Otimização de Itinerários de Voo utilizando Algoritmo Genético

F. W. P. Coelho* and T. L. F. Carvalho†
*Instituto de Ciênicas Tecnológicas, Universidade Federal de Itajubá, Itabira, Minas Gerais, Brasil
felipewallacepc@unifei.edu.br, thiago.carvalho@unifei.edu.br

***Resumo*—O presente artigo visa utilizar um algoritmo genético, implementado na linguagem *Python*, para otimizar os valores das passagens e os horários dos voos dos pesquisadores participantes do congresso sobre o aquecimento global em Roma. Através desse algoritmo, busca-se encontrar a melhor combinação de voos que minimize os custos das passagens e reduza o tempo de espera nos aeroportos, proporcionando uma logística eficiente e econômica para os participantes.**

***Keywords*—Algoritmo Genético, Otimização, Preço, Tempo de espera, *Python*.**

I. Introdução

Problemas de otimização estão presentes em diversos domínios e envolvem a busca pela melhor solução possível em um espaço de busca. Esses problemas surgem quando há a necessidade de encontrar valores ótimos para determinadas variáveis, considerando restrições e objetivos específicos. A otimização abrange desde simples problemas matemáticos até desafios complexos em áreas como engenharia, economia, logística e ciência da computação. O objetivo é maximizar ou minimizar uma função objetivo, levando em conta um conjunto de restrições e condições.

A otimização de problemas pode enfrentar diversas dificuldades, desde a complexidade computacional até a presença de múltiplos objetivos e restrições conflitantes. Alguns desafios comuns incluem a dimensionalidade do problema, onde o aumento do número de variáveis aumenta exponencialmente o espaço de busca, tornando a busca pela solução ótima mais difícil.

Dentre essas situações que envolvem otimização, podemos citar o caso da realização de um congresso sobre as questões relacionadas ao aquecimento global em Roma, na Itália. Durante o evento, uma mesa-redonda contará com a participação de seis pesquisadores europeus. Cada pesquisador precisa viajar de sua cidade de origem para Roma, onde o congresso será realizado, como exemplificado na figura 1. Após a chegada, eles precisam se deslocar do aeroporto até o local do congresso utilizando uma van disponibilizada pelos organizadores dessa conferência. Após o término do evento, eles precisam retornar ao aeroporto utilizando a mesma van para pegar os voos de volta para seus países de origem.

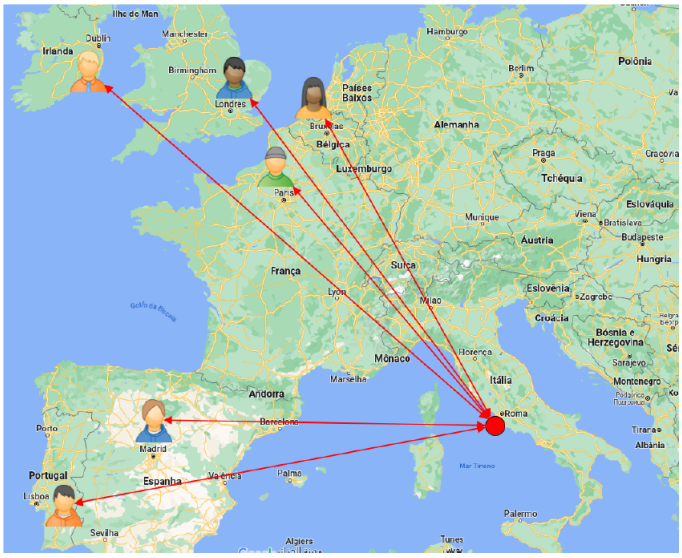


Figura 1: Pesquisadores e suas respectivas cidades com destino a Roma. Fonte: Izidoro, S. (2023)

Dois problemas interessantes são encontrar preços de passagens aéreas mais baixos e a redução no tempo de espera dos pesquisadores nos aeroportos. Para resolvê-los propõe-se a utilização de um Algoritmo Genético (GA), que é uma técnica inspirada no processo de evolução natural. Os GA's foram introduzidos em 1975 pelo matemático John H. Holland, cujo funcionamento se resume em explorar uma população de indivíduos que evoluem a cada geração, devido à ação dos operadores genéticos de seleção, mutação e *crossover*.

