Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
Teoria dos Grafos

Métodos Heurísticos Usando GRASP com Reconexão de Caminhos para o Problema da Árvore Geradora Mínima Generalizado com Coleta de Prêmios

Grupo 32

DÉBORA IZABEL ROCHA DUARTE - MAT 201776029

IVANYLSON HONORIO GONÇALVES – MAT 201776002

Professor: Stênio Sã Rosário F. Soares

Relatório do trabalho final da disciplina DCC059 - Teoria dos Grafos, parte integrante da avaliação da mesma.

Juiz de Fora
Março de 2020

1 Introdução

O presente Relatório Técnico tem como objetivo central descrever o uso de algoritmos construtivos gulosos para o Problema Árvore Geradora Mínima Generalizado em Grafos que, como o nome sugere, é uma generalização do Problema de Árvore Geradora Mínima. O PAGMG pode ser definido sobre um grafo não direcionado G(V,E) onde V representa o conjunto de vértices e E o conjuntos de arestas. Cada aresta possui associado um custo Neste problema, o conjunto de vértices V é particionado em m grupos disjuntos e não vazios e o objetivo é determinar uma árvore de custo mínimo que cubra exatamente um vértice de cada grupo, como introduzido em [1].

Outra versão encontrada na literatura e diretamente relacionado ao PAGMG é o de encontrar uma árvore que cubra pelo menos um vértice de cada grupo. Tais generalizações seguem a ideia do já conhecido Problema do Caixeiro Viajante Generalizado em que as cidades estão divididas por regiões e o caixeiro só precisa passar por uma cidade de cada região [2].

Aplicações para o PAGMG podem ser encontradas nas áreas de síntese de redes de telecomunicação, 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 [4].

Apesar de sua clara importância prática, a maior parte dos principais trabalhos relacinados é bastante recente. Dentre eles, há o de Feremants et al. [8], que apresentam e comparam várias formulações matemáticas, além de proporem um algorítimo *branch-and-cut* [7] que resolveu instâncias com até 200 vértices. Golden et al. [15] também trouxeram contribuições importantes para o problema, dentre elas duas heurísticas que encontram as soluções ótimas de quase todas as instâncias introduzidas por Feremans. Mais recentemente, Oncan et al. [24] apresentaram mais um procedimento para o cálculo de limites duais e um algoritmo de busca tabu que superou os resultados de Golden, além de apresentar instancias novas e de dimensões maiores.

O presente trabalho tem como objetivo propor algumas versões da heurística GRASP para o PAGMG. Foram implementados vários algorítmos construtivos, dentre eles alguns propostos neste trabalho e alguns da literatura. Teste preliminares indicaram que não deve existir um algoritmo com desempenho superior aos demais em todas as intâncias. Tais resultados motivaram a implementação de versões GRASP que utilizam vários algoritmos construtivos de forma adaptativa. O objetivo é fazer com que o algoritmo possa utilizar o melhor construtivo para cada instancia.

Também foram incorporados ao GRASP mecanismos de aprimoramento adicionais como Reconexão de Caminhos de Busca Local Iterada. Foi erificado que tais mecanismos propiciam

melhoras significativas sobre o desempenhos dos algorítimos. As heurísticas aqui propostas foram comparadas com os resultados anteriores obtidos por Oncan et al. [24].

Este trabalho apresenta ainda um algorítmo de geração de cortes para o cálculo de limites duais. Tal algoritmo utiliza uma formulação do Problema de Steiner em grafos direcionados. Também são propostas regras de redução das instâncias, baseadas no conceito de distância *bottleneck*.

O trabalho está organizado como segue: o capítulo 2 traz uma descrição mais detalhada do prblemas e uma revisão dos trabalhos da literatura, o capítulo 3 apresenta os algoritimos propostos, no capítulo 4 encontram-se os resultados computacionais e, por fim, no capítulo 5 são apresentadas as conclusões e algumas sugestões de trabalhos futuros.

2 Descrição do Problema

Neste capítulo será feita uma descrição do Problema da Árvore Geradora Mínima Generalizado (PAGMG) e dos trabalhos sobre ele relatados na literatura. Além disso, também serão traçados alguns comentários sobre problemas que se tornam importantes no estudo do PAGMG, como o Problema da Árvore Geradora Mínima e o Problema de Steiner em Grafos.

