Algorithmic Methods for Mathematical Models – COURSE PROJECT –

Ezgi Sena Karabacak, Davide Lamagna

May 25, 2025

Introduction

In this project, we address a resource-sharing optimization problem inspired by a cooperative of cryptocurrency miners. Each of the N members contributes equally to the acquisition of a shared GPU-enabled computer and requests access for a fixed number of days each month. To resolve scheduling conflicts, members can bid for priority over others on overlapping days, resulting in a directed graph of priority relations. The cooperative collects the total value of accepted bids, but to avoid deadlocks, the graph must remain acyclic.

Our goal is to construct a set of priority relations that maximizes the total bid value while ensuring acyclicity. We first formulate this as an Integer Linear Programming (ILP) problem and solve it using CPLEX. Due to scalability limitations of exact methods, we then develop three metaheuristic algorithms: a greedy constructive approach, a greedy + local search combination, and GRASP with optional local search refinement. These algorithms are evaluated across multiple problem sizes using synthetic instances with bids uniformly sampled in the range $[1, 10]$.

Our work focuses on both the design and experimental comparison of these algorithms in terms of solution quality and runtime efficiency. We tune the α parameter in GRASP, analyze performance trends across increasing problem sizes, and identify the most effective strategies under time constraints.

1 Formal Problem Definition

A cooperative of N members shares access to a single GPU-enabled computer for cryptocurrency mining. Each month, all members request access for an equal number of days, potentially leading to scheduling conflicts. To resolve these conflicts, pairwise priorities between members can be assigned. For every pair of members i and j (where $i \neq j$), if both request the computer on the same day, then:

- Member i can bid $m_{ij} > 0$ to gain priority over j.
- Similarly, j can bid $m_{ji} > 0$ to gain priority over i.
- If no conflict is possible (including $i = j$), $m_{ij} = m_{ji} = 0$.

The goal is to choose a subset of directed priority relations that:

1. Maximizes the total collected bid value.
2. Forms an acyclic graph (i.e., avoids cycles in priority, which would lead to deadlocks).

Inputs

- N: Number of members in the cooperative.
- $M = [m_{ij}]$: A $N \times N$ matrix where m_{ij} is the amount (in euros) that member i bids to get priority over member j.

Outputs

- A directed graph (or priority relation) indicating which member has priority over which, such that:
  - The resulting directed priority graph must be acyclic.
  - The total amount collected by the cooperative is maximized.

2 Integer Linear Programming Model

Decision Variables

- $x_{ij} \in \{0, 1\}$: Binary variable equal to 1 if member i has priority over member j, and 0 otherwise.
- $u_i \in \{1, 2, \ldots, N\}$: Integer variable representing the position of member i in a topological ordering used to prevent cycles.

Objective Function

Maximize the total collected bids from chosen priority relations:

$$\max \sum_{\substack{i=1 \\ i \neq j}}^{N} \sum_{j=1}^{N} x_{ij} \cdot m_{ij}$$

Constraints

1. **No Self-Priority:** No member can have priority over themselves:

$$x_{ii} = 0 \quad \forall\, i \in \{1, \ldots, N\}$$

2. **Acyclicity (Cycle Prevention):** Use topological ordering variables to ensure the resulting graph has no cycles. For all $i \neq j$:

$$u_i + 1 \leq u_j + (1 - x_{ij}) \cdot N$$

   This constraint enforces that if $x_{ij} = 1$ (i.e., i has priority over j), then $u_i < u_j$.

3. **Variable Domains:**

$$x_{ij} \in \{0, 1\}, \quad u_i \in \{1, 2, \ldots, N\} \quad \forall\, i, j \in \{1, \ldots, N\}$$

3 Heuristic algorithms

To complement the CPLEX optimization, we implemented three meta-heuristic algorithms: (1) a greedy constructive heuristic, (2) the same greedy approach followed by a local search improvement, and (3) a GRASP heuristic, which combines a randomized greedy construction with the same local search procedure for further refinement.

3.1 Greedy Constructive Heuristic

The greedy algorithm sorts all potential arcs (i, j) by bid value m_{ij} in descending order and incrementally adds them to the solution if they do not create cycles.

