e inclui os elementos em LRC de forma justa.

Escolha aleatória

Em um GRASP básico, a escolha do elemento da LRC a ser inserido é totalmente aleatória, ou seja, a distribuição de probabilidade de escolha do elemento é uniforme. Dessa forma, cada elemento tem probabilidade $\frac{1}{|LRC|}$.

Probabilidade igual para a escolha dos elementos permite maior diversidade para as soluções mas, entretanto, pode gerar soluções bastante ruins com grande frequência, caso a configuração de α e β não seja bem ajustada.

4.2.2 Fase de Melhoria

A fase de melhoria do GRASP é um procedimento de busca local aplicado à solução inicial gerada na fase de construção. Para a fase de melhoria, não existe um modelo genérico para todos os problemas.

Cada problema terá suas funções de perturbação e respectivas estruturas de vizinhança particulares, e ficará a critério do analista (ou dependente da dificuldade ou estrutura do problema) qual o tipo de algoritmo de busca local: primeira melhoria ou exaustivo, determinístico ou aleatório, parando ao percorrer uma distância máxima pré-definida ou ao atingir um ótimo local.

Se a fase de construção GRASP cumpre o objetivo de reduzir a distância entre a solução inicial e seu respectivo ótimo local, então pode ser interessante o uso de um procedimento exaustivo (*best accept*), para que se avalie mais densamente a vizinhança.

4.2.3 Características e vantagens do método

- Fácil paralelização: a principal vantagem que o GRASP adquire ao trabalhar de forma totalmente aleatória, somada ao fato de que cada iteração é independente das anteriores, é a fácil paralelização: cada processador recebe uma seqüência diferente de números aleatórios (sem intersecção alguma com as demais) e compartilha um espaço em comum para guardar a melhor solução encontrada.

- Pouco uso de memória: o GRASP básico praticamente não guarda informação das iterações já feitas, com exceção da melhor solução encontrada, se equivalendo neste aspecto ao *Simulated Annealing*. A estratégia do método é utilizar o tempo explorando diferentes regiões do espaço de soluções através de amostragem, ao invés de aprender sobre a estrutura deste. Esta característica pode tornar o método excessivamente dependente de aleatoriedade.

- Simplicidade: ao contrário das outras metaheurísticas, GRASP é simples e bastante modular. Pode-se utilizar para uma mesma fase de construção, diferentes procedimentos de busca local e vice e versa. No GRASP se configura o valor de α e se utilizado, o parâmetro β .

Capítulo 5

Algoritmo proposto

O algoritmo descrito originalmente na Seção 3.2, mais especificamente na Figura 3.1, será modificado nesta seção, tornando-o não determinístico.

5.1 Introdução

O algoritmo desenvolvido por Eades, Lin e Smyth[6] demonstrou uma forma heurística e relativamente eficiente para encontrar conjunto de arestas de retorno, utilizando-se da relação existente entre a ordenação de vértices[8].

Este algoritmo possui característica determinística, e sua heurística resume-se em avaliar a cardinalidade dos vértices para construir uma lista ordenada que será a base para encontrar o conjunto de retorno.

Como demonstrado na Seção 4, existe uma forma dinâmica de buscar conjuntos de retorno, utilizando heurística híbrida, que segundo estudos[4][5] tem melhorado em alguns casos, o tempo de processamento, em outros a qualidade dos resultados obtidos, pois utiliza um método adaptativo e multi-partida.

Esta forma foi estudada e aprimorada por diversos pesquisadores como em Pardalos[5], Pitsoulis[4] e outros.

Desta forma, propõe-se unir o algoritmo ELSFAS desenvolvido por Eades[6], com a característica não determinista do GRASP, tornando assim o algoritmo ELSFAS uma heurística híbrida, objetivando-se conseguir melhores soluções.

5.2 Algoritmo Proposto

Observando a Figura 5.1, podemos constatar algumas modificações no algoritmo com relação ao ELSFAS original.

