Implementação Paralela de uma Metaheurística GRASP com Path-Relinking para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos

Fabiano Vieira de Alvarenga[1], Marcelo Lisboa Rocha[1]

Marluce Rodrigues Pereira[2]

[1]Departamento de Ciência da Computação – Fundação UNIRG
Alameda Madrid Nº 545, Jardim Sevilha, CEP 77410-470, Gurupi – TO – Brasil
{fabianovieiraa, marcelolisboarocha}@yahoo.com.br

[2]Departamento de Ciência da Computação – UFLA
Caixa Postal 37, CEP 37200-000, Lavras – MG – Brasil
marluce@dcc.ufla.br

Resumo

A metaheurística GRASP é um processo iterativo, cujas iterações consistem de duas fases: uma fase de construção e outra de busca local. O algoritmo retorna a melhor solução encontrada depois de um determinado número de iterações. Neste trabalho, a metaheurística GRASP é aplicada conjuntamente à técnica path-relinking a um problema variante da Árvore Geradora Mínima (AGM), denominado Árvore Geradora de custo Mínimo com Grupamentos (AGMG). O objetivo é reduzir o tempo computacional e melhorar a qualidade das soluções GRASP através de uma implementação paralela desta metaheurística. Os resultados obtidos mostraram speedup linear para esta implementação.

1. Introdução

Os problemas de otimização combinatória possuem alta complexidade em sua solução, sendo em geral NP-Difíceis. Assim sendo, não é interessante a aplicação de técnicas exatas para solução de todos os problemas de otimização, principalmente quando se considera instâncias de grandes dimensões [4]. Assim, é comum a utilização de técnicas heurísticas ou metaheurísticas, tais como Algoritmos Genéticos [6] e *Greedy Randomized Adaptive Search Procedure* (GRASP) [3], para resolvê-los.

As metaheurísticas consistem em um processo iterativo ou refinamento de solução de problema que buscam organizar e direcionar heurísticas subordinadas, pela combinação de diferentes conceitos. Podem manipular uma solução completa, incompleta ou um conjunto de soluções tentando evitar parada prematura, através de mecanismos que permitam escapar de um ótimo local. Por isso, que a utilização de metaheurísticas permite obter-se resultados melhores do que com as heurísticas tradicionais.

O problema da Árvore Geradora Mínina (AGM) consiste em um grafo não direcionado, conectado e associado a cada arco uma distância (custo, tempo, etc) não negativa. O objetivo é encontrar o caminho mais curto de tal maneira que os arcos forneçam um caminho entre todos os pares de nós. Uma variante deste problema é a Árvore Geradora de Custo Mínimo com Grupamentos (AGMG). Esta variante consiste em particionar o conjunto de vértices do grafo associado em grupamentos disjuntos, onde a árvore geradora resultante não deve ligar necessariamente todos os nós e sim, todos os conjuntos. Desta forma, pelo menos um nó de cada um dos grupamentos terá uma ligação.

Existem diversas aplicações práticas que podem ser modeladas como o problema da Árvore Geradora de Custo Mínimo com Grupamentos (AGMG) [2, 11]. Por isso, sua solução de forma eficiente é de grande importância. Em busca desta eficiência, este trabalho apresenta a implementação paralela da metaheurística GRASP com *path-relinking* para o problema da AGMG, com o intuito de se encontrar boas soluções em um tempo computacional razoável.

As seções seguintes estão organizadas da seguinte forma. A Seção 2 apresenta detalhes sobre o problema da AGMG. A Seção 3 aborda a técnica *path-relinking*. A Seção 4 apresenta a heurística GRASP. A Seção 5 apresenta o algoritmo GRASP com *path-relinking*. A Seção 6 aborda os resultados computacionais e a Seção 7 as conclusões e trabalhos futuros.

2. Problema da Árvore Geradora de Custo Mínimo com Grupamentos

O problema da AGMG é uma variante da AGM. O problema consiste em particionar o conjunto de vértices de um grafo em subconjuntos disjuntos, onde a árvore geradora resultante não deve ligar necessariamente todos os vértices e sim todos os conjuntos.

A AGMG consiste em um grafo $G = (V, E)$ onde V representa um conjunto de vértices e E é o conjunto de arestas que ligam os vértices de V. O conjunto de vértices V é particionado em k grupamentos, onde $V = V_1 \cup V_2 \cup ... \cup V_k$. O objetivo da AGMG é ligar pelo menos um nó (vértice) de cada grupamento (conjunto), de modo que a árvore geradora possa ligar todos os grupamentos do grafo com custo mínimo, onde o custo corresponde à soma dos valores de todas as arestas que formam a AGMG.

