Solucionando Problemas de Otimização Baseado em Uso, Custo e Influência: Minimizando Custo Utilizando Branch and Bound

Jean Carlo Sanchuki Filho

Resumo

O presente artigo tem como objetivo solucionar o problema de otimização de seleção de pessoas visando minimizar o custo, porém também preenchendo condições, utilizando o método de Branch and Bound em C.

Palavras-chave: Otimização. C. Programação. Branch and Bound.

Introdução

Muitas vezes um diretor de um filme, está em busca de atores para um papel, porém, devido a representatividade, ele precisa que esses atores façam parte de certos grupos.

O problema consiste em descobrir qual o menor custo possível de um elenco enquanto o mesmo se encaixa em todos os grupos requeridos.

Branch and Bound

O Branch and Bound é um método criado por Ailsa H. Land e Alison Grant Harcourt em 1960, o método consiste em gerar uma árvore ou um grafo de possibilidades onde cada sub-árvore é chamado de Branch (Ramo), porém, enquanto gera os Branches ele verifica um Bound (Funções limitantes, específicas para cada tipo de problema) e elimina os Branches onde se há certeza de que a sub-árvore não encontrará uma solução ótima.

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

O problema

Minimização de custo de um elenco representativo

Uma produtora de filmes deseja fazer um filme, buscando gastar o mínimo e para evitar polêmicas, deseja representar *G* grupos da sociedade.

Precisamos de *P* personagens dentre os *A* atores que juntos representam todos os *G* grupos.

Modelagem do problema

A ideia inicial para gerar os Branches com as possibilidades é fazer com que cada nível da árvore represente um Ator e caso ele seja um filho (Nó gerado a partir de outro nó) a esquerda, está sendo adicionado ao possível elenco e a direita, não está sendo considerado para o possível elenco. (Ver, Figura 1).

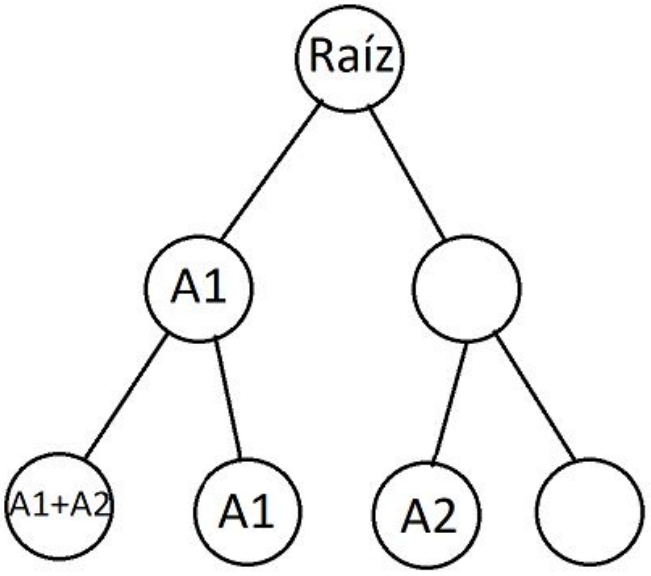

Figura 1: Exemplo de ramificação do Branch and Bound com 2 *A 's.*

Para os Bounds foi utilizado o “Lazy Bound” que retorna o custo de todos os atores utilizados até chegar ao nó atual e um Bound baseado em custo e se existe algum grupo novo no elenco.

Para percorrer as árvores, foi utilizado um vetor de Nós simulando uma pilha, onde, se um nó irá gerar filhos, seus filhos são adicionados ao início da fila, tendo

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

assim prioridade de execução e então removendo o nó que gerou os filhos da fila. Caso o Nó não gere filhos, ele será removido da fila sem adicionar nada a ela.

Para verificar se um nó deve ser percorrido ou não, verificamos se a sub-árvore de um nó possui atores suficientes para preencher o elenco, caso não, o mesmo não será percorrido, caso contrário, ele passará por outras verificações e possivelmente gerará filhos (O mesmo para as outras verificações). Outra verificação a ser feita é, caso já tenhamos uma possível solução, se o custo do Nó atual for maior que a menor solução atual, o Nó não será percorrido. Um dos Bounds possui uma condição exclusiva, onde ele verifica se os grupos já foram preenchidos, caso já tenham, é adicionado os atores com os menores custos e que ainda não foram utilizados, ao elenco atual (Vamos chamá-lo de “Group Bound”).

