Engenharia de Computação

Tecnologia e Sociedade
Solução do Problema da Mochila com Algoritmo Genético

Helena Tavares

Julho de 2022

1. Introdução

Esse trabalho foi realizado para a disciplina de Tecnologia e Sociedade e teve como objetivo implementar uma solução para o problema da Mochila, utilizando Algoritmos Genéticos.

Ao longo do relatório será contextualizado o Problema da Mochila, o conceito de Algoritmo Genético, a implementação em Python, a explicação do código e os resultados finais.

2. Problema da Mochila

O Problema da Mochila é um conhecido problema de otimização combinatória que consiste em preencher uma mochila de forma que seja obtido o maior valor possível para o peso máximo suportado pela mochila. Ou seja, cada item inserido, tem um peso e um valor, devemos escolher os itens que somados tenham o maior valor possível, dentro do peso limite.

3. Algoritmo Genético

Algoritmo Genético, ou GA, é inspirado no princípio da evolução de Darwin e nos princípios da genética. São algoritmos probabilísticos que utilizam uma técnica de busca e otimização paralela e adaptativa inspirado no princípio de reprodução e seleção natural. Esse algoritmo utiliza técnicas que imitam a genética evolutiva como mutação, crossover, hereditariedade.

O objetivo de um Algoritmo Genético é encontrar a melhor solução para um problema de busca ou otimização, gerando e classificando os melhores indivíduos conforme sua pontuação avaliativa durante a evolução das populações.

Em um Algoritmo Genético, o cromossomo representa a estrutura de dados que contém uma das possíveis soluções do problema proposto. Esse cromossomo é submetido a um processo evolutivo. Nesse processo o cromossomo é avaliado, selecionado, recombinado (crossover) e é também onde acontece a mutação. O ciclo ocorre diversas vezes, e a cada execução, espera-se que seja obtidos indivíduos mais evoluídos e mais aptos.

Os indivíduos mais aptos são selecionados para a reprodução e essa seleção pode ser feita através de algumas formas como o mecanismo de pressão seletiva ou intensidade de seleção, assim como o mecanismo de torneio. A pressão seletiva é calculada a partir da razão entre o desvio padrão e a média da aptidão. Já na seleção por torneio, um grupo de indivíduos é escolhido de forma aleatória, e o de maior aptidão entre eles é selecionado.

Também existe seleção por truncamento, que consiste em selecionar apenas os indivíduos já pré-classificados baseados num limiar X e que tem a mesma probabilidade de serem escolhidos. E por último a seleção por normalização, que se dá pela ordenação dos indivíduos, conforme pontuação relacionada a aptidão. O melhor e o pior indivíduo são definidos e é feita uma distribuição linear que esteja dentro desses valores.

Os melhores pares de indivíduos são selecionados para gerarem novos indivíduos que não serão exatamente iguais aos pais, mas conterão as características genéticas de ambos genitores envolvidos.

4. Método

Foram criadas listas que contem tamanho igual ao número de itens contidos na mochila. Os valores de cada item pode ser 0 ou 1, o que representa a presença ou ausência dele na solução proposta. Foi criado um conjunto de listas contendo os pesos dos itens e seus valores.

Com a definição de pesos e valores de todos itens envolvidos, foi extraído o número total de itens assim como a capacidade máxima da mochila

em relação ao peso, o tamanho da população por geração e o número de gerações desejadas. A partir desses metadados, o Algoritmo Genético é chamado e executa os seguintes passos:

1) Inicia uma população aleatória baseada no número de indivíduos desejados;
2) Avalia os cromossomos dos itens contidos na mochila através da função Fitness e descarta os indivíduos que o peso total dos itens seja maior que o limite da capacidade da mochila;
3) Baseado no valor contido na mochila de cada indivíduo considerado apto, a função roleta é executada para sortear os cromossomos que vão cruzar entre si, sem elitismo. Os cromossomos de maior valor são combinados para gerar a próxima geração. O operador de cruzamento escolhido foi de “um ponto” e o corte realizado sempre no meio do cromossomo e no caso de número ímpar de genes, com truncamento do número pelo valor inferior (Função Chão). Os indivíduos então gerados pelos genitores compõem a nova geração.
4) Depois de criada a nova geração, alguns dos filhos sofrem mutação. A taxa definida como padrão foi de 5%, ou seja, cada filho tem a chance de 5% de ter algum de seus genes alterados. Caso isso ocorra, o gene a ser mutado é selecionado de forma aleatória. Se o seu valor for 0 muda para 1, e assim ocorre o contrário também.
5) Retorna ao item 2 até que se alcance o número de gerações desejado e então o algoritmo é finalizado. Ao final é gerado um gráfico referente a evolução média da população.

