INF05010 Otimização Combinatória
Algoritmo Memético para o Problema de Múltiplos Contêineres

Marco Antônio Chitolina da Silva - 00308226

1 Introdução

O objetivo deste trabalho é implementar a meta-heurística de algoritmo memético para resolver o Problema dos Múltiplos Contêineres (1.1).

1.1. Problema dos Múltiplos Contêineres (PCMCDC):

Dados n itens e m contêineres, queremos distribuir uma parcela destes itens nos contêiners, visando maximizar o valor acumulado ao passo que respeitamos a capacidade máxima de cada um dos contêineres, i.e., não alocando mais peso do que o contêiner suporta. Cada contêiner k tem uma capacidade Ck, e cada item i ocupa um volume Vi. Além disso, cada item tem um valor vi. Por fim, cada dupla de itens (i, j) presentes no mesmo contêiner tem uma valor extra vij.

2 Formulação do Programa Inteiro

Função objetivo (0): valor a ser maximizado consiste de duas componentes: valor individual de cada um dos itens selecionados (valorItens (1)) e valor das duplas de itens atribuídos ao mesmo contêiner (valorPares (2)).
valorItens (1): todos os itens selecionados contribuem para a função objetivo com o seu valor.
valorPares (2): todas as duplas de itens presentes no mesmo contêiner contribuem para a função objetivo com o seu valor extra.
(3), (4): restrições adicionais provenientes da variável and(i, j, k) introduzida na restrição (2) para verificar se dois itens estão no mesmo contêiner.
(5): não posso passar o peso máximo dos contêineres ao tentar atribuir itens para eles.
(6): só posso atribuir um item específico a no máximo um contêiner.
(7): restrição de cardinalidade da variável de atribuição.
(8): restrição de cardinalidade da variável que verificar se dois itens estão ou não presentes no mesmo contêiner.

Como (3) e (4) expandem para $m*n^2$ restrições; (5), para m restrições, (6), para n restrições; (7), para $m*n$ restrições; (8), para $m*n^2$ restrições, temos um total de $2*m*n^2 + m + n$ restrições ($O(m*n^2)$ restrições).

Formulação Matemática:

$$aik = \begin{cases} 1 & , \text{se o item } i \text{ foi atribuído ao contêiner } k. \\ 0 & , \text{caso contrário.} \end{cases}$$

n = número de itens,
m = número de contêineres,
ci = volume (capacidade) do item i,
Ck = volume (capacidade) do contêiner k,
vi = valor do item i,
vij = valor do par de itens (i,j) no mesmo contêiner,

max valorItens + valorPares (0)
s.a.

$$valorItens = \sum_{\substack{i \in [n], \\ k \in [m]}} a_{ik} * v_i \quad (1)$$

$$valorPares = \sum_{\substack{i \in [n], \\ j \in [n] \\ k \in [m]}} and_{ijk} * v_{ij} \quad (2)$$

$$2 * and_{ijk} \leqslant a_{ik} + a_{jk}, \ \forall i \in [n], \ j \in [n], \ k \in [m] \quad (3)$$

$$a_{ik} + a_{jk} \leqslant and_{ijK}, \ \forall i \in [n], \ j \in [n], \ k \in [m] \quad (4)$$

$$\sum_{i \in [n]} a_{ik} * c_i \leqslant C_k, \ \forall_k \in [m] \quad (5)$$

$$\sum_{k \in [m]} a_{ik} \leqslant 1, \ \forall_i \in [n] \quad (6)$$

$$a_{ik} \in \{0, 1\}, \ \forall i \in [n], \ k \in [m] \quad (7)$$

$$and_{ijk} \in \{0, 1\}, \ \forall i \in [n], \ j \in [n], \ k \in [m] \quad (8)$$

3 Conceitos de genética

Como algoritmos genéticos são baseados na teoria de evolução das espécies, alguns conceitos da biologia são associados com eles.

- Indivíduo: solução. Uma solução é um conjunto de atribuições de itens a determinados contêineres (por exemplo, {aik, ai(k+1), ai(k+2)})
- Gene: variável de uma solução. aik (o item i foi atribuído ao contêiner k).
- Alelo: valor de uma variável. aik = 1 (caso tenha sido atribuído) ou aik = 0 (caso contrário).
- Cross-over: recombinação (troca de material genético entre dois indivíduos). Trocar todo conteúdo do contêiner de um indivíduo com o respectivo contêiner do outro indivíduo.

- Mutação: perturbação. Alterar as atribuições a determinado contêiner, podendo atribuir novos itens ou “desatribuir” itens previamente selecionados.
- Descendentes: soluções combinadas. Ir renovando a população original.
- População: conjunto de soluções.
- Aptidão: função objetivo. Soma dos valores dos itens, junto com a soma dos valores dos pares de itens.