Como tanto o ELSFAS quanto o GRASP está descrito em detalhes nas seções anteriores, será descrito aqui o funcionamento e os critérios adotados referente as modificações apresentadas na Figura 5.1.

Inicialmente (linha 3) o algoritmo constrói uma lista ordenada dos vértices, tendo como critério de ordenação a cardinalidade do vértice, sendo portanto o resultado do cálculo $d^+(u) - d^-(u)$ necessário para todos os vértices do grafo. Isto é necessário para estimar o custo dos elementos candidatos, o resultado prático será a construção de uma lista onde os vértices estão em ordem decrescente de cardinalidade. Após construída a lista de vértices ordenados de forma decrescente, o algoritmo inicia a classificação dos vértices iniciando-se pela busca de sumidouros, fontes, e na ausência destes, dos vértices com maior cardinalidade. As regras para classificação não foram alteradas permanecendo idênticas ao ELSFAS original.

Função ELSFASGRASP**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. Estimar o custo dos elementos (construir lista de vértices ordem decrescente de cardinalidade)
4. **enquanto** ($G \neq \emptyset$) **faça**
5. **se** existe sumidouro u em G **então**
6. colocar u no início de S_2
7. $v = u$
8. **senão se** existe fonte u em G **então**
9. colocar u no final de S_1
10. $v = u$
11. **senão**
12. Construa uma lista restrita de candidatos
13. Solução \leftarrow vértice escolhido aleatoriamente da lista restrita de candidatos
14. $u \leftarrow$ Solução
15. colocar u no começo de S_1
16. $v = u$
17. **fim se**
18. remover u e todas suas arestas incidentes de G
19. Recalcular e atualizar cardinalidade dos vértices da lista para obter candidatos
20. **fim enquanto**
21. concatenar S_1 e S_2 em uma lista S
22. **retornar** o conjunto de arestas para trás da ordenação S

Figura 5.1: Pseudocódigo do algoritmo após modificação

O critério adotado para a escolha do vértice, quando o grafo não contiver nem fontes nem sumidouros, podem ser observados nas linhas 12 e 13, e utiliza a seguinte estratégia:

1. (linha 12) Construir a lista restrita de candidatos: será eleito para esta lista os vértices com maior cardinalidade;
2. Havendo mais de um vértice que satisfaz o critério acima, efetua-se o sorteio aleatório de um dos vértices que compõem a lista restrita de vértices candidatos;
3. (linha 13) O elemento escolhido aleatoriamente será a resposta para ser adicionado a lista de classificação de vértices.

A cada vértice encontrado, este é removido do grafo (linha 18 da Figura 5.1), e removido todas as arestas incidentes ao vértice. Logo após é recalculado e atualizado a cardinalidade dos vértices afetados pela remoção (linha 19 da Figura 5.1), e inicia-se o ciclo novamente a procura de vértices sumidouros, fontes ou vértices de maior cardinalidade, até que o grafo esteja vazio.

Concluído esta parte de classificação, o resultado será uma lista S com os vértices ordenados segundo a classificação do algoritmo, sendo que agora, utilizando-se do critério não determinístico.

As implicações da escolha aleatório dos vértices na lista S , muda completamente a ordem de classificação dos vértices, e conforme aplicado no ELSFAS, as arestas que apontam para trás na ordenação da lista S compuseram o conjunto de retorno.

Capítulo 6

Resultados obtidos

Nesta seção estão relacionados os testes realizados com o algoritmo original e o algoritmo modificado, ELFAS e ELSFASGRASP respectivamente. Algumas observações a respeito dos resultados é descrita para melhor entendimento e avaliação.

