**Código Fonte**

import random

import smtplib

from email.mime.text import MIMEText

import pandas as pd

import os

from cryptography.fernet import Fernet

from deap import base, creator, tools, algorithms

# Parâmetros de entrada

NUM\_SENSORS = 10

NUM\_AREAS = 5

CRITICITY\_LEVELS = [3, 5, 2, 4, 1]

PROB\_ATTACK = [0.1, 0.4, 0.05, 0.3, 0.15]

COST = [500, 1000, 300, 800, 200]

CAPACITY = [100, 200, 50, 150, 75]

VULNERABILITIES = [True, False, True, False, True] # Se uma área possui vulnerabilidade por falta de patch

CVSS\_SCORES = [8.5, 0, 7.2, 0, 9.3] # Pontuações CVSS para áreas vulneráveis

MAX\_COST = 2500

MAX\_CAPACITY = 400

HISTORY\_FILE = "historico\_criticidade.txt"

# Função para carregar a chave de criptografia

def load\_encryption\_key():

with open("encryption\_key.key", "rb") as key\_file:

return key\_file.read()

# Função para descriptografar as credenciais

def load\_encrypted\_credentials():

key = load\_encryption\_key()

fernet = Fernet(key)

with open("encrypted\_credentials.txt", "rb") as file:

encrypted\_sender\_email = file.readline().strip()

encrypted\_receiver\_email = file.readline().strip()

encrypted\_password = file.readline().strip()

sender\_email = fernet.decrypt(encrypted\_sender\_email).decode()

receiver\_email = fernet.decrypt(encrypted\_receiver\_email).decode()

password = fernet.decrypt(encrypted\_password).decode()

return sender\_email, receiver\_email, password

# Função para enviar e-mail de alerta usando as credenciais descriptografadas

def send\_email\_alert(subject, body):

sender\_email, receiver\_email, password = load\_encrypted\_credentials()

smtp\_server = "smtp.gmail.com"

smtp\_port = 587

msg = MIMEText(body)

msg["Subject"] = subject

msg["From"] = sender\_email

msg["To"] = receiver\_email

try:

with smtplib.SMTP(smtp\_server, smtp\_port) as server:

server.starttls()

server.login(sender\_email, password)

server.sendmail(sender\_email, receiver\_email, msg.as\_string())

print("Alerta enviado com sucesso!")

except Exception as e:

print(f"Erro ao enviar o e-mail: {e}")

# Função de fitness com avaliação de vulnerabilidade e CVSS

def eval\_allocation(individual):

total\_critic = 0

total\_cost = 0

total\_capacity = 0

for i in range(NUM\_AREAS):

if individual[i] == 1:

vulnerability\_factor = CVSS\_SCORES[i] if VULNERABILITIES[i] else 1

total\_critic += CRITICITY\_LEVELS[i] \* PROB\_ATTACK[i] \* vulnerability\_factor

total\_cost += COST[i]

total\_capacity += CAPACITY[i]

if total\_cost > MAX\_COST or total\_capacity > MAX\_CAPACITY:

return 0,

return total\_critic,

# Classificação de criticidade com base na pontuação CVSS

def classify\_criticidade\_cvss(cvss\_score):

if cvss\_score >= 9.0:

return "Crítica"

elif 7.0 <= cvss\_score < 9.0:

return "Alta"

elif 4.0 <= cvss\_score < 7.0:

return "Média"

elif 0.1 <= cvss\_score < 4.0:

return "Baixa"

else:

return "Nenhuma"

# Função para gerar relatório

def generate\_report(best\_ind, best\_fitness, criticidade, total\_cost, total\_capacity):

areas = [f"Área {i+1}" for i in range(NUM\_AREAS)]

allocation = best\_ind

vulnerability\_status = ["Vulnerável" if VULNERABILITIES[i] else "Seguro" for i in range(NUM\_AREAS)]

cvss\_classification = [classify\_criticidade\_cvss(CVSS\_SCORES[i]) for i in range(NUM\_AREAS)]

