

Robô Coletor de Latas

- **Integrantes**

Yan Stivaletti E Souza: 11821BCC002

John Vitor da Silva Cunha: 11821BCC005

- **Objetivo**

Programar um robô que pegue todas as latas dentro de uma malha quadriculada, respeitando. Inicialmente, o robô está na posição (0,0). Ele pode se deslocar apenas nos eixos x e y, ou seja, possui apenas quatro movimentos possíveis: Direita (D), Esquerda (E), Para Cima (C) e Para Baixo (B). Não são permitidos movimentos na diagonal. O robô se desloca apenas uma célula por vez.

Logo após a criação do robô, deve-se implementar um Algoritmo genético capaz de melhorar a solução do menor número de passos possíveis para se alcançar todas as latas.

- **Algoritmo Genético**

Representação do Cromossomo:

O algoritmo criado toma como cromossomo o caminho percorrido pelo robô para coletar todas as latas e é dado pelo uso dos símbolos {'D','E','C','B'}, representando direita, esquerda, cima e baixo respectivamente. Portanto a representação final de um cromossomo no código é dado por um vetor com os n passos realizados pelo robô. Por exemplo:

```
Direções do robô: ['D', 'D', 'D', 'D', 'B', 'E', 'E', 'E', 'E', 'B', 'D', 'B', 'E', 'B', 'D', 'D', 'D', 'D']
```

Função Objetivo:

A função objetivo criada recebe um vetor de direções e movimenta o robo na malha, coletando as latas por onde ele passar:

```
def funcObjetivo(self, latas, malha, direcoes):  
    # seta o robô para a posição (0,0)  
    self.linha = 0  
    self.coluna = 0  
    malha2 = deepcopy(malha)  
    qtdDeLatas = len(latas)  
    posicaoAnterior = (0, 0)  
    direcoes2 = [] # vetor que armazena as direções por onde o robô  
    passou
```

```
# percorre todo o vetor das direções
for direcao in direcoes:

    # se recolheu todas as latas então a função é finalizada
    if qtdDeLatas == 0:
        return (malha2, qtdDeLatas, direcoes2)
```

A função objetivo também realiza uma série de verificações para que seja possível cumprir as especificações de movimento do robô dentro da malha, como por exemplo:

```
if direcao == 'D' and (self.coluna != (len(malha2) - 1)) and (
    (self.linha, self.coluna+1) != posicaoAnterior):
    posicaoAnterior = (self.linha, self.coluna)
    self.coluna += 1
    direcoes2.append('D')
```

Neste caso, o trecho de código verifica se a direção lida é a direita, se o robô está em uma das paredes e se o robô não anula o movimento anterior, caso todas as verificações sejam verdadeiras, o robô irá andar para a direita. O mesmo foi feito para as restantes das direções.

O retorno da função objetivo é a malha gerada, a quantidades de latas restantes na malha e o vetor de direções que o robô caminhou:

```
return (malha2, qtdDeLatas, direcoes2)
```

Avaliação e Decodificação Cromossômica:

Dentro de uma população de resultados, cada cromossomo é analisado individualmente de acordo com o sucesso da operação de coleta de latas e da quantidade de passos realizados durante a coleta. Um robô que conseguiu coletar um maior número de latas com o menor número de passos terá uma nota avaliativa melhor do que aquela que pegou menos latas em um maior número de passos. Tem-se como ideal o fato de que o robô consiga pegar todas as latas dentro da malha, logo realizar o menor número de passos em detrimento de não coletar todas as latas não é uma opção viável.

Em termos de código a função fitness realiza todos os testes para avaliar se a nota dos cromossomos dentro de uma população:

```
def fitness(populacao, latas, malha, robo):
    fit = []

    # para cada cromossomo chama a função objetivo
    for c in populacao:
        (_, qtdDeLatas, direcoes) = robo.funcObjetivo(latas, malha, c)

        # se sobrar 1 lata na malha então a nota do cromossomo é
        definida em seu número de passos * 2
```

```

        if qtdDeLatas == 1:
            nota = len(direcoes) * 2

            # se todas as latas foram recolhidas então a nota do
            cromossomo é o número de passos
        elif qtdDeLatas == 0:
            nota = len(direcoes)

            # se nem todas as latas foram recolhidas então a nota do
            cromossomo é a quantidade de latas que faltaram ser recolhidas
            # * o número de passos
        else:
            nota = qtdDeLatas * len(direcoes)

        fit.append(nota) # armazena a nota do cromossomo no vetor de
fit

    return fit # retorna o vetor de fit dos cromossomos

# função para selecionar um cromossomo proporcionalmente pelo seu
fit

