Universidade Federal de Uberlândia

Computação Evolutiva

Trabalho apresentado ao
Professor Paulo Henrique, da disciplina de
Computação Evolutiva do
Curso de Ciência da Computação.

Alunos:
Brenner Borges
Pedro Henrique Fernandes de Oliveira
Vinícius Riquieri

Matrículas:
11421BCC013
11521BCC015
11321BCC002

Uberlândia – Outubro – 2021

Descrição do problema

É de nosso conhecimento a listagem (dataset.xlsx) com todos os filmes vencedores do Oscar, onde temos as informações como nome do filme, ano de lançamento, pontuação, gênero principal e secundário, e uma breve sinopse.
O problema é minimizar a quantidade de dias para que um indivíduo assista os filmes ganhadores do Oscar, reservando 240 minutos de tempo por dia para assistir a eles.
Temos duas restrições para assistir os filmes, são elas: os filmes só poderão ser assistidos por completo, ou seja, não pode começar um filme no dia, e terminar no outro. E somos obrigados a assistir, no caso de sequências, na ordem correta, neste caso o filme “O Poderoso Chefão 1” deverá ser assistido antes do 2, não necessariamente no mesmo dia ou dias subsequentes.
Se duas ou mais soluções empatam na quantidade de dias gastos para assistir os filmes, o critério de desempate utilizado é a média de rating diário e diversidade de gênero diário. O desempate será explicado posteriormente na seção da função de avaliação(fitness).

O Algoritmo Genético

1. Indivíduos

O indivíduo (que representa uma possível solução do problema) consiste em um vetor de ID de filmes, no qual esse vetor pode ser mapeado em ID (int), nome, rating(float), duração(int, em minutos) e gênero, através da biblioteca “namedtuple”

```
Movies = namedtuple('Movies', ['id', 'name', 'rating', 'duration', 'genre'])
```

mapeamento do dos filmes em id, name, rating, duration, genre

Foi usado como base para geração de indivíduos o array [0 ... 92], que neste caso, chamaremos de “**indivíduo fundador**” ou “**indivíduo primordial**”, pois a partir dele que são geradas as populações.

```
primordial = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92]
```

representação do indivíduo denominado como primordial ou fundador

Cada elemento do array, representa o ID de seu respectivo filme ordenado por ano (e.g.: o filme 0(zero) corresponde ao “Wings”, lançado em 1927, e o filme 92 corresponde ao “Nomadland”, lançado em 2020). Portanto, o “indivíduo primordial”, nada mais é que um array de ID de filmes em ordem crescente.

2. População e Gerações

```
def generate_pop(pop_size):
    popu = []
    indv = []
    i = 0
    while i < pop_size:
        indv = random.sample(movListId, len(movListId))
        # Se godfather 2 vier antes do 1, um troca a posição com o outro
        if indv.index(44) > indv.index(46):
            indv[indv.index(44)] = 46
            indv[indv.index(46)] = 44
            # checa se o ultimo individuo gerado já consta na população
            if indv not in popu:
                popu.append(indv)
                i += 1
    if len(popu) == 1:
        return popu[0]
    return popu
```