Conforme representado na Figura 2, temos uma árvore onde o custo de A1 < A2 e P = 2, inicia-se com a raíz sendo adicionada à pilha, gerando seus filhos e então é removida da pilha, conforme representado pelos vetores a direita, ao chegar ao Nó 3, se é gerado o melhor custo atual, por se tratar de um nó onde temos um possível resultado, o nó é removido da pilha sem adicionar filhos, o Nó 4 é removido por não possuir sub-árvore com atores suficientes para preencher o elenco, indo para o Nó 2, removido por também não possuir atores suficientes.

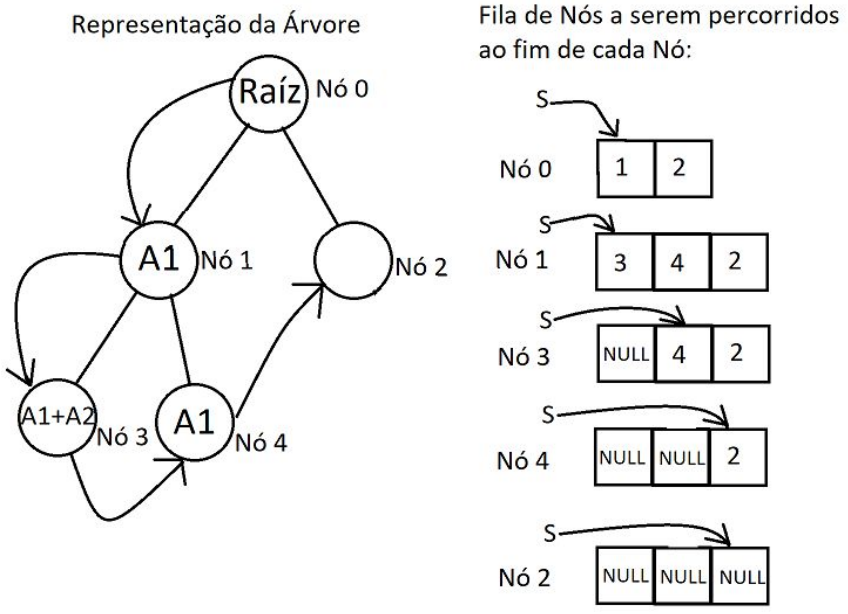

Figura 2: Relação Árvore e Pilha de Nós

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

Lazy Bound

Conforme citado acima, o Lazy Bound consiste em calcular a soma do elenco atual + o ator a ser adicionado.

Group Bound

Este Bound consiste em verificar se o Ator a ser adicionado representa algum grupo que até o momento não foi representado, caso represente alguém novo, o custo é somado como no Lazy Bound, caso contrário, o Nó não será percorrido, porém, o Group Bound possuí uma condição exclusiva durante a criação de nós, onde assim que o número de grupos for preenchido ele busca adicionar ao elenco os atores não usados que possuam o menor custo, após este processo, verifica se o custo total é menor que o menor custo atual.

Lazy Bound vs Group Bound

Utilizando os exemplos iniciais anexos ao código, verifica-se uma predominância do Lazy Bound sobre o Group Bound. Porém ao executar uma árvore de probabilidades maior, nota-se uma otimização com o Group Bound (Ver, Figura 3).