4 Parâmetros

- num_solucoes_populacao_original (natural positivo): tamanho da população original. Escolhi este parâmetro para variar e comparar os resultados.
- num_participantes_torneio (natural positivo, menor ou igual ao num_solucoes_populacao_original): número de participantes do torneio. Fixado como sendo 10% num_solucoes_populacao_original.
- num_particao (natural, menor ou igual ao num_solucoes_populacao_original): quantos contêineres serão trocados durante a operação de cross-over. Escolhi deixar o valor desse parâmetro sendo definido aleatoriamente.
- alpha (real, entre 0 e 1, cuja soma com beta deve resultar em 1): porcentagem de quantos descendentes serão da população original. Fixado como sendo 10% (0.1)
- beta (real, entre 0 e 1, cuja soma com apha deve resultar em 1): porcentagem de quantos descendentes serão da nova população. Fixado como sendo 90% (0.9).

5 Algoritmo memético

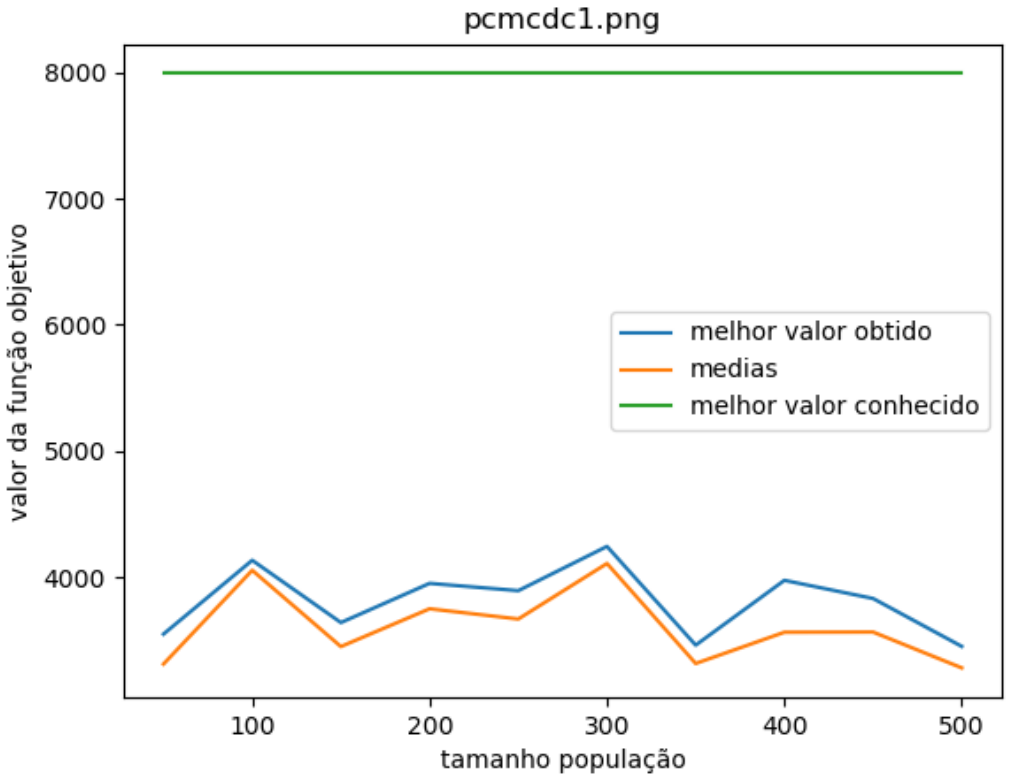
```
1 Heurística:
2 P = gera_população()
3 while not critério_parada_satisfeito():
4  Q = {}
5  while |Q| < |P|:
6   s1 = torneio(P)
7   s2 = torneio(P)
8   recombinação(s1, s2)
9   mutação(s1)
10  mutação(s2)
11  busca_local(s1)
12  busca_local(s2)
13  Q = Q U {s1, s2}
14 P = seleção(P, Q)
15 return melhores_indivíduos(P)
16
```