5. Implementação

A seguir, a implementação do problema proposto: solução para o problema da Mochila, utilizando Algoritmo Genético, implementado em Python.

```
1   from ga import *
2   # peso,valor
3   pesoEvalor = [[5, 40], [9, 20],
4                 [8, 30], [15, 65],
5                 [3, 20], [70, 300],
6                 [5, 200], [22, 60],
7                 [90, 300], [7, 200]]
8   pesoMaximo = 150
9   nCromossomos = 150
10  geracoes = 50
11  nItens = len(pesoEvalor)
```

Nas primeiras linhas do código, é criado o *array* pesoEvalor, que vai receber os valores referentes ao peso e valor de cada item, logo após é criado e setado o pesoMaximo, relacionado a capacidade máxima da mochila, o nCromossomos, referente ao número máximo de cromossomos, número de gerações final, e o número de itens, gerado a partir do tamanho do *array* pesoEvalor. Ou seja, são definidos os parâmetros necessários para a execução que serão processadas pelas funções contidas no ga.py

Depois disso, é gerada a população inicial já calculando a média da pontuação e adicionando ao histórico. O loop for itera sobre as gerações e evolui os indivíduos a partir da função evolve, também adicionando a pontuação média de cada geração ao histórico.

```
populacao = population(nCromossomos, nItens)
historicoFitness = [mediaFitness(populacao, pesoMaximo, pesoEvalor)]
for i in range(geracoes):
    populacao = evolve(populacao, pesoMaximo, pesoEvalor, nCromossomos)
    historicoFitness.append(mediaFitness(populacao, pesoMaximo, pesoEvalor))
```

Por fim, são utilizados laços de repetição para imprimir a média de valor contida na mochila relacionada a cada geração criada, assim como o peso máximo suportado, o peso dos itens e seu valor. Podemos visualizar essas informações também de forma visual, com o gráfico gerado referente a evolução de cada geração.

```
# imprimir os resultados da execução
for indice,dados in enumerate(historicoFitness):
    print ("Geracao: ", indice," | Media de valor na mochila: ", dados)

print("\nPeso máximo:",pesoMaximo,"g\n\nItens disponíveis:")
for indice,i in enumerate(pesoEvalor):
    print("Item ",indice+1,": ",i[0],"g | R$",i[1])

print("\nExemplos de boas solucoes: ")
for i in range(5):
    print(populacao[i])

# gerar os gráficos da execução
from matplotlib import pyplot as plt
plt.plot(range(len(historicoFitness)), historicoFitness)
plt.grid(True, zorder=0)
plt.title("Problema da mochila")
plt.xlabel("Geracao")
plt.ylabel("Valor medio da mochila")
plt.show()
```

No arquivo ga.py, contém as principais funções utilizadas pelo algoritmo, baseadas em Algoritmo Genético. A função individual retorna um cromossomo aleatório e a population retorna a população de indivíduos geradas pela função anterior.

```
from random import getrandbits, randint, random, choice

def individual(nItens):
    """Cria um membro da populacao"""
    return [ getrandbits(1) for x in range(nItens) ]

def population(nIndividuos, nItens):
    """"Cria a populacao"""
    return [ individual(nItens) for x in range(nIndividuos) ]
```

Aqui temos a função fitness que é quem contabiliza o valor total e o peso total do indivíduo. Se o peso ultrapassar o limite, retorna -1 e isso indica que o indivíduo não vai ser utilizado, senão, retorna o valor total que a mochila contém para o cromossomo em questão.

A função mediaFitness obtem o valor de cada mochila considerando somente se o número for maior ou igual a zero, que é a condição para considerar o indivíduo apto. A partir desses indivíduos é calculada e retornada a média dos indivíduos aptos.

```
11  def fitness(individuo, pesoMaximo, pesoEvalor):
12      """Faz avaliacao do individuo"""
13      pesoTotal, valorTotal = 0, 0
14      for indice, valor in enumerate(individuo):
15          pesoTotal += (individuo[indice] * pesoEvalor[indice][0])
16          valorTotal += (individuo[indice] * pesoEvalor[indice][1])
17
18      if (pesoMaximo - pesoTotal) < 0:
19          return -1
20      #retorna -1 se a capacidade foi ultrapassada
21      return valorTotal
22      #retorna o valor do indivíduo, em caso de indivíduo apto
23
24  def mediaFitness(populacao, pesoMaximo, pesoEvalor):
25      #utiliza apenas itens que não excedam a capacidade máxima
26      """Encontra a avalicao media da populacao"""
27      summed = sum(fitness(x, pesoMaximo, pesoEvalor) for x in populacao if fitness(x, pesoMaximo, pesoEvalor) >= 0)
28      return summed / (len(populacao) * 1.0)
```

A função selecaoRoleta, recebe a lista das listas onde cada elemento da estrutura principal é uma lista de dois valores contendo a avaliação do cromossomo e os genes que o compõe.

