ECOI22 - Inteligência Artificial - Resolução do problema de Voos Aéreos através de GA

J. P. V. Moreira[1], W. G. L. Rabelo[2]
[1,2]Institute of Technological Sciences, Universidade Federal de Itajubá, Itabira, Minas Gerais, Brazil
joaopvm_21@unifei.edu.br, wesleygrabelo@unifei.edu.br

***Abstract*—Este artigo apresenta a aplicação de um algoritmo genético para otimizar reservas de voos, focando na minimização tanto do custo total quanto dos tempos de espera para uma viagem de ida e volta envolvendo múltiplos destinos. O problema é formulado para selecionar a melhor combinação de voos, considerando diversos fatores como horários de partida, horários de chegada e preços. O algoritmo emprega operadores genéticos, incluindo seleção por torneio, crossover de ponto único e mutação, para evoluir uma população de soluções potenciais ao longo de gerações sucessivas.**

***Keywords*—Algoritmo Genético, Otimização de Voos, Otimização Combinatória, Ajuste de Parâmetros, Computação Evolucionária, Python.**

I. Introduction

Algoritmos genéticos (GAs) são uma classe de métodos heurísticos de busca e otimização inspirados nos princípios da seleção natural e da evolução biológica, conforme proposto por Charles Darwin. Eles foram inicialmente introduzidos por John Holland na década de 1970 e têm sido amplamente utilizados para resolver problemas complexos de otimização em diversas áreas, incluindo engenharia, economia, bioinformática e inteligência artificial .

O funcionamento básico de um algoritmo genético envolve a manutenção de uma população de soluções candidatas, chamadas indivíduos, que evoluem ao longo de várias gerações. Cada indivíduo na população é representado por um cromossomo, que pode ser codificado como uma string de bits, números ou outros formatos, dependendo do problema específico. O processo evolutivo nos AGs é guiado por três principais operadores genéticos: seleção, crossover e mutação.

Os algoritmos genéticos têm várias vantagens, incluindo a capacidade de lidar com espaços de busca grandes e complexos, e a flexibilidade para serem aplicados a uma ampla gama de problemas. No entanto, a escolha dos parâmetros do AG, como tamanho da população, taxas de crossover e mutação, e o método de seleção, é crucial para o seu desempenho e pode exigir um ajuste fino baseado na natureza específica do problema .

A popularidade dos AGs se deve, em parte, à sua simplicidade e à capacidade de encontrar boas soluções em um tempo computacional razoável. Eles têm sido aplicados com sucesso em problemas de otimização combinatória, planejamento, aprendizado de máquina, e até na arte generativa, demonstrando sua versatilidade e eficácia .

Neste trabalho, exploramos a aplicação de algoritmos genéticos para a otimização de reservas de voos, um problema combinatório clássico que envolve a minimização dos custos e dos tempos de espera. Apresentamos um estudo detalhado dos parâmetros do AG e suas influências no desempenho, fornecendo uma análise compreensiva baseada em experimentos rigorosos.

II. Materials and Methods

Esta seção descreve a metodologia proposta, incluindo uma breve descrição do GA, os parâmetros usados para validar o algoritmo, fornece uma visão geral do conjunto de dados e os detalhes sobre como foi feito a otimização dos voos.

A. Dataset

Para esse algoritmos, utilizamos um conjunto de dados de vários voos com seus respectivos preços como espaço de busca que pode ser encontrada em *https://drive.google.com/file/d/14d5gwHdxN6LZjlkXRM-pd29vLgON7aI2/view*.

