DESENVOLVIMENTO E ANÁLISE EXPERIMENTAL DE HEURÍSTICAS GRASP PARA UMA GENERALIZAÇÃO DO PROBLEMA DA ÁRVORE GERADORA MÍNIMA

Cristiane Maria Santos Ferreira
Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria 156 - Bloco E - 3º andar
São Domingos - Niterói - RJ - CEP: 24210-240
cferreira@ic.uff.br

Luís Satoru Ochi
Instituto de Computação - Universidade Federal Fluminense
Rua Passo da Pátria 156 - Bloco E - 3º andar
São Domingos - Niterói - RJ - CEP: 24210-240
satoru@ic.uff.br

Elder Magalhães Macambira
Departamento de Estatística - Universidade Federal da Paraíba
Cidade Universitária s/n – João Pessoa – PB – CEP: 58051-900
elder@de.ufpb.br

RESUMO

Dado um grafo *G* cujos vértices estão divididos em grupos, o Problema da Árvore de Cobertura Mínima Generalizado (PAGMG) consiste em encontrar uma árvore que cubra todos os grupos de *G*, de forma que a soma do custo das arestas de *T* seja mínima. Aplicações deste problema são encontradas em redes de telecomunicações, redes de distribuição de energia elétrica, e em sistemas de irrigação agrícola. Este trabalho apresenta três versões da heurística GRASP para o PAGMG, uma versão tradicional, uma versão que utiliza a técnica de reconexão de caminhos e uma versão que aplica um procedimento de busca local iterada. São apresentados testes comparativos com algoritmos da literatura ilustrando a eficiência dos métodos propostos.

PALAVRAS CHAVE. Metaheurísticas, Otimização em Grafos, Programação com Memória Adaptativa

ABSTRACT

Given a graph *G* whose vertices are divided into clusters, the Generalized Minimum Spanning Tree Problem consists of finding a tree spanning all clusters of *G*, in such a way that minimizes the total cost of the edges. Applications of this problem can be found in telecommunications networks, power distribution networks and in agricultural irrigation systems. This paper presents three versions of GRASP heuristic to solve this problem: a conventional one, a version using path relinking and a version that applies an iterated local search procedure. We present comparative testes with algorithms proposed in literature to illustrate the effectiveness of the proposed methods.

KEYWORDS. Meta-heuristics, Graph Optimization, Adaptive Memory Programming

1. Introdução

Este trabalho aborda uma generalização do clássico Problema da Árvore Geradora Mínima, em que o conjunto de vértices está dividido em grupos e o objetivo é determinar uma árvore de custo mínimo que cubra exatamente um vértice de cada grupo. Este problema é denominado Problema da Árvore Geradora Mínima Generalizado (*Generalized Minimum Spanning Tree Problem* - PAGMG) e foi introduzido por [14].

Aplicações para o PAGMG podem ser encontradas nas áreas de *design* de redes de telecomunicações, onde redes locais precisam ser interconectadas para formar redes maiores [3], na localização de instalações e mesmo na descrição de processos físicos que compõem redes [12].

O presente trabalho tem como objetivo propor algumas versões da heurística GRASP para o PAGMG. Duas delas incorporam o conceito de memória adaptativa, tentando armazenar informações relevantes das iterações já efetuadas pelo algoritmo, com o intuito de tentar melhorar o trabalho de busca nas iterações seguintes. Esse armazenamento é baseado na noção de *conjunto elite*, que armazena as melhores soluções distintas encontradas no decorrer da execução.

As versões GRASP que utilizam conjunto elite, além do modelo tradicional de construção-aprimoramento, também utilizam como mecanismo adicional a técnica de *Reconexão de Caminhos* e um procedimento de busca local iterada.

O trabalho está organizado como segue: a próxima seção traz uma descrição do problema e uma revisão dos trabalhos da literatura; na seção 3 são descritas as versões GRASP propostas; na seção 4 encontram-se os resultados computacionais e, por fim, na seção 5 são apresentadas as conclusões e algumas sugestões de trabalhos futuros.

2. O Problema da Árvore Geradora Mínima Generalizado

O PAGMG pode ser definido sobre um grafo não-direcionado $G(V,E)$ onde V representa o conjunto de vértices e E o conjunto de arestas. Cada aresta e possui um custo $c_e \in R^+$. Neste problema, o conjunto de vértices V é particionado em m grupos disjuntos $V_1, ..., V_m$ e o objetivo é determinar uma árvore de custo mínimo que cubra exatamente um vértice de cada grupo.

Diferentemente do Problema da Árvore Geradora Mínima, o PAGMG é classificado como NP-difícil [14] e de nosso conhecimento existe apenas um número reduzido de trabalhos relacionados até o presente momento.

A Figura 1 ilustra o exemplo de uma solução para o PAGMG em um grafo cujo conjunto de vértices está particionado em quatro grupos. No exemplo, as arestas (3,5), (3,7) e (7,11) compoem a solução.