```

No final, selecionamos um cromossomo proporcional ao seu fit pela função de seleção, retornando o índice dos fits analisados:

```

def selecao(fit):
    fit = sorted(fit)

    prob = [x / sum(fit) for x in fit]

    rand = rd.uniform(0, max(prob))

    for i in range(len(prob)):
        if rand < prob[i]:
            return i

```

Crossover e Mutação:

O crossover foi implementado de maneira que ao selecionarmos dois indivíduos como pais, o menor será analisado e usado para definir as mudanças geradas nos pais, ou seja, não há a criação de um filho, mas sim a alteração de ambos os pais para gerar-se novos resultados. A alteração é dada por meio de a seleção de uma posição aleatória do menor pai e caso a taxa de crossover seja alcançada haverá uma série de concatenações entre os vetores existentes para acarretar as futuras alterações nos indivíduos pai, a seguir tem-se o trecho do código:

```

def crossover(populacao, p1, p2, pc):

```

```

if len(populacao[p1]) < len(populacao[p2]):
    aux = populacao[p1]
else:
    aux = populacao[p2]

c = rd.randrange(1, len(aux)-1)

if rd.uniform(0,1) < pc:
    temp = populacao[p1][0:c]
    populacao[p1][0:c] = populacao[p2][0:c]
    populacao[p2][0:c] = temp

```

Já a mutação é dada de maneira randômica, percorrendo todos os indivíduos da população. Dada a taxa de mutação inserida será gerado um número aleatório que permite permita ou não a mutação dentro de um novo indivíduo, caso ela ocorra a mesma é dada por uma série de comparações com números randômicos que acarretam a mudança cromossômica, alterando as direções em que o robô se movimentará:

```

def mutacao(populacao, pm):
    for i in range(len(populacao)):
        if rd.uniform(0,1) < pm:
            c = rd.randrange(0, len(populacao[i]))

            if populacao[i][c] == 'D':
                if rd.random() < 2/3:
                    populacao[i][c] = 'E'
                else:
                    if rd.random() < 1/2:
                        populacao[i][c] = 'C'
                    else:
                        populacao[i][c] = 'B'

            if populacao[i][c] == 'E':
                if rd.random() < 2/3:
                    populacao[i][c] = 'D'
                else:
                    if rd.random() < 1/2:
                        populacao[i][c] = 'C'
                    else:
                        populacao[i][c] = 'B'

            if populacao[i][c] == 'B':
                if rd.random() < 2/3:

```

```

        populacao[i][c] = 'C'
    else:
        if rd.random() < 1/2:
            populacao[i][c] = 'D'
        else:
            populacao[i][c] = 'E'

    if populacao[i][c] == 'C':
        if rd.random() < 2/3:
            populacao[i][c] = 'B'
        else:
            if rd.random() < 1/2:
                populacao[i][c] = 'D'
            else:
                populacao[i][c] = 'E'

```

Compilação do Algoritmo Genético:

O algoritmo genético criado implementa todos os conceitos de fitness, crossover e mutação ao longo de um looping que é dado pelo número de gerações que decidimos analisar. Começa-se mostrando o fit original para compararmos a melhora que o algoritmo genético pode oferecer, inicializando na primeira geração temos a seleção proporcional das melhores fits que já possuímos, logo após usamos os conceitos de crossover e mutação dentro dos fits para gerarmos novos resultados. No final esses novos resultados serão analisados com os dados antigos que já possuíamos para ocorrer a comparação do fit antigo com o novo fit. Temos também, para cada geração, a amostra do fit mínimo e a média. No final da compilação, quando todas as gerações foram analisadas, há a amostragem do fit original e o fit depois do AG para compararmos a eficiência do mesmo em aprimorar a solução.

```

def rodaAG(populacao, numeroGeracoes, pc, pm, latas, malha,
robo):

    fit = fitness(populacao, latas, malha, robo)
    fitOriginal = deepcopy(fit)
    print(f'\nFit original: {fit}')

    gen = 0
    print(f'\nGeração: 0, Fit mínimo: {min(fit)}, Média:
{mean(fit)}')
    sleep(0.7)

    while gen < numeroGeracoes:

```

```

pool = []

for _ in range(len(populacao)):
    pool.append(selecao(fit))

oldpop = deepcopy(populacao)
oldfit = deepcopy(fit)

for i in range(len(populacao)):
    populacao[i] = deepcopy(oldpop[pool[i]])

for i in range(0, len(populacao), 2):
    crossover(populacao, i, i+1, pc)

for _ in populacao:
    mutacao(populacao, pm)

fit = fitness(populacao, latas, malha, robo)

print(f'\nFit antigo: {oldfit}')
print(f'Fit atual: {fit}')

for i in range(len(fit)):
    if oldfit[i] < fit[i]:
        populacao[i] = deepcopy(oldpop[i])

fit = fitness(populacao, latas, malha, robo)
print(fit)

gen += 1
print(f'Geração: {gen}, Fit mínimo: {min(fit)}, Média:
{mean(fit)}')
    #sleep(0.7)

fit = fitness(populacao, latas, malha, robo)
print(f'\nFit original: {fitOriginal}')
print(f'Fit depois do AG: {fit}')
melhorFit = indiceDoMenor(fit)
melhorFitOriinal = indiceDoMenor(fitOriginal)
[_, qtdDeLatas, direcoes] = robo.funcObjetivo(latas, malha,
populacao[melhorFit])

