Relatório de Modelagem e Otimização de Algoritmos Algoritmo Genético aplicado ao problema das P-Medianas-Capacitadas

Fernando E. A. de Carvalho, Vinícius P. de Camargo

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)
Av. Colombo, 5790 - UEM - Bloco C56 – Maringá – PR – Brazil

{ra88408,ra88063}@uem.br

1. Introdução

O problema das p-mediana capacitado é uma um problema já muito conhecido na literatura [Mulvey and Beck 1984], [Osman and Christofides 1994] e [Maniezzo et al. 1998], tratado pela sigla PPMC, ele é classificado como NP-difícil [Garey and Johnson 1979] e assim, dificilmente encontrar-se-á um algoritmo que o resolva de forma eficiente. O problema apresentado trata-se de um grafo onde temos uma quantidade V de vértices e o objetivo é que se faça ligações entre um subconjunto $P \subseteq V$, sendo este selecionados de modo que a soma da distância entre os V vértices e as P medianas seja mínima, deve-se levar em consideração que cada vértice tem uma demanda que deve ser atendida, e uma capacidade de atendimento, inclusive os vértices selecionados como medianas devem considerar suas próprias demandas. Por exemplo, um vértice selecionado como mediana que possua capacidade 100 e demanda 10, teria uma capacidade de 90 para atender os demais vértices. Algoritmos genéticos (AG) são algoritmos bioinspirados que procuram intensivamente em uma população (conjunto de soluções) e tentam usar sempre as melhores para gerar novas soluções boas [Holland 1992].

2. Implementação

Como solução do problema é utilizado um AG que adota as seguintes características dos operadores na implementação utilizada:

- **Elitismo:** usado para preservar uma parcela da população que apresenta uma qualidade maior (leia-se menor soma das distâncias);
- **Cruzamento sexuado:** onde dois pais trocam suas medianas em um ponto K aleatório e geram dois filhos na tentativa de melhorar a qualidade da solução final;
- **Mutação:** Troca-se algumas medianas aleatoriamente para que haja uma maior variância genética;

Para o desenvolvimento do algoritmo as seguintes Classes foram utilizadas:

- **Main:** Classe que encapsula todas as partes do algoritmo genético, sua representação é feita em uma classe única por restrição do sistema run.codes; Esta classe possui os seguintes atributos:
  - **ELITISMO**, que indica quantas soluções serão preservadas de uma geração da população para a próxima;
  - **TAXA_MUTACAO**, que indica qual a taxa de soluções na qual a mutação ocorre;

  - **ITER_SEM_MOD**, número de iterações consecutivas na qual o algoritmo finaliza caso não haja mudanças no melhor resultado;
  - **TAM_POP**, tamanho da população de soluções;
  - **PENALIZACAO**, função de penalização, que empiricamente estabeleceu-se em 32000;
  - **nrVertices**, lido do arquivo;
  - **nrMedianas**, lido do arquivo;
  - **vertices**, vetor estático com os vértices lidos do arquivo;
  - **vetorDistancia**, vetor no qual soma-se as distâncias de todos para todos e as armazena para uso posterior;
  - **randomizer**, inicia uma instância aleatória para um randomizador;
- E os seguintes métodos:
  - **main**, que recebe os dados do arquivo/terminal e carrega os dados no vetor de distâncias, após isso, executa a p-mediana-capacitado;
  - **calculaDistancia**, que preenche o vetorDistancia com as distâncias de cada vértice para todos os outros;
  - **encontraDistancia**, que retorna a distância entre dois vértices;
  - **p-mediana-capacitado**, que retorna a solução da aplicação do algoritmo da p-mediana-capacitado, passando por solução inicial, iterações segundo critério de não-modificação da melhor solução, elitismo, abordagem de solução geracional, havendo ainda a possibilidade de mutação e busca local;
  - **fazBuscaLocal**, que procura o menor vizinho local recursivamente, pela técnica de best improvement;
  - **pegaMenorVizinhoLocal**, que busca o menor vizinho da determinada vizinhança.

