Departament d'Arquitectura de Computadors
UNIVERSITAT POLITÈCNICA
DE CATALUNYA

AMMM Course Project

Optimal Priority Assignment in Cooperative GPU Scheduling

Ezgi Sena Karabacak
Davide Lamagna

May 25, 2025

Problem Overview

- A cooperative of *N* users shares a GPU
- Users bid to gain priority over others on overlapping days
- **Objective:** Maximize collected bids while avoiding cycles (deadlocks)
- **Key constraint:** Resulting priority graph must be **acyclic**

Formal Problem Definition

Inputs:

- Number of members *N*
- Bid matrix m_{ij}
- Bids are only meaningful if members i and j have overlapping requests.

Output:

- A directed acyclic graph indicating priority relations

Goal:

- Maximize total bid value without forming cycles

ILP Model

Decision variables:

- $x_{ij} \in \{0, 1\} \rightarrow$ 1 if member i has priority over j.
- u_i $\in \{1, 2, ..., N\} \rightarrow$ topological rank of node i

Objective:

$$\text{Maximize} \sum_{i \neq j} x_{ij} \cdot m_{ij}$$

Constraints:

- No self-priority: $x_{ii} = 0$ for all i
- Acyclicity via topological order:

  If $x_{ij} = 1$, then $u_i < u_j$

  (Enforced with: $u_i + 1 \leq u_j + (1 - x_{ij}) * N$)

Heuristic Algorithms

Why heuristics?

- CPLEX gives optimal results but becomes slow or times out for large instances (N > 45).
- We need **faster**, **scalable** alternatives that still give good-quality solutions.

Implemented Heuristics:

- Greedy Constructive Heuristic
- Greedy + Local Search
- GRASP
- GRASP + Local Search

Goal: Find near-optimal solutions much faster than CPLEX, especially for large problem sizes.

Greedy Constructive Heuristic

- Sort all possible arcs (i, j) by bid value m_{ij} in descending order.
- Start with an empty graph.
- Add each arc one by one if it does not create a cycle.
- Continue until all arcs have been checked.

Advantages:

- Very fast.
- Produces a valid (acyclic) solution.
- Quality depends heavily on early decisions.

Limitation:

- Can miss better configurations due to early greedy choices.

Greedy Pseudocode

Algorithm 1: Greedy Constructive Heuristic

Input: Bid matrix m_{ij}
Output: Acyclic orientation A
Initialize $A \leftarrow \varnothing$;
Add all nodes $i \in \{1, \ldots, N\}$ 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

Local Search Strategy

- Improves the greedy solution by modifying the topological order.
- Swaps each node with nearby nodes (+1, +2, +3) to find better bid values.
- Only keeps swaps that maintain acyclicity and increase total value.
- Uses **Best Improvement**: chooses the best swap per iteration.
- Stops when no more improving moves are found or timeout is reached.