# Criação do DataFrame

data = {

"Área": areas,

"Alocação de Sensores (1 = Sim, 0 = Não)": allocation,

"Nível de Criticidade": CRITICITY\_LEVELS,

"Probabilidade de Ataque": PROB\_ATTACK,

"Custo de Implementação": COST,

"Capacidade Utilizada": CAPACITY,

"Vulnerabilidade por Falta de Patch": vulnerability\_status,

"Pontuação CVSS": CVSS\_SCORES,

"Classificação CVSS": cvss\_classification

}

df = pd.DataFrame(data)

# Informações gerais

summary = {

"Fitness": best\_fitness,

"Criticidade Global": criticidade,

"Custo Total": total\_cost,

"Capacidade Total": total\_capacity

}

summary\_df = pd.DataFrame(list(summary.items()), columns=["Parâmetro", "Valor"])

# Exporta o relatório em Excel

with pd.ExcelWriter("relatorio\_monitoramento.xlsx", engine='openpyxl') as writer:

df.to\_excel(writer, sheet\_name="Detalhes de Alocação", index=False)

summary\_df.to\_excel(writer, sheet\_name="Resumo", index=False)

print("Relatório gerado com sucesso: relatorio\_monitoramento.xlsx")

# Função para verificar o histórico de criticidade

def check\_criticidade\_history(criticidade):

if not os.path.exists(HISTORY\_FILE):

with open(HISTORY\_FILE, "w") as file:

file.write(f"{criticidade}\n")

return False

# Leitura do histórico

with open(HISTORY\_FILE, "r") as file:

history = file.readlines()

# Adicionar a criticidade do dia atual

history.append(f"{criticidade}\n")

# Manter apenas os últimos 3 dias no histórico

if len(history) > 3:

history = history[-3:]

# Salvar o histórico atualizado

with open(HISTORY\_FILE, "w") as file:

file.writelines(history)

# Verifica se os últimos 3 dias foram "Média"

if history.count("Média\n") >= 3:

return True

return False

# Criação de indivíduos e população

creator.create("FitnessMax", base.Fitness, weights=(1.0,))

creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()

toolbox.register("attr\_bool", random.randint, 0, 1)

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr\_bool, n=NUM\_AREAS)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Funções de avaliação, cruzamento e mutação

toolbox.register("evaluate", eval\_allocation)

toolbox.register("mate", tools.cxTwoPoint)

toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

toolbox.register("select", tools.selTournament, tournsize=3)

def main():

population = toolbox.population(n=50)

# Algoritmo Genético

algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=40, verbose=False)

# Avaliação da melhor solução

best\_ind = tools.selBest(population, 1)[0]

best\_fitness = best\_ind.fitness.values[0]

# Cálculo de custo e capacidade total

total\_cost = sum([COST[i] for i in range(NUM\_AREAS) if best\_ind[i] == 1])

total\_capacity = sum([CAPACITY[i] for i in range(NUM\_AREAS) if best\_ind[i] == 1])

# Classificação da criticidade global

criticidade = classify\_criticidade\_cvss(best\_fitness)

print(f"Melhor alocação de sensores: {best\_ind}")

print(f"Criticidade da melhor solução: {criticidade}")

print(f"Custo total: {total\_cost}")

print(f"Capacidade total: {total\_capacity}")

# Geração do relatório

generate\_report(best\_ind, best\_fitness, criticidade, total\_cost, total\_capacity)

# Verifica o histórico de criticidade e envia alerta se necessário

if check\_criticidade\_history(criticidade):

email\_subject = "Alerta de Criticidade Média - 3 Dias Consecutivos"

email\_body = f"""

A criticidade da alocação de sensores permaneceu em nível 'Média' por 3 dias consecutivos.