Esta variante da AGM é classificada como NP-Difícil [2]. Somente para alguns casos particulares, este problema possui complexidade polinomial [10]. Por exemplo, quando o número de grupamentos for igual a um, a solução é formada por um único vértice. Já, quando houver exatamente dois grupos, a solução será uma aresta de menor peso conectando estes dois grupos. Outro caso particular simples de ser resolvido é quando cada grupamento possui um único elemento, reduzindo a AGMG ao problema da AGM [10].

Na Figura 1, tem-se um exemplo de uma aplicação para sistemas de irrigação em áreas desérticas, onde se tem uma área (representada por um grafo) particionada em 8 regiões (grupamentos) que necessitam de irrigação. O problema consiste em criar uma rede de irrigação de menor caminho que possa tocar no mínimo um vértice de cada região, onde o nó fonte é onde se localiza a fonte de água. Esta rede não pode cruzar as regiões, podendo apenas passar pelas suas fronteiras (arestas). Assim, este problema da irrigação pode ser modelado com uma AGMG. Considerando um Grafo $G=(V, E)$, os vértices de V representam os vértices das fronteiras das regiões, incluindo a fonte e, o peso de cada aresta de E corresponde à distância entre os seus vértices.

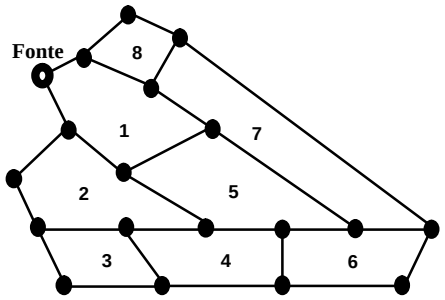

Figura 1. Uma aplicação para o problema da AGMG.

Na Figura 2, é mostrada uma solução para o sistema de irrigação da Figura 1, onde se observa a AGMG que está representada por linhas mais grossas, partindo do nó fonte e percorrendo todos os grupamentos (regiões) do grafo, ligando pelo menos um vértice de cada grupamento, sem cruzar as fronteiras (arestas) ou formar ciclos.

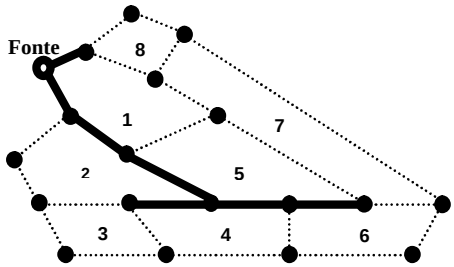

Figura 2. Solução para o grafo da Figura 1.

Na literatura, o trabalho mais relevante em ABMG foi realizado por Dror et al. [2], onde são apresentadas uma formulação matemática, três heurísticas de construção, uma de busca local e um Algoritmo Genético. A segunda referência é o trabalho de Ochi et al. [10] onde os autores apresentam um Algoritmo Genético incorporando módulos de busca local.

3. Path-Relinking

O uso de *path-relinking* (PR) em um procedimento GRASP como uma estratégia de intensificação aplicada a cada ótimo local foi primeiramente proposto por [7]. Esta primeira utilização foi seguida por diversas extensões, melhorias e aplicações bem sucedidas [1]. Estratégias de implementação são investigadas com detalhes por Resende e Ribeiro [12]. De acordo com [13] as duas estratégias básicas de aplicação de *path-relinking* em procedimentos GRASP são as seguintes:

- aplicada como uma estratégia de pós-otimização entre todos os pares de soluções de elite;
- aplicada como uma estratégia de intensificação a cada ótimo local obtido após a fase de busca local.

A aplicação da técnica de *path-relinking* como uma estratégia de intensificação a cada ótimo local é mais eficaz do que empregá-la como um procedimento de pós-otimização [13]. Neste contexto, a *path-relinking* é aplicada a pares (x_1, x_2) de soluções, onde x_1 é uma solução localmente ótima obtida após o uso do procedimento de busca local e x_2 é uma solução selecionada aleatoriamente de um conjunto formado por um número limitado, MaxElite, de soluções de elite encontradas ao longo da execução do GRASP. Este conjunto está, originalmente, vazio. Cada solução obtida pela busca local é considerada como uma candidata a ser inserida no conjunto de elite, se ela é diferente de todas as outras soluções que estão atualmente neste conjunto [8]. Se o conjunto de elite já possui MaxElite soluções e a candidata é melhor que a pior solução existente no conjunto, então a primeira substitui a última. Se o conjunto de elite ainda não está completo, a candidata é simplesmente inserida.

