Segundo Trabalho Prático (CI1238)

Frank Wolff Hannemann

29 de julho de 2024

1 O Problema

Um órgão do governo quer montar uma comissão para tratar de um certo assunto. Mas este órgão, preocupado com a representatividade da comissão, quer que todos os grupos (previamente elencados) da sociedade sejam representados. Uma pessoa pode fazer parte de mais de um destes grupos, portanto podemos ter menos representantes que grupos.

Dados um conjunto S de grupos, um conjunto C de candidatos, e um subconjunto $S_c \subseteq S$, para cada candidato c, indicando os grupos dos quais c faz parte, devemos encontrar um conjunto $X \subseteq C$ tal que $\cup_{c \in X} S_c = S$ e $|X|$ seja mínimo.

1.1 Formato de entrada e saída

Os formatos de entrada e saída são descritos a seguir e devem ser usados a entrada e a saída padrão (stdin e stdout).

Entrada: A entrada é formada por um conjunto de números inteiros. Os números podem estar separados por um ou mais espaços, tabs ou fim de linha. A entrada inicia com os valores de $\ell = |S|$ e $n = |C|$ na primeira linha (separados por espaço). Considere que $S = \{1, \ldots, \ell\}$ e $C = \{1, \ldots, n\}$. Em seguida, temos n blocos (um para cada candidato). Um bloco de candidato começa com um valor, s, indicando o número de grupos da sociedade dos quais ele faz parte. Em seguida, temos s números, que são os índices dos grupos.

Saída: Os números do conjunto X, em uma mesma linha, separados por espaço (simples) e sem espaço no começo nem no fim da linha, ordenados em ordem crescente. Caso a instância não tenha solução viável, informe isso escrevendo "Inviavel" (sem acento) na saída."[1]

2 Modelagem da função limitante

A proposta para função limitante foi a seguinte:

$$\text{Bound}(E, F) = |E| + \begin{cases} 0, & \text{se } \ell = |\cup_{c \in E} S_c|\,, \\ 1, & \text{se } S_c \geq |S - \cup_{c \in E} S_c|\,, \\ 2, & \text{se } S_c < |S - \cup_{c \in E} S_c|\,. \end{cases}$$

Ou seja, se o número de grupos existentes for igual ao de grupos escolhidos, não é necessário escolher mais nenhum candidato, logo $Bound(E, F) = |E|$.

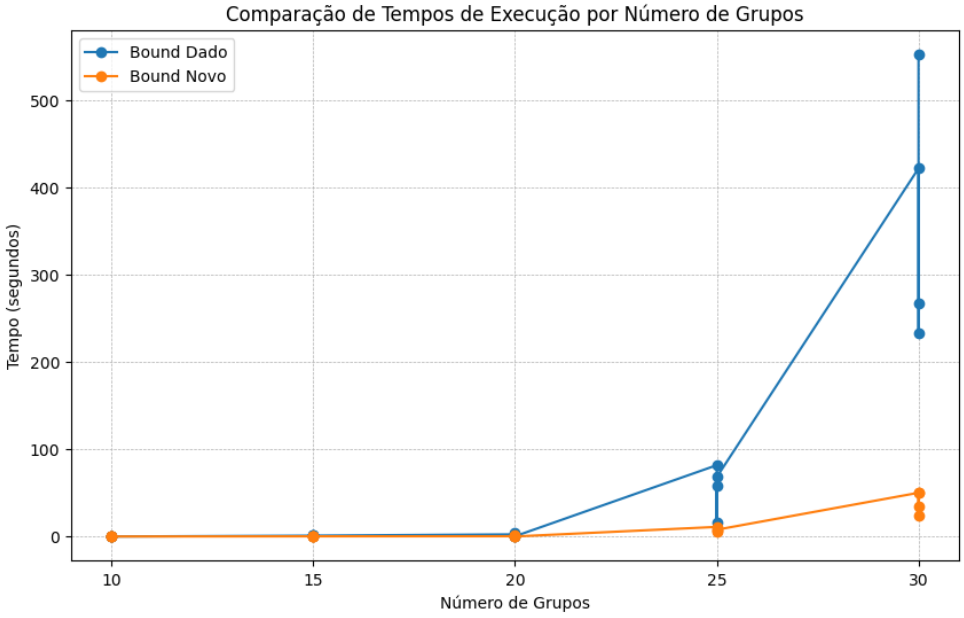
No entanto, se o número de grupos do próximo candidato for maior ou igual ao número de grupos faltantes, $Bound(E, F) = |E| + 1$, de modo que pelo menos um candidato deve ser adicionado para compreender todos os grupos.