Local Search Pseudocode

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 ← Σ_{(i,j)∈A} m_ij;
repeat
    for each i ∈ {1, ..., N − 1} do
        for δ ∈ {1, 2, 3} do
            j ← i + δ;
            if j < N then
                Swap order_i and order_j;
                Construct new graph A' from new order;
                Compute f' ← Σ_{(u,v)∈A'} m_uv;
                if A' is acyclic and f' > f then
                    Accept new order and update fitness f ← f';
until no improvement or timeout;
return A'
```

GRASP Heuristic

- Builds solutions iteratively using:
  - A Restricted Candidate List (RCL) based on bid values.
  - Random selection from the RCL (controlled by parameter **α**).
- After construction, applies Local Search for refinement.

Key Benefit:

- Combines exploration (randomness) and exploitation (local search).
- Avoids greedy traps and finds better-quality solutions.

Tuned Parameter:

α = 0.1 gave the best performance in our experiments.

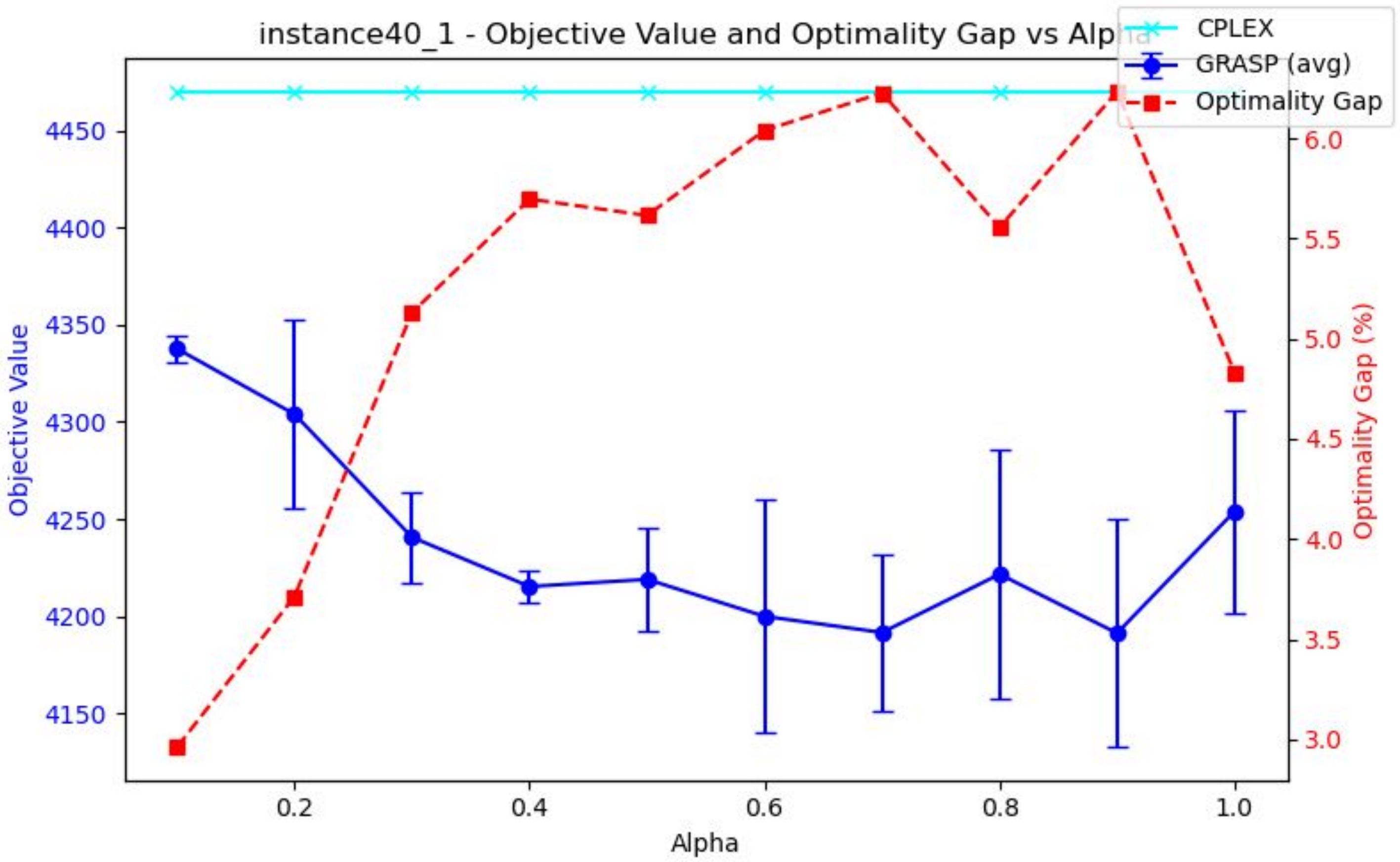
GRASP Pseudocode

Algorithm 3: GRASP with Local Search

