

Problema do caixeiro viajante utilizando Algoritmos Genéticos

Diego Pontes Pasquini; Gabriela Araújo Coelho; Hugo Gustavo Valin Oliveira da Cunha; Jessiane Gomes Andrade; Tatiane Fernanda de Souza Silva

Dezembro de 2017

Disciplina: Inteligência Computacional
Professora: Dra. Rita Maria da Silva Julia

Resumo

Neste trabalho foi implementado um programa utilizando Algoritmos Genéticos(AGs) com o objetivo de resolver o problema do caixeiro viajante. O projeto foi desenvolvido em linguagem Java, visando sua portabilidade.

Palavras-chave: Algoritmos Genéticos. Inteligência Computacional. Caixeiro viajante.

1 Introdução

A pesquisa sobre a área de Computação Evolutiva(CE) tem se expandido rapidamente, principalmente para resolver problemas de otimizaçãocombinatória NP-difícil.

Os Algoritmos Evolutivos(AEs) são métodos de busca estocásticos que imitam a evolução biológica natural. Eles são compostos por uma sequência de passos até a solução, sendo estes passos os mesmos para uma ampla gama de problemas, fornecendo robustez e flexibilidade.

Existem vantagens para utilizarmos esses algoritmos como o desenvolvimento de algoritmos capazes de encontrar soluções adequadas para problemas complexos, ainda não resolvidos por outras técnicas computacionais, a simplicidade dos métodos utilizando princípios básicos de Teoria da Evolução e Genética que podem ser modelados por poucas e simples linhas de código e também a adaptação relativamente fácil para problemas das mais diversas áreas.

Os principais Algoritmos Evolutivos são:

1. Algoritmos Genéticos (AG);
2. Estratégias Evolutivas (EE);
3. Programação Evolutiva (PE);
4. Programação Genética (PG);
5. Genetic Programming (GP);
6. Sistemas Classificadores (SC);

1.1 Descrição do Problema Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um problema combinatório fundamentalmente matemático que tem perdurado por longo tempo. Não se sabe ao certo um documento que comprove sua autoria, muito menos a origem do nome. Em meados de 1800, problemas relacionados ao PCV começaram a ser desenvolvidos, mas somente em 1950 ficou conhecido globalmente em sua forma geral. [1].

O PCV é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

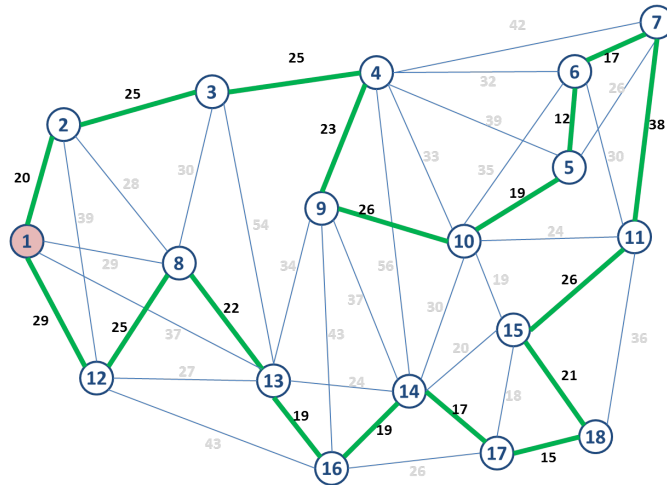


Figura 1: Exemplo de solução para um PVC.

Os algoritmos heurísticos constituem uma alternativa computacionalmente viável encontrada pelos pesquisadores para resolver o PCV. A limitação

desta alternativa é a qualidade da solução obtida, que não necessariamente é a ótima. Dentre os métodos heurísticos que encontram soluções de forma eficiente estão os Algoritmos Genéticos (AGs).

Os AGs são métodos de busca e otimização inspirados na biologia evolucionária da população de seres vivos [2][3] [4]. Como uma técnica de otimização e busca, os algoritmos genéticos apresentam uma função objetivo (também chamado aptidão ou fitness), utilizada para avaliar cada uma de uma soluções obtidas, uma vez que o método gera um conjunto de soluções, e não apenas uma.

1.2 Aplicações

Existem várias aplicações para o problema do caixeiro viajante. [5] Um exemplo é o processo de perfuração de placas de circuitos impressos em sua confecção. Uma placa de circuito impresso pode conter centenas ou milhares de furos para soldagem de componentes eletrônicos. Como os furos podem ser de tamanhos diferentes, é necessária a troca da ferramenta. Essa troca de ferramenta é um processo lento. Para que a ferramenta não seja trocada várias vezes, é necessária uma otimização perfurando primeiro todos os furos de mesmo diâmetro. Assim, esse processo pode ser visto como um problema do caixeiro viajante. Percorrendo-se o melhor caminho possível, economiza-se tempo, aumentando a produtividade do processo [6].

Uma segunda aplicação é a que ocorre na análise de estruturas de cristais na cristalografia por raios-x. A cristalografia analisa a disposição dos átomos em sólidos. Para isso um difratômetro obtém informações da estrutura de um material cristalino medindo a intensidade dos raios-x refletidos do material, saindo de várias posições. Em alguns experimentos, o difratômetro deve realizar até 30.000 deslocamentos para fazer as medidas. Isso significa que pode haver um percurso com até 30.000 pontos de medida. Para minimizar o tempo gasto na mensuração, deve-se escolher um percurso mínimo entre os pontos de medida, o qual pode ser modelado através do problema do caixeiro viajante [6].

2 Metodologia

Para o PCV, cada cromossomo codifica uma solução, isto é, a rota que possui a menor distância. A função de adaptação (fitness), que avalia cada cromossomo, está relacionada com o tamanho da rota, no qual depende da ordem em que as cidades aparecem na mesma.

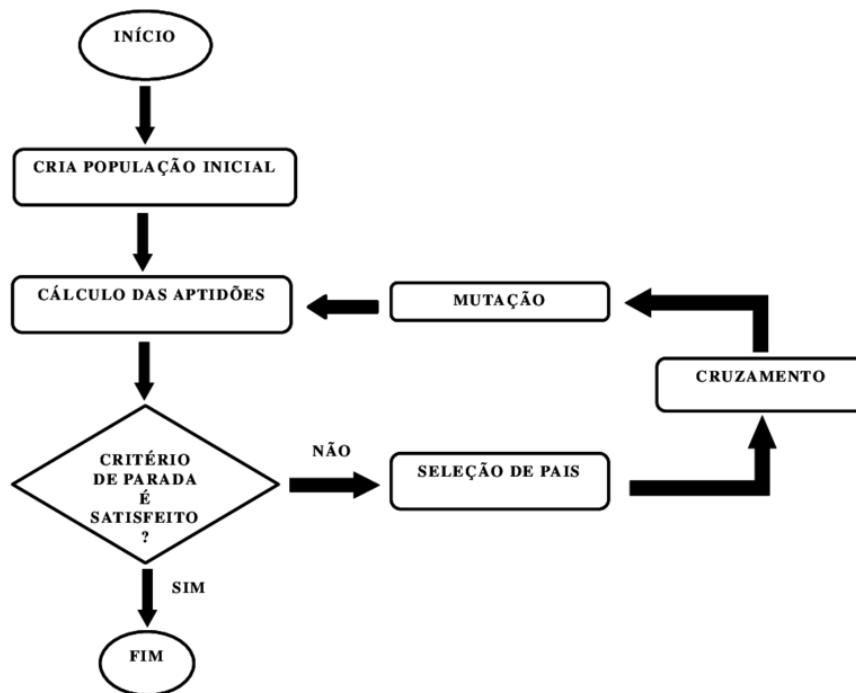


Figura 2: Esquema geral de funcionamento do AG.

Foi utilizado uma matriz 42x42 para montar o grafo das cidades com as distâncias entre cada uma delas. A representação de um cromossomo, para esta aplicação, é um vetor de tamanho 42 com a ordem a das cidades que irão ser percorridas.

Estrutura de um cromossomo: $[d_0, d_1, d_2, \dots, d_{41}]$, onde d é a distância entre a cidade i e a cidade $i-1$.

A aptidão (*fitness*) é o cálculo da a distância total que é percorrida, sendo o objetivo é encontrar a menor distância.

O método utilizado para fazer a seleção dos pais foi o método da roleta. Em seguida, a partir dos pais selecionados para a reprodução, é feito um *Crossover* OX para gerar os filhos. Esses filhos sofrem mutação por um algoritmo de mutação pode inversão e ocorre a reincerção dos mesmo na população calculando a aptidão de cada um.

