Grupo:
Guilherme Henrique Scarpel - 11711BCC001
Matheus Henrique Ferreira - 11521BCC020

Tema #2: Filmes vencedores do Oscar

Sumário

1. Introdução

Para o problema de filmes vencedores do Oscar abordado neste trabalho, temos acesso a um dataset contendo informações relevantes sobre todos os filmes vencedores do Oscar tais como: rating, duração, título, ano de lançamento, gênero, etc.
O objetivo deste trabalho era implementar um algoritmo genético que fosse capaz de montar uma espécie de cronograma, com uma ordem de filmes a serem vistos em cada dia, de modo a maximizar o rating diário. O indivíduo deve assistir todos os filmes vencedores do Oscar, porém ele só dispõe de 240 minutos diários, isto é no máximo 4 horas assistidas por dia. Não há possibilidade de fracionar filmes, isto é, se o indivíduo começou a assistir um filme em um determinado dia, ele deve, necessariamente, assistí-lo por completo naquele dia. Este limite de tempo diário faz com que, em certos dias, o indivíduo não possa assistir mais de um filme. Esses detalhes serão tratados mais adiante neste relatório.

A linguagem escolhida para a implementação do algoritmo genético foi Python, graças à simplicidade de trabalhar com dados nesta linguagem.
A representação escolhida para os cromossomos foi um vetor de inteiros, que contém os índices dos filmes que podem ser agrupados com outros filmes em um único dia. Como os filmes são sempre os mesmos, inicialmente, ao ler os dados do dataset, é gerada uma lista de filmes (*movies*). Esta lista trata-se de uma lista de dicionários, onde cada dicionário é um filme, mais informações a respeito da representação de um filme podem ser encontradas no Capítulo 3. Para gerar os cromossomos (chamados de samples na implementação), há uma separação entre os filmes que podem ser agrupados com outros filmes e filmes que não

podem ser agrupados com outros filmes. Este processo é feito para otimizar o algoritmo, já que filmes que não podem ser agrupados com outros filmes, isto é, são assistidos sozinhos em um dia, não possuem rating variável e possuem o número fixo de dias necessários para assistí-los.
Exemplo de um cromossomo: [14, 22, 18, 15, 91, 0, 4, ...]
Onde ao invés de utilizar os índices de todos os filmes do dataset, são incluídos apenas aqueles que podem ser agrupados com pelo menos um filme, de modo a otimizar o uso de recursos ao executar o programa.

Uma solução viável para o problema é aquela cuja ordem de filmes produz a maior média de ratings diários. Obtida somando todos os ratings diários e dividindo pelo número de dias necessários para assistir todos os filmes.

2. Características do Dataset

Ao analisar o dataset com os filmes em questão, é possível notar que nem todos os filmes podem ser assistidos em conjunto com algum outro, de modo a aumentar o rating diário de filmes vistos por uma pessoa. Na verdade, 29 dos filmes contidos no dataset tem uma duração grande o suficiente que o torna impossível de assistir em conjunto com o filme de menor duração, Marty, de apenas 90 minutos de duração.
Isolando-os, há 64 filmes que podem ser assistidos junto de mais algum outro em um único dia. Porém, ao ordenar tais filmes por sua duração e pegar somente a primeira ocorrência de cada valor, obtêm-se o seguinte vetor:

[90, 93, 99, 100, 101, 102, 104, 105, 107, 108, 109, 110, 111, 112, 113, 115, 116, 118, 119, 120, 122, 123, 124, 125, 126, 128, 129, 130, 131, 132, 133, 134, 135, 136, 138, 142, 144]

Analisando-o de modo a encontrar o último valor cuja soma com o seus sucessores seja menor ou igual a 240 minutos, o filme de maior duração que ainda pode ser agrupado com um filme que tenha duração superior a dele, possui 119 minutos. É possível observar estes dados na Tabela 1.

Duração	Nº de filmes que pode ser agrupados
90	36
93	35
99	32
100	31
101	30
102	29
104	27
105	25
107	22
108	20
109	18
110	16
111	14
112	12
113	10
115	8
116	6
118	3
119	1

Tabela 1: Número de filmes que podem ser agrupados, dada uma duração.

A primeira coluna representa a duração de um filme, já a segunda, o número de filmes que podem ser agrupados com o filme em questão. O valor para em 119, a maior duração que permite o agrupamento com mais algum outro filme de duração superior na lista. Contudo, é preciso destacar que filmes cuja frequência da duração é maior que um, podem ser agrupados entre si, aumentando o número de possibilidades. Porém, ao analisar o resultado acima, nota-se que, conforme a duração do filme cresce, menor a possibilidade de agrupamento de cada filme.