```

```

    resultado = [qtdDeLatas, direcoes, fit[melhorFit],
fitOriginal[melhorFitOriginal]]
    return resultado

```

Estruturação e operados implementados:

A estrutura do algoritmo genético criado é dada por steady-state, pois como explicado anteriormente na seleção e fitness do algoritmo, nós não criamos novos filhos, apenas substituímos os pais já existentes, ou seja, ao alterarmos o pai, automaticamente ele é eliminado e tem-se a nova geração de indivíduos. Há também dentro do código operadores de elitismo implementados, ao compararmos todos os indivíduos pais selecionamos sempre o melhor dos pais para realizar as operações de crossover, portanto o elitismo é implementado de maneira sequencial e indireta, selecionando o melhor dos pais para compartilhar os melhores genes, ou seja, os resultados mais eficazes.

Bateria de Testes realizados durante a criação do AG

- Separamos os experimentos de acordo com as seguintes possibilidades de malhas:
 - 5 por 5 (caso básico)
 - 10 por 10 (caso medio)
 - 50 por 50 (pior caso)
- Adicionamos também os seguintes fatores para a resolução do problema:
 - Num de Gerações: 10,50
 - Taxa de Crossover: 0.2,0.8
 - Taxa de Mutação: 0.2,0.8
- No total realizamos 12 experimentos que medem a eficácia e capacidade de resolução do nosso algoritmo:

1-----5,10,0.2,0.2

Malha com as latas: [['robo', 0, 0, 0, 'lata'], [0, 'lata', 0, 0, 0], [0, 'lata', 0, 0, 0], ['lata', 0, 0, 0, 0], [0, 0, 0, 0, 'lata']]

Posições das latas: [[0, 4], [1, 1], [2, 1], [3, 0], [4, 4]]

Fit original: [18, 34, 22, 30]

Geração: 0, Fit mínimo: 18, Média: 26

Quantidade de latas restantes na malha: 0

Direções do robô: ['D', 'D', 'D', 'D', 'B', 'E', 'E', 'E', 'E', 'B', 'D', 'B', 'E', 'B', 'D', 'D', 'D', 'D']

Número de passos para pegar todas as latas antes do AG: 18

Número de passos para pegar todas as latas depois do AG: 18

2-----5,50,0.2,0.2

Malha com as latas: [['robo', 0, 0, 0, 'lata'], [0, 'lata', 0, 0, 0], [0, 'lata', 0, 0, 0], ['lata', 0, 0, 0, 0], [0, 0, 0, 0, 'lata']]

Posições das latas: [[0, 4], [1, 1], [2, 1], [3, 0], [4, 4]]

6-----10,50,0.2,0.2

9-----50,10,0.2,0.2

[illegible]

[illegible]

[illegible]

