Algorithmic Methods for Mathematical Models
– COURSE PROJECT –

Thomas Aubertier & Alba Soldevilla

December 8, 2025

Figure 1: He is the night

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Contents

1 Problem Statement

We formally define the problem as follows.

1.1 Inputs

- N: number of crossings, indexed by $i, j = 1, \ldots, N$.
- K: number of camera models, indexed by $k = 1, \ldots, K$.
- P_k: purchase price in € for a camera unit of model k.
- R_k: coverage range of model k, with $1 \leq R_k \leq 49$.
- A_k: maximum number of consecutive days that a camera of model k can operate, with $2 \leq A_k \leq 6$.
- E_k: power consumption per day when a camera of model k is turned on, with $0 \leq E_k$.
- M_{ij}: minimum range required for a camera located at crossing i to cover crossing j. Coverage is possible if and only if $M_{ij} \leq R_k$. By definition, $M_{ij} \geq 50$ indicates infeasibility. Moreover, $M_{ii} = 0$ for all $1 \leq i \leq N$.
- A weekly schedule, indexed by $d = 1, \ldots, 7$, being $Monday = 1, ..., Sunday = 7$.

Note that in the original project statement the power consumption is denoted by C_k. For simplicity, and to avoid clashes with other variables in our implementation, we rename it as E_k throughout this report.

1.2 Outputs

The required outputs are:

1. The set of crossings where cameras are installed.
2. The model assigned to each installed camera.
3. A weekly activation schedule indicating, for every installed camera, on which days it is operating.

1.3 Constraints

The solution must satisfy:

- At most one camera can be installed at each crossing.
- Every crossing must be covered on every day of the week.
- Activation only if installed.
- A camera of model k cannot operate for more than A_k consecutive days.
- If a camera starts operating, it must stay on for at least 2 consecutive days.

1.4 Objective

Since Wayne Corporation is currently going through a very bad financial situation, the objective is to minimize the total cost consisting of:

- the installation cost of the cameras, and
- the operational cost over the 7-day schedule.

2 Integer Linear Programming Model

2.1 Decision Variables

- Installation variable:

$$x_{ik} = \begin{cases} 1, & \text{if a camera of model } k \text{ is installed at crossing } i, \\ 0, & \text{otherwise.} \end{cases}$$

- Operation variable:

$$y_{ikd} = \begin{cases} 1, & \text{if that camera is ON on day } d, \\ 0, & \text{otherwise.} \end{cases}$$

2.2 Objective Function

$$z = \min \left(\sum_{i=1}^{N} \sum_{k=1}^{K} P_k x_{ik} + \sum_{i=1}^{N} \sum_{k=1}^{K} \sum_{d=1}^{7} E_k \, y_{ikd} \right)$$

2.3 Constraints

1. At most one camera per crossing:

$$\sum_{k=1}^{K} x_{ik} \;\leq\; 1 \qquad \forall i = 1, \ldots, N.$$

2. Coverage of every crossing on every day by at least one operating camera:

$$\sum_{i=1}^{N} \sum_{\substack{k=1 \\ M_{ij} \leq R_k}}^{K} y_{ikd} \;\geq\; 1, \qquad \forall j = 1, \ldots, N, \; \forall d = 1, \ldots, 7.$$

3. Activation only if installed:

$$y_{ikd} \;\leq\; x_{ik} \qquad \forall i = 1, \ldots, N, \; \forall k = 1, \ldots, K, \; \forall d = 1, \ldots, 7.$$

4. A camera of model k cannot operate more than A_k consecutive days:

$$\sum_{h=0}^{A_k} y_{ik,\, ((d+h-1) \bmod 7)+1} \;\leq\; A_k \qquad \forall i = 1, \ldots, N, \; \forall k = 1, \ldots, K, \; \forall d = 1, \ldots, 7.$$

5. Minimum 2-day operation period:

$$y_{ikd} - y_{ik,d-1} \;\leq\; y_{ik,d+1} \qquad \forall i = 1, \ldots, N, \; \forall k = 1, \ldots, K, \; \forall d = 2, \ldots, 6.$$