O Problema da Árvore Geradora Mínima Generalizado é definido sobre um grafo não direcionado G(V,E) cujos vértices estão particionados em *m* grupos $\{V_1, V_2, \ldots, V_m\}$. Dado que $|V| = n$, tem-se que $V = V_1 \cup V_2 \cup \ldots V_m$ e $V_l \cap V_k = \emptyset \ \forall l, k \in \{1, \ldots, m\}, l \neq k$. Assume-se que as arestas estão definidas apenas entre vértices de grupos diferentes e cada aresta e possui um custo associado $c_e \in R^+$. O objetivo do problema é determinar uma árvore de custo mínimo que cubra exatamente um vértice de cada grupo. Tal problema foi introduzido em 1995 por Myung [23] e é denominado *Equality Generalized Minimum Spanning Tree Problem* (E-GMSTP).

Diferentemente do Problema da Árvore Geradora Mínima, vastamente abordado na literatura e resolvido de maneira eficiente em tempo polinomial [20], o PAGMG é classificado como *NP-difícil* [23] e há um número restrito de trabalhos relacionados de que se tem conhecimento.

Uma outra generalização do problema da Árvore Geradora Mínima, também abordada na literatura, é o problema de encontrar uma árvore que cubra pelo menos um vértice de cada grupo. Muitos autores vêm denominando essa variante de L-GMSTP (L vem de *at least*).

O modelo é o seguinte:

$$\min \sum_{e \in E} x_e c_e - \sum_{v \in V} p_v z_v \tag{1}$$

$$\text{s.t.:} \sum_{v \in V_k} z_v = 1, \quad \forall k = 1, \ldots, r \tag{2}$$

$$\sum_{e \in E} x_e = r - 1 \tag{3}$$

$$\sum_{e=(u,v) | u \in V_i, v \in V_j} x_e = y_{ij}, \quad \forall i = 1, \ldots, r, i \neq j \tag{4}$$

$$\sum_{e=(u,v) | u \in V_i} x_e \leq z_v, \quad \forall i = 1, \ldots, r, \forall v \in V \setminus V_i \tag{5}$$

$$y_{ij} = \lambda_{kij} + \lambda_{kji}, \quad \forall i, j, k = 1, \ldots, r, i \neq j, i \neq k, \tag{6}$$

$$\sum_{j \in \{1,\ldots,r\} \setminus \{i\}} \lambda_{kij} = 1, \quad \forall i, k = 1, \ldots, r, i \neq k \tag{7}$$

$$\lambda_{kkj} = 0, \quad \forall j, k = 1, \ldots, r, j \neq k \tag{8}$$

$$\lambda_{kij} \geq 0, \quad \forall i, j, k = 1, \ldots, r, i \neq j, j \neq k \tag{9}$$

$$x_e, z_v \geq 0, \quad \forall e = (u, v) \in E, \forall v \in V \tag{10}$$

$$y_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, r, i \neq j \tag{11}$$

A função objetivo (1) minimiza o custo total das arestas selecionadas menos a soma dos premiados. As restrições (2) garantem que exatamente apenas um vértice seja selecionado de cada cluster. Restrição (4) garante que a solução contém exatamente r − 1 arestas. As restrições (4) permitem que apenas as bordas dos clusters conectado no gráfico global pode ser selecionado. As restrições (5) garantem que as arestas se conectem apenas selecionadas vértices. As restrições (6) indicam que uma aresta (i, j) está na árvore geradora se e somente se i for o pai de j ou j é o pai de i. As restrições (7) garantem que, se uma árvore está enraizada em k, então cada vértice tem exatamente um pai, exceto para o vértice raiz k. As restrições (8) garantem que o vértice k não tenha pai. Finalmente, as restrições (9) - (11) definem o domínio das variáveis.

3 Abordagens Gulosas para o Problema

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 NPDifícil). 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, 9, 10]. 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.

3.1 Algoritmo Guloso

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 eque une os grupos g1 e g2 - determina a seleção dos vértices de g1 e g2 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 gu o grupo ao qual o vértice u pertence. O array y armazena o vértice que será utilizado por cada grupo na solução.