Com isso em mente, o problema dos filmes vencedores de Oscar torna-se, na verdade, um problema de agrupamento somente dos filmes que podem ser assistidos em conjunto, de modo a maximizar os ratings diários dos filmes. Os outros 29 filmes, que não podem ser agrupados, podem ser descartados momentaneamente do problema, uma vez que só podem ser assistidos em um dia, tornando seu score imutável, assim como a quantidade de dias necessários para os assistir, 29 dias.

Também foi possível notar que o dataset em questão possui uma peculiaridade. Existe uma ocorrência que se trata de uma sequência de um dos filmes vencedores do Oscar, esta sequência é: O Poderoso Chefão 2. Assim, para montar um cronograma válido, o primeiro filme deve, obrigatoriamente, ser assistido antes do segundo filme. Entretanto, por sorte, este mesmo filme possui 202 minutos de duração, impossibilitando-o de ser assistido com qualquer outro filme da lista. O mesmo ocorre com o primeiro filme (O Poderoso Chefão) que possui duração de 175 minutos e não pode ser assistido no mesmo dia de outro filme da lista. Dessa forma, em casos onde o cronograma final colocar o segundo filme precedendo o primeiro, é possível simplesmente trocá-los de ordem, uma vez que isso não afetará a média de scores nem o número de dias gastos para assistir os filmes. Além disso, nenhuma solução é invalidada, prevenindo o descarte e talvez exclusão de um cromossomo que possa ser a melhor solução do problema, senão por esse empecilho.

3. Detalhes do algoritmo

O algoritmo genético foi implementado seguindo a estrutura geracional. Existem alguns parâmetros que podem ser modificados na hora de executar o algoritmo, tais como: número de samples (*numSamples*), número de gerações (*generations*) e quantidade de vezes que o algoritmo será executado (*numTries*).
Inicialmente são gerados os indivíduos da população inicial. Em todas as gerações o processo é o mesmo, os indivíduos são submetidos à função objetivo e recebem um valor (*fitness*) e um vetor (*days*) com os filmes que devem ser assistidos em cada dia. Em seguida, os indivíduos são submetidos à roleta para seleção dos pais e realização do *crossover*. Após a geração dos filhos, pode ou não acontecer o processo de mutação, depois disso os dois piores indivíduos da população são substituídos pelos novos indivíduos gerados no *crossover*. No final, para gerar o cronograma completo, são reunidos os dias que têm apenas um filme com os dias que têm mais de um filme e é feita a validação para sempre o filme Poderoso Chefão ser assistido antes de sua sequência.
No término da execução do algoritmo é gerado um arquivo “log.txt”, que contém algumas informações relevantes. Caso o parâmetro *numTries* escolhido tenha sido maior do que 1, são mostradas estatísticas como: desvio padrão, variância, fitness médio, pior execução, melhor execução, além da melhor solução obtida e seu cronograma. Já se o parâmetro *numTries* tiver recebido o valor 1, é mostrado o valor de fitness obtido e o cronograma correspondente a única execução. É possível observar o início do arquivo “log.txt” gerado nas figuras 1 e 2, que mostram casos de uma execução e de 20 execuções (*numTries*=20), respectivamente.
Mais detalhes sobre o processo citado anteriormente podem ser encontrados nas próximas seções deste capítulo.

```
 - Informações da execução -

Número de tentativas: 1
Fitness obtido: 10.917910447761198

------------------------

 - Melhor solução -

Dias necessários para assistir todos os filmes: 67
Cronograma sugerido:
 - Dia 1
  All About Eve

 - Dia 2
  On the Waterfront
  Mutiny on the Bounty

 - Dia 3
  From Here to Eternity
  Midnight Cowboy
```

Figura 1: Exemplo de execução única do algoritmo

```
 - Informações da execução -

Número de tentativas: 20
Média do valor fitness: 10.926424860989176
Desvio padrão: 0.06450551604744353
Variância: 0.004160961600546995
Pior valor fitness obtido: 10.757352941176476
Melhor valor fitness obtido: 11.083333333333337
Obtido na iteração: 9

-----------------------

 - Melhor solução -

Dias necessários para assistir todos os filmes: 66
Cronograma sugerido:
 - Dia 1
  The Sound of Music

 - Dia 2
  Million Dollar Baby

 - Dia 3
  Schindler's List
```