Número de passos para pegar todas as latas depois do AG: 5074

Posições das latas: [0, 4], [0, 21], [0, 22], [0, 24], [0, 29], [0, 38], [0, 42], [0, 43], [0, 45], [0, 46], [0, 48], [1, 5], [1, 7], [1, 19], [1, 37], [1, 39], [1, 47], [2, 1], [2, 14], [2, 15], [2, 19], [2, 30], [2, 42], [2, 45], [2, 46], [3, 5], [3, 7], [3, 9], [3, 13], [3, 14], [3, 22], [3, 25], [3, 32], [3, 35], [3, 37], [3, 39], [3, 41], [3, 47], [4, 2], [4, 15], [4, 21], [4, 24], [4, 28], [4, 36], [4, 37], [4, 38], [4, 42], [4, 43], [4, 45], [4, 48], [4, 49], [5, 5], [5, 8], [5, 10], [5, 11], [5, 13], [5, 15], [5, 17], [5, 18], [5, 35], [5, 48], [6, 0], [6, 4], [6, 5], [6, 6], [6, 15], [6, 24], [6, 27], [6, 37], [6, 38], [6, 40], [7, 4], [7, 6], [7, 16], [7, 24], [7, 29], [7, 36], [7, 46], [7, 47], [7, 49], [8, 8], [8, 12], [8, 15], [8, 16], [8, 22], [8, 27], [8, 29], [8, 30], [8, 39], [8, 40], [8, 42], [9, 1], [9, 12], [9, 13], [9, 14], [9, 15], [9, 16], [9, 17], [9, 27], [9, 36], [9, 40], [9, 45], [10, 0], [10, 11], [10, 19], [10, 24], [10, 28], [10, 29], [10, 34], [10, 42], [10, 43], [10, 44], [10, 47], [11, 3], [11, 17], [11, 19], [11, 31], [11, 32], [11, 37], [11, 46], [11, 47], [12, 9], [12, 12], [12, 13], [12, 16], [12, 18], [12, 20], [12, 45], [13, 4], [13, 22], [13, 27], [13, 32], [13, 34], [13, 36], [13, 41], [13, 42], [13, 45], [14, 1], [14, 2], [14, 6], [14, 8], [14, 11], [14, 12], [14, 23], [14, 32], [14, 41], [14, 43], [14, 49], [15, 3], [15, 6], [15, 7], [15, 9], [15, 10], [15, 11], [15, 12], [15, 14], [15, 17], [15, 19], [15, 25], [15, 30], [15, 36], [15, 37], [16, 2], [16, 5], [16, 13], [16, 14], [16, 21], [16, 24], [16, 25], [16, 29], [16, 30], [16, 32], [16, 33], [16, 35], [16, 42], [16, 49], [17, 0], [17, 2], [17, 4], [17, 15], [17, 19], [17, 21], [17, 27], [17, 28], [17, 38], [17, 39], [17, 40], [17, 42], [17, 44], [17, 46], [18, 5], [18, 6], [18, 13], [18, 14], [18, 15], [18, 27], [18, 35], [18, 38], [18, 40], [18, 44], [18, 45], [19, 5], [19, 14], [19, 18], [19, 25], [19, 32], [19, 37], [19, 39], [19, 40], [19, 42], [19, 43], [19, 48], [20, 3], [20, 13], [20, 15], [20, 25], [20, 31], [20, 34], [20, 42], [20, 48], [20, 49], [21, 0], [21, 1], [21, 4], [21, 5], [21, 7], [21, 13], [21, 17], [21, 22], [21, 26], [21, 34], [21, 39], [21, 48], [22, 13], [22, 22], [22, 27], [22, 34], [22, 41], [22, 49], [23, 2], [23, 4], [23, 10], [23, 18], [23, 19], [23, 22], [23, 27], [23, 30], [23, 37], [23, 40], [23, 48], [24, 6], [24, 17], [24, 21], [24, 25], [24, 26], [24, 30], [24, 36], [24, 38], [24, 43], [24, 44], [25, 3], [25, 21], [25, 23], [25, 31], [25, 37], [25, 42], [25, 44], [25, 46], [26, 1], [26, 12], [26, 14], [26, 20], [26, 24], [26, 26], [26, 28], [26, 29], [26, 31], [26, 38], [26, 42], [27, 0], [27, 8], [27, 19], [27, 22], [27, 35], [27, 37], [27, 38], [27, 42], [27, 46], [28, 7], [28, 10], [28, 15], [28, 16], [28, 21], [28, 28], [28, 33], [28, 35], [28, 37], [28, 39], [28, 43], [29, 3], [29, 4], [29, 15], [29, 20], [29, 22], [29, 24], [29, 26], [29, 28], [29, 33], [29, 34], [29, 40], [29, 44], [29, 45], [30, 1], [30, 13], [30, 18], [30, 29], [30, 30], [30, 32], [30, 39], [30, 47], [31, 1], [31, 13], [31, 14], [31, 16], [31, 20], [31, 25], [31, 28], [31, 35], [32, 7], [32, 15], [32, 16], [32, 25], [32, 31], [32, 38], [32, 41], [32, 42], [32, 48], [33, 10], [33, 12], [33, 20], [33, 24], [33, 31], [33, 34], [33, 38], [33, 41], [34, 0], [34, 1], [34, 2], [34, 5], [34, 8], [34, 13], [34, 16], [34, 28], [34, 45], [34, 46], [34, 47], [34, 48], [34, 49], [35, 0], [35, 4], [35, 6], [35, 9], [35, 10], [35, 12], [35, 14], [35, 15], [35, 19], [35, 26], [35, 33], [35, 38], [35, 42], [35, 48], [36, 2], [36, 12], [36, 13], [36, 18], [36, 30], [36, 39], [36, 42], [36, 43], [37, 14], [37, 17], [37, 27], [37, 36], [37, 38], [37, 48], [38, 1], [38, 2], [38, 7], [38, 14], [38, 16], [38, 27], [38, 38], [38, 39], [39, 5], [39, 23], [39, 36], [39, 47], [40, 7], [40, 10], [40, 13], [40, 15], [40, 16], [40, 17], [40, 27], [40, 36], [40, 39], [40, 43], [41, 2], [41, 6], [41, 19], [41, 20], [41, 23], [41, 27], [41, 29], [41, 33], [41, 44], [41, 45], [42, 6], [42, 7], [42, 12], [42, 24], [42, 27], [42, 30], [42, 33], [42, 37], [42, 43], [42, 45], [42, 47], [43, 14], [43, 15], [43, 19], [43, 34], [43, 40], [43, 41], [43, 46], [43, 47], [44, 1], [44, 9], [44, 11], [44, 12], [44, 17], [44, 24], [44, 27], [44, 29], [44, 30], [44, 33], [44, 34], [44, 35], [44, 38], [44, 40], [44, 46], [44, 49], [45, 4], [45, 12], [45, 19], [45, 20], [45, 27], [45, 31], [45, 40], [45, 41], [46, 3], [46, 7], [46, 16], [46, 20], [46, 21], [46, 26], [46, 31], [46, 32], [46, 36], [47, 6], [47, 11], [47, 30], [47, 34], [47, 35], [47, 37], [47, 45], [47, 47], [48, 1], [48, 4], [48, 10], [48, 13], [48, 15], [48, 17], [48, 21], [48, 26], [48, 29], [48, 31], [48, 36], [48, 46], [49, 3], [49, 4], [49, 6], [49, 8], [49, 17], [49, 27], [49, 33], [49, 37], [49, 41], [49, 49]]

Geração: 0, Fit mínimo: 5076, Média: 7362

[illegible]

[illegible]

[illegible]

Número de passos para pegar todas as latas antes do AG: 5076
Número de passos para pegar todas as latas depois do AG: 5066

[illegible]

Posições das latas: [0, 4], [0, 2], [0, 20], [0, 22], [0, 24], [0, 29], [0, 30], [0, 38], [0, 42], [0, 43], [0, 45], [0, 46], [0, 48], [1, 5], [1, 7], [1, 19], [1, 37], [1, 39], [1, 47], [2, 1], [2, 14], [2, 15], [2, 19], [2, 30], [2, 42], [2, 45], [2, 46], [3, 5], [3, 7], [3, 9], [3, 13], [3, 14], [3, 22], [3, 25], [3, 32], [3, 35], [3, 37], [3, 39], [3, 41], [3, 47], [4, 2], [4, 15], [4, 21], [4, 24], [4, 28], [4, 36], [4, 37], [4, 38], [4, 42], [4, 43], [4, 45], [4, 48], [4, 49], [5, 5], [5, 8], [5, 10], [5, 11], [5, 13], [5, 15], [5, 17], [5, 18], [5, 35], [5, 48], [6, 0], [6, 4], [6, 5], [6, 6], [6, 15], [6, 24], [6, 27], [6, 37], [6, 38], [6, 40], [7, 4], [7, 6], [7, 16], [7, 24], [7, 29], [7, 36], [7, 46], [7, 47], [7, 49], [8, 8], [8, 12], [8, 15], [8, 16], [8, 22], [8, 27], [8, 29], [8, 30], [8, 39], [8, 40], [8, 42], [9, 1], [9, 12], [9, 13], [9, 14], [9, 15], [9, 16], [9, 17], [9, 27], [9, 36], [9, 40], [9, 45], [10, 0], [10, 1], [10, 19], [10, 24], [10, 28], [10, 29], [10, 34], [10, 42], [10, 43], [10, 44], [10, 47], [11, 3], [11, 17], [11, 19], [11, 31], [11, 32], [11, 37], [11, 46], [11, 47], [12, 9], [12, 12], [12, 13], [12, 16], [12, 18], [12, 20], [12, 45], [13, 4], [13, 22], [13, 27], [13, 32], [13, 34], [13, 36], [13, 41], [13, 42], [13, 45], [14, 1], [14, 2], [14, 6], [14, 8], [14, 11], [14, 12], [14, 23], [14, 32], [14, 41], [14, 43], [14, 49], [15, 3], [15, 6], [15, 7], [15, 9], [15, 10], [15, 11], [15, 12], [15, 14], [15, 17], [15, 19], [15, 25], [15, 30], [15, 36], [15, 37], [16, 2], [16, 5], [16, 13], [16, 14], [16, 21], [16, 24], [16, 25], [16, 29], [16, 30], [16, 32], [16, 33], [16, 35], [16, 42], [16, 49], [17, 0], [17, 2], [17, 4], [17, 15], [17, 19], [17, 21], [17, 27], [17, 28], [17, 38], [17, 39], [17, 40], [17, 42], [17, 44], [17, 46], [18, 5], [18, 6], [18, 13], [18, 14], [18, 15], [18, 27], [18, 35], [18, 38], [18, 40], [18, 44], [18, 45], [19, 5], [19, 14], [19, 18], [19, 25], [19, 32], [19, 37], [19, 39], [19, 40], [19, 42], [19, 43], [19, 48], [20, 3], [20, 13], [20, 15], [20, 25], [20, 31], [20, 34], [20, 42], [20, 48], [20, 49], [21, 0], [21, 1], [21, 4], [21, 5], [21, 7], [21, 13], [21, 17], [21, 22], [21, 26], [21, 34], [21, 39], [21, 48], [22, 13], [22, 22], [22, 27], [22, 34], [22, 41], [22, 49], [23, 2], [23, 4], [23, 10], [23, 18], [23, 19], [23, 22], [23, 27], [23, 