II. Referencial Teórico

Como principais referências técnicas para o desenvolvimento deste trabalho foram utilizados um tutorial [4] e um repositório do GitHub [5], ambos construídos a partir da linguagem Python. Os demais artigos (vide a seção de referências) foram utilizados para consultar alguns conceitos de Algoritmos Genéticos.

No caso particular dos GA's, os operadores genéticos de inicialização da população, função de aptidão, seleção, reprodução, mutação, atualização e finalização são os operadores comuns e presentes na maioria (senão em todos) os códigos com os mais variados fins.

O fluxograma da Figura 2 ajuda a exemplificar as etapas descritas nos Itens 1 a 7 de um algoritmo genético:

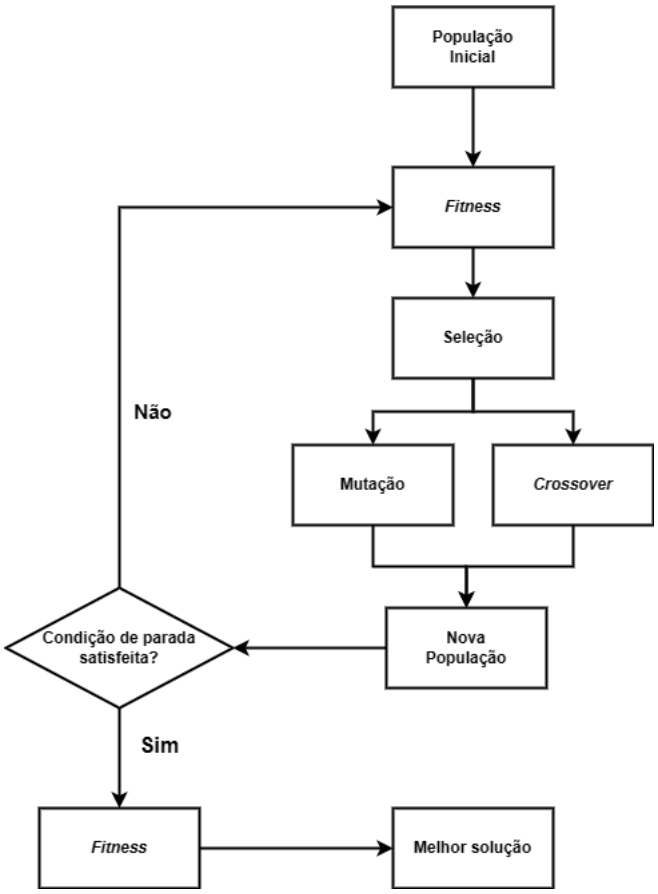


Figura 2: Fluxograma de um GA. Fonte: Adaptado de Izidoro, S. (2023)

1) **Inicialização da população:** A população é criada de forma randômica para garantir uma diversidade genética alta, e que não haja influência nos resultados com uma população muito bem definida.
2) **Função de aptidão (*Fitness*):** Maneira como é avaliado um indivíduo x como bom ou ruim, no que diz respeito à sua configuração genética que é útil ao propósito do algoritmo.
3) **Seleção:** Após construída uma função de aptidão, os indivíduos mais aptos devem ser selecionados para "sobreviverem". Há diversos métodos usados para fazer a seleção, por exemplo:
   - Roleta: É uma técnica que atribui probabilidades de seleção a cada indivíduo com base no seu *fitness*. Quanto maior a aptidão, maior a probabilidade de ser selecionado para reprodução.
   - Torneio: É uma abordagem que seleciona indivíduos com base em sua aptidão relativa, por meio de competições entre subconjuntos aleatórios da população. É uma estratégia eficaz para promover diversidades e explorar diferentes soluções, ajudando a encontrar melhores soluções ao longo das gerações.
4) **Reprodução:** A reprodução entre os genes permite o repasse de características desejáveis de um grupo aos outros indivíduos, aumentando a convergência para um resultado ótimo.
5) **Mutação:** A mutação é o processo de modificar aleatoriamente características de um indivíduo qualquer para se obter determinadas características que, muito provavelmente, não existiam no grupo após os processos de seleção e reprodução.
6) **Atualização:** A atualização é a etapa de substituição da população antiga pela nova população após as etapas de seleção, reprodução e mutação.
7) **Finalização:** Após todas as etapas do GA, há uma condição de parada imposta no código que é verificada toda vez após a atualização.

Desta forma, todas etapas citadas anteriormente são executadas até uma determinada condição de parada, seja quando o GA atingir um determinado número de gerações ou quando o *fitness* desejado for alcançado.

III. Materiais e Métodos

Esta seção descreve a metodologia proposta, incluindo uma breve descrição do Algoritmo Genético (definição do indivíduo, operadores genéticos, função *fitness*, método de seleção, etc.)

A. *Dataset*

Os dados utilizados nesse trabalho foram retirados de um arquivo disponibilizado pelo professor Sandro Carvalho Izidoro, que podem ser encontrados no *https://drive.google.com/file/d/14d5gwHdxN6LZjlkXRM-pd29vLgON7aI2/view*.

Este conjunto de dados inclui as cidades de partida de cada pesquisador (Lisboa, Madrid, Paris, Dublin, Bruxelas e Londres) e os 120 voos correspondentes (ida e volta) com seus respectivos preços. Por exemplo, um voo saindo de **Londres** às **08:27** e chegando em **Roma** às **10:45** com um custo de **139 euros**.

B. *Modelagem do Individuo*

O indivíduo do Algoritmo Genético (GA) construído para a implementação foi definido como um conjunto de 12 voos. Esses voos foram divididos em duas partes: a primeira parte consistia em seis voos partindo de Lisboa, Madrid, Paris, Dublin, Bruxelas ou Londres com destino a Roma, enquanto a segunda parte incluía os seis voos de Roma com destino às cidades de origem de cada pesquisador. Cada voo desse conjunto apresenta informações como horário de embarque e desembarque, além do preço da passagem aérea.

A população inicial foi estabelecida com 25 indivíduos, sendo que cada um desses indivíduos foi criado a partir de uma seleção aleatória entre os 120 voos disponíveis na base de dados. É importante ressaltar que nenhum voo se repetiu em um mesmo indivíduo.

A implementação de um indivíduo pode ser vista no código 1.

Código 1: Gerando voos de ida

```
def gera_aleatorio(self):

    voos_ida = []
    voos_volta = []

    while len(aeroportos_visitados) < 6:

        voo = random.choice(list(self.df.items()))
        voo_ida = str(voo[1][0])

        if str(voo[1][0]) not in
        ↪  aeroportos_visitados and voo_ida !=
        ↪  'FCO':
            voos_ida.append(voo)

            aeroportos_visitados.add(str(voo_ida))
```

Dado que a população inicial é formada de forma aleatória, existe a chance de um indivíduo possuir mais de um voo com o mesmo local de partida. Para lidar com essa situação, foi introduzida uma variável que verifica se o voo selecionado aleatoriamente já está presente no vetor de voos. Isso garante que não haja repetição de voos no mesmo indivíduo.

Importante salientar, que, para facilitar as manipulações para os operadores de seleção e cruzamento, foi necessário ordenar o vetor de voos de ida em ordem alfabética com base na cidade de embarque.

Após adicionar escolher os voos de ida, resta adicionar os voos de volta, como pode ser visto no código 2.