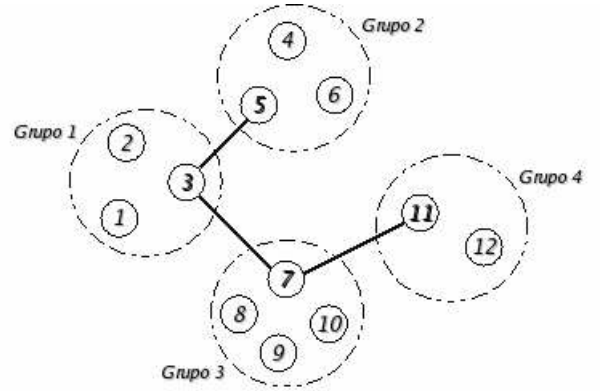

Figura 1 - Exemplo de uma solução para o Problema da Árvore de Cobertura Mínima

Aplicações do problema podem ser encontradas, por exemplo, na área de telecomunicações, onde as redes regionais precisam ser interconectadas por uma árvore que contenha uma conexão para cada sub-rede. Para essa interconexão, um vértice de cada sub-rede deve ser selecionado como *gateway* [14]. O PAGMG também se aplica a vários problemas relacionados à localização de instalações que precisam estar conectadas por meio de rodovias ou

links de comunicação. Um exemplo acontece quando uma empresa quer estabelecer centros regionais de distribuição para suas lojas e precisa selecionar um ponto em cada região a fim de construir uma rede de comunicação interconectando esses centros [16]. Outra aplicação é descrita por [1] na área de irrigação agrícola.

3. Trabalhos Relacionados

O PAGMG foi primeiramente abordado por [14], que demonstrou que o problema é fortemente NP-difícil. Myung também propôs quatro formulações de programação inteira mista e um algoritmo *branch-and-bound*, que resolveu para a otimalidade instâncias com até 100 vértices.

Dentre os principais trabalhos, pode-se citar [4], que na sua tese de doutorado fez uma investigação da estrutura poliédrica do PAGMG. A tese também propôs um algoritmo *branch-and-cut* e quatro famílias de desigualdades válidas: desigualdades de Hammock, de ciclos ímpares, de casamento de ciclos ímpares e de "buracos" ímpares. O *branch-and-cut* conseguiu resolver todas as instâncias euclideanas com até 160 vértices e todas as instâncias com custos aleatórios com até 200 vértices. Também resolveu 150 das 169 instâncias adaptadas do TSPLIB, considerando um tempo limite de duas horas. Comparando com o algoritmo de [14], o algoritmo de Feremans apresentou limites melhores e conseguiu resolver instâncias bem maiores.

[9] apresentam uma comparação minuciosa de alguns algoritmos construtivos e duas heurísticas.

Os construtivos comparados são adaptações dos clássicos algoritmos de Kruskal, Prim e Sollin para o Problema da Árvore Geradora Mínima. As adaptações de Kruskal e Sollin apresentaram resultados melhores que Prim, e dentre os dois, há uma ligeira vantagem para o algoritmo baseado em Kruskal. As heurísticas propostas consistem de: uma busca local bastante simples, porém eficiente, e um algoritmo genético.

Os autores compararam os algoritmos nas instâncias adaptadas do TSPLIB, das quais 150 possuem o valor ótimo conhecido. As soluções do procedimento de busca local e do algoritmo genético ficaram, em média, a 0,07% e 0,01% da otimalidade, respectivamente. O algoritmo genético também apresentou um desempenho ligeiramente melhor nos testes com instâncias cujo ótimo não é conhecido e com instâncias aleatórias propostas pelos autores. Vale colocar que tal algoritmo consome aproximadamente o dobro do tempo da busca local.

4. GRASP

GRASP (*Greed Randomized Adaptive Search Procedure*) é uma meta-heurística *multi-start*, proposta por [6], em que cada iteração é composta de uma fase de construção e de uma fase aprimoramento (busca local). Durante o processo, a solução incubente é armazenada e atualizada sempre que a fase de aprimoramento resulta em uma solução melhor. Ao final de um número estabelecido de iterações, o algoritmo retorna a melhor solução encontrada.

No GRASP, a fase de construção é iterativa, adaptativa, semi-gulosa e randômica. Contudo as iterações GRASP são totalmente independentes, não havendo nenhum tipo de memória nem aprendizagem no decorrer da execução. Daí muitas vezes ele ser chamado de *multistart*. Conseqüentemente, nenhuma informação relevante obtida nas iterações anteriores são incorporadas no processo de busca das iterações remanescentes.

Muitos pesquisadores vêm então utilizando técnicas adicionais, no intuito de aproveitar de alguma forma resultados obtidos para tentar encontrar melhores soluções, como acontece com o GRASP reativo [7] e com o uso de reconexão de caminhos [10,9], por exemplo.

