GRASP COM MEMÓRIA ADAPTATIVA PARA O PROBLEMA DA ÁRVORE DE COBERTURA MÍNIMA GENERALIZADO

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

O Problema da Árvore de Cobertura Mínima Generalizado consiste em dado um grafo *G* cujos vértices estão divididos em grupos, encontrar uma árvore *T* que cubra pelo menos um (ou exatamente um) vértice de cada grupo de *G*, de forma que a soma do custo das arestas de *T* seja mínima. Aplicações deste problema podem surgir em redes de telecomunicações, redes de distribuição de energia elétrica, e em sistemas de irrigação agrícola. Este trabalho apresenta versões da heurística GRASP com Memória Adaptativa para este problema utilizando diferentes métodos de construção e busca local. Além das instâncias encontradas na literatura, consideremos outras de maior dimensão para avaliar os algoritmos propostos. Resultados experimentais mostram a eficiência dos métodos propostos.

PALAVRAS-CHAVE. Metaheurísticas, Otimização em Grafos, Adaptive Memory Programming.

ABSTRACT

The Generalized Minimum Spanning Tree Problem consists of given a graph *G* whose vertices are divided into clusters, finding a tree *T* spanning at least (or exactly) one vertex from each cluster 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 versions of GRASP heuristics with Adaptive Memory for this problem using different construction and local search procedures. In addition to instances found in literature, we have considered new larger instances to test the proposed algorithms. Experimental results illustrate the effectiveness of the proposed methods.

KEY-WORDS. Meta-heuristics, Graph Optimization, Programação com Memória Adaptativa.

1.Introdução

O Problema da Árvore de Cobertura Mínima Generalizado (PACMG) é uma generalização do clássico Problema da Arvore de Cobertura Mínima (ACM) e pode ser definido sobre um grafo ponderado $G = G(V, E)$, onde V representa o conjunto de vértices e $E = \{(i, j): i, j \in V, i \neq j\}$ o conjunto de arestas.

Neste problema, o conjunto V de vértices do grafo é particionado em k grupos V_i, $i \in K = \{ g_1, g_2, ..., g_k \}$. O PACMG tem como objetivo determinar uma árvore de custo mínimo passando por *pelo menos* um vértice de cada grupo. Um caso particular, também abordado na literatura [3,4,5,6,10,12], considera que a árvore deve passar por *exatamente* um vértice de cada grupo. Nesse último caso, o problema é denominado *Equality Generalized Minimum Spanning Tree Problem* e aqui denotado por EPACMG. Ao contrário do clássico problema ACM, o PACMG e o EPACMG são classificados como NP-Completos [3], limitando com isso o uso exclusivo de métodos exatos. A Figura 1 ilustra um exemplo de uma solução para o PACMG e EPACMG em um grafo cujos vértices estão particionados em 4 grupos.

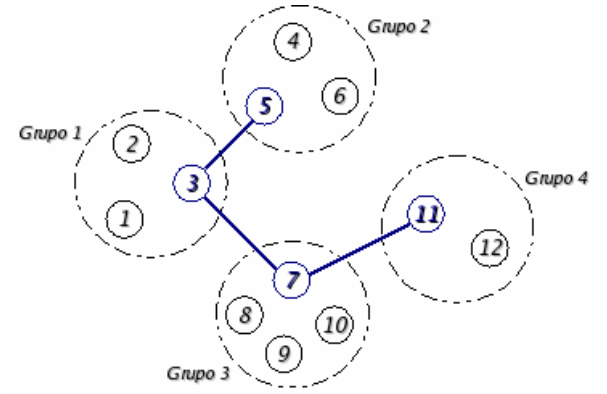

Figura 1- Exemplo de uma solução para o Problema da Árvore de Cobertura Mínima

Aplicações para o PACMG podem ser encontradas, por exemplo, na área de telecomunicações, onde as redes regionais precisam ser inter conectadas por uma árvore que contenha uma conexão para cada sub-rede. Para essa interconexão, um vértice deve ser selecionado como *hub* em cada rede local, e os vértices *hub* têm que ser conectados [4]. Podemos incuir também outras aplicações para o PACMG, como na área de irrigação de terras [2], e na física [13].

O PACMG foi proposto em [15] e apesar das suas inúmeras aplicações possíveis, ainda não tem sido muito explorado pela literatura afim. Dos trabalhos existentes, a maioria das contribuições têm sido para propostas de métodos exatos, tendo como conseqüência a disponibilidade somente de instâncias de pequeno porte na literatura.

Em [15], Myung demonstra que o PACMG é NP-difícil e propõe duas formulações de programação inteira para o problema, uma para o caso de grafos direcionados, e outra para grafos não-direcionados. O autor também propõe um algoritmo *branch-and-bound*, que testa em instâncias geradas aleatoriamente.