For the first day:

$$y_{ik1} - y_{ik7} \;\leq\; y_{ik2} \qquad \forall i = 1, \ldots, N, \; \forall k = 1, \ldots, K.$$

For the last day:

$$y_{ik7} - y_{ik6} \;\leq\; y_{ik1} \qquad \forall i = 1, \ldots, N, \; \forall k = 1, \ldots, K.$$

3 Algorithm's pseudo-code

3.1 Greedy

The greedy constructive heuristic builds a feasible solution incrementally by repeatedly selecting the candidate that provides the best trade-off between additional coverage and cost. At each iteration, the algorithm evaluates all feasible candidates, chooses the one with the highest quality ratio $q(c)$, updates the set of covered crossings, and removes candidates that violate problem constraints. The process continues until full weekly coverage is achieved for every crossing or no feasible candidates remain. Here you can see the pseudo-code of our implementation:

Algorithm 1 Greedy constructive heuristic

Require: Input data (K, P, R, A, E, N, M)
$C \leftarrow \textsc{InitCandidates}(K, R, A, N, M)$ ▷ Initialize candidates
$S \leftarrow \emptyset$ ▷ Initialize solution
$covered \leftarrow \emptyset$ ▷ Set of covered crossings, stores the pairs (j, d)
while $\neg\textsc{IsSolution}(covered, N)$ **do** ▷ While not all crossings are covered
    $(c^\star, q^\star, \_, \_) \leftarrow \textsc{EvaluateQuality}(C, covered, P, E)$
    if $c^\star =$ **null then**
        break ▷ No feasible candidate left
    $S \leftarrow S \cup \{c^\star\}$ ▷ Add best candidate to the solution
    $covered \leftarrow covered \cup c^\star.covers$ ▷ Update covered crossings
    $C \leftarrow \textsc{Feasibility}(C, c^\star)$ ▷ Update candidates by remove the infeasible ones
$total\_cost \leftarrow \textsc{TotalCost}(S, P, E)$
return $(S, covered, total\_cost)$

Implementation details:

- The function **InitCandidates** builds the initial candidate set C. For each crossing i and camera model k, it checks whether installing a camera of model k at crossing i is feasible. If so, it computes the set of crossings that would be covered according to the matrix M and generates all feasible weekly activation patterns (respecting the autonomy and minimum–activation constraints of model k). Each candidate $c \in C$ stores its model, the crossings it covers, and its weekly pattern.
- The procedure **Feasibility** updates the candidate set after selecting a new camera. Whenever a candidate $c^\star$ is added to the solution S, Feasibility removes from C all candidates associated with the same crossing as $c^\star$, thus enforcing the constraint that at most one camera can be installed at each crossing.
- The function **IsSolution** checks whether the current partial solution already satisfies the coverage requirement. Then IsSolution iterates over all crossings $j = 1, \ldots, N$ and all days $d = 1, \ldots, 7$ and returns *true* only if every pair (j, d) belongs to *covered*.
- Finally, the function **TotalCost** computes the objective value of the greedy solution S. For each selected candidate $c \in S$, it adds the installation cost $P_{k(c)}$ of its model and the operational cost over the week, given by $E_{k(c)}$ multiplied by the number of active days in its pattern. The total cost is then the sum of these contributions.

Quality evaluation: For this problem, we evaluate the quality of each candidate using the following formula:

$$q(c) = \frac{\text{gain}(c)}{\text{cost}(c)} = \frac{\left|covers(c) \setminus covered\right|}{P_{k(c)} + E_{k(c)} \cdot |days(c)|},$$

where the gain of a candidate corresponds to the number of *new* crossings that it can cover (i.e., those not already covered by previously selected candidates), and the cost corresponds to the model purchase price $P_{k(c)}$ plus the operational cost $E_{k(c)}$ multiplied by the number of days on which candidate c is active ($|days(c)|$). This function is implemented in **EvaluateQuality**, which evaluates all candidates in the set C and returns the one with the highest value of $q(c)$. The corresponding pseudo-code for this function is shown below:

Algorithm 2 EvaluateQuality

Require: Candidate set C, current covered set *covered*, vectors of purchasing cost P and operating cost E

$q_{\max} \leftarrow -\infty$, $q_{\min} \leftarrow +\infty$
$c_{\max} \leftarrow$ **null**, $c_{\min} \leftarrow$ **null**
for all $c \in C$ **do**
    $new\_covered \leftarrow c.covers \setminus covered$ ▷ New crossings that would be covered
    $gain \leftarrow |new\_covered|$
    $purchase\_cost \leftarrow P_{k(c)}$
    $operating\_cost \leftarrow E_{k(c)} \cdot |days(c)|$
    $cost \leftarrow purchase\_cost + operating\_cost$
    $q(c) \leftarrow \dfrac{gain}{cost}$ ▷ Greedy quality function
    if $q(c) > q_{\max}$ **then**
        $q_{\max} \leftarrow q(c)$
        $c_{\max} \leftarrow c$
    if $q(c) < q_{\min}$ **then**
        $q_{\min} \leftarrow q(c)$
        $c_{\min} \leftarrow c$
return $(c_{\max}, q_{\max}, c_{\min}, q_{\min})$

Note that the greedy heuristic only uses $c_{\max}$ (the candidate with maximum quality). The remaining returned values will be required later for GRASP, specifically for constructing the Restricted Candidate List (RCL).

Tie-breaking rule. When two or more candidates obtain the same quality value $q(c)$, we resolve ties by selecting the first candidate encountered during the iteration over the set C. This choice provides a simple and deterministic behaviour. However, alternative tie-breaking strategies could also be used (e.g. selecting the candidate with lower cost, higher gain, or even choosing randomly), and different criteria may lead the greedy algorithm to construct different solutions. In this project, we adopt the first-come rule for simplicity.

3.2 Local Search

Starting from the greedy solution, we apply a local search procedure based on 1-exchange moves: at each iteration, the algorithm tries to replace one installed camera by another feasible candidate, as long as the new solution remains feasible (full weekly coverage and at most one camera per crossing) and strictly improves the total cost.

The neighbourhood can be explored either with a *first improvement* policy (FI), where the search stops as soon as an improving move is found, or with a *best improvement* policy (BI), where all neighbours are evaluated and the best improving move is selected.

Algorithm 3 Local search with 1-exchange moves

Require: Input data (K, P, R, A, E, N, M), initial solution S (from greedy), policy $policy \in \{0, 1\}$

if $S =$ **null then**
  return (**null**, **null**, **null**) ▷ Greedy found no solution
$C_{\text{all}} \leftarrow \text{InitCandidates}(K, R, A, N, M)$ ▷ All possible candidates
$improved \leftarrow$ **true**
while *improved* **do**
  $improved \leftarrow$ **false**
  $current\_cost \leftarrow \text{TotalCost}(S, P, E)$
  $best\_delta \leftarrow 0$, $best\_move \leftarrow$ **null**
  for $c = 1$ **to** $|S|$ **do** ▷ Index of camera to remove
    $c_{\text{out}} \leftarrow S[c]$
    $S_{\text{without}} \leftarrow S \setminus \{c_{\text{out}}\}$
    for all $c_{\text{in}} \in C_{\text{all}}$ **do**
      if $\exists s \in S$ such that $\text{IsSameCamera}(c_{\text{in}}, s)$ **then**
        continue ▷ Skip identical camera
      if not $\text{ValidCrossingConstraint}(S_{\text{without}}, c_{\text{in}})$ **then**
        continue ▷ At most one camera per crossing
      $S_{\text{neigh}} \leftarrow S_{\text{without}} \cup \{c_{\text{in}}\}$
      $covered_{\text{neigh}} \leftarrow \text{ComputeCovered}(S_{\text{neigh}})$
      if not $\text{IsSolution}(covered_{\text{neigh}}, N)$ **then**
        continue ▷ Full weekly coverage not satisfied
      $new\_cost \leftarrow \text{TotalCost}(S_{\text{neigh}}, P, E)$
      $\Delta \leftarrow current\_cost - new\_cost$
      if $\Delta \leq 0$ **then**
        continue ▷ No improvement
      if $policy = 0$ **then** ▷ First Improvement
        $S \leftarrow S_{\text{neigh}}$
        $improved \leftarrow$ **true**
        break ▷ We found the First Improvement
      else ▷ Best Improvement
        if $\Delta > best\_delta$ **then**
          $best\_delta \leftarrow \Delta$
          $best\_move \leftarrow (c_{\text{out}}, c_{\text{in}})$
    if $policy = 0$ **and** *improved* **then**
      break ▷ Leave outer loop as well
  if $policy = 1$ **and** $best\_move \neq$ **null then**
    $(c^{\star}_{\text{out}}, c^{\star}_{\text{in}}) \leftarrow best\_move$
    $S \leftarrow \left(S \setminus \{c^{\star}_{\text{out}}\}\right) \cup \{c^{\star}_{\text{in}}\}$
    $improved \leftarrow$ **true**
$final\_covered \leftarrow \text{ComputeCovered}(S)$
$final\_cost \leftarrow \text{TotalCost}(S, P, E)$
return $(S, final\_covered, final\_cost)$

