Trabalho 2 - Otimização

Felipe Quaresma Vieira
Leonardo Marin Mendes Martin

Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil

I. Introdução

Este trabalho aborda a formação de uma comissão por um órgão governamental para tratar de um assunto específico. O objetivo é assegurar a representação de todos os grupos da sociedade relevantes para o tema em questão. Para isso, busca-se selecionar o menor número possível de candidatos, de forma que todos os grupos existentes sejam representados.

II. Modelagem

Definições

Sejam:

- S: o conjunto de todos os grupos.
- C: o conjunto de todos os candidatos.
- $S_c \subseteq S$: o subconjunto de grupos representados pelo candidato $c \in C$.

Variáveis de Decisão

Definimos uma variável de decisão binária x_c para cada candidato $c \in C$:

$$x_c = \begin{cases} 1 & \text{se o candidato } c \text{ é escolhido para a comissão} \\ 0 & \text{caso contrário} \end{cases}$$

Função Objetivo

Queremos minimizar o número de candidatos escolhidos. Portanto, a função objetivo é:

$$\min \sum_{c \in C} x_c$$

Restrições

Precisamos garantir que cada grupo $s \in S$ seja representado por pelo menos um candidato escolhido. Para cada grupo $s \in S$, temos a seguinte restrição:

$$\sum_{c \in C \text{ tal que } s \in S_c} x_c \geq 1, \quad \forall s \in S$$

Modelo Matemático Completo

O modelo completo é:

$$\begin{aligned} \text{Minimizar} \quad & \sum_{c \in C} x_c \\ \text{sujeito a} \quad & \sum_{c \in C \text{ tal que } s \in S_c} x_c \geq 1, \quad \forall s \in S \\ & x_c \in \{0, 1\}, \quad \forall c \in C \end{aligned}$$

Diante disso, temos os seguintes cenários de execução:

- **Sem cortes de Otimalidade e Viabilidade:**
  Para esse tópico, todas as combinações de candidatos devem ser testadas afim de obter o menor conjunto de candidatos possível. Ou seja, o backtracking funciona sem nenhum corte e todos os nodos da árvore serão contados.
- **Com cortes de Otimalidade e sem cortes por Viabilidade:** Nesse caso, introduziremos nossa função de Bound, que funciona da seguinte forma:

$$\mathrm{B}(E, F) = \begin{cases} |E| & \text{se Restantes}(E) = \emptyset \\ |E| + \left\lceil \frac{|\text{Restantes}(E)|}{\text{MaxCobertura}(F)} \right\rceil & \text{caso contrário} \end{cases}$$

  Assim, caso o Bound perceba que a solução eminente será maior que a solução ótima atual, então o nodo será cortado e não seguiremos mais com aquele "caminho"da árvore.
- **Sem cortes de Otimalidade e com cortes por Viabilidade:** Primeiramente, com a função "isInfeasible", verificamos se a quantidade de membros do conjunto resultante da união de todos os candidatos é igual a quantidade de grupos recebida como entrada. Caso não seja, o programa é interrompido.
  Depois disso, já na função de backtracking, apresentaremos nossa função de Cl. Na função de corte por viabilidade, verificamos se o conjunto de grupos restantes que faltam ser representados está contido no conjunto resultante da união dos grupos dos candidatos restantes. Dessa forma, saberemos se os candidatos restantes são ou não capazes de formular uma resposta válida. Caso não sejam, todo o caminho de nó que seria continuado a partir dali é descartado.
- **Com cortes de Otimalidade e com cortes por Viabilidade:** Nessa etapa, ambas funções (Cl e Bound), estão ativas. Ou seja, além de verificarmos se os candidatos restantes são capazes de fornecer uma resposta válida, também verificamos se percorrer aquele caminho da árvore resultará numa solução maior ou igual à uma ótima já calculada. Com isso, temos a menor quantidade de nodos percorridos possível em um curto período de tempo.

- **Observações:** É importante salientar que ordenamos de forma decrescente os candidatos de acordo com a quantidade de grupos que eles representam. Por isso, em algumas soluções, poderá haver casos em que os candidatos 1 2 3 sejam uma solução ótima, mas a resposta no terminal será 1 2 4. Isso ocorre porque o grupo número 4 cobre um conjunto de grupos maior e, portanto, é escolhido antes. Além disso, quando a entrada é inviável e o programa é compilado com -f, a resposta ótima será 1 até N, sendo N o número de candidatos obtidos na entrada. Isso acontece porque inicializamos o conjunto com todos os candidatos de forma a ter cardinalidade máxima.

III. Detalhes e Implementação

- **Estruturas:**
  - $Candidate$: Estrutura que armazena cada candidato. Todos os membros possuem índice, número de grupos e um conjunto com os grupos que representa.
  - $Result$: Guarda a solução atual e a solução ótima.
  - $Remaining$: Guarda um conjunto com os candidatos restantes a serem escolhidos e os grupos restantes a serem preenchidos.
  - $Options$: Armazena as opções de execução escolhidas.
- **Conjuntos:**
  - $RemainingGroups$: Conjunto dos grupos restantes a serem representados.
  - $Candidates$: Conjunto dos candidatos a serem escolhidos.
  - $Solution$: Conjunto que guarda a solução atual.
  - $DefinitveSolution$: Conjunto que guarda a solução ótima.
- **Função de Bound:**
  Seja S o conjunto de todos os grupos:

$$S = \{1, 2, \ldots, n\}$$

  Seja C o conjunto de todos os candidatos:

$$C = \{c_1, c_2, \ldots, c_m\}$$

  Cada candidato c_i está associado a um subconjunto de grupos $S_{c_i} \subseteq S$.
  Seja $E \subseteq C$ o conjunto de candidatos já escolhidos e $F = C \setminus E$ o conjunto de candidatos ainda disponíveis.
  A união dos grupos cobertos pelos candidatos em E é dada por:

$$\text{Cobertos}(E) = \bigcup_{c \in E} S_c$$

  Os grupos restantes não cobertos pelos candidatos em E são:

$$\text{Restantes}(E) = S \setminus \text{Cobertos}(E)$$

  A máxima cobertura por qualquer candidato em F é:

$$\text{MaxCobertura}(F) = \max_{c \in F} |S_c|$$

  O número de grupos restantes é:

$$\text{NumGruposRestantes} = |\text{Restantes}(E)|$$

  O número mínimo de candidatos necessários para cobrir todos os grupos restantes é:

$$\text{MinCandidatosNecessarios} = \left\lceil \frac{\text{NumGruposRestantes}}{\text{MaxCobertura}(F)} \right\rceil$$

  A função de bound `B` é definida como:

$$\text{B}(E, F) = \begin{cases} |E| & \text{se Restantes}(E) = \emptyset \\ |E| + \left\lceil \frac{|\text{Restantes}(E)|}{\text{MaxCobertura}(F)} \right\rceil & \text{caso contrário} \end{cases}$$

- **Função de Cl:** A função de Cl segue os seguintes passos:
  - **Retorno e Resultado:** Caso a função de ClCalcule retorne falso, o código retorna e o corte é realizado. Entretanto, caso retorne verdadeiro, o código continua.
  - **Conjunto União:** Cria um conjunto Cl_definitivo que é resultado da união dos grupos de todos os candidatos restantes. Ou seja, todos aqueles que estão em índices maiores que a variável de condição do for da função de backTracking.
  - **Contido:** Verificamos se o conjunto de grupos que ainda não foram representados está contido no conjunto da união citado anteriormente. Se sim, a função retorna verdadeiro. Caso contrário, retorna falso.

IV. Análise das Funções Limitantes

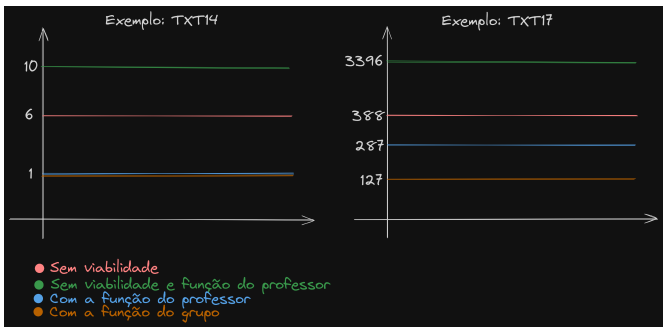
A função de bound `B` feita pelos alunos é definida como:

$$\text{B}(E, F) = \begin{cases} |E| & \text{se Restantes}(E) = \emptyset \\ |E| + \left\lceil \frac{|\text{Restantes}(E)|}{\text{MaxCobertura}(F)} \right\rceil & \text{caso contrário} \end{cases}$$

A função de bound $B(E, F)$ fornece uma estimativa mínima do número de candidatos necessários para cobrir todos os grupos a serem representados pela comissão. Se todos os grupos já foram cobertos pelos candidatos em E (isto é, $\text{Restantes}(E) = \emptyset$), o valor da função é simplesmente $|E|$, que é o número de candidatos já selecionados. Caso contrário, quando ainda existem grupos a serem cobertos, a função calcula o valor adicional necessário como $\left\lceil \frac{|\text{Restantes}(E)|}{\text{MaxCobertura}(F)} \right\rceil$, onde $|\text{Restantes}(E)|$ é o número de grupos que ainda precisam ser cobertos e $\text{MaxCobertura}(F)$ é o número máximo de grupos que um único candidato pode cobrir. Assim, $B(E, F)$ combina o número atual de candidatos com uma estimativa do número adicional necessário, fornecendo um limite superior que é sempre maior ou igual ao número mínimo de candidatos necessários para garantir a cobertura completa dos grupos. Essa função é eficiente em termos de cálculo e útil para avaliar soluções parciais, ajustando a busca por soluções ótimas ao fornecer uma estimativa prática e não subestimada do valor ótimo. Desta forma, a função oferece uma poda mais eficiente, adequada para instâncias maiores e mais complexas.

