Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

Trabalho III - Implementação do Algoritmo Genético

Problema: Resolva o problema da mochila para uma seleção de 10(dez) objetos qualquer. O peso suportado pela mochila e o par de atributos peso × valor pertencentes a cada objeto devem ser parâmetros de entrada.
Ao término da execução, deverá ser exibida a coleção de objetos a serem colocados na mochila.

1. ***Texto explicativo que apresente o modelo utilizado para representação do problema.***

   Resposta.

   Para representar o problema foi utilizado o paradigma de orientação a objetos, de forma que se pudesse estruturar todo o algoritmo genético. De uma forma geral, o problema da mochila no contexto do trabalho consiste em se obter o melhor valor financeiro possível com os objetos que serão colocados na mochila, de forma que não se ultrapasse o peso total dela. A estrutura desenvolvida no algoritmo genético coloca o cromossomo do indivíduo, de tamanho total de genes à quantidade de objetos, como uma sequência de 0's e 1's, onde para cada gene, o '1' corresponde a um objeto que foi colocado dentro da mochila e o '0' a um objeto que não foi colocado. Este cromossomo é gerado de forma aleatória, no momento de criação de cada indivíduo, e é o fator de resposta para o problema, pois com ele pode-se verificar o valor financeiro total que a mochila carrega, e se ela ultrapassou ou não o seu peso máximo.

2. ***Texto explicando o código implementado (2 pontos).***

   Resposta.

   Para implementar o algoritmo genético que resolva o problema da mochila, utilizamos da linguagem Python e definimos 3 classes principais: a classe "Objeto", onde se armazenará as informações do nome, do peso e do valor de cada um dos objetos carregados (Figura 1); a classe "Indivíduo" que armazena os itens, o limite da mochila, a geração do indivíduo, o valor financeiro total encontrado em sua mochila (fitness), o peso total carregado em sua mochila, seu cromossomo (Figura 2), e as funções de avaliação, do cruzamento e da mutação; e a classe Gen que contém as

Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

informações referente à geração, como a sua população, a quantidade de indivíduos, os próprios indivíduos, o fitness total, os indivíduos elite, o melhor indivíduo, as lista dos melhores indivíduos de cada geração e dos fitness totais de cada geração (Figura 3),e as funções de classificar a população, de gerar o fitness geral, de identificar o melhor indivíduo, o método de seleção, a identificação dos elites e a de execução do algoritmo genético.

```python
#Definição da Classe Objeto
class Objeto():
    def __init__(self,nome,peso,valor):
        self.nome = nome
        self.peso = peso
        self.valor = valor

    def getNome(self):
        return self.nome

    def getPeso(self):
        return self.peso

    def getValor(self):
        return self.valor
```

(Figura 1 - Definição da Classe Objeto)

```python
#Definição da Classe Individuo
class Individuo():
    def __init__(self,itens, limite, geracao=0):
        self.itens = itens #Lista de todos os Itens que o individuo pode carregar
        self.limite = limite #Tamanho da mochila
        self.geracao = geracao #Geração atual do indivíduo
        self.fitness = 0 #Avaliação do Indivíduo
        self.mochilaIndividuo = 0 #Peso total carregado pelo Indivíduo
        self.cromossomo = [] #Cromossomo do indivíduo

        for i in range(len(itens)):
            if(random()) < 0.5:
                self.cromossomo.append("0")
            else:
                self.cromossomo.append("1")
```

(Figura 2 - Definição da Classe Indivíduo)

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

```
class Gen():
    def __init__(self,tamPopulacao):
        self.tamPopulacao = tamPopulacao #Tamanho da População da Geração
        fitnessTotal = 0 #Fitness Geral da População da Geração
        self.populacao = [] #Indivíduos da População da Geração
        self.geracaoPopulacao = 0 #Geração da População
        self.best = 0 #Melhor indivíduo de todas as Gerações
        self.elite = [] #Elite da População da Geração
        self.melhoresIndividuos = [] #Melhores Indivíduos de cada Geração
        self.fitnessGerais = [] #Fitness Gerais de cada Geração
```

(Figura 3 - Definição da Classe Gen)

Na nossa main, os objetos são carregados a partir de um arquivo csv, que contém 10 itens separados por vírgula, com os valores de nome x peso x valor financeiro do objeto. Os valores podem ser alterados dentro do arquivo, bem como incrementar ou decrementar a quantidade de objetos, e os itens serão carregados em uma lista de mesmo nome.

Ainda na main, são definidos os valores necessários para o funcionamento do código. Define-se um valor numérico (float ou int) para o peso máximo que a mochila pode suportar, a quantidade de indivíduos na população (int), a quantidade de indivíduos que serão caracterizados como elite (int), a taxa de mutação (float) e a quantidade de gerações que serão rodadas pelo algoritmo (int).

Após os valores pertinentes informados na main, agora são chamadas as funções de geração da população inicial e das próximas gerações. Para isso, é chamada a classe Gen, que contém o código de execução do algoritmo genético, que, para a população inicial, ele faz em ordem:

1. **Cria os indivíduos de acordo com a quantidade de indivíduos informado na main;**
2. **Avalia cada um dos indivíduos;**
3. **Classifica toda a população daquela geração;**
4. **Identifica o melhor indivíduo de todas as gerações (como é a inicial, esse será o primeiro melhor indivíduo);**
5. **Salva o melhor indivíduo na lista de melhores indivíduos de cada geração;**
6. **Seta o fitness geral daquela geração;**
7. **Salva o fitness geral daquela geração na lista de melhores fitness de cada geração;**
8. **Pega os elites da geração atual de acordo com o valor setada na main.**

Para a criação de uma nova geração, um laço de repetição é gerado, até que se chegue na quantidade de gerações desejada. Dentro do laço, segue-se os seguintes passos:

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

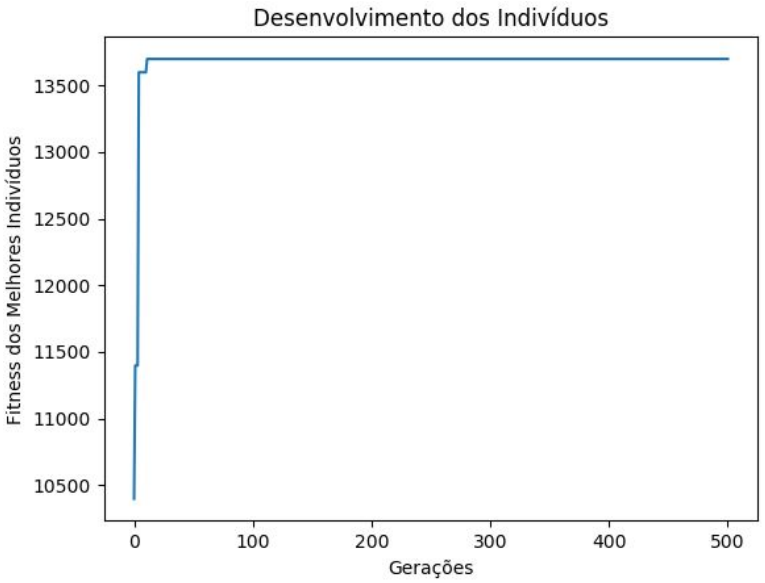
1. **Cria-se um laço para selecionar 2 indivíduos para serem cruzados. O laço tem um "salto" de 2 em 2, pois serão gerados 2 filhos no cruzamento. A quantidade de indivíduos gerados nele é do tamanho total de indivíduos na população(setado na main) menos a quantidade de indivíduos elite;**
2. **Gera-se os filhos a partir do cruzamento desses indivíduos;**
3. **Adiciona os filhos em uma lista de nova população;**
4. **Adiciona os indivíduos elite à lista da nova população;**
5. **Substitui a lista da população pela nova população**
6. **Classifica toda a população daquela geração;**
7. **Incrementa a geração da população;**
8. **Identifica o melhor indivíduo de todas as gerações (como é a inicial, esse será o primeiro melhor indivíduo);**
9. **Salva o melhor indivíduo na lista de melhores indivíduos de cada geração;**
10. **Seta o fitness geral daquela geração;**
11. **Salva o fitness geral daquela geração na lista de melhores fitness de cada geração;**
12. **Pega os elites da geração atual de acordo com a quantidade setada na main.**

Após a execução de todo o código, como resultado, será exibido na tela as características do melhor indivíduo encontrado (Figura 4). Será mostrado o seu cromossomo, a geração que foi encontrado, os objetos que carrega na sua mochila, o seu fitness e o peso carregado na mochila. Também será exibido um gráfico (Figura 5) que mostra os melhores indivíduos x geração.

```
---------------
Melhor Indivíduo:
Cromossomo = ['1', '0', '0', '0', '1', '1', '1', '0', '0', '1']
Geração = 11
---------------
Objetos na Mochila a serem levados:
Nome: Iphone, Preço: R$ 2199.12, Peso: 0.77
Nome: RelÃ³gio, Preço: R$ 4199.12, Peso: 1.25
Nome: Ã“culos, Preço: R$ 99.99, Peso: 0.27
Nome: Colar, Preço: R$ 3000.0, Peso: 0.67
Nome: Iphone 7, Preço: R$ 4199.12, Peso: 1.77
---------------
Valor Total na Mochila (Fitness): R$ 13697.35
Peso na Mochila: 4.73 kg
```

(Figura 4: Saída Final do Programa - Características do melhor indivíduo encontrado)

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com



(Figura 5: Exemplo de um Gráfico - Fitness dos melhores Indivíduos em 500 Gerações)

3. ***Texto apresentando os operadores genéticos implementados (2 pontos).***
   Resposta.

