AMMM - Final Project

Hugo SINGEZ - Patrick CERKA

30/05/2025

Summary

- Formalization of the problem
- Integer linear programming model
- Greedy algorithm
- Local Search algorithm
- Grasp algorithm
- Comparison and analysis of the results

Formalization of the Problem

A cooperation of N player want to use a same computer. In case of conflict, they all bet money on other members in order to have priority.

Input:

- Number of member N
- Matrix of bets m

Output

- Total income z
- Matrix of priorities

Integer Linear Programming Model

Variables:

- BOOLEAN matrix *x* of priorities of size *n*n*
- INTEGER vector of rank *r* of size *n*
- INTEGER *z* of revenue

Objective function:

- **Maximize z**, the income for the cooperation

Constraints:

- z is the total income for the cooperation *(1)*
- Only one of 2 member can have priority *(2)*
- The rank of each member is lower than *n (3)*
- The graph is acyclic (if *x_ij = 1* then *ri* < *rj*) *(4)*

$$z \leq \sum_{i=1}^{N} \sum_{j=1}^{N} m_{ij} \cdot x_{ij} \qquad (1)$$

$$x_{i,j} + x_{j,i} \leq 1 \qquad (2)$$

$$r_i \leq N \qquad (3)$$

$$r_i - r_j + 1 \leq (1 - x_{i,j}) * N \qquad (4)$$

Greedy Algorithm

- Compute the outgoing arcs sum for each node;
- Sort nodes in descending order of their scores;
- Build a DAG by connecting earlier to later nodes in the sorted order;
- In this way, we avoid deadlocks by creating a topological order.

Greedy Algorithm: pseudocode

Algorithm 1: Build Acyclic Priority Matrix

```
Input: Matrix m, Integer N
Output: Priority matrix x
Initialize empty list node_score
for i ← 0 to N − 1 do
    sum_row ← 0
    for j ← 0 to N − 1 do
        sum_row ← sum_row + m[i][j]
    end
    Append (sum_row, i) to node_score
end
Sort node_score in descending order by sum_row
Initialize array order of size N
for i ← 0 to N − 1 do
    order[i] ← node_score[i].index
end
Initialize x as an N × N zero matrix
for i ← 0 to N − 1 do
    for j ← i + 1 to N − 1 do
        u ← order[i]
        v ← order[j]
        x[u][v] ← 1
    end
end
Print matrix x and score sum of m[i][j] where x[i][j] = 1
local_search_order(order, m)
return y
```

Local Search Algorithm

- Swap adjacent nodes if it improves total score (m[order[i]][order[j]]);
- Repeat until no improvement is possible;
- Construct new matrix y from improved order.

Local Search Algorithm: pseudocode

Algorithm 3: Local Search on Order Array

```
Input: Order array order, matrix m
Output: Improved order with local search
N ← length of order
improved ← true
while improved do
    improved ← false
    for i ← 0 to N − 2 do
        swap(order[i], order[i+1])                      // Swap adjacent elements
        new_score ← evaluate_order(order, m)
        swap(order[i], order[i+1])                      // Revert swap
        current_score ← evaluate_order(order, m)
        if new_score > current_score then
            swap(order[i], order[i+1])                  // Accept improvement
            improved ← true
        end
    end
end
Printing logic...
```

