O problema da cobertura única

Gabriel Saraiva de Gouveia[1], Olga Maria dos Santos Gonçalves[1], Rafael Tavares Oliveira[1] e Yasser Farid Pereira[1].

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM-PR)

Maringá – PR – Brasil

{ra129145, ra130002, ra129969, ra129706}@uem.br

Resumo. *Este artigo apresenta uma formulação em programação linear inteira para o problema de cobertura única, uma variação do clássico problema de cobertura máxima, com experimentos empíricos para avaliar o desempenho do modelo e a qualidade das soluções fornecidas pelo solver Coin-or Branch and Cut (CBC).*

1. Fundamentação

Este trabalho apresenta a aplicação de um algoritmo com base em um modelo de programação linear inteiro para o problema da cobertura única, este problema é uma variante do problema de cobertura de conjuntos. A cobertura única, conhecida como UCP (Unique Covering Problem), tem como objetivo cobrir o maior número de elementos uma única vez, podendo ter mais de um subconjunto, mas minimizando a quantidade de subconjuntos que cobrem o elemento e podendo assim, deixar elementos fora de cobertura.

O UCP foi apresentado por Demaine et al. (2006) e tem aplicações em telefonia, redes sem fio e rádios broadcast. Ele consiste em um universo U com n elementos e S um conjunto de m subconjuntos de elementos de U, cujo o objetivo é encontrar um subconjunto S′, onde $S' \subseteq S$, que maximize a quantidade de elementos cobertos por um único subconjunto.

O tema, a modelagem e as instâncias foram fundamentados no artigo proposto por Mazaro (2011). O método aplicado utiliza o algoritmo branch-and-cut para resolver o UCP. Diversos experimentos foram conduzidos para avaliar o método proposto a fim de identificar o gap de otimalidade e o tempo gasto para resolver diferentes instâncias testes do problema.

A seguir descreveremos a modelagem de PLI e o desenvolvimento do programa com o resolvedor utilizado. Posteriomente, apresentaremos detalhes das instâncias usadas no experimento, bem como, a análise dos resultados e do tempo de execução.

2. Modelo de Programação Linear Inteira

Função Objetivo:

$$\max \sum_{i=1}^{n} y_i$$

Maximizar Σ yi, onde yi = 1 se o elemento i for coberto exatamente uma vez.

Restrições:

$$\begin{aligned} \text{sujeito a} \quad & \sum_{j=1}^{m} a_{ij} x_j = z_i && , \forall i, 1 \leq i \leq n \\ & y_i \leq z_i && , \forall i, 1 \leq i \leq n \\ & (M-1)y_i + z_i \leq M && , \forall i, 1 \leq i \leq n \\ & y \in \{0,1\}^n, z \in \mathbb{Z}^n, x \in \{0,1\}^m. \end{aligned}$$

Onde:

- n: cardinalidade do universo U
- m: cardinalidade do conjunto S
- aij: variável que representa a matriz de incidência
- xj: variável que indica se o subconjunto j foi escolhido.
- zi: número total de vezes que o elemento i foi coberto.
- yi: variável binária que vale 1 se o elemento i foi coberto uma única vez.
- M: constante grande (por exemplo, o número total de subconjuntos).

Considere o exemplo, no conjunto da esquerda, nossa instância inicial, tem o universo U = {1,2,3,4,5,6,7,8,9,10} com n = 10 e um conjunto S = {A, B, C} com m = 3, a solução ótima encontrada pelo método proposto é a cobertura dos subconjuntos {A, C}, que cobrem 9 elementos de 10, com somente uma cobertura não única (elemento 2). Essa é a essência do Problema da Cobertura Única (UCP): não basta cobrir os elementos, é buscado o máximo de coberturas únicas, aceitando não cobrir alguns os elementos ou ter o mínimo de coberturas não únicas.