As funções de operadores genéticos foram todas implementadas em função da classe **Indivíduo( )**. A classe indivíduo possui atributos tais como: a lista de todos os itens que o indivíduo pode carregar (itens), o tamanho da mochila (limite), a geração atual do indivíduo (geracao), a avaliação do indivíduo (fitness), o peso total carregado pelo indivíduo (mochilaIndividuo), e por fim, o cromossomo do indivíduo representado por um vetor composto por um e zeros, indicando a existência ou não de um objeto na mochila respectivamente (cromossomo). Todos esse atributos fazem parte da inicialização do indivíduo juntamente com a operação de atribuição randômica dos genes que irão compor o cromossomo do mesmo.

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

```python
#Definição da Classe Individuo
class Individuo():
    def __init__(self,itens, limite, geracao=0):
        self.itens = itens #Lista de todos os Itens que o individuo pode carregar
        self.limite = limite #Tamanho da mochila
        self.geracao = geracao #Geração atual do indivíduo
        self.fitness = 0 #Avaliação do Indivíduo
        self.mochilaIndividuo = 0 #Peso total carregado pelo Indivíduo
        self.cromossomo = [] #Cromossomo do indivíduo

        for i in range(len(itens)):
            if(random()) < 0.5:
                self.cromossomo.append("0")
            else:
                self.cromossomo.append("1")
```

**(Figura 6:Função para inicializar um indivíduo
)**

A operação de atribuição de valores é bastante simples e consiste na geração de um número aleatório entre 0 e 1. Se o número gerado for menor que 0.5 será atribuído o valor 0 ao gene do cromossomo, se o valor for maior ou igual a 0.5 o valor 1 por sua vez que será aplicado ao gene.

Função Avaliação:

A função de avaliação foi criada para que através de uma função heurística que utiliza os valores de fitness e peso da mochila, se torne possível determinar a qualidade do cromossomo que está sendo avaliado. A avaliação consiste em somar o valor de todos os objetos que estão sendo carregados pelo indivíduo, logo, um cromossomo considerado bom é aquele que possui um bom fitness (valor agregado dos objetos que estão sendo carregados) e que não ultrapasse o limite pré estabelecido que a mochila pode suportar. Em situações que o peso da mochila ultrapasse esse limite, o cromossomo passa a não ser tão interessante para realizar cruzamento entre indivíduos e é aplicado uma redução do seu fitness onde apenas 0,1 % do seu valor é considerado para fins futuros.

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

```python
#Função Heurística de Avaliação do Cromossomo
def avaliacao(self):
    self.fitness = 0
    self.mochilaIndividuo = 0
    for i in range(len(self.cromossomo)):
        if (self.cromossomo[i]=="1"):
            self.fitness += self.itens[i].getValor()
            self.mochilaIndividuo += self.itens[i].getPeso()

    if self.mochilaIndividuo > self.limite:
        self.fitness = self.fitness * 0.001
```

(Figura 7: Função para avaliar cada indivíduo)

Função Seleção:

Esta função tem como objetivo selecionar os indivíduos de maneira aleatória, tendo em vista que os melhores indivíduos possuem uma maior probabilidade de serem escolhidos, contudo não significa que cromossomos considerados ruins sejam excluídos do processo de escolha. Para isso é gerado um número aleatório entre 0 e 1 afim de corresponder a porcentagem do fitness total de uma população de indivíduos, logo se uma geração possui o fitness de 1000 e o número aleatório gerado é 0.3 (Seleção da roleta), a sua roleta "para" no valor correspondente a 30 % do fitness da população resultando em 300. Para selecionar é feito um laço que percorre desde a primeira posição do vetor da população e observa o fitness do indivíduo, em seguida é comparado se a soma dos fitness do indivíduos que já foram percorridos é menor que o valor da roleta, se for menor o índice para percorrer a população é incrementado de 1 e avança o indivíduo, se não for menor, é retornado o indivíduo correspondente ao último índice salvo (OBS: o índice começa em -1)

```python
def selecao(self, fitnessTotal):
    selecionado = -1
    roleta = random() * fitnessTotal #Gera um número aleatório para corresponder a porcentagem do fitness total da população
    soma = 0
    i = 0
    while i < len(self.populacao) and soma < roleta:# laço para selecionar o indivíduo
            soma += self.populacao[i].fitness # Enquanto as condições forem respeitadas, soma o valor dos fitness
            selecionado+=1# avança o índice do selecionado
            i+=1
    return self.populacao[selecionado]# retorna o indivíduo selecionado
```

(Figura 8: Função para selecionar o indivíduo para cruzamento)

