UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDREI POCHMANN KOENICH

PEDRO COMPANY BECK

TURMA A

RELATÓRIO - TRABALHO FINAL

Disciplina: Otimização Combinatória

Professor: Marcus Rolf Peter Ritt

Porto Alegre, março de 2023.

1 Introdução

O relatório a seguir tem como objetivo especificar como ocorreu a implementação de uma metaheurística, destinada a resolver o problema da Seleção de Maior Distância Mínima Total. A metaheurística utilizada é conhecida como Greedy Randomized Adaptive Search Procedure (GRASP).

2 Definição do problema

O problema da seleção de Maior Distância Mínima Total pode ser enunciado da forma descrita abaixo.

Instância: um grafo bipartido completo $G = (M \cup L, A)$ com distâncias d_a nas arestas $a \in A$, e uma constante $l \leq |L|$.

Solução: uma seleção $S \subseteq L$ com l elementos.

Objetivo: para uma seleção S, definimos a distância mínima $d(m) = min_{s \in S} d_{ms}$ de um elemento $m \in M$ para um elemento do conjunto S, que foi selecionado. O objetivo é maximizar a soma das distâncias mínimas $\sum_{m \in M} d(m)$.

3 Formulação do programa inteiro

3.1 Descrições das variáveis:

$x_j \rightarrow$ variável com valor igual a um caso o vértice j seja selecionado para integrar o conjunto de vértices S, com j correspondendo a um vértice de L, ou com valor igual a zero, caso contrário.

$y_{ij} \rightarrow$ variável com valor igual a um caso o par de vértices (i, j) seja selecionado (com o vértice i correspondendo a um vértice de M, e o vértice j correspondendo a um vértice de L selecionado para integrar o conjunto de vértices S), ou com valor igual a zero, caso contrário.

3.2 Descrição das constantes, recebidas na instância de entrada:

$d_{ij} \rightarrow$ distância existente entre os pares de vértices (i, j), representada por meio dos pesos das arestas (com o vértice i correspondendo a um vértice de M, e o vértice j correspondendo a um vértice de L).

3.3 Programa inteiro:

maximiza

$$\sum_{i \in M} \sum_{j \in L} d_{ij} y_{ij}$$

sujeito a

$$\sum_{j \in L} x_j = l \qquad (1)$$

$$\forall i \in M, \sum_{j \in L} y_{ij} = 1 \qquad (2)$$

$$\forall i \in M, \forall j \in L, x_j - y_{ij} \geq 0 \qquad (3)$$

$$\forall i \in M, \forall j \in L, \forall k \in L, d_{ij}(x_k + y_{ij} - 1) \leq d_{ik} x_k \qquad (4)$$

$$\forall j \in L, x_j \in \{0,1\} \qquad (5)$$

$$\forall i \in M, \forall j \in L, y_{ij} \in \{0,1\} \qquad (6)$$

3.4 Descrições das restrições:

(1) → Garantia de que serão selecionados exatamente l vértices do conjunto L.

(2) → Garantia de que, para cada vértice do conjunto M, será selecionada exatamente uma aresta que realiza uma conexão com algum vértice de L.

(3) → Vínculo entre as variáveis x_j e y_{ij}. Garantia de que não é possível selecionar uma aresta que realiza uma conexão com um vértice de L, sem que o vértice de L em questão tenha sido selecionado.

(4) → Garantia de que, para cada vértice de M, será selecionada sempre a aresta de menor peso, que realiza uma conexão com algum vértice de L.

(5) e (6) → Garantia de que as variáveis x_j e y_{ij} assumirão apenas valores binários.

4 Descrição geral da metaheurística GRASP

Conforme [1], as metaheurísticas possuem o propósito de compensar certas deficiências que, ao longo dos anos, foram verificadas experimentalmente quando da aplicação de determinadas heurísticas para a resolução de problemas NP-difíceis. Isso é realizado por meio de diferentes estratégias para conduzir uma busca em espaços que, originalmente, não seriam explorados por meio do uso de heurísticas comuns, a fim de encontrar soluções de alta qualidade.