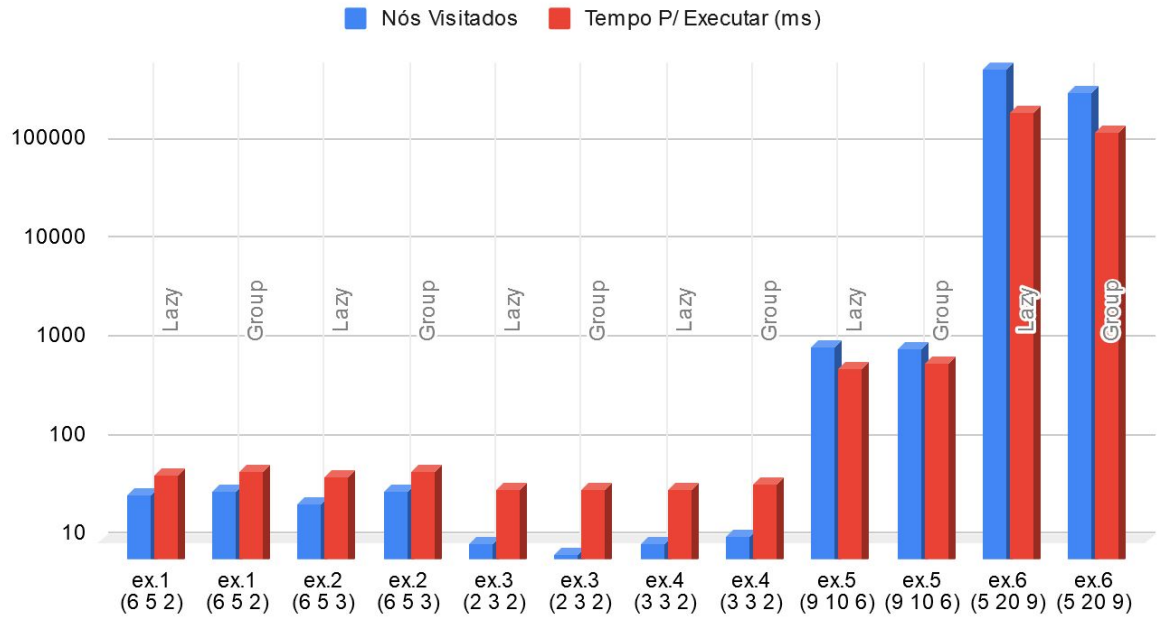

Figura 3: Comparação Lazy Bound x Group Bound para os Exemplos iniciais anexados ao Código, dados em Escala Logarítmica.

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

Visando a busca por indícios de que o Group Bound é melhor que o Lazy Bound em certos casos, foram realizados testes semelhantes ao Exemplo 6 (Ver, Figura 4).

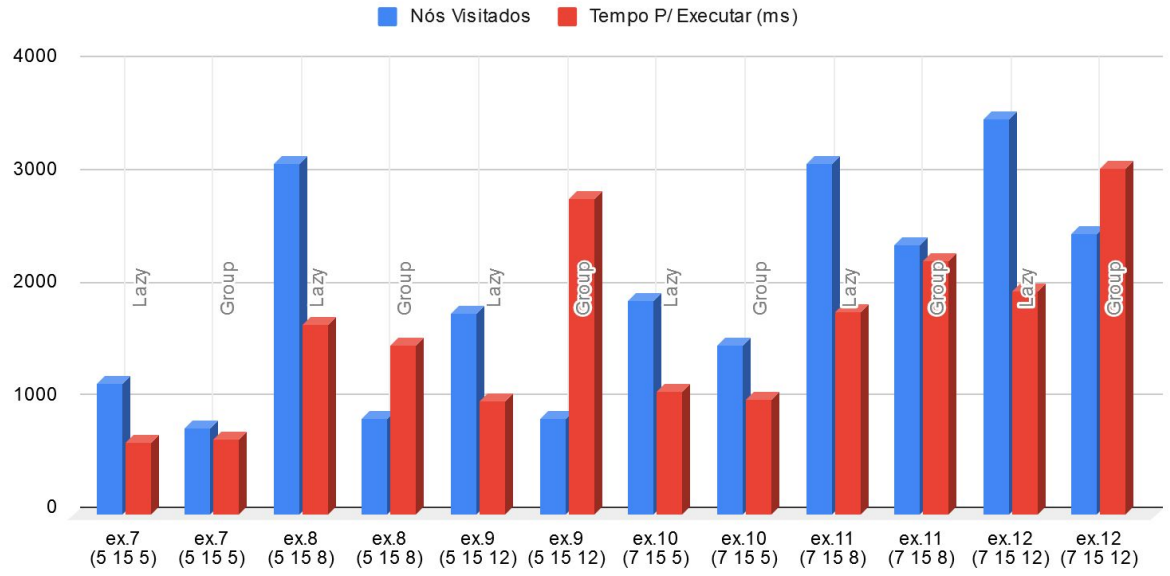

Figura 4: Comparação Lazy Bound x Group Bound para Exemplos onde acredita-se haver melhora para o Group Bound.