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

Função Cruzamento:

Para realizar a função de cruzamento é necessário que dois cromossomos sejam criados (Pai 1 e Pai 2). A estratégia adotada para o cruzamento dos mesmos foi a de cruzamento de dois pontos, logo foi necessário estabelecer os pontos de corte que particionaria cada indivíduo em 3 pedaços (tendo em vista que os pontos não poderiam ser nem a primeira posição e nem a última posição, sendo que o segundo corte teria um acréscimo de que não poderia ser uma posição igual ao primeiro corte gerado). Os pontos são criados aleatoriamente respeitando o tamanho do cromossomo, contudo para fins de ter um controle melhor da criação desses pontos, primeiro corte sempre vai ser encontrado um número de 1 a 4 , e o segundo corte a partir da posição do (corte 1) +1 até a posição 8 do vetor (o problema consiste em 10 elementos/genes)

```
#Função de Cruzamento de Dois Pontos
def cruzamento(self,segundoIndividuo):
    primeiroCorte = randint(1,4)
    segundoCorte = randint(primeiroCorte+1,8)
    filho1 = self.cromossomo[0:primeiroCorte]+ segundoIndividuo.cromossomo[primeiroCorte:segundoCorte] + self.cromossomo[segundoCorte::]
    filho2 = segundoIndividuo.cromossomo[0:primeiroCorte] + self.cromossomo[primeiroCorte:segundoCorte] + segundoIndividuo.cromossomo[segundoCorte::]

    filhos = [Individuo(self.itens, self.limite,self.geracao+1),
              Individuo(self.itens, self.limite,self.geracao+1)]

    filhos[0].cromossomo = filho1
    filhos[1].cromossomo = filho2

    return filhos
```

(Figura 9: Função para cruzar dois indivíduos)

Um filho é o resultado da soma dos cortes dos cromossomos pais dele. EX:

Pai 1: ['0', '0', '1', '1', '0', '0', '0', '0', '1', '1']
Pai 2: ['0', '0', '1', '0', '1', '0', '0', '0', '1', '0']

Corte 1: 4
Corte 2: 6

O filho 1 por sua vez apresenta a seguinte estrutura:
0011 (Pai 1) + 01 (Pai 2) + 0011 (Pai 1)
CORTE 1 *CORTE 2*
O filho 2 por sua vez apresenta a seguinte estrutura:
0010 (Pai 2) + 00 (Pai 1) + 0010 (Pai 2)

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

CORTE 1 *CORTE 2*

Filho 1: ['0', '0', '1', '1', '1', '0', '0', '0', '1', '1']
Filho 2: ['0', '0', '1', '0', '0', '0', '0', '0', '1', '0']

Após a geração dos dois filhos a sua geração será incrementada em um, correspondendo ao surgimento de um novo indivíduo de uma nova geração. Um vetor contendo esses dois indivíduos é retornado para fins futuros.

Função Mutação:

A função mutação é necessária para a manutenção da variedade genética de uma população de indivíduos. Para realizar a mutação é preciso estipular uma taxa (1%, 5% , 10%, etc..), em seguida é gerado um número aleatório entre 0 e 1 onde cada gene do cromossomo a ser mutado testará a seguinte condição: Se o número aleatório gerado for menor do que a taxa estipulada anteriormente, então o gene sofrerá alteração. Ex:

Taxa de mutação : 10 %
Cromossomo : 1100010110
Cromossomo mutado: 1100010111

```
#Função de Mutação
def mutacao(self, txMutacao):
    for i in range(len(self.cromossomo)):
        if(random() < txMutacao):
            if(self.cromossomo[i]=='0'):
                self.cromossomo[i]='1'
            else:
                self.cromossomo[i]='0'
    return self
```

(Figura 10: Função para aplicar uma taxa de mutação ao indivíduo)

Função Elitismo:

Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra – DCET
Disciplina de Inteligência Artificial - SIUNEB 19.1
Professor: *Ernesto Massa*
Adriano da Silva Maurício - asmauricio1994@gmail.com
Everaldo Lima de Souza Jr. - everald.souzajr@gmail.com
Filipe Bomfim Santos Furtado – lipe.bomfim99@gmail.com

A função eliteGeracao() tem a simples tarefa de retornar uma quantidade estipulada anteriormente (elitismo) de indivíduos que são considerados os melhores de uma geração, na qual sempre são encontrados nas posições iniciais de uma população quando a mesma já está classificada (ordenadas de maneira decrescente de acordo com seu fitness). Os indivíduos selecionados serão passados para a geração posterior, pois eles possuem características promissoras para geração de novos filhos.

```
def eliteGeracao(self, elitismo):
    self.elite = list(self.populacao[0:elitismo])
```

(Figura 11: Função para retornar os indivíduos considerados elites)