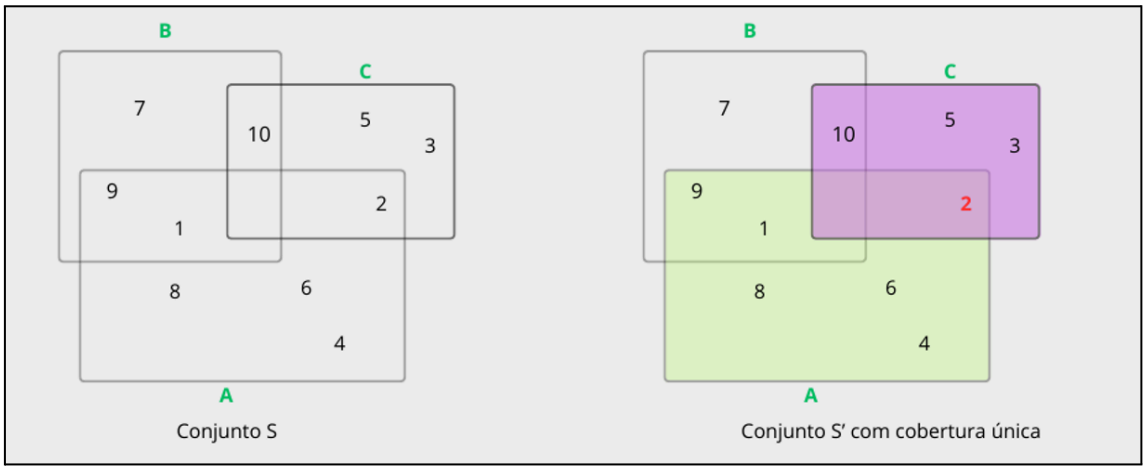

Figura 1. Exemplo de UCP

3. Metodologia

O desenvolvimento do algoritmo foi realizado com foco na modelagem das restrições e na leitura de instâncias. Foi implementado com a linguagem Python 3.11 e a biblioteca OR-Tools da Google. O solver escolhido foi o Coin-or Branch and Cut (CBC) na versão 2.10.12, o CBC é voltado para a resolução de problemas de Programação Inteira Mista. Ele é baseado no *Branch-and-Cut*, que estende o método clássico de *Branch-and-Bound* com a adição de cortes para acelerar a convergência à solução ótima inteira.

O método consiste em:

1. **Carregamento da instância:** leitura e carregamento dos dados em memória de todas as instâncias em uma pasta.
2. **Construção do modelo:** definição das variáveis de decisão, da função objetivo e restrições.
3. **Processamento:** aplicação do algoritmo de branch-and-cut fornecido pelo solver CBC nas instâncias para encontrar uma solução ótima.
4. **Resolução:** registrado o log de execução de cada instância pelo solver CBC em tempo de execução e opcionalmente registrado resultados relevantes para análises (tempo de execução, gap, coberturas únicas…)

Algoritmo: UniqueCoverSolver (instancias)

1. **Início**
2. **para cada** `arquivo` **em** `Instâncias` **faça:**
3. (*Linha = L, Coluna = C, Matriz = M*) ← Ler_Instancia(*arquivo*)
4. *Solver* ← Inicializa_Solver_CBC()
5. Cria_Variaveis(*L, C, Solver*)
6. Define_FuncaoObjetivo_Maximizar_Cobertura_Unica(*Solver*)
7. Define_Restricoes_Cobertura(*M, Solver*)
8. *Solucao* ← Executa_Solver_CBC(*tempo_maximo_de_execucao*)
9. Escreve_resultados(*Solucao, tempo, Gap*)
10. **Fim**

A execução do algoritmo foi através de um computador pessoal, disposto das seguintes especificações: sistema operacional Windows 11 Home Single Language, versão 24H2, com processador 13th Gen Intel(R) Core(TM) i5-13450HX 2.40 GHz e 16GB de memória RAM.

Foram utilizadas 8 instâncias do conjunto Set Covering Problem de tamanhos variados, obtidas na *OR-Library* de Beasley (2010). Todas seguem o formato: dois primeiros números que indicam o número de linhas, correspondentes aos elementos a serem cobertos; e um número de colunas, correspondentes aos subconjuntos disponíveis, logo após, descreve os custos associados a cada subconjunto (de 1 a 100). E então inicia o conteúdo com a quantidade de elementos do subconjunto seguido pelos elementos separados por espaço.

```
10 3
1 1 1 1 1 1 1 1 1 1
6
1 2 4 6 8 9
4
1 7 9 10
4
2 3 5 10
```

Figura 2. Formato das instâncias txt

Computacionalmente é lido em forma de matriz, onde as linhas são os elementos e as colunas os subconjuntos, caso o elemento seja coberto pelo subconjunto, na interseção do Subconjunto X Elemento é adicionado 1, caso contrário é 0.