30], [23, 37], [23, 40], [23, 48], [24, 6], [24, 17], [24, 21], [24, 25], [24, 26], [24, 30], [24, 36], [24, 38], [24, 43], [24, 44], [25, 3], [25, 21], [25, 23], [25, 31], [25, 37], [25, 42], [25, 44], [25, 46], [26, 1], [26, 12], [26, 14], [26, 20], [26, 24], [26, 26], [26, 28], [26, 29], [26, 31], [26, 38], [26, 42], [27, 0], [27, 8], [27, 19], [27, 22], [27, 35], [27, 37], [27, 38], [27, 42], [27, 46], [28, 7], [28, 10], [28, 15], [28, 16], [28, 21], [28, 28], [28, 33], [28, 35], [28, 37], [28, 39], [28, 43], [29, 3], [29, 4], [29, 15], [29, 20], [29, 22], [29, 24], [29, 26], [29, 28], [29, 33], [29, 34], [29, 40], [29, 44], [29, 45], [30], [1], [30, 13], [30, 18], [30, 29], [30, 30], [30, 32], [30, 39], [30, 47], [31, 1], [31, 13], [31, 14], [31, 16], [31, 20], [31, 25], [31, 28], [31, 35], [32, 7], [32, 15], [32, 16], [32, 25], [32, 31], [32, 38], [32, 41], [32, 42], [32, 48], [33, 10], [33, 12], [33, 20], [33, 24], [33, 31], [33, 34], [33, 38], [33, 41], [34, 0], [34, 1], [34, 2], [34, 5], [34, 8], [34, 13], [34, 16], [34, 28], [34, 45], [34, 46], [34, 47], [34, 48], [34, 49], [35, 0], [35, 4], [35, 6], [35, 9], [35, 10], [35, 12], [35, 14], [35, 15], [35, 19], [35, 26], [35, 33], [35, 38], [35, 42], [35, 48], [36, 2], [36, 12], [36, 13], [36, 18], [36, 30], [36, 39], [36, 42], [36, 43], [37, 14], [37, 17], [37, 27], [37, 36], [37, 38], [37, 48], [38, 1], [38, 2], [38, 7], [38, 14], [38, 16], [38, 27], [38, 38], [38, 39], [39, 5], [39, 23], [39, 36], [39, 47], [40, 7], [40, 10], [40, 13], [40, 15], [40, 16], [40, 17], [40, 27], [40, 36], [40, 39], [40, 43], [41, 2], [41, 6], [41, 19], [41, 20], [41, 23], [41, 27], [41, 29], [41, 33], [41, 44], [41, 45], [42, 6], [42, 7], [42, 12], [42, 24], [42, 27], [42, 30], [42, 33], [42, 37], [42, 43], [42, 45], [42, 47], [43, 14], [43, 15], [43, 19], [43, 34], [43, 40], [43, 41], [43, 46], [43, 47], [44, 1], [44, 9], [44, 11], [44, 12], [44, 17], [44, 24], [44, 27], [44, 29], [44, 30], [44, 33], [44, 34], [44, 35], [44, 38], [44, 40], [44, 46], [44, 49], [45, 4], [45, 12], [45, 19], [45, 20], [45, 27], [45, 31], [45, 40], [45, 41], [46, 3], [46, 7], [46, 16], [46, 20], [46, 21], [46, 46], [46, 51], [46, 36], [47, 6], [47, 11], [47, 30], [47, 34], [47, 35], [47, 37], [47, 45], [47, 47], [48, 1], [48, 4], [48, 10], [48, 13], [48, 15], [48, 17], [48, 21], [48, 26], [48, 29], [48, 31], [48, 36], [48, 46], [49, 3], [49, 4], [49, 6], [49, 8], [49, 17], [49, 27], [49, 33], [49, 37], [49, 41], [49, 49]]

Geração: 0, Fit mínimo: 5076, Média: 7362

[illegible]

[illegible]

[illegible]