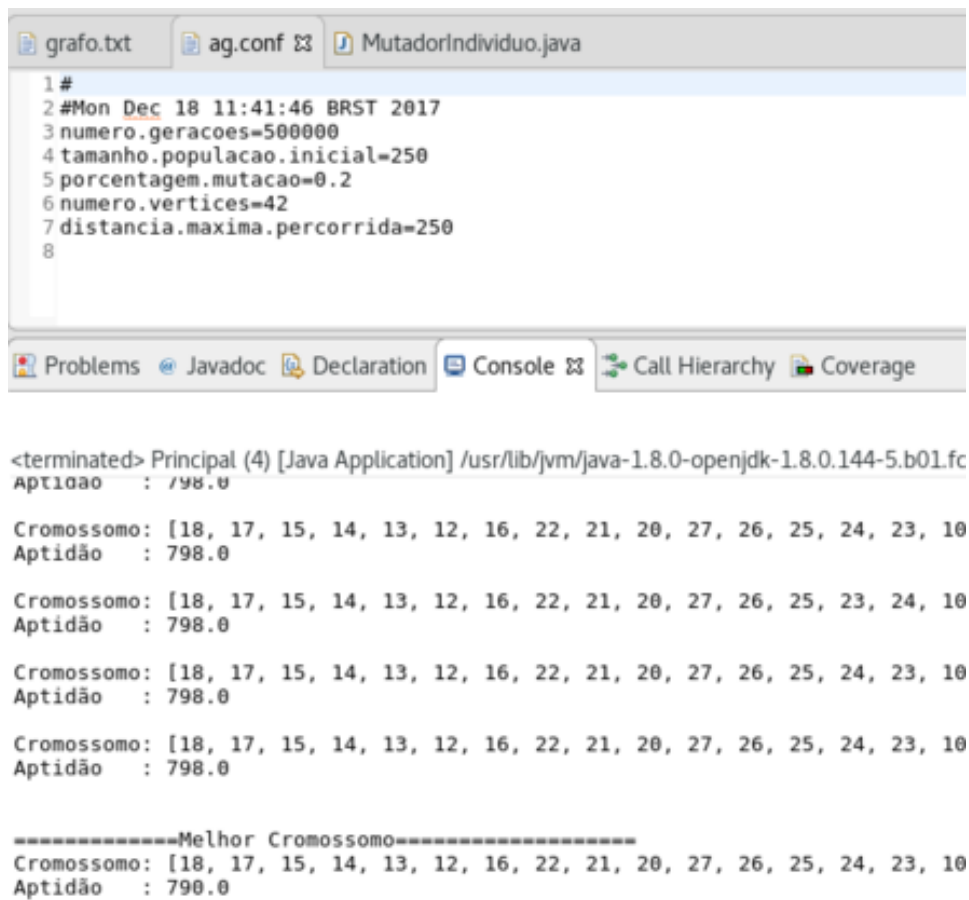
3 Configuração do AG

O programa implementado possui um arquivo de configurações onde fica bem mais simples mudar os valores dos elementos principais de um AG. Esse arquivo possui uma variável para guardar o número de gerações que o programa irá rodar, uma variável que irá guardar a quantidade de indivíduos que serão gerados na população inicial, a porcentagem de mutação que será

sofrida pelos filhos gerados, a quantidade de vértices do grafo, e por último, a distância máxima percorrida.

4 Resultados

Nos primeiros resultados podemos utilizamos a mutação que troca dois genes, escolhidos aleatoriamente, de posição.



```
1#
2#Mon Dec 18 11:41:46 BRST 2017
3numero.geracoes=500000
4tamanho.populacao.inicial=250
5porcentagem.mutacao=0.2
6numero.vertices=42
7distancia.maxima.percorrida=250
8
```

```
<terminated> Principal (4) [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.144-5.b01.fc
Aptidão : 798.0

Cromossomo: [18, 17, 15, 14, 13, 12, 16, 22, 21, 20, 27, 26, 25, 24, 23, 10
Aptidão : 798.0

Cromossomo: [18, 17, 15, 14, 13, 12, 16, 22, 21, 20, 27, 26, 25, 23, 24, 10
Aptidão : 798.0

Cromossomo: [18, 17, 15, 14, 13, 12, 16, 22, 21, 20, 27, 26, 25, 24, 23, 10
Aptidão : 798.0

Cromossomo: [18, 17, 15, 14, 13, 12, 16, 22, 21, 20, 27, 26, 25, 24, 23, 10
Aptidão : 798.0

=====Melhor Cromossomo=====
Cromossomo: [18, 17, 15, 14, 13, 12, 16, 22, 21, 20, 27, 26, 25, 24, 23, 10
Aptidão : 799.0
```

Figura 3: Resultado da mudança de aptidão através das gerações utilizando a mutação invertendo 2 genes.