5. Resultados

O resultado dos experimentos com as 8 instâncias foi limitado a 3600 segundos de execução, o objetivo é avaliar o desempenho do método proposto em encontrar soluções ótimas em um tempo de execução viável. Para compreendermos a qualidade da solução, foi calculado o GAP de integralidade em %, o GAP é usado para indicar o quão próximo a solução encontrada está da solução ótima conhecida ou de um limite.

Na Tabela 1 é exposto o resultados experimentais, mostrando as instâncias da OR Library usadas e sua quantidade de elementos (n) e quantidade de conjuntos (m), seguido pelo tempo de execução do solver até encontrar a solução ótima ou até o limite de tempo, depois temos o Gap e o Valor da solução factível encontrada e a média de cobertura dos elementos (C).

Tabela 1 - Resultados experimentais

Instâncias	n	m	Tempo	Gap	Valor obtido	C
scp49.txt	200	1000	0h 25min 34s 990ms	0.00%	200	1
scp57.txt	200	2000	0h 0min 4s 570ms	0.00%	200	1
scp63.txt	200	1000	1h 0min 0s 80ms	28.00%	144	1.36
scpa5.txt	300	3000	1h 0min 0s 60ms	14.00%	258	1.21
scpb3.txt	300	3000	1h 0min 0s 240ms	79.00%	64	2.44
scpc3.txt	400	4000	1h 0min 0s 60ms	35.00%	261	1.13
scpd1.txt	400	4000	1h 0min 1s 610ms	90.00%	41	2.87
scpe4.txt	50	500	1h 0min 0s 170ms	12.00%	44	1.2

Nota-se que o método resolveu as instâncias scp49 e scp57 de forma ótima já que o gap é 0%, nas demais não obtive o mesmo desempenho, seja pelo tamanho das instâncias ser maior e o tempo de limitação não ser adequado ou o limitante superior atual não ser bom. Se compararmos os resultados ao do artigo Mazaro (2011) que tem a

mesma limitação de tempo de 3600 segundos, as três primeiras instâncias têm gap iguais já as outras possuem gap maior do que as relatadas do artigo de Mazaro, mas vale ressaltar que os resultados do artigo de Mazaro são referentes a grupos de instâncias e o deste experimento é de uma única instância do grupo.

Observando a Tabela 1, não podemos tomar unicamente a quantidade de elementos e conjuntos como medida para a dificuldade de encontrar a solução ótima, pois a instância scpe4 tem o menor número de elementos e conjuntos, mas ainda não teve o mesmo desempenho que a scp49 e scp57 que são maiores. Já a cobertura média dos conjuntos nos diz muito sobre a dificuldade e o tempo para encontrar uma solução ótima, podemos notar que quanto menor a média de cobertura dos elementos mais próximo da solução ótima (menor o GAP).

4. Considerações

Section titles must be in boldface, 13pt, flush left. There should be an extra 12 pt of space before each title. Section numbering is optional. The first paragraph of each section should not be indented, while the first lines of subsequent paragraphs should be indented by 1.27 cm.

Referências

Beasley, J. E., 2010. Or-library. http://people.brunel.ac.uk/˜mastjjb/jeb/orlib/scpinfo.html.

MAZARO, Regina; HOSHINO, Ayako, (2011, agosto). O problema de cobertura única. Anais do XLIII Simpósio Brasileiro de Pesquisa Operacional, Ubatuba, São Paulo. Recuperado de http://www.din.uem.br/sbpo/sbpo2011/pdf/88121.pdf

GOOGLE OR-Tools, 2025. Guia da ferramenta para Python.

COIN-OR. *COIN-OR Branch-and-Cut MIP Solver (Cbc)*. https://github.com/coin-or/Cbc.