Melhor alocação de sensores: {best\_ind}

Criticidade: {criticidade}

Fitness: {best\_fitness}

Custo Total: {total\_cost}

Capacidade Total: {total\_capacity}

"""

send\_email\_alert(email\_subject, email\_body)

# Enviar e-mail de alerta para criticidade alta ou crítica

if criticidade in ["Alta", "Crítica"]:

email\_subject = "Alocação de Sensores - Alerta de Segurança"

email\_body = f"""

Melhor alocação de sensores foi encontrada:

Alocação: {best\_ind}

Criticidade: {criticidade}

Fitness: {best\_fitness}

Custo Total: {total\_cost}

Capacidade Total: {total\_capacity}

"""

send\_email\_alert(email\_subject, email\_body)

if \_\_name\_\_ == "\_\_main\_\_":

main()

1. **Explicação do Código Consolidado:**
2. **Criptografia de Credenciais**:
   * As credenciais são armazenadas de forma criptografada em um arquivo (encrypted\_credentials.txt) e são descriptografadas durante a execução com a função load\_encrypted\_credentials. A chave de criptografia é armazenada em encryption\_key.key.
3. **Função de Avaliação com CVSS**:
   * As vulnerabilidades de cada área são avaliadas com base nas pontuações CVSS (Common Vulnerability Scoring System). Isso afeta a criticidade final da solução, conforme especificado pela função eval\_allocation.
4. **Relatório**:
   * Um relatório detalhado é gerado em Excel, listando as alocações de sensores, a criticidade das áreas, a vulnerabilidade devido à falta de patches, e as pontuações CVSS.
5. **Verificação de Criticidade Persistente**:
   * Se a criticidade for "Média" por três dias consecutivos, um alerta será enviado por e-mail. Além disso, se a criticidade atingir os níveis "Alta" ou "Crítica", um alerta imediato será disparado.
6. **Algoritmo Genético**:
   * O algoritmo genético é utilizado para otimizar a alocação de sensores nas áreas críticas da infraestrutura de TI, considerando vulnerabilidades e restrições de custo e capacidade.

Explicação sobre o objetivo do trabalho

Esse algoritmo tem como objetivo otimizar a alocação de sensores de monitoramento em diferentes áreas de uma infraestrutura de TI, levando em consideração fatores como criticidade, probabilidade de ataque, custo e capacidade das áreas. Ele utiliza **algoritmos genéticos**, que são métodos de otimização inspirados no processo de evolução natural, para encontrar a melhor solução que maximiza a proteção enquanto minimiza os custos e limita a utilização de recursos.

1. **Passos do Algoritmo:**
2. **Entrada de Parâmetros**: O sistema recebe parâmetros como o número de sensores, áreas a serem monitoradas, nível de criticidade das áreas, probabilidade de ataque, custos de implementação, capacidades, e se as áreas têm vulnerabilidades ou não. Além disso, são definidos limites de custo total e capacidade total que a solução deve respeitar.
3. **Indivíduos e População**: Cada indivíduo na população representa uma possível alocação de sensores nas áreas. A alocação é representada por uma lista de valores binários (0 ou 1), onde 1 significa que um sensor será alocado na área correspondente e 0 significa que não será.
4. **Função de Fitness**: A função de fitness é usada para avaliar a qualidade de cada alocação de sensores. Ela calcula o nível de criticidade total, somando os fatores de criticidade das áreas com base em sua vulnerabilidade e probabilidade de ataque. Se o custo total ou a capacidade total da solução exceder os limites estabelecidos, a solução é penalizada (recebe uma pontuação 0).
5. **Operadores Genéticos**:
   * **Cruzamento**: Combina dois indivíduos (soluções) para criar novos indivíduos, misturando suas características.
   * **Mutação**: Muda aleatoriamente uma parte da solução para explorar novas possibilidades.
   * **Seleção**: Escolhe os melhores indivíduos com base em sua função de fitness para formar a próxima geração.
6. **Execução do Algoritmo Genético**: O algoritmo executa várias gerações, aplicando cruzamento, mutação e seleção até encontrar a melhor alocação de sensores.
7. **Relatórios e Alertas**: Após encontrar a melhor solução, o algoritmo gera um relatório detalhado com os resultados, incluindo as áreas onde os sensores foram alocados, a criticidade de cada área, os custos e as capacidades. Se for detectada uma criticidade alta ou contínua, o sistema envia alertas por e-mail para informar a equipe de segurança.
8. **Objetivo do Algoritmo:**

