Algoritmos Genéticos - Resolvendo o Problema do Caixeiro Viajante

João Baptista Cardia Neto

Setembro 2016

1 Introdução

O presente trabalho visa listar e mostrar as soluções feitas para o problema do Caixeiro Viajante utilizando algoritmos genéticos. A implementação do sistema foi quebrada em partes e os arquivos são listados abaixo:

Arquivo	Implementação
agCaixeiro.m	Função que resolve problema do caixeiro viajante utilizando algoritmo genético;
crossover.m	Efetua o crossover entre dois pais gerando um descendente, faz isso com order operator
fitnessCalc.m	Percorre um array com um path contendo uma solução e faz o somatório de todas as distâncias.
generateGraphRepr.m	Gera representação gráfica da solução
generateInitialPop.m	Gera população inicial aleatória utilizando randperm
generateNewPop.m	Gera nova população com base nos selecionados na roleta
mutation.m	Efetua mutação
runRoullete.m	Roda roleta para seleção
selectionFit.m	Faz a seleção utilizando o método da roleta para escolher
$tsp_280_impl.m$	Programa principal para solução

2 Parâmetros

O sistema foi todo parametrizado, sendo os dois primeiros parâmetros a taxa de Seleção e Mutação. A função principal possui um conjunto de parâmetros além desses, sendo eles:

```
[solV,path] = agCaixeiro(
    0.6, %taxa de selecao
    0.5, %taxa de mutation
    size(citiesData,1)*2, %populacao inicial
    citiesData, %dados das cidades
    size(citiesData,1)*2, %tamanho maximo da populacao
    100000, %numero de iteracoes
    0 %mostrar graficos passo a passo
);
```

Todas as implementações estão comentadas e listadas em cada um dos seus arquivos.

Importante comentar que, além da implementação padrão, ao aplicar a mutação a uma possível solução é feito uma escolha randômica sobre qual algoritmo de mutação será utilizado. O código é demonstrado, em partes, abaixo:

```
    ...
    son2 = crossover(
        selectedPeople(n+1,:),
        selectedPeople(n,:)
    );
    %Se ainda for necessario fazer mutacoes
    %randomiza um numero se
    %diz se vai fazer ou nao
    if numberMutation > 0 && randi([0 1]) == 1
        %sorteia entre os dois tipos de mutacao
        son = mutation(son,randi([1 2]));
        %sorteia entre os dois tipos de mutacao
        son2 = mutation(son2,randi([1 2]));
        numberMutation = numberMutation - 2;
    end
    ..
```

3 Experimentos

Foram efetuados três experimentos, a tabela abaixo descrimina cada um deles:

Experimento	Taxa Seleção	Taxa Mutação	População	Número de iterações
1	0.6	0.5	102	100000
2	0.5	0.2	102	100000
3	0.9	0.5	102	100000

O experimento três foi o que mais se aproximou do ótimo, tendo a solução como as cidades na ordem 22, 3, 36, 35, 20, 29, 2, 16, 50, 21, 34, 30, 10, 39, 33, 45, 15, 44, 42, 40, 41, 19, 37, 17, 4, 13, 25, 24, 43, 7, 23, 48, 27, 46, 51, 6, 14, 18, 47, 12, 5, 49, 9, 38, 11, 32, 1, 8, 26, 31, 28 e o custo do caminho 443. Abaixo são

exibidos os gráficos com os resultados dos experimentos, eles também podem ser alcançados executando o programa de resolução.

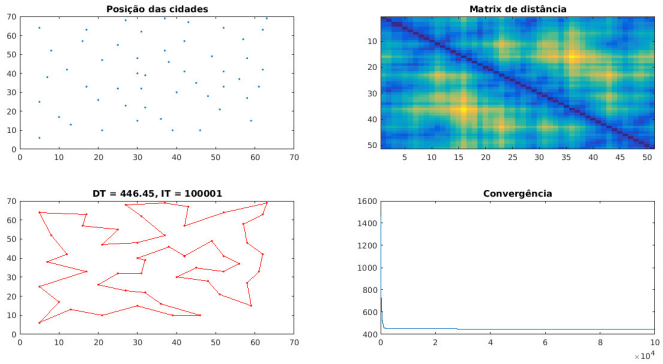

Figure 1: Resultado do experimento 1.

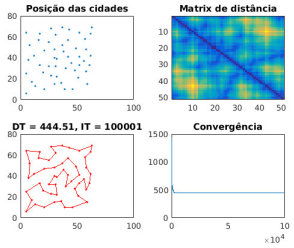

Figure 2: Resultado do experimento 2.

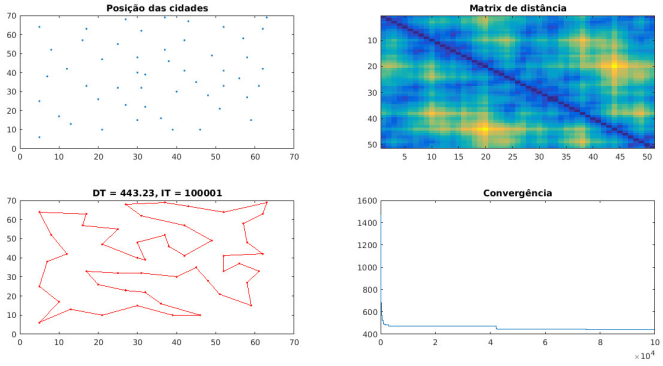

Figure 3: Resultado do experimento 3.