Trabalho Computacional
Problema das p-Medianas

Gilson Urbano Ferreira Dias - 2015430959
Moises Mendes de Assis - 2014015524
guf@ufmg.br, moisesmendes@ufmg.br

Junho de 2017

1 Introdução

Uma importante linha de pesquisa envolve as aplicações dos problemas de localização de facilidades. Estes problemas tratam a questão de onde localizar um objeto que é chamado facilidade. A facilidade deverá interagir com outros objetos que possuirão localizações fixas. Em um problema de localização deseja-se estabelecer os locais onde serão sediadas facilidades (fábricas, depósitos, hospitais, escolas, etc.) para atender, da melhor maneira possível, um conjunto espacialmente distribuído de pontos de demanda.

O problema de p-medianas é um problema clássico de localização. O objetivo é determinar os locais de p facilidades (denominadas medianas) em uma rede de n nós de modo a minimizar a soma das distâncias entre cada nó de demanda e a mediana mais próxima.

O problema a ser tratado envolve a localização de p facilidades e a designação de clientes a facilidades de modo a minimizar a soma das distâncias de clientes a facilidades e tal que cada cliente seja atendido por uma única facilidade.

2 Solução do Problema

Antes de realizar a implementação definimos a notação a seguir:

Dado um grafo completo e não orientado $G = (V; E)$, em que V é o conjunto de n vértices e E é o conjunto de arestas representando as distâncias entre os vértices, o objetivo é encontrar um subconjunto $V_p \in V$ com cardinalidade p, em que V_p representa o conjunto de medianas do problema, de modo que a soma das distâncias entre os vértices restantes em $\{V - V_p\}$ é o vértice mais próximo em V_p seja a menor possível.

Considerando que:

- n = representa o número de vértices;
- p = representa o número de medianas que serão instaladas;
- d_{ij} = representa a distância entre os vértices i e j;

A formulação matemática do Problema das p-medianas é dada por

$$\min \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ij} \tag{1}$$

$$\text{sa. } \sum_{j=1}^{N} x_{ij} = 1 \; \forall \; i \in \{V - V_p\} \tag{2}$$

$$x_{ij} \leq y_j \; \forall \; i \in \{V - V_p\} \text{ e } j \in V_p \tag{3}$$

$$\sum_{j=1}^{N} y_j = p \; \forall \; j \in V_p \tag{4}$$

$$x_{ij}, y_j \in \{0, 1\} \tag{5}$$

em que:

$$x_{ij} = \begin{cases} 1, & \text{se o } i\text{-ésimo vértice é atendido pela mediana (vértice) } j; \\ 0, & \text{caso contrário} \end{cases} \tag{6}$$

$$y_j = \begin{cases} 1, & \text{se o } j\text{-ésimo vértice for uma das } p\text{-medianas}; \\ 0, & \text{caso contrário} \end{cases} \tag{7}$$

A restrição (2) assegura que um vértice (cliente) seja atendido por uma única mediana (facilidade). A restrição (3) afirma que um vértice (cliente) somente seja atendido por uma mediana que esteja instalada. A restrição (4) garante que são designadas exatamente p medianas. A restrição (5) define que as variáveis de decisão envolvidas são binárias.

2.1 Funções e Procedimentos

As funções foram declaradas da seguinte forma:

- **leitura()**: Função que efetua a leitura do arquivo de entrada. A saída é um array de células, em que cada linha contém as coordenas (x, y) de um vértice, exceto a primeira que contém os valores de n e p.
- **fitness()**: Essa função calcula a aptidão de uma solução candidata. Para isso, cada vértice é associado à facilidade mais próxima de acordo com a distância euclidiana entre eles. Retorna-se a distância total, que é mais utilizada durante a execução do algoritmo, mas também retorna-se vetores com a ordem dos vértices, a mediana associada a cada um e a distância de cada vértice à sua mediana.
- **roleta()**: Utiliza a estratégia da roleta para definir os pais que se cruzarão para gerar uma nova população. Os indivíduos são mapeados em uma em vetor de segmentos no intervalo $[0, 1]$, sendo que o tamanho de cada segmento é proporcional a probabilidade de seleção do indivíduo. Então é selecionado um número aleatório e esse número é buscado até ser encontrado em um dos segmentos do intervalo, sendo esse processo análogo a girar uma roleta com um único ponto de seleção. Roda-se a Roleta λ vezes para selecionar λ indivíduos que serão utilizados no cruzamento para geração de uma nova população.