6.1 Descrição da Metodologia dos Testes

Para coletar os dados para montagem dos gráficos foram adotados os seguintes procedimentos:

1. ELSFAS: Uma vez sendo determinístico, todas as execuções produzirão respostas idênticas para o mesmo grafo, sendo assim, foi efetuado uma execução do algoritmo ELSFAS para cada grafo, e coletado o número de arestas de retorno encontrado;
2. ELSFASGRASP: Devido a característica não determinística, cada execução o algoritmo retorna um conjunto de retorno diferente, em virtude da aleatoriedade na escolha. Sendo assim para obtenção de um número representativo para comparar com o resultado do ELSFAS, foi calculado a média do resultado de cinco execuções do algoritmo ELSFASGRASP.

Os testes foram executados em um computador com processador Intel, pentium 4 3GHz e com 2 GB de memória RAM. O sistema operacional utilizado foi o Microsoft Windows XP versão 2002 SP2. A linguagem escolhida para implementação foi o Java e a IDE utilizada para implementação e execução dos testes foi o NetBeans.

6.2 Gráficos Comparativos

Os testes apontaram um ligeiro ganho na qualidade das respostas utilizando-se o ELSFASGRASP. A aleatoriedade na escolha dos vértices melhoraram entre 1,9 e 3,1 por cento o número de arestas no conjunto de retorno encontrado.

Os gráficos a seguir demonstram detalhadamente o teste comparativo, sendo que em poucos grafos o ELSFAS se mostrou mais eficiente que o ELSFASGRASP.

Além dos gráficos para aferição da qualidade dos conjuntos de retorno obtido considerando os dois algoritmos, fez-se um comparativo do tempo gasto por cada um dos algoritmos para obter os resultados.

A conclusão que se obteve é que houve melhora significativa nos conjuntos de retorno processados pelo ELSFASGRASP, e sobretudo a um custo no incremento de tempo de processamento insignificante, conforme pode ser visto na Figura 6.2.5.

