

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 $S_a \subseteq S$ indicando 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:

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