Facultat d'Informàtica de Barcelona

Master in Innovation and Research in Informatics

High Performance Computing

Algorithmic Methods for Mathematical Models

COURSE PROJECT

Name	Email
Can Beydogan	can.beydogan@estudiantat.upc.edu
Sefa Böyükdikmen	sefa.boyukdikmen@estudiantat.upc.edu

Fall 24

Barcelona, December 10, 2024

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Contents

1 Problem Statement

The goal of this problem is to select faculty members from the university to form a committee to update the curriculum of the computer science degree. The selection process must match department-specific requirements and ensure that the selected members are compatible, according to a compatibility matrix. Furthermore, the average compatibility among selected members should be maximized while satisfying the following constraints.

- Each department p must have exactly n[p] participants selected.
- The compatibility between two selected participants cannot be zero.
- If two selected members have poor compatibility ($0 < m[i][j] < 0.15$), a third participant must be selected who has high compatibility ($m[i][k] > 0.85$) and $m[k][j] > 0.85$) with both of them.

1.1 Inputs

- D is the number of departments in the faculty.
- n[1..D] is the array of size D, department p has exactly a number n[p] of participants in the commission.
- N is the total number of faculty members.
- d[1..N] the department of i will be denoted by d[i].
- m[1..N][1..N] is a N x N matrix where each element m[i][j] is the compatibility between members i and j. The diagonal of the compatibility matrix consists of 1's.

1.2 Outputs

The output includes :

- The solution (if one exists) with optimal objective value.
- The selection of faculty members, represented by the array $x[1..N]$, where:

$$x[i] = \begin{cases} 1 & \text{indicates that faculty member } i \text{ is selected for the commission,} \\ 0 & \text{indicates that faculty member } i \text{ is not selected.} \end{cases}$$

2 Integer Linear Programming Model

The problem is described as an Integer Linear Programming (ILP) model in this section. Depending on a number of limitations, such as departmental needs, compatibility, and participant conflicts, the objective of this approach is to choose a subset of faculty members for a commission while optimizing their overall compatibility.
Decision variables, an objective function, and a collection of constraints that control the selection process are all included in this ILP model.

2.1 Decision Variable

The decision variables x[i] represents whether a faculty member i is in the commission or not. Decision variable x[i] is 1 if member i is selected, otherwise 0.

2.2 Objective Function

m[i][j] represents the compatibility between members i and j. The objective is to maximize the average compatibility among all pairs of selected participants. The objective function calculation formula is as follows:

$$\text{maximize}\left(\frac{1}{\text{numberOfPairs}}\sum_{i=1}^{N}\sum_{j=i+1}^{N} m[i][j]\cdot x[i]\cdot x[j]\right)$$

Where:

$$x[i] = \begin{cases} 1 & \text{if faculty member } i \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$$

The number of pairs is calculated as:

$$\text{number of pairs} = \frac{\text{number of selected participants} \times (\text{number of selected participants} - 1)}{2}$$

The goal is to select faculty members such that the average compatibility among the selected participants is maximized.

2.3 Constraints

The model includes the following constraints to ensure that the selection meets all requirements:

1. **Department Participation:** Each department p must contribute exactly $n[p]$ participants:

$$\sum_{\substack{i=1 \\ d[i]=p}}^{N} x[i] = n[p], \quad \forall p \in \{1,\ldots,D\}, \quad i \in \mathbb{Z}$$

2. **Zero Compatibility:** Faculty members with zero compatibility cannot both be selected:

$$x[i] + x[j] \leq 1, \quad \forall i, j \in \{1, \ldots, N\} \text{ such that } m[i][j] = 0$$

3. **Conflict Mediation:** If the compatibility between two selected members i and j is poor ($0 < m[i][j] < 0.15$), there must exist a third participant k such that $m[i][k] > 0.85$ and $m[k][j] > 0.85$:

$$x[i] + x[j] \leq 1 + \sum_{\substack{k=1 \\ m[i][k]>0.85, m[k][j]>0.85}}^{N} x[k], \quad i, j, k \in \{1, \ldots, N\}$$

3 Heuristic Algorithms

Heuristic algorithms offer an efficient way to solve combinatorial optimization problems like this problem, where finding the optimal solution using Integer Linear Programming which might be too time-consuming. The goal of the heuristics algorithms is to generate high-quality solutions in a more time-efficient manner, especially as the problem size increases. This is achieved by applying constructive and iterative techniques that gradually improve the solution.
In this section, three meta-heuristic algorithms are explained. These are Greedy construction, a greedy constructive + a local search procedure and GRASP (Greedy Randomized Adaptive Search Procedure). These methods develop a comprehensive approach that emphasizes both efficiency and solution quality to address the problem of assigning participants to departments under certain constraints.