B. Functions

Começando com a função para ler o conjunto de dados de voos de ida e volta, ela recebe um parâmetro que é o caminho para o arquivo a ser lido. Em seguida, ela abre o arquivo em modo de leitura e itera sobre cada linha do arquivo. Se a linha contiver exatamente 5 partes (ou seja, se for no formato esperado), ela extrai os valores correspondentes para origem, destino, partida, chegada e preco, e cria um dicionário com esses valores. Em seguida, adiciona esse dicionário à lista voos e retorna a lista completa de voos lidos do arquivo.

```python
def ler_voos_arquivo(caminho_arquivo):
    voos = []
    with open(caminho_arquivo, 'r') as file:
        for linha in file:
            partes = linha.strip().split(',')
            if len(partes) == 5:
                origem, destino, partida, chegada, preco = partes
                voo = {
                    "origem": origem,
                    "destino": destino,
                    "partida": partida,
                    "chegada": chegada,
                    "preco": int(preco)
                }
                voos.append(voo)
    return voos
```

Fig. 1. Função para ler o conjunto de dados de voos de ida e volta.

Agora a função fitness se baseia Esta função calcula o fitness (aptidão) de um indivíduo, que é uma combinação do custo total dos voos e o tempo total de espera calculado pela função de calcular tempo de espera. O fitness é usado para comparar indivíduos e selecionar os mais adequados para reprodução na busca pela solução ótima do problema. Quanto menor o valor retornado, melhor é o indivíduo em termos de custo e tempo de espera. Dentro da função fitness foi feita uma ”subfunção” para obter o valor do tempo de espera e, esta função calcula o tempo total de espera entre os voos de ida e volta de um indivíduo (uma possível solução). Ela itera sobre os índices pares do indivíduo, que representam os voos de ida, e para cada voo de ida, calcula o tempo de espera em Roma antes do voo de volta. Se o voo de volta partir no mesmo dia, o tempo de espera é a diferença entre a chegada em Roma e a partida; caso contrário, é a diferença entre a chegada e a meia-noite, somada à hora de partida do voo de volta.

```python
def hora_para_minutos(hora):
    horas, minutos = map(int, hora.split(":"))
    return horas * 60 + minutos

def calcular_tempo_espera(individuo):
    tempo_espera_total = 0

    for i in range(0, len(individuo), 2):
        voo_ida = individuo[i]
        voo_volta = individuo[i + 1]

        chegada_roma = hora_para_minutos(voo_ida['chegada'])
        partida_volta = hora_para_minutos(voo_volta['partida'])

        # Tempo de espera em Roma antes do voo de volta
        if partida_volta > chegada_roma:
            tempo_espera_total += partida_volta - chegada_roma
        else:
            # Se o voo de volta parte no dia seguinte
            tempo_espera_total += (24 * 60 - chegada_roma) + partida_volta

    return tempo_espera_total

def calcular_fitness(individuo):
    custo_total = sum(voo['preco'] for voo in individuo)
    tempo_espera_total = calcular_tempo_espera(individuo)
    return custo_total + tempo_espera_total  # Quanto menor, melhor
```

Fig. 2. Função fitness para cálculo do melhor indivíduo.

A função torneio implementa um torneio de seleção para escolher indivíduos da população para reprodução em um algoritmo genético. A função seleciona aleatoriamente k indivíduos da população usando random.sample(populacao, k). Em seguida, ela retorna o melhor indivíduo dos selecionados com base em seu fitness, que é calculado pela função calcular fitness. O parâmetro key=lambda ind: calcular fitness(ind) é uma função lambda(uma função anônima como () =¿ em js, por exemplo) que define a chave de ordenação para a função min. Isso garante que o indivíduo retornado seja o que tem o menor valor de fitness, ou seja, o mais adequado em termos de custo e tempo de espera.

A função de crossover começa escolhendo aleatoriamente um ponto de corte entre 1 e o comprimento de um dos pais menos 1 (para garantir que haja material genético para trocar). O ponto de corte define onde a troca de material genético ocorrerá.