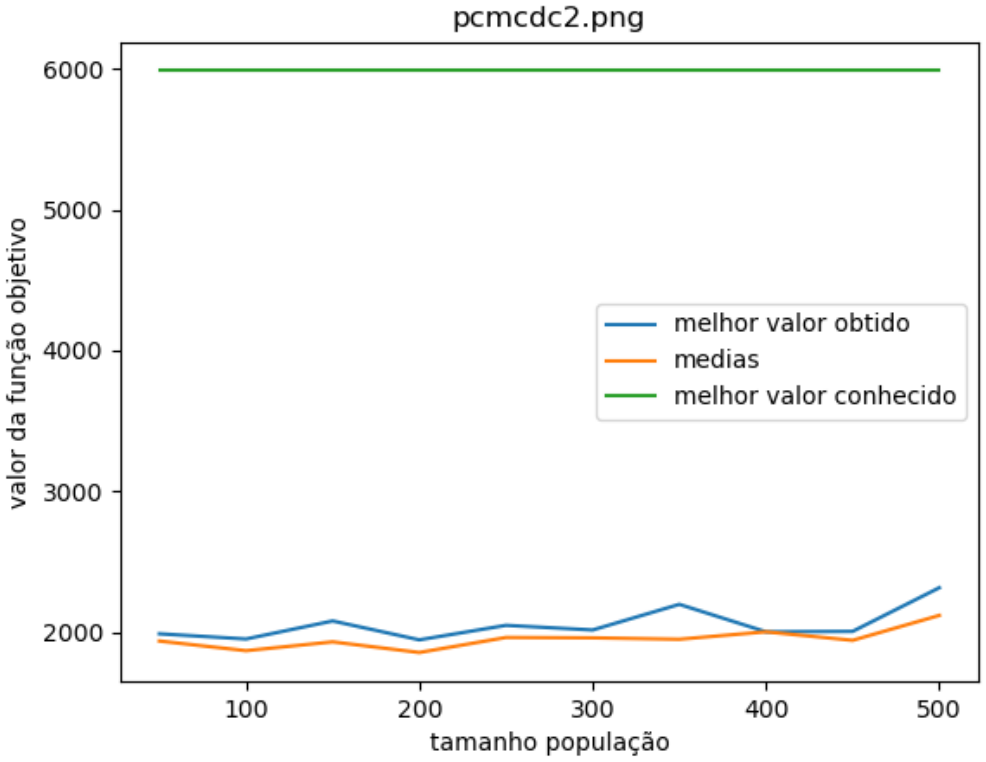
- gera_população(): cria aleatoriamente uma população de indivíduos (itens são aleatoriamente alocados para contêineres).

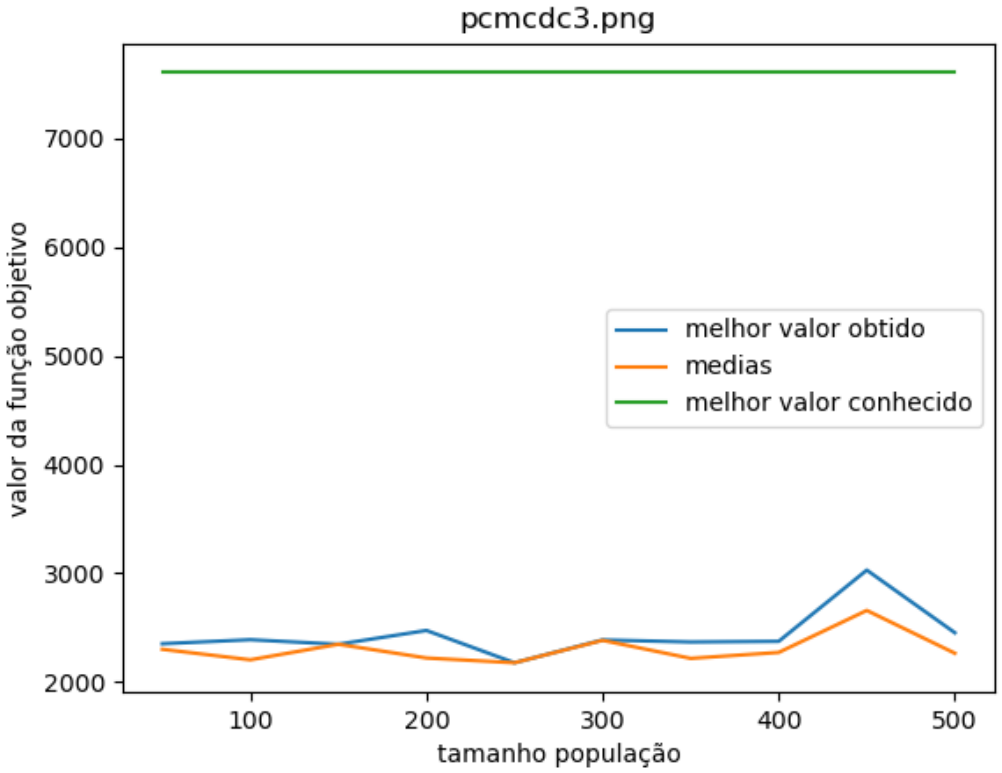
- critério_parada_satisfeito(): escolhi o tempo como critério de parada. Cada execução levou 1 hora para ser concluída. Para cada instância do problema, executei o algoritmo 10 vezes, alterando somente o num_solucoes_populacao_original. Ele, para cada instância, começou com o valor 50 (50 indivíduos na população original) e foi aumentando de 50 em 50 indivíduos.
- torneio(P): seleciona aleatoriamente num_participantes_torneio indivíduos da população P a fim de competirem mortalmente num torneio que pode haver somente um vencedor: o mais apto (i.e., o indivíduo cuja função objetivo apresenta o maior valor).
- recombinação(s1, s2): troca num_particao contêineres da solução s1 com a solução s2 (ou seja, todos os itens atribuídos aos contêiners que foram selecionados passam de uma solução para a outra).
- mutação(s): troca as atribuições dos itens de um contêiner, podendo atribuir para esse contêiner um item que não estava previamente selecionado, podendo também "desatribuir" um item que estava previamente atribuído (pode ser útil nos casos de um item muito pesado e pouco valioso estar ocupando o espaço de outros itens melhores).
- busca_local(s): melhora o valor da solução s até chegar num mínimo local. Escolhi implementar da seguinte maneira: à medida que eu passo pelos contêineres, verifico se eles ainda têm capacidade de armazenar mais itens. Se possuem, atribuo o primeiro item que aparece. Paro quando não consigo mais atribuir itens aos contêineres, atingindo, assim, um mínimo local.
- seleção(P, Q): seleciono os alpha melhores indivíduos da população original (P) e beta melhores indivíduos da nova população (Q).
- melhores_indivíduos(P): retorna os indivíduos mais aptos da população P.