Na linha 50, o formato da estrutura é alterado: são criadas duas novas listas, uma contendo as avaliações e outra contendo todos os códigos genéticos dos indivíduos. Senso assim fica mais fácil calcular o fitness total, que é o somatório de todas as avaliações da geração em questão. Esse valor é passado para a função sortear.

A função sortear monta a tabela com as probabilidades da roleta. Sorteia um valor e verifica qual indivíduo foi escolhido, além disso, ela possui um parâmetro opcional chamado índice_a_Ignorar, que é utilizado na segunda vez que a função é chamada com o índice do pai que foi definido. Dessa forma o cromossomo é ignorado e removido da roleta para evitar a chance de ser selecionado duas vezes e acontecer o que chamamos de elitismo, que é quando um gene é levado puramente para a próxima geração. Ao final, com o índice pai e mãe, retornamos os dois indivíduos que tesão seus genes misturados para gerar um novo indivíduo.

```python
def selecaoRoleta(pais):
    """Seleciona um pai e uma mae baseado nas regras da roleta"""
    def sortear(fitnessTotal, indice_a_ignorar=-1):
        # parametro garante que não vai selecionar o mesmo elemento
        """Monta roleta para realizar o sorteio"""
        roleta, acumulado, valorSorteado = [], 0, random()

        if indice_a_ignorar!=-1:
        #desconta do total, o valor que sera retirado da roleta
            fitnessTotal -= valores[0][indice_a_ignorar]

        for indice, i in enumerate(valores[0]):
            if indice_a_ignorar==indice:
        #ignora o valor ja utilizado na roleta
                continue
            acumulado += i
            roleta.append(acumulado/fitnessTotal)
            if roleta[-1] >= valorSorteado:
                return indice

    valores = list(zip(*pais))
        #cria 2 listas com os valores fitness e os cromossomos
    fitnessTotal = sum(valores[0])

    indicePai = sortear(fitnessTotal)
    indiceMae = sortear(fitnessTotal, indicePai)

    pai = valores[1][indicePai]
    mae = valores[1][indiceMae]

    return pai, mae
```

A função envolve é a principal e engloba o algoritmo genético como um todo. Incialmente ela obtém a lista de listas que contem a avaliação e o código dos indivíduos. Esses valores são ordenados do maior para o menor e passados para a selecaoRoleta que retorna os dois indivíduos que devem ser cruzados. Utilizando corte de listas do Python, um novo indivíduo é criado com a primeira metade do material genético do pai e a segunda metade do material genético da mãe. Com o novo indivíduo criado, ele é adicionado a lista de filhos.

Ao final, cada filho tem 5% chance de sofrer mutação, como foi definido no método. Se o indivíduo tiver que ser alterado um gene aleatório do seu cromossomo é alternado de zero para um, ou vice-versa.

A lista de filhos é retornada e o código se repete, caso ainda não tenha sido geradas as gerações definidas, ou finalizado caso já tenham sido geradas todas as gerações desejadas.

```python
def evolve(populacao, pesoMaximo, pesoEvalor, nCromossomos, mutate=0.05):
    """Tabula cada individuo e o seu fitness"""
    pais = [ [fitness(x, pesoMaximo, pesoEvalor), x] for x in populacao if fitness(x, pesoMaximo, pesoEvalor) >= 0]
    pais.sort(reverse=True)

    # reproducao
    filhos = []
    while len(filhos) < nCromossomos:
        homem, mulher = selecaoRoleta(pais)
        meio = len(homem) // 2
        filho = homem[:meio] + mulher[meio:]
        filhos.append(filho)

    # mutacao
    for individuo in filhos:
        if mutate > random():
            pos_to_mutate = randint(0, len(individuo)-1)
            if individuo[pos_to_mutate] == 1:
                individuo[pos_to_mutate] = 0
            else:
                individuo[pos_to_mutate] = 1

    return filhos
```

6. Resultados

Com a execução do código apresentado, os resultados obtidos foram os seguintes:

Para um peso máximo de 150g, com 10 itens disponíveis e 50 gerações.