Algorithm 1: Greedy Constructive Heuristic

```
Input: Bid matrix $m_{ij}$
Output: Acyclic orientation $A$
Initialize $A \leftarrow \varnothing$;
Add all nodes $i \in \{1, \dots, N\}$ to $A$;
Sort all arcs $(i, j)$ with $i \neq j$ by $m_{ij}$ in descending order;
for each $(i, j)$ in sorted list do
    Temporarily add edge $(i, j)$ to $A$;
    if $A$ remains acyclic then
        Permanently keep edge $(i, j)$;
    else
        Remove edge $(i, j)$ from $A$;
return $A$
```

3.2 Greedy + Local Search

This method improves the greedy solution using a local search based on permutations of the topological ordering. Instead of exhaustively testing all possible node swaps, we limit the search to local permutations by only swapping each node i with nearby nodes at positions $i+\delta$, where $\delta \in \{1, 2, 3\}$. A swap is accepted if it yields an acyclic graph and improves the total fitness.

We use a **best improvement strategy**, meaning that in each iteration, all allowed swaps are evaluated and the one yielding the highest improvement is applied. The search continues until no improving move is found or a timeout is reached.

Algorithm 2: Local Search Algorithm

```
Input: Initial solution $A$, bid matrix $m_{ij}$
Output: Improved solution $A'$
Initialize order ← topological sort of $A$;
Compute initial fitness $f \leftarrow \sum_{(i,j)\in A} m_{ij}$;
repeat
    for each $i \in \{1, \dots, N-1\}$ do
        for $\delta \in \{1, 2, 3\}$ do
            $j \leftarrow i + \delta$;
            if $j < N$ then
                Swap $order_i$ and $order_j$;
                Construct new graph $A'$ from new order;
                Compute $f' \leftarrow \sum_{(u,v)\in A'} m_{uv}$;
                if $A'$ is acyclic and $f' > f$ then
                    Accept new order and update fitness $f \leftarrow f'$;
until no improvement or timeout;
return $A'$
```

3.3 GRASP Heuristic

GRASP iteratively constructs solutions using a randomized greedy criterion, followed by a local search improvement phase. We implemented two variants: the basic GRASP using only the construction phase, and an extended version where the local search algorithm is applied after each construction. The latter allows refining solutions and escaping local optima.

Cost Function: The cost of adding arc (i, j) is simply:

$$c_{ij} = m_{ij}$$

RCL Threshold: Let C be the sorted list of available arcs by bid. Then the RCL is defined using the threshold:

$$\text{threshold} = m_{\min} + (m_{\max} - m_{\min}) \cdot (1 - \alpha)$$

where:

- $m_{\max}$ and $m_{\min}$ are the maximum and minimum bid values in the candidate list.
- $\alpha \in [0, 1]$ is a parameter controlling greediness.

Stopping Criteria: The GRASP loop continues until one of the following conditions is met:

- A maximum number of consecutive iterations without improvement is reached (set to 50).
- The total execution time exceeds the time limit (set to 60 seconds).

Algorithm 3: GRASP with Local Search

```
Input: Bid matrix $m_{ij}$, parameter $\alpha$, time budget
Output: Best acyclic orientation $A^*$
Initialize $A^* \leftarrow \varnothing$, $f^* \leftarrow 0$;
repeat
    Build RCL from current candidates based on $\alpha$;
    Randomly select $(i, j) \in$ RCL;
    Add $(i, j)$ to $A$ if no cycle is formed;
    Repeat until no more candidates;
    if local search is enabled then
        Apply Local Search to improve $A$ to get $A'$ with fitness $f$;
    else
        $A' \leftarrow A$
    if $f > f^*$ then
        Update $A^* \leftarrow A'$, $f^* \leftarrow f$;
until stopping criteria met;
return $A^*$
```

4 Parameter Tuning and Comparative Results

In all experiments, the bids m_{ij} were randomly generated integers with values in the range $[1, 10]$. This maximum bid value was fixed across all instances to ensure consistent scaling of the objective values and to facilitate fair comparisons across different algorithms.

To keep the report concise, we present only representative plots for tuning and comparison. The structure of the full experimental data—including all objective values, runtimes, algorithm iterations, and α-sweep results—is summarized in Appendix A. The complete dataset is also provided in a shared spreadsheet linked therein.

4.1 Tuning of the GRASP Parameter α

To calibrate the randomized greedy phase of GRASP, we conducted a parameter tuning analysis for the α parameter, which controls the balance between greediness and randomness in candidate selection. Lower α values bias the selection toward the highest bid values, while higher values increase randomness.