First of all, the algorithms that are commonly used in these heuristic algorithms are as follows: Feasibility Check and Calculate Objective Function.

As shown in the Feasibility Check algorithm 1, it ensures that the candidate solution satisfies the constraints of the problem. It verifies that compatibility conditions between members are satisfied. If any constraint is violated, the solution is counted infeasible. This step is crucial for filtering out invalid solutions early, ensuring only feasible ones proceed to optimization.

Algorithm 1 Feasibility Check

Input: $D, n, N, d, m,$ solution
Output: True or False

```
if | solution | ≠ ∑ n then return False
for each member in solution do
    if not ( ∀other ∈ solution, m[member][other] > 0 or ∀other ∈ solution, (m[member][other] ≥ 0.15 or ∃k ∈ solution, m[member][k] > 0.85 ∧ m[k][other] > 0.85) ) then
        return False
return True
```

As shown in Calculate Objective Algorithm 2, it computes the average compatibility of the compatibility matrix values $m[i][j]$ for all pairs $i < j$ where i

and j are faculty members from the `selected solution` . The result is normalized by dividing the total sum by the number of pairs (p). This process calculates an objective function that represents the average of the matrix values for all unique pairs from the selected indices.

Algorithm 2 Calculate Objective

Input: m, selected
Output: objective

$n \leftarrow |\text{selected}|$
$p \leftarrow \frac{n \cdot (n-1)}{2}$
$sum \leftarrow 0$
for all $i \in$ selected **do**
    for all $j \in$ selected **do**
        if $i < j$ **then**
            $sum \leftarrow sum + m[i][j]$
$objective \leftarrow \frac{sum}{p}$
return $objective$

3.1 Greedy Constructive Algorithm

This algorithm uses a greedy approach to construct a partial solution by iteratively selecting the best option based on compatibility and departmental constraints. Greedy algorithms make locally optimal choices at each step with the goal of reaching a global optimum solution.

The algorithm 3 starts with an empty solution and sorts members by compatibility in descending order. Thus, the best options are selected in order. After that, it checks if adding a member respects the departmental participant limit and compatibility conditions. If these conditions are satisfied, the member is added to the partial solution.At the end of the algorithm, if the required number of participant is equal to size of the partial solution,it return partial solution. Otherwise, it returns "INFEASIBLE".

Algorithm 3 Greedy Construction Algorithm

Input: D, n, N, d, m
Output: partial_solution

partial_solution $\leftarrow \emptyset$
dep_participantN[d] $\leftarrow 0, \forall d \in \{1, \ldots, D\}$
compatibilities $\leftarrow$ sort($N, \sum m[i][j]$, DESC)
for member in compatibilities **do**
    department $\leftarrow d$[member]
    if dep_participantN[department] < n[department - 1] **then**
        if $\forall$other $\in$ partial_solution, m[member][other] > 0 and
        $\forall$other $\in$ partial_solution, (m[member][other] ≥ 0.15 or
        $\exists k \in$ partial_solution, m[member][k] $> 0.85 \wedge m[k]$[other] > 0.85) **then**
            partial_solution $\leftarrow$ partial_solution $\cup$ {member}
            dep_participantN[department] $\leftarrow$ dep_participantN[department]+1
    if | partial_solution | $= \sum n$ **then**
        break
if | partial_solution | $< \sum n$ **then return** INFEASIBLE
return partial_solution

3.2 Greedy Constructive and Local Search Procedure

The algorithm 4 is a local search method that uses the best improvement search strategy policy with a reassignment neighborhood (one element exchange) to iteratively improve a given solution which is obtained from the greedy constructive algorithm. In best-improving strategy, all neighbors are investigated, and the current solution is replaced by the best neighbor. Reassignment neighborhood, or one-element exchange, is a local search method where one member in a solution is swapped with another member that is not in the solution to find nearby solutions.

This algorithm starts with an initial solution and tries to find a better solution by swapping one element at a time, within the solution. For each swap, the algorithm checks if the new solution is feasible 1 and improves the objective function 2. If there is an improvement, the algorithm keeps the new solution and continues searching for further improvements. This continues until no further improvements can be made. Then, the algorithm returns the best solution and its objective value.

