Otimização - Trabalho 2

Gabriel de Oliveira Pontarolo `GRR20203895`
Rodrigo Saviam Soffner `GRR20205092`

Setembro 2022

1 Introdução

Esse artigo trata da descrição de uma possível implementação utilizando um algoritmo com a técnica de *branch and bound* para o problema do elenco, o qual consiste em selecionar um determinado subconjunto de atores que atendem a certos requisitos de forma a diminuir o custo total.

2 O problema

Uma produtora de filmes precisa selecionar um elenco para um filme de forma a minimizar o custo de produção ao mesmo tempo que todos os grupos sociais elencados previamente sejam representados. Temos que:

- O conjunto S representa os grupos sociais;
- O conjunto A representa os atores que podem ser selecionados;
- O conjunto P representa os papéis que podem ser dados aos atores;
- Para cada ator $a \in A$ temos associado um conjunto $S_a \subseteq S$ indiciando os grupos dos quais faz parte e um custo v_a;
- O subconjunto $X \subseteq A$ representa os atores selecionados, ou seja, a solução do problema;
- Para que a solução seja viável $|X| = |P|$;
- Para que a solução seja viável $\bigcup_{a \in X} S_a = S$;
- O valor de $\sum_{a \in X} v_a$ deve ser mínimo;

3 A modelagem

Nessa sessão será feita a análise das decisões feitas em relação a modelagem do algoritmo de *branch and bound*, com ênfase nos aspectos de viabilidade e otimalidade, assim como a geração dos nós da árvore gerada pelas chamadas recursivas.

Iremos inicializar o conjunto $X_{opt} \leftarrow \emptyset$ e a variável $opt \leftarrow \infty$ (pois se trata de um problema de minimização) que representam o conjunto X ótimo atual e $\sum_{a \in X_{opt}} v_a$, respectivamente. Antes da execução do algoritmo em si, o conjunto A, implementado utilizando um vetor, foi ordenado de acordo com os valores v_a, em ordem crescente, para ser utilizado futuramente na função de *bound*.

Assim, para saber se atingiu um nó folha e a solução é viável, o algoritmo executa o teste $|X| = |P|$ e $\bigcup_{a \in X} S_a = S$. Em caso verdadeiro, se $\sum_{a \in X} v_a \leq opt$, então $opt \leftarrow \sum_{a \in X} v_a$ e $X_{opt} \leftarrow X$. Note que, no primeiro nó folha viável, o teste esse teste sempre será verdadeiro. Também é possível que ele atinja um nó folha quando $|X| = |P|$ e $\bigcup_{a \in X} S_a = S$ é falso, porém $A = \emptyset$ é verdadeiro. Nesse caso, a solução é inviável.

Se o algoritmo não atingiu um nó folha, ele irá testar se ainda é possível gerar uma solução viável naquele ramo da árvore, ou seja, se $|X| + |A| \geq |P|$ e $(\bigcup_{a \in X} S_a) \cup (\bigcup_{a \in A} S_a) = S$. Caso seja falso, tal ramo da árvore será "cortado", ou seja, não irá executar a chamada recursiva.

O último teste executado antes da chamada recursiva utiliza uma função de *bound*, ou limitante, para saber se ainda é possível atingir um valor melhor que o ótimo naquele ramo. Note que, para que haja um ganho no algoritmo, a função escolhida para calcular o limitante precisa ser mais eficiente do que a "descida" até a folha do ramo atual da árvore. Entretanto, o valor retornado não precisa ser exato, pois servirá apenas como um limite superior, mas deve ser "otimista", retornado um valor melhor do que aquele que seria obtido no ramo. Nesse caso, sendo este um problema de minimização, temos que se *bound(X, A)* $\leq$ *opt* é falso, a chamada recursiva não será executada. A análise da função escolhida será feita na próxima sessão desse artigo.

Por fim, são executadas duas chamadas recursivas. Na primeira, um elemento de A é removido e do conjunto e inserido em X. Na segunda, é apenas removido um elemento de A.

4 A função limitante

Foram feitos testes com duas funções de *bound* diferentes, sendo a primeira delas:

$$B_1(X,\ A) = \sum_{a \in X} + \ (|P| - |X|) \times min\{v_a \mid a \in A\}$$

Onde $B_1(X,\ A)$ é a soma dos valores v_a dos atores já escolhidos no conjunto X mais o valor mais baixo do conjunto A multiplicado pelo número restante de papéis. Observe que essa função ignora as restrições de grupos sociais e a limitação de apenas um papel por ator, para se encaixar nas especificações previamente citas para uma função de *bound*.