As primeiras heurísticas para o PACMG são propostas em [2], que apresenta quatro heurísticas, um algoritmo genético, um método para gerar limites inferiores baseado em relaxação lagrangeana além de duas formulações de programação matemática. Outra formulação matemática é proposta em [4], e mais tarde os mesmos autores fazem uma comparação entre oito formulações de programação inteira para o EPACMG [5].

Um dos trabalhos mais citados sobre o problema é [4], que propõe um *branch-and-cut* para as duas versões e consegue encontrar a solução ótima para praticamente todas as instâncias testadas.

Existem ainda outros trabalhos, como [17], que apresenta uma heurística baseada em colônia de formigas; e [15], que propõe uma heurística que incorpora técnicas de diversificação baseadas em probabilidades.

Este trabalho está organizado da seguinte forma: A próxima seção descreve os algoritmos implementados; na seção 3 são descritas as versões do GRASP aqui propostas; os resultados computacionais são apresentados na seção 4; e por fim, a seção 5 conclui o trabalho.

2.Algoritmos propostos e utilizados

As metaheurísticas têm se mostrado uma das alternativas mais promissoras para a solução aproximada de problemas de elevado nível de complexidade computacional (NP-Completo e NP-Difícil). Dentre as meta-heurísticas existentes, o *Greedy Randomized Adaptive Search Procedure* (GRASP), proposto por [7], tem se destacado como uma das mais competitivas em termos da qualidade das soluções alcançadas [8]. Entretanto tanto o GRASP como as demais metaheurísticas existentes apresentam usualmente como gargalo a necessidade de especializá-los para cada problema no sentido de manter a sua eficiência.

Neste trabalho apresentamos um estudo experimental da heurística GRASP, abordando diferentes versões com memória adaptativa desta técnica. A inclusão de memória entre as iterações do GRASP tem se mostrado promissora na solução de diferentes problemas aplicativos [1, 11, 16].

Foram implementadas quatro heurísticas de construção, algumas já existentes na literatura e outras propostas neste trabalho. Como as instâncias podem ter características diferentes, não é garantido que as heurísticas que funcionam bem para uma determinada instância o façam para todas as outras. Assim, procurou-se implementar algoritmos de construção que tenham, de certa forma, características diferentes entre si. A seguir são descritos os algoritmos implementados, considerando-se um grafo G com um conjunto E de arestas e um conjunto V de vértices, este último dividido em k grupos.

Dentre as heurísticas implementadas, há adaptações dos clássicos algoritmos de Kruskal e Prim para o Problema da Árvore de Cobertura Mínima. Essas adaptações foram primeiramente propostas em [4]. Em virtude de resultados pouco satisfatórios em testes preliminares, não foi considerada a adaptação de Prim nos testes finais.

Construtivo C1 - No algoritmo original de Kruskal, as arestas são ordenadas pelos seus pesos de forma não-decrescente. A partir daí, uma árvore geradora mínima é construída inserindo-se cada aresta $e \in E$ seguindo a ordem definida, se a inserção de e não resultar na formação de um ciclo. O processo termina quando todos os vértices estiverem presentes na árvore.

Para a versão adaptada, a entrada são apenas as arestas que unem vértices de grupos diferentes (*no presente trabalho essa observação não é necessária porque nas instâncias utilizadas não há arestas entre vértices do mesmo grupo*). As arestas também são inseridas por ordem de seus pesos, mas como o objetivo é cobrir exatamente um vértice de cada grupo, a inclusão da aresta e- que une os grupos g_1 e g_2 - determina a seleção dos vértices de g_1 e g_2 que estarão presentes na árvore. Logo, além de evitar arestas que resultem na formação de ciclos, o algoritmo também descarta arestas que implicariam na seleção de um segundo vértice para algum dos grupos.

A Figura 2 ilustra o pseudo-código do algoritmo. T representa a árvore que está sendo construída, c o número de componentes conexas a cada iteração e g_u o grupo ao qual o vértice u pertence. O array γ armazena o vértice que será utilizado por cada grupo na solução.

```
1: ordenar(E);
2: T ← ∅;
3: c ← k;
4: para todo g ∈ G faça
5:    γ[g] ← 0;
6: fim para
7: enquanto c > 1 faça
8:    e(u, v) ← proximaAresta(E);
9:    se !geraCiclos(e) então
10:      se (γ[g_u]= u ou γ[g_u]= 0) e (γ[g_v]= v ou γ[g_v]= 0) então
11:         T ← T ∪ e;
12:         c ← c − 1;
13:         se γ[g_u] = 0 então
14:            γ[g_u] ← u;
15:         fim se
16:         se γ[g_v] = 0 então
17:            γ[g_v] ← v;
18:         fim se
19:      fim se
20:   fim se
21: fim enquanto
```