Algorithm 3: Algoritmo Guloso

```
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
```

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 cmin+ (cmax – cmin)α, onde cmin e cmax são o menor e o maior peso dentre as arestas de E, respectivamente, e α varia de 0 a 1. Seleciona-se então uma 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.

3.2 Algoritmo Guloso Randomizado

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 vi, pertencente ao grupo i, a cada vértice vj, tal que i≠j. Para cada grupo, seleciona-se o vértice cuja distância média é mínima. O pseudo-código está ilustrado na Figura

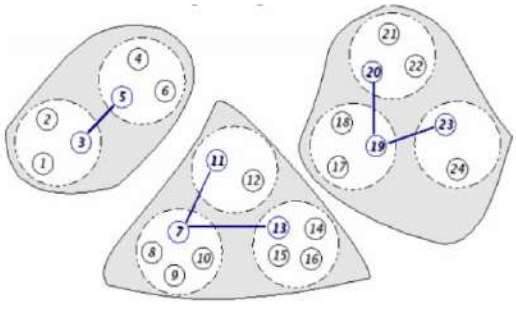
3.Nele, a função vertices(g) retorna o conjunto de todos os vértices do grupo g, e a função custo(u,v) retorna o custo da aresta que une os vértices u e v. Ao final do algoritmo, aplica-se o algoritmo de Prim (indicado na função ACM(γ)) sobre os vértices presentes em γ. É também proposta uma versão que 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 dmin+(dmax – dmin)α, onde dmin e dmax são, respectivamente, a maior e a menor distância média dos vértices daquele grupo.

Algorithm 4: Algoritmo Guloso Randomizado

```
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);
```

3.3 Algoritmo Guloso Randomizado Reativo

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.



Algoritmo GRASP

```
s* ← ∅;
para todo i ∈ {1, ..., It_max} faça
  s_0 ← constroi_solucao(α);
  s ← busca_local(s_0);
  se custo(s) < custo(s*) então
    s* ← s;
  fim se
fim para
retornar s*;
```

Algoritmo - Construção aleatória

```
para todo i ∈ {1, .., m} faça
  γ[i] ← rand(V_i);
fim para
T ← kruskal(γ, E(γ));
retornar T;
```

Algoritmo - Adaptação de Kruskal para o PAGMG

```
ordenar(E);
T ← ∅;
num ← m;
para todo i ∈ {1, ..., m} faça
  γ[i] ← 0;
fim para
enquanto num > 1 faça
  L ← constroi_LCR(E, α);
  e(u, v) ← rand(L);
  se !gera_ciclos(e) e (γ[g_u]= u ou γ[g_u]= 0) e (γ[g_v]= v ou γ[g_v]= 0) então
    T ← T ∪ e;
    num ← num − 1;
    se γ[g_u] = 0 então
      γ[g_u] ← u;
    fim se
    se γ[g_v] = 0 então
      γ[g_v] ← v;
    fim se
  fim se
fim enquanto
retornar T;
```

Algorithm Guloso Randomizado Reativo

```
1: procedimento GRASP REATIVO
2:     f(s*) ← +∞
3:     it ← 1
4:     Definir Ψ = {α_1, α_2, ..., α_m}
5:     para i ← 1 até m faça
6:         p_i ← 1/m; m_i ← 0
7:     fim para
8:     enquanto não CritérioParada faça
9:         α ← Selecione α_i ∈ Ψ com probabilidade p_i
10:        s_1 ← FaseConstrutiva(α)
11:        s_2 ← BuscaLocal(s_1)
12:        se f(s_2) < f(s*) então
13:            s* ← s_2
14:        fim se
15:        se it mod y = 0 então
16:            Recalcular m_i, ∀i ∈ {1,2,...,m}
17:            Recalcular q_i, ∀i ∈ {1,2,...,m} pela equação 2.19
18:            Atualizar as probabilidades p_i, ∀i ∈ {1,2,...,m} pela equação 2.18
19:        fim se
20:        it ← it + 1
21:    fim enquanto
22:    retorne s*
23: fim procedimento
```

4 Experimentos Computacionais

4.1 Descrição das instâncias

A fim de facilitar o entendimento do trabalho, as instâncias foram divididas em dois grupos:

Grupo 1

Composto por 169 instâncias com $48 \leq |V| \leq 226$. Foram geradas por Fischetti et al. [12] para o Problema do Caixeiro Viajante Generalizado e introduzidas no PAGMG por Feremans et al. [8].

