Aluno: André Miguel Sikora Marques GRR20182593 Login: amsm18
Trabalho 2 CI1238

Explicação do problema (Elenco Representativo)

Uma produtora de filmes quer fazer um filme e está a procura do elenco ideal. Tentando evitar polêmicas de representatividade do elenco em relação a grupos da sociedade, a produtora quer que todos os grupos (previamente elencados) da sociedade sejam representados. Um ator pode fazer parte de mais de um destes grupos, portanto podemos ter menos atores que grupos. Obviamente, é preciso que cada personagem do filme tenha um ator associado e cada ator cobra um valor para fazer o filme. Dados um conjunto S de grupos, um conjunto A de atores, um conjunto P de personagens, e, para cada ator $a \in A$, um conjunto, $S_a \subseteq S$ indicando os grupos dos quais a faz parte, devemos encontrar um elenco que tenha um ator para cada personagem (todos os atores podem fazer todas as personagens) e todos os grupos tenham um representante. Além disso, também temos um valor, v_a, associado com cada ator $a \in A$, e queremos que o custo do elenco seja mínimo.

Ou seja, devemos encontrar um subconjunto $X \subseteq A$ tal que:

- $|X| = |A|$;
- $\bigcup_{a \in X} S_a = S$;
- $\sum_{a \in X} v_a$ seja mínimo

Modelagem

A modelagem foi baseada nas discussões com o professor, que levaram ao seguinte pseudocódigo:

$Branch\_and\_bound(k, i, X)$
    $se\ k\ ==\ p\ \ e\ \ X$ é $representativo$
        $se\ custo(X) < opt$
            $opt \leftarrow custo(X)$
            $x\_opt \leftarrow X$
    $senão$
        $para\ j\ de\ i\ +\ 1\ até\ n$
            $se\ X$ é $viável\ e\ X$ é $ótimo$
                $X[j\ +\ k] \leftarrow j$
                $k\ ++$
                $Branch\_and\_bound(k, j, X)$
                $k\ --$

Ou seja, se há atores suficientes para todos os personagens em X, os atores destes personagens representam todos os grupos necessários, e esta solução é melhor que a anterior, então esta é a nova solução. Caso contrário, a função é chamada novamente para cada ator que ainda não está na solução. Aqui entram os cortes por viabilidade e otimalidade, se a próxima lista de atores não é viável nem ótima, então este nó não é executado.

Análise das funções limitantes

A função dada pelo professor foi:

$$B_{dada}(E, F) = \sum_{a \in E} v_a + (n - |E|) min\{v_a | a \in F\}$$

que dá a soma dos valores cobrados pelos atores já escolhidos, somado com o valor mais baixo cobrado pelos atores que ainda faltam decidir, multiplicado pelo número de personagens ainda sem ator.

A outra função desenvolvida faz algo parecido, que é a soma dos valores dos atores já escolhidos, somado com a soma dos valores dos atores mais baratos ainda não selecionados que preenchem o número de personagens restantes.

Como os exemplos utilizados são pequenos, o número de nós criados foi o mesmo, mas em exemplos maiores a segunda função tende a gerar menos nós e ser mais rápida.

Detalhes da implementação

```
def branch_and_bound(k, i, X):
    global n_personagens, n_grupos, n_atores, atores, custo_otimo, x_otimo, nos_criados

    nos_criados += 1

    if k == n_personagens:
        if representativo(X) and custo(X) < custo_otimo:
            custo_otimo = custo(X)
            x_otimo = X.copy()
        return
    else:
        j = i + 1
        while j <= n_atores:
            X.append(atores[j])
            if not viavel(X) or not otimo(X):
                X.pop()
                return
            k += 1
            branch_and_bound(k, j, X)
            k -= 1
            X.pop(-1)
            j = j + 1
```

```
def otimo(X):
    global n_personagens, atores, custo_otimo, opcao_A

    if opcao_A: #Professor
        custo_escolhidos = custo(X)
        nao_escolhidos = []
        for ator in atores:
            if ator not in X:
                nao_escolhidos.append(ator)
        minimo = 999999999999
        for ator in nao_escolhidos:
            if ator == None:
                continue
            if ator.valor < minimo:
                minimo = ator.valor
        personagens_sem_ator = n_personagens - len(X)
        flimitante = custo_escolhidos + (minimo * personagens_sem_ator)
        if custo_otimo > flimitante:
            return True
        return False
    else:   #Aluno
        custo_escolhidos = custo(X)
        nao_escolhidos = []
        for ator in atores:
            if ator not in X and ator != None:
                nao_escolhidos.append(ator)

        nao_escolhidos.sort(key=custo_ator)
        personagens_sem_ator = n_personagens - len(X)
        custo_nao_escolhidos = 0
        for i in range(personagens_sem_ator):
            custo_nao_escolhidos += nao_escolhidos[i].valor

        flimitante = custo_escolhidos + custo_nao_escolhidos
        if custo_otimo > flimitante:
            return True
        return False
```

A função `branch_and_bound(k,i,X)` faz praticamente a mesma coisa que o descrito no pseudocódigo, com a inclusão de um `pop()` para remover o ator tentado na recursão anterior, e uma variável para calcular o número de nós criados. Se o valor da variável `custo_otimo` for 999999999999 após a execução dessa função, significa que o problema dado é inviável, pois este é o valor com o qual inicializa a variável.

A função limitante criada pelo aluno ordena a lista de não escolhidos e soma os menores valores, ao invés de encontrar o mínimo valor e multiplicar pelo número de personagens que faltam ser preenchidos, como na do professor.

Exemplos utilizados: