Project 2 Report: Set Covering Problem

# Codebase Description

Code can be run via main.jl file in the project folder. Functions for Set Covering Heuristics can be found in ./src/setCoveringHeuristics.jl file. [[Online repository]](https://github.com/Realife-Brahmin/IntCombOpt567.jl)

# Problem Description

Using various heuristic techniques, solve the Set Covering Problem for the two given test cases: *phase1* and *cap360*, each representing a ‘graph’ of various poles (facilities) and meters (customers). Report the results as the total number of poles used to cover all meters, and the time taken for the simulation.

# Methodology and Parameter Values

**cleanupRepeats** = Number of iterations (pole additions) a cleanup is run on the entire . Cleanup is also always done at least once at the end, if all the meters are covered. Value set to 100 if preprocessing is used or 1 otherwise.

**Preprocess\_steps** = Number of time the sequence of Preprocess1, Preprocess2 and Preprocess3 are run per iteration. Set as 1 typically.

Important Note regarding Preprocessing: In all my preprocessing functions including Preprocess1, Preprocess2 and Preprocess3, the decisions are NOT taken in a sequential manner, rather a batch of poles/meters are identified for the cleanup procedure and they’re all subsequently cleaned up afterwards. This is because my ‘graph’ altering functions such as addPole!, discardPole!, removePole!, ignoreMeter! have been found to be more time-consuming, and calling them after every single decision was not allowing me to meet time benchmarks.

**Preprocess2\_steps** = Maximum number of poles discarded in one *sweep* of Preprocess2. Set as 100.

**Preprocess3\_steps** = Maximum number of meters ignored in one *sweep* of Preprocess3. Set as 50.

**Preprocess2\_check\_steps** = Maximum number of pairs of poles compared as part of one *sweep* of Preprocess2. Set as 60\_000 for greedy and 120\_000 for score 2.

**Preprocess3\_check\_steps** = Maximum number of pairs of meters compared as part of one *sweep* of Preprocess3. Set as 60\_000 for greedy and 120\_000 for score 2.

Note regarding Modified Greedy: Score 1 results were typically inferior to Score 2 results, so I’ve only given Score 2 results for that row.

For cap360 test case with preprocessing, I’ve also attached snips from the terminal after the script ran.

# Result 1 (Without Preprocessing)

### *phase1:*

|  |  |  |
| --- | --- | --- |
| Scoring Function | #Poles Used | Time Taken [s] |
| Greedy | 24 | 0.001 |
| Modified Greedy (Score 2) | 25 | 0.001 |

### *cap360:*

|  |  |  |
| --- | --- | --- |
| Scoring Function | #Poles Used | Time Taken [s] |
| Greedy | 617 | 2 |
| Modified Greedy (Score 2) | 618 | 2 |

A screenshot of a computer code

AI-generated content may be incorrect.A screenshot of a computer code

AI-generated content may be incorrect.

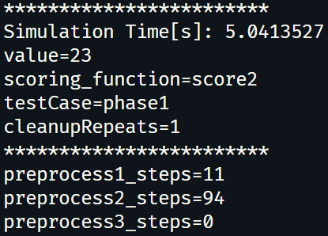
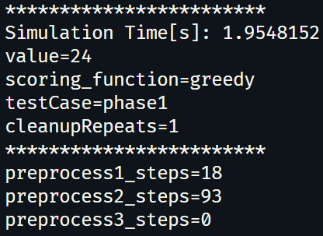
# Result 2 (With Preprocessing)

Note only for *phase1*: For *phase1* test case, at the time of taking the snapshots, preprocess3 was not implemented (but benchmarks were being achieved anyway).

preprocess\_steps was set to 100 (i.e. pp1 and pp2 were run 100 times one after the other each iteration) and preprocess2\_step was set to 1 (i.e. for each pp1 and pp2 call, exactly one pole was added or discarded).

### *phase1:*

|  |  |  |
| --- | --- | --- |
| Scoring Function | #Poles Used | Time Taken [s] |
| Greedy | 24 | 2 |
| Modified Greedy (Score 2) | 23 | 5 |



### *cap360:*

|  |  |  |
| --- | --- | --- |
| Scoring Function | #Poles Used | Time Taken [s] |
| Greedy | 580 | 100 |
| Modified Greedy (Score 2) | 552 | 211 |

A screen shot of a computer code

AI-generated content may be incorrect. A screen shot of a computer program

AI-generated content may be incorrect.

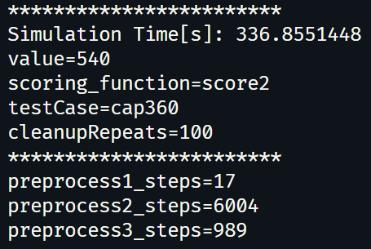
# Result 3 (With ‘Full’ Preprocessing)

Increased Preprocess\_check\_steps and Preprocess\_steps to 120\_000, 120 for both Preprocess2 and Preprocess3

### *cap360:*

|  |  |  |
| --- | --- | --- |
| Scoring Function | #Poles Used | Time Taken [s] |
| Greedy | 546 | 148 |
| Modified Greedy (Score 2) | 540 | 337 |

A computer screen shot of a code

AI-generated content may be incorrect.

# References

1. Prof. Bala Krishnamoorthy’s course notes as part of MATH 567 Integer and Combinatorial Optimization taught at Washington State University, Spring 2025. Problem Statement and Lecture notes retrieved Integer Optimization: Lecture Notes and videos. (2025, April 24). Retrieved from <https://bala-krishnamoorthy.github.io/FilesMath567/S25/LecNotes/index.html>.