Cinco instâncias desse grupo (15spain47, 27europ47, 50gr96africa, 35gr37america e 34gr202europe) representam a disposição geográfica real de cidade e seu agrupamento segue a divisão natural das cidades em regiões. As demais instâncias do Grupo 1 foram adaptadas do repositório do Problema do caixeiro Viajante, TSPLIB [30].

Dentre as 169 instâncias, 150 possuem valor ótimo conhecido. O melhor limite dual para as 19 restantes foi apresentado por Oncan et al. [24].

Grupo 2

Composto por 101 instâncias com $299 \leq |V| \leq 783$, introduzidas por Oncan et al. [24]. Tais instâncias também são adaptadas do TSPLIB. Não há limites duais na literatura para essas

instâncias. Nas instâncias geradas a partir do TSPLIB, o agrupamento dos vértices foi realizado por meio de duas técnicas: *Cluster Centering e Grid Clusterization,* descritas a seguir.

Em *Cluster Centering,* os vértices são agrupados em $m = \left\lceil \frac{|V|}{5} \right\rceil$ grupos. Primeiramente m vértices são selecionados como centro de cada grupo. Para tanto, o primeiro centro é selecionado aleatoriamente, o segundo é o mais distante do primeiro, o terceiro é o mais distante dos outros dois, e assim sucessivamente. Em seguida, cada um dos vértices restantes é associado ao centro mais próximo.

A técnica *Grid Clusterization* foi aplicada apenas em instâncias cujas coordenadas são conhecidas. O plano cartesiano é dividido de forma a gerar uma grade de dimensões $NG \times NG$, onde cada célula corresponde a um grupo de vértices. O número NG é o menor inteiro tal que os grupos contenham no mínimo em e no máximo $|V|/\mu$ vértices. O parâmetro μ expressa o número aproximado de vértices por grupo da instância.

A tabela abaixo apresenta o número de instâncias geradas por cada técnica de agrupamento. A segunda coluna é referente às instâncias cujo agrupamento segue a disposição geográfica dos vértices.

Grupo	Geo	*Cluster Centering*	*Grid Clusterization*				Total
			$\mu = 3$	$\mu = 5$	$\mu = 7$	$\mu = 10$	
1	5	36	32	32	32	32	169
2	-	21	20	20	20	20	101
Total	5	57	52	52	52	52	270

Instância	$\|V\|$	$\|E\|$	m	LB (Oncan)	Geração de Cortes	
					LB	Tempo (s)
Cluster centering						
40d198	198	2371	40	**7044**	**7044**	21,53
41gr202	202	2595	41	**242**	**242**	26,18
45ts225	225	1834	45	**62246**	**62246**	66,26
46pr226	226	1522	46	**55515**	**55515**	165,55
Grid Clusterization com $\mu = 3$						
67d198	198	1375	67	**8283**	**8283**	10,25
68gr202	202	1926	68	**292,5**	**292,5**	41,65
75ts225	225	900	75	**79019**	**79019**	3,79
84pr226	226	891	84	**62527**	**62527**	9,7
Grid Clusterization com $\mu = 5$						
40d198	198	2675	40	**7098**	**7098**	15,67
41gr202	202	6300	41	**232**	**232**	176,59
45ts225	225	2945	45	60578	***60578,7**	269,2
50pr226	226	1947	50	**56721**	**56721**	40,81
Grid Clusterization com $\mu = 7$						
32d198	198	3222	32	**6501**	**6501**	44,41
31gr202	202	5184	31	**202**	**202**	49,76
35ts225	225	2649	35	**50813**	**50813**	55,45
33pr226	226	1890	33	**48249**	**48249**	220,32
Grid Clusterization com $\mu = 10$						
25d198	198	4003	25	**6185**	**6185**	246,9
21gr202	202	9052	21	**177**	**177**	194,38
25ts225	225	4180	25	**40339**	**40339**	152,51