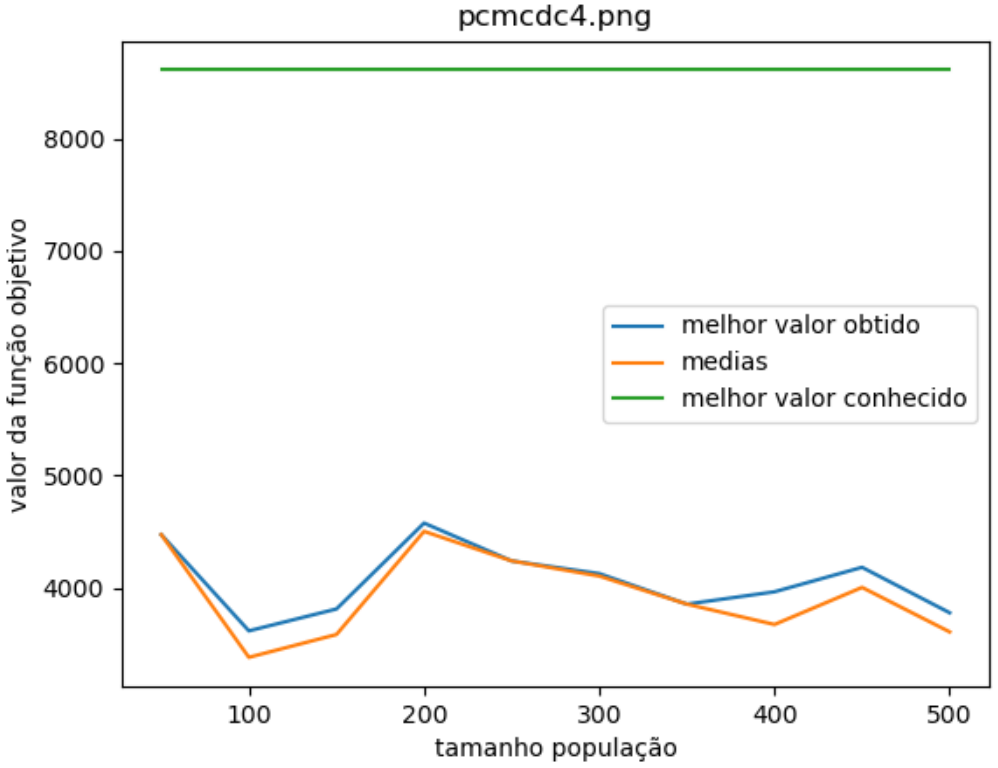
6 Resultados obtidos

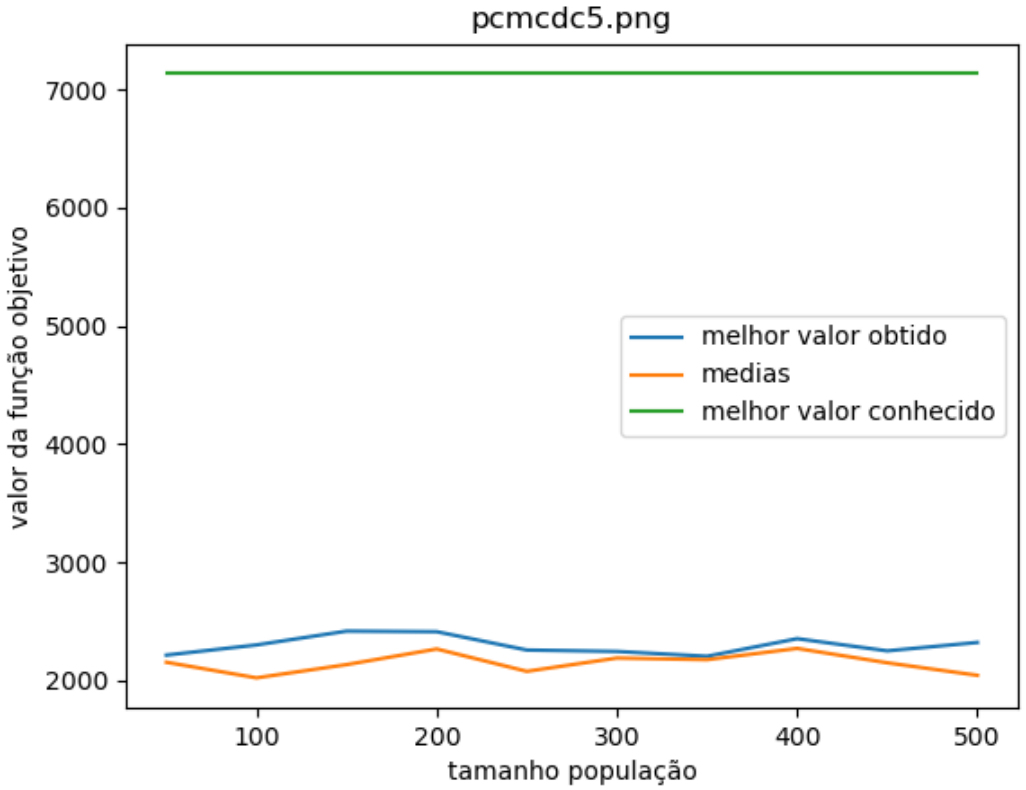
Primeiramente, irei analisar os resultados obtidos por rodar a heurística do algoritmo memético para as 10 instâncias de PCMCDC disponíveis nesta página. Vale salientar que os resultados obtidos e discutidos aqui foram obtidos ao rodar a heurística memética, utilizando este computador. Para cada uma das instâncias, o número de indivíduos da população começa em 50, e, para cada execução, vai subindo de 50 em 50. Como