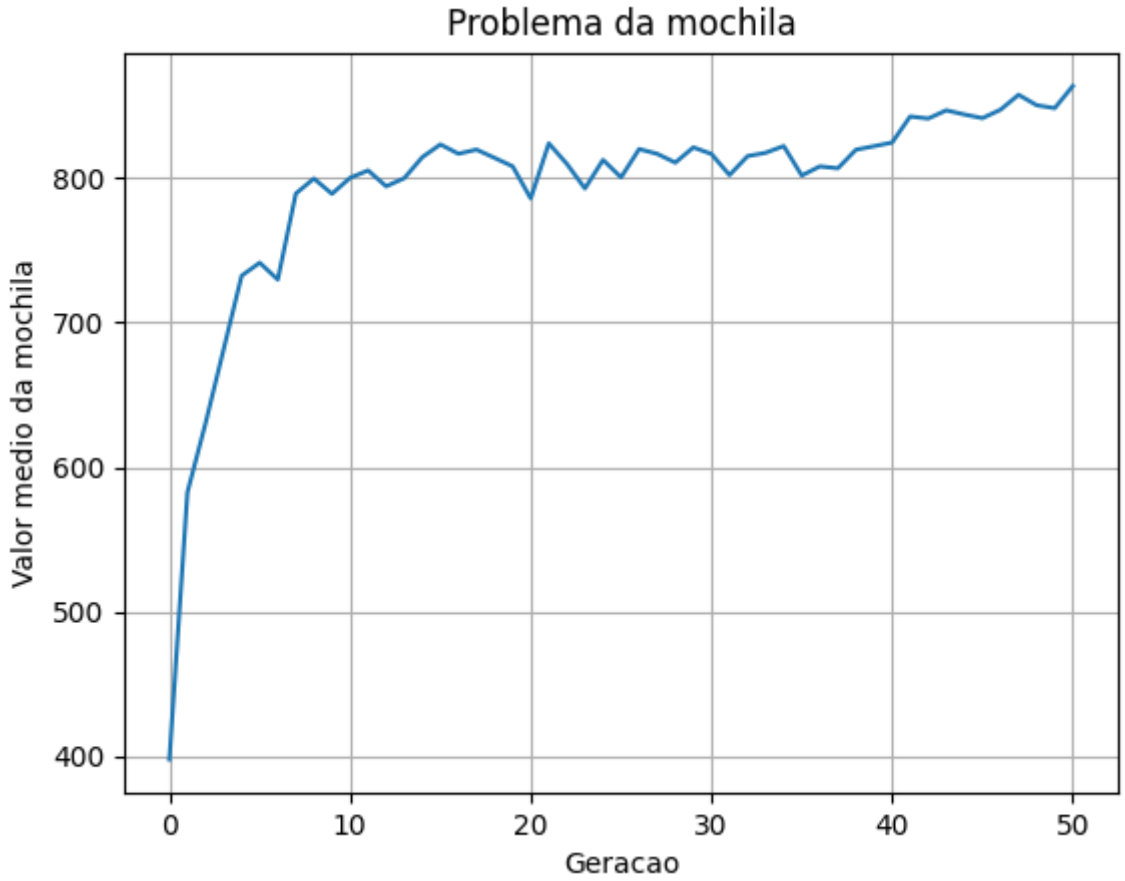
```
Peso máximo: 150 g

Itens disponíveis:
Item  1 :  5 g | R$ 40
Item  2 :  9 g | R$ 20
Item  3 :  8 g | R$ 30
Item  4 :  15 g | R$ 65
Item  5 :  3 g | R$ 20
Item  6 :  70 g | R$ 300
Item  7 :  5 g | R$ 200
Item  8 :  22 g | R$ 60
Item  9 :  90 g | R$ 300
Item  10 :  7 g | R$ 200
```