4. GRASP

GRASP (*Greedy Randomized Adaptive Search Procedure*) é um processo iterativo, onde cada iteração GRASP consiste em duas fases, uma fase de construção e uma fase de busca local [3]. Na Figura 3 é apresentado o pseudocódigo do algoritmo GRASP.

4.1. Fase de Construção

Na fase de construção, a solução viável é iterativamente construída elemento por elemento. A heurística é adaptativa porque os benefícios associados com cada elemento são atualizados a cada iteração da fase de construção para refletir as mudanças ocorridas pela seleção de elementos anteriores.

```
Procedimento GRASP
1   f(s) ←∞ ;
2    Para k de 1 até MaxIter faça
3        x ← GulosoAleatorizado();
4        x ← BuscaLocal(x);
5          Se f(s') < f(s) então
6             s ← s' ;
7          Fim Se
8    Fim Para
9    Retornar s ;
Fim Procedimento
```

Figura 3. Algoritmo GRASP.

A parte aleatória corresponde à forma de escolha dos melhores candidatos da lista. Cada iteração é composta por três subclasses [9]:

- Construção da Lista Restrita de Candidatos (LRC), a qual contém um conjunto reduzido de elementos candidatos a pertencer à solução;
- Escolha aleatória do elemento na LRC e inclusão de elemento na solução;
- Adaptação ou recálculo da função gulosa para os elementos ainda não pertencentes à solução.

A melhor solução encontrada ao longo de todas as iterações GRASP realizadas é retornada como resultado. Na fase de construção do GRASP, uma solução viável é construída iterativamente, um elemento da solução por vez, até que a solução esteja completa. Os elementos candidatos que compõem a solução são ordenados em uma lista, chamada de lista de candidatos. Esta lista é ordenada por uma função gulosa que mede o benefício que o elemento escolhido mais recentemente concede à parte da solução já construída. Um subconjunto LRC é formado pelos melhores elementos da lista de candidatos.

O tamanho da LRC é controlado por um parâmetro α, onde para $\alpha = 0$ tem-se um comportamento puramente guloso do algoritmo e para $\alpha = 1$, um comportamento aleatório. A componente probabilística do método é devida à escolha aleatória de um elemento da LRC. Este procedimento permite que diferentes soluções de boa qualidade sejam geradas.

Desta forma, o principal parâmetro a ser configurado no GRASP é a cardinalidade da LRC. A média e a variância do valor de função objetivo das soluções construídas são diretamente afetadas por tal parâmetro [1] e [12].

4.2. Fase de Busca Local

Métodos de busca local em problemas de otimização constituem uma família de técnicas baseadas na noção de vizinhança, ou seja, são métodos que percorrem o espaço de pesquisa passando, iterativamente, de uma solução para outra que seja sua vizinha.

A fase de busca local de GRASP aproveita a solução inicial da fase de construção e explora a vizinhança ao redor desta solução. Se um melhoramento é encontrado, a solução corrente é atualizada e novamente a vizinhança ao redor da nova

solução é pesquisada. O processo se repete até nenhum melhoramento ser encontrado.

É preciso ter cuidado em:

- Escolher uma vizinhança apropriada;
- Usar estruturas de dados eficientes para acelerar a busca local;
- Ter uma boa solução inicial.

5. Algoritmo GRASP com Path-Relinking Proposto

A aplicação da técnica de *path-relinking* no procedimento GRASP tem como objetivo melhorar a qualidade das soluções obtidas pelo GRASP, sendo ilustrado na Figura 4 o pseudocódigo do algoritmo proposto (denominado GRASP-PR) utilizando a metaheurística GRASP com *path-relinking*. Nele, o usuário define o número de execuções desejadas para o algoritmo e o tamanho do conjunto de elite *MaxElite*. A partir de então, executa-se a fase de construção (linha 3 da Figura 3), que procura construir uma solução viável e de qualidade. Logo em seguida, é executada a fase de busca local (linha 4 da Figura 3), procurando refinar a solução inicial. Após a fase de construção e o refinamento da solução (busca local).