- **Vertice:** Classe que representa um vértice do problema da p-mediana-capacitado, contendo os seguintes atributos:
  - **id**, que é recebido em ordem de leitura do arquivo;
  - **x**, que representa a posição do vértice no eixo x;
  - **y**, que representa a posição do vértice no eixo y;
  - **capacidade**, que representa a capacidade de um vértice caso este seja selecionado como mediana;
  - **demanda**, que representa o quanto este vértice precisa consumir da capacidade da mediana.
- E os seguintes métodos:
  - **encontraMedianaMaisProximaComCapacidade**, que encontra a mediana mais próxima que ainda tem capacidade disponível, sempre levando em conta que uma mediana já consome por padrão sua própria demanda.

- **Mediana:** classe que representa uma mediana, e ainda sim é um tipo de vértice com algumas informações extras, como:
  - **somaDasDistancias**, que armazena a soma das distâncias da mediana para os vértices ligados a ela;
  - **capacidadeUsada**, que armazena a capacidade da mediana que já foi utilizada;
- E os seguintes métodos:

  - **consultaDistancia**, na qual retorna-se a distância entre uma mediana e o vértice;
  - **adicionaDadosVertices**, que adiciona a distância do vértice na soma das distâncias e diminui a capacidade disponível da mediana.

- **Solução:** classe que representa uma solução, possuindo uma qualidade e uma lista de medianas. Como métodos ela possui:
  - **fazMutacao**, que é ativado pela taxa de mutação as vezes, pegando uma quantidade aleatória de medianas da solução e substituindo por outras. Este método representa o operador de mutação do algoritmo genético;
  - **avaliarQualidadeSolucao**, que avalia a qualidade de uma solução e penaliza caso necessário;
  - **geraSolucaoAleatoria**, que junto com a função criaPrimrasMedianas, cria uma primeira solução para o problema.
  - **ligaVerticesAsMedianasERetornaUltimoIndiceUsado**, que tenta ligar todos os vértices a alguma mediana criada anteriormente e retorna o índice do maior vértice utilizado, sendo esta informação utilizada no caso da necessidade de penalização (número de índices menor do que a número de vértices menos número de medianas).

- **População:** classe que representa o conjunto das soluções, tendo como métodos:
  - **fazCruzamento**, que representa o operador de cruzamento, recebendo um pai e uma mãe e cruzando suas medianas até um ponto aleatório, a menos que o pai ou mãe já contenham esta mediana. Cada aplicação desta função em duas soluções, gera duas novas soluções;
  - **encontraMelhorSolucaoDaLista**, que encontra a melhor solução da população;
  - **avaliarQualidadePopulacao**, que soma e retorna a qualidade de todas as soluções da população;
  - **selecaoDeRoleta**, que retorna uma solução pseudo-aleatória, já que há um grande viés intencional para que seja retornada uma boa solução. Este método representa o operador de seleção.

3. Resultados

Os testes foram executados em uma máquina com a especificação abaixo e obtiveram os resultados mostrados na tabela 3 (com os ótimos referenciais obtidos em [Stefanello et al. 2009]) e apresentaram a convergência do gráfico 1 para o teste SJC1.dat.

- Processador: Intel i5 3ª geração CPU @ 2.50Ghz;
- Memória RAM: 6Gb (4Gb alocados para os testes);
- Sistema Operacional: Windows 10 Home Edition;
- Versão JVM: 1.8.

Caso	MS	Alg	Alg_bl	GAP%
Teste2.dat	966	1065,38	1034,15	7,05
SJC1.dat	17.288,99	17.596,02	17.580,23	1,68
SJC2.dat	33.270,94	35.505,12	35.035,58	5,30
SJC3a.dat	45.335,16	49.527,11	48.136,4	8,38
p3038_600.dat	122.710,58	206.606,16	200.047,82	63,02
p3038_900.dat	92.309,52	164.777,24	158.186,15	71,36
p3038_1000.dat	85.854,05	153.033,79	143.912,38	67,62

Tabela 1. Resultados obtidos com a execução do algoritimo genético e de busca local

