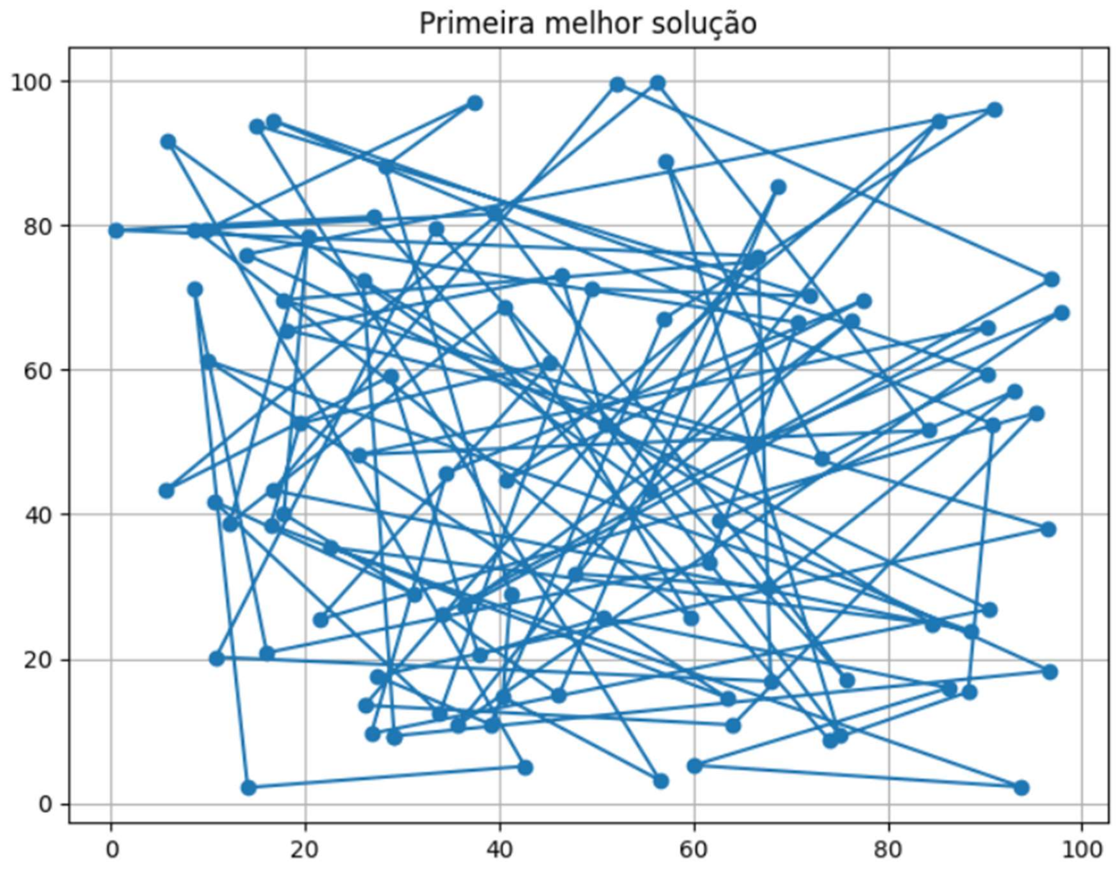
UFPR – Universidade Federal do Paraná
Setor de Educação Profissional e Tecnológica
Especialização em Inteligência Artificial Aplicada
Disciplina: Tópicos de IA – Prof. Dr. Dieval Guizelini