Após os métodos de construção e busca local, é criado o conjunto de soluções de elite, conforme os passos no algoritmo da Figura 4:

- Inicialmente verifica se a quantidade de soluções elite é menor que *MaxElite* (linha 5) e se a solução *y* pertence ao conjunto elite (linha 6).
- Se a solução *y* não pertence ao conjunto elite (linha 7), então a *y* é inserido no conjunto elite (linha 8).
- Se a quantidade de soluções elite não for menor que *MaxElite*, é verificado se a solução *y* é melhor do que a pior solução do *ConjElite* (linha 12). Caso a solução *y* seja melhor (linha 13), verificar se a solução *y* pertence ao *ConjElite* (linha 14).
- Se a solução *y* não pertence ao conjunto elite (linha 15), então a *y* é inserida no conjunto elite (linha 16).
- Após a criação do conjunto de soluções de elite é aplicado o *path-relinking* a cada 200 iterações do GRASP (linha 20).
- Finalmente, após os métodos de construção, busca local e a aplicação do *path-relinking* a melhor solução é retornada (linha 25).

```
Procedimento GRASP-PR
1   f(S) ← 0; ConjElite ← 0; MaxElite ← n;
2   Para i de 1 até MaxIter Faça
3       Aplicar o procedimento de construção
        para obter uma solução viável S ;
4       Aplicar busca local em S gerando uma nova
        solução S* ;
5       Se ContElite < MaxElite então
6           Verificar se y ∈ ConjElite;
7           Se y ∉ ConjElite Então
8               Inserir y no ConjElite;
9           Fim Se
10      Fim Se
11      Senão
12          Verificar se y é melhor que o pior
            do ConjElite;
13          Se y melhor Então
14              Verificar se y ∈ ConjElite;
15              Se y ∉ ConjElite Então
16                  Inserir y no ConjElite;
17              Fim Se
18          Fim Se
19      Fim Senão
20      Aplicar path_relinking a cada n iterações;
21      Se custo de f (S*) < f (S) Então
22          S ← S* ;
23      Fim Se
24  Fim Para
25  Retornar S;
Fim Procedimento
```

Figura 4. Algoritmo GRASP com *path-relinking*.

5.1. Algoritmo Path-Relinking

O algoritmo de *path-relinking* unidirecional inicia determinando o conjunto de movimentos *(S', E')* que será aplicado a *S*(solução inicial) até chegar a *E* (solução guia).

Cada iteração do procedimento de *path-relinking* unidirecional possui os seguintes passos:

- Sortear uma solução do conjunto de elite *(E')* (linha 2);
- Verificar e marcar na solução *E'* e na solução inicial obtida *(S')* as arestas que são iguais nas duas (linha 4, 5 e 6);
- Se o número de arestas de *S'* for menor que o de *E'*, fazer os testes de troca das arestas de *S'* e *E'* diferentes. Após isto, inserir em *S'* as arestas de *E'* que faltam. "A cada troca, criar a árvore e fazer o processo de busca local - poda" (linha 10, 11 e 12).
- Se o número de arestas de *S'* for maior que o de *E'*, fazer os testes de troca das arestas de *S'* e *E'* diferentes. Após isto, retirar de *S'* as arestas que não

têm em *E'*. "A cada troca, criar a árvore e fazer o processo de busca local - poda" (linha 14, 15 e 16).

- Se o número de arestas de *S'* for igual ao de *E'*, fazer os testes de troca das arestas de *S'* e *E'* diferentes. "A cada troca, criar a árvore e fazer o processo de busca local - poda" (linha 18, 19 e 20).
- Atualizar a solução corrente *S* (linha 22).
- Verificar se a solução corrente, *S* é melhor que a melhor solução, $S^{mellhor}$ (linha 24). Caso afirmativo, atualizar a $S^{mellhor}$ (linha 24).

Uma representação do algoritmo *path-relinking* é descrita conforme a Figura 5.

5.1.1. Algoritmo GRASP com Path-Relinking Paralelo (GRASP-PRP)

O principal motivo da paralelização da heurística GRASP foi o de melhorar o desempenho da técnica e também pela facilidade de paralelização da mesma. A heurística GRASP paralela foi implementada seguindo o modelo mestre-escravo, de acordo com os seguintes passos:

1. O processador mestre recebe o número de iterações passadas pelo usuário, para serem executadas pelo algoritmo.
2. O processador mestre divide o número de iterações pelo número de processadores escravos mais um.
3. O processador mestre e os escravos executam o mesmo trecho de código da heurística GRASP seqüencial, com o número de iterações passado a cada um deles.
4. O processador mestre é então o responsável por receber as soluções dos escravos, o mestre compara todas as soluções incluindo a sua, e posteriormente seleciona-se a melhor entre todas.

Em geral, cada processador executa *MaxIter/p* iterações, onde *MaxIter* e *p* são, respectivamente, o número total de iterações do GRASP e o número de processadores. Cada processador possui uma cópia do algoritmo seqüencial, uma cópia dos dados do problema e uma semente independente para gerar sua própria seqüência de números aleatórios. Neste caso, uma única variável global é necessária para armazenar a melhor solução dentre aquelas encontradas por todos os processadores.