Figura 2: Exemplo de execução do algoritmo com parâmetro numTries=20

3.1 Representação de um Filme

Um filme no algoritmo será representado utilizando um dicionário, uma vez que ele torna capaz a junção de várias propriedades pertinentes a um objeto em um mesmo lugar. Cada filme terá o formato apresentado na Figura 3 abaixo.

```
{
    id: Identificador do filme (sua posição na tabela de filmes),
    name: Nome do filme
    rating: Rating do filme,
    duration: Duração do filme
}
```

Figura 3: Formato da representação de um filme na forma de dicionário

3.2 Representação de um Cromossomo

Inicialmente é considerado um vetor com todos os 93 filmes do dataset, porém, como dito anteriormente, os filmes que não podem ser agrupados com outros filmes podem

ser descartados momentaneamente, pois como cada um deles vai ocupar um dia de maneira exclusiva, o número de dias para assistí-los é fixo e o rating deles é imutável. Com isso em mente, foi criada uma função no algoritmo chamada *sortMovies*, esta função separa os filmes em dois grupos (filmes que podem ser agrupados e filmes que não podem ser agrupados), esta função pode ser visualizada na Figura 4. Após a geração de uma população com n samples, onde cada sample é um vetor de inteiros representando os índices dos filmes que podem ser agrupados no dataset, a ideia foi utilizar uma estrutura para representar o cromossomo no decorrer do código, agrupando informações referentes a ele. A estrutura escolhida para essa representação foi um dicionário, que contém valores relevantes sobre cada indivíduo da população.
O dicionário utilizado para cada indivíduo da população segue a estrutura apresentada na Figura 5.

```python
# Sort movies in two groups: can and cannot group
def sortMovies(movies, minDuration = 90, watchTime = 240):
  canGroup = []
  cannotGroup = []

  for movie in movies:
    if(movie['duration'] + minDuration <= watchTime): canGroup.append(movie)
    else: cannotGroup.append(movie)

  return [canGroup, cannotGroup]
```

Figura 4: Função que divide os filmes em dois grupos

```
{
    sample: Contém o vetor dos filmes a serem assistidos,
    fitness: O valor de fitness do cromossomo (é reutilizado em casos onde o cromossomo não foi substituído
             na fase do crossover ou não sofreu nenhuma modificação na fase de mutação),
    days: Contém o vetor de vetores de filmes, representando os dias e quais filmes serão assistidos em cada um deles
}
```

Figura 5: Estrutura do dicionário utilizado para representar um indivíduo

3.3 Função Objetivo

Quatro parâmetros são necessários na função objetivo: um vetor que representa a sequência de filmes que podem ser agrupados, a soma dos ratings dos filmes que não podem ser assistidos em conjunto com outros em um dia, o número de dias necessários para assistí-los e o tempo limite diário (este parâmetro pode ser modificado caso haja o desejo de montar cronogramas com outro limite diário, mas por padrão ele recebe o valor 240, correspondente ao limite estipulado para o problema).

O código encontrado na Figura 6 representa a função objetivo criada para solucionar o problema dos filmes.

```python
def getFitness(movies, cannotGroupRatings, cannotGroupDays, watchTime):
  days = []
  ratings = cannotGroupRatings
  duration = 0
  currentRating = 0
  selectedMovies = []

  for movie in movies:
    if(duration + movie['duration'] <= watchTime):
      selectedMovies.append(movie)
      duration += movie['duration']
      currentRating += movie['rating']
    else:
      if(len(selectedMovies)>0):
        days.append(selectedMovies)
        ratings += currentRating

      selectedMovies = [movie]
      currentRating = movie['rating']
      duration = movie['duration']

  days.append(selectedMovies)
  ratings += currentRating

  return [days, ratings/(len(days) + cannotGroupDays)]
```

Figura 6: Função objetivo do algoritmo genético

A função soma a duração dos filmes enquanto ela é menor ou igual a 240, armazenando tais filmes em vetor chamado *selectedMovies*. Além disso, ela soma os ratings de todos eles, para efetuar o cálculo da média de *ratings* diária no final.

Quando a soma da duração do filme atual com a duração de outro filme ultrapassa 240 minutos, a função adiciona o vetor *selectedMovies* no vetor *days*, que representa todos os dias necessários para assistir os filmes. Ademais, a soma de ratings dos filmes contidos no vetor *selectedMovies* é adicionado a um contador de ratings geral, chamado ratings, já inicializado com o valor contido na variável *cannotGroupRatings*. Por fim, a função substitui o vetor de *selectedMovies* por um novo vetor que contém apenas o filme atual, além de substituir o valor contido na variável *currentRating* com o valor de *rating* do mesmo.

