UNIVERSIDADE DO SUL DE SANTA CATARINA

ALEXANDRE SCHUELTER E RAFAEL ROCHA

MODELOS EVOLUCIONÁRIOS E TRATAMENTO DE INCERTEZAS:

APLICAÇÃO DE UM ALGORITMO GENÉTICO NA INDÚSTRIA METALÚRGICA

Tubarão

2023

A USINDI Metal Eireli precisa produzir um conjunto de peças para um lote de vagões ferroviários. Cada peça é feita a partir de uma chapa de aço, que deve ser cortada para o tamanho correto antes de passar pelos processos de dobra, solda, pintura e acabamento. O objetivo é minimizar o desperdício de material e o tempo de produção, ao mesmo tempo que se atende aos requisitos de quantidade e qualidade das peças.

Figura 1: Triângulo de Freio de bitola larga - Peça de segurança para vagões ferroviários.

Para isso, a empresa dispõe de um conjunto de chapas de aço de diferentes tamanhos, e um conjunto de equipamentos de corte. Cada equipamento tem uma capacidade de corte, que varia de acordo com o tamanho da chapa e o tipo de corte a ser feito. O tempo de corte também varia de acordo com esses fatores.

Figura 2: Chapa de Aço - Matéria-prima.

Figura 3: Chapa de Aço - Matéria-prima.

O problema consiste em determinar a melhor combinação de chapas e equipamentos de corte para produzir todas as peças, minimizando o desperdício de material e o tempo de produção. O desperdício de material é medido pela diferença entre a área total de chapa usada e a área total das peças produzidas. O tempo de produção é medido pela soma dos tempos de corte de todas as peças.

Figura 4: Sucata gerada pela sobra de corte, material desperdiçado

Figura 5: Sucata gerada pela sobra de corte, material desperdiçado

Essa sucata ou material não aproveitado, é gerada durante os processos de corte a laser e corte plasma. O aproveitamento da matéria-prima é definido atualmente de forma manual, sem uma otimização adequada.

Figura 7: Máquina de corte de aço por método Plasma.

Figura 8: Exemplo de má otimização da área de corte na matéria-prima.

Para resolver esse problema, podemos utilizar um algoritmo genético que explore diversas soluções para a combinação de chapas e equipamentos de corte. Cada indivíduo da população representa uma solução, que é composta por uma sequência de valores que indicam qual chapa e qual equipamento de corte serão usados para cada peça. O fitness de cada indivíduo é calculado como uma função do desperdício de material e do tempo de produção. A seleção dos pais, o crossover e a mutação são realizados de forma a favorecer a geração de soluções melhores ao longo das gerações.

Foi utilizada a linguagem Python para a resolução desse problema utilizando algoritmos genéticos sem nenhuma biblioteca terceira ou o PyGAD.
Inicialmente, definimos as classes necessárias para representar as peças e as chapas de aço.

A classe PECAS representa cada peça a ser produzida, com suas dimensões: comprimento e largura. A classe Chapa representa cada chapa de aço disponível, com suas dimensões: comprimento e largura. Cada chapa de aço, na industria, pode comportar peças de diversificados tamanhos, variando em largura e comprimento. O que se busca é uma otimização dos espaços utilizados nessa matéria-prima.

Abaixo, a implementação da classe **create_individuals**, com os indivíduos a serem gerados a fim de se chegar ao indivíduo ótimo, nas outras classes..

```python
def create_individual():
    # Criação de um indivíduo (solução) aleatório
    individual = []
    pieces = list(range(NUM_PIECES))
    random.shuffle(pieces)
    while pieces:
        sheet = []
        for _ in range(random.randint(1, NUM_PIECES)):
            if pieces:
                sheet.append(pieces.pop())
            else:
                break
        individual.append(sheet)
    return individual
```

