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

# Algoritmo Genético:

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

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:

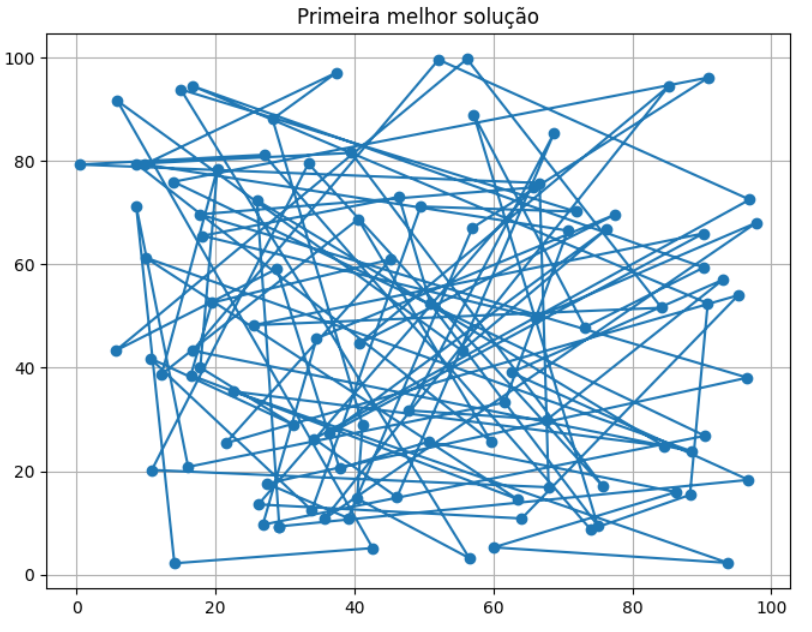
        best\_overall = best

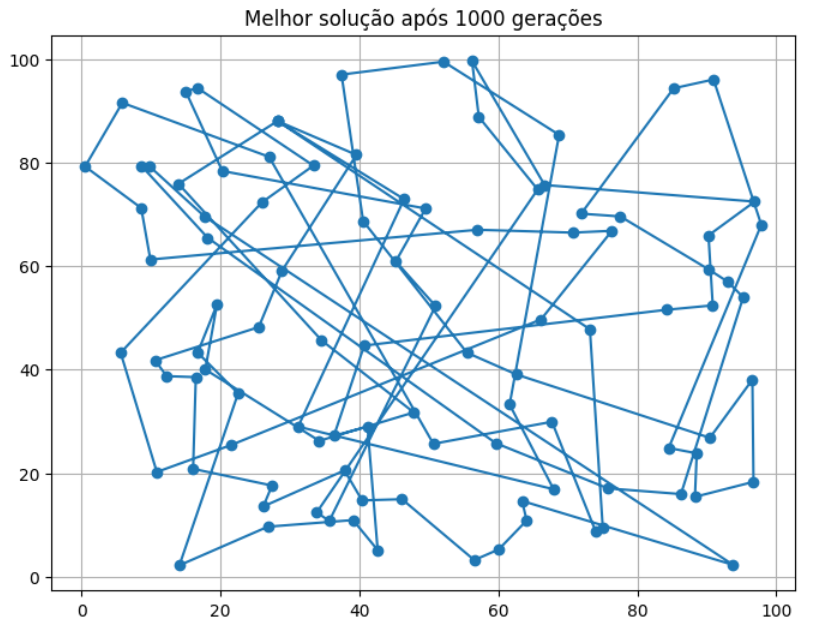
        best\_overall\_distance = best\_distance

    history.append(best\_overall\_distance)

plot\_path(first\_best, cities, "Primeira melhor solução")

plot\_path(best\_overall, cities, f"Melhor solução após {GENERATIONS} gerações")





# Compare a representação de dois modelos vetoriais:

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()