```
Input: Bid matrix m_ij, parameter α, time budget
Output: Best acyclic orientation A*
Initialize A* ← ∅, f* ← 0;
repeat
    Build RCL from current candidates based on α;
    Randomly select (i, j) ∈ 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′ ← A
    if f > f* then
        Update A* ← A′, f* ← f;
until stopping criteria met;
return A*
```

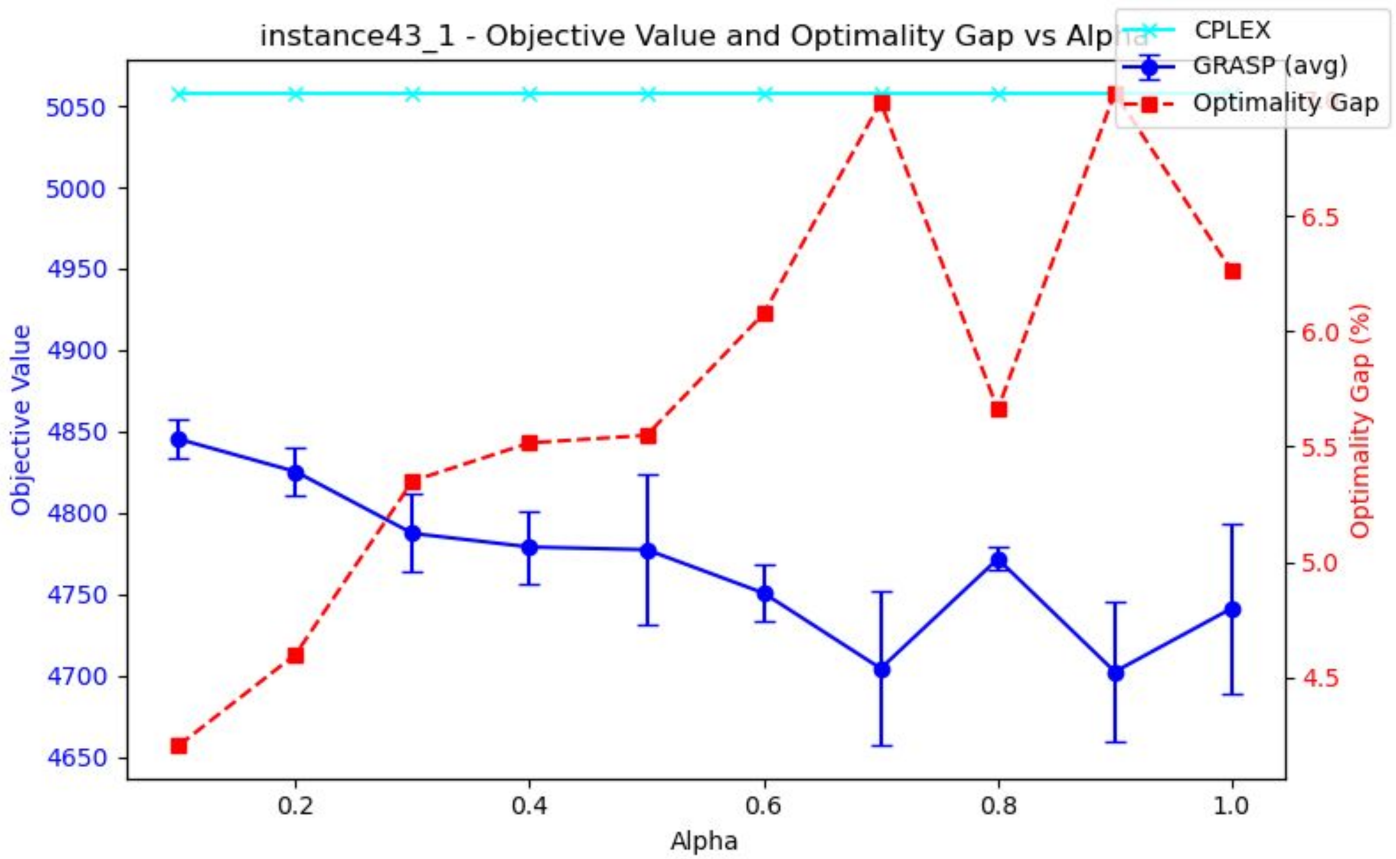
Alpha Tuning

N=40



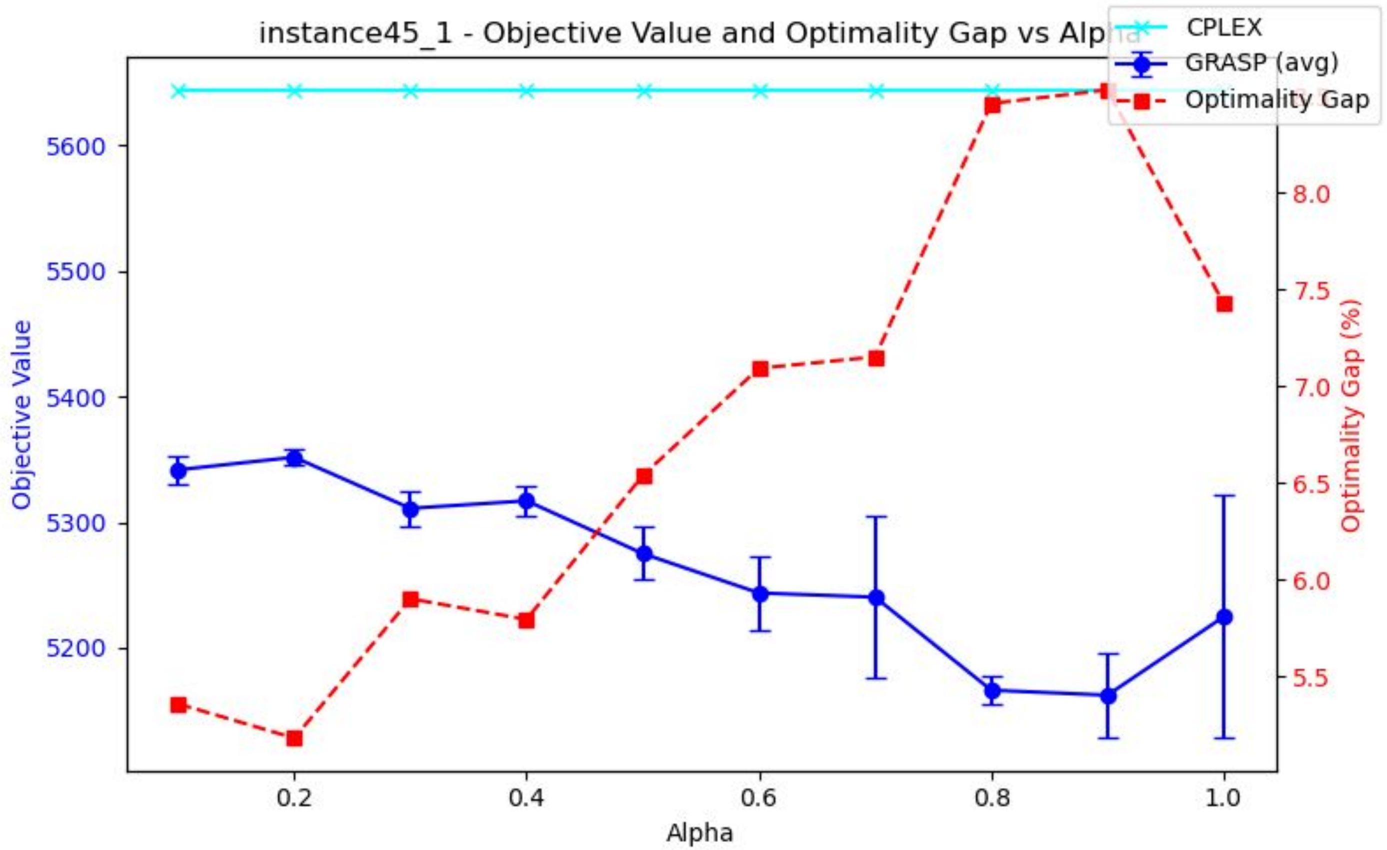
Alpha Tuning

N=43



Alpha Tuning

N=45



Experimental Setup