Nesse contexto, de acordo com [2], a metaheurística GRASP pode ser compreendida como uma metaheurística de múltiplos inícios, com cada uma de suas iterações sendo composta por duas fases, sendo uma a de construção e outra a da busca local. Na fase de construção, gera-se um conjunto de possíveis soluções, com algumas dessas soluções sendo posteriormente restritas a um grupo de candidatos. Uma vez gerado esse grupo de candidatos, escolhe-se aleatoriamente uma dessas soluções para, a partir dela, iniciar uma busca local, a fim de melhorá-la.

5 Descrição da implementação e do algoritmo, com utilização da metaheurística GRASP

5.1 Plataforma de implementação

O trabalho foi implementado e testado em sistema operacional Windows 10 64-bits, com um processador Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, com memória RAM instalada de 16GB. A linguagem de programação utilizada foi C, com utilização do compilador GNU GCC. Não foram utilizadas flags de compilação.

5.2 Pseudocódigo representando a implementação

Com base na definição geral da metaheurística GRASP apresentada na seção 4, podemos adaptá-lo para a nossa implementação da forma descrita abaixo.

```
GRASP(seed,inicios,construcoes,candidatos,arquivo_entrada,arquivo_saida)
1: matriz_adjacencia ← LeituraDados(arquivo_entrada)
2: melhor_solucao = InicializaMelhorSolucao()
3: for k = 1,...,inicios
4:    solucao ← ConstrucaoGulosaRandomizada(seed,construcoes,candidatos,matriz_adjacencia)
5:    solucao ← busca_local(solucao)
6:    if solucao > melhor_solucao then
7:          melhor_solucao ← solucao
8:    endif
9: endfor
10: GravacaoDados(melhor_solucao,arquivo_saida)
```

5.3 Parâmetros de entrada

`seed`: representa o valor utilizado para geração de números aleatórios, durante a execução da implementação. O usuário pode escolher qualquer valor inteiro do intervalo $[0,9999]$.

`inicios`: representa a quantidade de inícios aleatórios (segundo a construção gulosa-randomizada) que ocorrerão, para que a partir desses inícios ocorra a busca local.

`construcoes`: representa a quantidade de soluções aleatórias que será gerada pela implementação, na etapa da construção gulosa-randomizada. A partir de um grupo de candidatos dessas soluções aleatórias, ocorrerá uma escolha (também aleatória) da solução inicial, a partir da qual será executada a busca local.

`candidatos`: representa, dentre todas as soluções iniciais aleatórias geradas na construção gulosa-randomizada, quantas das melhores soluções obtidas participarão do "torneio" de seleção. Por exemplo, caso o valor desse parâmetro seja igual a dez, então uma dentre as dez melhores soluções (dentre todas as que foram geradas na construção) será selecionada, para que a partir dela seja iniciada a busca local.

`arquivo_entrada`: corresponde ao nome do arquivo de entrada contendo todas as informações da instância, o qual deve ser fornecido pelo usuário. As informações da instância incluem as cardinalidades dos conjuntos M e L, o valor da constante l e todos os valores associados a cada uma das arestas do grafo $G = (M \cup L, A)$.

`arquivo_saida`: corresponde ao nome do arquivo de saída que será gerado pela implementação, contendo todas as informações da melhor solução obtida, com a aplicação da metaheurística. No arquivo de saída serão informados os valores dos parâmetro `seed`, `construcoes`, `candidatos`, o valor da solução inicial (correspondente à busca local que revelou o melhor valor de máximo local), o valor da função objetivo obtido com as buscas e a descrição completa de quais são as ligações efetuadas entre os vértices dos conjuntos, junto com a respectiva distância da ligação (isto é, para cada vértice do conjunto M, será informado o vértice do conjunto L com o qual está conectado, junto com a distância representada pelo peso da aresta que realiza a conexão).

