# Elenco Representativo


### UNIVERSIDADE FEDERAL DO PARANÁ - UFPR

03/setembro/2022

**Eduardo Gobbo Willi Vasconcellos Gonçalves - GRR20203892**


---

## Informações

Foram utilizadas as bibliotecas comuns de **Python**, como **sys**, **time** e **cmath** para retornar o relatório na saída **stdout**, para calcular o **tempo** de execução no relatório do terminal e para inicializar o valor ótimo do custo do elenco em **+infinito**, respectivamente.

Logicamente, o trabalho foi totalmente desenvolvido em Python, executado com **#!/bin/python3.9**.

Versão local de python utilizada para desenvolvimento: **v3.9.5**, versão do dinf: **v3.9.2**.

## O Problema

Dado um conjunto de **Atores A**, onde cada ator **Representa R** grupos e tem um **Custo C** de contrato. Qual o **Elenco E** de menor custo para uma peça com **N Personagens** cujo elenco deve representar **G Grupos**.

Temos então que queremos:

$$\text{Min} \sum_{e \in E} e.C$$

Respeitando as condições:

$$\bigcup_{e \in E} e.R = G$$
$$E \subseteq A$$
$$|E| = N$$

## Modelagem

Foi utilizada a técnica de Branch and Bound vista em aula para resolver este problema, sendo o pseudo código abaixo uma representação do algoritmo aplicado.

```
def Branch_and_Bound(l: int):
        global X, optX, optP, Cl # (l= 0, 1, ...)
        if(X is solucao_viavel):
                P = Custo(X)
                if(P < optP):
                        OptP = P
                        optX = X

        # Computar escolhas
        if l == atores:
                Cl = []
        elif X is viavel:
                Cl = [0, 1]
        else
                Cl = [0]


        # Percorre Arvore de decisões
        nextchoice = []
        nextbounds = []
        count = 0
        for a in Cl:
                X_l = a
                nextchoice[count] = a
                nextbounds[count] = Bound(X)
                count = count + 1

        # Decide elenco
        for i in 0..count-1:
                if(nextbounds[i]) >= OptPrice:
                        return
                X[l] = nextchoice[l]
                Branch_and_Bound(l+1)
```

Onde a função `solucao_viavel` representa todas as restrições mencionadas na sessão problema.

A função `Viavel` é utilizada para realizar cortes por viabilidade na árvore, onde esta função retorna o resultado da expressão abaixo, dado um vetor de escolhas $X \subseteq \{0, 1\}$:

$$\sum_{x \in X} x < N$$

Observe que este corte não toma em conta os grupos dos atores escolhidos por $X$.
Contudo, este limitante mais simples impede que haja uma poda indevida da árvore, onde as subárvores podadas poderiam conter soluções ótimas.

Por último, existem duas implementações da função `Bound`, a função dada e a função modelada nova. A primeira modelagem é bem simples e não toma em conta a possibilidade de diferentes atores terem diferentes custos:

$$\text{Bound}_{dada}(X) = \sum_{a \in E} a.C \ + (N - |E|) * min\{f.C \mid f \ \in \ A \backslash E\}$$

Pois caso faltem $K$ atores a serem escolhidos para $K$ personagens, assume-se que o menor custo disponível no conjunto de atores é o mesmo para todos os $K$ atores.

Logo, a versão nova da função `Bound` toma em conta este ponto negativo da função dada,
sendo sua modelagem como a seguinte:

$$\text{Falta} = ordem\_crescente(\{f.C \mid A \setminus E\})$$
$$\text{Bound}_{propria}(X) = \sum_{a \in E} a.C \ + \sum_{f \in Falta} f.C$$

Onde para $K$ vagas restantes, é escolhido os $K$ atores mais baratos disponíveis. Novamente a questão de desconsiderar os grupos representados pelos atores mais baratos revem a tona, caso tivesse sido implementado algo do gênero, seria necessário tomar um cuidado extra ao podar uma subárvore. Pois poderia existir uma solução ótima cujo conjunto de grupos antecedentes nos nós da árvore fossem podados.