Gerações e suas respectivas pontuações após cada evolução.

```
Geracao:  0  | Media de valor na mochila:  397.96666666666664
Geracao:  1  | Media de valor na mochila:  582.5333333333333
Geracao:  2  | Media de valor na mochila:  630.2
Geracao:  3  | Media de valor na mochila:  681.2
Geracao:  4  | Media de valor na mochila:  732.2
Geracao:  5  | Media de valor na mochila:  741.1666666666666
Geracao:  6  | Media de valor na mochila:  729.5333333333333
Geracao:  7  | Media de valor na mochila:  788.9666666666667
Geracao:  8  | Media de valor na mochila:  799.3333333333334
Geracao:  9  | Media de valor na mochila:  788.7666666666667
Geracao:  10  | Media de valor na mochila:  799.8
Geracao:  11  | Media de valor na mochila:  804.9333333333333
Geracao:  12  | Media de valor na mochila:  793.9666666666667
Geracao:  13  | Media de valor na mochila:  799.6
Geracao:  14  | Media de valor na mochila:  814.1
Geracao:  15  | Media de valor na mochila:  822.9
Geracao:  16  | Media de valor na mochila:  816.4666666666667
Geracao:  17  | Media de valor na mochila:  819.3333333333334
Geracao:  18  | Media de valor na mochila:  813.6
Geracao:  19  | Media de valor na mochila:  807.7666666666667
Geracao:  20  | Media de valor na mochila:  785.6
Geracao:  21  | Media de valor na mochila:  823.6333333333333
Geracao:  22  | Media de valor na mochila:  809.4666666666667
Geracao:  23  | Media de valor na mochila:  792.4666666666667
Geracao:  24  | Media de valor na mochila:  812.2333333333333
Geracao:  25  | Media de valor na mochila:  800.1666666666666
Geracao:  26  | Media de valor na mochila:  819.7
Geracao:  27  | Media de valor na mochila:  816.4
Geracao:  28  | Media de valor na mochila:  810.3333333333334
Geracao:  29  | Media de valor na mochila:  820.8666666666667
Geracao:  30  | Media de valor na mochila:  816.3333333333334
Geracao:  31  | Media de valor na mochila:  801.7333333333333
Geracao:  32  | Media de valor na mochila:  814.8333333333334
Geracao:  33  | Media de valor na mochila:  817.1333333333333
Geracao:  34  | Media de valor na mochila:  821.8
Geracao:  35  | Media de valor na mochila:  801.4333333333333
Geracao:  36  | Media de valor na mochila:  807.7333333333333
Geracao:  37  | Media de valor na mochila:  806.4333333333333
Geracao:  38  | Media de valor na mochila:  819.2666666666667
Geracao:  39  | Media de valor na mochila:  821.7
Geracao:  40  | Media de valor na mochila:  824.2
Geracao:  41  | Media de valor na mochila:  842.2333333333333
Geracao:  42  | Media de valor na mochila:  840.8
Geracao:  43  | Media de valor na mochila:  846.4666666666667
Geracao:  44  | Media de valor na mochila:  843.6
Geracao:  45  | Media de valor na mochila:  841.0666666666667
Geracao:  46  | Media de valor na mochila:  846.9
Geracao:  47  | Media de valor na mochila:  857.2
Geracao:  48  | Media de valor na mochila:  850.0
Geracao:  49  | Media de valor na mochila:  847.9666666666667
Geracao:  50  | Media de valor na mochila:  863.2666666666667
```

Gráfico da evolução das gerações:



7. Conclusão

Com o desenvolvimento do presente trabalho podemos notar que, ao alterar os parâmetros, conforme interesse, pudemos ver alguns comportamentos do algoritmo como por exemplo, se a proporção entre peso e valor dos indivíduos for homogênea, o algoritmo leva mais gerações para encontrar uma solução.

Se houve muita mutação, o algoritmo perde o foco e se torna muito aleatório, tornando mais difícil de encontrar a solução ótima global, em contrapartida, se não houve mutação, ele fica preso em uma solução ótima local.

A quantidade de itens impacta bastante no número de gerações necessárias para encontrar a solução ótima e aumenta o tempo de execução do código.

Visto isso, demonstramos que o algoritmo genético pode sofrer várias mutações dependendo do interesse da aplicação, tanto ao setar parâmetros, quanto sobre escolher métodos para seleção, conforme citado no item que aborda o funcionamento de um Algoritmo Genético.

Independente da técnica utilizada para escolher os indivíduos, os dados de entrada serem coerentes é essencial para a execução e para encontrar uma solução ótima.

Um Algoritmo Genético atende de forma muito satisfatória a solução para o Problema da Mochila assim como atende problemas de busca com alta massa de dados.

8. Referências

Pacheco, Marco Aurélio Cavalcanti. Disponível em: <http://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/CE-intro_apost.pdf> Acessado em: 05/07/2022

Algoritmos Genéticos. Disponível em: https://sites.icmc.usp.br/andre/research/genetic/ Acessado em: 05/07/2022

Kato, Rodrigo. Algoritmos Genéticos. Disponível em <https://bioinfo.com.br/algoritmos-geneticos/> Acessado em: 05/07/2022

Feofiloff, Paulo. Mochila booleana. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html> Acessado em: 05/07/2022

Dantas, Lucas. Problema da Mochila. Disponível em < https://noic.com.br/materiais-informatica/curso/dp-02/> Acessado em: 06/07/2022