6.2.1 Análise de Grafos com 50 Vértices

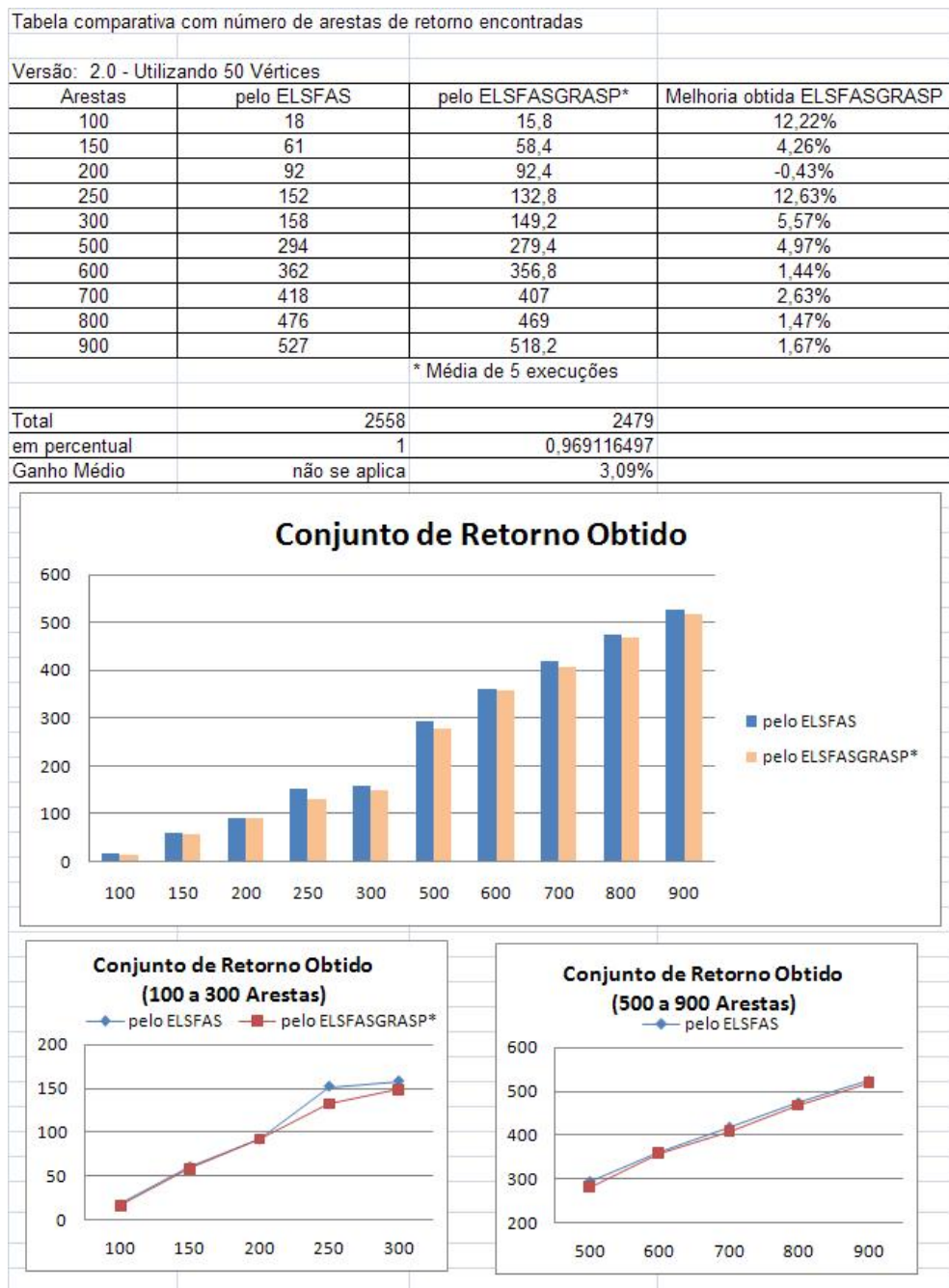


Figura 6.1: Teste com 50 Vértices

6.2.2 Análise de Grafos com 100 Vértices

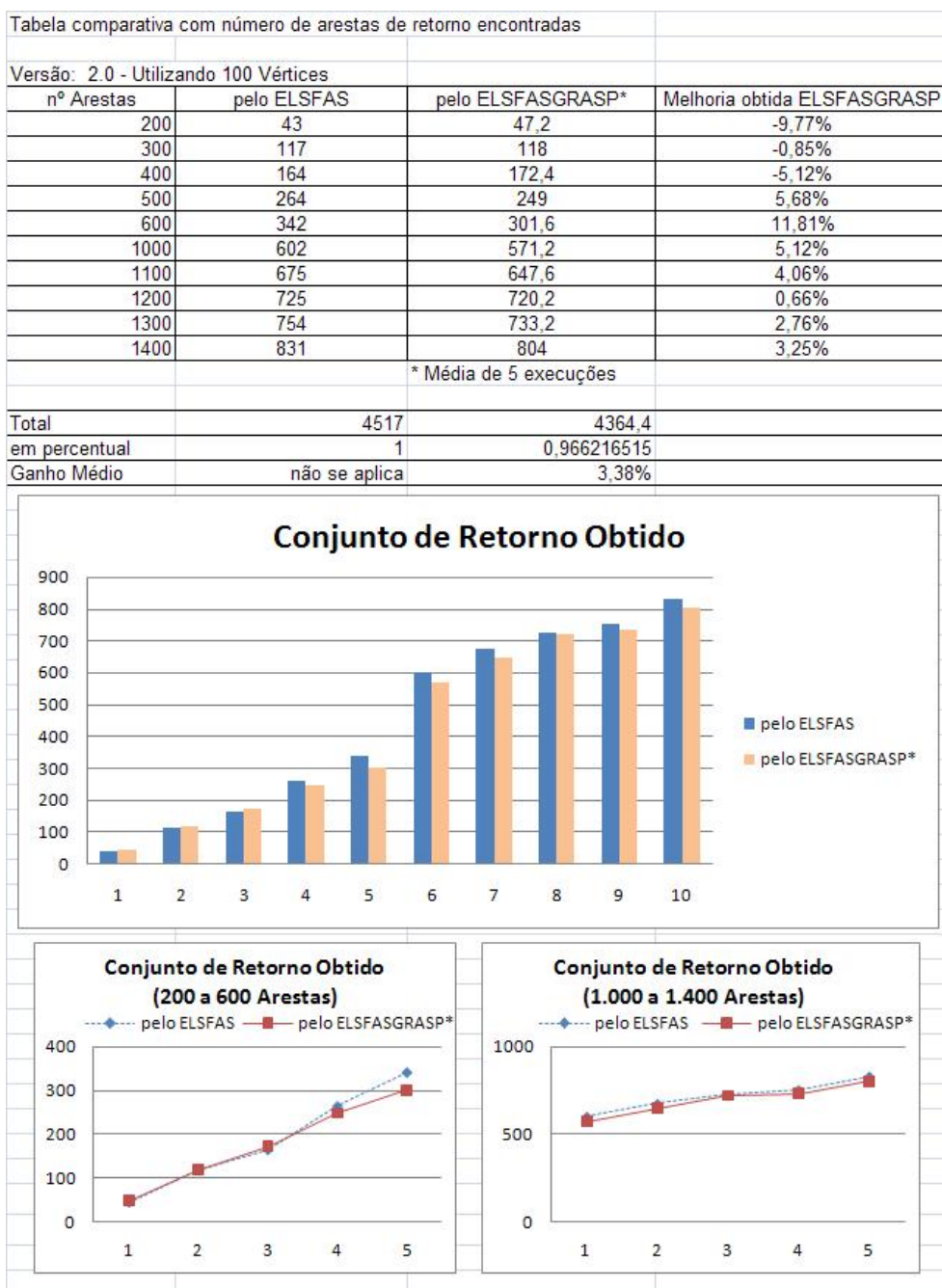


Figura 6.2: Teste com 100 Vértices

6.2.3 Análise de Grafos com 500 Vértices

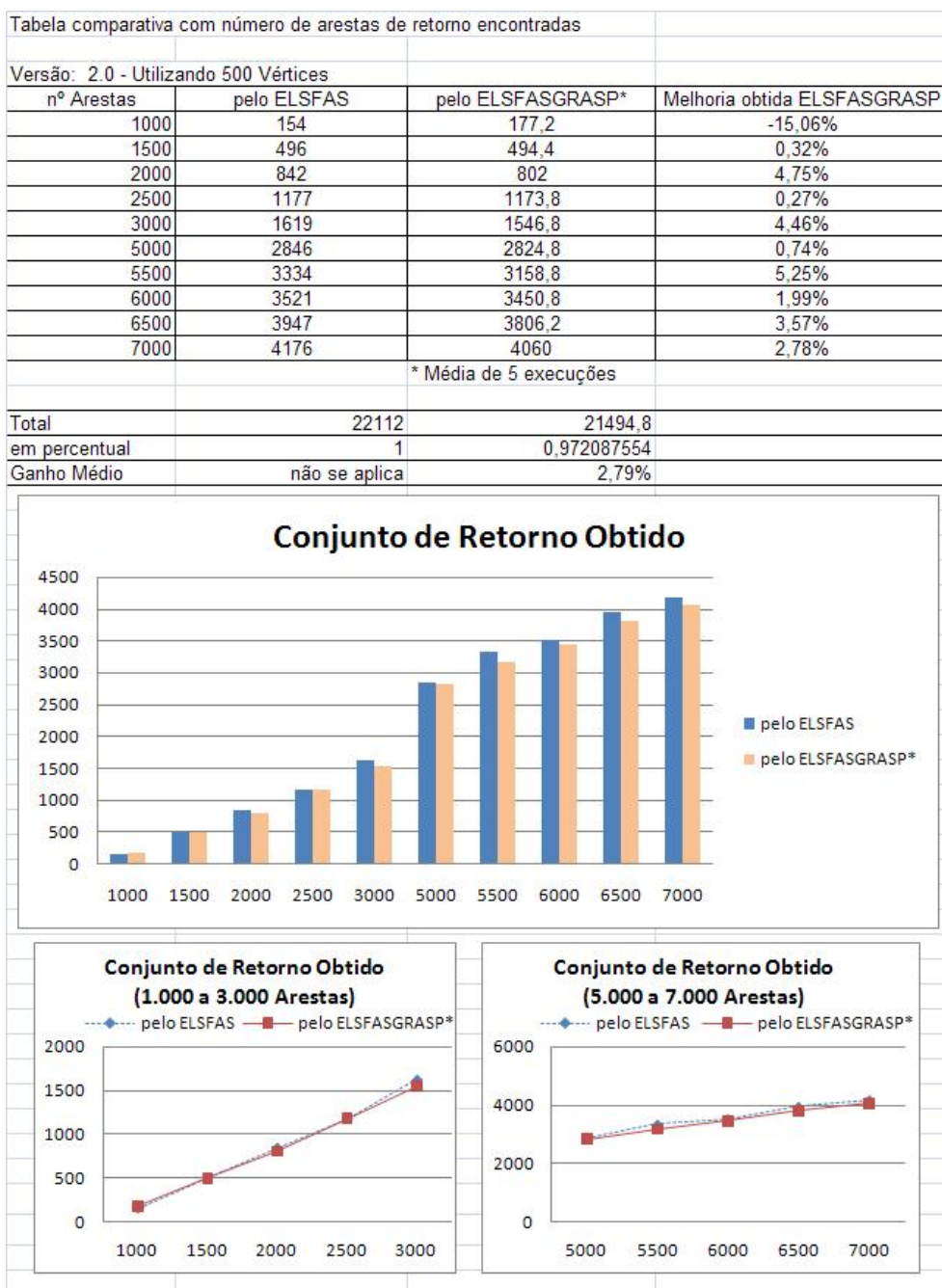


Figura 6.3: Teste com 500 Vértices

6.2.4 Análise de Grafos com 1.000 Vértices

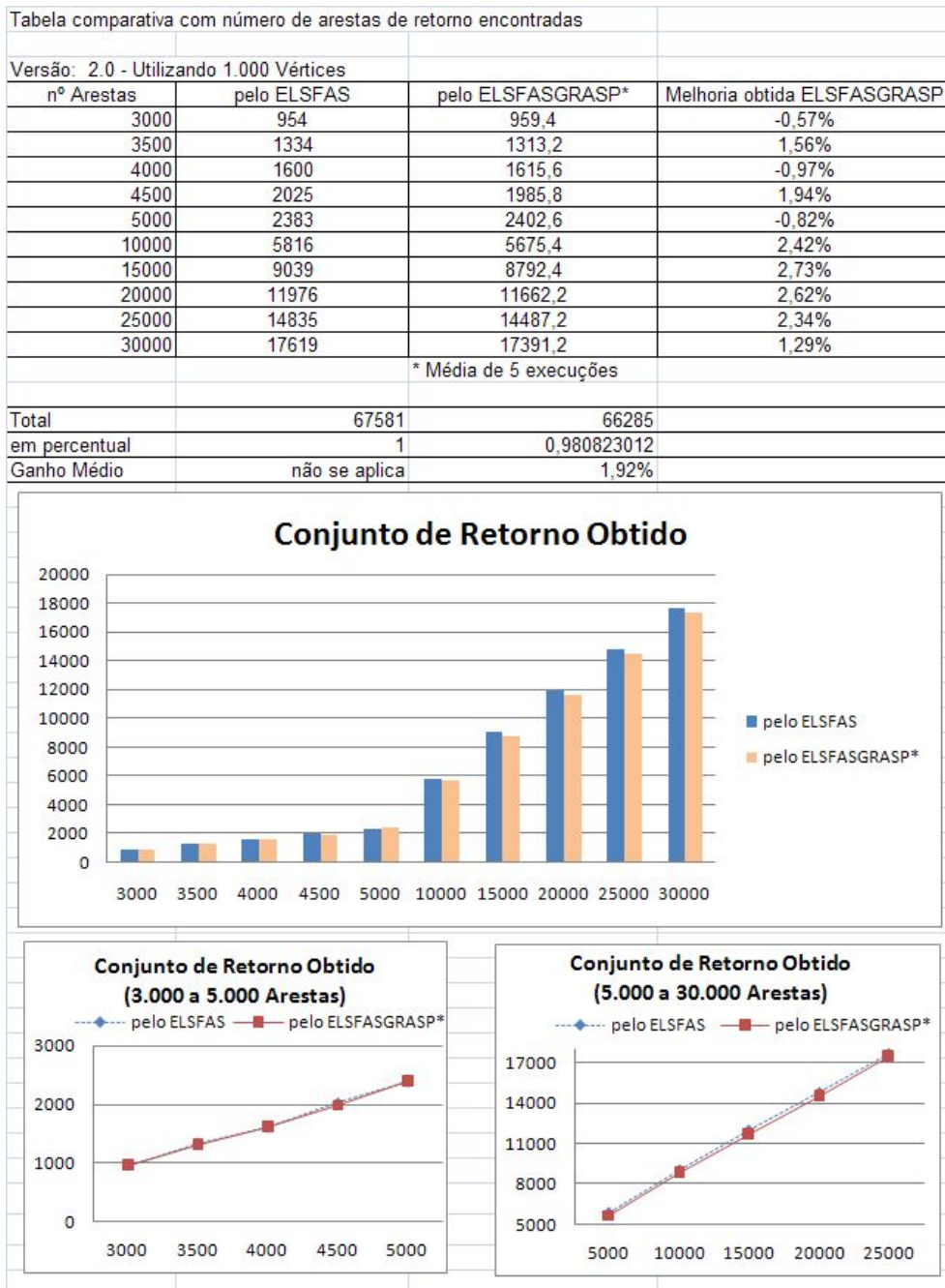


Figura 6.4: Teste com 1.000 Vértices

6.2.5 Tempo de Processamento dos Algoritmos