To ensure relevant tuning results, we selected medium-to-large problem sizes for this experiment. Specifically, we chose $N = 40$, 43, and 45, where problem size begins to meaningfully impact runtime and solution space. For each value of $\alpha \in \{0.1, 0.2, \ldots, 1.0\}$, we ran the algorithm 3 times and plotted the average objective value along with the standard deviation.

Figures 1 to 3 show the average objective values obtained by GRASP and the corresponding optimality gap with respect to the CPLEX solution.

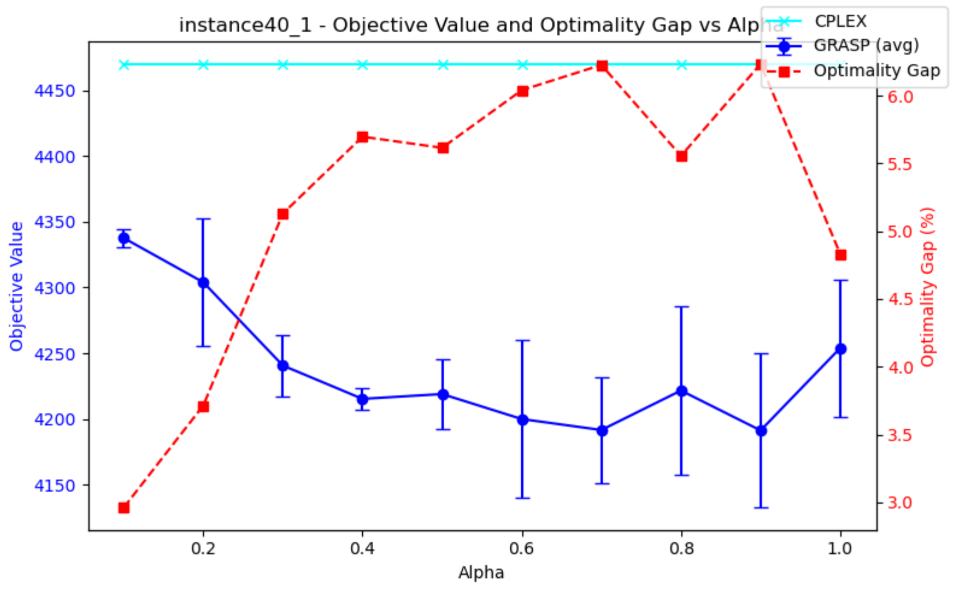

Figure 1: Objective Value and Optimality Gap vs α for instance40_1

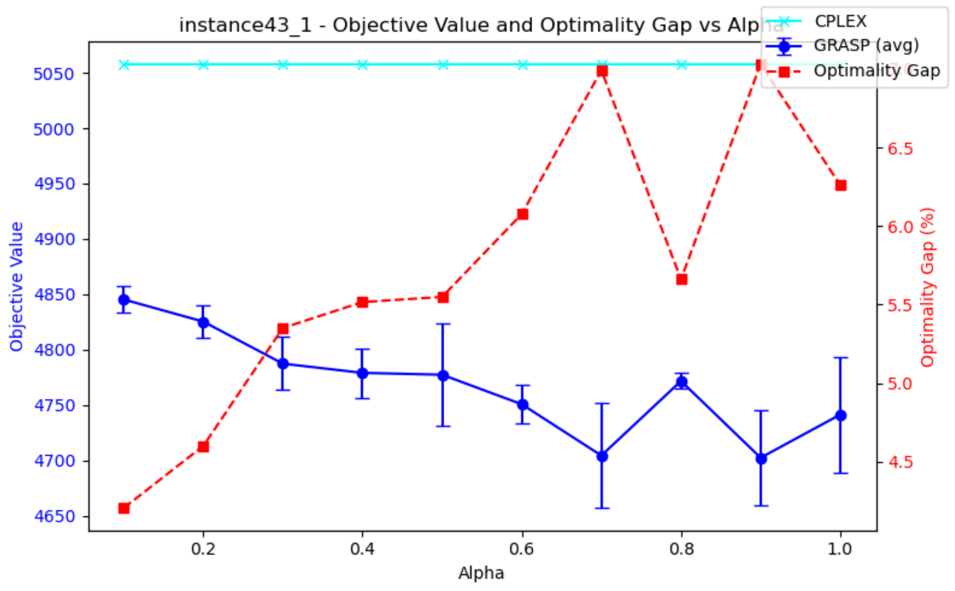

Figure 2: Objective Value and Optimality Gap vs α for instance43_1

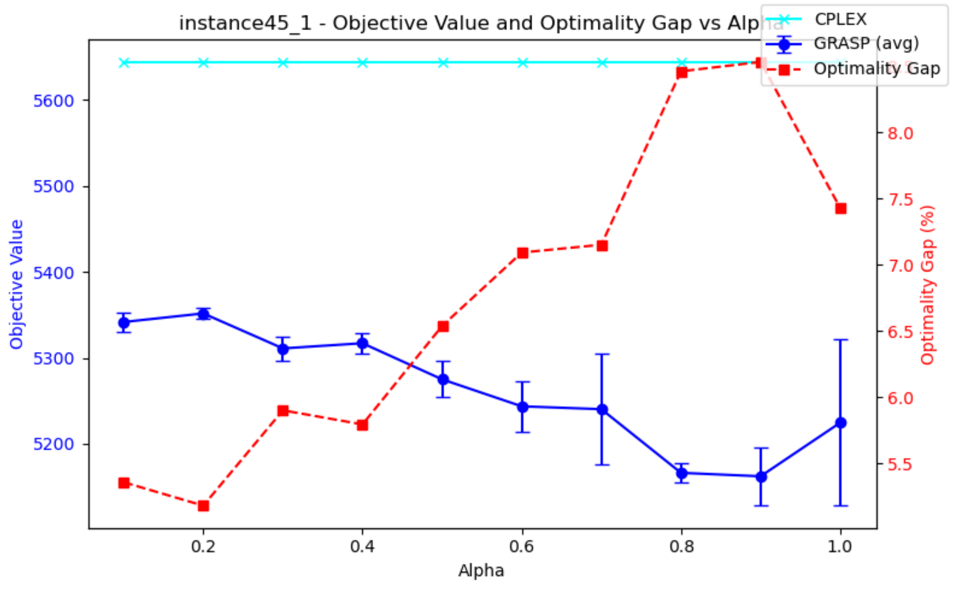

Figure 3: Objective Value and Optimality Gap vs α for instance45_1