Figura 2- Pseudo-código da adaptação do algoritmo de Kruskal

Como o algoritmo de Kruskal e sua versão para o PACMG são gulosos, considerou-se também uma versão com componentes aleatórios na seleção das arestas que serão incluídas na árvore. Nessa versão a seleção é feita a partir de um sub-conjunto de *E*, chamado Lista de Candidatos Restrita, ou LCR - composto de todas as arestas cujo peso é menor ou igual a $c_{min} + (c_{max} - c_{min})\alpha$, onde c_{min} e c_{max} são o menor e o maior peso dentre as arestas de *E*, respectivamente, e α varia de 0 a 1. Seleciona-se então um das arestas da LCR para compor a árvore de forma aleatória. Pode-se perceber que o valor de α indica então o quão gulosa será a escolha dos vértices: caso α seja 0, por exemplo, trata-se exatamente da adaptação de Kruskal descrita anteriormente.

Construtivo C2 - Em [10] foi proposta uma heurística construtiva relativamente simples, que seleciona de forma aleatória um vértice de cada grupo, e a partir daí utiliza algum dos algoritmos clássicos para calcular uma árvore de cobertura mínima entre os *K* vértices.

Foi implementada uma versão modificada da heurística descrita acima, em que os vértices não são mais escolhidos de forma aleatória, e sim de acordo com um determinado critério. Esse critério é a distância média de cada vértice v_i, pertencente ao grupo *i*, a cada vértice v_j, tal que $i \neq j$. Para cada grupo, seleciona-se o vértice cuja distância média é mínima. O pseudo-código está ilustrado na Figura 3.Nele, a função *vertices(g)* retorna o conjunto de todos os vértices do grupo *g*, e a função *custo(u,v)* retorna o custo da aresta que une os vértices *u* e *v*. Ao final do algoritmo, aplica-se o algoritmo de Prim (indicado na função *ACM(γ)*) sobre os vértices presentes em γ.

É também proposta uma versão que seleciona os vértices aleatoriamente a partir de uma LCR montada para cada grupo e que inclui todos os vértices cuja distância média – calculada conforme explicado anteriormente – é menor ou igual a $d_{min} + (d_{max} - d_{min})\alpha$, onde d_{min} e d_{max} são, respectivamente, a maior e a menor distância média dos vértices daquele grupo.

```
1: para todo g ∈ G faça
2:   d_min ← ∞;
3:   para todo v ∈ vertices(g) faça
4:     d ← 0;
5:     para todo v_i ∈ V− vertices(g) faça
6:       d ← d+ custo(v, v_i);
7:     fim para
8:     se d < d_min então
9:       d_min ← d;
10:      s ← v_i;
11:    fim se
12:  fim para
13:  γ[g] ← s;
14: fim para
15: T ← ACM(γ);
```

Figura 3- Pseudo-código da heurística construtiva C2

Construtivo C3 - Uma heurística aqui proposta para o problema divide os grupos em *c* grupos maiores e aplica a adaptação de Kruskal sobre cada um desses grupos, resultando em uma floresta com *c* árvores. Com isso, há uma grande chance de que as árvores geradas estejam próximo do ótimo.

A Figura 4 mostra um exemplo do funcionamento da heurística para $c = 3$. A união das árvores é feita com base na idéia de "grupos vizinhos", em que são testadas as possibilidades entre todos os grupos próximos entre si. No exemplo, os grupos maiores à esquerda e central são "vizinhos", assim como os grupos maiores central e à direita. Essa heurística foi proposta especialmente para o caso de instâncias de grande porte.

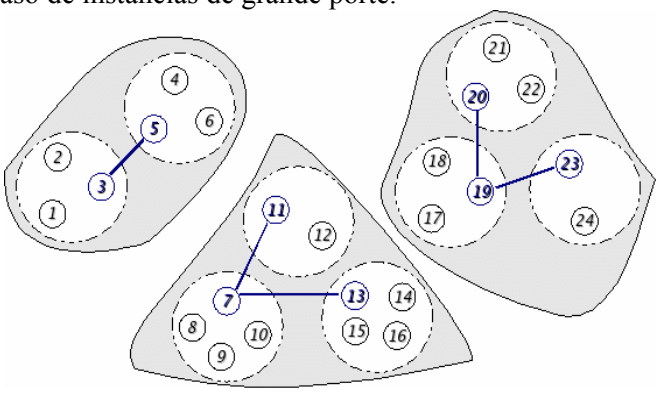

Figura 4 – Heurística de Construção C3: Divisão em três grandes grupos e construção de árvores de cobertura entre os grupos maiores

Construtivo C4 - Por fim, foi implementada uma heurística construtiva que utiliza um mecanismo bem diferente das demais na construção de uma solução, proposta neste trabalho. O algoritmo parte de uma permutação dos grupos, conforme ilustrado na Figura 5, e constrói um caminho mínimo de v_1 a v_2 (*vértices fictícios origem e destino do caminho mínimo*). Uma observação é que são consideradas apenas as arestas entre grupos adjacentes na permutação, o que garante que esse caminho passa por exatamente um vértice de cada grupo. Logo, excluindo-se v_1 e v_2, tem-se uma solução viável para o EPACMG.