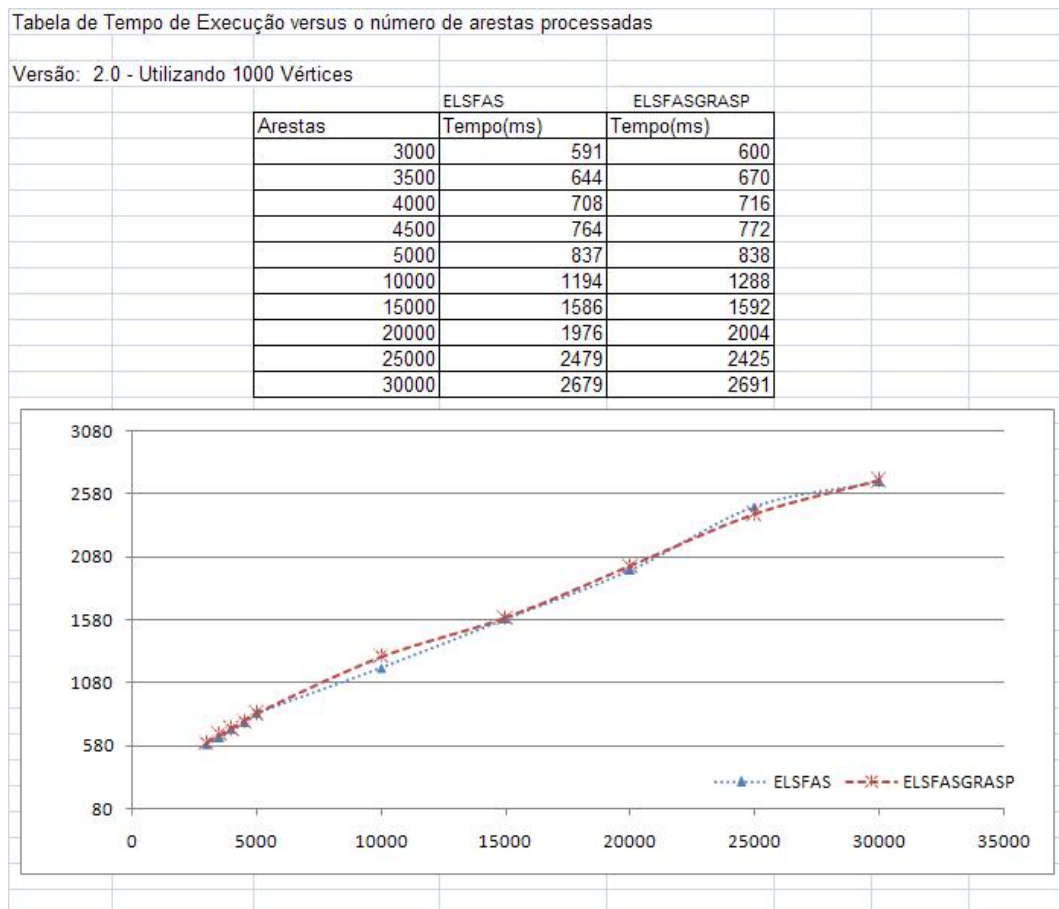


Figura 6.5: Comparativo dos algoritmos: Tempo gasto com processamento

Capítulo 7

Conclusão

No presente trabalho monográfico foi efetuado a revisão de grafos e grafos simples, objetivando dar suporte ao assunto conjunto de retorno. Uma breve descrição sobre o problema do conjunto de retorno foi feita, assim como o algoritmo ELSFAS e o GRASP.

O problema do conjunto de retorno foi estudado por alguns autores que propuseram formas de resolvê-lo, como foi o caso dos autores Eades, Lin