Código 2: Gerando voos de volta

```
%Voos de volta

    while len(aeroportos_visitados) > 0:

        voo = random.choice(list(self.df.items()))
        voo_ida = str(voo[1][1])

        if voo_ida in aeroportos_visitados and
        ↪  str(voo[1][0]) == 'FCO':

            voos_volta.append(voo)
            aeroportos_visitados.remove(voo_ida)

    voos_volta = sorted(voos_volta, key=lambda x:
    ↪  x[1][1])

    self.ind = voos_ida + voos_volta
```

Para obter os voos de retorno de Roma, utilizamos novamente a variável "aeroportos_visitados"para verificar se o voo selecionado aleatoriamente já estava presente nesse vetor. Se o voo já estivesse no vetor, adicionávamos o voo de retorno ao vetor "voos_volta"e removíamos o aeroporto correspondente do vetor de aeroportos. Se o voo não estivesse no vetor, selecionávamos um novo voo aleatoriamente. Essa abordagem garantiu que não houvesse repetição de voos de retorno e aeroportos nos resultados.

C. *População*

A população é simplesmente a coleção dos indivíduos. Esta deve ser gerada de forma aleatória em um primeiro momento. Como a criação de indivíduos já é feita aleatoriamente então a população, se criada a partir da função gera_aleatorio, deve idem ser aleatória. Sua implementação é visível no código 3

Código 3: Gerando população inicial

```
def pop_inicial(self):
    pop = []
    for i in range(0, self.popsize):
        ind = individuo(self.df)
        ind.gera_aleatorio()
        ind.calc_fitness()
        pop.append(ind)
```

Acerca de detalhes de implementação, a população é criada com base no hiperparâmetro "popsize", que indica o tamanho máximo da população, e pode ser ajustado posteriormente.

D. *Função Fitness*

A *fitness* é a forma de avaliar aptidão do indivíduo para uma solução ótima do problema. No caso desse algoritmo o cálculo proposto foi criar uma única variável numérica que contém a soma do preço da passagem de todos os voos, o que inclui a ida e a volta para cada destino, e os tempos de espera na ida e na volta, convertidos para minutos, como exposto no código 4.

Código 4: Função Fitness

```
def calc_fitness(self):

    self.preco = 0

    for c in range(0, 12):

        if c < 6:
            if int(self.ind[c][1][3]) <
            ↪  self.primeiro_ida:
                self.primeiro_ida =
                ↪  int(self.ind[c][1][3])

            if int(self.ind[c][1][3]) >
            ↪  self.ultimo_ida:
                self.ultimo_ida =
                ↪  int(self.ind[c][1][3])
        else:

            if int(self.ind[c][1][2]) <
            ↪  self.primeiro_volta:
                self.primeiro_volta =
                ↪  int(self.ind[c][1][2])

            if int(self.ind[c][1][2]) >
            ↪  self.ultimo_volta:
                self.ultimo_volta =
                ↪  int(self.ind[c][1][2])

        self.preco += int(self.ind[c][1][4])

    self.fitness = self.ultimo_ida -
    ↪  self.primeiro_ida
    self.fitness += self.ultimo_volta -
    ↪  self.primeiro_volta
    self.fitness += self.preco
```

O tempo de espera na ida é dado pela diferença entre o horário em que o primeiro e o último pesquisador chegam em Roma, dessa forma temos sempre como base o maior tempo de espera enfrentado por um deles no aguardo da van. Na volta o

cálculo é o mesmo, mas, dessa vez, são levados em conta os horários em que os pesquisadores partirão para suas cidades de origem.

E. Operadores genéticos

Seleção

O método de seleção em algoritmos genéticos é responsável por escolher indivíduos promissores com base em sua função objetivo (*fitness*). Geralmente, os indivíduos com melhor *fitness* têm uma maior probabilidade de serem selecionados para reprodução, permitindo que suas características sejam transmitidas para a próxima geração.