From these experiments, we observed that lower α values (around 0.1–0.2) yield better solutions on average, both in terms of absolute objective value and optimality gap. As a result, we used $\alpha = 0.1$ in all subsequent GRASP experiments.

4.2 Elapsed Time Comparison

In our experiments, we observed that CPLEX runtime begins to exceed 1 minute starting from $N = 39$. For this reason, all subsequent experiments were conducted with $N > 39$ to focus on the scalability of the heuristics. Additionally, from $N = 45$ onward, some CPLEX runs reached the time limit, and for $N \geq 48$, timeout occurred consistently in every run.

Figures 4 and 5 compare the average elapsed time (with variance) of all algorithms across increasing problem sizes N. The first figure includes CPLEX, while the second focuses solely on the metaheuristics.

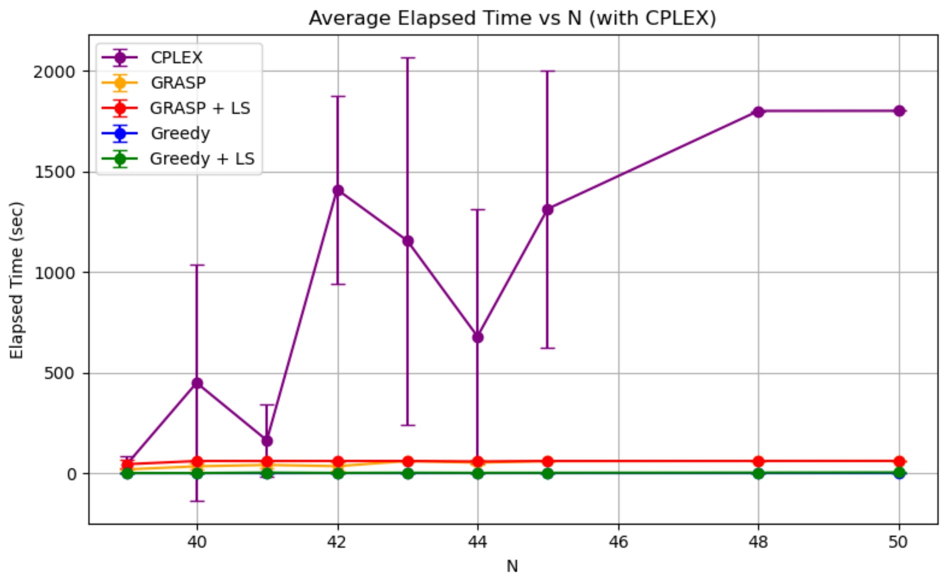

Figure 4: Average Elapsed Time vs N (including CPLEX)

