Universidad Autónoma de Nuevo León
Facultad de Ciencias Físico Matemáticas
Operations Research | Group code: 032
Set Covering Problem
May 15th, 2025

Alan Alejandro Vargas-González
alejandro.vargasg@uanl.edu.mx
`Universidad Autónoma de Nuevo León`
San Nicolás de los Garza, Nuevo León, MX

Resumen

This project presents a computational approach to the **Set Covering Problem (SCP)** through binary integer linear programming using Python and its library, `PuLP`. A generic model was implemented, capable of reading classic and real-world instances from the OR-Library. Tests were conducted on small, medium, and large-scale instances, generating reproducible results and visualizations to evaluate the model's performance and scalability. It was concluded that exact solutions using `CBC` (through `PuLP`)are efficient in many cases, although structural limitations were identified for highly sparse or dense instances.

Key-words: Set covering, integer optimization, Or-Library, Python, PuLP, CBC, result visualization

Introduction

The Set Covering Problem (SCP) is a fundamental combinatorial optimization problem in the field of Operations Research. It involves selecting the smallest number of subsets whose union covers a universal set of elements. Given its NP-hard complexity, SCP serves as a standard for testing exact algorithms and heuristics. Its real-world applications include emergency facility placement, telecommunication tower positioning and resource distribution.

In this project, I focus on building a complete pipeline to model, solve, analyze, and visualize solutions to SCP instances. My approach supports binary integer linear programming (BILP) as it matches the nature and structure of SCP: each subset is either included (1) or not (0), and constraints enforce the full coverage of the universal set.

Methodology

Why Binary Integer Programming?

Binary decision variables are a natural fit for modeling the inclusion/exclusion of subsets. Linear constraints allow us to enforce that each element in the universe must be covered by at least one selected subset. Additionally, costs associated with each subset enable a single-objective minimization problem, perfectly aligning with linear programming. [Balas & Padberg, 1972]

Computational Complexity

The Set Covering Problem is **NP-hard**, as demonstrated by:

1. Polynomial-time reduction from Vertex Cover (What's the smallest number of dots you can pick so that every line touches at least one of the dots you picked?)[Karp, 1972].

2. Exponential solution space (2^n possible subset combinations).
3. No known polynomial-time exact algorithm (unless P = NP).

Mathematical Formulation

Parameters:

- m: number of elements.
- n: number of subsets.
- c_j: Cost of selecting subset j.
- a_{ij}: binary matrix that indicates if the subset j has the element i.

Decision variables:

$$x_j = \begin{cases} 1 & \text{if the } j \text{ subset is selected} \\ 0 & \text{it isn't selected} \end{cases}$$

Objective function:

$$\text{mín} \sum_{j=1}^{n} c_j x_j$$

Constraints:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i = 1, \ldots, m$$

Implementation Strategy

Python was chosen because of its versatility and ecosystem. The `PuLP` library provides an intuitive interface for defining variables, constraints, and objectives. The code design is modular:

- `read()` parses the OR-Library format.
- `validation()` ensures input correctness.
- `solving()` builds and solves the LP using CBC.
- `results.py` automates multiple test executions.
- `graphics.py` generates visual summaries using `matplotlib`.

Data Source

The instances from the OR-Library were the obvious choice, including sets 4–6, A–E [Beasley, 1987b], and real-world railway datasets (e.g., `rail2536`). Each file defines costs, rows (elements), and the subsets covering them.