O método usado para realizar a seleção foi o torneio, como pode ser visto no código 5

Código 5: Função Torneio

```
def torneio(self, pop):

  aux = uniform(0, 1)
  torn = sample(pop, self.ntoneio)

    if aux <= 0.75:
        i = min(torn, key=lambda indiv:
        ↪  indiv.fitness)
    else:
        i = max(torn, key=lambda indiv:
        ↪  indiv.fitness)
  return i
```

Essa abordagem envolve a seleção de um indivíduo com base em sua aptidão relativa, por meio de uma competição entre subconjuntos da população. Na nossa implementação, selecionamos dois indivíduos aleatórios da população após gerar um número aleatório entre 0 e 1. Se esse número for menor ou igual a 0.75, selecionamos o indivíduo com menor valor de *fitness*. Por outro lado, se o número for maior que 0.75, selecionamos o indivíduo com maior valor de fitness. Essa estratégia nos permite equilibrar a diversidade e o desempenho dos indivíduos durante o processo de seleção.

Cruzamento

O cruzamento é o processo de juntar dois indivíduos, combinar seus atributos e gerar um novo indivíduo.

O cruzamento criado consta no código 6:

Código 6: Função *Crossover*

```
def crossover(self, pai1: individuo, pai2:
↪  individuo):
    ind_filho = individuo(self.df)
    childP = []
    geneA = int(random.random() * len(pai1.ind))
    geneB = int(random.random() * len(pai1.ind))
    startGene = min(geneA, geneB)
    endGene = max(geneA, geneB)
    for i in range(0, 12):
        if startGene <= i <= endGene:
            childP.append(pai1.ind[i])
        else:
            childP.append(pai2.ind[i])

    ind_filho.ind = childP
    ind_filho.calc_fitness()
    return ind_filho
```

A função descrita 6 recebe dois indivíduos (pai1 e pai2) como entrada e realiza o cruzamento entre eles.

Primeiro, é criado um novo indivíduo (ind_filho) para armazenar o resultado do cruzamento. Em seguida, são selecionados aleatoriamente dois pontos de corte (geneA e geneB) dentro do indivíduo "pai1". Esses pontos de corte definem uma região genética que será trocada entre os pais.

O loop percorre os genes dos indivíduos pais e realiza a troca genética na região definida pelos pontos de corte. Os genes dessa região são copiados do pai1 para o filho, e os genes fora dessa região são copiados do pai2 para o filho.

Por fim, o filho resultante é atualizado com a nova sequência de genes (childP) e é calculado o seu valor de fitness através da função calc_fitness().

Mutação

A operação de mutação é utilizada para introduzir a variabilidade genética nos indivíduos. A mutação utilizada foi a uniforme, caso, um gene dentro do indivíduo, que equivale a um voo, é selecionado aleatoriamente e seu valor é alterado para outro valor válido dentro do espaço de busca. A mutação é visível no Código 7:

Código 7: Função Mutação

```
def mutacao(self, ind: individuo):
    for c in range(0, int(len(ind.ind))):

        aux = uniform(0, 1)

        if aux < self.taxa_mutacao:
            limite_inf =
            ↪  math.floor(int(ind.ind[c][0]) / 20)
            ↪  * 20
            limite_sup = limite_inf + 19
            indice = random.randint(limite_inf,
            ↪  limite_sup)

            if c < 6:
                while indice % 2 == 0:
                    n_rand =
                    ↪  random.randint(limite_inf,
                    ↪  limite_sup)
                    indice = n_rand

                resultado = indice, ind.df[indice]

                ind.ind[c] = resultado
            else:
                while indice % 2 != 0:
                    n_rand =
                    ↪  random.randint(limite_inf,
                    ↪  limite_sup)
                    indice = n_rand

                resultado = indice, ind.df[indice]

                ind.ind[c] = resultado
    return ind
```

A mutação ocorre em cada gene do indivíduo. Para cada gene, um número aleatório é gerado no intervalo de 0 a 1. Se esse número for menor que uma determinada taxa de mutação, o gene será mutado.