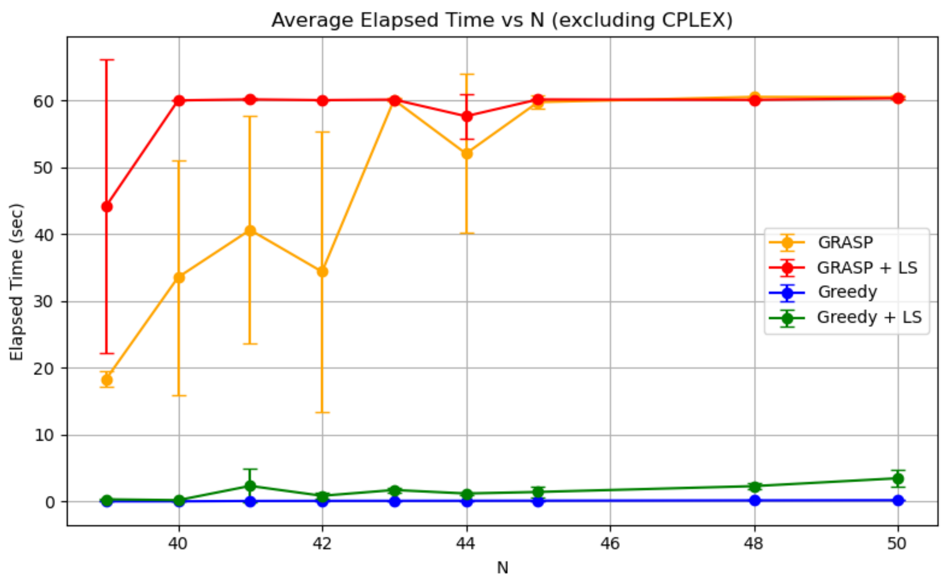

Figure 5: Average Elapsed Time vs N (excluding CPLEX)

CPLEX is significantly more time-consuming, especially as N increases. GRASP and Greedy methods remain efficient and exhibit near-constant runtime. Notably, the addition of local search has only a minor impact on execution time for both GRASP and Greedy variants, demonstrating a good trade-off between quality and performance.

4.3 Objective Value and Optimality Gap Comparison

Figures 6 and 7 present the average objective value and optimality gap (compared to CPLEX) for each algorithm across multiple values of N.

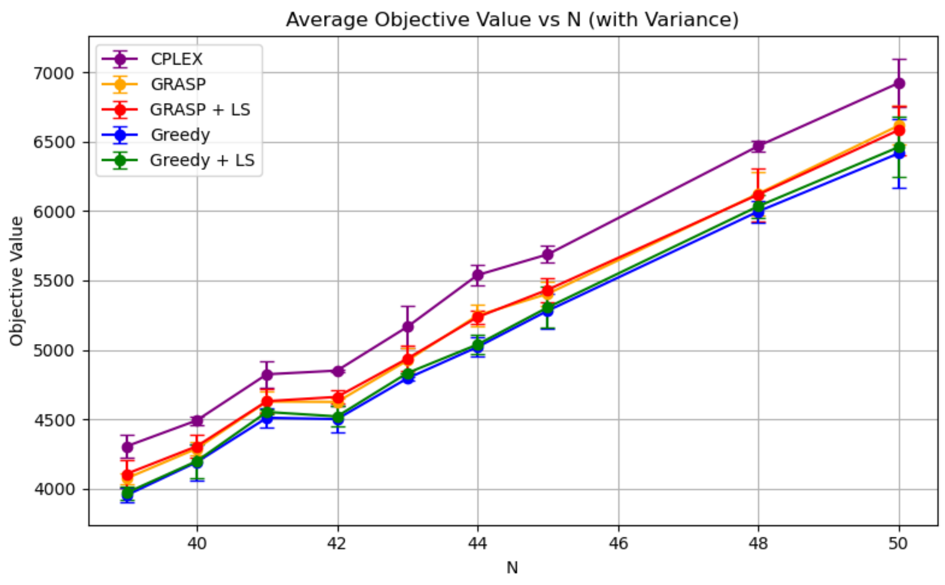

Figure 6: Average Objective Value vs N (with variance)