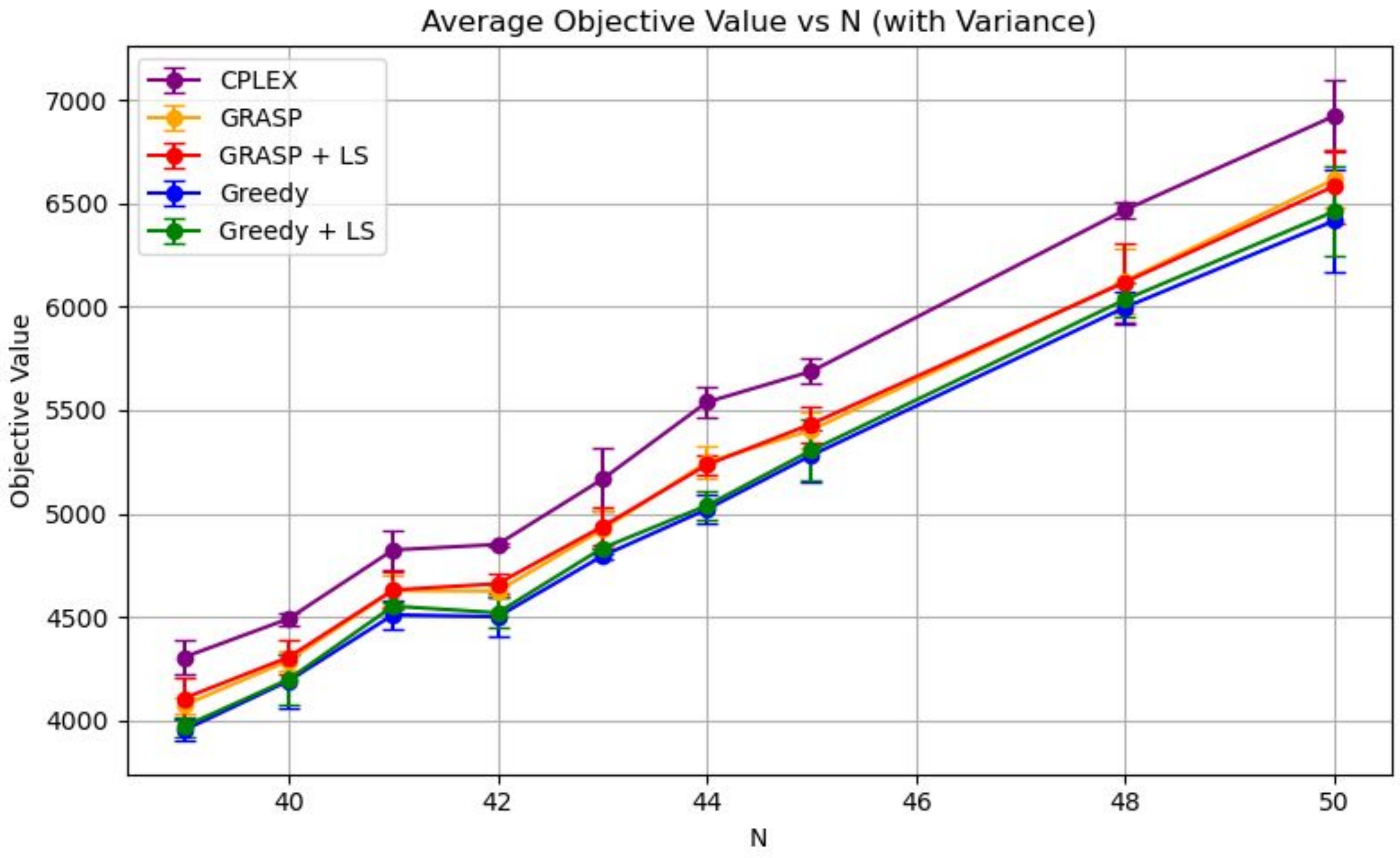
- Instance sizes tested: **N = 40** to **N = 50**
- Bid values: Random integers in the range [1, 10]
- GRASP tuning:
  - $\alpha \in \{0.1, 0.2, ..., 1.0\}$
  - Each value tested 3 times
  - Selected $\alpha = 0.1$ based on best average performance
