# Algoritmo Genético: Problema do Caixeiro Viajante

## Descrição do Problema

O problema do caixeiro viajante (*Travelling Salesman Problem* – TSP) é de natureza combinatória e é uma referência para diversas aplicações, e.g., projeto de circuitos integrados, roteamento de veículos, programação de produção, robótica, etc. Em sua forma mais simples, no TSP o caixeiro deve visitar cada cidade somente uma vez. Dado o custo da viagem (ou distância) entre cada uma das cidades, o problema do caixeiro é determinar qual o itinerário que possui o menor custo.

## Objetivo da Atividade.

Esta atividade consiste em implementar um algoritmo genético para resolver o problema do caixeiro. As coordenadas das cidades serão passadas em separado pelo professor.

## Entrega da Atividade

Deve-se enviar para o e-mail do professor da disciplina um arquivo executável, o código fonte e um relatório.

O algoritmo genético deve ser implementado de preferência em **MatLab** ou **Scilab**. O relatório deve contemplar os seguintes aspectos:

- pseudo-código do algoritmo genético;
- forma de codificação (representação) dos indivíduos;
- função de avaliação e mecanismo de seleção;
- operadores genéticos utilizados;
- critérios para escolha de quais indivíduos comporão a nova geração;
- critério de parada;
- listagem do código-fonte, com bom detalhamento das classes e funções;
- análise paramétrica do tamanho da população e das taxas de crossover e mutação;

É interessante a utilização de gráficos para acompanhar a evolução do melhor *fitness* e do *fitness* médio da população;

Data de entrega: **duas semanas após o enunciado em sala de aula.**

## Considerações

Deve-se calcular a distância Euclidiana entre uma cidade e todas as demais.

As três representações mais utilizadas para o Problema do Caixeiro Viajante são: *adjacency, ordinal e path.* Pesquise sobre elas.

Na implementação da atividade sugere-se a representação *path*, mas isto não impede que vocês implementem as outras duas.

A - Para a representação *path*, os tipos clássicos de crossover são:

1 - *Order Operator* (OX)

Proposto por Davis. Constrói um descendente escolhendo uma subsequência de um tour de um pai e preservando a ordem relativa das cidades do outro pai.

Exemplo:

p1 = (1 2 3 | 4 5 6 7 | 8 9)
p2 = (4 5 2 | 1 8 7 6 | 9 3) - mantém os segmentos selecionados
o1 = (x x x | 4 5 6 7 | x x)
o2 = (x x x | 1 8 7 6 | x x)

A seguir, partindo do ponto do segundo corte de um dos pais, as cidades do outro pai são copiadas na mesma ordem, omitindo-se as cidades que estão entre os pontos de corte. Ao se atingir o final do vetor continua-se no início do mesmo.

o1 = (2 1 8 | 4 5 6 7 | 9 3)
o2 = (3 4 5 | 18 7 6 | 9 2)

2 - *Partially-Mapped Operator* (PMX)

Proposto por Goldberg e Lingle. Constrói um descendente escolhendo uma subsequência de um tour de um pai e preservando a ordem e a posição do outro pai de quantas cidades forem possíveis.

Exemplo:

p1 = (1 2 3 | 4 5 6 7 | 8 9)
p2 = (4 5 2 | 1 8 7 6 | 9 3) - troca os segmentos selecionados
o1 = (x x x | 1 8 7 6 | x x)
o2 = (x x x | 4 5 6 7 | x x) - define o mapeamento (1-4,8-5,7-6,6-7)

Copia os genes remanescentes em p1 para o1 usando o mapeamento definido:
o1 = (1-4 2 3 | 1 8 7 6 | 8-5 9) = (4 2 3 | 1 8 7 6 | 5 9)
Constrói o2 da mesma forma:
o2 = (4-1 5-8 2 | 4 5 6 7 | 9 3) = (1 8 2 | 4 5 6 7 | 9 3)

3 - *Cycle Crossover* (CX)

Proposto por Oliver, I.M., D.J. Smith e J. R. C. Holland. Este operador gera os filhos de forma que cada cidade e sua posição venha de um dos pais.

Exemplo:

p1 = (1 2 3 4 5 6 7 8 9)
p2 = (4 1 2 8 7 6 9 3 5)

a partir do primeiro pai, pega-se a primeira cidade e coloca no filho 1.
f1 = (1 x x x x x x x)
a próxima cidade escolhida será a correspondente da primeira posição do pai 2, mas na posição do pai 1, nesse caso é a cidade 4, e ficará na posição (4).
f1 = (1 x x 4 x x x x)
a próxima será a cidade 8, e assim por diante até que teremos
f1 = (1 2 3 4 x x x 8 x)
onde não temos mais escolhas, portanto completamos com as cidades do pai 2
f1 = (1 2 3 4 7 6 9 8 5)

Similarmente,

f2 = (4 1 2 8 5 6 7 3 9)

B - Para a representação *path* os tipos clássicos de mutação são:

*1 - Reciprocal* Exchange

A mutação de Troca Recíproca sorteia duas cidades e inverte sua posição.

Exemplo:
(1 2 3 **4** 5 6 **7** 8 9) → (1 2 3 **7** 5 6 **4** 8 9)

2 – *Inversion*

A mutação é feita invertendo-se a ordem dos segmentos selecionados. Exemplo:
(1 2 3 | **4 5 6 7** | 8 9) => (1 2 3 | **7 6 5 4** | 8 9)

C – Para mais detalhes consulte artigo disponível na página do curso: Artigos complementares, Algoritmos genéticos III.