MODELO DE INTELIGÊNCIA ARTIFICIAL PARA APLICAÇÃO LOGÍSTICA EM EMPRESAS DE TRANSPORTE

Bruno Moretto
Maurício Andrade Gomes
Vinícius Finger

RESUMO

Imagine que uma empresa de transporte de cargas para empresas do varejo que precise otimizar a logística para que satisfaça a demanda, respeitando a capacidade máxima de carga de cada caminhão e tenha o melhor lucro possível. Um problema similar ao conhecido “problema da mochila”, porém usando algoritmo genético construído em Python para fazer a seleção de melhores conjuntos de itens.

Palavras-chave: Inteligência Artificial, Algoritmo Genético, Biologia, Logística, Análise Combinatória

ONDE ENTRA A BIOLOGIA

Para esse problema, podemos utilizar algoritmos genéticos inspirados na biologia e no princípio Darwiniano da evolução de espécies, aplicando conceitos como hereditariedade, mutação, seleção natural e recombinação genética. Para isso, implementamos uma simulação de computador, onde as identidades dos indivíduos representadas por cromossomos, onde serão criadas novas combinações genéticas até que uma dessas combinações satisfaça a condição de parada, que significará que chegou ao indivíduo apto.

SIMILARIDADE COM O PROBLEMA DA MOCHILA

Um problema muito parecido com esse que escolhemos tratar é o clássico problema matemático de otimização combinatória exposto por Richard Karp em 1972, o “Knapsack Problem”, ou Problema da mochila, em português. Esse problema consiste em preencher uma mochila com objetos com diferentes pesos e valores com o maior valor possível. Esse problema pode ser resolvido de diversas maneiras, inclusive chegar a uma melhor combinação exata, porém como iremos trabalhar com diversas possíveis combinações de produtos, ainda mais tratando-se de varejo, que as possíveis combinações são praticamente infinitas. Para se ter uma ideia, em um conjunto com 100 produtos, teriam aproximadamente 9.332621541^{157} possíveis combinações, o que tornaria a resolução do problema da mochila por outros meios mais demorado ou até mesmo inviável.

POR QUE A LINGUAGEM PYTHON

Por tratar-se de inteligência artificial, o algoritmo pode facilmente atingir um grau de complexidade muito alto. Por isso, escolhemos uma linguagem *“human-friendly”*, para que se otimize ao máximo o entendimento do código. Outro bom motivo é a ampla comunidade e popularidade com a inteligência artificial, possuindo diversos frameworks e achando

muitos conceitos sendo explicados usando Python. Isso sem contar a versatilidade do código criado ser multi-plataforma.

A ESTRUTURA DO PROJETO

O projeto está dividido em diversos módulos. O principal, chama-se Inteligência.py e foi atribuída a ele a responsabilidade de chamar os métodos e fazer o manuseamento de dados. Há também um módulo somente com configurações para o sistema, chamado configs.py. Nele, é possível configurar a capacidade de carga do caminhão, o número de gerações, número de indivíduos na geração e o número de itens. Já o arquivo auxiliar.py serve para instanciar alguns objetos (como os itens que estão disponíveis para serem carregados, por exemplo), processar informações não ligadas à biologia, como método para avaliar o melhor indivíduo de uma população, calcular o valor total de um indivíduo e processar o HTML que será exibido o gráfico de evolução da inteligência artificial. Já por último e não menos importante, temos o core da nossa aplicação: genetica.py, o responsável por toda a parte inspirada na biologia genética, onde teremos a criação de um indivíduo (que será explicado como funciona na próxima seção), a criação de uma população, o cálculo do fitness de um indivíduo único, de uma população inteira, a evolução e mutação de indivíduos, o sorteio de quem serão o pai e a mãe e também a reprodução desses indivíduos.

ALGORITMO GENÉTICO

O Algoritmo Genético está contido dentro da classe de Algoritmos de Pesquisa Local, dos quais o caminho para resolução não é tão importante quanto o resultado. Essa classe de Algoritmos são eficientes e úteis para problemas de otimização pura, em que o foco é encontrar o melhor estado de acordo com uma função objetivo (função que busca maximizar ou minimizar, dependendo do objetivo do problema).

Um Algoritmo Genético é responsável por gerar estados sucessores com a combinação de dois estados que podemos chamar de estados pais. O processo ocorre da seguinte maneira: A população atual é classificada pela função de aptidão, resultando em pares para acasalamento. Esses pares são responsáveis por gerar descendentes que estão sujeitos a mutações. Os Algoritmos Genéticos iniciam com k estados gerados aleatoriamente, esses estados são chamados de população. Cada estado ou indivíduo é representado como um conjunto de caracteres sobre um alfabeto finito (normalmente com dígitos 1s e 0s). Cada estado é classificado através da função de adequação (fitness function). Essa função deverá retornar valores mais altos para melhores estados. Após ocorrer a classificação, serão selecionados pares aleatórios para a reprodução (Crossover). Para que isso ocorra, é escolhido aleatoriamente um ponto de cruzamento a partir das posições no conjunto de caracteres. Cada indivíduo, após ser gerado, está sujeito a uma mutação aleatória com uma pequena probabilidade. A mutação pode ocorrer de diferentes formas, mas a base da ideia consiste em alterar um valor dentro do conjunto de caracteres, conforme a probabilidade definida anteriormente.

O critério de parada define a ação que irá determinar a finalização do algoritmo. Essa ação pode ser entendida como uma quantidade limite de populações, tempo decorrido ou quando algum estado esperado é atingido, entre outros.

CRIAÇÃO DE INDIVÍDUOS E POPULAÇÃO

No sistema, um indivíduo será um array de tamanho N, onde N é o número de itens a serem carregados no caminhão. Cada posição desse array representará um item, onde se houver 0, significa que o item não será carregado, e se houver 1, será carregado. Então supondo que N = 8, podemos ter um indivíduo assim:

0	1	1	0	1	1	1	0

A geração de cada valor de cada representação de item é feita pseudo-aleatoriamente utilizando a função "*getrandbits"* do Python.

```
individuo = [getrandbits(1) for x in range(numeroDeItens)]
```

Figura 1: Algoritmo de geração do indivíduo

Já a população é um conjunto de indivíduos com o tamanho do número de itens no sistema:

```
populacao = [criaIndividuo(numeroDeItens) for x in range(numeroDeIndividuos)]
```

Figura 2: Algoritmo da função de geração da população

CÁLCULO DO FITNESS

O fitness é a avaliação de um indivíduo sobre algum critério pré-definido. Neste caso, será o valor total de uma combinação de itens sem exceder a capacidade de carga do caminhão. Usaremos o seguinte somatório, onde x = valor do item e n = número de itens:

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + x_3 + x_4 + x_n$$

E também será feito um somatório para verificar se o peso total daquela combinação não excede o peso total do caminhão, onde y = peso do item e n = número de itens:

$$\sum_{i=1}^{n} y_i = y_1 + y_2 + y_3 + y_4 + y_n$$

A implementação dessa regra em código ficou assim:

```
def calculaFitness(individuo, capacidadeDeCarga, itens):
    #Faz a avaliação de um único indivíduo
    pesoTotal = 0
    valorTotal = 0
    indiceItem = 0

    while indiceItem < len(individuo):
        pesoTotal = pesoTotal + (individuo[indiceItem] * itens[indiceItem].getPeso())
        valorTotal = valorTotal + (individuo[indiceItem] * itens[indiceItem].getValor())
        indiceItem += 1

    if (capacidadeDeCarga - pesoTotal) < 0:
        return -1 #Peso total do caminhão excedido.
    else:
        return valorTotal
```

Figura 3: Implementação do cálculo de fitness

Já para calcular a média fitness, ou seja, de uma geração inteira, é só aplicar a função de cálculo fitness indivíduo por indivíduo:

```
def calculaMediaFitness(populacao, pesoMaximo, itens):
    #Calcula a avaliação media da população de indivíduos
    somatorio = sum(calculaFitness(individuo, pesoMaximo, itens) for individuo in populacao if calculaFitness(individuo, pesoMaximo, itens) >= 0)
    mediaFitness = somatorio / (len(populacao) * 1.0)

    return mediaFitness
```

Figura 4: Cálculo da média fitness de uma população

EVOLUÇÃO DA POPULAÇÃO

A evolução é dada pela chamada da função evoluiPopulacao(), que recebe a população, a capacidade de carga do caminhão e os itens, e por fim retorna a população. Essa função cria um array com um array de duas posições para cada indivíduo da população. Na primeira posição, é a média fitness daquele indivíduo. A segunda, os cromossomos. A seguir os primeiros 10 indivíduos dentro do array “pais”:

```
00: [1810000, [0, 1, 0, 1, 1, 0, 1]]
01: [1, [1, 0, 0, 0, 0, 0, 0]]
02: [500000, [0, 0, 0, 0, 0, 1, 0]]
03: [810001, [1, 1, 0, 1, 0, 1, 0]]
04: [1800001, [1, 0, 0, 1, 1, 0, 1]]
05: [1500000, [0, 0, 1, 0, 0, 1, 1]]
06: [1310000, [0, 1, 1, 1, 0, 1, 0]]
07: [1510000, [0, 1, 1, 0, 0, 1, 1]]
08: [1500000, [0, 0, 1, 0, 0, 1, 1]]
09: [1010001, [1, 1, 1, 0, 0, 0, 1]]
```

Após a criação desse array e a estruturação dos dados, ele é ordenado de modo a exibir primeiro os indivíduos com o fitness mais alto.