Neste trabalho, são propostas três versões GRASP para o PAGMG: um GRASP puro (GRASP), que segue o modelo tradiocional explicado anteriomente; um GRASP com reconexão de caminhos (GRASP+RC) e um GRASP que além de reconexão de caminhos, incopora um procedimento de busca local iterada (GRASP+RC+ILS).

Nesta seção serão apresentados os algoritmos que compõem as três versões GRASP propostas: a heurística de construção, o procedimento de busca local, e os mecanismo adicionais de reconexão de caminhos e busca local iterada.

Heurística de Construção

Foram implementadas diferentes heurísticas construtivas, dentre elas adaptações dos algoritmos de Kruskal, Prim e uma heuristica baseada no cálculo das distâncias médias (HDM) para o PAGMG. Resultados preliminares demonstraram que a HDM apresenta o melhor desempenho médio. Essa heurística HDM será explicada a seguir.

O objetivo da HDM é selecionar um vértice $\gamma[i]$ em cada grupo V_i levando em consideração os grupos a que V_i poderá estar conectado na solução. A seleção de $\gamma[i]$ é feita com base em sua distância média a um conjunto $S_i \subseteq \{V \setminus V_i\}$,dado que S_i é composto por vértices que se encontram a uma distância máxima D_{max} do grupo V_i.

Ao início da construção, define-se aleatoriamente uma ordem para percorrer os grupos. Em seguida, a cada grupo V_i. percorrido, constrói-se o conjunto S_i. A seguir, calcula-se a distância de cada vértice $v \in V_i$ aos elementos de S_i. A distância $d\{v,Si\}$ de um vértice v a S_i é o custo médio das arestas entre v e os vértices presentes em S_i.

A seleção do vértice $\gamma[i]$ é feita aleatoriamente dentre os elementos de uma *Lista de candidatos Restritos* (LCR), formada pelo subconjunto dos melhores candidatos como feito tradicionalmente na fase de construção do GRASP. Neste caso, a LCR é composta de cada vértice v cuja distância $d\{v,Si\}$ é menor ou igual a $d_{min} + \alpha(d_{max} - d_{min})$, onde d_{min} e d_{max} são a menor e a maior distância de todos os vértices do grupo V_i, respectivamente. O parâmetro $\alpha \in [0,1]$ indica o quão gulosa será essa seleção. Para $\alpha = 1$, por exemplo, o comportamento do algoritmo é totalmente aleatório [6]. Uma observação importante é que na construção de S_i, nos grupos que já possuem vértices fixados na solução, apenas estes integrarão S_i.

Busca Local

A busca local implementada foi utilizada no PAGMG primeiramente por Golden et. al. [9]. A fim de facilitar o cálculo da vizinhança, considera-se cada solução como um conjunto de m vértices, um de cada grupo. A avaliação das soluções é feita calculando-se a árvore de custo mínimo que cubra os vértices presentes na solução. Dada uma solução inicial $\underline{s}$, o procedimento pode ser descrito basicamente nas três etapas abaixo:

- Aleatoriamente define-se uma ordem em que os grupos serão visitados
- A cada visita a um grupo V_i, calcula-se todos os vizinhos de s em relação a V_i. A solução corrente passa a ser o melhor vizinho s', caso este seja melhor que s.
- Repete-se o passo 2 até que m grupos sejam visitados sem que haja melhoras na solução corrente.

Considerando que s é composta pelos vértices $\gamma[1], ..., \gamma[m]$, sua vizinhança em relação ao grupo V_i são todas as soluções $s' = \{\gamma'[1], ..., \gamma'[m]\}$ em que $\gamma'[i] \neq \gamma[i]$ e $\gamma'[j] = \gamma[j]$, $\forall j \in \{1,..,m\}\setminus\{i\}$. Ou seja, em relação ao grupo V_i, s possui $|V_i|$-1 vizinhos.

Reconexão de Caminhos

O mecanismo de Reconexão de Caminhos (*Path Relinking*), proposto originalmente para a busca tabu e *scatter search* por Glover, tem como objetivo encontrar soluções intermediárias entre duas boas soluções extremas. O algoritmo parte de uma *solução base* s_0, e passo-a-passo a transforma em s_d, chamada *solução guia*. Nesse trajeto, entende-se que pode ser encontrada uma solução melhor que s_0 e s_d.

No procedimento implementado neste trabalho, as soluções são representadas por conjuntos de m vértices e a sua avaliação é feita pelo cálculo da árvore geradora mínima - a mesma representação usada na busca local.

Se s_0 e s_d são duas soluções com d vértices diferentes entre si, um movimento de s_0 para s_d a substituição de um vértice em s_0 por um vértice de s_d. A reconexão de caminhos procede da seguinte forma: partindo de s_0 um movimento para s_d é gerado e tem-se uma solução intermediária s_1; a seguir mais um movimento é gerado, de s_1 a s_d; e assim por diante até que se chegue a uma solução s_{d-1},após $d-1$ movimentos realizados.