O objetivo principal do algoritmo é otimizar a alocação de sensores de forma a maximizar a cobertura das áreas mais críticas e vulneráveis, respeitando os limites de custo e capacidade. Com essa abordagem, é possível garantir uma cobertura eficiente da infraestrutura de TI, reduzindo o risco de ataques enquanto otimiza o uso de recursos.

1. **Ganhos da Implementação:**
2. **Otimização de Recursos**: O algoritmo aloca os sensores nas áreas que mais necessitam, garantindo o melhor uso possível dos recursos disponíveis (sensores, custo e capacidade).
3. **Redução de Riscos**: As áreas mais vulneráveis e críticas são protegidas primeiro, o que reduz a probabilidade de incidentes de segurança graves.
4. **Automação e Eficiência**: O processo é automatizado e pode ser reavaliado regularmente para garantir que a infraestrutura de TI esteja sempre bem protegida, ajustando a alocação conforme necessário.
5. **Monitoramento Contínuo**: O histórico de criticidade é monitorado, e alertas são gerados quando padrões preocupantes (como criticidade contínua) são detectados, permitindo ação rápida da equipe de segurança.

Essa implementação é altamente vantajosa para empresas que precisam maximizar a proteção de suas infraestruturas de TI com recursos limitados.

**Comparar os resultados obtidos com o código elaborado com métodos de solução convencionais**

Para comparar os resultados obtidos pelo algoritmo genético com métodos de solução convencionais, você pode implementar abordagens mais simples, como a Força Bruta e o Método Guloso (também conhecido como Heurística Gananciosa). Essas abordagens fornecem soluções que podem ser comparadas em termos de eficiência, tempo de execução e qualidade da solução.

Aqui está uma explicação de como implementar essas técnicas para realizar a comparação:

# Implementação do Método de Força Bruta

A abordagem de Força Bruta tenta todas as combinações possíveis de alocação de sensores nas áreas, avalia o custo e a capacidade de cada combinação e encontra a melhor solução possível. Esse método garante a solução ótima, mas é ineficiente para problemas com muitas áreas.

import itertools

def brute\_force\_solution():

best\_solution = None

best\_fitness = -1

# Gera todas as combinações possíveis de alocação de sensores (0 ou 1)

for combination in itertools.product([0, 1], repeat=NUM\_AREAS):

total\_critic = 0

total\_cost = 0

total\_capacity = 0

for i in range(NUM\_AREAS):

if combination[i] == 1:

vulnerability\_factor = CVSS\_SCORES[i] if VULNERABILITIES[i] else 1

total\_critic += CRITICITY\_LEVELS[i] \* PROB\_ATTACK[i] \* vulnerability\_factor

total\_cost += COST[i]

total\_capacity += CAPACITY[i]

# Verifica se a solução respeita as restrições de custo e capacidade

if total\_cost <= MAX\_COST and total\_capacity <= MAX\_CAPACITY:

if total\_critic > best\_fitness:

best\_fitness = total\_critic

best\_solution = combination

return best\_solution, best\_fitness

# Chama a função de força bruta para comparação

brute\_solution, brute\_fitness = brute\_force\_solution()

print(f"Melhor solução (Força Bruta): {brute\_solution}, Fitness: {brute\_fitness}")