- **cross()**: Para o cruzamento aplica-se uma máscara de bits, conforme exemplo da Tabela 2.1.

0	1	1	0	1

Tabela 1: Máscara de Bits

46	152	173	216	257
58	154	199	234	275
0	1	1	0	1
58	152	173	234	257
46	154	199	216	275

Tabela 2: Resultado do cruzamento

Os filhos então recebem as características conforme o modelo a seguir. A primeira e a segunda linha são os pais, na terceira linha temos a máscara de bits 2.1, e na quarta linha e quinta linha são os filhos. Eles são gerados de acordo a máscara: caso 0 na máscara, o primeiro filho recebe a informação do segundo pai, e caso 1, recebe informação do primeiro pai; no segundo filho, caso 1 na máscara o primeiro filho recebe a informação do segundo pai e caso 0, recebe informação do primeiro pai.

- **mutacao()**: A mutação é feita através da troca de um valor do vetor coluna, verificando se o vértice aleatório escolhido já não existe nessa solução candidata.

2.2 Programa Principal

Na função principal **pmedianas**, são declaradas e inicializadas as variáveis que controlam toda a execução, **filename**, **npop**, **crosProb**, **crossMut**, **tmax**, sendo o nome do arquivo de entrada, tamanho da população, probabilidade de cruzamento, probabilidade de mutação, e o número máximo de interações. Existe um caso *default*, mas essas variáveis podem ser passadas como parâmetros de entrada.

A estratégia evolucionária implementada para solucionar o problema das p-medianas proposto segue o pseudocódigo a seguir:

Algoritmo 1: Algoritmo Evolucionário

```
t ← 1
Inicializar população P_t = {n_{t,i}; i = 1, ..., p}
para i = 1 até npop faça
    Calcula fitness
fim
enquanto algum critério de parada não for satisfeito faça
    Seleciona pais Roleta
    para i = 1 até npop faça
        Cruzamento
        Mutação
    fim
    Avaliação μ + λ
    Seleção dos mais aptos
    t ← t + 1
fim
```

3 Avaliação Experimental

3.1 Testes

Vários testes foram realizados com o programa de forma a verificar seu funcionamento. Os testes foram realizados em um Intel Core i5-4210U rodando a 1.70GHz, 6Gb de memória ram e sistema operacional *Windows 10*.

3.1.1 Teste 1

O primeiro teste foi realizado com os seguintes parâmetros: Tamanho da população ($npop = 30$), probabilidade de cruzamento ($crossProb = 1$) e probabilidade de mutação ($crossMut = 0.9$). Para esse teste utilizamos um conjunto espacialmente distribuído de pontos de demanda de tamanho $n = 324$ e desejamos estabelecer $p = 5$ locais onde serão sediadas facilidades (medianas).

