



## Sistemas Inteligente/ Inteligência Artificial

# Trabalho AV3 - Busca/Otimização Meta-heurística

Professor: Prof. Msc. Paulo Cirillo Souza Barbosa

## Discussão inicial sobre o trabalho.

O presente trabalho se trata da resolução de diferentes problemas mediante busca/otimização meta-heurística. Tais procedimentos são considerados solucionadores de problemas complexos que utilizam estratégias heurísticas para explorar o espaço de estado em busca da melhor solução possível. As abordagens desenvolvidas no presente trabalho são do tipo meta-heurística, pois, são aplicáveis a uma variedade de problemas de busca/otimização podendo ser facilmente adaptadas para lidar com diferentes tipos de objetivos e restrições. Os problemas propostos no presente trabalho são divididos como:

- A primeira parte trata-se da utilização dos algoritmos: *Hill Climbing*\*; *Local Random Search(LRS)*; *Global Random Search(GRS)* e *Simulated Annealing*. Para soluções de problemas do tipo contínuo com espaço de estados infinito.
- A segunda parte, trata-se da utilização do projeto de um algoritmo genético para solução de dois problemas discretos (combinatória).

## 1) Problema de Minimização/Maximização de função Custo/Objetivo

Algoritmos e otimização são métodos de busca, em que o objetivo é encontrar uma solução de um problema de otimização. Cada problema deste tipo, consiste de elementos elementares como:

- Uma função objetivo que representa a quantidade a ser otimizada, ou seja, uma quantidade que deseja-se minimizar (custo) ou maximizar (objetivo).
- Um conjunto de variáveis desconhecidas  $\mathbf{x}$ , que afetam o valor da função objetivo. Se  $\mathbf{x}$  representa as variáveis independentes, então  $f(\mathbf{x})$  quantifica a qualidade de uma solução candidata.
- Um conjunto de restrições, que limitam os valores que as variáveis independentes podem assumir. A maioria dos problemas, definem uma restrição de limites (caixa) representando o domínio dos valores de cada variável presente em  $\mathbf{x}$ . Contudo, as restrições podem ser mais complexas ao excluir certas soluções candidatas do conjunto de soluções.

Um problema de otimização sem restrição, pode ser escrito da seguinte forma:

$$\begin{aligned} &\text{minimize } f(x), \mathbf{x} = (x_1, x_2, \dots, x_p) \\ &\text{subject to } x_j \in \text{dom}(x_j) \end{aligned}$$

em que  $\text{dom}(x_j)$  é o domínio da variável  $x_j$  e para um problema contínuo, este domínio para cada variável é o conjunto dos reais ( $\mathbb{R}$ ).

Desta maneira, é solicitado que resolva os problemas exibidos na presente seção, utilizando os algoritmos descritos na seção anterior. Para cada um deles, projete uma sequência de 100 rodadas de modo que se armazene a cada rodada a solução obtida pelo algoritmo implementado. Estas rodadas não devem ser confundidas com as iterações de cada algoritmo (ou seja, cada algoritmo pode possuir um valor de quantidade máxima de iterações). No final deste experimento, deve-se compor uma tabela com a moda de soluções de cada algoritmo. Outra definição importante para todos os algoritmos implementados na presente seção, é o teste se um candidato gerado está no limite imposto pelas variáveis independentes (restrição do tipo caixa). Além disso, leve em consideração os seguintes hiperparâmetros que são específicos de cada algoritmo:

- Para o algoritmo **Hill Climbing (Subida de Encosta)**, escolha como ponto inicial, o valor de limite inferior do domínio de  $\mathbf{x}$ . Para gerar um candidato a partir de  $\mathbf{x}_{best}$ , utilize como critério de vizinhança  $|\mathbf{x}_{best} - \mathbf{y}| \leq \varepsilon$ , em que  $\mathbf{y}$  é um possível candidato da vizinhança e  $\varepsilon$  deve ser definido com um valor pequeno (0,1). Para este hiperparâmetro, faça a identificação do menor valor de abertura que encontra a solução ótima.
- Para o algoritmo **Busca Local Aleatória (Local Random Search - LRS)**, deve-se especificar o valor de desvio-padrão  $0 < \sigma < 1$ . O candidato inicial, que nada mais é o  $\mathbf{x}_{best}$  deve ser gerado utilizando um número aleatório gerado por uma distribuição uniforme com os limites impostos pelas variáveis independentes. Para o hiperparâmetro  $\sigma$ , faça a identificação do menor valor que encontra a solução ótima.
- Para o algoritmo de **Busca Aleatória Global (Global Random Search - GRS)**, gere um novo candidato através de um número aleatório gerado por uma distribuição uniforme com os limites impostos pelas variáveis independentes. Para o hiperparâmetro  $\sigma$ , faça a identificação do menor valor que encontra a solução ótima.
- Para o algoritmo de **Têmpera Simulada (Simulated Annealing)**, faça a definição do valor inicial de Temperatura  $T$ , o valor do desvio-padrão  $0 < \sigma < 1$ . Para o hiperparâmetro  $\sigma$ , faça a identificação do menor valor que encontra a solução ótima. Para a etapa de resfriamento, deve-se escolher uma das equações presentes no slide 106/108.

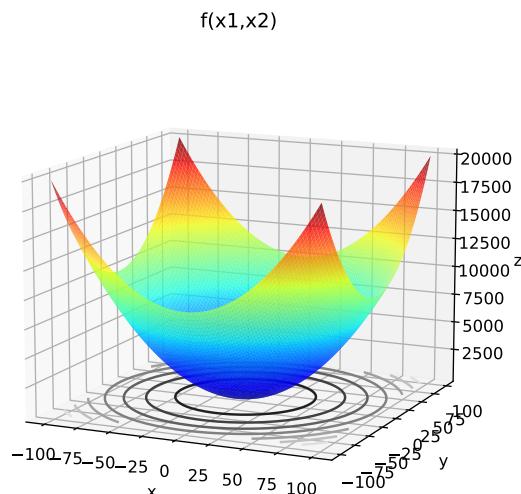
Como critérios de parada de cada algoritmo, deve-se definir uma quantidade máxima de iterações em 1000 e também projetar uma possível parada antecipada (antes da quantidade máxima de iterações), que pode ser implementada ao verificar que quando não há melhoria na solução  $\mathbf{x}_{best}$  a cada  $t$  iterações. Além disso, cada algoritmo deve ser executado uma quantidade  $R$  de vezes. O resultado do modelo (ótimo/subótimo encontrado), se dá pelo resultado mais frequentista da função de avaliação.

Os problemas de otimização a serem resolvidos nesta seção, são:

1. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = x_1^2 + x_2^2$$

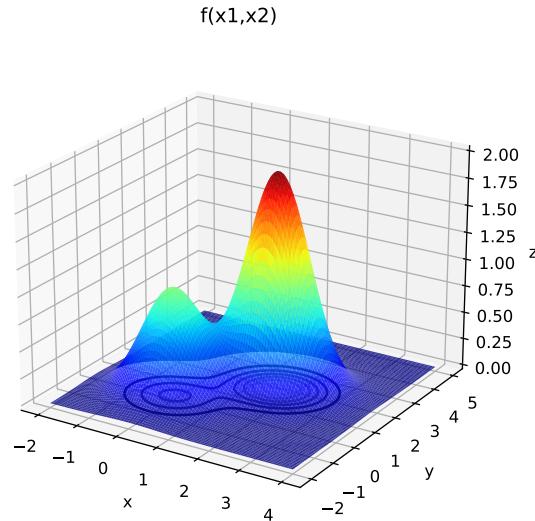
em que o domínio das variáveis são  $x_1, x_2 \in [-100, 100]$ . A presente função possui como gráfico:



2. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **máximo** da função:

$$f(x_1, x_2) = e^{-(x_1^2+x_2^2)} + 2 \cdot e^{-(x_1-1.7)^2+(x_2-1.7)^2}$$

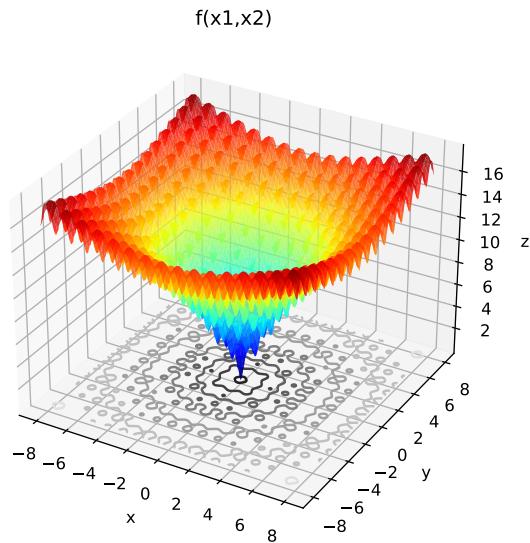
em que o domínio das variáveis são  $x_1 \in [-2, 4]$  e  $x_2 \in [-2, 5]$ . A presente função possui como gráfico:



3. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = -20 * e^{-0.2\sqrt{0.5 \cdot (x_1^2+x_2^2)}} - e^{0.5 \cdot (\cos(2\pi x_1) + \cos(2\pi x_2))} + 20 + e^1$$

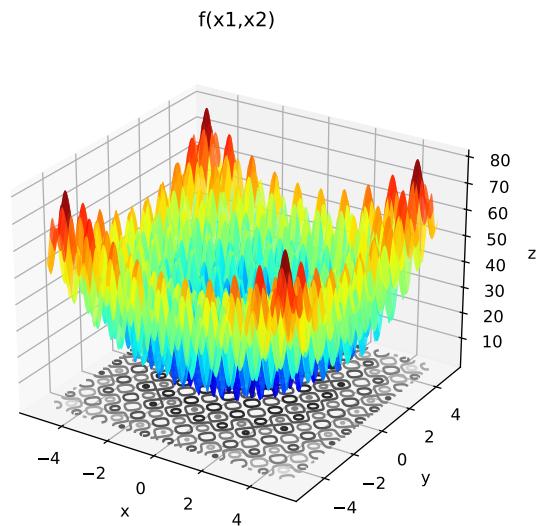
em que o domínio das variáveis são  $x_1, x_2 \in [-8, 8]$ . A presente função possui como gráfico:



4. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = (x_1^2 - 10 \cdot \cos(2\pi x_1) + 10) + (x_2^2 - 10 \cdot \cos(2\pi x_2) + 10)$$

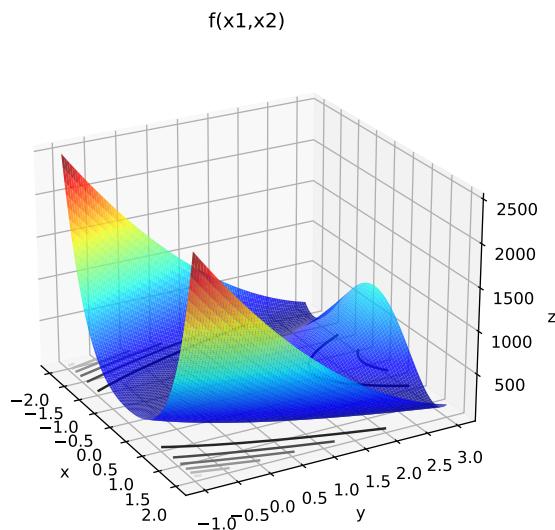
em que o domínio das variáveis são  $x_1, x_2 \in [-5.12, 5.12]$ . A presente função possui como gráfico:



5. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = (x_1 - 1)^2 + 100 \cdot (x_2 - x_1^2)^2$$

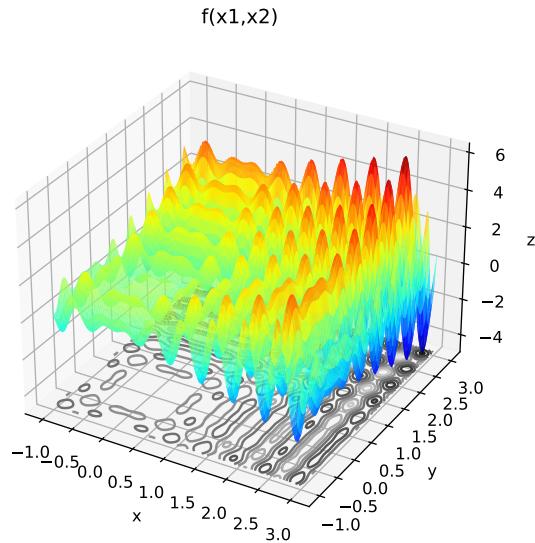
em que o domínio das variáveis são  $x_1 \in [-2, 2]$  e  $x_2 \in [-1, 3]$ . A presente função possui como gráfico:



6. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **máximo** da função:

$$f(x_1, x_2) = x_1 \cdot \sin(4\pi x_1) - x_2 \cdot \sin(4\pi x_2 + \pi) + 1$$

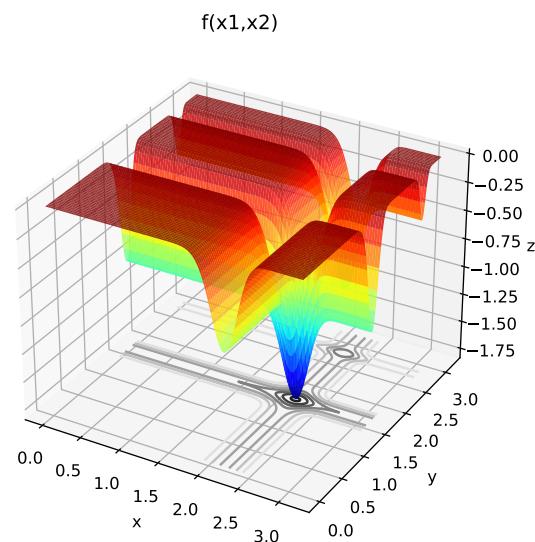
em que o domínio das variáveis são  $x_1 \in [-1, 3]$  e  $x_2 \in [-1, 3]$ . A presente função possui como gráfico:



7. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = -\sin(x_1) \cdot \sin(x_1^2/\pi)^{2-10} - \sin(x_2) \sin(2x_2^2/\pi)^{2-10}$$

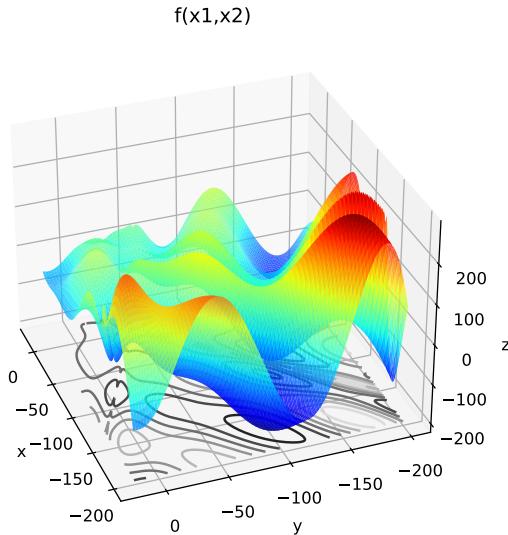
em que o domínio das variáveis são  $x_1 \in [0, \pi]$  e  $x_2 \in [0, \pi]$ . A presente função possui como gráfico:



8. Considere as variáveis independentes  $\mathbf{x} = (x_1, x_2)$ , encontre o valor **mínimo** da função:

$$f(x_1, x_2) = -(x_2 + 47) \cdot \sin(\sqrt{|x_1/2 + (x_2 + 47)|}) - x_1 \cdot \sin(\sqrt{|x_1 - (x_2 + 47)|})$$

em que o domínio das variáveis são  $x_1 \in [-200, 20]$  e  $x_2 \in [-200, 20]$ . A presente função possui como gráfico:



## 2) Problema de domínio discreto.

Para a segunda seção do presente trabalho, pede-se que se resolva dois problemas de domínio discreto: problema do Caixeiro-viajante e o problema das 8 damas. A solução destes deve ser realizada utilizando um projeto de Algoritmo Genético.