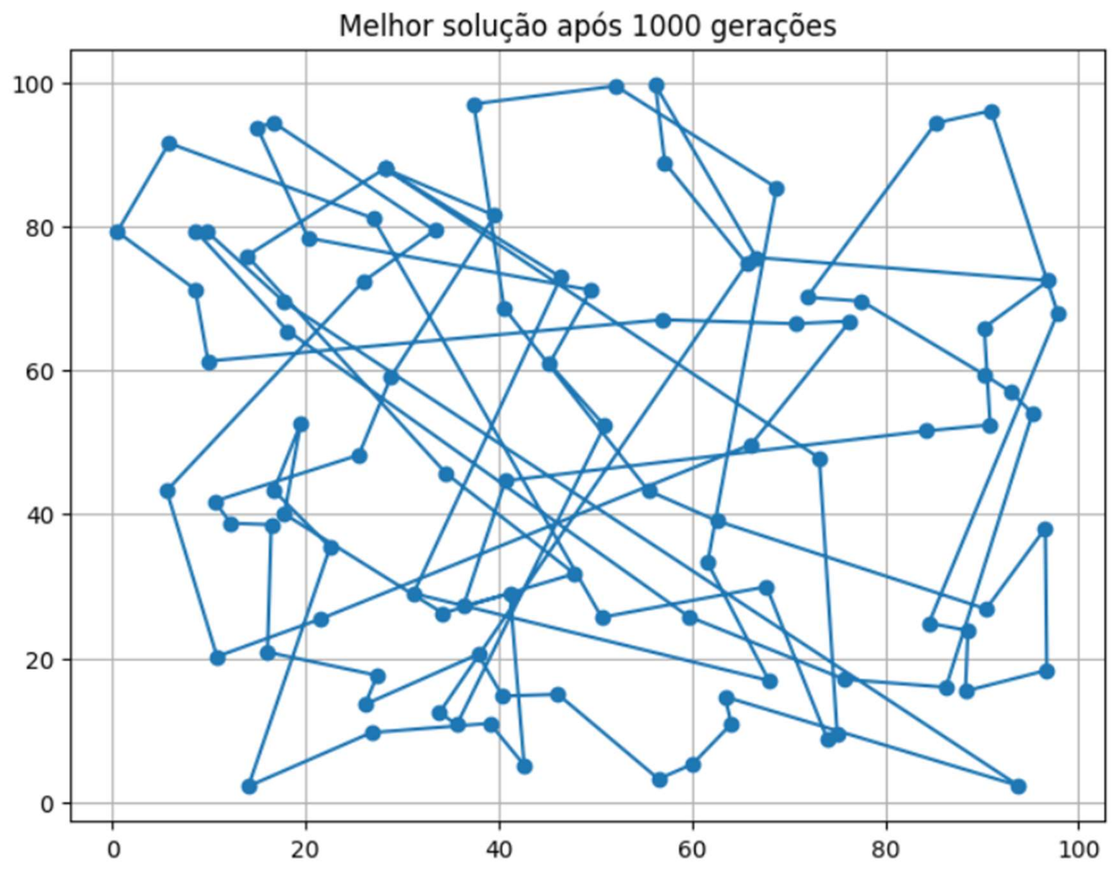
Alunos da equipe: Paulo Sergio Herval Silva Junior, Pedro de Sousa Alves Graça, Matheus Eduardo de Arazão e Anderson Felipe de Paiva

Algoritmo Genético:

```python
import numpy as np
import matplotlib.pyplot as plt
import random

# Parâmetros
NUM_CITIES = 100
POP_SIZE = 100
GENERATIONS = 1000
MUTATION_RATE = 0.01

cities = np.random.rand(NUM_CITIES, 2) * 100

def total_distance(path, cities):
    coords = cities[path]
    return np.sum(np.linalg.norm(np.diff(coords, axis=0), axis=1))

def create_individual(start_city=0):
    individual = list(range(1, NUM_CITIES))
    random.shuffle(individual)
    return [start_city] + individual + [start_city]

def create_population():
    return [create_individual() for _ in range(POP_SIZE)]

def order_crossover(parent1, parent2):
    start, end = sorted(random.sample(range(1, NUM_CITIES), 2))
    child = [-1] * NUM_CITIES
    child[start:end] = parent1[start:end]
    p2_index = 1
    for i in range(1, NUM_CITIES):
        if child[i] == -1:
            while parent2[p2_index] in child:
                p2_index += 1
            child[i] = parent2[p2_index]
    return [child[0]] + child[1:] + [child[0]]

def mutate(individual):
    if random.random() < MUTATION_RATE:
        idx1, idx2 = random.sample(range(1, NUM_CITIES), 2)
        individual[idx1], individual[idx2] = individual[idx2], individual[idx1]
    return individual
```