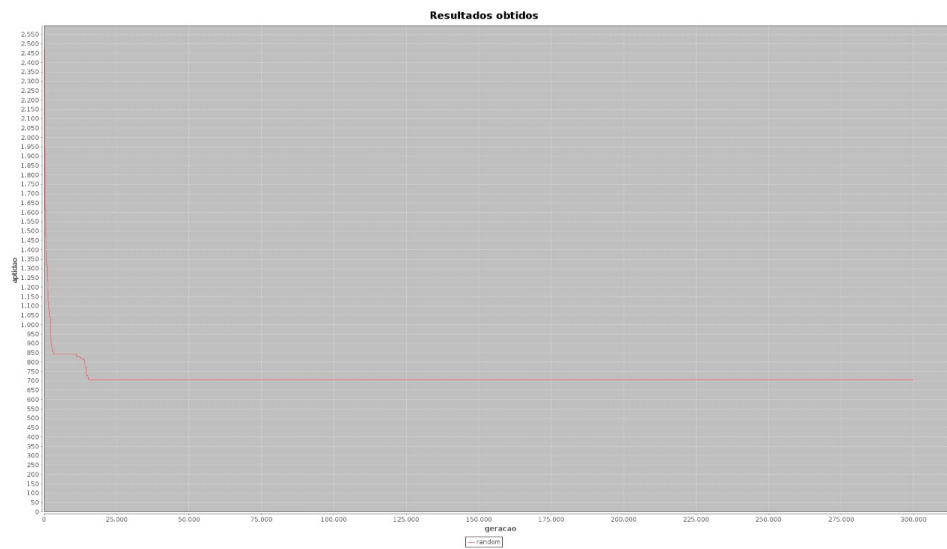


Figura 4: Evolução do AG através das gerações

Neste resultado podemos utilizamos a mutação que inverter uma sequência de genes, onde o ponto inicial e o ponto final da sequência são escolhidos aleatoriamente.

```
1 #
2 #Mon Dec 18 11:41:46 BRST 2017
3 numero.geracoes=500000
4 tamanho.populacao.inicial=250
5 porcentagem.mutacao=0.2
6 numero.vertices=42
7 distancia.maxima.percorrida=250
8

Problems Javadoc Declaration Console Call Hierarchy Coverage

<terminated> Principal (4) [Java Application] /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.144-5.b01.fc26.x86_64
Aptidao : 702.0

Cromossomo: [7, 6, 5, 4, 3, 2, 0, 41, 1, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30]
Aptidao : 702.0

Cromossomo: [10, 9, 8, 7, 6, 5, 4, 3, 2, 0, 1, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32]
Aptidao : 702.0

Cromossomo: [30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]
Aptidao : 702.0

Cromossomo: [39, 38, 37, 36, 35, 34, 33, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19]
Aptidao : 702.0

=====Melhor Cromossomo=====
Cromossomo: [18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38]
Aptidao : 699.0
```

Figura 5: Resultado da mudança de aptidão através das gerações utilizando a mutação por inversão.

5 Conclusão

Com esse trabalho podemos concluir que a escolha de bons métodos heurísticos para os problemas complexos fazem toda a diferença nos resultados. E, neste caso, comparando a mutação invertendo apenas dois genes do cromossomo com a mutação que inverte uma parte do cromossomo, podemos observar que no segundo caso chegmos em um resultado bem melhor.

Referências

- [1] David L. Applegate, Robert E. Bixby, Vasek Chvátal and William J. Cook *The Traveling Salesman Problem. A Computational Study*, Princeton University Press, 2006
- [2] Darwin, Charles *The Traveling Salesman Problem. On the Origin of Species by Means of Natural Selection*, London 1859
- [3] Holland, John H. *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI 1975

- [4] Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989
- [5] Reinelt, G. *The Traveling Salesman: Computational Solutions for TSP Applications*, Lecture Notes in Computer Science, 840, Springer-Verlag, 1994
- [6] Freddo, Ademir Roberto and Brito, Robison Cris. *Implementação da Metaheurística GRASP para o Problema do Caixeiro Viajante Simétrico*. Disponível em: <http://www.inf.ufpr.br/aurora/disciplinas/topicosia2/downloads/trabalhos/GraspTSP.pdf>. Acesso em: 19/12/2017