cada instância foi executada 10 vezes, o número de indivíduos na população varia de 50 até 500. As curvas em verde representam o melhor valor conhecido para tal instância. As curvas em azul representam a média dos valores da última população em execução. As curvas em laranja representam o maior valor da última população em execução.

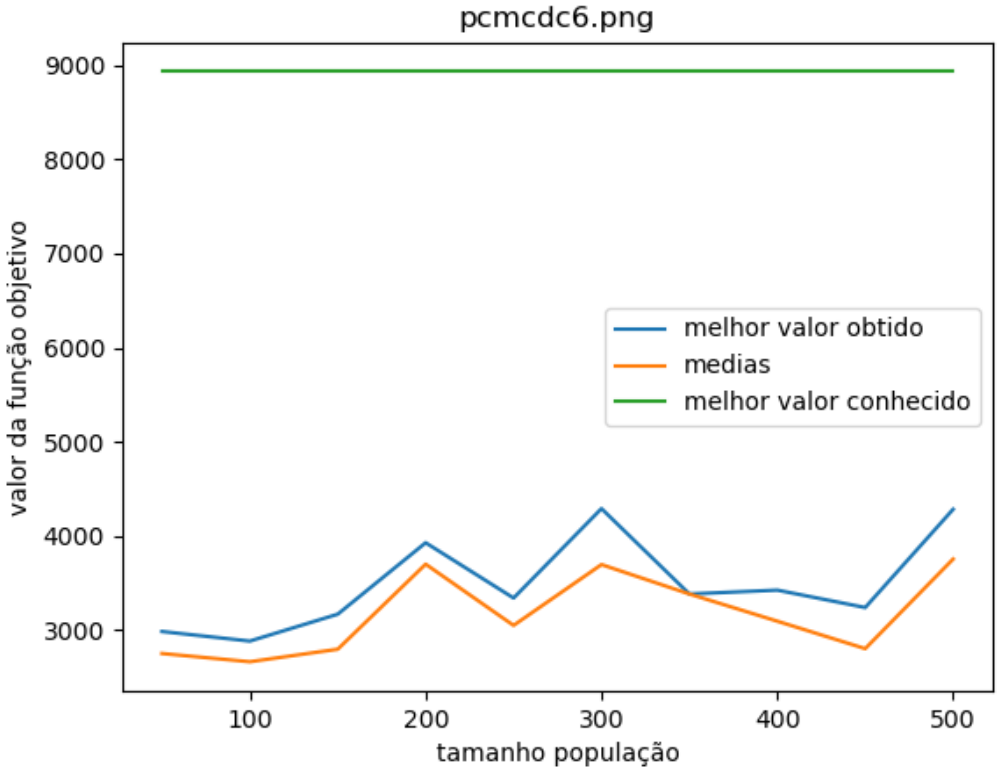

pcmcdc1.png
8000
7000
6000
5000
4000
valor da função objetivo
melhor valor obtido
medias
melhor valor conhecido
100
200
300
400
500
tamanho população