Em seguida, vamos definir a função de fitness dos indivíduos, para buscar avaliar cada indivíduo:

```python
def fitness(individual):
    # Função de aptidão para avaliar o indivíduo
    waste = 0
    for sheet in individual:
        occupied_area = len(sheet) * PIECE_LENGTH * PIECE_WIDTH
        total_area = STEEL_SHEET_LENGTH * STEEL_SHEET_WIDTH
        waste += total_area - occupied_area
    return waste
```

Essa função retorna o desperdício de material após os testes com cada individuo.
Agora, vamos definir as funções de selection e crossover, as quais respetivamente selecionam os indivíduos com maior aptidão (fitness) para a reprodução e a outra, faz o cruzamento entre dois indivíduos a fim de gerar um novo indivíduo com um fitness melhorado:

```python
def selection(population):
    # Seleção dos indivíduos mais aptos para reprodução
    population.sort(key=fitness)
    selected_individuals = population[:int(len(population) * 0.5)]
    return selected_individuals
```

```python
def crossover(parent1, parent2):
    # Cruzamento entre dois indivíduos para gerar um novo indivíduo
    cut_point = random.randint(0, min(len(parent1), len(parent2)))
    child1 = parent1[:cut_point] + parent2[cut_point:]
    child2 = parent2[:cut_point] + parent1[cut_point:]
    return child1, child2
```

A função crossover realiza o cruzamento entre dois indivíduos, escolhendo aleatoriamente as peças de cada um para formar um novo indivíduo.

```
def mutation(individual):
    # Mutação do indivíduo para introduzir variação na população
    if random.random() < MUTATION_RATE:
        sheet_index = random.randint(0, len(individual) - 1)
        piece_index = random.randint(0, len(individual[sheet_index]) - 1)
        individual[sheet_index][piece_index] = random.randint(0, NUM_PIECES - 1)
```

A função mutation realiza a mutação em um indivíduo, escolhendo aleatoriamente uma peça e substituindo-a por outra escolhida aleatoriamente da lista de peças disponíveis, com uma probabilidade definida pelo parâmetro mutation_rate.

Após essas implementações, será definida a população sendo a mesma iniciada após esse processo:

```
# Inicialização da população
population = [create_individual() for _ in range(POPULATION_SIZE)]
```

Essa função recebe como entrada a lista de peças a serem produzidas, a lista de chapas de aço disponíveis, o tamanho da população, o tamanho do torneio para seleção de pais, a taxa de mutação e o número de gerações a serem executadas.

A função inicializa a população com indivíduos gerados aleatoriamente, calcula o fitness de cada indivíduo e executa o algoritmo genético por um número de gerações definido pelo parâmetro generations.

```
# Execução do algoritmo genético
for generation in range(GENERATIONS):
    # Seleção dos indivíduos mais aptos
    parents = selection(population)

    # Reprodução da população
    offspring = []
    for i in range(0, len(parents), 2):
        child1, child2 = crossover(parents[i], parents[i+1])
        offspring.append(child1)
        offspring.append(child2)

    # Mutação da população
    for individual in offspring:
        mutation(individual)

    # Atualização da população
    population = offspring
```

Ao final, a função retorna o melhor indivíduo encontrado, com o menor desperdício de material e menor tempo de produção.

```
# Indivíduo mais apto da última geração
best_individual = min(population, key=fitness)
```

Com essa implementação, é possível utilizar o algoritmo genético para encontrar a melhor combinação de peças e chapas de aço, minimizando o desperdício de material e o tempo de produção.

Para demonstrar o funcionamento, podemos apresentar os resultados que tivemos nas seguintes experiências:

As medidas são todas consideradas utilizando comprimento e largura, em centímetros.

O programa utilizado para executar o script criado é o Visual Studio Code com a extensão do Python.