Número de passos para pegar todas as latas antes do AG: 5076
Número de passos para pegar todas as latas depois do AG: 5074

[illegible]

Posições das latas: [0, 4], [0, 2], [0, 22], [0, 24], [0, 29], [0, 38], [0, 42], [0, 43], [0, 45], [0, 46], [0, 48], [1, 5], [1, 7], [1, 19], [1, 37], [1, 39], [1, 47], [2, 1], [2, 14], [2, 15], [2, 19], [2, 30], [2, 42], [2, 45], [2, 46], [3, 5], [3, 7], [3, 9], [3, 13], [3, 14], [3, 22], [3, 25], [3, 32], [3, 35], [3, 37], [3, 39], [3, 41], [3, 47], [4, 2], [4, 15], [4, 21], [4, 24], [4, 28], [4, 36], [4, 37], [4, 38], [4, 42], [4, 43], [4, 45], [4, 48], [4, 49], [5, 5], [5, 8], [5, 10], [5, 11], [5, 13], [5, 15], [5, 17], [5, 18], [5, 35], [5, 48], [6, 0], [6, 4], [6, 5], [6, 6], [6, 15], [6, 24], [6, 27], [6, 37], [6, 38], [6, 40], [7, 4], [7, 6], [7, 16], [7, 24], [7, 29], [7, 36], [7, 46], [7, 47], [7, 49], [8, 8], [8, 12], [8, 15], [8, 16], [8, 22], [8, 27], [8, 29], [8, 30], [8, 39], [8, 40], [8, 42], [9, 1], [9, 12], [9, 13], [9, 14], [9, 15], [9, 16], [9, 17], [9, 27], [9, 36], [9, 40], [9, 45], [10, 0], [10, 19], [10, 19], [10, 24], [10, 28], [10, 29], [10, 34], [10, 42], [10, 43], [10, 44], [10, 47], [11, 3], [11, 17], [11, 19], [11, 31], [11, 32], [11, 37], [11, 46], [11, 47], [12, 9], [12, 12], [12, 13], [12, 16], [12, 18], [12, 20], [12, 45], [13, 4], [13, 22], [13, 27], [13, 32], [13, 34], [13, 36], [13, 41], [13, 42], [13, 45], [14, 1], [14, 2], [14, 6], [14, 8], [14, 11], [14, 12], [14, 23], [14, 32], [14, 41], [14, 43], [14, 49], [15, 3], [15, 6], [15, 7], [15, 9], [15, 10], [15, 11], [15, 12], [15, 14], [15, 17], [15, 19], [15, 25], [15, 30], [15, 36], [15, 37], [16, 2], [16, 5], [16, 13], [16, 14], [16, 21], [16, 24], [16, 25], [16, 29], [16, 30], [16, 32], [16, 33], [16, 35], [16, 42], [16, 49], [17, 0], [17, 2], [17, 4], [17, 15], [17, 19], [17, 21], [17, 27], [17, 28], [17, 38], [17, 39], [17, 40], [17, 42], [17, 44], [17, 46], [18, 5], [18, 6], [18, 13], [18, 14], [18, 15], [18, 27], [18, 35], [18, 38], [18, 40], [18, 44], [18, 45], [19, 5], [19, 14], [19, 18], [19, 25], [19, 32], [19, 37], [19, 39], [19, 40], [19, 42], [19, 43], [19, 48], [20, 3], [20, 13], [20, 15], [20, 25], [20, 31], [20, 34], [20, 42], [20, 48], [20, 49], [21, 0], [21, 1], [21, 4], [21, 5], [21, 7], [21, 13], [21, 17], [21, 22], [21, 26], [21, 34], [21, 39], [21, 48], [22, 13], [22, 22], [22, 27], [22, 34], [22, 41], [22, 49], [23, 2], [23, 4], [23, 10], [23, 18], [23, 19], [23, 22], [23, 27], [23, 30], [23, 37], [23, 40], [23, 48], [24, 6], [24, 17], [24, 21], [24, 25], [24, 26], [24, 30], [24, 36], [24, 38], [24, 43], [24, 44], [25, 3], [25, 21], [25, 23], [25, 31], [25, 37], [25, 42], [25, 44], [25, 46], [26, 1], [26, 12], [26, 14], [26, 20], [26, 24], [26, 26], [26, 28], [26, 29], [26, 31], [26, 38], [26, 42], [27, 0], [27, 8], [27, 19], [27, 22], [27, 35], [27, 37], [27, 38], [27, 42], [27, 46], [28, 7], [28, 10], [28, 15], [28, 16], [28, 21], [28, 28], [28, 33], [28, 35], [28, 37], [28, 39], [28, 43], [29, 3], [29, 4], [29, 15], [29, 20], [29, 22], [29, 24], [29, 26], [29, 28], [29, 33], [29, 34], [29, 40], [29, 44], [29, 45], [30], [1], [30, 13], [30, 18], [30, 29], [30, 30], [30, 32], [30, 39], [30, 47], [31, 1], [31, 13], [31, 14], [31, 16], [31, 19], [31, 25], [31, 28], [31, 35], [32, 7], [32, 15], [32, 16], [32, 32], [32, 35], [32, 31], [32, 38], [32, 41], [32, 42], [32, 48], [33, 10], [33, 12], [33, 20], [33, 24], [33, 31], [33, 34], [33, 38], [33, 41], [34, 0], [34, 1], [34, 2], [34, 5], [34, 8], [34, 13], [34, 16], [34, 28], [34, 45], [34, 46], [34, 47], [34, 48], [34, 49], [35, 0], [35, 4], [35, 6], [35, 9], [35, 10], [35, 12], [35, 14], [35, 15], [35, 19], [35, 26], [35, 33], [35, 38], [35, 42], [35, 48], [36, 2], [36, 12], [36, 13], [36, 18], [36, 30], [36, 39], [36, 42], [36, 43], [37, 14], [37, 17], [37, 27], [37, 36], [37, 38], [37, 48], [38, 1], [38, 2], [38, 7], [38, 14], [38, 16], [38, 27], [38, 38], [38, 39], [39, 5], [39, 23], [39, 36], [39, 47], [40, 7], [40, 10], [40, 13], [40, 15], [40, 16], [40, 17], [40, 27], [40, 36], [40, 39], [40, 43], [41, 2], [41, 6], [41, 19], [41, 20], [41, 23], [41, 27], [41, 29], [41, 33], [41, 44], [41, 45], [42, 6], [42, 7], [42, 12], [42, 24], [42, 27], [42, 30], [42, 33], [42, 37], [42, 43], [42, 45], [42, 47], [43, 14], [43, 15], [43, 19], [43, 34], [43, 40], [43, 41], [43, 46], [43, 47], [44, 1], [44, 9], [44, 11], [44, 12], [44, 17], [44, 24], [44, 27], [44, 29], [44, 30], [44, 33], [44, 34], [44, 35], [44, 38], [44, 40], [44, 46], [44, 49], [45, 4], [45, 12], [45, 19], [45, 20], [45, 27], [45, 31], [45, 40], [45, 41], [46, 3], [46, 7], [46, 16], [46, 20], [46, 21], [46, 26], [46, 31], [46, 32], [46, 36], [47, 6], [47, 11], [47, 30], [47, 34], [47, 35], [47, 37], [47, 45], [47, 47], [48, 1], [48, 4], [48, 10], [48, 13], [48, 15], [48, 17], [48, 21], [48, 26], [48, 29], [48, 31], [48, 36], [48, 46], [49, 3], [49, 4], [49, 6], [49, 8], [49, 17], [49, 27], [49, 33], [49, 37], [49, 41], [49, 49]]