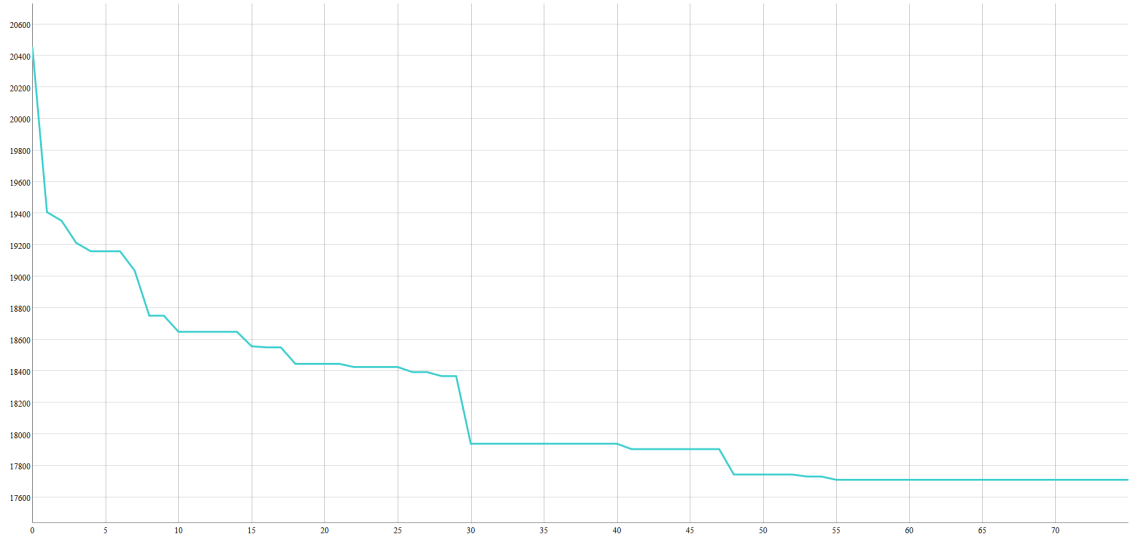

Figura 1. Gráfico de convergência para o caso de teste SJC1dat.

4. Instruções de execução

As instruções de execução assumem que o leitor tenha um compilador de java instalado em seu computador.

1. Abra a pasta que contém o arquivo Main.java
2. Abra o terminal
3. Digite javac Main.java
4. Digite java -Xmx4G Main < arquivoDesejado.in //Neste exemplo a memória máxima alocada para a Heap da JVM (*Java Virtual Machine*) é 4G, mas esta quantidade não resolve todas as instâncias e o arquivoDesejado.in deve ser substituido pelo arquivo que deseja-se testar.

5. Conclusão

Apesar de produzir soluções não tão próximas do ótimo da literatura, o algoritmo genético implementado consegue algumas vezes chegar bem próximo dos resultados ótimos, e por ser baseado em gerações randômicas os resultados podem variar de uma execução para outra, ou seja, são estocásticos. A alteração de parâmetros como tamanho da população,

elitismo, taxa de mutação, vizinhos locais, e a seleção para o crossover também podem influenciar no resultado, assim como a aplicação ou não da busca local. Assim, concluise que o algoritmo pode chegar a resultados aceitáveis, mas sendo o problema abordado amplamente na literatura, melhores e mais rápidos algoritmos podem ser encontrados.

Referências

[Garey and Johnson 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

[Holland 1992] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, 267(1):66–73.

[Maniezzo et al. 1998] Maniezzo, V., Mingozzi, A., and Baldacci, R. (1998). A bionomic approach to the capacitated p-median problem. *Journal of Heuristics*, 4(3):263–280.

[Mulvey and Beck 1984] Mulvey, J. M. and Beck, M. P. (1984). Solving capacitated clustering problems. *European Journal of Operational Research*, 18(3):339 – 348.

[Osman and Christofides 1994] Osman, I. H. and Christofides, N. (1994). Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 1(3):317–336.

[Stefanello et al. 2009] Stefanello, F., Müller, F. M., and de Araújo, O. C. B. (2009). Estudo sobre a resolução do problema das p-medianas capacitado. In Buriol, L., Ritt, M., and Benavides, A., editors, *Anais do III ERPOSul - Encontro Regional de Pesquisa Operacional da Região Sul*, page 5, Porto Alegre - RS, Brasil. Instituto de Informática da UFRGS.