Para essa amostragem consideramos os parâmetros :
População = 500
Gerações = 100

Exemplo N°1

	Unidade de Medida : CM					
	Chapa de Aço	Peça 1	Peça 2	Peça 3	Peça 4	Peça 5
Comprimento	600	50	75	100	125	600
Largura	200	100	150	200	250	200

Ao passar os parâmetros para o algoritmo desta forma:

```python

# Define o tamanho da chapa de metal
CHAPA_COMPRIMENTO = 600
CHAPA_LARGURA = 200

# Define as peças de metal (comprimento, largura)
PECAS = [(50, 100), (75, 150), (100, 200), (125, 250), (600, 200)]
#PECAS = [(50, 100), (50, 100),  (75, 150), (100, 200), (125, 250), (300, 100),
#PECAS = [(50, 100), (50, 100),  (75, 150), (100, 200), (125, 250), (300, 100),
#PECAS = [(50, 100), (50, 100),  (75, 150), (100, 200), (125, 250), (300, 100),

# Define o número de indivíduos na população
POPULACAO = 500
```

E iniciar o algoritmo, temos o retorno a seguir no console:

```
PS C:\Users\alexa\OneDrive\Área de Trabalho>  c:; cd 'c:\Users\alexa\OneDrive\Área de Trab
ython.python-2023.6.1\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '5348
Melhor Indivíduo: [0, 0, 0, 0, 1]
Valor da Função Objetivo: 120000
Área Total da Chapa: 120000
Área Total das Peças Selecionadas: 120000
Área Desperdiçada: 0
Percentual de Aproveitamento: 100.00%
PS C:\Users\alexa\OneDrive\Área de Trabalho>
```

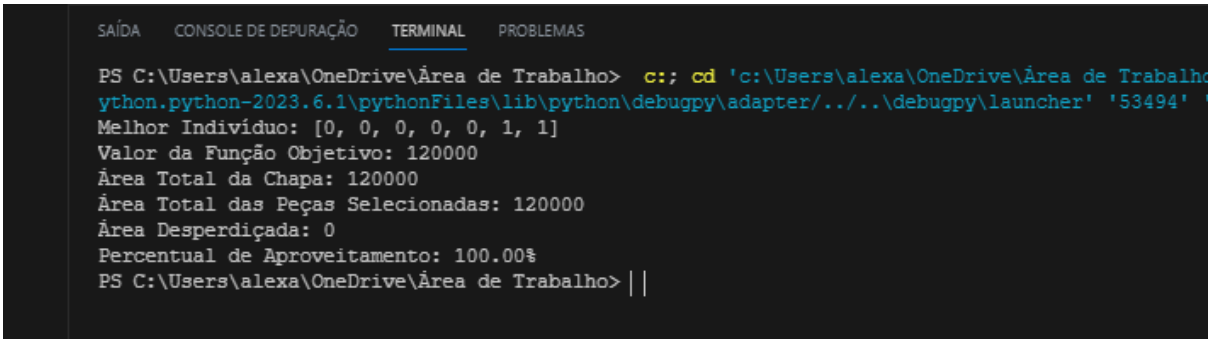
Indicando que o Melhor indivíduo gerado foi [0, 0, 0, 0, 1] ou seja, apenas a chapa número 5 foi selecionada já que nesse exemplo ela tem comprimento de 600 por 200 de largura, ocupando a área toda da chapa.
Também podemos visualizar que a área desperdiçada nesse exemplo foi 0, tendo 100% de aproveitamento.

Exemplo N°2

	Unidade de Medida : CM							
	Chapa de Aço	Peça 1	Peça 2	Peça 3	Peça 4	Peça 5	Peça 6	Peça 7
Comprimento	600	50	50	75	100	125	300	300
Largura	200	100	100	150	200	250	100	100

O exemplo número 2 abrange um número maior de peças, sem utilizar a peça número 5 do exemplo anterior, que estava gerando o melhor aproveitamento.

Também podemos notar que incluímos um elemento Peça 5 que tem uma largura maior do que a chapa de aço, então o algoritmo deve interpretar e não considerar a peça 5 já que ela ultrapassa o parâmetro da chapa.