3.3 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) combine a randomized greedy construction with a local search phase. In each iteration, the algorithm builds a feasible solution using a greedy cost function, but instead of always selecting the best candidate, it samples one element from a Restricted Candidate List (RCL) controlled by a parameter $\alpha \in [0, 1]$. Over multiple iterations, the best solution found is returned.

Overall GRASP procedure. The high-level pseudo-code of GRASP is shown below.

Algorithm 4 GRASP metaheuristic

Require: Input data (K, P, R, A, E, N, M), local search ls, *policy*, maximum iterations *maxIt*, parameter α

$S^* \leftarrow$ **null**, $covered^* \leftarrow$ **null**
$cost^* \leftarrow +\infty$
for $it = 1$ **to** *maxIt* **do**
    $(S, covered, cost) \leftarrow$ GraspConstruct$(K, P, R, A, E, N, M, \alpha)$
    if $S =$ **null then**
        continue ▷ No feasible solution in this iteration
    if ls **then**
        $(S', covered', cost') \leftarrow$ LocalSearch$(K, P, R, A, E, N, M, S, policy)$
    if $cost' < cost^*$ **then**
        $S^* \leftarrow S'$
        $covered^* \leftarrow covered'$
        $cost^* \leftarrow cost'$
if $S^* =$ **null then**
    return (**null**, **null**, **null**)
else
    return $(S^*, covered^*, cost^*)$

Constructive phase. The constructive phase of GRASP is a randomized variant of the greedy heuristic. At each iteration, it first evaluates all candidates to obtain $q_{\max}$ and $q_{\min}$, then builds the RCL (Restricted Candidate List) and randomly selects one candidate from it.

Algorithm 5 GRASP constructive phase

Require: Input data (K, P, R, A, E, N, M), parameter α

$C \leftarrow$ InitCandidates(K, R, A, N, M)
$S \leftarrow \emptyset$
$covered \leftarrow \emptyset$
while ¬IsSolution$(covered, N)$ **and** $C \neq \emptyset$ **do**
    $(c_{\max}, q_{\max}, \_, q_{\min}) \leftarrow$ EvaluateQuality$(C, covered, P, E)$
    if $c_{\max} =$ **null then**
        break
    RCL $\leftarrow$ BuildRCL$(C, covered, P, E, q_{\max}, q_{\min}, \alpha)$
    if RCL $= \emptyset$ **and** $c_{\max} \neq$ **null then**
        RCL $\leftarrow \{c_{\max}\}$
    else if RCL $= \emptyset$ **then**
        break
    $c^{\text{rand}} \leftarrow$ RandomChoice(RCL)
    $S \leftarrow S \cup \{c^{\text{rand}}\}$
    $covered \leftarrow covered \cup c^{\text{rand}}.covers$
    $C \leftarrow$ Feasibility(C, c^{rand})
if ¬IsSolution$(covered, N)$ **then**
    return (**null**, **null**, **null**)
else
    $total\_cost \leftarrow$ TotalCost(S, P, E)
    return $(S, covered, total\_cost)$

Restricted Candidate List (RCL). Given the current candidate set C and the greedy quality function $q(c)$, GRASP builds the RCL as

$$\text{RCL} = \{c \in C \mid q(c) \geq q_{\max} - \alpha(q_{\max} - q_{\min})\}.$$

where the parameter $\alpha \in [0, 1]$ controls the size of the RCL. When $\alpha = 0$, only the best candidate is selected (pure greedy). Larger values of α allow more candidates into the RCL, adding randomness and increasing diversification during the construction phase.

Algorithm 6 BUILDRCL

Require: Candidate set C, current covered set *covered*, costs P, E, quality bounds $q_{\max}, q_{\min}$, parameter α

```
RCL ← ∅
for all c ∈ C do
    new_covered ← c.covers \ covered
    gain ← |new_covered|
    cost ← P_{k(c)} + E_{k(c)} · |days(c)|
    q(c) ← gain/cost
    threshold ← q_max − α(q_max − q_min)
    if q(c) ≥ threshold then
        RCL ← RCL ∪ {c}
return RCL
```

4 Comparisons and Results

This section presents the experimental evaluation of our methods. We first tune the parameters of GRASP and then compare the performance of all algorithms against the CPLEX model.

4.1 Parameter Tuning

To study the behaviour of GRASP and tune its parameters, we generated three instances of increasing size: a small instance (N = 4, K = 2), a medium instance (N = 10, K = 5), and a large instance (N = 20, K = 10).

Maximum Iterations (*maxIt*): To analyse the effect of the iteration limit, we ran several tests with different iteration limits using a fixed $\alpha = 0.5$, which provides a balanced level of randomness and avoids both purely greedy behaviour and fully random construction. All of the tests were executed with local search. The following graphs show the obtained results.

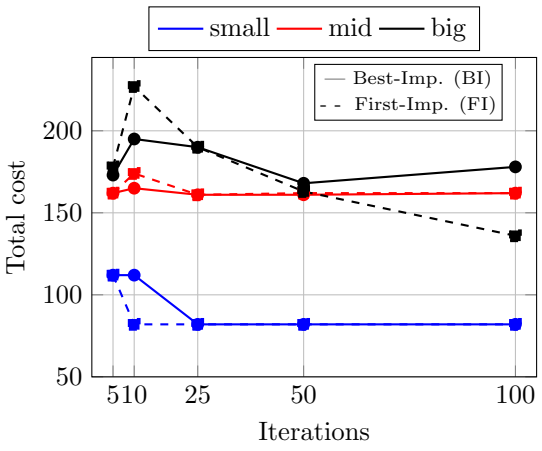

Figure 2: Total cost vs. iterations, with $\alpha = 0.5$.

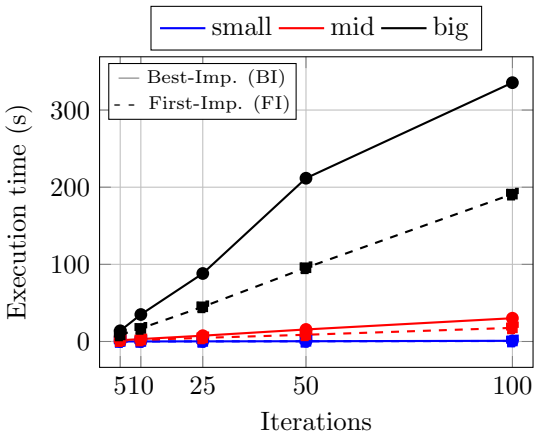

Figure 3: Elapsed time vs. iterations, with $\alpha = 0.5$.

The results show that, although increasing the number of iterations generally improves solution quality, GRASP is a stochastic metaheuristic and occasional runs with fewer iterations may still outperform longer ones. Execution time grows roughly linearly with the iteration limit. Therefore, for the remaining experiments we fixed the maximum number of iterations to 50, as it provides a good compromise between solution quality and running time. We also fixed the policy to First Improvement, since it yields results comparable to Best Improvement while being noticeably faster.

Alpha parameter (α): We also ran several test with the previous instances generated in order to find the alpha that best fits our project. The results are shown in the following graph:

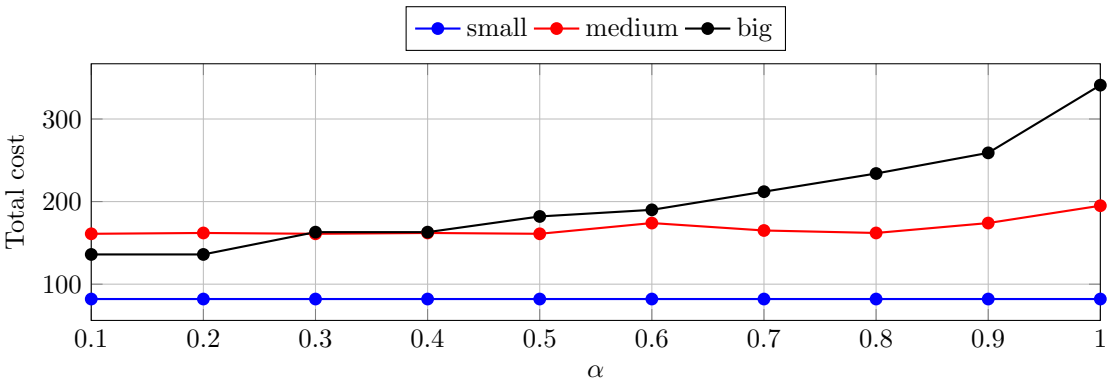

Figure 4: Effect of α on total cost (50 iterations, First Improvement policy).

For the small instance, all values of α produce the same cost, indicating that randomness has little influence when the search space is small. For the medium instance, values in the range $\alpha \in [0.1, 0.5]$ yield the best results, while higher values introduce excessive randomness and degrade solution quality. For the big instance, this effect is even more pronounced: increasing α generally worsens the cost, especially beyond $\alpha > 0.6$. Based on these observations, we fix $\alpha = 0.1$ for the remaining experiments, as it consistently provides the best overall performance. That was also the case for labs 4 and 5.

4.2 Instance Generation

In this section we compare the performance of the heuristic methods with the CPLEX model, evaluating both solution quality and computation time as the instance size increases. The instances used in our experiments are summarised below:

Instance	**N**	**K**	$\boldsymbol{N \cdot K}$
1	5	5	25
2	7	7	49
3	10	10	100
4	20	20	400
5	30	30	900
6	50	50	2500
7	75	75	5625
8	2	20	40
10	10	20	200
11	40	2	80
12	40	20	800

Table 1: Set of generated instances used for performance evaluation.

Most of the instances satisfy $N = K$, but it is also interesting to examine whether N and K contribute equally to the problem's complexity. The last four instances are only used in Section 4.5.

4.3 Greedy Performance

This section compares the three available settings for Greedy algorithm :

- Greedy constructive (simple)
- Greedy constructive + Local search with first improvement policy
- Greedy constructive + Local search with best improvement policy

We will look at two metrics : the optimal solution found and the execution time.

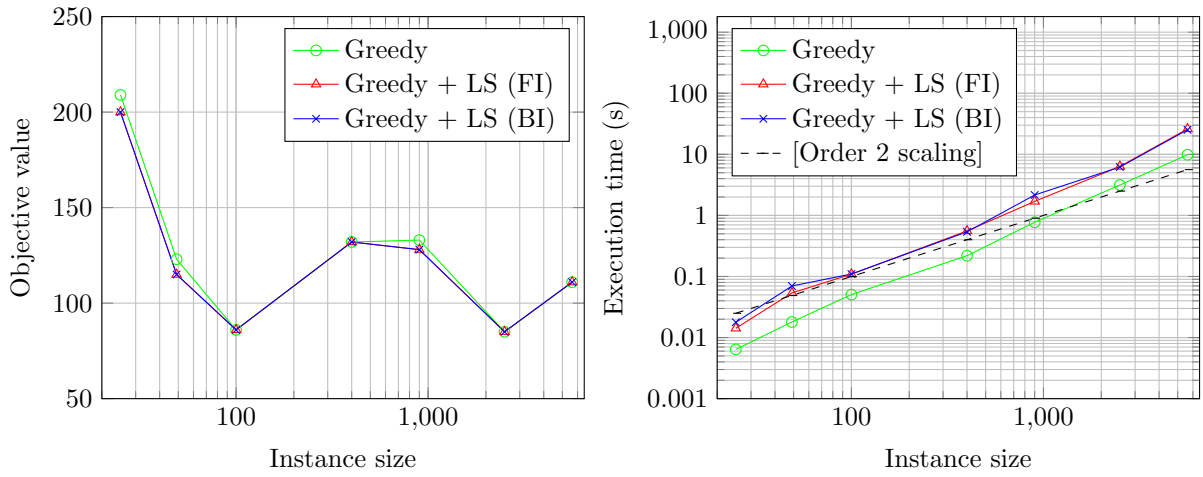

Figure 5: Objective value and execution time as a function of the instance size for Greedy.

Overall, the performance are very similar regardless of the policy used. In particular, all are following a scaling a little above order 2 (time $\sim N \cdot K$).

Not using local search saves some time but slightly degrades solution quality. Since none of the tested instance sizes produced prohibitive running times, we prioritise solution quality and therefore favour the use of local search.

Best-Improvement yields similar solution quality while requiring higher running times, making First-Improvement a more efficient trade-off. This is consistent with the results from Section 4.1. Consequently, the First-Improvement setting will be used for evaluating Greedy in the comparison against GRASP and ILP.

4.4 GRASP Performance

As with Greedy, we want to evaluate the benefits of using local search into GRASP. For all instances, we will run 50 iterations of each case. We therefore consider the following cases:

- GRASP constructive (simple)
- GRASP constructive + Local search with first improvement policy
- GRASP constructive + Local search with best improvement policy

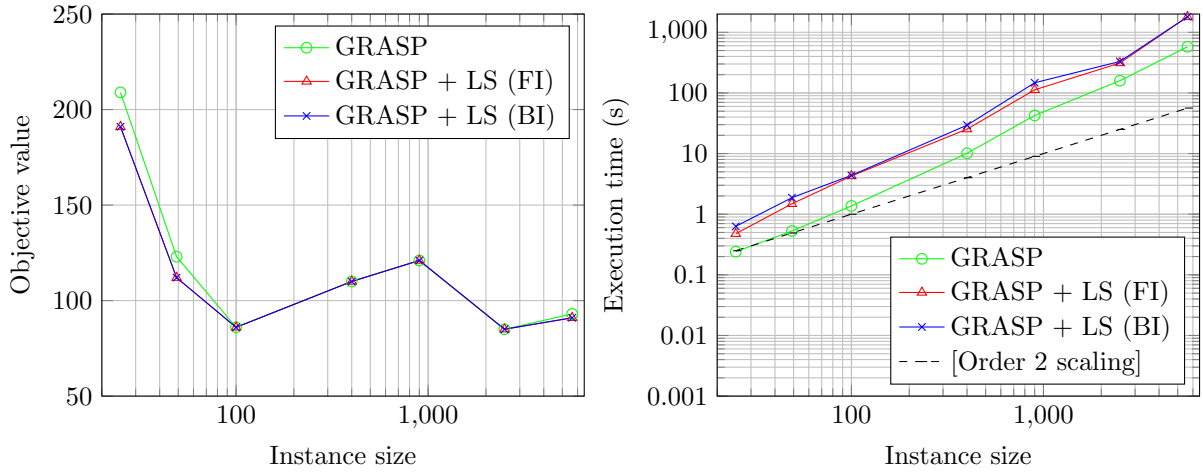

Figure 6: Objective value and execution time as a function of the instance size for GRASP.

First of all, the scaling is similar to Greedy for all models. Using local search can double or even triple the total execution time, with Best-Improvement being the most expensive variant.

Interestingly, plain GRASP becomes as efficient as GRASP with local search for instance sizes above 100, performing worse only on the smallest cases (and for size 5625). Although this configuration could be considered appealing, its behaviour remains more stochastic because it explores fewer candidate solutions. Moreover, since its scaling is comparable, the time savings are not as large as they may initially appear[1].

As with Greedy, First-Improvement consistently outperforms Best-Improvement in terms of running time, while achieving very similar solution quality.

4.5 Comparison between Heuristics and CPLEX

Now that we have chosen the best setting from Greedy and GRASP, let's compare them with the ILP implementation with CPLEX :

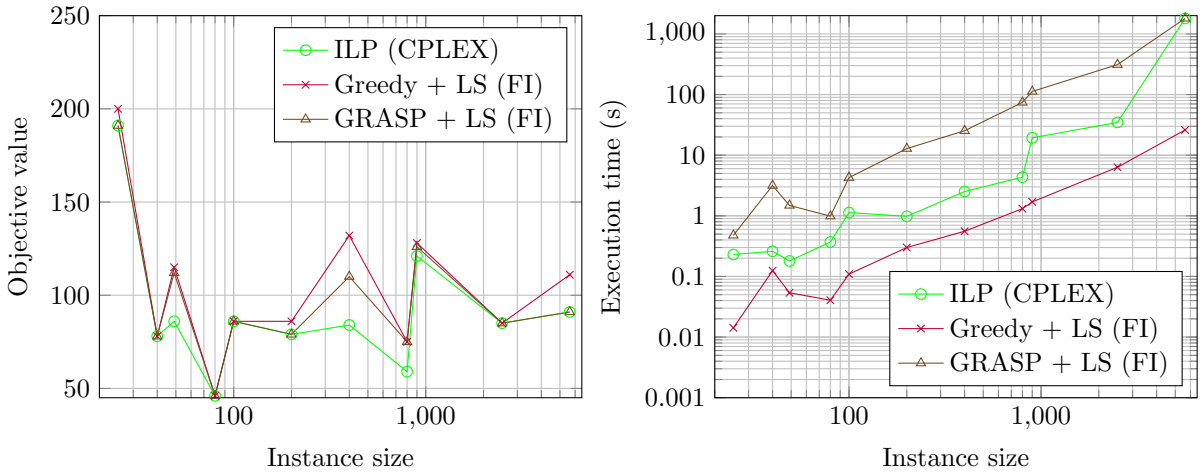

Figure 7: Objective value and execution time as a function of the instance size for all methods used.

Models where $K > N$ require significantly more time than those with $K = N$, even when the instance size $K \cdot N$ is the same (e.g., instances 40 and 200). The opposite can be said for instances with $K < N$ (e.g., 80 and 800). This indicates that the number of crossings has a stronger impact on computational performance than the number of camera models, which is reasonable given that the coverage matrix M grows with N. These instances exhibit a slightly different impact on ILP execution time, although the relative ranking of methods remains unchanged.

Among the three models, GRASP performs the worst overall, delivering lower-quality solutions and significantly higher running times than ILP for all tested instances.

However, the choice between ILP and Greedy ultimately depends on the computing resources Wayne Corporation can utilize. In practice, ILP becomes impractical for large instances, whether due to the number of crossings or the variety of camera models.

Given that Gotham City is a very large city, the best option is probably to use the Greedy algorithm with First-Improvement local search. Most of the time, the objective value is close to ILP while being mostly 10 times faster. Its running-time scaling is much more stable (plot being linear), whereas the ILP running time grows exponentially making it safer and more reliable in the long run (note that ILP timeouts at 5625).

It is also worth noting that CPLEX makes use of all available processors, whereas our Python implementation of Greedy runs on a single core. Solving the problem with ILP is therefore way more expensive and may become a bottleneck if Wayne Corporation has limited computational resources or a tight budget.

GitHub Repository

[•] https://github.com/AUBERTIERThomas/AMMM_MIRI-Project

[1] The final conclusions do not change whether we choose GRASP or GRASP + FI.