O processo se repete até que o último filme seja analisado. Depois disso, a função de fitness retorna o vetor de dias, representando o cronograma dos filmes (que podem ser agrupados, posteriormente, são acrescentados os dias dos filmes que não podem ser agrupados no cronograma), e o valor de fitness em si, constituído da média diária de *ratings* de todos os filmes, inclusive aqueles que não podem ser assistidos em conjunto com outros em um mesmo dia.

3.4 Crossover

Depois que o fitness de todos os cromossomos da população é calculado, chega a hora de aplicar o *crossover*. Para isso é utilizado o método da roleta, onde cada indivíduo tem uma chance de ser selecionado, esta chance é proporcional ao fitness do indivíduo. Assim, os indivíduos com maior valor de fitness têm mais chance de serem selecionados.

Diferente de problemas como o da Mochila (*Knapsack Problem*), onde um cromossomo representa a existência ou não de determinados itens dentro da mochila, aqui um determinado índice no cromossomo é sempre correspondente ao mesmo filme. Dessa forma, ao utilizar operadores de permutação de recombinação, há grandes chances de um filme ser duplicado no cromossomo resultante, tornando-o inválido. Com isso em mente, para efetuar o *crossover* de dois cromossomos no problema dos filmes, foi utilizado um operador de reprodução para permutação. Mais especificamente, o *Crossover* de Ordem (OX), onde uma parte do cromossomo de um dos pais é mantida e o resto é preenchido com os valores do segundo pai, que ainda não estão no novo cromossomo. A Figura 7 ilustra o processo de *crossover* onde dois pais geram dois filhos por meio do *Crossover* de Ordem.

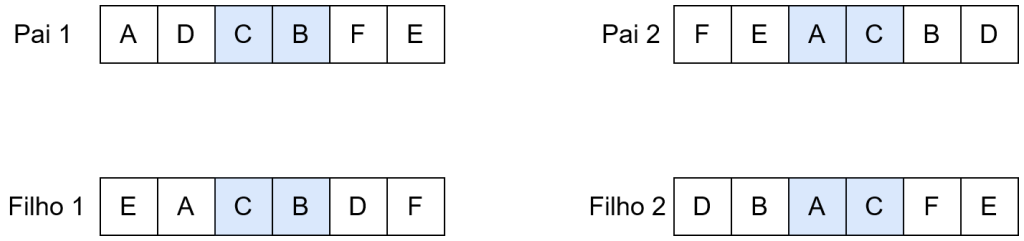

Figura 7: Crossover de Ordem

Como visto na imagem anterior, é separada uma “área de corte” nos dois pais, demarcada por um ponto de corte inicial e um ponto de corte final. Para o Filho 1, a área de corte do Pai 1 é copiada, para o Filho 2 a área de corte do Pai 2 é copiada. Para preencher as posições restantes do Filho 1, são copiados genes do Pai 2 que não pertencem ao Filho 1, começando a partir do gene posicionado após o segundo ponto de corte. Esse processo continua copiando os genes do Pai 2, como se ele fosse uma lista circular. Para preencher as posições restantes do Filho 2 é feito um processo similar ao citado anteriormente, porém pegando os genes faltantes a partir do Pai 1.

Foi utilizado no código um parâmetro chamado *maxCrossoverSplitSize* que inicialmente recebeu o valor de 0.5, pois o tamanho máximo do corte deve ser a metade do cromossomo. O tamanho da área de corte gerada é aleatório, mas esse parâmetro serve como um limite. Nos testes feitos neste trabalho foram utilizados os valores 0.5 e 0.25 para este parâmetro. Na Figura 8 é possível observar a função de *crossover* implementada, onde é gerada uma área de corte aleatória que obedece o parâmetro *maxCrossoverSplitSize* e o processo do *Crossover* de Ordem descrito anteriormente é realizado.

```python
def orderCrossover(p1, p2, maxSplitSize=0.5):
  length = len(p1)
  splitSize = randint(2, int(length * maxSplitSize))
  startSplitAt = randint(0, length - splitSize)
  endSplitAt = startSplitAt + splitSize

  return [ getChild(p2, p1[startSplitAt: endSplitAt], startSplitAt),
    getChild(p1, p2[startSplitAt: endSplitAt], startSplitAt) ]
```

Figura 8: Função que implementa o Crossover de Ordem

3.5 Mutação

A mutação neste problema é um caso especial de permutação. A partir de um cromossomo, um valor aleatório é gerado. Se ele for menor que um pequeno valor (0.1), o cromossomo sofre mutação.
A forma de mutação utilizada no algoritmo foi a seguinte: se o cromossomo foi escolhido para sofrer mutação, um novo valor é gerado aleatoriamente. Ele estará no intervalo [1, n], onde n, por padrão, foi definido como 10, sem nenhum motivo em especial. Este valor representará o número de permutações que o cromossomo atual sofrerá.