## Análise dos Bounds

Fazendo uma análise com o caso de existirem
**10 grupos**, com **20 atores** e **5 Personagens**, foram obtidos os seguintes resultados:

### Resultados:

### Todos os atores custam 1000

- Sem nenhum tipo de poda

```
./elenco -o -f          < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 2097151 nodos na arvore de busca
A execucao da busca durou 10557.246208190918 ms
```

- Com Bound Dado

```
./elenco -a -f  < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 1966098 nodos na arvore de busca
A execucao da busca durou 30663.28239440918 ms
```

- Com Bound Próprio

```
./elenco -f     < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 1966098 nodos na arvore de busca
A execucao da busca durou 31094.749927520752 ms
```

- Com viabilidade

```
./elenco -o     < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 82159 nodos na arvore de busca
A execucao da busca durou 336.902379989624 ms
```

- Com Viabilidade **E** Bound Dado

```
./elenco -a     < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 82024 nodos na arvore de busca
A execucao da busca durou 987.4255657196045 ms
```

- Com Viabilidade **E** Bound Próprio

```
./elenco        < testes/hollywood.txt
1 2 3 4 20
5000
foram visitados 82024 nodos na arvore de busca
A execucao da busca durou 1026.8990993499756 ms
```

### Todos os Atores tem custos entre 0 e 1000

- Sem nenhum tipo de poda

```
./elenco -o -f          < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 2097151 nodos na arvore de busca
A execucao da busca durou 10516.110897064209 ms
```

- Com Bound Dado

```
./elenco -a -f  < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 1966099 nodos na arvore de busca
A execucao da busca durou 32314.927101135254 ms
```

- Com Bound Próprio

```
./elenco -f     < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 1966099 nodos na arvore de busca
A execucao da busca durou 31389.797687530518 ms
```

- Com Viabilidade

```
./elenco -o     < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 82159 nodos na arvore de busca
A execucao da busca durou 331.6318988800049 ms
```

- Com Viabilidade **E** Bound Dado

```
./elenco -a     < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 82025 nodos na arvore de busca
A execucao da busca durou 1076.5049457550049 ms
```

- Com Viabilidade **E** Bound Próprio

```
./elenco        < testes/wollyhood.txt
1 2 3 4 18
1985
foram visitados 82025 nodos na arvore de busca
A execucao da busca durou 1149.3840217590332 ms
```

## Conclusão

Como esperado, em ambas as variações de custo, a busca sem nenhum tipo de poda obteve o maior tempo de busca e percorre todos os possíveis nodos da árvore.

Pode-se observar uma grande diferença no tempo de execução entre cortes somente com **viabilidade** ativo e qualquer outra opção, onde somente *viabilidade* é severamente mais rápido que os demais. Isso se deve ao fato de que, para um numero baixo de personagens a serem escolhidos, a poda por viabilidade já é suficiente para eliminar o espaço de busca. O que é revelado ao se analisar o tempo de execução dos demais, onde o tempo de execução da função `Bound` não é vantajosa para um numero pequeno de personagens.

Agora ao analisar o tempo de execução entre **Bound próprio** e **Bound dado** com *viabilidade* ativo, observamos que ambas as soluções percorrem o mesmo número de nodos, visto que ambas cortam somente 134 nodos a mais na arvore em relação aos cortes com somente *viabilidade* ativo. Devido ao mesmo motivo mencionado acima, existe somente uma pequena diferença entre seus tempos de execução.

Entre **Bound próprio** e **Bound dado** sem cortes por *viabilidade*, temos um número de nodos visitados generosamente maior, quando comparado com *viabilidade* ativo. Isto se deve ao fato de a profundidade percorrida na árvore é muito maior, onde existem somente 5 papéis de personagens a serem preenchidos por atores, a árvore com podas por viabilidade tem uma profundidade de no máximo 5. Em contrapartida, dependendo da avaliação do Bound, a profundidade de busca pode chegar até 20, que é o tamanho de uma busca exaustiva.

Nos testes mostrados acima não houve uma clara diferença de performance entre `Bound_Dada` e `Bound_propria` na métrica de nodos percorridos, utilizando custos para cada ator aleatoriamente. Contudo neste último exemplo abaixo é ilustrado uma situação em que os cortes de `Bound_propria` percorrem menos nodos que `Bound_Dada`, devido a uma escolha mais atenta de custos dos atores:

**8 Grupos, 28 Atores & 5 Personagens**

```
./elenco -f < testes/richard.txt
1 9 11 17 22
22
foram visitados 2432 nodos na arvore de busca
A execucao da busca durou 58.78949165344238 ms

./elenco -f -a < testes/richard.txt
1 9 11 17 22
22
foram visitados 3036 nodos na arvore de busca
A execucao da busca durou 71.79713249206543 ms
```

Onde podemos notar que o número de nodos percorridos com o Bound próprio é significativamente menor que a Bound Dada, notando que em relação ao espaço de busca completo, o limitante exclui grande parte deste espaço mesmo sem cortes por viabilidade.