Aplica-se ainda uma busca local sobre essa solução, trocando-se as posições dos grupos (*dois a dois, e depois três a três*), na permutação gerada inicialmente, e recalculando-se o caminho mínimo. É importante salientar que a permutação dos grupos é gerada de forma que os grupos mais próximos tenham uma probabilidade maior de ficarem adjacentes, e que exista ainda um certo grau de aleatoriedade, para que várias execuções possam gerar permutações distintas.

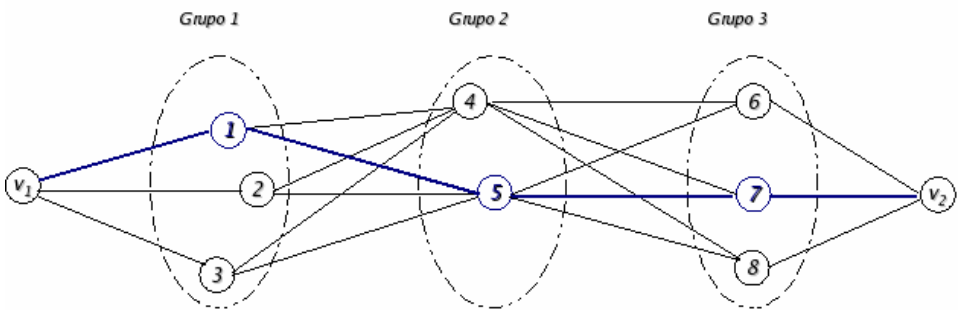

Figura 5 – Exemplo de uma solução gerada com a heurística construtiva C4

Sabe-se que a solução gerada por essa heurística tende a ser inferior às soluções de outras, pelo fato de ser limitada a um caminho. Mas a idéia é que a aplicação de um mecanismo de busca local que refine esse caminho (tornando-o outro tipo de árvore, se for o caso) resulte em boas soluções dependendo das características das instâncias.

Busca Local BL - A busca local implementada (proposta em [10]) tem um funcionamento simples, e pode ser descrita nas três etapas abaixo:

1. Aleatoriamente define-se uma ordem em que os grupos serão visitados (uma simples permutação dos grupos)
2. A cada visita a um grupo, considera-se cada um de seus vértices *v* e calcula-se uma árvore de cobertura mínima considerando *v* e os vértices definidos nos outros grupos. O vértice que proporcionar a árvore de menor custo passa então a fazer parte da solução.
3. Repete-se o passo 2 até que *k* grupos tenham sido visitados sem que haja melhora na solução corrente.

A árvore de cobertura mínima calculada a cada movimento foi implementada com o algoritmo clássico de Prim.

Reconexão de Caminhos RC - Além destes procedimentos foi também proposto um módulo de busca intensiva usando o conceito de Reconexão de Caminhos (*Path Relinking*).

O mecanismo de Reconexão de Caminhos (RC) tem como objetivo encontrar soluções intermediárias entre duas boas soluções. O algoritmo parte de uma determinada solução (base), e passo-a-passo a transforma em outra (alvo). Nesse trajeto, entende-se que pode ser encontrada uma solução melhor que as duas soluções extremas.

Dadas duas soluções para o EPACMG e tomando-se uma como base *b* e uma como alvo *a*, pode-se considerar um movimento de *b* para *a,* a troca de um vértice utilizado por *b* por um vértice utilizado por *a*. Assim, no mecanismo de RC adotado neste trabalho, a cada movimento verifica-se quais são os grupos em que *a* e *b* utilizam vértices diferentes. Para cada grupo *g* em que os vértices das duas soluções são diferentes, constrói-se uma solução intermediária entre *b* e *a*, substituindo-se na solução *b* o vértice do grupo *g* por aquele utilizado em *a*. A nova solução base será a de menor custo entre as soluções intermediárias geradas.

É importante salientar que a cada movimento a árvore de cobertura mínima é refeita através do algoritmo de Prim, como acontece na Busca Local. E toda vez que o conjunto elite é atualizado com a inserção de uma solução intermediária *s*, aplica-se também uma busca local sobre *s*.

A Figura 6 ilustra um exemplo do funcionamento do RC em que a solução alvo é o grafo ilustrado na Figura 1. A solução base, nesse caso, possui como vértices 3, 6, 8 e 12, dos quais 6, 8 e 12 não constam na solução alvo. Há, portanto, três possibilidades de movimento, das quais é escolhida aquela que troca o vértice 6 pelo 5. Após esse passo, a solução intermediária difere do alvo por dois vértices, 12 e 8. Trocando o vértice 12 pelo 11, a solução fica a apenas um movimento do alvo (troca de 8 por 7).