O próximo passo é fazer a reprodução dos indivíduos em si, onde o pai e mãe se dão por um sorteio pseudo-aleatório utilizando o método da roleta (que terá uma seção somente para explicá-lo). Para a reprodução, o indivíduo pai e mãe são partidos no meio, e o indivíduo filho será a junção da primeira metade do pai com a segunda metade da mãe e após isso, será inserido no array filhos:

```
#Reprodução dos indivíduos
filhos = []

while len(filhos) < numeroDeIndividuos:
    pai, mae = sorteiaPais(pais)
    meioDoCromossomo = len(pai) // 2

    #Filho vai ser a primeira metade do pai + a primeira metade da mãe
    filho = pai[:meioDoCromossomo] + mae [meioDoCromossomo:]
    filhos.append(filho)
```

Após isso, cada filho tem 5% (o que pode ser modificado através do parâmetro “mutacao” passado como parâmetro do método) de sofrer uma mutação em um de seus cromossomos. A mutação inverterá o valor do cromossomo escolhido. Veja a seguir:

```
    for filho in filhos:
        if mutacao > random():
            cromossomoASerMutado = randint(0, len(filho)-1)

            if filho[cromossomoASerMutado] == 1:
                filho[cromossomoASerMutado] = 0
            else:
                filho[cromossomoASerMutado] = 1
```

E assim o processo se repete até a limitação de gerações, sempre registrando a média fitness da população.

```
for geracao in range(geracoes):
    populacao = evoluiPopulacao(populacao, capacidadeDeCarga, itens, numeroDeIndividuos)
    mediaFitnessPopulacao = calculaMediaFitness(populacao,capacidadeDeCarga,itens)
    historicoFitness.append(mediaFitnessPopulacao)
```

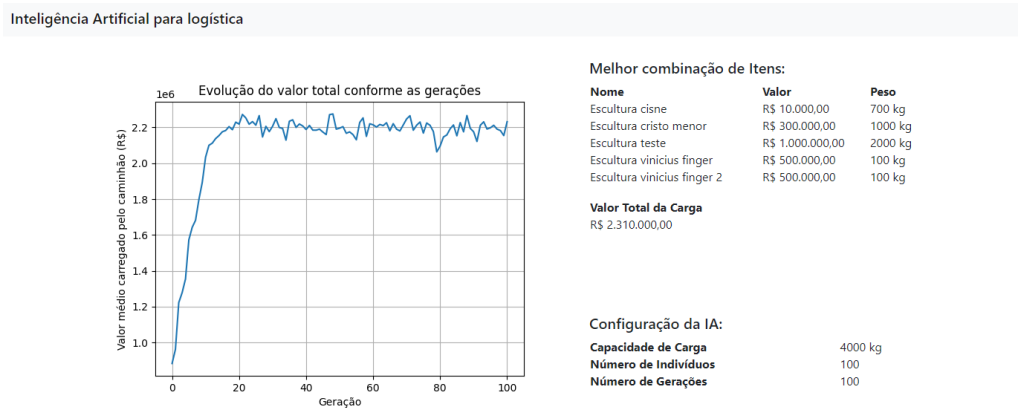
MÉTODO DA ROLETA E SELEÇÃO DOS PAIS

Os indivíduos pai e mãe serão selecionados através do método da roleta. Nesse método, cada indivíduo é representado na roleta de acordo com sua aptidão. Um indivíduo com maior aptidão, representa uma maior representação na roleta, consequentemente uma maior possibilidade de ser escolhido, e vice-versa.

```
roleta = []
acumulado = 0
valorSorteado = random()

#fitness total ignora o indivíduo pai, se já houver
if indiceIgnorado != -1:
    fitnessTotal -= valores[0][indiceIgnorado]

for indice, i in enumerate(valores[0]):
    if indiceIgnorado == indice:
        continue
    acumulado += i
    roleta.append(acumulado/fitnessTotal)

    if roleta[-1] >= valorSorteado:
        return indice
```

EXIBIÇÃO DOS DADOS

Ao terminar de processar todas as gerações, a inteligência artificial montará uma página web com um gráfico de histórico da média do preço médio com o passar das gerações. A direita, será exibida a melhor combinação de itens possível (ou melhor indivíduo de todas as gerações), o valor de cada item, peso, e o valor total da carga. Abaixo, as configurações da IA, como peso máximo do caminhão, número de indivíduos por geração e número de gerações.



CONSIDERAÇÕES FINAIS

Ao final deste trabalho, percebe-se a evolução que os algoritmos estão oferecendo ao mundo. Diversas áreas podem utilizar os algoritmos genéticos para melhorar seus processos ou estratégia. A situação proposta no trabalho foi para uma empresa de logística, mas o seu uso pode ser aplicado em diversas outras áreas.

Aqui se aprendeu sobre o que é um algoritmo genético, brevemente, explicando todas suas características e lógicas. Também se mostrou uma aplicação prática para estes.

Espera-se que ao final do artigo o leitor tenha conseguido entender o poder do algoritmo genético e como ele é eficiente.

REFERÊNCIAS

Fontes: (formatar)
IME-USP: Problema da mochila booleana
https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html

Weisstein, Eric W. "Knapsack Problem." From MathWorld--A Wolfram Web Resource.
https://mathworld.wolfram.com/KnapsackProblem.html

Rosseta Code – Knapsack Problem http://rosettacode.org/wiki/Knapsack_problem