4.2 Ambiente computacional do experimento e conjunto de parâmetros

Os testes computacionais foram realizados em um computador Celereon 1.1GHZ e 2GB de memória principal, rodando um sistema operacional Linux Lubuntu 19.10. Os programas foram implementados na linguagem C++ e compilados com g++ versão 4.0.2 utilizando as opção de compilação -o main. Para fazer o teste a máquina precisa de uma biblioteca do C++ chamado boost (*https://www.boost.org/*) pode ser da versão 1.75.0 para fazer a instalação da na sua máquina para funcionar os recursos usados para resolver este problema. O gerador de números aleatorios foi o da biblioteca random. O número de interações que foram feitos foi padrão sendo 100 no guloso e no guloso randomizado e 300 para o guloso randomizado reativo, foram 30 execuções no guloso e no guloso randomizado e 90 no guloso randomizado reativo, para o valor dos gulosos randomizados α = 0.6,0.7 e 0.9.O tamanho do bloco em 5 da atualiazção das probabilidades adotados no algoritmo guloso randomizado reativo.

4.3 Resultados quanto à qualidade e tempo

Para verificar a qualidade de uma solução heurística foi computado e GAP em relação as melhores soluções conhecidas. Para avaliação dos resultados obtidos neste trabalho foi utilizada a diferença percentual entre a melhor solução encontrada pelo algoritmo a ser comparado e o melhor resultado contido na literatura.

$$GAP = 100 * \frac{f_{algoritmo} - f_{melhor}}{f_{melhor}}$$

Na equação, $f_{algoritimo}$ é a melhor solução encontrada pelo algoritimo que está sendo avaliado (valor da função objetivo) e f_{melhor} é a melhor solução obtida pelos algoritimos da literatura.

Nesta seção, são feitas as comparações dos resultados obtidos pelas abordagens propostas com os melhores resultados contidos na literatura. Para cada uma das instâncias os algoritmos foram executados 30 vezes, considerando os parâmetros definidos na seção 4.4 e os resultados obtidos pela função objetivo foram utilizados para calcular o GAP. O tempo de execução dos algoritmos foi contabilizado considerando o tempo de CPU. O tempo máximo definido para a execução do modelo.

Assim, as heurísticas de construção foram executadas para cada instância em duas diferentes formas: gulosas e gulosas randomizadas reativas. A tabela abaixo exibe os resultados gerados pelos algoritmos de construção guloso (coluna 3), o guloso randomizado (coluna 4 a 6) reativo (coluna 7). Os valores destacados em negrito indicam os melhores resultados. Nos resultados apresentados na tabela abaixo, a primeira coluna indica o número da instância utilizada, a segunda coluna mostra o valor da melhor média das soluções encontradas dentre todas as execuções realizadas para os diferentes algoritmos apresentados.

As demais colunas da tabela fornecem detalhadamente as métricas resultantes das execuções de cada abordagem, como o desvio relativo da média das soluções do algoritmo com relação á melhor média para instância (coluna 2), e o tempo médio de execução, dado em segundos. Observação: Todas instâncias foram utilizadas conforme os documentos disponibilizados pelo docente para este problema foi utilizado os dois documentos para este problema.