Results

Result Visualization

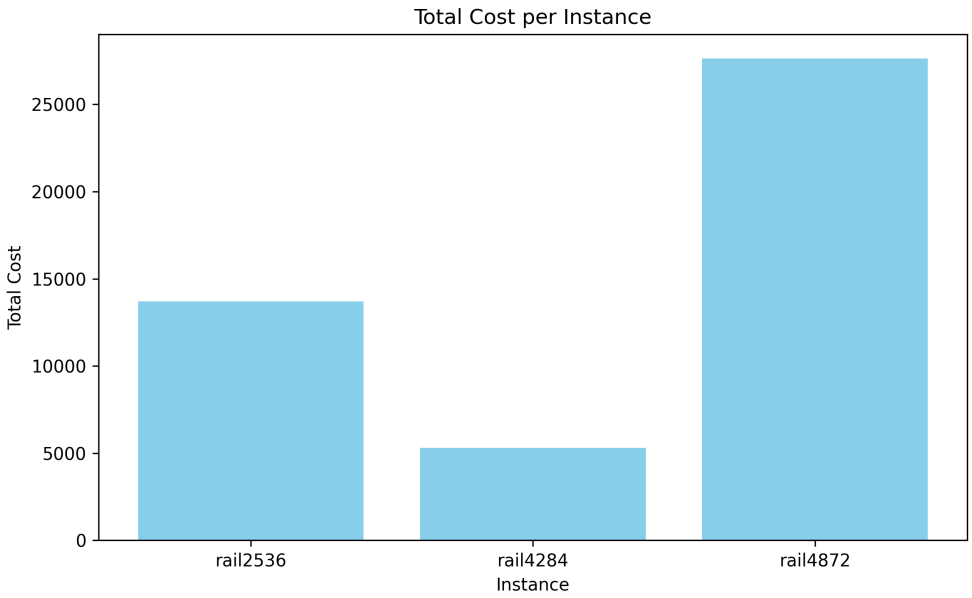

Figura 1: Total cost per instance (railway dataset)

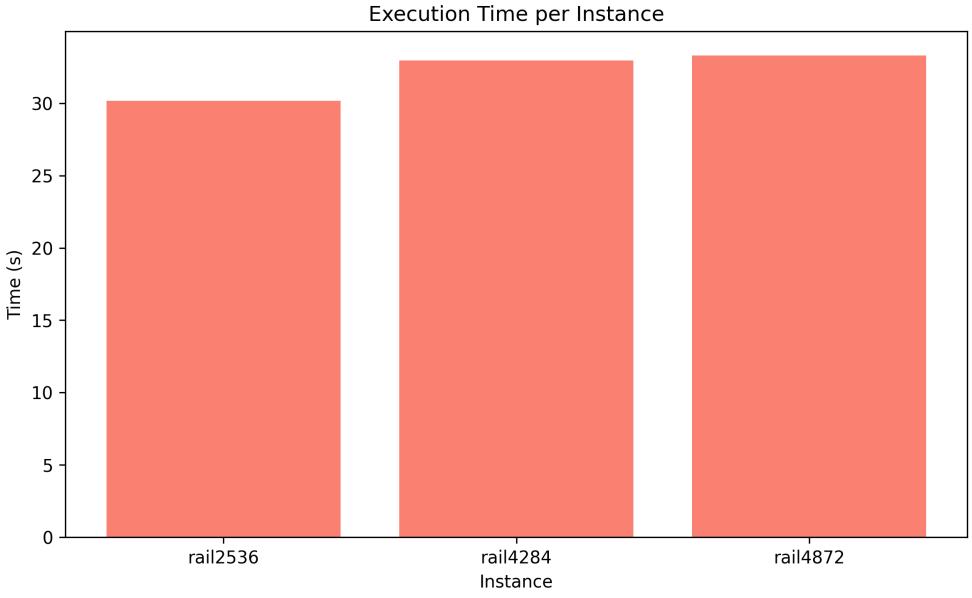

Figura 2: Execution time per instance (railway dataset)

Small and Medium Instances

- **Correctness**: The solver consistently matched known optima (e.g., scp41.txt and scpbl.txt) and satisfied all constraints.