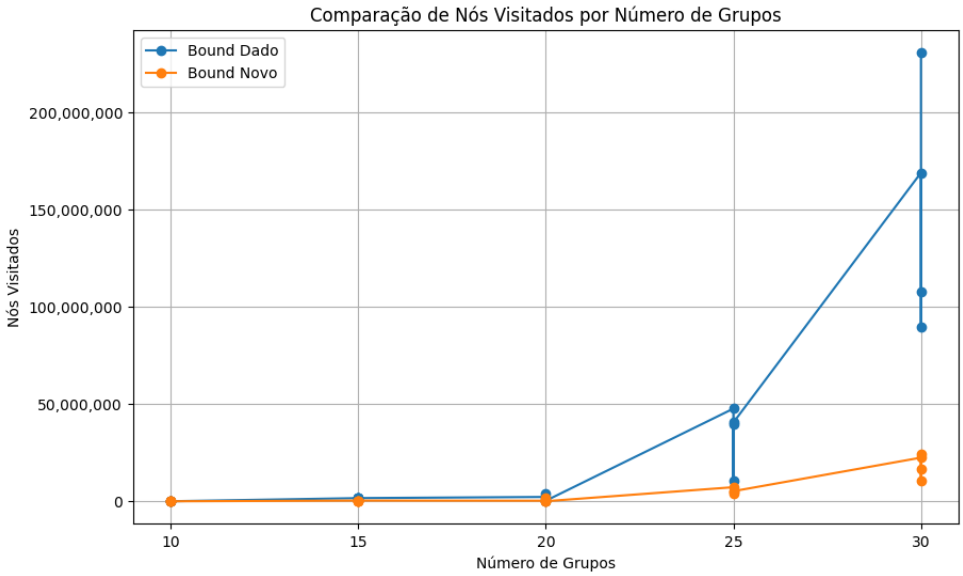
E, finalmente, se o número de grupos do próximo candidato for menor do que aqueles que ainda não foram incluídos, pelo menos dois candidatos devem ser escolhidos para satisfazer todos os grupos, e $Bound(E, F) = |E| + 2$.

Desta forma, é esperado que esta função seja mais eficaz ao identificar ramos para poda.

3 Análise das funções limitantes

Temos como resultado das análises de performance das funções limitantes que o Bound Novo se saiu muito melhor, conseguindo podar muitos ramos da árvore de execução.





Qtd Grupos - Teste	Bound Dado		Bound Novo	
	Tempo (s)	Nós Visitados	Tempo (s)	Nós Visitados
10 - 1	0.014410	50,807	0.003784	14,359
10 - 2	0.015248	51,887	0.005747	19,243
10 - 3	0.014509	43,173	0.003280	10,939
10 - 4	0.003301	12,561	0.000735	2,739
15 - 1	0.961038	1,628,179	0.198241	363,987
15 - 2	0.933024	1,555,031	0.207197	371,619
15 - 3	1.080198	1,751,983	0.207140	363,971
15 - 4	1.020485	1,638,199	0.206239	367,869
20 - 1	2.454547	2,228,025	0.300567	301,413
20 - 2	3.947737	3,944,041	1.583257	1,721,049
20 - 3	0.427440	408,947	0.213956	223,691
20 - 4	0.261322	270,331	0.024696	29,087
25 - 1	81.741837	47,601,869	10.996515	7,235,497
25 - 2	58.444766	39,628,611	6.537080	5,048,951
25 - 3	15.678501	10,683,017	5.283419	3,749,127
25 - 4	68.050969	40,466,851	7.762537	5,194,163
30 - 1	422.047397	168,771,629	50.134976	22,422,717
30 - 2	233.667702	89,469,249	23.651158	10,475,049
30 - 3	267.119463	107,403,883	35.125611	16,479,345
30 - 4	552.533453	230,442,425	50.520171	24,492,579

Tabela 1: Comparação de Nós Visitados e Tempos de Execução para Bound Dada e Bound Nova em Diferentes Testes

4 Implementação

Para implementar as funções de Branch & Bound, os exemplos do livro Combinatorial Algorithms [2] foram utilizados.

A estrutura de dados utilizada foi *Candidato* e *CandidatoList*. Candidato armazena o ID do candidato, o número de grupos a que ele pertence, e um vetor que contém os grupos a que ele pertence. A estrutura *CandidatoList* possui um vetor de candidatos, o número de candidatos que estão na lista, e a capacidade, que representa o número total de grupos que o candidato pode possuir, determinando o tamanho da lista de candidatos, podendo ser aumentada se necessário.

Todas as funções relacionadas às operações de alocação, criação, impressão e liberação de memória de *Candidato* e *CandidatoList* estão no arquivo "candidatos.c".

Na *main*, está a função para verificação dos grupos já cobertos pelos candidatos escolhidos. Também estão definidas as funções de implementação das *bounds* e a que encontra o menor conjunto de candidatos. Para alterar os parâmetros do programa, foi utilizada a estrutura de dados booleana *Flags*.

Referências

1 GUEDES, A. **Segundo Trabalho Prático**. [s.n.], 2024. Disponível em: <https://www.inf.ufpr.br/andre/Disciplinas/CI1238-2024-1/trabalho2.pdf>.

2 KREHER, D.; STINSON, D. **Combinatorial Algorithms - Generation, Enumeration and Search**. [S.l.]: CRC Press, 1999.