Se o gene estiver nas primeiras seis posições do indivíduo, o índice selecionado deve ser ímpar (voos de ida), portanto

devemos é verificado se o índice é par. Se for par, um novo índice aleatório é gerado até que um índice ímpar seja obtido. Em seguida, o valor do gene é substituído pelo novo índice e pelo valor correspondente no espaço de busca(df).

Se o gene estiver nas últimas seis posições, o processo é semelhante, mas o índice selecionado deve ser par (voos de volta). Se o índice for ímpar, um novo índice aleatório é gerado até que um índice par seja obtido. Em seguida, o valor do gene é substituído pelo novo índice e pelo valor correspondente no espaço de busca(df).

Elitismo

O elitismo consiste em preservar o(s) melhor(es) indivíduo(s) de uma geração para a próxima, garantindo que suas características favoráveis sejam transmitidas. Isso ajuda a manter a qualidade da população e acelerar a convergência para soluções ótimas.

Código 8: Elitismo

```
best_oldgen = min(pop, key=lambda indiv:
↪  indiv.fitness)
best_newgen = min(new_pop, key=lambda indiv:
↪  indiv.fitness)

if int(best_oldgen.fitness) <=
↪  int(best_newgen.fitness):
  pior = max(new_pop, key=lambda indiv:
  ↪  indiv.fitness)
  indice_pior = new_pop.index(pior)
  new_pop[indice_pior] = best_oldgen
```

No código 8, estamos comparando o melhor indivíduo da população anterior com o melhor indivíduo da nova população. Se o melhor indivíduo da população anterior tiver um valor de fitness menor ao melhor indivíduo da nova população, substituímos o pior indivíduo da nova população pelo melhor indivíduo da população anterior.

Parâmetros

A execução do algoritmo genético foi feita com os seguintes parâmetros:

- Tamanho da população: Optamos por ter uma população de tamanho reduzido, com 25 indivíduos, devido à limitação da quantidade de voos disponíveis na base de dados. Ter uma população maior poderia demandar mais recursos computacionais e resultar em uma estagnação na evolução do algoritmo.
- Número de gerações: O número de gerações escolhido foi de 800 gerações e esse número foi obtido após realizar diversos testes.
- Tamanho do elitismo: Dado o tamanho limitado da nossa população, escolhemos utilizar um elitismo de tamanho 1, ou seja, preservamos apenas o melhor indivíduo de cada geração.
- Parâmetro relacionado ao torneio: 2
- Taxa de cruzamento: Optamos por uma taxa de cruzamento de 65% (0.65). Essa taxa moderada evita a perda de características desejáveis durante o cruzamento.
- Chance de mutação: Definimos uma chance de mutação de 10% (0.1). Esse valor foi escolhido para permitir uma pequena variação genética na população, contribuindo para a diversidade e evolução do algoritmo.

IV. Resultados

Para verificar a acurácia do algoritmo, o GA foi executado 30 vezes para base de dados anteriormente citada e os valores de média, mediana, mínimo e máximo de *fitness* foram registrados para populações de 25 indivíduos e para gerações de tamanhos 100, 400 e 800 respectivamente, na Tabela I.

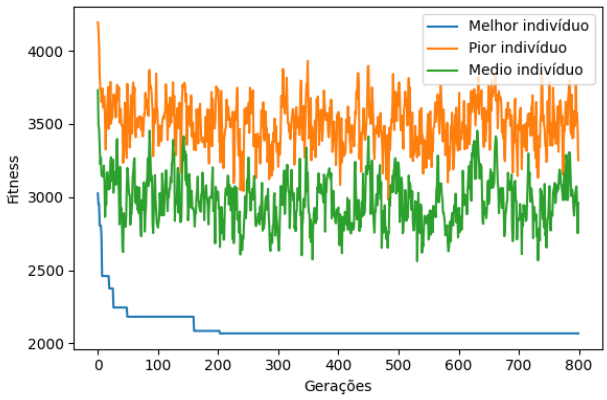
Tabela I: Estatística do fitness pelo número de gerações

Gerações	**100**	**400**	**800**
Média	2337.28	2202.20	2165.37
Mediana	2318	2204	2152
Min	2159	2068	2068
Max	2539	2385	2345