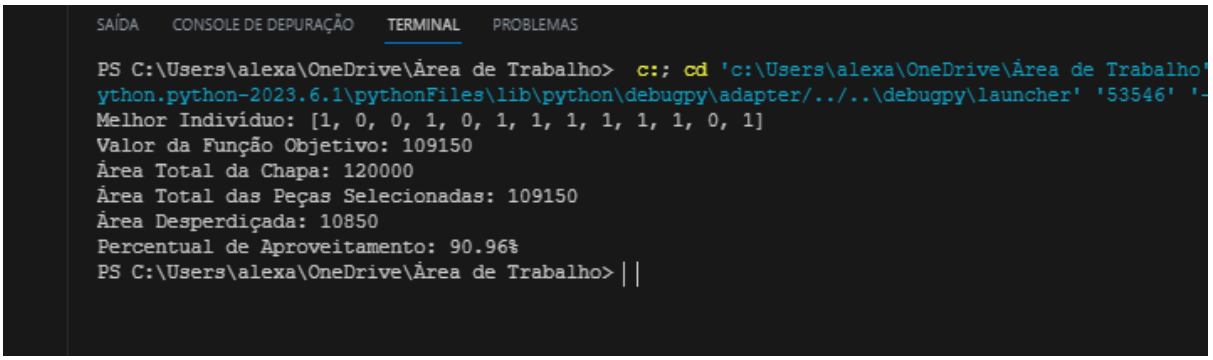


Como resultado, podemos interpretar que o algoritmo selecionou as peças 6 e 7, pois elas juntas fornecem o melhor aproveitamento.(2x peças de 300x100 = 1x 600x200)

Exemplo N°3

	Unidade de Medida : CM												
	Chapa de Aço	Peça 1	Peça 2	Peça 3	Peça 4	Peça 5	Peça 6	Peça 7	Peça 8	Peça 9	Peça 10	Peça 11	Peça 12	Peça 13
Comprimento	600	70	50	75	100	125	50	50	50	70	50	50	50	100
Largura	200	20	100	150	10	250	20	20	10	30	10	25	70	40

No exemplo 3, trouxe um maior número de peças, qual vários tipos variados de peças pequenas, chegando a um nível de aproveitamento de 90.96% da chapa e conseguindo produzir praticamente todas as peças