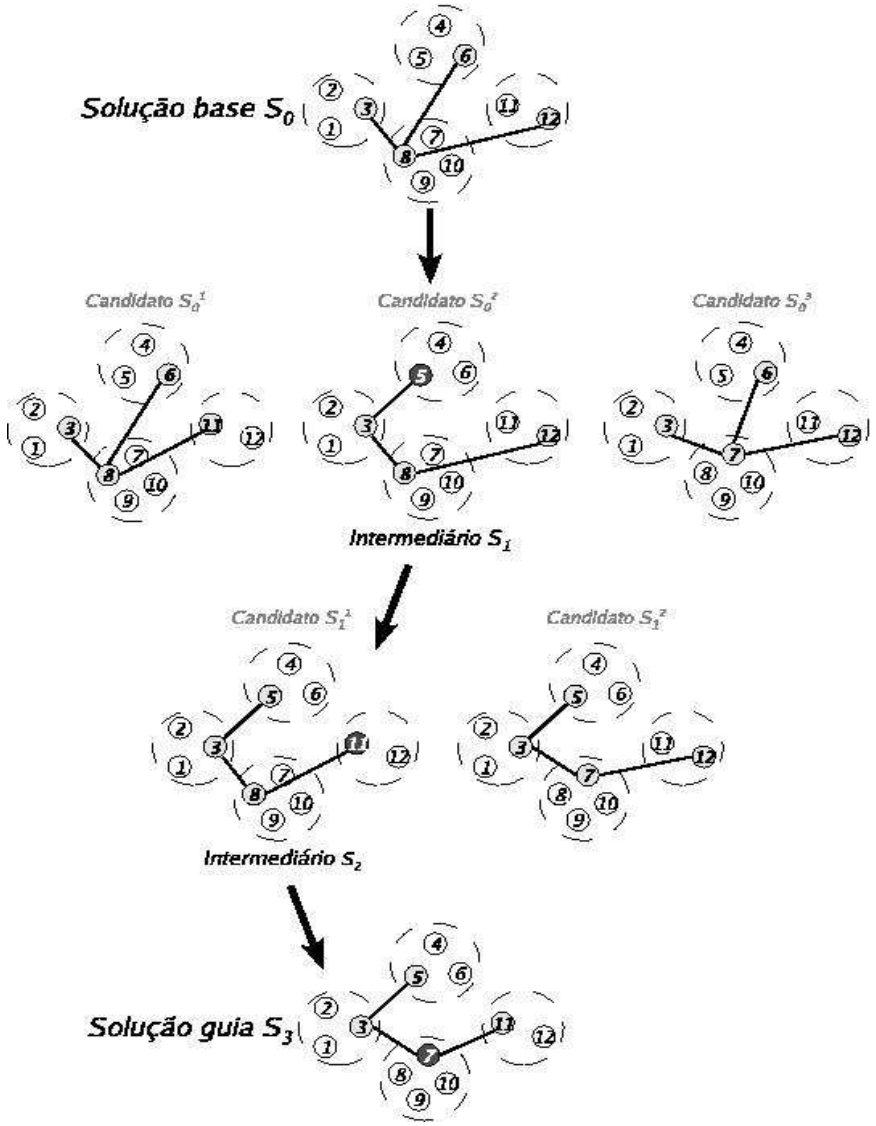

Figura 2 - Ilustrando o mecanismo de reconexão de caminhos

A Figura 2 ilustra um exemplo do funcionamento do mecanismo, nesse caso s_d é o grafo ilustrado na Figura 1. Como s_0 = {3, 6, 8, 12} e a solução guia s_d = {3, 5, 7, 11} há três diferenças entre as duas soluções, d= 3. Há, portanto, três candidatos para gerar s_1: {3, 6, 8, 11}, {3, 5, 8, 12} e {3, 6, 7, 12}. A melhor opção é a troca do vértice 6 por 5. Na próxima iteração, os candidatos são {3, 5, 8, 11} e {3, 5, 7, 12}. A primeira opção é selecionada para ser s_2. Para o movimento seguinte, a única possibilidade é a troca do vértice 8 por 7. Com essa mudança, tem-se a solução guia s_3 e o procedimento termina.

Neste trabalho, o GRASP com reconexão de caminhos mantém um conjunto elite *CE* contendo as *t* melhores distintas soluções encontradas durante a execução até o momento. Esse conjunto é atualizado a cada iteração, caso a solução gerada pela busca local seja melhor que a pior solução em *CE*. Sempre que o algoritmo atinge *max_iter* iterações sem atualização do conjunto elite, são removidas todas as suas soluções (com exceção da melhor).

A reconexão de caminhos é ativada a partir da iteração μ (dado de entrada), e ocorre sempre que a solução da busca local é no máximo *p*% pior que a melhor solução do conjunto elite. Aplica-se o procedimento entre *s* e a solução de *CE* que maximize *d* (em outras palavras,

fazer a reconexão com a solução elite *mais diferente de s*. A solução base por definição será sempre a melhor dentre as duas soluções extremas. Como estratégia de intensificação, o valor de *p* é proporcional ao número de iterações sem atualização de *CE*, ou seja, quanto mais difícil for encontrar uma solução com a qualidade das soluções elite, mais provável ocorrer reconexão de caminhos.