### 2.1) Problema das 8 rainhas.

Como uma contextualização inicial, faz sentido uma breve explicação sobre do que se tratam os dois problemas. O primeiro, trata-se de posicionar 8 rainhas em um tabuleiro de xadrez, de modo que nenhuma consiga atacar as outras. O problema foi proposto por Max Bezzel (1848) e Franz Nauck (1850) publicou sua primeira solução, com a extensão do quebra-cabeça com  $n$  rainhas. O problema de encontrar as soluções para o problema de 8 rainhas pode ser custoso (computacionalmente falando), pois, existem 4.426.165.368 arranjos diferentes das 8 rainhas em um tabuleiro  $8 \times 8$ . É possível reduzir os custos computacionais, ao aplicar uma regra simples e inicial ao posicionar uma rainha por coluna (reduzindo assim a  $16.777.216$  arranjos  $8^8$ ). Apesar do que já foi contextualizado, o problema possui 92 soluções distintas.

Para resolver o problema das 8 rainhas utilizando um algoritmo genético, é necessário a explicação de alguns detalhes sobre o problema. Inicialmente, considere que o tabuleiro do jogo possui enumeração de suas colunas, crescentemente, da esquerda para a direita. As enumerações das linhas são realizadas de maneira decrescente, de cima para baixo, como pode ser visualizado no exemplo exibido pela Figura 1. Em seguida, deve-se posicionar as rainhas de modo que cada rainha tenha a sua coluna (diminuindo a complexidade da busca). Assim, cada indivíduo da população possui como cromossomo, um vetor que representa cada gene desse indivíduo. Para o presente problema, cada cromossomo possui 8 genes, indicando o número da linha em que aquela rainha se encontra na  $p$ -ésima coluna. Ou seja, como um exemplo, pode-se verificar na Figura 1 que cada índice dessa sequência cromossômica representa a coluna que a rainha está e o valor no vetor, é a linha que a rainha está posicionada. Ou seja, para a sequência 51426147, a primeira rainha (coluna 1) está na linha 5 e a última rainha (coluna 8) está na linha 7.

Deve-se entender que é necessário projetar a função de aptidão. Neste caso, esta pode ser baseada na quantidade de pares de rainhas que estão se atacando. Há neste problema um total de 28 pares de rainhas diferentes que podem se atacar. Uma função de aptidão interessante, pode ser  $\Psi(\mathbf{x}) = 28 - h(\mathbf{x})$ , em que  $h(\cdot)$  é o número de pares atacantes para a solução (que é um indivíduo),  $\mathbf{x}$ . Dito isso, este é um problema de minimização ou maximização da função  $\Psi$ ?

Assim, para o projeto do algoritmo genético que resolve tal problema, faça:

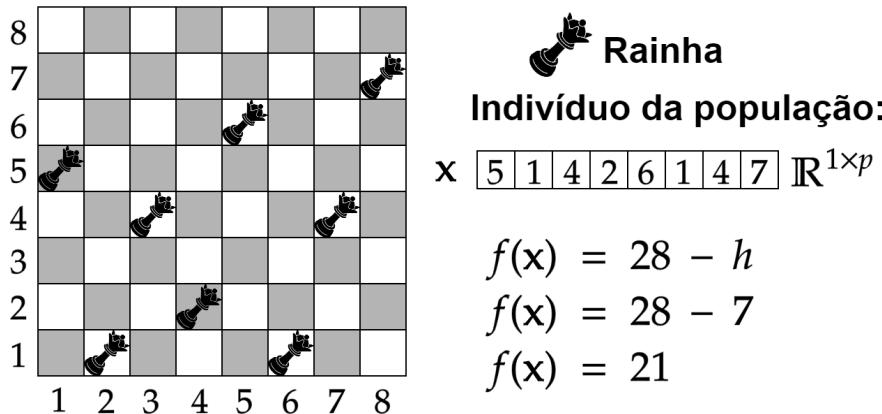


Figura 1: Problema das 8-rainhas.

1. Faça a definição da quantidade  $N$  de indivíduos em uma população e quantidade máxima de gerações.
2. Projete o operador de **seleção**, baseado no método da roleta.
3. Na etapa de **recombinação**, escolha um entre os seguintes operadores: **Recombinação de um ponto** ou **Recombinação de dois pontos**. A probabilidade de recombinação nesta etapa deve ser entre  $85 < p_c < 95\%$ .
4. Na prole gerada, deve-se aplicar a mutação com probabilidade de 1% (neste caso, é interessante avaliar os diferentes procedimentos exibidos).
5. O algoritmo deve parar quando atingir o máximo número de gerações ou quando a função custo atingir seu valor ótimo.