Método de geração de população

Como dito na seção anterior a população é gerada com base no indivíduo primordial, é feito um *shuffle* no “indivíduo fundador”, onde na geração do indivíduo verifica-se se “Poderoso Chefão" vem antes do “Poderoso Chefão 2” (troca os filmes se caso negativo), e se não repete indivíduos gerados. Esse shuffle é feito pop_size vezes (seja pop_size = o tamanho da população), gerando assim um array de arrays. Por exemplo, se pop_size = 1, o método retornará um array com 1 elemento que é outro array (correspondente a 1 indivíduo), com pop_size = 2, geraria um array com 2 arrays (indivíduos). Veja figura abaixo com pop_size = 3

```
populacaoTamanho3 = [
[12, 14, 74, 4, 48, 76, 43, 50, 38, 88, 92, 68, 69, 80, 91, 30, 20, 22, 28,
36, 42, 0, 81, 25, 60, 54, 5, 65, 2, 34, 58, 84, 70, 27, 83, 41, 11, 57, 63,
1, 8, 44, 13, 49, 55, 51, 37, 47, 90, 46, 89, 18, 39, 15, 19, 26, 72, 87, 78,
77, 61, 66, 10, 3, 73, 29, 16, 56, 62, 64, 32, 35, 71, 40, 86, 24, 33, 52,
21, 23, 6, 9, 31, 7, 85, 82, 67, 53, 45, 79, 75, 59, 17],
[87, 15, 49, 53, 81, 47, 4, 72, 84, 66, 42, 56, 52, 31, 83, 80, 79, 43, 54,
14, 2, 41, 63, 24, 39, 44, 58, 1, 86, 78, 40, 33, 6, 3, 60, 51, 57, 21, 74,
59, 25, 17, 46, 18, 8, 48, 76, 90, 28, 75, 26, 62, 71, 9, 34, 19, 50, 23, 0,
82, 12, 5, 7, 85, 69, 22, 29, 27, 11, 16, 36, 30, 92, 13, 45, 67, 61, 91, 73,
37, 55, 88, 64, 68, 10, 70, 77, 89, 20, 65, 38, 35, 32],
[64, 29, 92, 38, 0, 71, 84, 16, 22, 30, 2, 66, 44, 6, 28, 51, 12, 54, 82, 23,
4, 79, 61, 18, 13, 48, 70, 31, 3, 67, 83, 49, 43, 91, 56, 7, 47, 52, 90, 55,
17, 65, 75, 25, 26, 11, 60, 40, 50, 9, 46, 63, 21, 80, 59, 68, 53, 88, 10, 5,
77, 86, 72, 33, 32, 39, 34, 20, 69, 78, 24, 8, 27, 57, 37, 81, 76, 14, 35,
85, 15, 45, 36, 62, 87, 1, 73, 74, 42, 41, 19, 58, 89]
]
```

neste exemplo foi gerado uma população com n_pop = 3, gerando um array com 3 arrays(3 indivíduos, representados por diferentes cores)

Depois de diversos testes (demonstrados abaixo na seção 8 de testes), definimos que a população contém 300 indivíduos (n_pop = 300) e são geradas 1000 gerações (n_gen = 1000), taxa de crossover em 0,9 (r_cross = 0.9) e taxa de mutação 0,01 (r_mut = 0,01).

3. Avaliação de Aptidão

Avaliamos que um indivíduo é melhor que o outro se a quantidade de dias gastos para assistir todos os filmes é menor. Assim, percorremos o vetor contabilizando os dias gastos, da seguinte maneira: iniciamos um contador de dias (days) e um contador de tempo (duração total de filmes, movTime), enquanto houver filme no vetor, incrementa movTime com a duração do filme atual, se for maior que n_watchtime (neste caso n_watchtime = 240) adicionamos 1 dia, zeramos o movTime e continuamos no mesmo índice (mesmo elemento no vetor). Senão (menor ou igual que n_watchtime), apenas adicionamos 1 ao índice “i” e vamos para o próximo filme. Caso algum indivíduo tenha a posição do filme 44 maior que a posição do filme 46, isso significa que será assistido “O Poderoso Chefão 2”, antes de “O Poderoso Chefão", o que invalida esse indivíduo, devido à restrição mencionada na Descrição do Problema. Então, esse indivíduo é punido com uma avaliação de 999.

```
def fitness(cromo):
    i = 0
    days = 1
    movTime = 0
    if cromo.index(44) > cromo.index(46):
        return 999
    while i < len(cromo):
        movTime += moviesTuple[cromo[i]].duration
        if movTime > n_watchtime:
            days += 1
            movTime = 0
        else:
            i += 1
    return days
```

Implementação da função de avaliação

- Critérios de Desempate

Se por acaso acontecer de dois indivíduos possuírem o mesmo fitness, desempatamos de acordo com a média de rating diária. Separamos a média de rating por dia, e comparamos dia a dia, pontuando o indivíduo que tem a média maior. Assim, quem tiver a maior média em mais quantidade de dias, vence o primeiro critério de desempate.

```
def tiebreaker_rating(indv1,indv2):
   #...Mapeamento dos indivíduos por rating dos dias dos respectivos
      filmeID…#
   best1 = 0
   best2 = 0

   for i in range(len(r1avg)):
       if r1avg[i]>r2avg[i]:
           best1 += 1
       else:
           best2 += 1

   if best1 > best2:
       return indv1
   elif best2 > best1:
       return indv2
   else:
       return tiebreaker_genre(indv1,indv2)
```
Método para resolver o primeiro critério desempate

