Relatório do Segundo Trabalho

Vinícius Maurício Ribeiro

Introdução

O objetivo deste trabalho é resolver o problema do Elenco Representativo através do algoritmo *Branch and Bound.*

O problema consiste em encontrar o custo mínimo de um elenco de atores onde todos os grupos da sociedade previamente escolhidos são representados. Cada ator cobra um valor e faz parte de alguns grupos. Além disso, cada ator pode interpretar qualquer personagem do elenco. É possível haver mais atores que personagens do elenco. Nesse caso, os atores restantes não são escolhidos.

Mais informações sobre o problema disponíveis em trabalho2.pdf.

Modelagem

Variáveis teóricas do problema:

l: (constante) total de grupos;

m: (constante) total de atores;

n: (constante) total de personagens;

A: (constante) conjunto de todos os atores (A = {1..m});

S: (constante) conjunto de todos os grupos (S = {1..l}) (não utilizado);

P: (constante) conjunto de todos os personagens (P = {1..n}) (não utilizado);

vi: (constante) valor cobrado pelo ator i;

Si: (constante) conjunto de grupos dos quais o ator i faz parte;

E: (variável) conjunto de atores atualmente escolhidos;

F: (variável) conjunto de atores atualmente não escolhidos (E U F = A);

c: (variável) custo total do conjunto de atores escolhidos;

B: (variável) valor limitante dado pela função de *bound;*

Obs: valores ditos constantes são para uma instância do problema.

A solução do problema, isto é, um conjunto de atores X que satisfaz os requisitos para o elenco é igual ao conjunto E após a finalização do algoritmo de Branch and Bound.

Para a modelagem do problema foram criadas duas estruturas de dados: o Ator, composto pelo valor cobrado e conjunto de grupos ao qual ele faz parte; e a Escolha, que combina as variáveis E, F, c e B.

Uma escolha pode ser entendida como um nó da árvore de busca do algoritmo de Branch and Bound. Cada nó possui um valor diferente para cada uma das variáveis E, F, c e B.

A criação da estrutura de dados Escolha facilitou a organização dos dados, mas tornou a implementação do algoritmo de Branch and Bound um pouco diferente do apresentado em [1], uma vez que no segundo caso, o autor separa a solução viável atual, o custo e o limitante do conjunto de escolhas.

Já para a modelagem atual foi considerado que uma escolha é composta por uma solução, um custo e um limitante. A solução ser inviável torna a escolha inviável. Apesar de interpretações ligeiramente diferentes, na prática os algoritmos são equivalentes.

Implementação

O trabalho foi implementado em C++, utilizando as bibliotecas fornecidas pela linguagem.

Para a implementação do conjunto de próximas escolhas e conjunto de atores foram utilizadas versões customizadas da TAD set [2]. Essa estrutura de dados armazena seus elementos em ordem; por isso permite passar uma função de comparação para usar em seu algoritmo de sorting interno.

Para o conjunto de atores foi utilizada uma função de comparação de menor valor. Ou seja, o conjunto armazena atores em ordem crescente de valor cobrado. Isso

é especialmente útil para as funções de bound, que precisam dos atores com os menores valores cobrados.

Já para o conjunto de próximas escolhas foi utilizada uma função de comparação do menor limitante; ou seja, o conjunto armazena as próximas escolhas na ordem crescente de limitante/bound. Útil para o próprio algoritmo de Branch and Bound que utiliza essa informação para “visitar” o próximo nó da árvore.

Em relação aos nós da árvore de busca, foi considerado que sempre que as próximas escolhas são geradas, novos nós da árvore são criados. O total de nós da árvore leva isso em consideração.

Análise da função limitante

A função limitante criada é muito semelhante à disponibilizada pelo professor, com a pequena diferença de que, ao invés de multiplicar *x* quantidade de personagens faltantes pelo ator faltante com menor custo, é realizada uma soma dos *x* atores faltantes com menor custo, ou seja:

$$B(E, F) = \sum_{a \in E} v_a + \sum_{i=1}^{(n-|E|)} v_{fi}$$

Onde vf é uma tupla ordenada em ordem crescente de valor cobrado pelos atores faltantes e vfi é o valor cobrado i-ésimo ator dessa tupla.