Já a função do professor, determina se uma solução parcial pode levar a uma solução melhor do que a melhor solução atual realizando uma poda bastante simples, baseada apenas na contagem de candidatos e uma checagem mínima. Nesse caso, podendo não ser eficaz para instâncias maiores onde uma análise mais detalhada dos grupos e candidatos restantes é necessária.

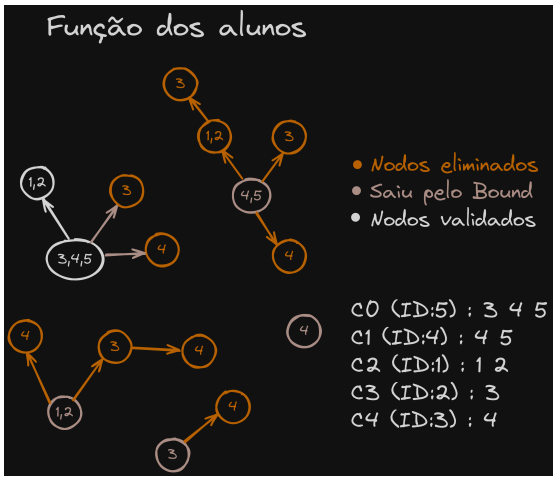
Com isso, podemos analisar o desempenho das duas funções utilizando dois exemplos de entrada:

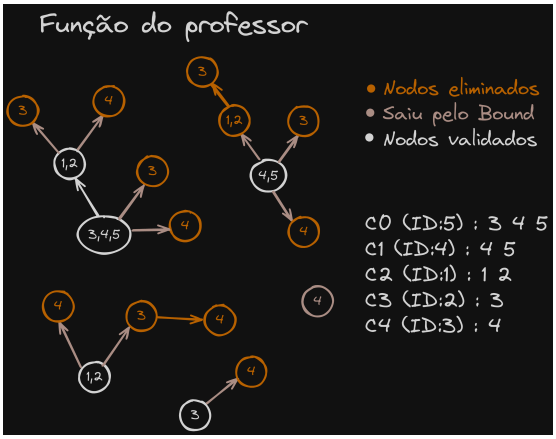


No gráfico, utilizamos os exemplos 14, com poucas entradas, e 17, que possui a maior entrada, para verificarmos o desempenho das funções limitantes do professor e do grupo.

Como podemos ver, a função do professor utiliza mais vezes o bound, já que apenas verifica o nodo sucessor. Assim, como a nossa função analisa mais profundamente a árvore, menos bounds são necessários. Já que cortamos o caminho em seu ”início”.

As imagens a seguir representam o funcionamento das duas funções de forma visual no exemplo 14. Sabendo que a mesma lógica mencionada no parágrafo anterior segue para o exemplo 17 e que há uma drástica diferença do uso das funções, podemos concluir que a função desenvolvida pelos alunos é mais eficaz que a estabelecida pelo professor.





V. Testes e Exemplos

Foi feito um script chamado *gera_relatorio.sh* para automatizar a execução do programa *comissao* com todas as combinações possíveis das opções *-a*, *-o* e *-f*, além da execução sem opções. Este script registra apenas as mensagens de erro (stderr) e coloca em um arquivo chamado *relatorio.txt*.

- **Uso:**
  Para utilizar o script, execute-o passando o caminho do arquivo de exemplo como argumento:

```
./gera_relatorio.sh exemplos/txt17.txt
```

  O script irá gerar um arquivo chamado *relatorio.txt* contendo as mensagens de erro das diversas execuções do programa *comissao*.

- **Resultados de testes**
  Temos um subdiretório chamado `exemplos` com 18 possíveis entradas para o trabalho. Nessa seção, mostraremos o resultado de um deles.

Exemplo 17

Opções	N	Tempo
-o -f	29991	0,535302
-o	1771	0,059465
-a -f	1036	0,011722
-a	159	0,014328
-f	40	0,000318
-	24	0,003477

Tabela I: Tabela de comparação de opções com número de nós (N) e tempo para o exemplo 17.

VI. Conclusão

Os dados apresentados demonstram que o algoritmo Branch and Bound foi extremamente eficaz na resolução do problema, sendo muito mais eficiente em termos de número de nós e tempo, conforme mostrado nas tabelas.

Além disso, os resultados indicam que a função desenvolvida pelos alunos superou a função do professor, o que é evidenciado pelo menor número de nós percorridos e uso do bound.

É importante notar que, devido à complexidade da função Cl, houve um aumento no tempo de execução. Contudo, essa função se destacou positivamente na redução do número de nós.

Concluímos, portanto, que o algoritmo de Branch and Bound para o problema da comissão representativa obteve sucesso e a função dos alunos demonstrou uma eficiência notável.