Um dos processadores atua como mestre, lendo e distribuindo os dados do problema, distribuindo as iterações pelos processadores e coletando a melhor solução obtida por cada um deles.

```
Procedimento PR;
1  S^melhor ← S;
2    Sortear um solução E'
3    Para i de 1 até n_va faça
4        Se as aresta de E' e S' são iguais então
5            Marcas as arestas de E e S;
6        Fim Se
7    Fim Para
8    Calcular números de movimentos
9    Para i de 1 até num_movimentos faça
10       Se S'(n_a) menor E' (n_a) então
11           Fazer teste de trocas S(a) ≠ E(a) ;
12           A cada troca criar um arvore e aplicar
             busca local e poda
13       Fim Se
14       Se S'(n_a) maior E'(n_a) então
15           Fazer teste de trocas S(a) ≠ E(a) ;
16           A cada troca criar um arvore e aplicar
             busca local e poda;
17       Fim Se
18       Se S'(n_a) igual E' (n_a) então
19           Fazer teste de trocas S(a) ≠ E(a) ;
20           A cada troca criar um arvore e  aplicar
             busca local e poda
21       Fim Se
22       S ← solução do melhor movimento;
23       Se S < S^melhor então
24           S^melhor ← S;
25       Fim se
26   Fim Para
27 Retornar S^melhor ;
Fim Procedimento PR
```

Figura 5. Pseudo-código do procedimento de *path-relinking*.

6. Resultados Computacionais

Os algoritmos foram submetidos às instâncias disponibilizadas por [2] para o problema AGMG. Cada uma das 20 instâncias é um grafo conexo e particionado em vários grupamentos, onde o número de grupamentos varia de 4 a 50, o número de nós varia de 25 a 500 e o número de arestas varia de 50 a 5000. Cada grupamento possui inúmeros vértices conectados entre si e a cada aresta é atribuído um peso (custo, valor ou distância).

O algoritmo GRASP-PRP foi implementado utilizando a linguagem C e a biblioteca *Message Passing Interface* (MPI) de troca de mensagens [14]. O algoritmo foi executado em um *cluster* disponível no Núcleo de Atendimento em Computação de Alto Desempenho (NACAD) da Coordenação dos Programas de Pós-graduação de Engenharia (COPPE) da Universidade Federal do Rio Janeiro (UFRJ). O

cluster é formado por 16 nós, cada nó com 2 processadores Pentium III de 1 Ghz e 512 Mb de RAM, interligados por uma rede Gigabit Ethernet e sistema operacional Linux RedHat 7.3.

Na execução das 20 instâncias (Gmst1, Gmst2,..., Gmst20) utilizadas para teste, o algoritmo GRASP-PRP foi executado utilizando os seguintes parâmetros:

- Alfa: sorteado aleatoriamente entre 0 e 1 a cada iteração do GRASP;
- Critério de parada do GRASP: 5000 iterações;
- O *path-relinking* foi aplicado a cada 200 iterações do GRASP;
- Tamanho do conjunto de elite: 10.

Adiante, serão feitas análises e comparações dos resultados obtidos pelo GRASP-B (GRASP sem *path-relinking*) e pelo GRASP-PR com outro resultado da literatura conforme especificado em [16]. Aqui será feita uma comparação com o Algoritmo Genético proposto em [2] aqui denominado AG-DROR. Os resultados são apresentados na Tabela 1.

Tabela 1. Desempenho do algoritmo AG-DROR, GRASP-B e GRASP-PR.

	AG-DROR		GRASP-B		GRASP-PR	
Instâncias	**Solução**	**Tempo**	**Solução**	**Tempo**	**Solução**	**Tempo**
Gmst1	23	1,31	23	0,00	23	0,59
Gmst2	41	4,61	41	0,00	41	0,69
Gmst3	36	7,80	36	0,00	36	0,75
Gmst4	18	5,77	18	0,00	18	2,01
Gmst5	27	28,45	27	0,00	27	3,14
Gmst6	60	25,38	55	0,00	55	5,82
Gmst7	67	60,37	67	6,19	67	5,99
Gmst8	55	132,81	53	0,00	53	7,97
Gmst9	35	38,66	37	0,00	37	9,08
Gmst10	50	109,20	49	2,31	48	14,16
Gmst11	62	86,84	50	2,15	50	22,52
Gmst12	64	101,50	75	13,77	75	25,83
Gmst13	44	107,51	44	8,07	44	49,01
Gmst14	58	600,39	58	10,21	58	52,99
Gmst15	57	201,34	60	11,46	60	74,75
Gmst16	148	310,35	124	31,75	124	133,66
Gmst17	91	1338,96	91	167,45	91	171,54
Gmst18	101	1605,90	91	195,65	91	224,52
Gmst19	98	2102,50	94	397,03	92	297,42
Gmst20	141	2283,59	127	787,62	123	1003,90