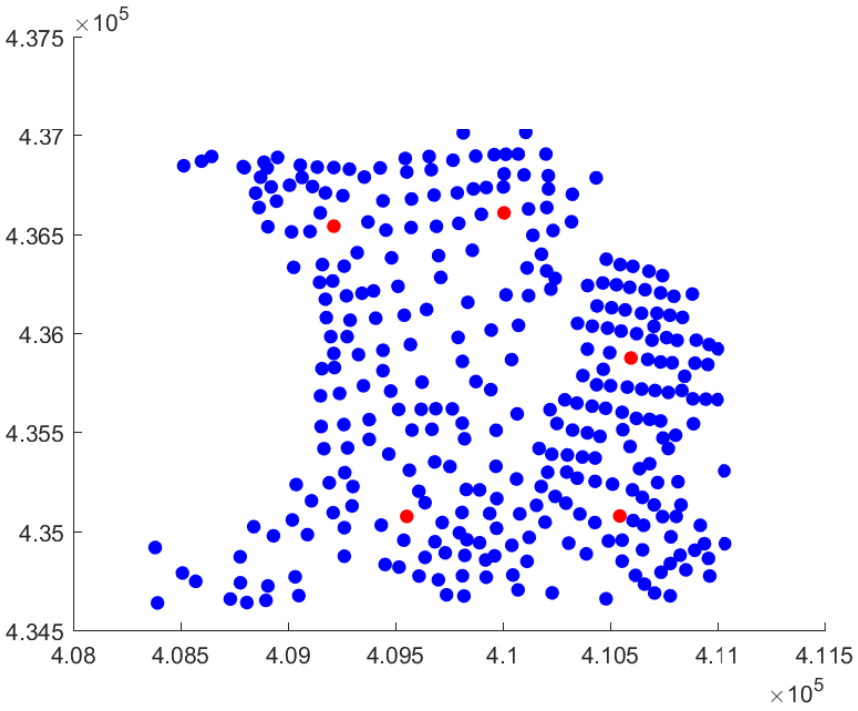

Figura 1: Vértices e Medianas - Teste 1

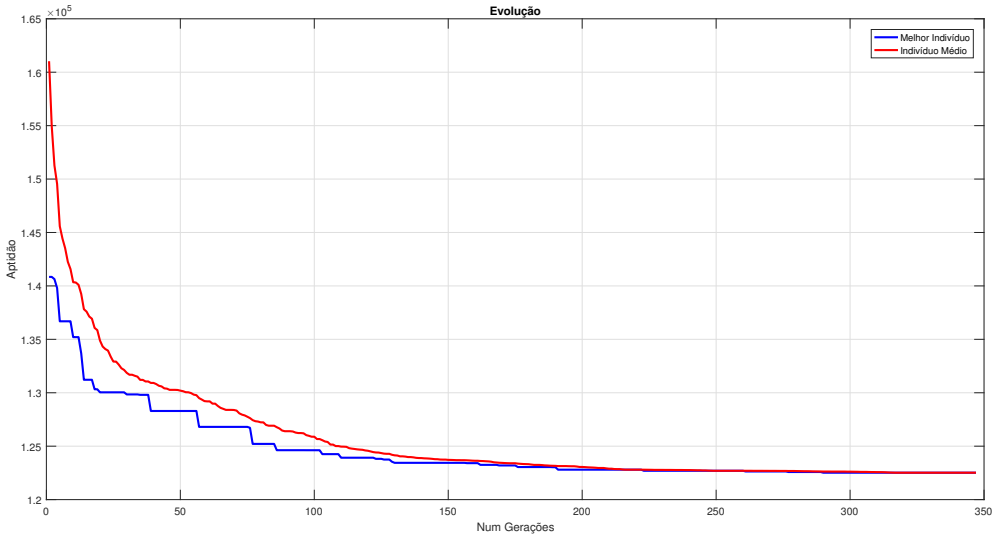

Figura 2: Convergência - Teste 1

O resultado após pouco mais de 300 interações é um $fitness = 1.2252 \times 10^5$, com convergência monotônica como mostrado na Figura 2. O melhor é indivíduo nunca piora assim como o restante da população, o que é possível graças a estratégia $\mu + \lambda$ implementada.

3.1.2 Teste 2

No segundo repetimos os parâmetros inicias do teste anterior, e aumentamos o número de pontos de demanda para $n = 818$ e encontramos um valor de $fitness = 6.0603 \times 10^5$ após 300 gerações do algoritmo, um pouco menos do que o teste anterior e novamente constatamos convergência monotônica como já havíamos esperado.