Com ele serão gerados dois novos valores aleatoriamente, representando índices distintos do cromossomo. O valor contido em cada um dos índices, que são filmes, serão trocados de lugar. O processo se repetirá o número de vezes definido pelo primeiro valor gerado. A Figura 9 ilustra a função que implementa a mutação no algoritmo genético.

```python
def applyMutation(arr, mutationRate=0.1, maxMutations=10):
  if(random() <= mutationRate):
    length = len(arr)

    for _ in range(randint(1, maxMutations)):
      arr['sample'] = swap(arr['sample'], *generateTwoIndex(length))

    arr['fitness'] = None

    return arr
  else: return arr
```

Figura 9: Função que aplica a mutação

4. Experimentos e Resultados

Foram realizados experimentos com diferentes valores para os parâmetros *numTries*, *generations, numSamples* e *maxCrossoverSplitSize.* Abaixo são apresentadas as

tabelas 2 e 3 com resultados obtidos nos experimentos utilizando o parâmetro *maxCrossoverSplitSize*=0.25. Para a Tabela 2, além do *maxCrossoverSplitSize* citado foram utilizados *numTries*=50, *generations*=1000 e *numSamples*=100.
Já para a Tabela 3 foram utilizados os parâmetros: *numTries*=50, *generations*=5000 e *numSamples*=100.

Resultados	
Média (Fitness)	10.9121
Desvio Padrão	0.0933
Variância	0.0087
Pior Fitness	10.7573
Melhor Fitness	11.2538
Dias	65

Tabela 2: Resultados para 50 execuções com 100 indivíduos e 1000 gerações (maxCrossoverSplitSize = 0.25)

Resultados	
Média (Fitness)	11.2541
Desvio Padrão	0.0606
Variância	0.0036
Pior Fitness	11.0833
Melhor Fitness	11.4296
Dias	64

Tabela 3: Resultados para 50 execuções com 100 indivíduos e 5000 gerações (maxCrossoverSplitSize = 0.25)

Como observado nas tabelas acima, apenas aumentando o número de gerações de 1000 para 5000, foi possível obter a solução na qual são necessários 64 dias para assistir todos os filmes, a solução ótima para este problema. Com 1000 gerações o programa demorou 27 segundos para ser executado 50 vezes. Já com 5000 gerações, o programa passou a ser executado 50 vezes em, aproximadamente, 2 minutos.
O programa também foi testado com 2500 gerações e os mesmos parâmetros descritos anteriormente e o cronograma final de filmes foi de 65 dias, igual ao resultado obtido para 1000 gerações. É possível observar também pelas tabelas anteriores a variância e o desvio padrão baixos para ambos experimentos com diferentes números de gerações.

Também foram realizados outros experimentos, com *maxCrossoverSplitSize*=0.5. Os outros parâmetros foram iguais aos anteriores: *numTries*=50, *generations*=1000 e *numSamples*=100 para os resultados da Tabela 4 e para a Tabela 5 foram utilizadas 5000 gerações.

Resultados	
Média (Fitness)	10.8826
Desvio Padrão	0.0748
Variância	0.0056
Pior Fitness	10.7573
Melhor Fitness	11.0833
Dias	66

Tabela 4: Resultados para 50 execuções com 100 indivíduos e 1000 gerações (maxCrossoverSplitSize = 0.5)

Resultados	
Média (Fitness)	11.2576
Desvio Padrão	0.0654
Variância	0.0042
Pior Fitness	11.0833
Melhor Fitness	11.4296
Dias	64

Tabela 5: Resultados para 50 execuções com 100 indivíduos e 5000 gerações (maxCrossoverSplitSize = 0.5)

Nos resultados apresentados acima, é possível observar que mesmo com a variável maxCrossoverSplitSize com valor 0.5 a solução ótima foi encontrada para 5000 gerações. Já para o caso de 100 indivíduos com 1000 gerações foi gerado um cronograma de 66 dias.

5. Conclusão

Após a realização de diferentes experimentos com parâmetros distintos e várias execuções do algoritmo (inclusive com diferente número de samples), foi possível observar que o desvio padrão e a variância se mantiveram baixos em todos os experimentos. Isso significa que os dados se mantiveram próximos da média e não estavam espalhados por uma ampla gama de valores.

A implementação de um algoritmo genético como este, que trata o problema dos filmes do Oscar, com o intuito de criar um cronograma para assistí-los maximizando o rating diário, foi muito interessante, pois permitiu que conceitos teóricos estudados ao longo do semestre pudessem ser aplicados na prática.