5.4 Descrições das estruturas de dados utilizadas na implementação

Para a representação da matriz de adjacência, obtida por meio da leitura do arquivo de entrada, utilizou-se um array de duas dimensões do tipo float. As dimensões do array são dadas por $|M| \times |L|$.

Para a representação de uma possível solução para a instância do problema da Seleção da Maior Distância Mínima Total, utiliza-se uma estrutura denominada `Arestas` (essa estrutura é definida com a diretiva `typedef struct`, da linguagem C) contendo um valor inteiro que representa um vértice do conjunto L e um valor do tipo float, que representa a distância existente entre um vértice

de M. Durante a implementação, é utilizado um array linear de tamanho $|M|$ definido a partir dessa estrutura, com o índice de cada posição do array representando o vértice M. Assim, a primeira posição do array irá conter as informações da ligação envolvendo o primeiro vértice do conjunto M (ou seja, qual o vértice do conjunto L com o qual ele está conectado e qual a distância da ligação), a segunda posição irá conter as informações da ligação envolvendo o segundo vértice do conjunto, e assim por diante.

Durante o procedimento de busca local, todos os vizinhos de uma determinada solução são representados por meio de um array definido a partir de uma estrutura denominada `Vizinhos` (essa estrutura é definida com a diretiva `typedef struct`, da linguagem C). Cada posição do array definido a partir dessa estrutura contém um outro array interno de valores inteiros. Nesse array mais interno, estão contidos todos os números correspondentes aos vértices do conjunto L (ou seja, esse array interno possui tamanho igual ao valor da constante l), além de um valor float representando qual seria o valor da função objetivo (ou seja, das somas das distâncias mínimas) caso realizássemos as conexões entre os vértices de M e os vértices presentes nesse array interno, com observância das restrições do programa inteiro apresentado na seção 3.3. A descrição sobre o modo como ocorrem as gerações das vizinhanças é apresentada na seção 5.7.

5.5 Inicialização da primeira solução

Uma vez que as instâncias apresentam apenas valores positivos associados às arestas do grafo em análise, optou-se por inicializar a solução inicial como zero. Essa solução será atualizada em todos os casos em que ocorrer identificação de uma solução melhor, durante a busca local executada após a construção gulosa-randomizada.

5.6 Construção gulosa-randomizada

Uma construção randomizada é realizada utilizando um array com dimensão igual a $|L|$. Inicialmente, o array é inicializado com todos os valores inteiros no intervalo $[0, |L| - 1]$. Em seguida, todos esses valores são alocados em posições aleatórias no array, por meio do algoritmo de Fisher-Yates. Por fim, os primeiros l vértices do array são selecionados para que façam parte do conjunto $S \subseteq L$ de vértices selecionados. O número de vezes em que esse processo é repetido, para a geração de várias soluções, corresponde ao parâmetro de entrada `construcoes`. A partir disso, seleciona-se alguma das melhores soluções aleatórias inicializadas para iniciar a busca local utilizando o parâmetro `candidatos` como referência, conforme explicado na seção 5.3.

5.7 Descrição da busca local e da geração das vizinhanças

Durante a busca local, uma solução parcial irá conter um determinado conjunto de vértices selecionados. Foram exploradas duas formas distintas de geração de vizinhança, sendo uma determinística e outra estocástica, com cada uma delas sendo descrita a seguir.

A geração da vizinhança de forma determinística é realizada permutando cada um dos vértices selecionados por cada um dos vértices não selecionados. Considerando que em um determinado ponto da busca local existem l vértices selecionados e $|L| - l$ vértices não selecionados, o número de vizinhos é dado, portanto, pela operação $(|L| - l) * l$.

Exemplificando, assumindo que o conjunto L seja composto pelos vértices $\{0,1,2,3,4,5,6\}$, com uma constante $l = 3$. Se, em um determinado ponto da busca local os vértices selecionados para compor a solução são $\{2,4,6\}$, então existirão $(|L| - l) * l = (7 - 3) * 3 = 12$ vizinhos, sendo eles:

$\{0,4,6\}, \{1,4,6\}, \{3,4,6\}, \{5,4,6\}$, obtidos substituindo o vértice 2, por algum vértice não selecionado;

$\{2,0,6\}, \{2,1,6\}, \{2,3,6\}, \{2,5,6\}$, obtidos substituindo o vértice 4, por algum vértice não selecionado;

$\{2,4,0\}, \{2,4,1\}, \{2,4,3\}, \{2,4,5\}$, obtidos substituindo o vértice 6, por algum vértice não selecionado.

A geração da vizinhança de forma estocástica é realizada da seguinte maneira: o primeiro vizinho é construído substituindo o primeiro vértice do conjunto de vértices selecionados pelo primeiro vértice do conjunto de vértices não-selecionados, o segundo vizinho é obtido substituindo os dois primeiros vértices do conjunto de vértices selecionados pelos dois primeiros vértices do conjunto de vértices selecionados, e assim por diante, até que todos os vértices do conjunto de vértices selecionados tenham sido substituídos, ou até que todos os vértices do conjunto de vértices não-selecionados sejam esgotados. Ao final da geração da vizinhança, o conjunto de vértices não selecionados é embaralhado, antes da geração da vizinhança para a próxima iteração referente à busca local.

Exemplificando, assumindo que o conjunto L seja composto pelos vértices $\{15,95,77,23,88,64,29\}$, com uma constante $l = 4$. Se, em um determinado ponto da busca local os vértices selecionados para compor a solução são $\{95,29,15,77\}$, com um conjunto de vértices não-selecionados igual a $\{23,88,64\}$, então existirão três vizinhos, sendo eles:

$\{23,29,15,77\}$, substituindo o primeiro vértice anteriormente selecionado;

$\{23,88,15,77\}$, substituindo os dois primeiros vértices anteriormente selecionados;

$\{23,88,64,77\}$, substituindo os três primeiros vértices anteriormente selecionados.

Uma vez que cada vizinho representa uma possível seleção de vértices, cada um deles possui um valor da função objetivo associado, representado pela soma das distâncias mínimas em relação aos vértices do conjunto M. Em cada passo da busca local, escolhe-se a seleção de vértices com o maior valor da função objetivo, até que um máximo local seja atingido, o que caracteriza a terminação da busca local.

6 Resultados obtidos com a utilização dos solvers CPLEX e GLPK

O programa linear foi implementado como um modelo no solver CPLEX versão 22.1.1, utilizando a biblioteca de C++ disponibilizada. O programa foi testado em um computador com o processador Intel Core i5-8350U de 4 cores/8 threads, 16 GB de memória RAM e sistema operacional macOS 13.2.1 Ventura. Para instâncias menores foi possível obter resultados corretos, porém dentro do tempo limite máximo de 1 hora, não foi encontrada nenhuma solução para as instâncias disponibilizadas.

Além do CPLEX, também foi implementado um modelo no solver GLPK versão 5.0, e as instâncias .ins disponibilizadas foram convertidas para o formato .dat para ser utilizadas no GLPK, também não foi possível obter nenhuma solução dentro do tempo limite de 1 hora para as instâncias disponibilizadas, porém para instâncias menores foi possível testar e verificar os resultados obtidos pelo modelo, que novamente estavam corretos, evidenciando a formulação correta do programa inteiro referente ao problema.

7 Resultados obtidos com a aplicação da metaheurística GRASP

A seguir, serão apresentados os resultados obtidos com a implementação desenvolvida, com as instâncias disponibilizadas.

Tanto para os testes com vizinhança determinística quanto para os testes com vizinhança estocástica, utilizou-se apenas um início (segundo a construção gulosa-randomizada característica da metaheurística GRASP). Em cada início, foram realizadas duzentas construções aleatórias. Dentre as duzentas soluções iniciais aleatórias, apenas as dez melhores foram selecionadas como participantes do torneio de seleção randômico. Para cada teste, serão apresentados o valor das soluções inicial e final, o valor do desvio percentual entre a solução inicial (SI) e a solução final (SF), calculado como $(SI - SF) * 100/SI$. Para cada uma das tabelas, a linha correspondente ao melhor valor obtido para a solução final será destacada em verde.