Em seguida, a função cria dois filhos combinando partes dos pais. O filho 1 é formado pela primeira parte de pai 1 (do início

```python
def torneio(populacao, k):
    selecionados = random.sample(populacao, k)
    return min(selecionados, key=lambda ind: calcular_fitness(ind))

```

Fig. 3. Função de torneio.

até o ponto de corte) concatenada com a segunda parte de pai 2 (do ponto de corte até o final). O filho 2 é formado de forma análoga, mas com as partes dos pais invertidas e por fim, a função retorna os dois filhos gerados.

```python
def crossover_um_ponto(pai1, pai2):
    ponto_corte = random.randint(1, len(pai1) - 1)
    filho1 = pai1[:ponto_corte] + pai2[ponto_corte:]
    filho2 = pai2[:ponto_corte] + pai1[ponto_corte:]
    return filho1, filho2
```

Fig. 4. Função de Crossover.

A função de mutação começa escolhendo aleatoriamente um índice de mutação entre 0 e o comprimento do indivíduo menos 1. Isso determina qual voo será mutado.

Em seguida, ela extrai a cidade de origem do voo que será mutado. Isso é usado para garantir que o novo voo escolhido tenha a mesma cidade de origem.

Então, a função seleciona aleatoriamente um novo voo válido com a mesma cidade de origem do voo que será mutado. Isso é feito usando uma list comprehension para criar uma lista de voos válidos com a mesma cidade de origem e, em seguida, escolhendo aleatoriamente um voo dessa lista.

Por fim, a função substitui o voo no índice de mutação do indivíduo pelo novo voo selecionado. Essa mutação introduz variação na população de indivíduos, ajudando a explorar diferentes soluções no espaço de busca do algoritmo genético.

```python
def mutacao_troca_voo(individuo, voos_validos):
    indice_mutacao = random.randint(0, len(individuo) - 1)
    cidade = individuo[indice_mutacao]['origem']
    novo_voo = random.choice([voo for voo in voos_validos if voo['origem'] == cidade])
    individuo[indice_mutacao] = novo_voo
```

Fig. 5. Função de Mutação.

O algoritmo genético começa inicializando uma população de indivíduos usando a função inicializar populacao com os voos válidos e o tamanho da população. Em seguida, cria uma lista historico custos para armazenar o melhor custo encontrado em cada geração. Em cada geração, a função seleciona os melhores indivíduos (elites) para preservar na próxima geração

com base na taxa de elitismo. Em seguida, ela continua a preencher a nova população até atingir o tamanho da população desejada, usando o operador de seleção por torneio, crossover de um ponto, e mutação de troca de voo. Após todas as gerações, a função retorna o melhor indivíduo encontrado e o histórico dos melhores custos encontrados em cada geração.

III. Results

Como parâmetros iniciais, foram escolhidos os seguintes valores, que são considerados valores bem comuns em um algoritmo genético.

```
# Definindo os parâmetros
tamanho_populacao = 80
numero_geracoes = 120
tamanho_torneio = 2
prob_crossover = 0.5
prob_mutacao = 0.05
taxa_elitismo = 0.05
```

Fig. 6. Parâmetros iniciais.

A partir desses parâmetros, foi obtido valores proximos de custo total das passagens de 2050 a 2300 reais, sendo o valor obtido na figura 7 2248 reais.

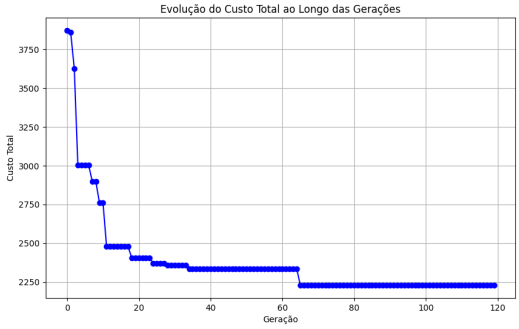

Fig. 7. Resultados iniciais.