Esse mecanismo é ativado a partir de uma determinada iteração *i*, e depois disso, toda vez que a busca local encontra uma boa solução - para que uma solução seja considerada boa, seu custo não pode ser mais que *p*% maior que o custo da melhor solução encontrada até o momento.

Para a aplicação do RC, é mantido um conjunto elite com as quatro melhores soluções encontradas até o momento pelo algoritmo. O RC sempre é feito entre a solução *s* encontrada pela busca local e a solução do conjunto elite "mais diferente" de *s*, no sentido da melhor para a pior solução. Entende-se aqui por solução "mais diferente" de uma solução *s*, aquela cuja diferença de custo para *s* seja maior.

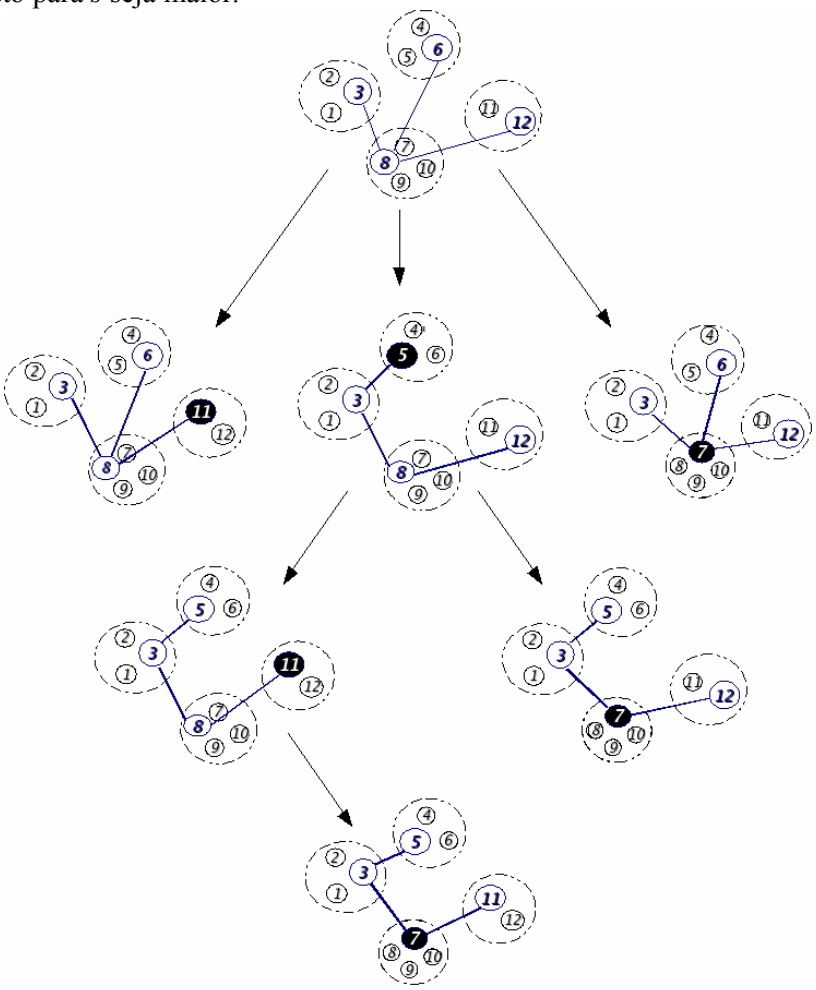

Figura 6- Exemplo de uma execução do mecanismo de Reconexão de Caminhos

3. GRASP com Memória Adaptativa para o EPACM

Neste trabalho partimos do pressuposto de que versões GRASP com memória adaptativa, como proposto neste trabalho, tendem a produzir soluções médias de melhor qualidade do que versões totalmente estáticas (usando único valor de α, um único método construtivo, e sem o módulo de Reconexão de Caminhos). Tal decisão é baseada em trabalhos anteriores onde verificamos que versões adaptativas são nitidamente superiores do que versões tradicionais do GRASP [1, 11, 16].

A idéia básica do GRASP com memória adaptativa proposto é, durante uma execução, inicialmente treinar várias heurísticas construtivas e após o treinamento utilizar somente as combinações que geraram soluções melhores nas iterações remanescentes do GRASP. Dessa forma, procura-se encontrar para cada instância, a melhor combinação de métodos construtivo + busca local.