7.1 Resultados com geração da vizinhança de forma determinística

Para os testes com vizinhança determinística, foram realizados dez testes para cada uma das dez instâncias, com cada teste correspondendo a um valor diferente para a seed da geração de valores randômicos.

Tabela 1 – Testes com a instância mdmt39.112.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3620	6049	-67,10	6m1s	6021
3608	6051	-67,71	6m34s	7200
3611	6102	-68,98	6m9s	8487
3570	6049	-69,44	6m14s	9692
3612	6055	-67,64	6m45s	916
3596	**6123**	**-70,27**	**8m58s**	**5051**
3639	6105	-67,77	9m22s	6821
3617	6085	-68,23	8m41s	8656
3605	6117	-69,68	8m37s	357
3569	5950	-66,71	8m43s	2046
Média aritmética das dez soluções finais: 6068.60				

Tabela 2 – Testes com a instância mdmt39.112.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3707	**5805**	**-56,60**	**6m15s**	**9787**
3710	5704	-53,75	6m09s	1012
3682	5741	-55,92	6m28s	2217
3700	5676	-53,41	5m46s	3484
3683	5767	-56,58	7m11s	4614
3702	5751	-55,35	8m22s	986
3758	5686	-51,30	7m14s	2625
3707	5783	-56,00	8m27s	4042
3753	5741	-52,97	8m24s	5698
3741	5645	-50,90	7m40s	7344
Média aritmética das dez soluções finais: 5729.90				

Tabela 3 – Testes com a instância mdmt39.225.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
2829	4118	-45,56	17m17s	4986
2848	4132	-45,08	18m21s	8373
2888	4129	-42,97	17m19s	1968
2879	4155	-44,32	16m43s	2593
2852	**4188**	**-46,84**	**19m59s**	**5872**
2834	4175	-47,32	21m57s	8846
2865	4150	-44,85	22m26s	3147
2852	4122	-44,53	22m8s	7542
2894	4196	-44,99	24m29s	1879
2866	4170	-45,50	24m16s	6676
Média aritmética das dez soluções finais: 4153.50				

Tabela 4 – Testes com a instância mdmt39.225.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
2910	**4326**	**-48,66**	**18m26s**	**8316**
2873	4262	-48,35	17m00s	1927
2886	4180	-44,84	17m03s	5258
2882	4138	-43,58	16m46s	8599
2867	4158	-45,03	15m50s	1884
2895	4230	-46,11	23m33s	1431
2927	4294	-46,70	23m37s	3277
2914	4083	-40,12	24m16s	7904
2871	4265	-48,55	23m10s	2659
2892	4224	-46,06	22m21s	7198
Média aritmética das dez soluções finais: 4216.00				

Tabela 5 – Testes com a instância mdmt40.56.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
4943	7928	-60,39	1m26s	6918
5114	7760	-51,74	1m18s	7199
4972	7691	-54,69	1m23s	7454
5059	7946	-57,07	1m28s	7728
5015	7902	-57,57	1m32s	8015
4944	7762	-57,00	1m35s	1577
4912	7882	-60,46	1m47s	1887
5043	**7979**	**-58,22**	**1m56s**	**2237**
5075	7896	-55,59	1m47s	2616
4998	7925	-58,56	2m15s	2965
Média aritmética das dez soluções finais: 7867.10				

Tabela 6 – Testes com a instância mdmt40.56.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
5040	8045	-59,62	1m23s	5566
4937	8121	-64,49	1m29s	5837
5001	**8165**	**-63,27**	**1m19s**	**6128**
5065	7960	-57,16	1m17s	6386
4968	8149	-64,03	1m25s	6640
4954	8021	-61,91	1m47s	3406
4969	8108	-63,17	1m44s	3755
5086	8129	-59,83	1m41s	4095
5077	8102	-59,58	1m49s	4425
4975	8149	-63,80	1m52s	4781
Média aritmética das dez soluções finais: 8094.90				