Observando-se os resultados da Tabela 1, e comparando a qualidade das soluções obtidas por GRASP-B em relação a AG-DROR, verifica-se que o GRASP-B obteve em 20 instâncias, 8 melhores resultados, 9 resultados iguais e apenas 3 resultados inferiores quando comparados aos do AG-DROR, porém em todos os casos em um tempo computacional consideravelmente menor.

Na seqüência, são apresentadas comparações entre o GRASP-PR em relação ao GRASP-B. Dos 20 resultados obtidos por GRASP-PR, 3 soluções apresentaram melhor valor em relação ao GRASP-B, sendo os outros 17 valores iguais. Contudo, a um custo computacional maior. Este acréscimo no custo computacional para o GRASP-PR foi gerado pelo acréscimo da técnica *path-relinking* ao GRASP-B. Este fato culminou na paralelização de GRASP-PR.

6.1. Análise e Comparações dos Resultados Paralelos

Os resultados obtidos são apresentados na Tabela 2, onde as instâncias variam de Gmst1 a Gmst20, com tempos de execução em segundos de CPU para 1, 2, 4 e 8 processadores (número máximo de processadores disponíveis para execução exclusiva no momento dos testes). Os respectivos tempos de execução correspondem à média de 10 execuções. Ressalta-se ainda que o GRASP-PRP utilizando um processador tem o mesmo desempenho que GRASP-PR. Assim, relações entre os resultados das Tabelas 1 e 2 podem ser feitas.

Uma forma eficiente de redução do tempo computacional para obtenção das soluções obtidas pelo algoritmo GRASP com *path-relinking*, é a paralelização do mesmo. Nos gráficos das Figuras 6 a 9, pode-se observar o *speedup* para todas as instâncias em função do acréscimo de processadores. A Figura 6 apresenta o gráfico de speedup para as 5 primeiras instâncias, enquanto que as Figuras 7, 8 e 9 apresentam, respectivamente, o speedup para as instâncias de 6 a 10, de 11 a 15 e de 16 a 20**.**

Tabela 2. Desempenho do algoritmo GRASP-PRP.

Instância	**Solução**	**Tempo de execução (seg.)**			
		1 proc.	**2 procs.**	**4 procs.**	**8 procs.**
Gmst1	23	0,59	0,29	0,15	0,10
Gmst2	41	0,69	0,42	0,21	0,10
Gmst3	36	0,75	0,39	0,21	0,11
Gmst4	18	2,01	0,99	0,52	0,33
Gmst5	27	3,14	1,51	0,87	0,39
Gmst6	55	5,82	2,97	1,46	0,78
Gmst7	67	5,99	3,08	1,57	0,78
Gmst8	53	7,97	4,00	2,03	1,05
Gmst9	37	9,08	4,71	243	1,36
Gmst10	48	14,16	7,09	3,62	1,91
Gmst11	50	22,52	11,24	5,70	2,99
Gmst12	75	25,83	12,76	6,49	3,35
Gmst13	44	49,01	24,41	12,36	6,24
Gmst14	58	52,99	28,61	13,87	7,03
Gmst15	60	74,75	37,24	18,99	9,68
Gmst16	124	133,66	67,58	33,81	17,18
Gmst17	91	171,54	86,42	43,65	22,04
Gmst18	91	224,52	112,91	56,31	28,52
Gmst19	92	297,42	149,36	74,55	37,48
Gmst20	123	1003,90	490,13	244,54	127,78

Nas Figuras de 6 a 9, pode-se observar que o algoritmo GRASP-PRP proposto apresentou *speedup*

linear para 2, 4 e 8 processadores, para todas as instâncias, mostrando um bom resultado.

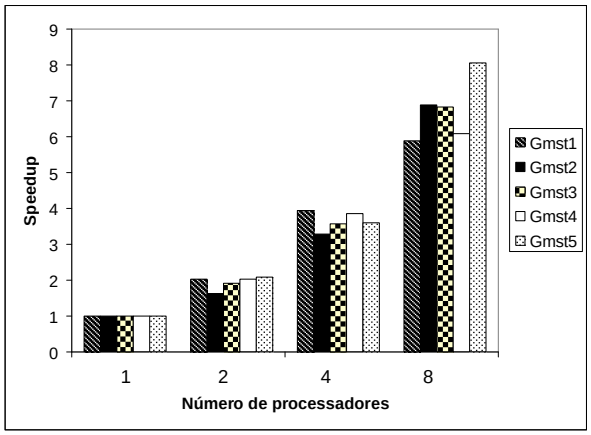

Figura 6. Speedup das instâncias Gmst1 a Gmst5.