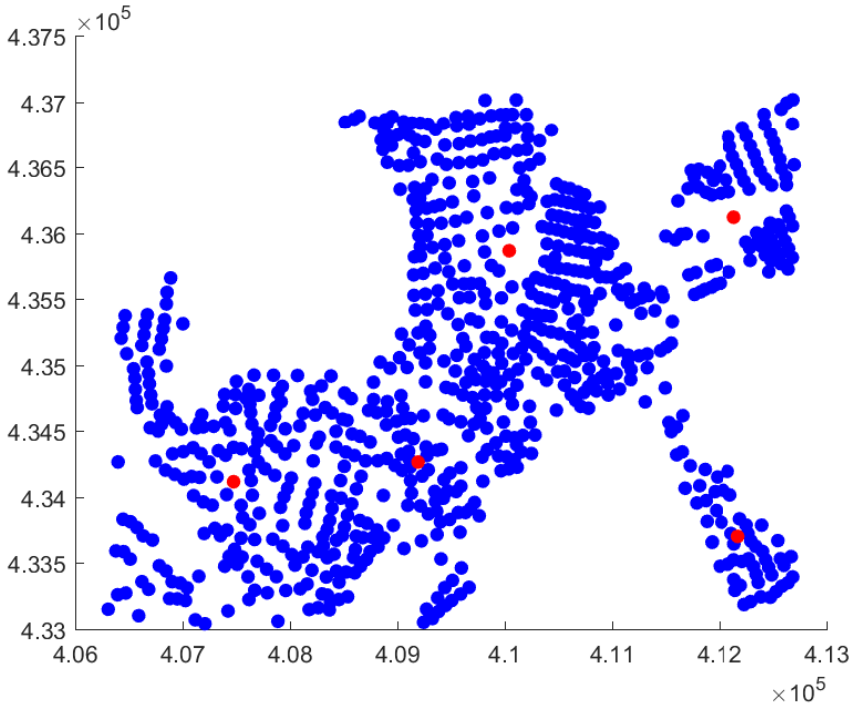

Figura 3: Vértices e Medianas - Teste 2

Nesse teste podemos visualizar que a estratégia conseguiu detectar bem alguns clusters que podem ser confirmados visualmente (Figura 3), isso nos motivou a realizar um teste maior, podemos visualizá-lo logo abaixo.

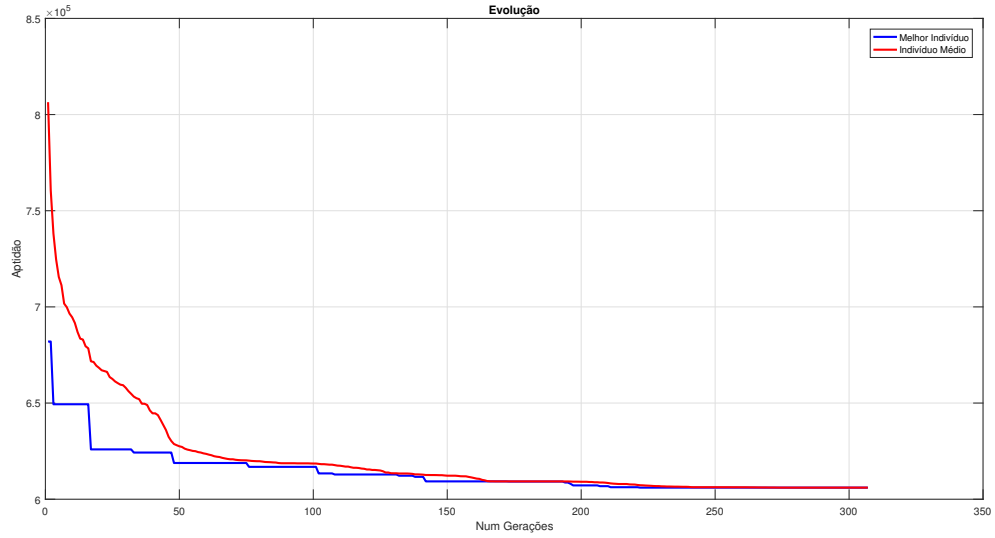

Figura 4: Convergência - Teste 2

3.1.3 Teste 3

No ultimo teste aumentamos o número de pontos de demanda para $n = 3282$ ou seja mais de três vezes a quantidade do teste anterior e desejamos encontrar $p = 5$ facilidades (medianas), o resultado após pouco mais 200 interações foi uma convergência monotônica e melhor solução com $fitness = 6.3855 \times 10^6$.