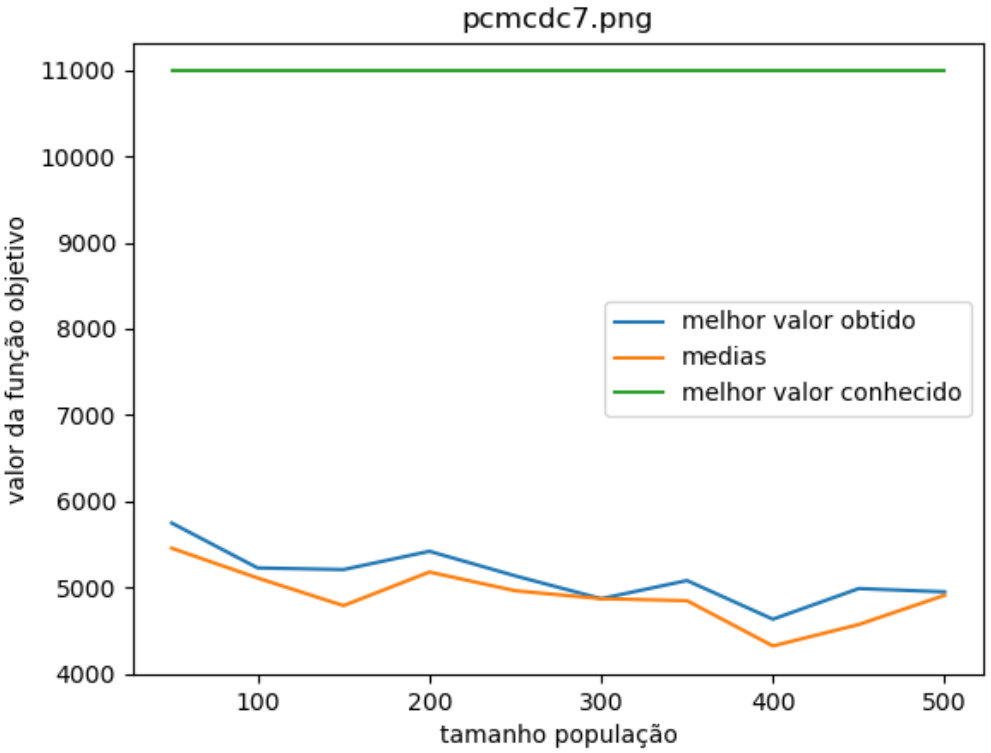

pcmcdc2.png
6000
5000
4000
3000
2000
valor da função objetivo
melhor valor obtido
medias
melhor valor conhecido
100
200
300
400
500
tamanho população