Da tabela I podemos observar que a média do fitness diminui à medida que o número de gerações aumenta, indicando uma possível convergência do algoritmo. A mediana também segue uma tendência de diminuição, indicando uma estabilização dos valores. O valor mínimo e máximo observados diminuem a medida que o número de gerações aumentam,

A forma mais fácil de visualizar a evolução é construindo o gráfico de fitness *versus* o avanço das gerações. Como exemplificado na Figura 3:

Figura 3: Minimização do fitness com o passar das gerações



Da Figura 3 percebe-se que o *fitness* tomado pelo melhor indivíduo converge para um valor mais baixo e se mantém estável com o passar das gerações, até atingir o valor mínimo conforme a modelagem implementada. Este comportamento é explicado pelo uso do elitismo, explicado anteriormente.

Percebe-se também da mesma figura que a variabilidade do pior indivíduo é muito alta. Isso ocorre naturalmente, pois o pior indivíduo não deve convergir para um bom valor de *fitness* e não existe qualquer tipo de tratamento sob ele.

Após a realização dos testes, alguns indivíduos interessantes foram gerados, sendo entre estes um com o menor tempo

de espera, Tabela II, um com o menor preço para passagens Tabela III, um com o melhor *fitness*, Tabela IV e um indivíduo com um custo total das passagens relativamente acessível e um tempo de espera razoável em comparação com os outros indivíduos Tabela V.

Tabela II: Individuo com menor tempo de espera total

Destino	Ida		Volta	
	Horário	Preço	Horário	Preço
Bruxelas	10:30 - 14:57	290 €	14:20 - 17:32	332 €
Paris	11:28 - 14:40	248 €	15:23 - 18:49	150 €
Dublin	12:34 - 15:02	109 €	15:25 - 16:58	62 €
Londres	12:08 - 14:59	149 €	13:37 - 15:33	142 €
Lisboa	12:18 - 14:56	172 €	15:07 - 17:21	129 €
Madrid	12:44 - 14:17	134 €	15:04 - 17:23	189 €
Tempo de espera na ida: 45 minutos				
Tempo de espera na volta: Uma hora e 48 minutos				
Preço total: 2106 €				

Tabela III: Individuo com menor preço total

Destino	Ida		Volta	
	Horário	Preço	Horário	Preço
Bruxelas	06:12 - 10:22	230 €	09:49 - 13:51	229 €
Paris	09:15 - 12:29	225 €	08:23 - 11:07	143 €
Dublin	11:16 - 13:29	83 €	10:33 - 12:03	74 €
Londres	08:27 - 10:45	139 €	08:19 - 11:16	122 €
Lisboa	09:15 - 12:03	99 €	08:04 - 10:59	136 €
Madrid	08:25 - 10:34	157 €	10:33 - 13:11	132 €
Tempo de espera na ida: Três horas e 7 minutos				
Tempo de espera na volta: Duas horas e 29 minutos				
Preço total: 1769 €				

Tabela IV: Individuo com melhor *fitness*

Destino	Ida		Volta	
	Horário	Preço	Horário	Preço
Bruxelas	06:12 - 10:22	230 €	09:49 - 13:51	229 €
Paris	09:15 - 12:29	225 €	08:23 - 11:07	143 €
Dublin	08:04 - 10:11	95 €	10:33 - 12:03	74 €
Londres	08:27 - 10:45	139 €	08:19 - 11:16	122 €
Lisboa	09:15 - 12:03	99 €	08:04 - 10:59	136 €
Madrid	08:25 - 10:34	157 €	10:33 - 13:11	132 €
Tempo de espera na ida: Duas horas e 18 minutos				
Tempo de espera na volta: Duas horas e 29 minutos				
Preço total: 1781 €				