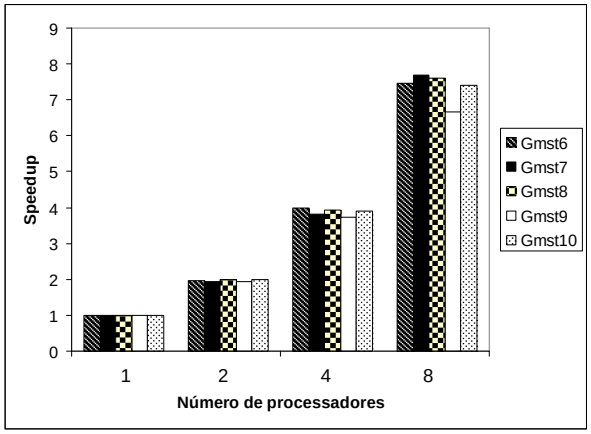

Figura 7. Speedup das instâncias Gmst6 a Gmst10 .

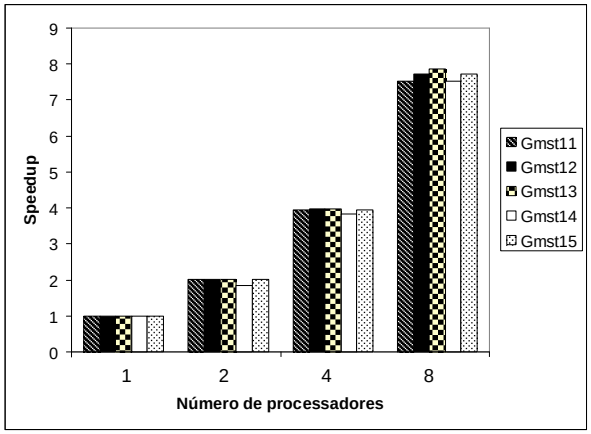

Figura 8. Speedup das instâncias Gmst11 a Gmst15.

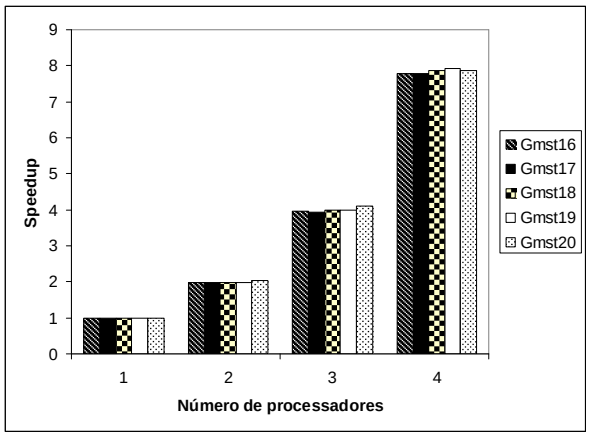

Figura 9. Speedup das instâncias Gmst16 a Gmst20.

7. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado um algoritmo paralelo para o problema da Árvore Geradora de custo Mínimo com Grupamentos (AGMG). O algoritmo desenvolvido utilizou a metaheurística GRASP, a técnica *path-relinking* e os recursos de computação paralela para obter resultados em menor tempo de execução, já que este problema é classificado como NP-Dificil, o que limita até o momento, o uso de técnicas exatas para encontrar soluções para instâncias grandes.

O algoritmo GRASP com *path-relinking* (GRASP-PR), seqüencial, melhora as soluções principalmente para instâncias de elevadas dimensões, mas apresenta um tempo computacional alto. Por isso, a heurística GRASP foi paralelizada.

O objetivo deste trabalho é encontrar boas soluções em tempo computacional razoável. O objetivo foi alcançado com a aplicação da metaheurística GRASP paralela com *path-relinking*. Os resultados obtidos para as instâncias executadas, para 1, 2, 4 e 8 processadores, apresentam *speedup* linear.