Busca Local Iterada

A idéia básica da Busca Local Iterada (*Iterated Local Search* - ILS) é aplicar pertubações sobre a solução corrente sempre que a mesma ebarrar em um ótimo local.

[17] citam quatro componentes básicos necessários para a implementação de um ILS: uma solução inicial, um procedimento de busca local, uma estratégia de pertubação e um critério de aceitação. A idéia básica será descrita a seguir: Aplica-se a busca local em uma solução inicial. A solução *s* gerada é tida como solução corrente. A seguir, *s* passa por uma pertubação seguida de uma busca local, gerando uma nova solução *s′*. Se o critério de aceitação for satisfeito, *s′* passa a ser a solução corrente. O processo se repete até que um critério de parada seja atingido.

Percebe-se que a natureza e o tamanho da pertubação sobre *s* são fundamentais para o bom funcionamento da busca. Se for aplicada uma pertubação pequena, a busca local possivelmente retornará ao mesmo ótimo local *s*. Em oposto, se a pertubação for muito grande, corre-se o risco de se estar reiniciando a busca.

A estratégia de pertubação adotada neste trabalho tem como base um grupo V_i. Dado que $\gamma[i]$ é o vértice de V_i presente na solução corrente *s*, a pertubação seleciona aleatoriamente um vértice $v \in \{ V_i \setminus \gamma[i]\}$ para compor a solução no lugar de $\gamma[i]$. O mesmo acontece com todos os grupos que se encontrem a uma distância máxima D_{max} de V_i. A solução proveniente dessa pertubação passa por uma busca local, gerando uma solução *s′*. O critério de aceitação é bastante simples: *s′* passa a ser a solução corrente se seu custo for menor que o de *s′*. A busca local é a mesma descrita na anteriormente.

Neste trabalho, a busca local iterada é aplicada sempre que o algoritmo atinge *max_iter* iterações sem atualizar o conjunto elite. Durante o ILS, sempre que uma solução melhor que a pior solução elite for encontrada, o conjunto elite é atualizado. No GRASP com busca local iterada, *CE* só é reiniciado se não for atualizado durante o ILS.

5. Resultados Computacionais

Nesta seção, estão apresentados os resultados comparativos obtidos por meio de testes empíricos realizados com o procedimento de busca local de [9] e as versões GRASP aqui propostas. Os testes computacionais foram realizados em um computador Pentium IV, com 3.2GHz e 1GB de memória principal, rodando um sistema operacional Linux versão 2.6.8-1.521. Os programas foram implementados na linguagem de programação C++ e compilados com g++ versão 4.0.2 utilizando as opções de compilação -o3 e march=pentium4. Os parâmetros utilizados foram p=2, t=4, D_{max} =c/3 (onde c é o custo médio das arestas), *max_iter* =50 e μ=20.

Os testes foram realizados sobre 169 instâncias com $48 \leq |V| \leq 226$, apresentadas em [3]. Essas instâncias foram geradas a partir de instâncias do repositório do Problema do Caixeiro Viajante. O agrupamento dos vértices foi realizado por [8] para o Problema do Caixeiro Viajante Generalizado, através de duas técnicas: *Center Clustering* e *Grid Clusterization*. [3] apresentou ainda cinco instâncias geográficas.

Dentre as instâncias utilizadas, 150 possuem valor ótimo conhecido. Para essas instâncias, cada algoritmo foi executado 10 vezes para cada instância. O critério de parada estipulado foi encontrar a solução ótima ou um tempo limite de 300s de execução. Todos os algoritmos conseguiram encontrar as soluções ótimas. O tempo médio necessário para a busca local de Golden foi de 1,06s, para o GRASP tradicional foi 2,81, para o GRASP+RC foi de 0,28s e para o GRASP+RC+ILS foi de 0,16s.

Nos testes com instâncias cujo valor ótimo não é conhecido, o critério de parada foi atingir um custo igual ou menor ao da melhor solução conhecida, ou um tempo de execução de

300s. Os resultados estão descritos na Tabela 1. A primeira coluna apresenta o nome da instância; nas três colunas seguintes tem-se o número de vértices, grupos e arestas, respectivamente; a coluna cinco traz o tempo da busca local de Golden; as últimas colunas apresentam os tempo médio das versões GRASP+RC e GRASP+RC+ILS, respectivamente. Por uma questão de espaço, não estão reportados os resultados do GRASP tradicional, mas estes são apenas um pouco melhores que os resultados de Golden.