Exemplos

Dada a entrada

```
7 5 3
5 3 1 2 3
10 2 4 5
20 5 7 6 5 4 3
15 2 6 7
30 7 1 2 3 4 5 6 7
```

Com 7 grupos, 5 atores e 3 personagens:

Estratégia	Nós árvore	Nós visitados	Tempo gasto (us)

bound aluno	14	3	798
bound professor	22	5	1238
bound aluno sem otimalidade	205	85	9703
bound professor sem otimalidade	205	85	9907
bound aluno sem viabilidade	14	3	652
bound professor sem viabilidade	23	6	1104
bound aluno sem ambos	325	325	20620
bound professor sem ambos	325	325	22248

Resposta obtida em todos os casos:

```
1 2 4
30
```

Como esperado, o corte por otimalidade foi o principal responsável pela redução de nós da árvore, uma vez que sem considerar a função limitante percorremos praticamente todos os ramos até uma certa altura da árvore; a partir da qual o corte por viabilidade passa a valer.

Note que o corte por viabilidade efetivamente limita a árvore de buscas até uma certa altura; altura a partir da qual a exploração implica num conjunto de atores escolhidos maior que o conjunto de personagens. Isso é refletido pelo total de nós sem otimização nenhuma (325) vs o total de nós com corte por viabilidade (205)

Nos casos onde existia corte por otimalidade não foi necessário aumentar a árvore até a altura limitante por viabilidade; os cortes por otimalidade foram suficientes. Exceto no caso curioso do bound professor, onde o corte por viabilidade impactou em um nó a menos gerado e percorrido (23 vs 22 e 6 vs 5). Nesse caso, a árvore de busca cresceu até a altura limite por viabilidade, e após o primeiro nodo dessa altura imediatamente cortamos os demais nodos de mesma altura e “subimos” na árvore.

Dada a entrada:

```
7 4 3
5 3 1 2 3
5 3 4 5 3
30 2 5 6
18 6 1 2 3 4 5 6
```

Com 7 grupos, 4 atores e 3 personagens:

Estratégia	Nós árvore	Nós visitados	Tempo gasto (us)
bound aluno	64	64	2801
bound professor	64	64	2409
bound aluno sem otimalidade	64	64	2971
bound professor sem otimalidade	64	64	3017
bound aluno sem viabilidade	64	64	2573
bound professor sem viabilidade	64	64	2790
bound aluno sem ambos	64	64	2784
bound professor sem ambos	64	64	2225

Saída obtida:

```
Inviavel
```

Como esse exemplo é inviável em todos os casos, toda a árvore de busca foi percorrida, e nenhuma solução foi encontrada. Os tempos medidos refletem apenas a variância das amostras

Uma diferença importante nesse trabalho está na entrada fornecida pelo professor na sessão 2.2.3 de trabalho2.pdf:

```
2 3 3
10 2
1
2
20 2
2
3
5 1
3
```

No caso do exemplo dado, a resposta optimal de acordo com o pdf engloba os atores 1 e 3, com custo final de 15.

Já no atual programa desenvolvido a saída é Inviável, uma vez que entende-se que existem 2 grupos a serem satisfeitos, mas os atores englobam um total de 3 grupos. Ou seja, não é possível saber se a saída optimal engloba apenas o ator 1, com custo 10 e os grupos 1 e 3; ou apenas o ator 2, com custo 20 e grupos 2 e 3, etc.

Referências

[1]Krehler, Donald e Stinson, Douglas. *Combinatorial Algorithms: Generation, Enumeration and Search.* 1998

[2] *set - C++ Reference* https://cplusplus.com/reference/set/set/

[3] *(Optional) C++ Sets with Custom Comparators* https://usaco.guide/silver/custom-cpp-stl?lang=cpp