

# NEIGHBOURHOOD-BASED METAHEURISTICS FOR THE SET COVERING PROBLEM

Heuristics and Metaheuristics

Rafael Marques Claro

November 2020

## SOLUTION REPRESENTATION

A vector that represents the selection, or not, of each subset:

```
std::vector<bool> x; // x[i] = 1 if subset i is selected, 0 otherwise
```

A vector that represents the already covered attributes:

```
std::vector<bool> y; // y[j] = 1 if attribute j is covered, 0 otherwise
```

A vector representative of which subsets cover each attribute:

```
std::vector<std::vector<int>> a; // attribute j covered by subset i
```

A vector representative of which attributes are covered by each subset:

```
std::vector<std::vector<int>> b; // subset i covers attribute j
```

## GENERAL ALGORITHM LOOP

- While (Uncovered attributes  $> 0$ ):
  - Dispatching rule (CH1, CH2 or CH3) – Select a subset
  - Update the vector of attributes covered by the selected subset
  - Update the vector of subsets that cover each attribute
  - Erase the already covered attributes from the attribute vector of each of the non used subsets
- Remove redundant subsets
- Implement Local Search
- Calculate the total cost of the selected subsets

# REDUNDANCY ELIMINATION PROCEDURE

- **Approval list:**

Subsets that are the only ones covering at least one attribute – cannot be removed

- **Rejection list:**

Subsets that are never the only ones covering an attribute – redundant subsets

$$\text{Subsets in the rejection list} = \text{Selected subsets} - \text{Subsets in the approval list}$$

**Algorithm loop:**

- While (Subsets in the rejection list > 0):
  - Update approval list and rejection list
  - Select the subset from the rejection list with the highest cost per number of attributes served
  - Remove the previous selected subset from the selected subsets vector
  - Update the vector of which subsets cover each attribute

## CONSTRUCTIVE HEURISTICS DISPATCHING RULES

**CH1:** At each stage, choose the subset that contains the highest coefficient of the number of uncovered attributes by its cost.

**CH2:** At each stage, select the subset with the lowest cost that serves the attribute that is covered by the fewest number of subsets

**CH3:** At each stage, randomly select one of the top three subsets that contain the highest coefficient of the number of uncovered attributes by its cost.

| Avg. % dev.<br>best known<br>solutions | CH1    | CH2    | CH3    |
|--|--------|--------|--------|
| Before<br>RE                           | 13.62% | 17.39% | 20.78% |
| After<br>RE                            | 5.57%  | 9.03%  | 9.93%  |
| Average<br>Elapsed<br>Time:            | 5.4 ms | 6.1 ms | 5.5 ms |

# LOCAL SEARCH

## IMPROVEMENT HEURISTICS

### **Best Improvement and First Improvement Algorithms Loop:**

While (New improved solution found):

- Neighbourhood Search ( $N=1$ )
  - Neighbour: Randomly add a not used subset to current best solution
  - Update the vector of subsets that cover each attribute
  - Remove redundant subsets
  - Compare the total cost of this neighbour with the best solution
  - If it represents a new best solution:
    - If FI → Stop searching
    - Else → Save it
  - Otherwise, continue searching for a better neighbour
- Update current best solution with the latest neighbour found

# LOCAL SEARCH

## FIRST IMPROVEMENT vs BEST IMPROVEMENT

Average percentage deviation from best known solutions

| Local Search \ Initial Solution | CH    | CH+RE |
|---------------------------------|-------|-------|
| FI                              | 5.52% | 5.49% |
| BI                              | 4.71% | 5.15% |

- Best Improvement → Iterates through all the neighbourhood, therefore better results.
- Before Redundancy Elimination → Wider search space, producing better results.

Fraction of instances that profit from local search

| Local Search \ Initial Solution | CH   | CH+RE  |
|---------------------------------|------|--------|
| FI                              | 100% | 80.95% |
| BI                              | 100% | 80.95% |

Average Elapsed Time (ms)

| Local Search \ Initial Solution | CH     | CH+RE  |
|---------------------------------|--------|--------|
| FI                              | 171.16 | 171.11 |
| BI                              | 557.01 | 255.50 |

- First Improvement → Stops at the first best neighbour, therefore shorter computing times.
- After Redundancy Elimination → Smaller search space, resulting in shorter computing times.

# GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

## Simple GRASP Approach:

- **Constructive Heuristic dispatching rule:**  
At each stage, randomly select one of the subsets that have the highest coefficient of the number of uncovered attributes by the cost of the subset.
- **Initial Solution:**  
CH+RE → As multiple iterations will be performed in this algorithm, the efficiency was the priority.

## Hybrid Grasp Approach:

- **Neighbourhood Enlargement:**  
Escaping from a local optimum → add a set of random unused subsets to the current solution
- **Dynamic  $\alpha$ :**

$$\alpha = \frac{coeff}{ip} + offset$$



# GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

## Algorithm loop:

- While ( $i < \text{max\_iterations}$ ):
  - Generate initial solution (with randomness)
  - Perform Local Search (BI)
  - Apply Redundancy Elimination
  - If it represents a new best Solution, save that Solution  
Otherwise, continue searching for a better solution
  - Increment  $i$

| $\alpha=0,2\%$ | Average Deviation | Avg. Opt. Solutions met | Avg. Elap. Time per Instance |
|----------------|-------------------|-------------------------|------------------------------|
| $i = 50$       | 1.75%             | 3.8                     | 19.8 s                       |
| $i = 500$      | 0.88%             | 10.4                    | 181.8 s                      |

# HYBRID GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

## Algorithm loop:

- While ( $i < \text{max\_iterations}$ ):
  - Generate initial solution (with randomness)
  - While (new improved solution found):
    - Perform Local Search (BI)
    - Apply Redundancy Elimination
    - If it represents a new best solution, save that solution
  - Else:
    - While (unused subsets can be selected):
      - Add a random set of unused subsets to the current solution
      - Apply Redundancy Elimination
      - If it represents a new best solution, save that solution
    - Otherwise, continue searching for a better solution
- Increment  $i$

| $\alpha = \frac{0,2}{i^4} + 0,002$ | Average Deviation | Avg. Opt. Solutions met | Avg. Elap. Time per Instance |
|------------------------------------|-------------------|-------------------------|------------------------------|
| $i = 50$                           | 1.71%             | 4.8                     | 24.0 s                       |
| $i = 500$                          | 0.69%             | 16                      | 228.6 s                      |