# Implementação do Método Guloso (Heurística Gananciosa)

No Método Guloso, as áreas são priorizadas com base em um critério, como a razão entre criticidade e custo, e os sensores são alocados nas áreas que têm o maior valor dessa razão, até que os limites de custo e capacidade sejam atingidos. Essa abordagem é rápida, mas nem sempre encontra a solução ótima.

def greedy\_solution():

# Calcula a razão criticidade/custo para cada área

ratios = [(CRITICITY\_LEVELS[i] \* PROB\_ATTACK[i] / COST[i], i) for i in range(NUM\_AREAS)]

# Ordena as áreas pela razão criticidade/custo em ordem decrescente

ratios.sort(reverse=True)

total\_critic = 0

total\_cost = 0

total\_capacity = 0

allocation = [0] \* NUM\_AREAS # Inicializa a solução com todos os sensores desativados

for ratio, i in ratios:

if total\_cost + COST[i] <= MAX\_COST and total\_capacity + CAPACITY[i] <= MAX\_CAPACITY:

allocation[i] = 1 # Aloca o sensor nessa área

vulnerability\_factor = CVSS\_SCORES[i] if VULNERABILITIES[i] else 1

total\_critic += CRITICITY\_LEVELS[i] \* PROB\_ATTACK[i] \* vulnerability\_factor

total\_cost += COST[i]

total\_capacity += CAPACITY[i]

return allocation, total\_critic

# Chama a função gulosa para comparação

greedy\_solution, greedy\_fitness = greedy\_solution()

print(f"Melhor solução (Método Guloso): {greedy\_solution}, Fitness: {greedy\_fitness}")

# Comparação dos Resultados

Agora, podemos realizar a comparação entre os métodos em termos de **fitness**, **custo total**, e **tempo de execução**. Podemos adicionar as chamadas para o **algoritmo genético**, **força bruta**, e **método guloso** em uma função única para fazer as comparações lado a lado.

import time

def compare\_methods():

# Algoritmo Genético

start\_time = time.time()

population = toolbox.population(n=50)

algorithms.eaSimple(population, toolbox, cxpb=0.5, mutpb=0.2, ngen=40, verbose=False)

best\_ind\_genetic = tools.selBest(population, 1)[0]

best\_fitness\_genetic = best\_ind\_genetic.fitness.values[0]

time\_genetic = time.time() - start\_time

# Força Bruta

start\_time = time.time()

best\_solution\_brute, best\_fitness\_brute = brute\_force\_solution()

time\_brute = time.time() - start\_time

# Método Guloso

start\_time = time.time()

best\_solution\_greedy, best\_fitness\_greedy = greedy\_solution()

time\_greedy = time.time() - start\_time

# Exibe os resultados

print("Comparação de Métodos:")

print(f"Algoritmo Genético: Melhor Fitness = {best\_fitness\_genetic}, Tempo = {time\_genetic:.4f} segundos")

print(f"Força Bruta: Melhor Fitness = {best\_fitness\_brute}, Tempo = {time\_brute:.4f} segundos")

print(f"Método Guloso: Melhor Fitness = {best\_fitness\_greedy}, Tempo = {time\_greedy:.4f} segundos")

# Chama a função de comparação

compare\_methods()

# Dados para teste

**Estrutura dos Dados:**

* **Número de Sensores**: 10
* **Número de Áreas**: 5
* **Níveis de Criticidade**: Variam de 1 a 5
* **Probabilidade de Ataque**: Variam de 0.01 a 0.5
* **Custos de Implementação**: Variam de 100 a 1000
* **Capacidades das Áreas**: Variam de 50 a 200
* **Vulnerabilidades**: True ou False
* **Pontuações CVSS**: Entre 0 e 10 (aplicável apenas para áreas vulneráveis)
* **Limites**:
  + Custo máximo permitido: 2500
  + Capacidade máxima permitida: 400

NUM\_SENSORS = 10

NUM\_AREAS = 5

CRITICITY\_LEVELS = [4, 3, 5, 2, 1]