Neste trabalho o *path-relinking* adicionou ao algoritmo GRASP uma única trajetória unidirecional, começando de uma solução localmente ótima e usando como guia uma solução escolhida aleatoriamente dentre aquelas do conjunto de elite. Como trabalhos futuros pode-se:

(1) Adicionar ao algoritmo GRASP uma única trajetória com *path-relinking* unidirecional contrária à proposta neste trabalho, começando de uma solução escolhida aleatoriamente dentre aquelas do conjunto de elite e usando como guia uma solução local ótima;

(2) Combinar os dois métodos de forma que se possa fazer uma trajetória bidirecional, executando o *path-relinking* nas duas direções;
(3) Estudar outras variantes do método GRASP tais como memória e aprendizado proposto por [5], onde os mesmos observaram que o procedimento GRASP básico não usa uma memória de longo prazo (isto é, informações coletadas de iterações prévias) e propuseram um esquema para tratar esta questão em heurística do tipo multipartida. Uma outra variante interessante seria o uso do Princípio da Otimalidade Progressiva (POP), de acordo com [13] uma implantação prática deste princípio em conjunto com GRASP consiste em aplicar um busca local durante alguns instantes específicos da fase de construção, e não a cada uma de suas iterações.

Agradecimentos

Os autores agradecem à coordenação do Núcleo de Atendimento em Computação de Alto Desempenho (NACAD) da COPPE-UFRJ pela utilização do *cluster* para realização dos experimentos.

8. Referências

[1] AIEX, R. M. (2002). *Uma investigação experimental da distribuição de probabilidade do tempo de solução em heurísticas GRASP e sua aplicação na análise de implementações paralelas*. Tese de Doutorado, PUC - Rio, Departamento de Informática, Rio de Janeiro.

[2] DROR M.; HAOUARI M. and CHAOUACHI J. (2000) *Generalized Spanning Trees*. European Journal of Operational Research, 120, p. 583-592.

[3] FEO T. A. and RESENDE M. G. C. (1995). *Greedy Randomized Adaptive Search Procedures*. Journal of Global Optimization 6, p. 109-133.

[4] FEREMANS, C.; LABBÉ, M. and LAPORTE, G. (2002). *A comparative analysis of several formulations for the generalized minimum spanning tree problem*. Networks, 39, p.29-34.

[5] FLEURENT, C. and GLOVER, F.(1999). *Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory*. INFORMS Journal on Computing, 11:198--204.

[6] HOLLAND, J. H. (1975). *Adaptation in Natural and Artificial System*. MIT Press.

[7] LAGUNA, M. and MARTÍ, R. (1999). *GRASP and path-relinking for 2- layer straight line crossing minimization***.** INFORMS Journal on Computing, 11:44-52.

[8] MARINHO E. H.(2005). *Heurísticas Busca Tabu para o Problema de Programação de Tripulações de Ônibus Urbano*, Dissertação de Mestrado em Computação, Universidade Federal Fluminense, Niterói RJ.

[9] MORELLI C. D. S. e VIEIRA L. T. (1999). *Análise de Métodos Heurísticos de Característica Gulosa*. IV Semana Acadêmica do PPGC (Programa de Pós-Graduação em Computação).

[10] OCHI L.; BRUNO L. e FERNANDO C. (2003). *Técnicas Para Melhorar o Desempenho de Algoritmos Evolutivos: Uma Aplicação Para o Problema de Árvore Geradora de custo Mínimo Com Grupamentos*. III Congresso Brasileiro de Computação. Itajaí-SC, p. 661-672.

[11] POP P. C. (2004). *New Models of the Generalized Minimum Spanning Tree Problem*. Journal of Mathematical Modelling and Algorithms, pp 153-166.

[12] RESENDE M. G. C. and RIBEIRO C. C. (2003). *Greedy randomized adaptive search procedures*. In Handbook of Metaheuristics, F. Glover and G. Kochenberger, eds., Kluwer Academic Publishers, pp. 219-249.

[13] ROSSETI, I. C. M. (2003). *Estratégias seqüenciais e paralelas de GRASP com reconexão por caminhos para o problema de síntese de redes a 2-caminhos***.** Tese de Doutorado, PUC - Rio, Departamento de Informática, Rio de Janeiro.

[14] SNIR, M.; OTTO, S.T. and WALKER, D. W. (1996). *MPI: The Complete Reference*. The MIT Press.

[15] CORMEN, T. ; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C (2002). *Algoritmos: Teoria e Prática. Tradução da 2a edição Americana*. Editora Campus.

[16] ALVARENGA, F. V. e ROCHA, M. L. *Melhorando o Desempenho da Metaheurística GRASP Utilizando a Técnica Path-Relinking: Uma Aplicação para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos*. XVIII SBPO. Goiânia-GO. Publicação em CD-ROM.