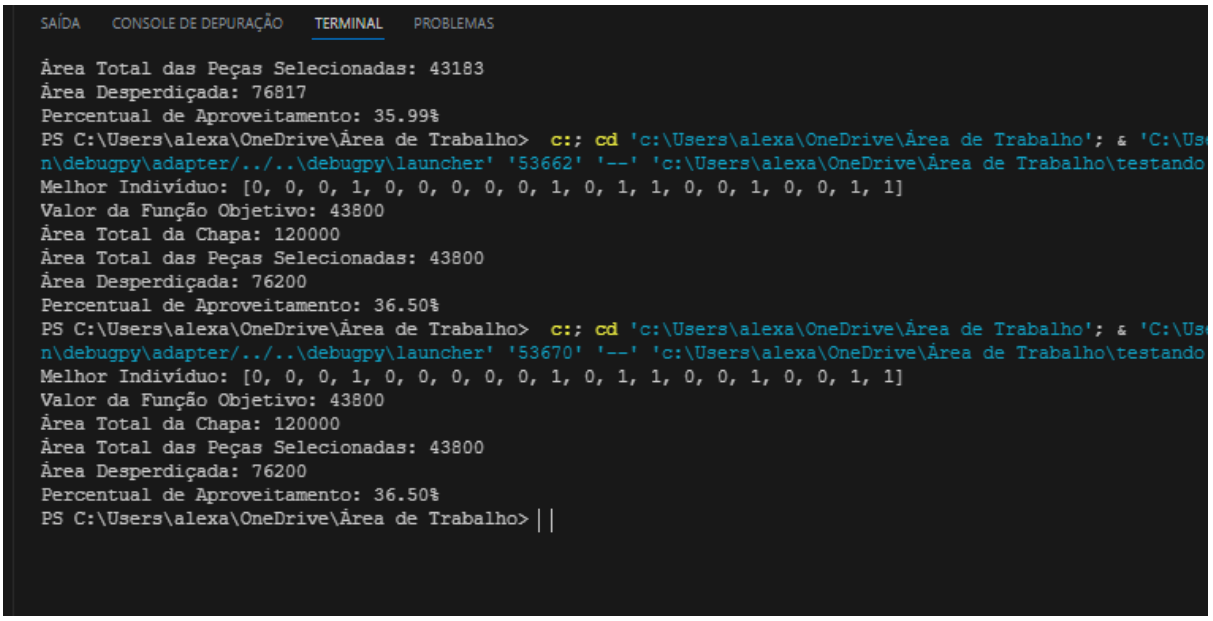


Exemplo n°4

	Unidade de Medida : CM																				
	Chapa de Aço	Peça 1	Peça 2	Peça 3	Peça 4	Peça 5	Peça 6	Peça 7	Peça 8	Peça 9	Peça 10	Peça 11	Peça 12	Peça 13	Peça 14	Peça 15	Peça 16	Peça 17	Peça 18	Peça 19	Peça 20
Comprimento	600	10	12	5	17	10	25	35	90	10	35	47	40	2	1	25	20	15	37	40	65	219
Largura	200	14	20	16	9	10	25	70	100	50	35	50	40	1	2	25	10	25	37	40	65	200
		0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	1	0	0	1	1	

0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1

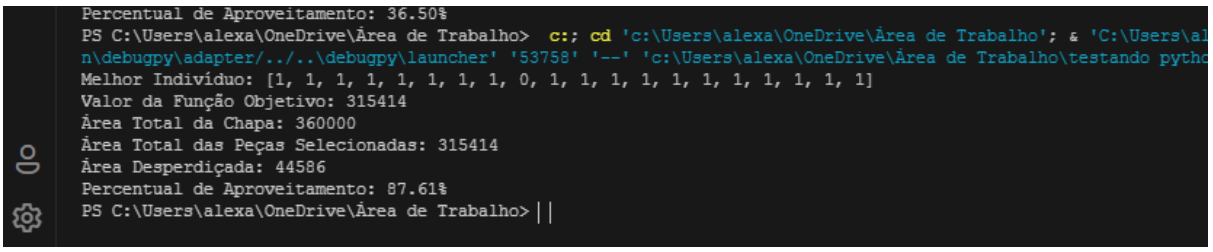
Nesse exemplo utilizei 20 peças de pequenos tamanhos para ver como o algoritmo se comportaria.



Partindo desse resultado, eu aumentei o número da população(para 5000) e as gerações(para 1000) para verificar se teria um número melhor de aproveitamento, porém não houve mudança significativa.

Fazendo a prova real, temos que a resposta para esse índice ser mais baixo é que o cálculo leva em consideração os parâmetros de comprimento e largura em uma dimensão, desprezando as combinações que podem ser feitas virando as peças.
Então utilizando a chapa de 600cm por 200cm, ocupamos 219cm de comprimento e 200cm de largura dessa chapa.

Para esse exemplo ficar com uma performance maior, o ideal seria trocar por uma chapa com mais largura, então executando os testes com 600cm de comprimento e 600cm de largura, chegamos aos seguintes resultados:



Em um cenário normal de trabalho, o usuário poderá comparar com o tamanho da chapa que ele tem, quais são todos os tipos de combinações que ele consegue fazer.

Para as considerações finais, entendemos que há várias situações que conseguimos utilizar os algoritmos genéticos para obter uma performance melhor, tudo baseado nos cruzamentos, mutações, cálculo do fitness.