PROB\_ATTACK = [0.2, 0.1, 0.4, 0.05, 0.3]

COST = [600, 300, 900, 150, 500]

CAPACITY = [120, 80, 200, 60, 100]

VULNERABILITIES = [True, False, True, False, True]

CVSS\_SCORES = [8.7, 0, 9.2, 0, 7.5] # Pontuação CVSS para áreas vulneráveis

MAX\_COST = 2500

MAX\_CAPACITY = 400

1. **Explicação dos Dados:**
2. **Níveis de Criticidade**:
   * Área 1: 4
   * Área 2: 3
   * Área 3: 5
   * Área 4: 2
   * Área 5: 1
3. **Probabilidade de Ataque**:
   * Área 1: 20%
   * Área 2: 10%
   * Área 3: 40%
   * Área 4: 5%
   * Área 5: 30%
4. **Custos de Implementação**:
   * Área 1: R$ 600
   * Área 2: R$ 300
   * Área 3: R$ 900
   * Área 4: R$ 150
   * Área 5: R$ 500
5. **Capacidades das Áreas**:
   * Área 1: 120
   * Área 2: 80
   * Área 3: 200
   * Área 4: 60
   * Área 5: 100
6. **Vulnerabilidades**:
   * Área 1: Vulnerável (True)
   * Área 2: Não vulnerável (False)
   * Área 3: Vulnerável (True)
   * Área 4: Não vulnerável (False)
   * Área 5: Vulnerável (True)
7. **Pontuações CVSS**:
   * Área 1: 8.7
   * Área 2: 0 (Não vulnerável)
   * Área 3: 9.2
   * Área 4: 0 (Não vulnerável)
   * Área 5: 7.5
8. **Limites**:
   * **Custo máximo permitido**: R$ 2500
   * **Capacidade máxima permitida**: 400
9. **Como Usar os Dados:**

Basta substituir as variáveis no seu código original pelos dados fictícios criados acima. Isso garantirá que você esteja usando um conjunto de dados realista e completo para testar o algoritmo de alocação de sensores.

# Substitua essas variáveis pelos dados fictícios criados

CRITICITY\_LEVELS = [4, 3, 5, 2, 1]

PROB\_ATTACK = [0.2, 0.1, 0.4, 0.05, 0.3]

COST = [600, 300, 900, 150, 500]

CAPACITY = [120, 80, 200, 60, 100]

VULNERABILITIES = [True, False, True, False, True]

CVSS\_SCORES = [8.7, 0, 9.2, 0, 7.5]

MAX\_COST = 2500

MAX\_CAPACITY = 400

# Explicação dos Métodos e Comparação

* **Força Bruta**: Tenta todas as combinações possíveis. Garante a solução ótima, mas pode ser muito lento em cenários maiores devido à explosão combinatória.
* **Método Guloso**: Prioriza as áreas mais críticas com base na razão criticidade/custo. É rápido, mas não garante uma solução ótima.
* **Algoritmo Genético**: Tenta encontrar uma solução ótima por meio de simulação de evolução natural. É uma abordagem probabilística, então pode encontrar uma boa solução (embora não sempre ótima) de forma mais eficiente do que a força bruta.

# Ganhos da Comparação:

* **Qualidade da Solução**: Comparar os valores de fitness obtidos por cada método permitirá avaliar qual abordagem oferece as melhores soluções.
* **Eficiência Computacional**: O tempo de execução é um fator crucial, especialmente em ambientes com recursos limitados ou quando a solução precisa ser rápida.
* **Escalabilidade**: O algoritmo genético e o método guloso podem lidar melhor com o aumento da complexidade (número de áreas, sensores etc.) do que a força bruta.

Com essa comparação, é possível determinar se os algoritmos genéticos trazem vantagens em relação aos métodos convencionais, tanto em termos de qualidade da solução quanto de tempo de execução, dependendo do tamanho e da complexidade do problema.