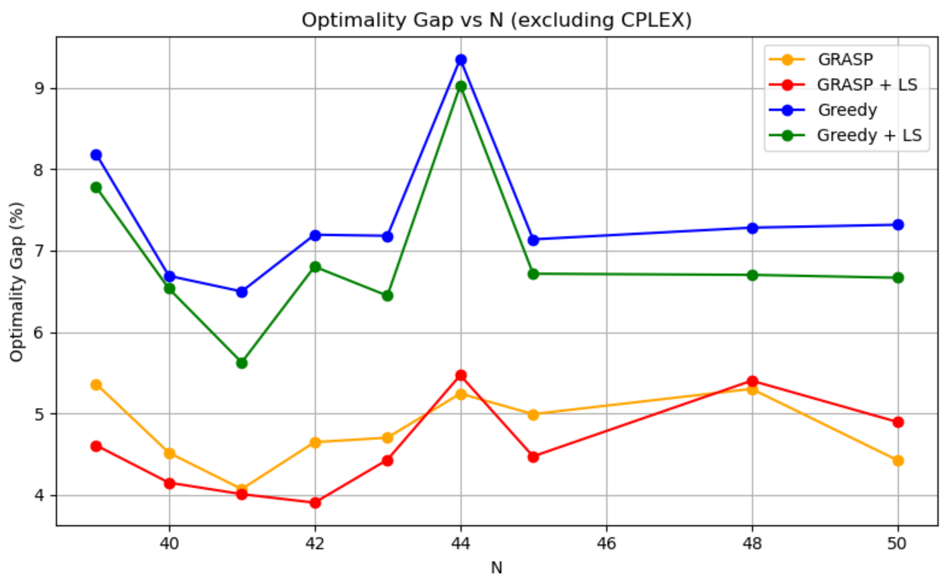

Figure 7: Optimality Gap vs N (excluding CPLEX)

As expected, CPLEX achieves the highest objective values. However, GRASP with Local Search consistently narrows the gap, achieving solutions within 4%–5% of the optimum. Greedy-based approaches are faster but less accurate, with higher gaps, especially without local search.

4.4 Summary

Overall, GRASP + Local Search offers the best compromise between solution quality and execution time. Greedy + Local Search provides a lightweight alternative with reasonable performance, while CPLEX guarantees optimality but becomes impractical for large instances due to frequent timeouts beyond $N = 45$.

Conclusion

In this project, we explored the optimization of a cooperative resource-sharing problem using both exact and heuristic methods. The ILP model, solved via CPLEX, consistently produced the best possible solutions, serving as a reliable benchmark for evaluating other methods. However, as the problem size increased, CPLEX became increasingly time-consuming and eventually impractical due to solver timeouts for instances larger than $N = 45$.

To address scalability, we implemented and evaluated three metaheuristic approaches: Greedy Constructive, Greedy + Local Search, and GRASP with optional local search refinement. Among these, GRASP + Local Search achieved solutions consistently close to the CPLEX optimum—often within 4–5%—while requiring significantly less computational time. Greedy + Local Search also provided a good trade-off between speed and accuracy, especially for mid-sized instances.

Overall, while CPLEX guarantees optimality, heuristic methods proved capable of producing near-optimal solutions with meaningful time savings, making them more suitable for large-scale or time-constrained scenarios. All code, data, and results are available for reproducibility and further exploration.

A Appendix

Table A.1: Objective Value and Runtime Results for All Algorithms (Sample)

N	Instance ID	Algorithm	Objective Value	Elapsed Time (sec)	Status	Iterations
39	1	CPLEX	4249	72.01	1	1
39	1	Greedy	3916	0.053	1	1
39	1	Greedy + LS	3936	0.443	1	1
39	1	GRASP	4047	17.533	1	100
39	1	GRASP + LS	4038	28.647	1	57
39	2	CPLEX	4362	10.61	1	1
...	...	...	...	...	...	...

Table A.2: GRASP α Tuning Results (Sample for instance40_1)

Instance	α	Exec1 (obj)	Exec2 (obj)	Exec3 (obj)	GRASP (avg)
instance40_1	0.1	4345	4336	4332	4337.67
instance40_1	0.2	4336	4328	4248	4304.00
instance40_1	0.3	4238	4265	4219	4240.67
instance40_1	0.4	4225	4209	4212	4215.33
...	...	...	...	...	...

Full results (all instances, alpha tuning, and timing data) are available at:

`Open Full Experimental Results Spreadsheet`

Source code and implementation details are available at:

`View GitHub Repository: github.com/ezgisenak/AMMM_Project`