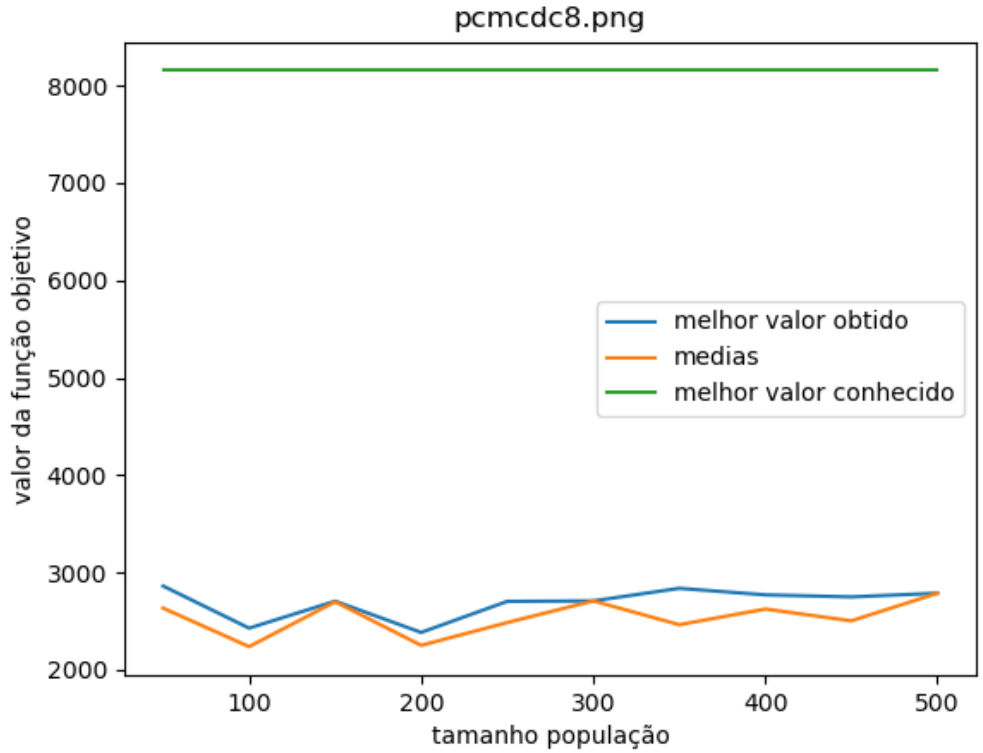

pcmcdc3.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
2000
3000
4000
5000
6000
7000
100
200
300
400
500

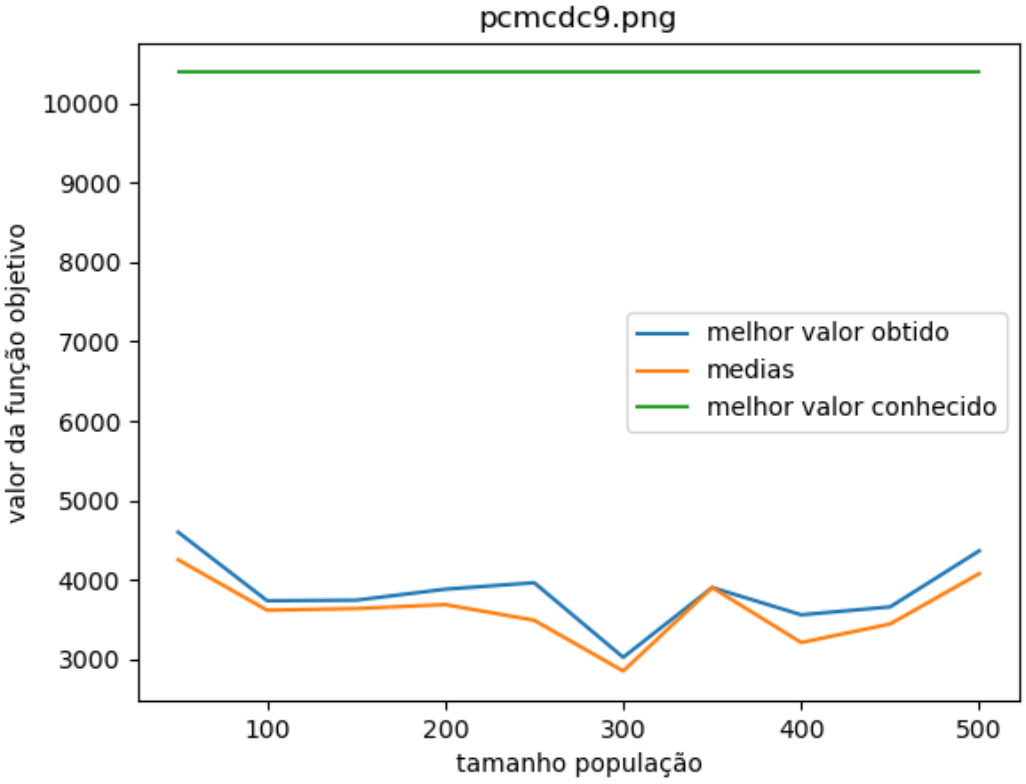

pcmcdc4.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
4000
5000
6000
7000
8000
100
200
300
400
500