Tabela 7 – Testes com a instância mdmt40.112.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3989	6069	-52,14	5m57s	2600
3975	6072	-52,75	5m56s	997
3934	5934	-50,84	5m26s	2160
4017	6069	-51,08	5m58s	3225
4037	6072	-50,41	5m59s	4394
3959	**6102**	**-54,13**	**8m16s**	**5146**
3958	5975	-50,96	7min19s	6766
3966	5977	-50,71	8m10s	8200
4028	6010	-49,21	7min49s	9800
3973	6038	-51,98	8min49s	1331
Média aritmética das dez soluções finais: 6031.80				

Tabela 8 – Testes com a instância mdmt40.112.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3913	**6210**	**-58,70**	**7m27s**	**7056**
4016	6170	-53,64	5m21s	8503
3952	6100	-54,35	6m11s	9551
3984	6089	-52,84	5m25s	762
3958	6176	-56,04	6m39s	1824
3901	6180	-58,42	8m49s	291
4002	6047	-51,10	8m15s	2018
4010	6137	-53,04	9m27s	3635
3888	6116	-57,30	7m48s	5486
3929	6130	-56,02	7m37s	7015
Média aritmética das dez soluções finais: 6135.50				

Tabela 9 – Testes com a instância mdmt40.225.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3039	4395	-44,62	16m58s	6278
3097	4420	-42,72	18m12s	9603
3071	4350	-41,65	15m40s	3169
3017	4343	-43,95	16m31s	6238
3078	4333	-40,77	15m57s	9475
3017	4324	-43,32	21m55s	8507
3073	4359	-41,85	21m45s	2801
3026	4323	-42,86	22m1s	7063
3053	4406	-44,32	21m49s	1377
3055	**4441**	**-45,37**	**23m37s**	**5651**
Média aritmética das dez soluções finais: 4369.40				

Tabela 10 – Testes com a instância mdmt40.225.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3122	4431	-41,93	16m12s	359
3188	4434	-39,08	16m50s	3533
3087	4390	-42,21	16m27s	6831
3120	4440	-42,31	16m20s	54
3135	4453	-42,04	15m26s	3254
3117	**4488**	**-43,98**	**24m56s**	**279**
3138	4451	-41,84	23m28s	2396
3102	4442	-43,20	25m20s	6994
3097	4454	-43,82	24m01s	1958
3176	4437	-39,70	21m57s	6663
Média aritmética das dez soluções finais: 4442.00				

7.2 Resultados com geração de vizinhança de forma estocástica

Para os testes com vizinhança estocástica, foram realizados cinco testes para cada uma das dez instâncias, também com variação dos valores de seed, para cada um dos testes sobre uma determinada instância.

Tabela 11 – Testes com a instância mdmt39.112.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3712	4078	-9,86	17m46s	5626
3572	**4113**	**-15,15**	**15m09s**	**8545**
3589	4107	-14,43	15m01s	1513
3589	4068	-13,35	14m34s	1688
3687	4090	-10,93	14m34s	4542
Média aritmética das cinco soluções finais: 4091.20				

Tabela 12 – Testes com a instância mdmt39.112.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3709	**4174**	**-12,54**	**14m32s**	**7396**
3751	4143	-10,45	14m39s	244
3700	4109	-11,05	14m33s	3114
3770	4114	-9,12	14m36s	5965
3718	4165	-12,02	14m35s	8825
Média aritmética das cinco soluções finais: 4141.00				

Tabela 13 – Testes com a instância mdmt39.225.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
2940	**3141**	**-6,84**	**42m33s**	**1683**
2899	3132	-8,04	42m23s	20
2868	3124	-8,93	42m18s	5556
2860	3109	-8,71	42m45s	2860
2967	3119	-5,12	42m33s	2967
Média aritmética das cinco soluções finais: 3125.00				