e Smyth[6], quando desenvolveram o algoritmo utilizado neste trabalho, o qual foi chamado de ELSFAS.

Após aperfeiçoarmos este algoritmo, verificou-se produtiva a alteração, pois produziu, em média, melhores respostas se comparado ao algoritmo ELSFAS original.

A contribuição deste trabalho está no fato de que o algoritmo aperfeiçoado gerou melhores resultados, sem contudo, incrementar de forma significativa o tempo computacional gasto para processamento.

A questão da busca local, verificada no procedimento GRASP, não foi abordada neste trabalho monográfico, o que poderá servir para futuros aperfeiçoamentos em outros trabalhos.

Referências Bibliográficas

- [1] BARBOSA, Marco Aurélio Lopes. *Algoritmos para Encontrar Conjuntos de Retorno*. 2005. 68 f. Dissertação (Mestrado em Ciência da Computação). Universidade Estadual de Maringá, Maringá.
- [2] BOAVENTURA NETTO, P.O., *Grafos: Teoria, Modelos e Algoritmos* xx ed. São Paulo. Ed. Edgard Blücher Ltda. 1996. p.x-xx.
- [3] BONDY, J.A. e MURTY, U.S.R. *Graph Theory With Applications*. 5a ed. New York/NY. Elsevier Science Publishing. 1982. p.1-23.
- [4] L. Pitsoulis and M. Resende. Greedy randomized adaptative search procedures, 2001.
- [5] O. Pardalos, T. Qian, and M. Resende. A greedy randomized adaptative search procedure for feedback vertex set, 1999.
- [6] 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.
- [7] P. Festa, P. Pardalos, and M. Resende. *Feedback set Problems* 1999.
- [8] D.H. Younger. *Minimum feedback arc sets for a directed graph*. IEEE Transactions on Circuit Theory, Vol. CT-10, p.238-245, 1963.