Tabela 1- Resultados comparativos para instâncias cujo valor ótimo não é conhecido

Instância	**|V|**	**|E|**	**m**	**Alvo**	**BL**		**GRASP +RC**		**GRASP + RC + ILS**	
					Custo	**T(S)**	**Custo**	**T(s)**	**Custo**	**T(s)**
40d198	198	18841	40	7044	7044	1,27	7044	0,26	7044	**0,29**
41gr202	202	19532	41	242	242	7,53	242	**0,45**	242	0,78
45ts225	225	24650	45	62269	62268	39,92	62268,8	1,21	62268	**0,58**
46pr226	226	24626	46	55515	55515	0,014	55515	**0,013**	55515	0,017
67d198	198	19101	67	8283	8283	39,48	8283	6,75	8283	**4,04**
68gr202	202	19826	68	293	293	1,19	293	**0,67**	293	1,05
75ts225	225	24900	75	79019	79019	88,89	79019	0,32	79019	**0,29**
84pr226	226	25118	84	62527	62527	**0,08**	62527	0,15	62527	0,13
40d198	198	18772	40	7098	7098	**0,12**	7098	0,29	7098	0,59
41gr202	202	19303	41	232	232	19,82	232	2,07	232	**1,71**
45ts225	225	24726	45	60588	60641,6	300	60588	61,02	60588	**23,18**
50pr226	226	24711	50	56721	56721	0,033	56721	**0,01**	56721	**0,01**
32d198	198	18372	32	6501	6501	**0,028**	6501	0,04	6501	0,04
31gr202	202	18872	31	203	203	0,33	203	**0,17**	203	0,18
35ts225	225	24544	35	50813	50813	1,52	50813	**0,56**	50813	0,67
33pr226	226	24355	33	48249	48249	**0,01**	48249	0,03	48249	**0,01**
25d198	198	18149	25	6185	6185	**0,05**	6185	0,09	6185	0,07
21gr202	202	17904	21	177	177	**0,22**	177	0,52	177	0,36
25ts225	225	24300	25	40339	40339	0,28	40339	0,23	40339	**0,13**

Nas instâncias consideradas fáceis, em que os algoritmos conseguem encontrar o alvo em menos de 1s, percebe-se que não há grandes diferenças entre os três algoritmos. Isso acontece porque os algoritmos encontram as soluções alvo antes mesmo que o mecanismo de reconexão de caminhos seja ativado no GRASP+RC. As diferenças aparecem nas instâncias mais difíceis. Somente as versões que aplicam reconexão de caminhos conseguem encontrar todos os alvos no tempo estipulado.

Adicionalmente foi realizada uma bateria de testes para a análise da função de distribuição cumulativa da probabilidade dos algoritmos atingirem uma determinada solução alvo em relação ao tempo. O objetivo é verificar o tempo necessário para que os algoritmos encontrem soluções de boa qualidade.

Em cada experimento, foram computados os tempos de 100 execuções idependentes para cada algoritmo. Nos gráficos das Figuras 3 e 4, estão apresentados os resultados obtidos com a instância 40krob200, respectivamente com um alvo fácil e um difícil. Em cada curva, o i-ésimo menor tempo de execução rt_i foi associado à probabilidade p_i = (i-0,5)/100, gerando pontos (rt_i, p_i) para i = 1, ..., 100.

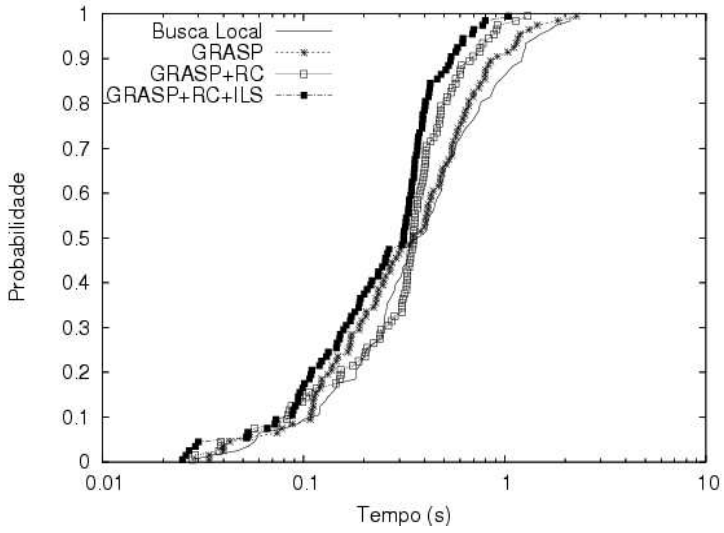

Figura 3 - Análise empírica dos algoritmos com a instância 40krob200 e alvo fácil