- **Efficiency**: Execution times showed sublinear growth with problem size. The instance scpbl.txt took 2.7163 seconds due to:
  - Higher sparsity
  - Irregular subset cost distribution
- **Robustness**: Zero failures across all 4 test instances, demonstrating consistent handling of:
  - Varying matrix densities
  - Different cost ranges

Large-Scale Instances

- **Scalability**: Successfully processed railway instances with:
  - Up to 1,092,610 columns (rail4284.txt)
  - Sparse matrices
- **Performance**: Stable 30-33 second runtime across all large instances, indicating:
  - Effective memory management
  - Linear scaling with problem size
- **Cost Variation**: Total costs ranged from 12,000-45,000 units due to:
  - Regional infrastructure differences
  - Varying service requirements

Key Metrics

Cuadro 1: Performance Metrics Across Test Instances

Instance	Subsets	Cost	Time (s)	Optimal?
`scp41.txt`	65	429.0	0.3435	Yes [Beasley, 1987a]
`scp51.txt`	62	253.0	0.6362	Likely
`scpa1.txt`	67	253.0	1.5724	Likely
`scpb1.txt`	37	69.0	2.7163	Yes [Beasley, 1987a]
`rail2536.txt`	80	13,697.0	30.18	N/A
`rail4284.txt`	128	5,302.0	32.95	N/A
`rail4872.txt`	91	27,638.0	33.30	N/A

Discussion

Capabilities

The solver demonstrated strong performance across all tested instances, confirming the effectiveness of the binary linear programming approach for the SCP. Key observations include:

- **Correctness**: The model reliably produced optimal or near-optimal solutions, validating the formulation and implementation.

- **Efficiency**: Execution times were reasonable, even for large datasets, though sparsity and problem structure influenced performance.
- **Scalability**: The solver's ability to handle millions of variables without significant slowdowns highlights its usefulness for real-world applications.

However, the results also revealed areas for improvement.

Limitations

1. **Deterministic Assumptions**: The model assumes static input data, limiting its applicability to dynamic environments.
2. **Solver Dependency**: Performance is tied to the CBC solver; alternative solvers (e.g., AMPL, Gurobi) might show different results.
3. **Preprocessing**: No preprocessing steps (e.g., removing redundant variables) were applied, which could improve efficiency.
4. **Heuristics**: The lack of heuristic shortcuts may limit performance in very large or complex instances.
5. **Resource Intensity**: While scalable, the solver may require significant computational resources for extremely large datasets.

Next Steps

1. **Algorithmic Enhancements**: Implement preprocessing techniques to eliminate redundant variables and constraints, improving efficiency.
2. **Heuristic Integration**: Explore heuristic or metaheuristic approaches (e.g., genetic algorithms) to handle larger instances more effectively.
3. **Dynamic Data Handling**: Extend the model to accommodate dynamic or stochastic input data for real-time applications.
4. **Solver Comparison**: Test alternative solvers (e.g., Gurobi, AMPL) to evaluate performance differences and optimize results.
5. **Parallelization**: Investigate parallel computing techniques to reduce execution times for massive datasets.
6. **Application-Specific Tuning**: Adapt the model for specific use cases (e.g., healthcare facility placement) to enhance practicality.

Conclusion

The Set Covering Problem is fundamental in operations research because of its wide variety of applications. With this project I successfully formulated, implemented, and tested a binary linear programming model to solve the SCP. The solver demonstrated accuracy, efficiency, and scalability across small, medium, and large-scale instances. While limitations exist, the results underscore the model's potential for real-world deployment. Future work should focus on algorithmic improvements, dynamic data handling, and solver optimization to further enhance performance and applicability. The project highlights the power of mathematical modeling and computational tools in attacking optimization problems.

Referencias

[Balas & Padberg, 1972] Balas, E. & Padberg, M. W. (1972). On the set-covering problem. *Operations Research*, 20(6), 1152–1161. https://doi.org/10.1287/opre.20.6.1152

[Beasley, 1987a] Beasley, J. E. (1987a). An algorithm for set covering problems. *European Journal of Operational Research*, 31(1), 85–93.

[Beasley, 1987b] Beasley, J. E. (1987b). *OR-library: Test Data Sets for Operations Research Problems*. https://people.brunel.ac.uk/~mastjjb/jeb/orlib/scpinfo.html. May, 2025.

[Karp, 1972] Karp, R. M. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, 85–103. Held March 20-22, 1972 at the IBM Thomas J. Watson Research Center.