Support function: Evaluate

Algorithm 2: Evaluate Order Score

Input: Order array *order*, Matrix m

Output: Score integer

$N \leftarrow$ length of *order*

$score \leftarrow 0$

for $i \leftarrow 0$ **to** $N-1$ **do**

    for $j \leftarrow i+1$ **to** $N-1$ **do**

        $score \leftarrow score + m[order[i]][order[j]]$

    end

end

return *score*

Grasp Algorithm

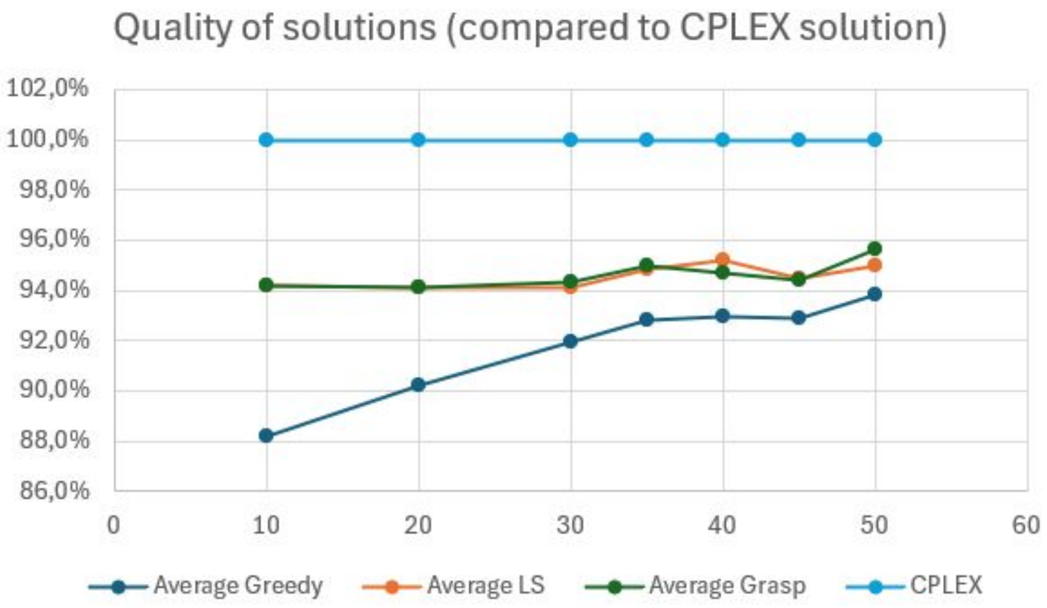
- Score each node (sum of outgoing edges).
- Create a Restricted Candidate List (RCL) based on parameter α.
- Randomly select from RCL to form the order.
- Build DAG from constructed order;
- Apply Local search on the GRASP solution.

Grasp Algorithm: pseudocode

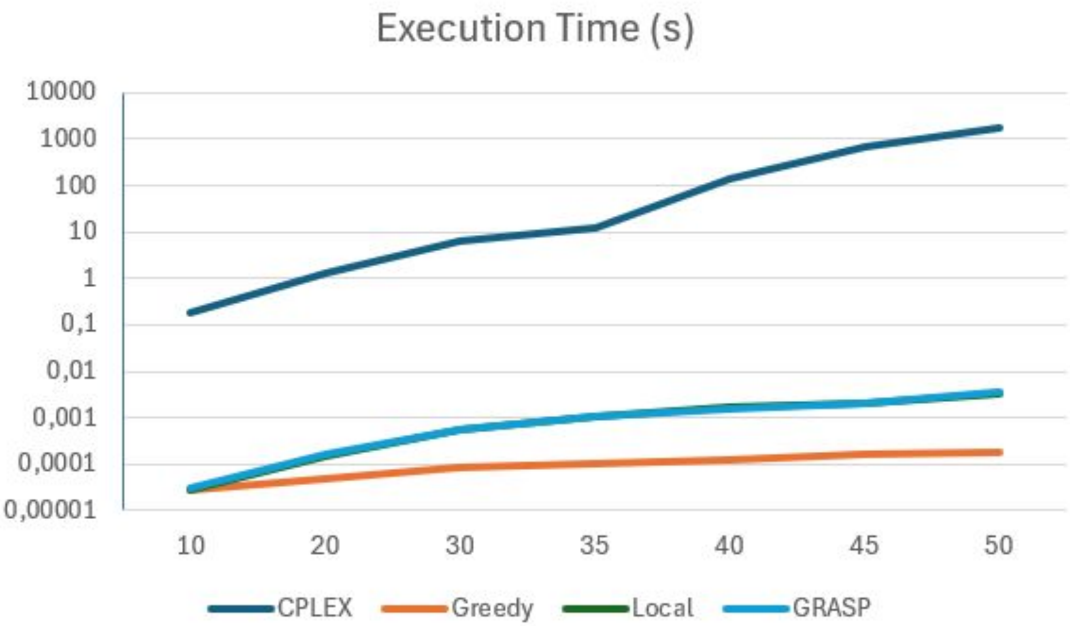
Algorithm 4: GRASP Construction of Priority Matrix

```
Input: Matrix m, Parameter α
Output: Priority matrix y
N ← size of m
Initialize empty list order
Initialize boolean array chosen of size N with false
Initialize array scores of size N
for i ← 0 to N − 1 do
    scores[i] ← Σ_{j=0}^{N−1} m[i][j]
end
while size of order < N do
    Initialize empty list candidates
    for i ← 0 to N − 1 do
        if chosen[i] = false then
            Append (scores[i], i) to candidates
        end
    end
    Sort candidates in descending order by score
    rcl_size ← max(1, ⌊α × size of candidates⌋)
    Randomly pick index pick in [0, rcl_size − 1]
    selected ← candidates[pick].index
    Append selected to order
    chosen[selected] ← true
end
Initialize y as an N × N zero matrix
for i ← 0 to N − 1 do
    for j ← i + 1 to N − 1 do
        u ← order[i]
        v ← order[j]
        y[u][v] ← 1
    end
end
Print matrix y and score sum of m[i][j] where y[i][j] = 1
local_search_order(order, m)
return y
```

Comparison of the quality of the solution



Comparison of the execution time



Questions ?