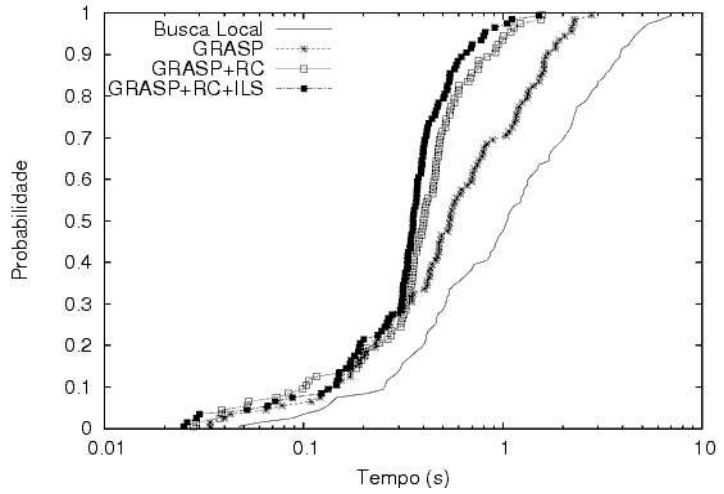

Figura 4 - Análise empírica dos algoritmos com a instância 40krob200 e alvo difícil

A análise dos gráficos pode ser feita considerando o alinhamento das curvas: a curva mais à esquerda indica o algoritmo que converge mais rápido para o valor alvo. Pode-se perceber que a busca local de [9] gasta um tempo maior para encontrar as soluções alvo, seguida da versão do GRASP tradicional. As curvas começam a divergir em torno de 0,4s, tempo médio em que a reconexão de caminhos é ativada. Na média, verificando os gráficos das figuras 3 e 4, percebe-se que o melhor desempenho (curva mais à esquerda) foi do GRASP+RC+ILS.

A influência do mecanismo de memória adaptativa através da reconexão de caminhos se torna mais evidente nos gráficos das Figuras 5 e 6, que apresentam os resultados para a instância 45ts225. Nesse caso, os algoritmos têm mais dificuldade para encontrar boas soluções. Nas Figuras 5 e 6, foram desconsideradas as execuções em que os algoritmos não conseguiram encontrar os alvos.

Mesmo considerando um alvo fácil, há uma diferença considerável nos tempos gastos pelos algoritmos. As versões com reconexão de caminhos convergem bem mais rápido, os algoritmos encontram a solução alvo em menos de 27s em 100\% das execuções, enquanto os outros dois algoritmos encontram em nos de 17%. Para um alvo difícil (figura 6), os algoritmos sem reconexão de caminhos não conseguem encontrar os alvos no tempo estipulado.

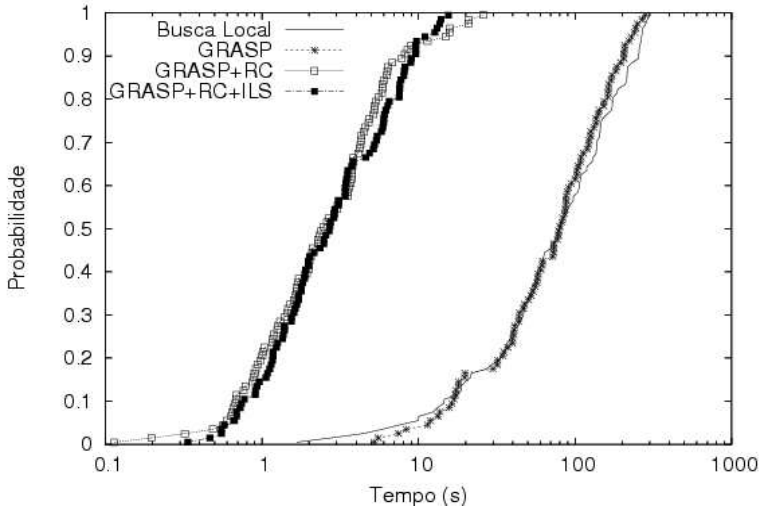

Figura 5 - Análise empírica dos algoritmos com a instância 45ts225 e alvo fácil

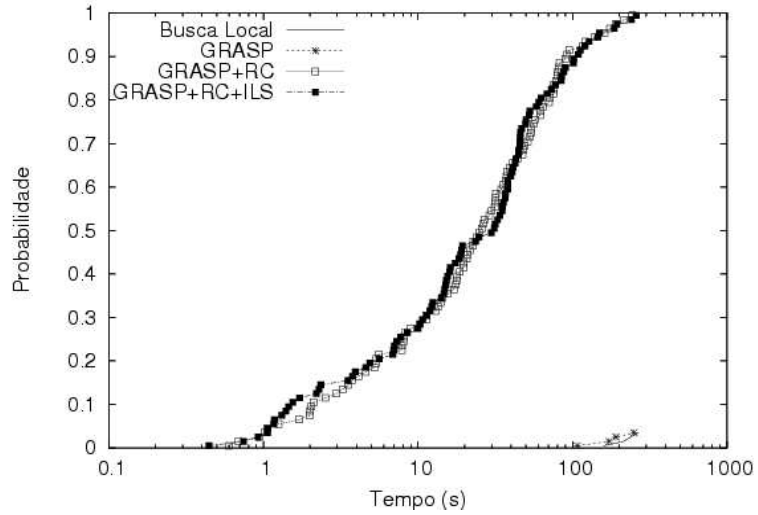

Figura 6 - Análise empírica dos algoritmos com a instância 45ts225 e alvo difícil

6. Conclusões e Trabalhos Futuros

Este trabalho abordou a utilização da heurística GRASP na resolução do PAGMG. Diante do fato de que o funcionamento do GRASP tradicional não se baseia em aprendizagem sobre a sua execução, o objetivo foi analisar a influência da utilização de memória para armazenar informações das melhores soluções obtidas até o momento (conjunto elite) sobre o desempenho do GRASP, além da influência do mecanismo de busca local iterada.