A segunda função limitante se utiliza do fato de que, como foi previamente citado, o conjunto A foi implementado utilizando um vetor e este foi previamente ordenado de acordo com o valor v_a de cada ator em ordem crescente. Desse modo, temos:

$$B_2(X,\ A) = \sum_{a \in X} + \ G(X,\ A)$$

Function `G(X, A)`**:**
- $i \leftarrow |P|$ - $|X|$
- $s \leftarrow 0$
- **for** $a \in A$ **do**
  - **if** i $\leq$ *0* **then**
    - **return** s
  - **end**
  - $s \leftarrow s + v_a$
  - $i \leftarrow i$ - 1
- **end**
- **return** s

Onde $B_2(X,\ A)$ é a soma dos valores v_a dos atores já escolhidos no conjunto X mais a função $G(X,\ A)$, a qual consiste de um algoritmo *guloso* que faz a soma dos valores v_a para os $|P|$ - $|X|$ primeiros $a \in A$, levando em conta o fato de que A está ordenado de acordo com o valor de cada ator em ordem crescente. Nota que essa, além de se encaixar nas especificações para função de *bound*, pode ser dita "melhor" do que $B_1(X,\ A)$, pois ela ignora apenas a restrição dos grupos sociais, retornando um valor mais próximo do qual seria obtido ao descer até a folha e, consequentemente, "cortando" mais ramos. Será possível notar a diferença na performance entre as duas sessão 6 desse artigo.

5 O algoritmo

Finalmente, temos a implementação do algoritmo de *branch and bound* para a solução do problema. Dados os conjuntos A, P e S; X_{opt} e opt globais; a função *sort* externa que ordena de acordo com os valores v_a de cada ator em ordem crescente; a função *bound* igual a B_1 ou B_2 previamente citadas, temos:

```
opt ← ∞
X_opt ← ∅
A ← sort(A)
X ← ∅
BB(X, A)

Function BB(X, A):
    if |X| = |P| and ⋃_{a∈X} S_a = S then
        if Σ_{a∈X} v_a ≤ opt then
            opt ← Σ_{a∈X} v_a
            X_opt ← X
        end
    else if A = ∅ then
        return
    else
        if (|X| + |A| ≥ |P|) and ((⋃_{a∈X} S_a) ∪ (⋃_{a∈A} S_a) = S) then
            B ← bound(X, A)
            if B ≥ opt then
                return
            else
                x ← A[1]
                A ← A - A[1]
                BB(X, A)
                BB(X ∪ {x}, A)
            end
        else
            return
        end
    end
```

6 Testes com o algoritmo

A partir do pseudocódigo, foi construído um programa com a linguagem de programação *C++* para que sejam executados os testes. A entrada (**stdin**) consiste dos valores $l = |S|$, $m = |A|$ e $n = |P|$ na primeira linha. Em seguida temos m blocos, um para cada ator, onde temos os valores v_a, $|S_a|$ e cada um dos $s \in S_a$ grupos do qual esse ator faz parte. Será considerado $S = \{1, \cdots, l\}$ e $A = \{1, \cdots, m\}$. A saída (**stdout**) consiste dos valores de X_{opt} na primeira linha e o valor de *opt* na segunda. Além disso, ele também imprime (na saída de erro **stderr**) o tempo de execução da função de *branch and bound* e o número de nós gerados na árvore. Com isso em mente, seguem alguns exemplos da execução comparando as duas funções B_1 e B_2:

- **Entrada 1:**
  2 3 2 10 2 1 2 20 1 2 5 2 1 2

  Função B_1:
  1 3
  15
  Tempo: 383 microssegundos
  Nodes: 10

  Função B_2:
  1 3
  15
  Tempo: 299 microssegundos
  Nodes: 10

- **Entrada 2:**
  10 10 7 10 3 1 4 5 20 4 1 6 7 8 15 2 8 3 30 4 1 2 3 5 14 3 4 5 6 18 5 2 4 6 8 10 13 2 7 10 17 3 3 5 7 15 1 9 19 4 7 9 10 1

  Função B_1:
  1 3 5 6 7 8 9
  102
  Tempo: 45897 microssegundos
  Nodes: 626

  Função B_2:
  1 3 5 6 7 8 9
  102
  Tempo: 24197 microssegundos
  Nodes: 174

- **Entrada 3:**

  10 25 10 27 6 1 3 4 5 7 8 54 6 1 2 6 7 8 9 10 1 1 12 9 1 2 3 4 5 6 7 9 10 18 2 3 5 36 4 8 9 10 5 49 5 3 5 6 7 10 14 2 3 7 40 6 1 2 3 5 6 8 10 6 2 3 5 6 9 10 36 4 10 2 4 6 11 3 1 4 5 19 9 1 3 4 5 6 7 8 9 10 10 4 10 3 6 7 51 9 1 2 3 4 5 7 8 9 10 11 3 8 10 5 22 3 9 2 10 58 7 2 3 6 7 8 9 10 40 4 9 10 3 6 25 6 4 6 7 8 9 10 27 6 1 2 4 5 9 10 84 9 1 2 3 4 5 7 8 9 10 30 8 1 2 3 4 5 6 7 8 52 10 1 2 3 4 5 6 7 8 9 10 10 2 3 7