- CPLEX timeout: 60 seconds
- Starts to fail beyond N = 45

All results include:

- Objective value
- Elapsed time
- Iterations (for heuristics)

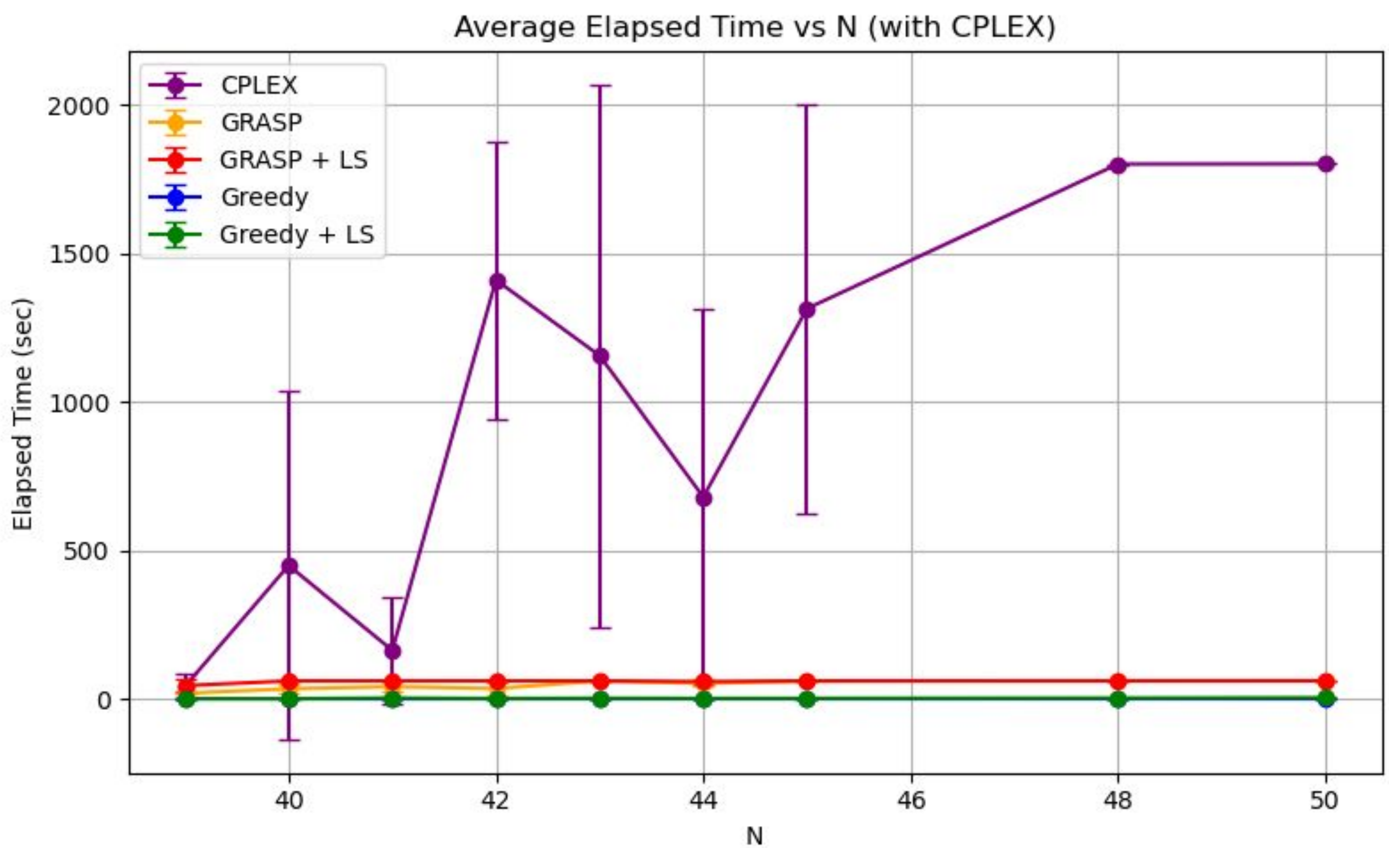
Results

Objective value vs N



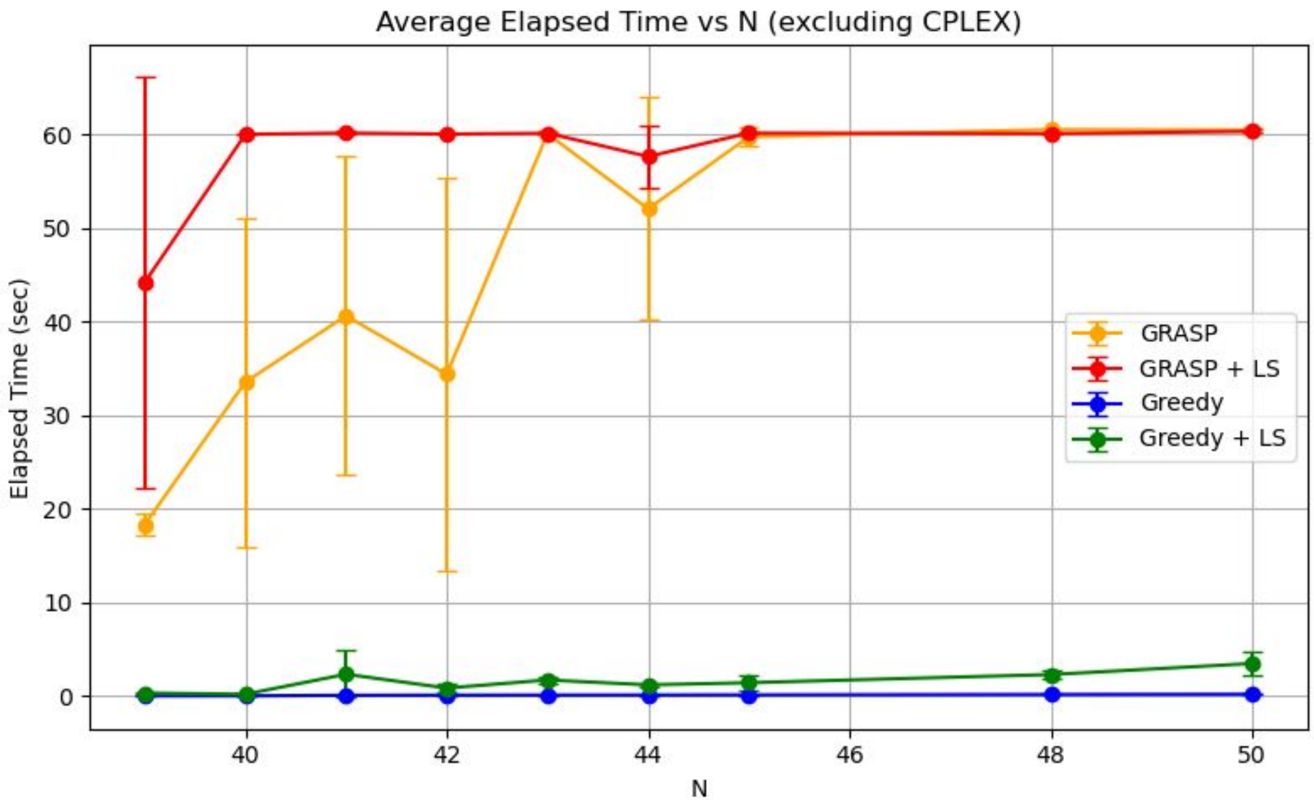