Geração: 0, Fit mínimo: 5076, Média: 7362

[illegible]

[illegible]

[illegible]

Número de passos para pegar todas as latas antes do AG: 5076
Número de passos para pegar todas as latas depois do AG: 5068

- Em o todos os experimentos o robô não deixou de pegar nenhuma lata
- A performance do algoritmo em diferentes malhas e variáveis evolutivas não se alterou
- Para a malha 5 por 5, o melhor resultado foi o de 50 gerações com a taxa de crossover e mutação igual a 0.8
- Para as malhas 10 por 10 e 50 por 50, o melhor resultado foi o de 50 gerações com taxa de crossover e mutação igual a 0.2
- Nem sempre a maior taxa de mutação e crossover trazem os melhores resultados
- A quantidade de gerações possui um grande papel na criação de novas soluções ótimas
- O melhor caminho encontrado pelo robô não é geralmente o mais eficaz, na malha 50 por 50, foram realizados mais 2500 passos, ou seja, o robô muitas vezes teve que caminhar mais do que o próprio tamanho da malha para recolher todas as latas, resultados como 5068 ou 5074 não são os melhores resultados possíveis

O trabalho prático implementado cumpre todos os objetivos discutidos, tanto o robô coletor de latas quanto também o algoritmo genético, foram abordados todos os conceitos estudados em sala de aula para a criação de um AG que otimize os resultados de um experimento. Porém, ao tratarmos da melhor solução ou a solução ótima para o experimento, não conseguimos alcançar o possível melhor resultado. Em casos como a malha 50 por 50, alcançamos resultados médios de 5000 passos, sendo que é possível realizar a mesma operação com a metade de passos ou menos, evidenciando a possibilidade de uma melhora nas operações de busca e recolhimento de latas.