De posse do primeiro resultado, aplique o seu projeto de algoritmo genético para executar enquanto as 92 soluções diferentes não forem encontradas. Avalie o custo computacional desta etapa.

## 2.2) Problema da caixeiro-viajante.

O problema do caixeiro viajante, é um problema de logística, em que há uma lista de cidades a serem visitadas e uma rota a ser desempenhada. Este problema é baseado na seguinte pergunta: Sejam cidades a serem visitadas e suas distâncias em pares, qual é a menor rota para visitar todas as cidades e retornar para a cidade origem? Este é um problema clássico de busca/otimização combinatória, e sua complexidade depende da quantidade de cidades existentes na lista. Pode-se pensar que é possível encontrar a solução do problema ao listar todas as possíveis combinações e compará-las umas com as outras. Contudo, pode não ser viável resolver este problema desta maneira, tendo em vista que com apenas 20 cidades, o número de rotas possíveis é  $20!$ . Assim, cabe a utilização de um algoritmo genético para solucionar tal problema.

Para o presente trabalho, considere que o contexto em que se encaixa o problema do caixeiro viajante é o seguinte: Um drone precisa realizar acesso a pontos em um espaço tridimensional e retornar à sua origem. Os pontos que o drone deve acessar, são gerados de maneira aleatória, e para fins didáticos um exemplo desses posicionamentos é exibido na Figura 2.

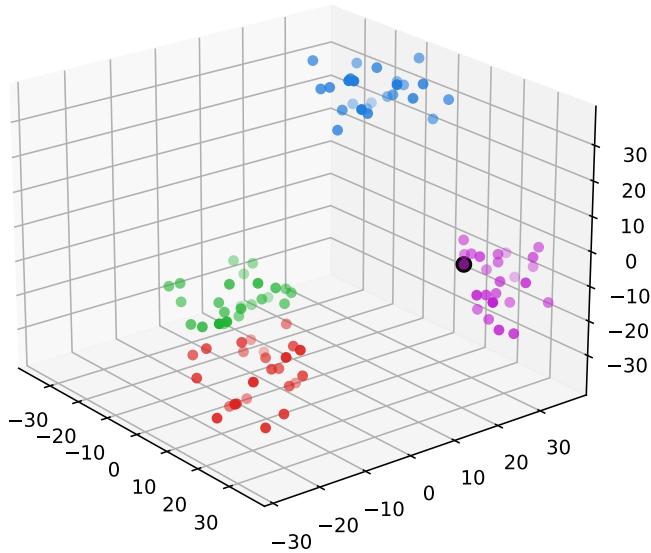


Figura 2: Problema das 8-rainhas.

Para o algoritmo genético, considere que cada indivíduo da população, são rotas que podem ser seguidas. Assim, cada estrutura cromossômica do problema é composta por  $k$  genes representando cada um dos pontos a serem visitados. Pode-se exemplificar o problema a ser resolvido, utilizando um exemplo em duas dimensões como se segue:

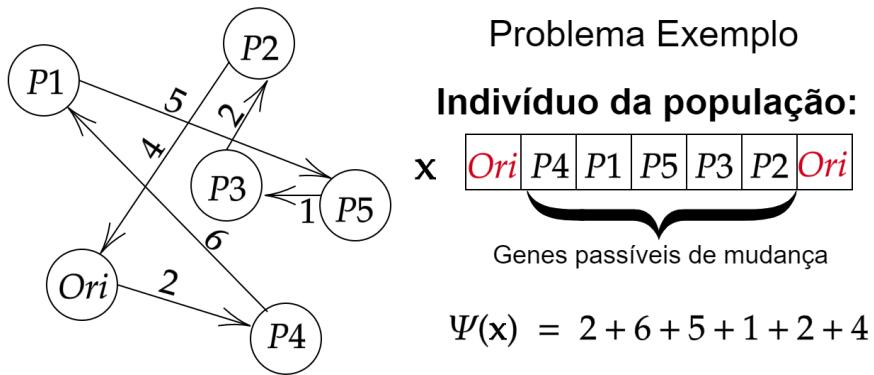


Figura 3: Problema do caixeleiro viajante.