Com o modelagem dos parâmetros e operadores genéticos, foram encontrados os seguintes parâmetros:

```
# Definindo os parâmetros
tamanho_populacao = 400
numero_geracoes = 100
tamanho_torneio = 10
prob_crossover = 0.8
prob_mutacao = 0.05
taxa_elitismo = 0.05
```

Fig. 8. Parâmetros finais.

A partir desses parâmetros, foi obtido valores proximos de custo total das passagens de 1600 a 1800 reais, sendo o valor obtido na figura 9 1628 reais.

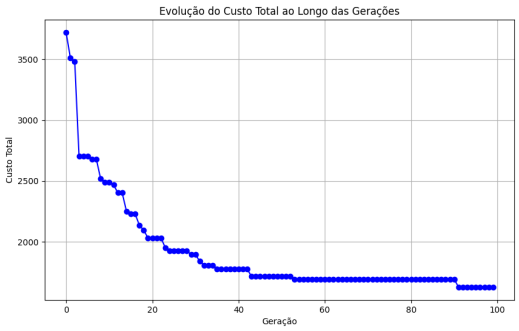

Fig. 9. Resultados finais.

Depois de executar o algoritmo genético trinta vezes, foram obtidos os seguintes valores da função fitness de custo médio, pior custo e melhor custo, assim como a média e desvio padrão dos mesmos.

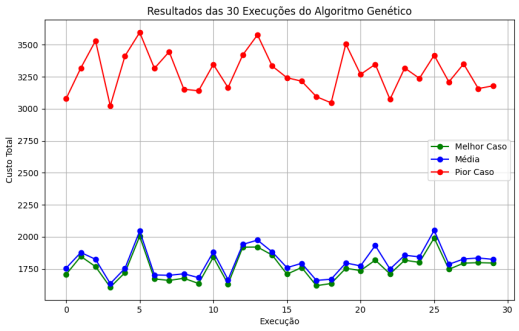

Fig. 10. Resultados das 30 Execuções do Algoritmo Genético.

TABLE I
ESTATÍSTICAS DOS CUSTOS

	Média	**Desvio Padrão**
Melhores Custos	1764.63	104.28
Custos Médios	1805.78	108.19
Piores Custos	3284.63	156.26

IV. CONCLUSION

Este estudo demonstrou a aplicação de algoritmos genéticos (AGs) para otimizar reservas de voos, com foco na minimização do custo total e do tempo de espera para viagens de ida e volta envolvendo múltiplos destinos. Os experimentos iniciais, usando parâmetros comuns do GA, resultaram em custos totais de ingressos variando de 2.050 a 2.300 reais. Após o ajuste fino dos parâmetros, os custos foram reduzidos ainda mais para uma faixa de 1.600 a 1.800 reais. O melhor custo alcançado nos experimentos foi de 1.628 reais.

Ao longo de 30 execuções do AG, o algoritmo encontrou consistentemente boas soluções, com os custos médio, melhor e pior sendo 1.764,63, 1.805,78 e 3.284,63 reais, respectivamente. Os resultados sublinham o potencial dos AGs na resolução de problemas de otimização combinatória, como a reserva de voos, onde tanto a eficiência de custos como a redução dos tempos de espera são cruciais.

Trabalhos futuros poderiam explorar a integração de restrições e variáveis adicionais, como preferências dos passageiros e preços dinâmicos, para melhorar ainda mais o processo de otimização. Além disso, comparar o desempenho dos AGs com outras abordagens heurísticas ou meta-heurísticas poderia fornecer insights mais profundos sobre os métodos mais eficazes para este domínio do problema.

REFERENCES

[1] Sandro Carvalho Izidoro. Introdução à meta-heurística genetic algorithms. *Universidade Federal de Itajubá - UNIFEI Campus Itabira*, 2022.

[2] J.H. Holland. Adaptation in natural and artificial systems. *University of Michigan Press*, 1975.

[3] M Mitchell. An introduction to genetic algorithms. *MIT Press*, 1998).

[1] [2] [3]