  Função B_1:
  3 4 5 8 10 12 13 14 16 25
  125
  Tempo: 238651557 microssegundos
  Nodes: 3293208

  Função B_2:
  3 4 5 8 10 12 13 14 16 25
  125
  Tempo: 170358 microssegundos
  Nodes: 1454

- **Entrada 4:**
  25 75 30 86 11 1 4 6 11 12 13 14 17 19 21 23 36 14 2 3 4 5 7 10 11 12 13 14 15 16 19 24 10 1 23 43 9 1 3 6 9 15 16 18 19 25 16 8 7 8 13 14 17 19 21 22 180 18 1 2 3 6 7 9 10 11 12 13 15 16 17 19 20 22 23 25 70 15 1 3 4 6 7 8 10 11 17 18 19 20 22 24 25 100 12 1 4 5 9 13 14 17 18 20 22 24 25 58 7 7 8 10 16 19 24 25 16 4 19 18 11 7 219 25 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 25 20 1 2 4 5 6 7 8 9 10 11 13 14 15 16 17 18 19 21 23 25 10 1 2 155 24 1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 32 8 1 5 9 13 17 22 24 25 16 4 16 9 18 25 48 7 2 8 12 17 21 22 23 54 20 2 3 4 5 6 7 8 10 11 12 13 14 16 17 18 19 20 22 24 25 23 3 9 12 22 22 6 3 6 8 14 17 18 10 1 14 17 21 1 2 5 6 7 8 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 34 5 1 2 12 16 25 12 6 1 5 8 15 16 19 10 4 16 8 19 6 10 3 1 13 22 11 2 4 6 99 10 2 5 9 10 11 17 18 20 22 25 10 1 22 103 24 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17 18 19 20 21 22 23 24 25 149 20 1 2 3 4 5 7 8 9 11 12 13 15 16 17 20 21 22 23 24 25 39 23 1 2 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 113 18 1 2 3 5 6 7 8 9 10 14 15 16 17 19 22 23 24 25 37 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 114 14 6 7 8 10 12 13 14 16 18 20 21 22 23 24 183 20 1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 17 18 20 23 25 25 18 1 4 5 6 7 8 11 12 13 14 16 19 20 21 22 23 24 25 29 4 12 4 13 23 67 7 12 13 16 17 19 23 25 9 1 11 34 25 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 201 21 1 2 3 5 6 7 8 9 10 11 12 14 15 16 17 18 19 20 21 24 25 107 15 1 2 3 4 5 8 10 13 14 15 19 20 23 24 25 29 12 2 4 6 7 9 10 16 18 20 22 23 25 105 22 1 2 3 4 5 6 7 8 11 12 13 14 16 17 18 19 20 21 22 23 24 25 176 21 1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 18 21 22 23 24 25 37 7 4 10 14 17 18 21

25 25 23 1 2 3 4 5 6 7 9 10 11 12 13 14 15 16 17 18 19 20 21 23 24 25 96
14 1 2 4 8 10 11 12 14 15 17 20 22 23 24 77 19 3 5 6 7 8 9 10 11 12 14 15
16 18 19 20 21 22 24 25 12 12 2 3 6 8 9 11 14 15 16 18 23 25 10 1 4 30 3 5
14 22 49 14 2 3 4 8 10 11 13 15 16 17 19 20 22 24 66 23 1 2 3 5 6 7 8 9 10
11 12 13 14 15 16 18 19 20 21 22 23 24 25 46 5 2 7 22 23 24 10 10 4 7 8
12 14 15 16 17 22 23 190 20 1 3 4 5 6 7 8 9 10 11 13 14 15 17 19 20 21 22
23 24 38 5 3 7 14 17 19 97 12 1 2 8 9 10 12 13 15 17 20 24 25 9 1 11 100
13 4 5 9 11 13 14 15 16 17 20 21 23 24 60 6 2 4 7 16 23 25 16 8 1 12 14 16
17 19 21 24 70 17 2 4 5 6 7 8 9 10 11 12 13 14 16 17 22 23 24 50 20 1 2 3
5 6 7 8 10 11 12 13 14 17 18 19 20 21 22 24 25 13 2 1 5 110 11 2 3 6 8 10
15 16 17 20 21 25 98 10 4 8 10 16 18 19 20 21 22 23 99 10 1 3 4 5 9 17 18
20 23 24 10 3 4 5 15 124 16 2 3 4 5 8 9 11 12 16 17 18 19 20 21 22 25 10 2
19 15 29 5 3 4 11 16 17 17 17 1 2 6 7 8 9 10 11 12 13 14 15 17 18 20 21 25

Função B_1:
Não foi possível testar essa entrada na com a função B_1 devido ao tempo de execução

Função B_2:
3 5 10 12 13 16 19 20 21 22 24 25 26 27 29 37 38 40 44 48 51 52 57 61 64
67 71 73 74 75
471
Tempo: 20459462 microssegundos
Nodes: 31424