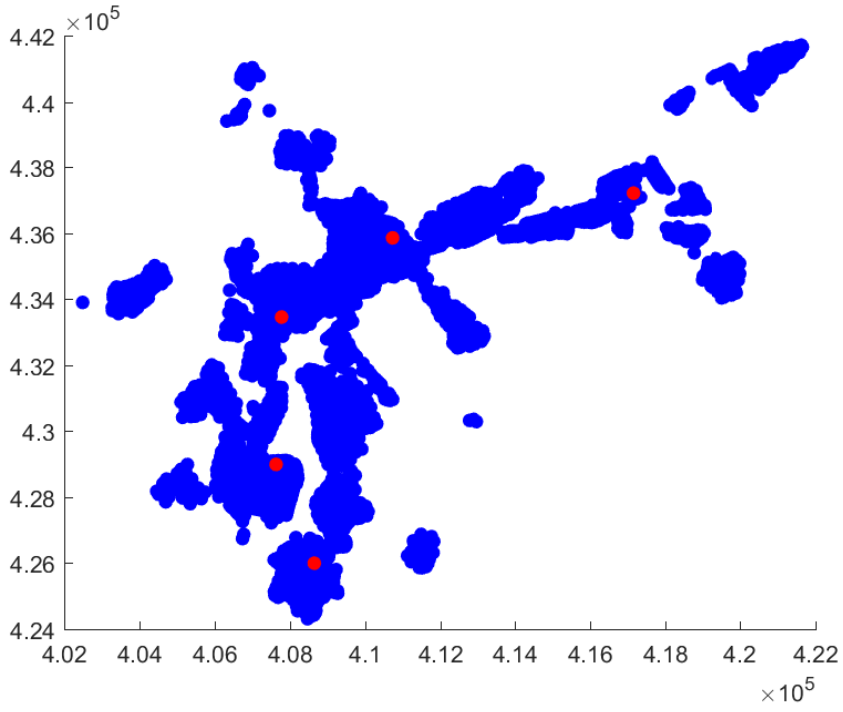

Figura 5: Vértices e Medianas - Teste 3

Nesse caso tem-se novamente clusters bem definidos (Figura 5) e foi possível verificar

que a estratégia evolucionária proposta conseguiu localiza-lós bem.

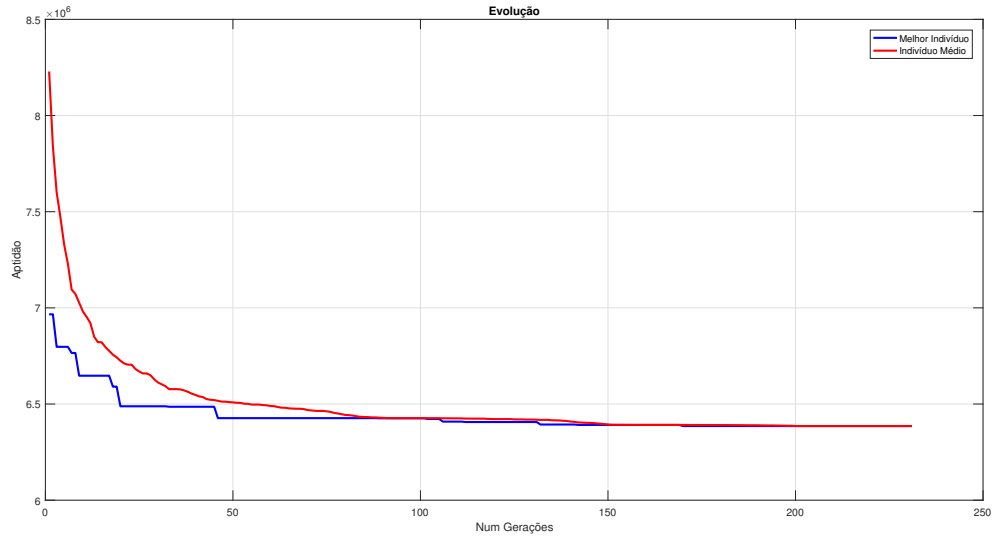

Figura 6: Convergência - Teste 3

4 Conclusão

A implementação foi simples, as técnicas utilizadas haviam sido estudadas durante as aulas de Computação Evolucionária, sendo necessário apenas algumas pesquisas e revisões sobre o assunto. O trabalho pôde ser concluído e obtiveram-se resultados dentro do esperado.

Os resultados mostram que a estratégia proposta apresentou bom desempenho nos testes. Mesmo para entradas grandes não foram necessárias muitas interações para o algoritmo convergir para uma solução promissora. Uma boa escolha dos parâmetros $crossMut$, $crossProb$ e $npop$ fazem com que o algoritmo tenha uma convergência mais rápida e satisfatória,de modo que é essencial conhecer bem o problema para decidir esses parâmetros. Nesse caso em particular, deixamos o valor da probabilidade alto com o propósito de ampliar a busca inicial e não ficar preso em uma bacia pouco promissora no restante das gerações. Como a estratégia é $\mu + \lambda$, isso não terá uma influência negativa quando o algoritmo estiver próximo de convergir, ou seja, não irá piorar a solução.

A Estratégia conseguiu lidar bem com o problema proposto. A decisão de implementar uma estratégia $\mu + \lambda$ revelou-se como um ponto importante da implementação, pois é garantida a convergência para uma solução que nos resultados dos testes foram vistas como promissoras. Isso torna a estratégia proposta uma boa ferramenta para problemas desse tipo.

Referências

[1] coord Cunha, Antonio Gaspar, coord. Takahashi, Ricardo, and coord. Antunes, Carlos Alberto Henggeler de Carvalho. *Manual de computação evolutiva e metaheurística*. Imprensa da Universidade de Coimbra, 2012.

[2] Agoston E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.