Tabela V: Individuo balanceado

Destino	Ida		Volta	
	Horário	Preço	Horário	Preço
Bruxelas	10:30 - 14:57	290 €	09:49 - 13:51	229 €
Paris	11:28 - 14:40	248 €	08:23 - 11:07	143 €
Dublin	12:34 - 15:02	109 €	10:33 - 12:03	74 €
Londres	12:08 - 14:59	149 €	08:19 - 11:16	122 €
Lisboa	12:18 - 14:56	172 €	08:04 - 10:59	136 €
Madrid	12:44 - 14:17	134 €	10:33 - 13:11	132 €
Tempo de espera ida: 45 minutos				
Tempo de espera volta: Duas horas e 29 minutos				
Preço total: 1938 €				

V. Conclusão

Este trabalho tem como o objetivo de encontrar uma solução boa para o problema de otimização citado na Seção I com base no algoritmo genético (GA). A função *fitness*, que combina o tempo de espera e o preço das passagens, foi utilizada para avaliar os indivíduos da população ao longo das gerações.

O Algoritmo proposto foi capaz de encontrar uma solução boa, se não satisfatória para a base de dados que possuía 120 voos.

Conforme os indivíduos gerados ficou evidente que o algoritmo deu pesos maiores aos valores das passagens, resultando em repostas com custos baixos, mas com tempos de espera de algumas poucas horas. Isso se dá devido à modelagem do problema.

Uma possível forma de equilibrar esses resultados seria calculando tempos de espera separados para cada pesquisador o que modificaria o *fitness*. Ainda, sim, os resultados podem ser considerados positivos, pois o melhor individuo tem valor bem acessível de passagens e tempo de espera razoável, condizentes com a realidade enfrentada nos aeroportos.

Bibliotecas usadas[1]

Pandas

A biblioteca pandas <https://pandas.pydata.org/> é uma biblioteca para análise e manipulação de dados. Para este trabalho, seu papel foi apenas de leitura dos dados, onde é possível ler os dados fornecidos pelo professor com facilidade, vide Código 9:

Código 9: Trecho do código usado para leitura do conjunto de dados com pandas

```
dados = pd.read_csv('./dados/flights.txt', sep=',',
↪  skipfooter=0, header=None, index_col=None,
↪  engine='python')

dados.rename({0: 'aeroporto_saida', 1:
↪  'aeroporto_chegada',
              2: 'horario_saida', 3:
              ↪  'horario_chegada', 4: 'preco'},
              ↪  axis=1, inplace=True)
```

De forma sucinta, os dados são lidos e é desconsiderado o cabeçalho. Além disso, as colunas dos dados lidos são renomeadas para “aeroporto_saida”, “aeroporto_chegada”, “horario_saida”, “horario_chegada” e “preço”.

Random

A biblioteca *random* <https://numpy.org/doc/stable/index.html> é uma biblioteca do python, que oferece funções para gerar números inteiros aleatórios, números de ponto flutuante, escolher elementos aleatórios de uma lista e embaralhar elementos.

A sua relevância neste trabalho foi, principalmente, a geração de números e sequências aleatórias para as operações que os necessitam, como a probabilidade de um gene ser mutado ou não e no momento em que geramos a primeira população do algoritmo.

[1]A biblioteca matplotlib <https://matplotlib.org/> também foi usada, mas como seu papel não tange à implementação do GA, mas sim na visualização dos dados, foi omitida.

REFERÊNCIAS

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Artificial Intelligence, Addison-Wesley Publishing Company, 1989.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

[3] Notas de aula do Prof. Sandro Carvalho Izidoro. Universidade Federal de Itajubá – UNIFEI, Campus Itabira. Maio 10, 2023.

[4] BROWNLEE, J. *Simple Genetic Algorithm From Scratch in Python*. Maceió, 3 mar. 2021. Machine Learning Mastery. Disponível em: <https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/>. Acesso em: 07 jun. 2023.

[5] GAD, A. Genetic Algorithm Python. GitHub. Disponível em <https://github.com/ahmedfgad/GeneticAlgorithmPython>. Acesso em: 08 jun. 2023.