Devido a idéia do método ser baseado em excluir quando não há novos grupos, em casos onde se o elenco possui poucos grupos e uma quantia grande de personagens o número de Nós percorridos, é significamente menor, no entanto, devido às verificações o tempo para executar, é maior que o Lazy Bound, mesmo ele possuindo mais Nós para percorrer.

O mesmo se encaixa para casos inviáveis (onde não há uma solução), sendo assim o Group Bound possuirá uma performance muito pior que o Lazy Bound, pois os elencos nunca vão completar os grupos requisitados (Ver, Figura 5).

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

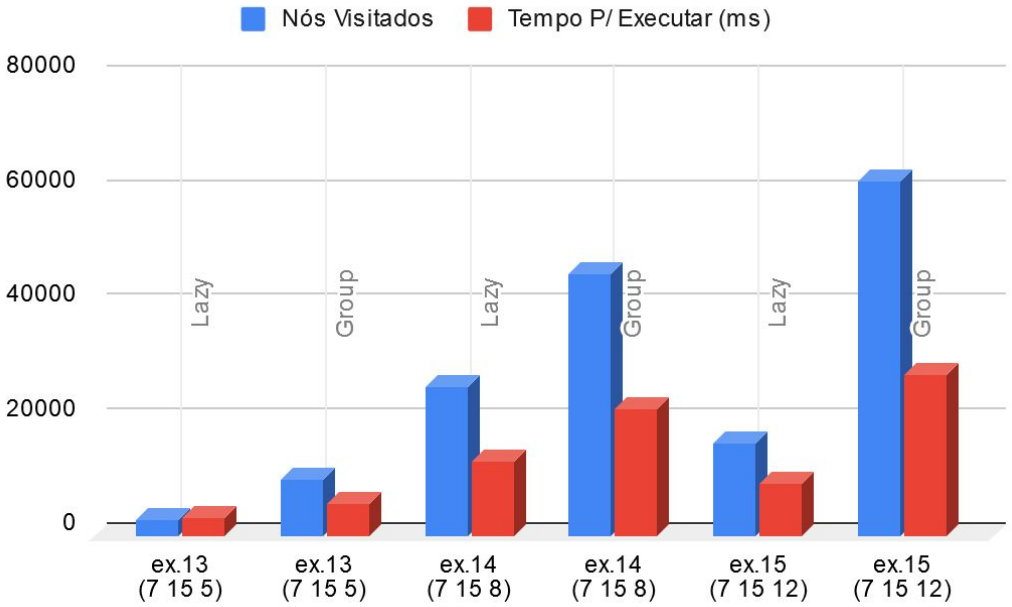

Figura 5: Comparação Lazy Bound x Group Bound para Exemplos onde não é possível encontrar um elenco viável.

Considerações Finais

Apesar dos resultados não tão promissores para o Group Bound e melhoras exclusivas para um grupo de casos, acredito que pelo fato de seu problema ser o custo/No percorrido ser maior que o do Lazy Bound, talvez seja possível realizar melhorias na implementação, visando minimizar seu custo para percorrer um Nó, ou seja, o Group Bound possui potencial para ser melhor que o Lazy Bound mesmo que somente para casos específicos, onde temos um número de grupos menor que o número de atores e personagens.

Sendo assim, apesar de ser pior em alguns casos, o Lazy Bound é melhor, porém com potencial de perder para o Group Bound.

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br

Referências

BARI, Abdul. 7 Branch and Bound Introduction. **Youtube**. 26 fev. 2018. Disponível em https://youtu.be/3RBNPc0_Q6g . Acesso em: 01 fev. 21.

BARI, Abdul. 7.1 Job Sequencing with Deadline - Branch and Bound. **Youtube**. 26 fev. 2018. Disponível em https://youtu.be/M7FI_z7_J2k . Acesso em: 01 fev. 21.

GeeksForGeeks. Branch and Bound Algorithm. 04 dez. 2018 Disponível em: https://www.geeksforgeeks.org/branch-and-bound-algorithm/ . Acesso em: 01 fev. 21.

Graduando em Ciência da Computação pela UFPR - Campus Politécnico de Curitiba. E-mail: jcsf18@inf.ufpr.br