Se por acaso o empate persistir, chamamos a função de desempate por variação de gênero diário, que soma pontos se num mesmo dia o indivíduo assistir filmes de gêneros diferentes.

```
def tiebreaker_genre(indv1,indv2):
   gscore1 = 1
   gscore2 = 1

   for i in range(n_movies):
       if moviesTuple[indv1[i]].genre != moviesTuple[indv1[i-1]].genre:
           gscore1 += 1
       if moviesTuple[indv2[i]].genre != moviesTuple[indv2[i-1]].genre:
           gscore2 += 1
   if gscore1 >= gscore2:
       return indv1
   elif gscore2 > gscore1:
       return indv2
```
Método para resolver o segundo critério desempate

E se mesmo assim o empate persistir, é usado a filosofia de combates de artes marciais, onde o atual campeão, em caso de empate, continua como campeão, ou seja, quando o gscore1 >= gscore2.

4. Método de Seleção

```
def selection_tournament(pop, scores, k=3):
    # first random selection
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k-1):
        # check if better (e.g. perform a tournament)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
        # check if scores are equal and perform a tiebreaker
        elif scores[ix] == scores[selection_ix] and scores[ix]!= 999:
            tiebreaker_rating(pop[selection_ix],pop[ix])
    return pop[selection_ix]
```

Método de seleção por torneio

Utilizamos como método de seleção o Torneio com k=3, que consiste em selecionar 3 indivíduos aleatoriamente e escolher o que apresentar melhor fitness.O processo de seleção termina quando se realiza uma quantidade de torneios igual ao tamanho da população.
Lembrando que para indivíduos em que Poderoso Chefão 2 vem antes do 1, pontuamos 999 (péssimo fitness, solução indesejável), como punição.

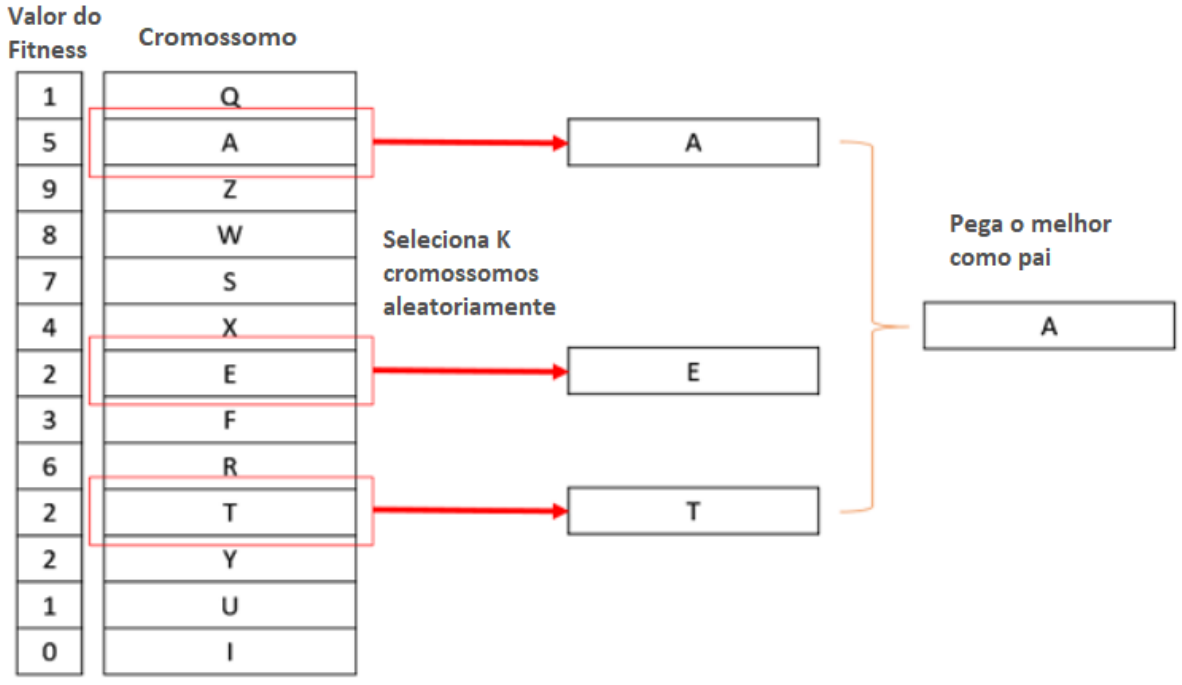

Figura adaptada de:
https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm

5. Cruzamento

Utilizamos o Order Crossover, que escolhe aleatoriamente pontos de crossover, copia a parte do meio (em cinza) do primeiro pai e preenche o resto começando a partir do segundo ponto de crossover, com os filmes que faltam na ordem do segundo pai. Repetimos o processo escolhendo pontos de crossover para o segundo pai, e preenchendo os valores faltantes, a partir do segundo ponto de crossover, na ordem do primeiro pai.
Utilizamos esse tipo de cruzamento pois, pelas nossas perspectivas ele é o mais simples e mais adequado ao problema, uma vez que não podemos ter indivíduos com cromossomos iguais e como o OX conseguimos garantir isso nessa etapa.

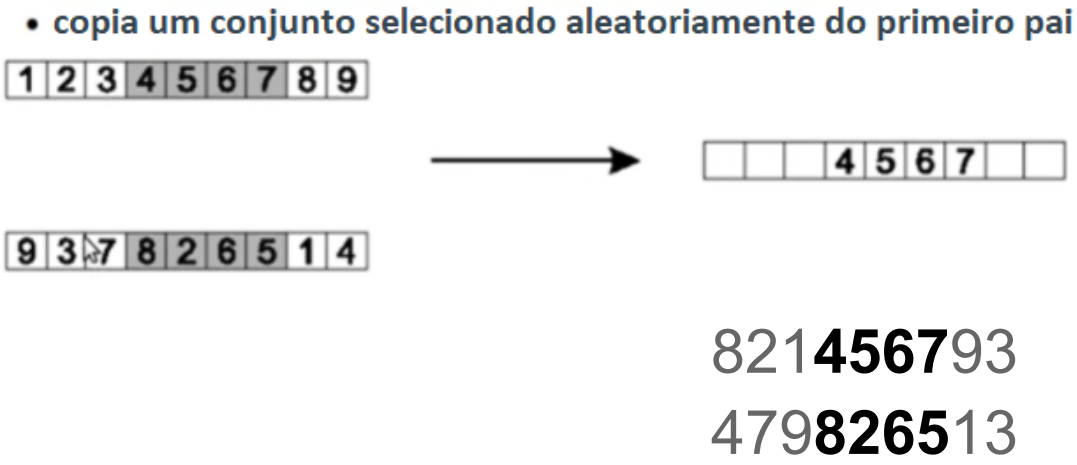

Figura adaptada do vídeo: https://www.youtube.com/watch?v=HATPHZ6P7c4

6. Mutação

```
def mutation(cromo, r_mut):
   # check for a mutation
   if rand() < r_mut:
       i = randint(0, n_movies)
       j = randint(0,n_movies)
       if i == j:
           j = randint(0,n_movies)

       cromo[i], cromo[j] = cromo[j], cromo[i]
```

Método de mutação

Utilizamos uma mutação simples que consiste em apenas trocar um filme de lugar com outro aleatoriamente no cromossomo do indivíduo. Taxa baixa, para não randomizar muito as melhores soluções.

7. Considerações Finais

Para desenvolvimento e execução do algoritmo foi usado configurações de software Python 3.8 no Windows 10 64-bit, e configurações de hardware como:

- CPU: i5 9th Gen 9300h 2.4GHz
- RAM: 8GB
- GPU: GTX 1650

Para rodar o algoritmo usar a versão do Python descrita acima. Assim que terminar a execução, no terminal constará um log das melhores soluções por geração, onde a última melhor solução é a solução do problema para aquela execução.
Será gerado o arquivo “schedule.txt”, com o array da melhor solução convertido em em um cronograma separado por dias, onde cada dia tem o nome do(s) filme(s) que será(ão) assistido(s), e o tempo total assistido em minutos daquele dia.

```
...
>New best by tiebreaker!! fitness([50, 26, 52, 24, 11, 0, 48, 85, 41, 45, 87,
57, 8, 44, 46, 79, 10, 40, 9, 35, 33, 83, 12, 69, 39, 17, 53, 76, 32, 49, 6,
54, 42, 78, 91, 86, 47, 73, 34, 29, 21, 64, 62, 60, 81, 38, 1, 43, 59, 51, 3,
5, 4, 80, 71, 30, 56, 92, 7, 31, 36, 68, 16, 70, 65, 14, 72, 88, 82, 23, 89,
15, 61, 19, 77, 58, 20, 18, 22, 2, 75, 74, 55, 63, 25, 37, 66, 13, 67, 27,
84, 28, 90]) = 64
From generation 999.

>Still the best by tiebreaker!!
>Still the best by tiebreaker!!
>Still the best by tiebreaker!!
############################
Done!

Best offspring:
[50, 26, 52, 24, 11, 0, 48, 85, 41, 45, 87, 57, 8, 44, 46, 79, 10, 40, 9, 35,
33, 83, 12, 69, 39, 17, 53, 76, 32, 49, 6, 54, 42, 78, 91, 86, 47, 73, 34,
29, 21, 64, 62, 60, 81, 38, 1, 43, 59, 51, 3, 5, 4, 80, 71, 30, 56, 92, 7,
31, 36, 68, 16, 70, 65, 14, 72, 88, 82, 23, 89, 15, 61, 19, 77, 58, 20, 18,
22, 2, 75, 74, 55, 63, 25, 37, 66, 13, 67, 27, 84, 28, 90]
It takes 64 days

OPEN "schedule.txt" FILE TO SEE THE SCHEDULE
--- 81.20545363426208 seconds ---
```

Log de execução do algoritmo

```
----Schedule---- (movieId - movieTitle - duration - rating - genre)
Day 1:
50 - The Deer Hunter - 172 min - 8.0 - Drama
WATCHTIME: 172 min

Day 2:
26 - On the Waterfront - 142 min - 8.8 - Crime
WATCHTIME: 142 min

Day 3:
52 - Ordinary People - 153 min - 7.5 - Drama
WATCHTIME: 153 min

Day 4:
24 - The Greatest Show on Earth - 162 min - 7.4 - Drama
WATCHTIME: 162 min

Day 5:
11 - Gone With the Wind - 131 min - 7.6 - Drama
0 - Wings - 107 min - 7.4 - Drama
WATCHTIME: 238 min

...
```

Arquivo com o cronograma gerado

8. Testes

- Testes com n_pop = 300 e n_gen = 1000:

  teste 01: 64 dias, solução encontrada na geração 991
  teste 02: 64 dias, solução encontrada na geração 997
  teste 03: 64 dias, solução encontrada na geração 999
  teste 04: 64 dias, solução encontrada na geração 875
  teste 05: 64 dias, solução encontrada na geração 982
  teste 06: 64 dias, solução encontrada na geração 957
  teste 07: 64 dias, solução encontrada na geração 828
  teste 08: 64 dias, solução encontrada na geração 998
  teste 09: 64 dias, solução encontrada na geração 838
  teste 10: 65 dias, solução encontrada na geração 999

- Testes com n_pop = 200 e n_gen = 800:

  teste 01: 65 dias, solução encontrada na geração 799
  teste 02: 64 dias, solução encontrada na geração 799
  teste 03: 64 dias, solução encontrada na geração 799
  teste 04: 65 dias, solução encontrada na geração 799
  teste 05: 64 dias, solução encontrada na geração 518
  teste 06: 65 dias, solução encontrada na geração 799
  teste 07: 64 dias, solução encontrada na geração 799
  teste 08: 64 dias, solução encontrada na geração 799
  teste 09: 65 dias, solução encontrada na geração 799
  teste 10: 64 dias, solução encontrada na geração 795

Lembrando que, em todos os testes, o valor da geração da solução encontrada corresponde ao último melhor indivíduo, considerando não somente a quantidade de dias, mas também o desempate de média de rating e variação de gênero.

Preferimos os valores n_pop = 300 e n_gen = 1000 pois raramente ele encontra solução pior que 64 dias.