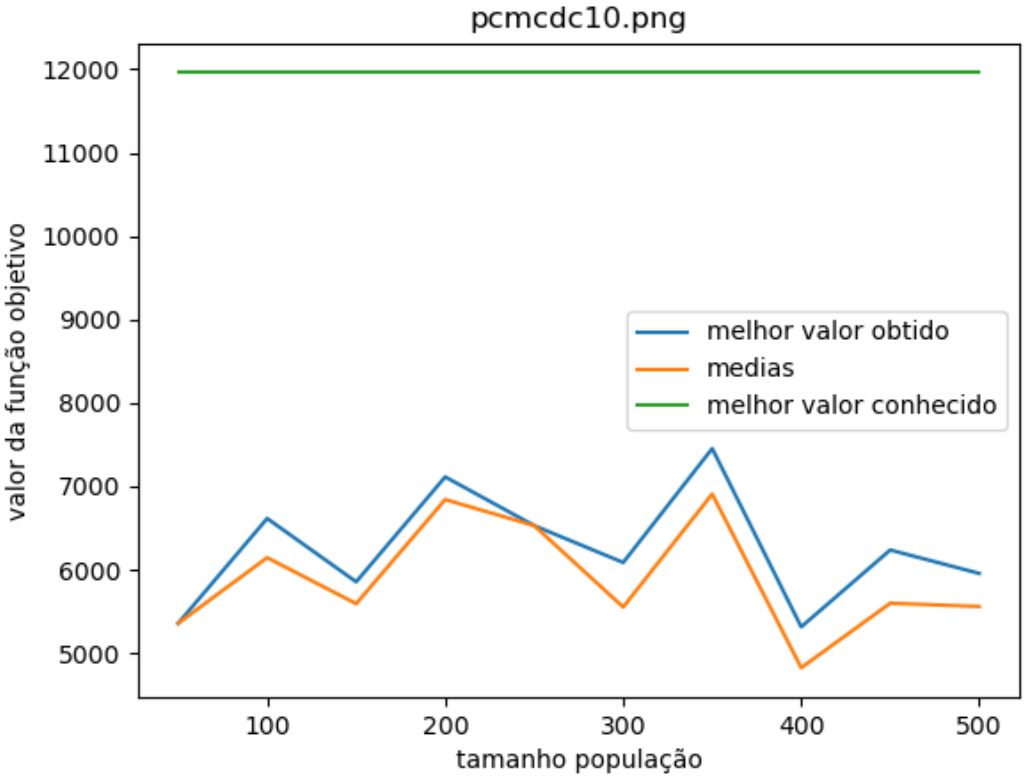

pcmcdc5.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
7000
6000
5000
4000
3000
2000
100
200
300
400
500


pcmcdc6.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
9000
8000
7000
6000
5000
4000
3000
100
200
300
400
500


pcmcdc7.png
11000
10000
9000
8000
7000
6000
5000
4000
valor da função objetivo
melhor valor obtido
medias
melhor valor conhecido
100
200
300
400
500
tamanho população


pcmcdc8.png
8000
7000
6000
5000
4000
3000
2000
valor da função objetivo
melhor valor obtido
medias
melhor valor conhecido
100
200
300
400
500
tamanho população


pcmcdc9.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
10000
9000
8000
7000
6000
5000
4000
3000
100
200
300
400
500


pcmcdc10.png
valor da função objetivo
tamanho população
melhor valor obtido
medias
melhor valor conhecido
12000
11000
10000
9000
8000
7000
6000
5000
100
200
300
400
500

Além dos resultados obtidos pela heurística, há também os resultados obtidos por rodar a formulação matemática, implementado com pulp como “Front End” e com o solver CPLEX como “Back End”, utilizando: este computador. Os resultados foram agrupados na tabela abaixo.

Inst.	01	02	03	04	05	06	07	08	09	10
BKV	7992	5985	7604	8610	7132	8935	10984	8154	10385	11958
O.F.M	6649	5585	6544	7222	6380	7922	8734	6972	8905	10095
D.P. F.M.	0,1680	0,0668	0,1394	0,1612	0,1054	0,1133	0,2048	0,1449	0,1425	0,1557
M.O.H	4243	2315	3030	4575	2419	4288	5750	2861	4600	7454
D.P. H.	0.46	0.61	0.6	0.46	0.66	0.52	0.47	0.64	0.55	0.37

Tabela 01: Instância (Inst.). Melhor valor conhecido (BKV). Valor obtido pela formulação matemática (O.F.M.). Diferença percentual da formulação matemática (D.P.F.M). Melhor valor obtido pela heurística (M.O.H). Diferença percentual da heurística (D.P.H).

7 Conclusão

Apesar do resultado da formulação matemática obtido pelo solver CPLEX ter se saído relativamente bem, atingindo até 93% de otimalidade com apenas 4 horas de execução, os resultados obtidos pela heurística deixam bastante a desejar, com o melhor sendo de apenas 63% de otimalidade. Acredito, no entanto, que os valores de baixa otimalidade na heurística não sejam problemas relacionados ao código, mas sim a dois outros fatores: tempo de execução muito baixo e computador muito lento para rodar a heurística.

Para averiguar se minha hipótese está correta, seria necessário testar por mais tempo (talvez 4 horas por execução ao invés de 1 hora), utilizando um computador mais rápido. Caso os resultados se mantivessem com níveis baixos de otimalidade, poderíamos rejeitar minha hipótese, e o problema seria intrínseco da minha implementação do algoritmo.

Apesar de não ter obtido bons resultados com minha implementação da heurística, eu diria que o trabalho foi bem-sucedido, em função de aprender com os erros dos experimentos.

8 Referências

https://www.inf.ufrgs.br/~mrpritt/lib/exe/fetch.php?media=inf05010:notas-11942.pdf