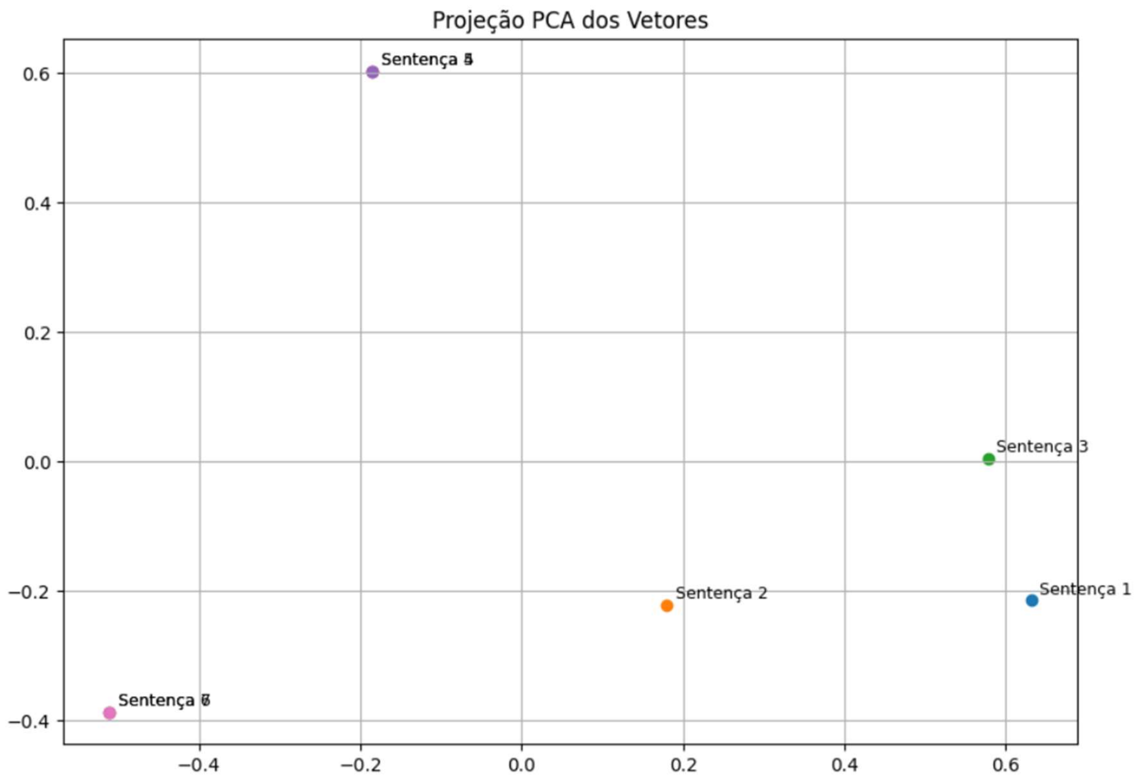
```python
def tournament_selection(population, scores, k=5):
    selected = random.sample(list(zip(population, scores)), k)
    return min(selected, key=lambda x: x[1])[0]

def evolve_population(population, cities):
    new_population = []
    scores = [total_distance(ind, cities) for ind in population]
    best_idx = np.argmin(scores)
    best_individual = population[best_idx]
    new_population.append(best_individual)

    while len(new_population) < POP_SIZE:
        parent1 = tournament_selection(population, scores)
        parent2 = tournament_selection(population, scores)
        child = order_crossover(parent1, parent2)
        child = mutate(child)
        new_population.append(child)

    return new_population, best_individual, scores[best_idx]

def plot_path(path, cities, title):
    plt.figure(figsize=(8, 6))
    x = [cities[i][0] for i in path]
    y = [cities[i][1] for i in path]
    plt.plot(x, y, marker='o')
    plt.title(title)
    plt.grid(True)
    plt.show()

population = create_population()
first_best = population[0]
first_best_distance = total_distance(first_best, cities)
best_overall = first_best
best_overall_distance = first_best_distance
history = [first_best_distance]

for generation in range(GENERATIONS):
    population, best, best_distance = evolve_population(population, cities)
    if best_distance < best_overall_distance:
        best_overall = best
        best_overall_distance = best_distance
    history.append(best_overall_distance)

plot_path(first_best, cities, "Primeira melhor solução")
plot_path(best_overall, cities, f"Melhor solução após {GENERATIONS} gerações")
```


Primeira melhor solução
0
20
40
60
80
100
0
20
40
60
80
100


Melhor solução após 1000 gerações
0
20
40
60
80
100

Compare a representação de dois modelos vetoriais:

```python
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA

sentencas = [
    "O gato dorme no tapete.",
    "O cachorro descansa no sofá.",
    "Um gato está deitado no tapete.",
    "O céu está azul e sem nuvens.",
    "Hoje o dia está ensolarado e limpo.",
    "Eu gosto de comer pizza aos sábados.",
    "Pizza é a minha comida preferida."
]


vetorizador = TfidfVectorizer()
X = vetorizador.fit_transform(sentencas)


pca = PCA(n_components=2)
X_pca = pca.fit_transform(X.toarray())


plt.figure(figsize=(10, 7))
for i, sentenca in enumerate(sentencas):
    plt.scatter(X_pca[i, 0], X_pca[i, 1])
    plt.text(X_pca[i, 0]+0.01, X_pca[i, 1]+0.01, f"Sentença {i+1}", fontsize=9)
plt.title("Projeção PCA dos Vetores")
plt.grid(True)
plt.show()
```


Projeção PCA dos Vetores
Sentença 4
Sentença 5
Sentença 3
Sentença 2
Sentença 1
Sentença 6
Sentença 7
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6