O segundo tipo de memória utilizado pelas duas versões se refere a atualização do parâmetro α da fase de construção do GRASP. Ou seja, a calibração deste parâmetro também é feita através de um treinamento entre valores candidatos. A versão com Reconexão de Caminhos utiliza adicionalmente um terceiro tipo de memória, através da criação e atualização de um conjunto elite. Espera-se com isso, obter heurísticas GRASP mais robustas, ou seja, com um desempenho mais regular para diferentes tipos de instâncias do problema analisado.

O GRASP implementado possui basicamente três etapas: treinamento, intensificação e diversificação. Na primeira etapa cada uma das heurísticas de construção é executada um determinado número *n* de vezes, armazena-se o custo médio das soluções geradas por cada iteração.

A partir dessa informação, na fase seguinte são executadas apenas as duas heurísticas que geraram melhores soluções, que são intercaladas de forma que a melhor heurística execute por mais tempo. Nessa fase, a heurística que atingiu melhores resultados na fase de treinamento executa $2n$ vezes consecutivas e em seguida, a segunda melhor heurística executa *n* vezes. O processo se repete até que seja atingido um tempo *t* sem melhora na melhor solução

Na última fase, de diversificação, o valor do parâmetro α é incrementado, de forma a levar a busca para espaços ainda não-percorridos. Sempre que o valor de α for atualizado um novo treinamento de algoritmos construtivos é efetuado.

Neste trabalho foram implementadas duas versões da meta-heurística GRASP. As duas com as mesmas fases de treinamento, intensificação e diversificação, utilizando as heurísticas construtivas descritas na seção 2. A busca local das duas é a mesma, diferindo apenas pela utilização ou não do mecanismo de Reconexão de Caminhos. A Tabela 1 descreve as diferenças entre os dois algoritmos.

Tabela 1- Configuração das heurísticas GRASP propostas

Versão GRASP	***Construtivo(s)***	***Busca local***	***Reconexão Caminhos***
GRASP Puro	C1, C2, C3 e C4	BL	-
GRASP+RC	C1, C2, C3 e C4	BL	RC

4.Resultados computacionais

Os testes computacionais foram realizados em um computador Pentium IV, com 2.6GHz e 512MB de memória principal, rodando um sistema operacional Linux versão 2.6.8-1.521. Os programas implementados na linguagem de programação c++ e compilados com g++ versão 4.0.2 utilizando as opções de compilação –o3 e march=pentium4.

As instâncias utilizadas sempre utilizam norma euclideana para o cálculo das distâncias entre dois vértices, e em sua maioria, foram extraídas do repositório de instâncias do Problema do Caixeiro Viajante e adaptadas por Fischetti em [9] para o problema do Caixeiro Viajante Generalizado.

Na criação das instâncias da literatura, Fischetti utilizou técnicas para o agrupamento dos vértices, chamadas *Center Clustering* e *Grid Clustering*. Na primeira técnica, o número de grupos é definido como $\lceil |V|/5 \rceil$, já a segunda recebe como parâmetro um valor (de forma que o número de grupos da instância seja sempre maior que $|V|/\mu$), onde V representa o conjunto de vértices e

μ=3, 5, 7 e 10. Existem ainda cinco instâncias cujo agrupamento foi feito de maneira a respeitar a disposição geográfica dos vértices. Os testes deste nosso trabalho foram realizados apenas com esse último grupo de instâncias e com aquelas criadas por *Center Clustering*, uma vez que essas foram as instâncias utilizadas por Feremans [3, 4, 5, 6] em seus testes.

As características de cada instância podem ser identificadas pelos seus nomes, que obedecem o seguinte formato: *gNomev*, sabendo que k é o número de grupos de uma instância e v é seu número de vértices.

Nos nossos testes, foi utilizado como critério de parada o encontro de uma solução cujo valor é melhor ou igual a um valor alvo e/ou um tempo limite de execução. Após testes preliminares, foram definidos como parâmetros n=5, t=75% do tempo limite, i=20, p=2 e c=4.

Na Tabela 2 são comparados os resultados das duas versões do GRASP adaptativo para instâncias com até 442 vértices, não testadas em [4] (*as instâncias nos foram enviadas mas os autores destas ainda não testaram em seus algoritmos*) e cujo ótimo ainda não é conhecido. O tempo limite usado como critério de parada foi 3600s de execução. Cada algoritmo foi executado três vezes para cada instância. A primeira coluna apresenta o nome de cada instância; a segunda coluna exibe o número de arestas; em seguida está o os resultados médios e os tempos médios de execução do GP e do G+RC. Na Tabela 2, valores em negrito representam a melhor solução obtida.