Instância	Guloso			Randomizado 0.6			Randomizado 0.7			Randomizado 0.8			Reativo		
	Tempo(s)	Custo	Gap(%)	Tempo(s)	Custo	Gap(%)	Tempo(s)	Custo	Gap(%)	Tempo(s)	Custo	Gap(%)	Tempo(s)	Custo	Gap(%)
107ali535.GTP	0,30282	121107	0,86	0,698	121086	0,36	0,701	121097	0,39	0,699	121107	0,36	0,698	121074	0,31
107att532.GTP	0,7221	44978	1,87	0,6821	36796	1,67	0,657	33798	1,58	0,631	24236	1,16	0,601	18072	0,87
99d493.GTP	0,7321	45097	1,93	0,6923	36891	1,77	0,687	33843	1,62	0,641	24311	1,24	0,631	18670	0,92
132d657.GTP	0,3597	36979	0,87	0,3487	33227	0,55	0,3108	29870	0,43	0,2768	22157	0,31	0,2279	20679	0,25
84FL417.GTP	0,436	9347	0,97	0,397	9106	0,89	0,358	8903	0,86	0,328	8432	0,59	0,251	8106	0,15
Grid Clusterization com u=3															
181ali535.GTP	0,3951	206891	1,32	0,3597	193224	1,21	0,3354	169874	0,87	0,2836	154779	0,67	0,1897	145978	0,17
182att532.GTP	0,4023	213477	1,47	0,3972	197887	1,46	0,3547	178678	0,92	0,3047	162241	0,72	0,1678	146887	0,16
171d493.GTP	0,4234	387200	1,56	0,40023	356770	1,35	0,3782	310698	1,1	0,2681	287006	0,64	0,1257	224880	0,23
221d657.GTP	0,5978	498721	1,68	0,44318	458790	1,33	0,41337	410754	1.2	0,3721	387069	0,76	0,3175	287322	0,31
142fl417.GTP	0,982	10547	1,29	0,897	10431	1,23	0,821	9881	0,89	0,799	9672	0,73	0,723	8635	0,35
Grid Clusterization com u=5															
108ali535.GTP	0,925	143570	1,21	0,879	138790	1,1	0,856	126870	0,98	0,823	121570	0,61	0,723	113981	0,26
110att532.GTP	0,983	159800	1,34	0,921	151004	1,2	0,81	148347	0,87	0,92	123479	0,51	0,754	123457	0,32
102d493.GTP	0,1002	234580	1,18	0,987	216970	1,06	0,861	198310	0,93	0,852	182640	0,57	0,821	172681	0,29
137d657.GTP	0,2687	356871	1,39	0,2548	310687	1,09	0,2497	287006	0,97	0,2377	259703	0,49	0,2432	214621	0,23
93fl417.GTP	0,746	12480	1,26	0,651	11069	1,03	0,534	10068	0,87	0,435	9844	0,34	0,331	8624	0,28
Grid Clusterization com u=7															
83ali535.GTP	0,1697	135780	1,19	0,1578	121597	1,07	0,1435	112578	0,83	0,1401	109872	0,31	0,1321	104191	0,18
80att532.GTP	0,1547	310378	0,1498	0,1687	298067	1,12	0,1431	246808	0,85	0,1357	224598	0,39	0,1257	182040	0,33
78d493.GTP	0,871	29761	0,798	0,982	26789	1,09	0,753	23447	0,97	0,647	20334	0,34	0,514	18251	0,31
96d637.GTP	0,935	37231	0,865	0,1253	35470	1,13	0,812	26897	0,83	0,735	24235	0,36	0,698	19245	0,32
61dl417.GTP	0,485	16542	0,365	0,368	13547	1,17	0,254	10214	0,88	0,168	10003	0,28	0,122	9367	0,3
Grid Clusterization com u=10															
57ali535.GTP	0,782	1462258	0,694	0,658	1297872	1,19	0,502	1036870	0,81	0,431	924234	0,27	0,351	854720	0,24
57att532.GTP	0,642	325488	0,536	0,593	278931	1,14	0,453	257803	0,92	0,322	204680	0,29	0,251	14312	0,36
52d493.GTP	0,784	31424	0,689	0,611	26870	1,08	0,498	24597	0,91	0,369	21698	0,32	0,269	15594	0,26
73d657.GTP	0,1253	363479	0,1003	0,936	312368	1,03	0,847	269879	0,8	0,624	224687	0,34	0,521	169821	0,21
42dl417.GTP	0,486	28792	0,369	0,401	25069	1,16	0,359	21036	0,83	0,237	13697	0,26	0,131	8689	0,25

Feito por Ivanylson HG
Lubuntu 19.10

5 Conclusões e Trabalhos Futuros

Este trabalho abordou uma generalização do Problema da Árvore Geradora Mínima, denominado Problema da Árvore Geradora Mínima Generalizado.

Foram implementados algoritmos construtivos da literatura e alguns propostos neste trabalho. Também foi proposto um mecanismo de reconexão de caminhos e de busca local iterada. O desempenho de cada algoritmo construtivo foi verificado em instâncias apresentadas na literatura. A partir desses experimentos, seis versões GRASP foram propostas e comparadas por meio de testes experimentais.