Tabela 14 – Testes com a instância mdmt39.225.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
2873	**3175**	**-10,51**	**42m17s**	**558**
2880	3152	-9,44	42m33s	6074
2904	3146	-8,33	42m39s	4411
2869	3133	-9,20	42m41s	2768
2865	3132	-9,32	42m57s	1131
Média aritmética das cinco soluções finais: 3147.60				

Tabela 15 – Testes com a instância mdmt40.56.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
4984	5664	-13,64	5m53s	6778
4954	5715	-15,36	5m51s	7931
5153	5715	-10,91	5m53s	9077
4997	**5735**	**-14,77**	**5m53s**	**230**
5024	5727	-13,99	5m53s	1383
Média aritmética das cinco soluções finais: 5711.20				

Tabela 16 – Testes com a instância mdmt40.56.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
5001	5718	-14,34	5m54s	2536
4970	**5791**	**-16,52**	**5m53s**	**3692**
4953	5624	-13,55	5m54s	4844
4960	5649	-13,89	5m52s	6000
5050	5663	-12,14	5m52s	7150
Média aritmética das cinco soluções finais: 5689.00				

Tabela 17 – Testes com a instância mdmt.40.112.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3944	4452	-12,88	14m35s	8299
3970	4414	-11,18	14m35s	1157
3969	4413	-11,19	14m34s	4014
3966	4454	-12,30	14m32s	6868
3916	**4457**	**-13,82**	**14m33s**	**9716**
Média aritmética das cinco soluções finais: 4438.00				

Tabela 18 – Testes com a instância mdmt.40.112.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3961	4418	-11,54	14m33s	2567
3949	**4444**	**-12,53**	**14m32s**	**2650**
3948	4409	-11,68	14m28s	5497
3912	4384	-12,07	14m31s	8332
3925	4369	-11,31	14m35s	1176
Média aritmética das cinco soluções finais: 4404.80				

Tabela 19 – Testes com a instância mdmt40.225.A				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3133	3345	-6,77	42m33s	4033
3047	3331	-9,32	42m31s	2370
3028	**3362**	**-11,03**	**42m39s**	**701**
3043	3306	-8,64	42m25s	6290
3067	3336	-8,77	42m31s	4600
Média aritmética das cinco soluções finais: 3336.00				

Tabela 20 – Testes com a instância mdmt40.225.B				
Solução Inicial	**Solução Final**	**Desvio Percentual (SI e SF)**	**Tempo Computacional Metaheurística**	**Seed**
3110	3398	-9,26	42m45s	2931
3129	3409	-8,95	42m41s	1307
3103	3416	-10,09	42m28s	6902
3159	**3431**	**-8,61**	**42m34s**	**5223**
3223	3426	-6,30	42m42s	3563
Média aritmética das cinco soluções finais: 3416.00				

8 Comparações dos resultados obtidos com vizinhança determinística e com vizinhança estocástica

Na tabela 21, visualiza-se, para cada uma das dez instâncias, os melhores valores obtidos com a aplicação da metaheurística, para as vizinhanças determinística (VD) e estocástica (VE), junto com o desvio percentual entre cada um dos valores.

Tabela 21 – Comparações no Modo de Geração de Vizinhanças

Instância	Melhor Valor com Vizinhança Determinística	Melhor Valor com Vizinhança Estocástica	Desvio Percentual (VD e VE)
mdmt39.112.A	6123	4113	32,83
mdmt39.112.B	5805	4174	28,10
mdmt39.225.A	4196	3141	25,14
mdmt39.225.B	4326	3175	26,61
mdmt40.56.A	7979	5735	28,12
mdmt40.56.B	8165	5791	29,08
mdmt40.112.A	6102	4457	26,96
mdmt40.112.B	6210	4444	28,44
mdmt40.225.A	4441	3362	24,30
mdmt40.225.B	4488	3431	23,55

9 Comparações envolvendo os melhores valores conhecidos até então

Na tabela 22, são apresentadas comparações envolvendo os melhores valores obtidos com a aplicação da metaheurística (apresentados nas tabelas anteriores, denotados por VM) com os melhores valores conhecidos até então para cada uma das instâncias (*best known values*, BKV). Nos quatro casos em que obtivemos valores que superam os melhores valores conhecidos (com desvio percentual positivo), a linha correspondente da tabela está destacada em verde.