Tabela 2- Comparação entre as quatro versões propostas para o GRASP

Instância	***|E|***	***GRASP Puro (GP)***		***GRASP + RC (G+RC)***	
		Custo	***Tempo(s)***	***Custo***	***Tempo(s)***
40d198	18841	7047,6	3601,8	**7044,0**	39,9
41gr202	19532	243,0	3214,0	**242,0**	169,9
45ts225	24650	**62268,0**	1271,4	**62268,0**	944,9
46pr226	24626	**55515,0**	4,0	**55515,0**	6,7
53gil262	33507	944,0	3601,7	**942,0**	3603,7
53pr264	34028	21895,0	3604,0	**21886,0**	3602,4
60pr299	43786	20344,0	3602,6	**20316,0**	3604,7
64lin318	49363	5962.67	3611,0	**5937,6**	3612.9
80rd400	78779	**18508,3**	3604,2	18513,0	3602,2
84fl417	84841	7984,3	3605,4	**7982,0**	3602,3
88pr439	94585	51925,3	3611,7	**51841,7**	3614,9
89pcb442	96360	19711,0	3610,7	**19645,3**	3628,2

Percebe-se pelos resultados médios da Tabela 2 que a versão com Reconexão de Caminhos encontra resultados melhores para a maior parte das instâncias, apesar de apresentar um tempo médio um pouco maior para algumas instâncias. Apenas em um caso a versão com busca local simples (GP) consegue resultado médio melhor. Alguns tempos são maiores que 3600s porque os algoritmos demoram mais de 1s para concluir a iteração.

A Tabela 3 compara os resultados encontrados pelos GRASP's aqui propostos com os valores ótimos encontrados por Feremans, com seu algoritmo *branch-and-cut* [4]. Na primeira coluna está o nome de cada instância; na segunda o número de arestas $|E|$; em seguida o valor ótimo; na quarta coluna apresenta-se o tempo gasto por Feremans, em segundos; nas colunas seguintes estão o custo médio e o tempo médio do GRASP Puro; nas duas últimas colunas estão o custo médio e tempo médio da versão do GRASP com Reconexão de Caminhos. Os resultados apresentados são a média de cinco execuções. Foi determinado como alvo o valor ótimo de cada instância, e o tempo limite 500 segundos de execução.

Tabela 3- Comparação dos resultados obtidos por Feremans e pela melhor versão do GRASP

Inst	***|E|***	***Ótimo***	***Tempo(s) Feremans***	***GRASP Puro***		***GRASP + RC***	
				Custo	***Tempo(s)***	***Custo***	***Tempo(s)***
15spain47	985	2393	5	**2393**	**0,01**	**2393**	**0,01**
27europ47	1042	13085	3	**13085**	**0,01**	**13085**	**0,01**
50gr96	4463	306	68	**306**	**0,08**	**306**	0,09

Inst	*|E|*	*Ótimo*	*Tempo(s) Feremans*	*GRASP Puro*		*GRASP + RC*	
				Custo	*Tempo(s)*	*Custo*	*Tempo(s)*
35gr137	8251	209	110	**209**	**0,54**	**209**	0,56
34gr202	19018	135	4558	**135**	136,66	**135**	**64,57**
10att48	1010	10923	4	**10923**	**0,05**	**10923**	0,07
10gr48	1017	1282	3	**1282**	**0,02**	**1282**	**0,02**
10hk48	995	4119	4	**4119**	**0,05**	**4119**	**0,05**
11eil51	1158	132	5	**132**	0,05	**132**	**0,04**
12brazil58	1464	9206	14	**9206**	**0,04**	**9206**	**0,04**
14st70	2248	233	20	**233**	**0,23**	**233**	0,25
16eil76	2660	186	47	**186**	1,95	**186**	**1,51**
16pr76	2661	46514	37	**46514**	**0,18**	**46514**	0,26
20gr96	4292	221	99	**221**	**0,77**	**221**	1,05
20rat99	4609	402	83	**402**	1,94	**402**	**1,52**
20kroa100	4727	7982	65	**7982**	**3,88**	**7982**	4,17
20krob100	4716	8111	73	**8111**	4,91	**8111**	**1,62**
20kroc100	4714	8041	87	**8041**	**3,31**	**8041**	4,97
20krod100	4716	7643	183	**7643**	**1,62**	**7643**	2,50
20kroe100	4702	8164	66	**8164**	**0,52**	**8164**	0,61
20rd100	4703	2779	55	**2779**	**0,77**	**2779**	1,57
21eil101	4776	204	76	**204**	**0,84**	**204**	1,49
21lin105	5130	6728	109	**6728**	**0,27**	**6728**	0,28
22pr107	5441	20398	244	**20398**	**182,09**	**20398**	193,75
24gr120	6820	2255	114	**2255**	5,41	**2255**	**1,66**
25pr124	7315	30174	753	**30174**	15,08	**30174**	**7,78**
26bier127	7191	58150	908	**58150**	**10,76**	**58150**	12,83
28pr136	8879	34104	406	**34104**	148,04	**34104**	**90,24**
28gr137	8892	329	1518	**329**	67,41	**329**	**12,09**
29pr144	9952	40055	861	**40055**	12,04	**40055**	**11,26**
30kroa150	10809	9815	426	**9815**	35,45	**9815**	**32,79**
30krob150	10807	10048	849	**10048**	**23,70**	**10048**	28,56
31pr152	11080	39109	1541	**39109**	**10,54**	**39109**	12,78
32u159	12031	18723	592	**18723**	79,98	**18723**	**35,36**
39rat195	18478	751	2120	**751**	258,34	**751**	**104,74**
40kroa200	19409	11634	2607	**11634**	201,66	**11634**	**162,84**
40krob200	19430	11244	5254	**11244**	297,27	**11244**	**171,25**