Results

Elapsed Time vs N



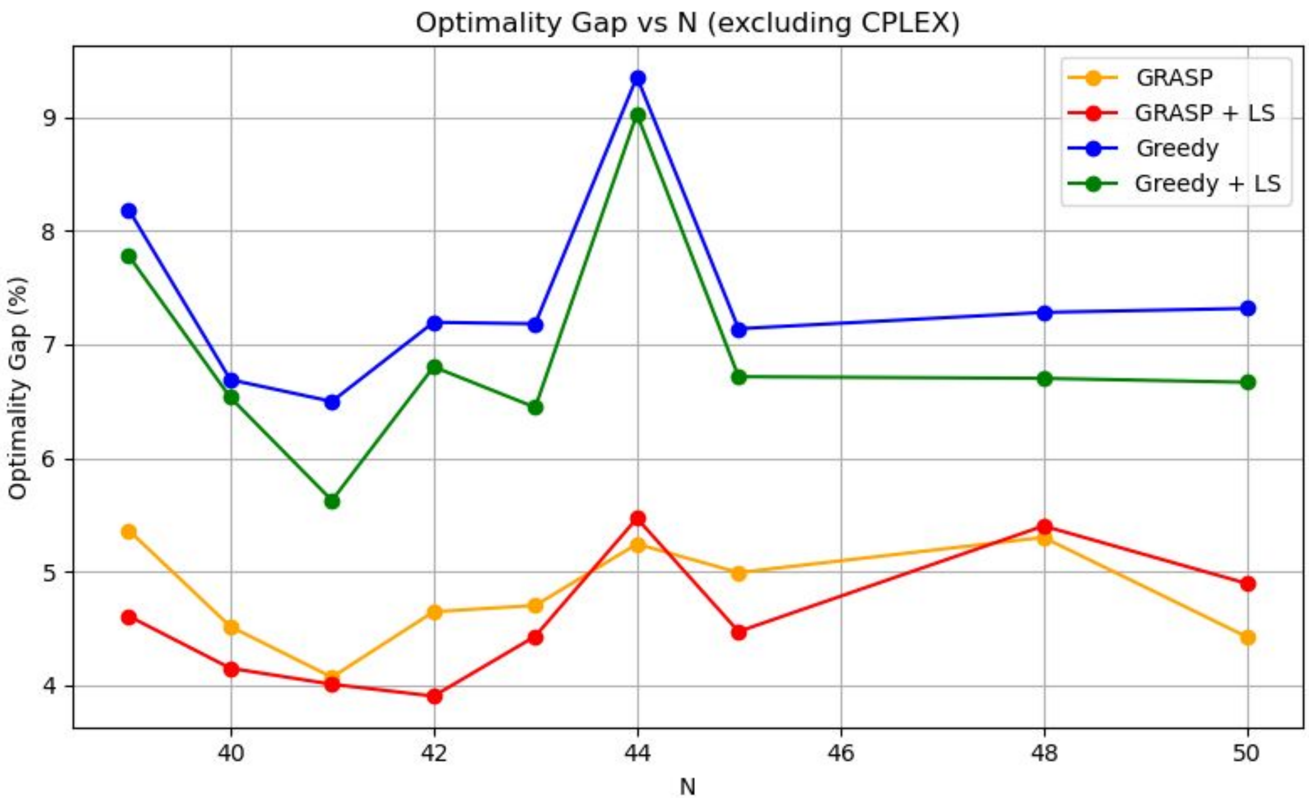
Results

Elapsed Time vs N



Results

Optimality Gap vs N



Conclusion

- CPLEX gives the best solutions but becomes impractical for large instances
- GRASP + Local Search offers the best trade-off:
  - Near-optimal solutions (within 4–5% of CPLEX)
  - Much faster and scalable
- Greedy + Local Search is a simple and fast alternative with reasonable quality.
- Heuristic methods are effective for large-scale or time-sensitive problems.

Thank you :)

Any questions?