Tabela 22 - Comparações com Melhores Valores Conhecidos

Instância	Melhor Valor com Metaheurística (VM)	Melhor Valor Conhecido (BKV)	Desvio Percentual (VM e BKV)
mdmt39.112.A	**6123**	**5935**	**3,07**
mdmt39.112.B	5805	6198	-6,77
mdmt39.225.A	4196	4654	-10,92
mdmt39.225.B	**4326**	**4260**	**1,53**
mdmt40.56.A	7979	8211	-2,91
mdmt40.56.B	**8165**	**8022**	**1,75**
mdmt40.112.A	6102	6271	-2,77
mdmt40.112.B	**6210**	**6198**	**0,19**
mdmt40.225.A	4441	4550	-2,45
mdmt40.225.B	4488	4492	-0,09

10 Conclusões sobre os resultados obtidos

Tanto na realização das buscas com vizinhança determinística quanto nas buscas com a vizinhança estocástica, é possível notar que, mesmo nos casos em que a busca local foi iniciada a partir de uma solução inicial que não corresponde à melhor (dentre aquelas que foram geradas para a respectiva instância), é possível obter resultados mais interessantes. Isso pode ser verificado, por exemplo, analisando os dados da tabela 1, no qual a melhor solução inicial gerada corresponde à uma função objetivo de valor 3620, entretanto, a melhor solução obtida para a instância da tabela foi encontrada a partir de uma busca local iniciada em uma solução inicial com valor igual a 3596, com a mesma situação ocorrendo nas outras instâncias analisadas. Assim, pode-se comprovar a utilidade da etapa da construção gulosa-randomizada, característica da metaheurística GRASP, que viabiliza a exploração de determinados espaços de busca mais benéficos para a maximização do valor da função objetivo.

Sobre a comparação entre os processos de busca com as vizinhanças determinística e estocástica, a realização dos experimentos evidenciou, de forma bastante segura e predominante, a superioridade da vizinhança determinística em relação à estocástica, no processo de obtenção de valores maiores para a função objetivo (conforme evidenciado na seção 8, tabela 21). O processo de geração da vizinhança de forma determinística mostrou-se capaz de revelar soluções mais interessantes, principalmente em razão do fato de que esse processo gera uma vizinhança de forma mais completa em relação à vizinhança estocástica (isto é, com mais opções de transições durante um passo da busca local), permitindo verificar os eventuais benefícios para a função objetivo mediante todas as permutações individuais de vértices selecionados com vértices não-selecionados (conforme explicado na seção 5.7). Assim, comprova-se a importância da geração de uma vizinhança logicamente coerente quando das aplicações de buscas locais com metaheurísticas.

Em relação aos valores dos máximos locais obtidos em cada experimento, na seção 7.1 foi possível verificar resultados bastante interessantes, levando em consideração as comparações com os melhores valores até então conhecidos para as instâncias, (observadas na seção 9, tabela 22), uma vez que foi possível obter valores melhores do que aqueles já conhecidos, para quatro das dez instâncias. Mesmo nos casos em que não foi possível superar os melhores valores conhecidos, obtivemos valores de desvio percentual relativamente pequenos, indicando a eficácia dos experimentos realizados. Assim, a realização de mais experimentos, com diferentes seeds e parâmetros de entrada, muito provavelmente possui capacidade de gerar resultados ainda melhores do que os obtidos.

A partir das conclusões apontadas, considera-se que o trabalho foi bem-sucedido.

Referências

[1] Blum C.; Roli A. (2003) **Metaheuristics in combinatorial optimization: Overview and conceptual comparison.** ACM Comput. Surv., Vol 35, N. 3, September , pp. 268-308.

[2] Resende M.; Ribeiro C. (2003) **Greedy Randomized Adaptive Search Procedures: Advances and Extensions.** pp 1-2.