Com os resultados da Tabela 3, percebe-se que ambas as versões do GRASP proposto (GP e G+RC) sempre conseguem atingir o valor ótimo das instâncias testadas por Feremans, com um tempo, em média, de apenas 5,7% do tempo do método exato para o GP e 4,7% para o G+RC.

Percebe-se também que a versão com Reconexão de Caminhos obteve melhores resultados em termos de tempo gasto nas instâncias que Feremans considera "difíceis" (onde o exato leva mais tempo). Acredita-se que isso se deve ao fato de que, nas instâncias mais fáceis não ser necessário um mecanismo adicional à busca local simples para que os algoritmos encontrem o

alvo. Entretanto, as diferenças entre os resultados das duas versões não se deve necessariamente à Reconexão de Caminhos, pois para a maioria das instâncias esse mecanismo pode nem ter sido ativado, uma vez que os algoritmos executaram, em média, 10,3 iterações, e a Reconexão de Caminhos passar a ser aplicada a partir da iteração 20. Em termos gerais ao menos nas simulações efetuadas neste trabalho, se mostrou que ambas as heurísticas aqui propostas sempre obtêm solução ótimas nas instâncias testadas por Feremans, mostrando com isso o potencial de versões GRASP com Memória Adaptativa.

5.Conclusões

O presente trabalho aborda o Problema da Árvore de Cobertura Mínima Generalizado, comparando várias heurísticas de construção e propondo duas versões adaptativas da heurística GRASP utilizando no caso do G+RC, diferentes tipos de memória. A meta principal ao propor estas heurísticas, foi principalmente obter algoritmos iterativos que sejam confiáveis (regulares) na solução de diferentes tipos de instâncias do problema analisado. Pelos resultados obtidos, acreditamos que pelo menos de forma empírica este objetivo foi alcançado.

Os resultados demonstram que as heurísticas GRASP' s propostas atingem os valores ótimos em todas as suas execuções para todas as instâncias de até 226 vértices testadas por Feremens [4]. Quando comparado com o algoritmo de *branch-and-cut* proposto em [4], estas se mostram mais eficientes, atingindo os mesmos resultados do exato mas em tempos significativamente menores.

Foi feita também uma comparação entre as duas versões do GRASP, a fim de verificar a influência do mecanismo de Reconexão de Caminhos sobre o desempenho do algoritmo. Os testes demonstraram que a versão que utiliza tal mecanismo apresenta resultados melhores para instâncias de grande porte, e que há um certo equilíbrio entre os resultados das duas versões para instâncias menores.

Trabalhos futuros poderiam realizados no sentido de melhorar a fase diversificação, de forma que a heurística possa escapar de ótimos locais ainda distantes de um ótimo global de maneira mais eficiente.

6.Agradecimentos

Os autores agradecem a CAPES e ao CNPq, pelo financiamento parcial deste trabalho.

7.Referências

[1] Bastos, L. O., Ochi, L. S. e Macambira, E. M. (2005), GRASP with Path Relinking for the SONET Ring Assignment Problem. *Proc. of the 5th International Conference on Hybrid Intelligence System (HIS2005)* in cooperation with IEEE Computational Intelligence Society, 239-244.

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

[3] 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.

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

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

[6] 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.

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

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

[9] Fischetti, M., Salazar, J. J. e Toth, P. (1995), The Symmetric Generalized Traveling Salesman Polytope, *Networks*, 26, 113-123.
[10] Golden, B., Raghavan, S. e Stanojevic D. (2005), Heuristic Search for the Generalized Minimum Spanning Tree, *INFORMS Journal on Computing*, 17, 290-304.
[11] 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.
[12] 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.
[13] Kansal, A. S. e Torquato, S. (2001), Globally and Locally Minimal Weigth Spanning Tree Networks, *Physica A*, 301, 601-619.
[14] Kruskal, J. B. (1956), On the Shortest Spanning Tree of Graph and the Salesman Problem, *Proceedings of the American Mathematical Society*, 7: 48-50.
[15] Myung, Y. S., Lee, C. H. e Tcha, D. W. (1995), On the Generalized Minimum Spanning Tree Problem, *Networks*, 26, 231-241.
[16] 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.
[17] 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.