Neste contexto, foram propostas três versões da heurística: uma tradicional, composta por um algoritmo construtivo proposto neste trabalho e um algoritmo de busca local da literatura (GRASP); uma versão que difere da primeira por utilizar o mecanismo de reconexão de caminhos (GRASP+RC) e uma última que além de reconexão de caminhos, aplica busca local iterada (GRASP+RC+ILS).

Os resultados comparativos com o algoritmo da literatura indicam que a aplicação de memória adaptativa, com a reconexão de caminhos, faz com que o GRASP convirga mais rápido para boas soluções, especialmente no caso de soluções alvo difíceis.

Como trabalhos futuros, propõe-se a implementação de um GRASP com memória adaptativa, que inicialmente treine várias heurísticas construtivas e buscas locais e após o

treinamento utilize somente as combinações que geraram soluções melhores nas iterações remanescentes. Dessa forma, procura-se encontrar para cada instância, a melhor combinação de métodos construtivo + busca local.

Propõe-se também a análise da utilização de memória em outros elementos do GRASP, como o parâmetro α, e parâmetros específicos dos algoritmos construtivos.

Referências

[1] Dror, M., Haouari, M. e Chaouachi, J. S. (2000), Generalized Spanning Trees, *European Journal of Operational Research*, 120, 583-592.

[2] Feremans, C., Labb\'e, M. e Laporte, G. (1999), The Generalized Minimum Spanning Tree Problem: Polyhedral Analysis ans Branch-and-Cut Algorithm, *Electronic Notes in Discrete Mathematics*, 3, 45-50.

[3] Feremans, C., Labb\'e, M. e Laporte, G. (2001), On Generalized Minimum Spanning Trees, *European Journal of Operational Research*, 134, 457-458.

[4] Feremans, C., Generalized Spanning Trees and Extensions, *Tese de doutorado*, Université Libre de Bruxelles, 2001.

[5] Feremans, C., Lodi, A., Toth, P. e Tramotani, A. (2005), Improving on Branch-and-Cut Algorithms for Generalized Minimum Spanning Trees, *Pacific Journal of Optimization*, 1, 491-508.

[6] Feo, T.A. e Resende, M.G.C. (1995), Greedy randomized adaptive search procedures, *Journal. of Global Optimization*, 6, 109—133

[7] Festa, P. e Resende M. G. C. (2004), An annotated bibliography of GRASP, *European Journal of Operational Research*

[8] Fischetti, M., Salazar, J. J. e Toth, P. (1995), The Symmetric Generalized Traveling Salesman Polytope, *Networks*, 26, 113-123.

[9] Golden, B., Raghavan, S. e Stanojevic D. (2005), Heuristic Search for the Generalized Minimum Spanning Tree, *INFORMS Journal on Computing*, 17, 290-304.

[10] Gonçalves, L. B., **Martins**, **S. L**. **e Ochi**, **L. S.** (2005), A GRASP with Adaptive Memory for a Period Vehicle Routing Problem. *Proc. of the IEEE International Conference on Computational Intelligence for Modelling Control and Automation* – CIMCA2005, Vol. 1, 721-727. Vienna, Austria.

[11] Haouari, M. e Chaouachi, J. S. (2006), Upper and Lower Bounding Strategies for the Generalized Minimum Spanning Tree Problem, *European Journal of Operational Research*, 171, 632-647.

[12] Kansal, A. S. e Torquato, S. (2001), Globally and Locally Minimal Weigth Spanning Tree Networks, *Physica A*, 301, 601-619.

[14] Myung, Y. S., Lee, C. H. e Tcha, D. W. (1995), On the Generalized Minimum Spanning Tree Problem, *Networks*, 26, 231-241.

[15] Silva, **G. C**., **Andrade**, **M**., **Ochi**, **L. S. e Martins**, **S. L.,** and **Plastino**, **A**. (2006), New heuristics for the Maximum Diversity Problem. To appear in *Journal of Heuristics* – SPRINGER.

[16] Shyu, S. J., Yin, P.Y., Lin, B. M. T. e Haouari, M. (2006), Upper and lower bounding strategies for the generalized minimum spanning tree problem, . European *Journal of Operational Research*,171, 632-647.

[17] Besten, M. L. den; Stützle, T.; Dorigo, M. (2001) Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In: BOERS, E. J. W. et al. (Ed.). *Applications of Evolutionary Computing:Proceedings of EvoWorkshops* 2001. Berlin, Germany: Springer Verlag. (Lecture Notes in Computer Science, v. 2037), p. 441-§452.