Os resultados computacionais indicaram que a utilização de mecanismos adicionais de aprimoramento propicia uma melhora significativa do desempenho da heurística GRASP, independentemente do algoritmo de construção utilizado. As versões que não contem nenhum desses mecanismos obtiveram resultados piores que as demais em praticamente todos os experimentos realizados. Além disso, as versões GRASP que aplicam busca local iterada encontram os melhores custos para mais de 80% das instâncias testadas.

A utilização de mais de um algoritmo de construção tende a fazer com os algoritmos encontrem boas soluções para um número maior de instâncias. Apesar do tempo de execução ser ligeiramente maior, as versões adaptativas apresentam um comportamento mais uniforme em relação às demais.

Em comparação com algoritmos da literatura, o GRASP adaptativo com reconexão de caminhos e busca local iterada é capaz de encontrar custos melhores para 22 das 101 instâncias, enquanto em apenas 8 instâncias seu custo médio não atinge o melhor resultado da literatura, obtido por um algoritmo de busca tabu. Em relação ás instâncias de que se conhece o limite dual, o gap médio das soluções do GRASP é de 0,031% contra 0,016% do algoritmo tabu. Foi constatado também que o tempo de execução do GRASP proposto é compatível com os tempos apresentados em trabalhos relacionados.

Também foi proposto um algoritmo de geração de cortes para o cálculo de limites duais baseado em uma formulação para o Problema de Steiner em Grafos Direcionados. O limite encontrado pelo algoritmo corresponde ao valor ótimo de todas as instâncias cujo ótimo é conhecido. Em relação às 101 instâncias de que não se conhece um limite dual, o algoritmo encontrou o limite para 82 delas.

O conceito de Distância *Bottleneck* utilizado no pré-processamento de instâncias do Problema de Steiner em Grafos foi adaptado para o PAGMG. Com isso, foram produzidas regras que propiciam a redução das instâncias para 14% do seu tamanho original, em média.

Como trabalhos futuros, propõe-se um estudo mais aprofundado de técnicas de memória adaptativa, para a implementação de algoritmos que possam ser configurados no decorrer da execução, propiciando uma calibração mais adequada ás características de cada instância. Tal

estudo envolve, por exemplo, a calibração do prâmetro $\propto$ e de parâmetros específicos dos algoritmos apresentados.

Foram implementadas adaptações do algoritmos de Kruskal e Prim como algoritmos construtivos. Assim, como trabalhos futuros, propõe-se a implementação da adaptação de Sollin e comparação de seu desempenho com as demais heurísticas. Além disso, propõe-se também a estipulação de uma distância mínima para busca local, como por exemplo, a ideia de vizinhança variada (VNS).

Em relação ao algoritmo de geração de cortes, é interessante implementar um procedimento de remoção de cortes que não são relevantes para o problema linear corrente. Também é interessante a implementação de um algoritmo *brach-and-cut* com base nessa geração de cortes.

6 Referências

[1] AHUA, R.K, Magnant, T.L. *Network Flows: Theory, Algorithms and Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, 1993

[2] Aiex, R. Resende, M. Ribeiro, C. *Probability distribution of solution time in GRASP: An experimental investigation. Jornal of Heuristics 8 (2002),* 343-373

[3] Den Besten, M.L., Stutzle, T. Dorigo, M. *Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In Applications os Evolutionary Computing: Proceedings of Evo Workshps 2001 (Berlim, Germant, 2001),* E. J. W. Boers, J. Gottlieb, P. L. Lanzi, R. E. Smith, Computer Science, Springer Verlag, p. 441-452

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

[5] Duin, C. W. Volgenanti, A. Voss, S. *Solving group Steiner problems as Steiner problems. European Journal of Operation Research 154 (2004)* 323-329

[6] Feo, T., Resende, M. Smith, S. *A greedy randomized adaptive serach procedure for maximum independente set. Operations Research 42* (1994), 860-878

[7] Ferem ANS, C. *Generalized Spanning Trees and Extentions.* PhD thesis, Université Libre de Bruxelles, *2001*

[8] Ferem ANS, C. Labré, M. Laporte, G. *On generalized minimum spanning trees. European Journal of Operational Research* 134 (2001), 457-458

[9] Ferem ANS, C. Labré, M. Laporte, G. *On generalized minimum spanning trees. European Journal of Operational Research* 134 (2001), 457-458