Assim, de posse do algoritmo base presente na seção de anexos, faça o que se pede:

1. Faça a definição de quantos pontos devem ser gerados por região. Escolha um valor  $30 < N_{\text{Pontos}} < 60$ .
2. Faça a definição da quantidade  $N$  de indivíduos em uma população e quantidade máxima de gerações.
3. Projete o operador de **seleção**, baseado no método do torneio.
4. Na etapa de **recombinação**, como este trata-se de um problema de combinatória, não pode haver pontos repetidos na sequência cromossômica. Desta maneira, pede-se que desenvolva uma variação do operador de recombinação de dois pontos. Assim, cada seção selecionada de modo aleatório deve ser propagada nos filhos e em seguida, a sequência genética do filho deve ser completada com os demais pontos sem repetição.

5. Na prole gerada, deve-se aplicar a mutação com probabilidade de 1%. Para o problema do caixeiro viajante, deve-se aplicar uma mutação que faz a troca de um gene por outro de uma mesma sequência cromossômica.
6. O algoritmo deve parar quando atingir o máximo número de gerações ou quando a função custo atingir seu valor ótimo aceitável (de acordo com a regra descrita no slide 31/61).
7. Faça uma análise se de qual é a moda de gerações para obter uma solução aceitável. Além disso, analise se é necessário incluir um operador de elitismo em um grupo  $N_e$  de indivíduos.

## 5) Relatório.

Além das implementações, o presente trabalho deve ser entregue em modelo de relatório. Este deve possuir as características descritas nos slides de apresentação do curso. Desta maneira, deve possuir:

1. Título.
2. Resumo.
3. Introdução.
4. Fundamentação Teórica (Revisão Bibliográfica).
5. Metodologia.
6. Resultados.
7. Conclusões.
8. Referências.

O modelo para trabalho pode ser encontrado neste [LINK](#)

## 6) Observações.

- Obs1: O envio das implementações é **obrigatório**. Caso a equipe não realize esta entrega, será atribuído nota **zero** para os respectivos alunos.
- Obs2: A data estipulada para entrega do trabalho, também é um critério avaliativo. Assim, caso haja atraso na entrega do trabalho, será aplicada: **de 00:15h até 24h: penalidade de 20% ; 24:15h até 48h: penalidade de 40% ; acima de 48h: penalização máxima (100%)**.
- Ob3: Os trabalhos e implementações serão enviadas a um software anti-plágio. Qualquer caracterização de plágio ocasionará em nota zero para ambas equipes.
- Obs4: Para o presente trabalho, não será permitido o uso de bibliotecas que tenham as implementações prontas dos modelos Perceptron de qualquer algoritmo requisitado no trabalho. Caso sua equipe faça o uso de tais bibliotecas, serão descontadas as pontuações proporcionais.

## Anexos.

O primeiro anexo é um código auxiliar para a primeira parte do presente trabalho. Neste algoritmo, há um exemplo de como plotar o gráfico de uma função contínua, bem como um exemplo de posicionar um candidato neste gráfico tridimensional.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x1,x2):
5     return (x1**2+x2**2)
6
7 x1 = np.linspace(-100,100,1000)
8 X1,X2 = np.meshgrid(x1,x1)
9 Y = f(X1,X2)
10

```

```
11
12 x1_cand,x2_cand=50,50
13 f_cand = f(x1_cand,x2_cand)
14
15 fig = plt.figure()
16 ax = fig.add_subplot(projection='3d')
17 ax.plot_surface(X1,X2,Y,rstride=10,cstride=10,alpha=0.6,cmap='jet')
18 ax.scatter(x1_cand,x2_cand,f_cand,marker='x',s=90,linewidth=3,color='red')
19
20 ax.set_xlabel('x')
21 ax.set_ylabel('y')
22 ax.set_zlabel('z')
23 ax.set_title('f(x1,x2)')
24 plt.tight_layout()
25 plt.show()
```