Algorithm 4 Local Search Algorithm

Input: $D, n, N, d, m,$ solution
Output: best_solution, best_objective

```
best_solution ← solution
best_objective ← calculate_objective(m, best_solution)
is_improved ← True
while is_improved do
    is_improved ← False
    for each i in best_solution do
        current_member ← best_solution[i]
        current_department ← d[current_member]
        for each j in {0, ..., N − 1} do
            if j in best_solution or d[j] ≠ current_department then
                continue
            new_solution ← best_solution
            new_solution[i] ← j
            if is_feasible(D, n, N, d, m, new_solution) then
                new_objective ← calculate_objective(m, new_solution)
                if new_objective > best_objective then
                    best_solution ← new_solution
                    best_objective ← new_objective
                    is_improved ← True
                    break
return best_solution, best_objective
```

3.3 GRASP

GRASP (Greedy Randomized Adaptive Search Procedure) is a meta-heuristic algorithm designed to solve combinatorial optimization problems. GRASP works well in situations where exact optimization methods are too slow or difficult because the solution space is too large or complex. There are two phase in the GRASP algorithm. First one is Greedy Construction with GRASP and Local Search.

Algorithm 5 GRASP Main Function

Input: D, n, N, d, m, iterations, α
Output: best_solution, best_objective

$best\_solution \leftarrow$ None
$best\_objective \leftarrow -\infty$
for $k = 1$ **to** iterations **do**
    $initial\_solution \leftarrow$ GreedyConstructionGRASP($D, n, N, d, m, 0$ **if** $i = 1$ **else** $alpha$))
    if $initial\_solution =$ INFEASIBLE **then**
        continue
    ($local\_best\_sol, local\_best\_obj$) $\leftarrow$ LocalSearch($D, n, N, d, m, initial\_sol$)
    if $local\_best\_obj > best\_objective$ **then**
        $best\_sol \leftarrow local\_best\_sol$
        $best\_obj \leftarrow local\_best\_obj$
return ($best\_solution, best\_objective$)

The GRASP function 5 runs for a number of iterations. Firstly, it found a partial solution using a modified greedy construction approach, then if the solution is feasible, it improves this solution using a local search algorithm 4. In the local search algorithm, the cost function is calculated by using calculate objective function 2. The Greedy Construction with GRASP is called in each iteration to find a partial solution by selecting members based on their compatibility scores. This selection process is controlled by a Restricted Candidate List (RCL) 1. The parameter α determines the level of randomness in the RCL. The range of α value is between 0 and 1. The larger α value adds more randomness by considering a wider range of candidates while the smaller α value makes the selection more predictable .The list contains members whose compatibilities are within a specific range, and one member is then selected randomly from the RCL. After that, it checks if selected member respects the departmental participant limit and compatibility conditions. If these conditions are satisfied, the member is added to the partial solution. At the end of the algorithm, if the required number of participant is equal to size of the partial solution,it return partial solution. Otherwise, it returns "INFEASIBLE".

```
RCL = {member | member >= max(compatibility) - alpha * (max(compatibility) -
↪  min(compatibility))}
```

Code 1: The Formula of Restricted Candidate List

Algorithm 6 Greedy Construction with GRASP

Input: D, n, N, d, m, α
Output: partial_solution or **None**

partial_solution $\leftarrow \emptyset$
dep_participantN[d] $\leftarrow 0, \forall d \in \{1, \ldots, D\}$
compatibilities[i] $\leftarrow \sum m[i], \forall i \in \{1, \ldots, N\}$
candidates $\leftarrow$ sort(N, compatibilities[i], DESC)
while | partial_solution |< n **and** candidates $\neq \emptyset$ **do**
    max_compatibility $\leftarrow$ compatibilities[candidates[0]]
    min_compatibility $\leftarrow$ compatibilities[candidates[-1]]
    RCL $\leftarrow$ {member $\in$ candidates | compatibilities[member] $\geq$ max_compatibility $- \alpha \cdot$ (max_compatibility $-$ min_compatibility)}
    selected_member $\leftarrow$ random.choice(RCL)
    dept $\leftarrow d$[selected_member]
    if dep_participantN[dept] $< n$[dept $- 1$] **then**
        if $\forall$other $\in$ partial_solution, m[member][other] > 0 and $\forall$other $\in$ partial_solution, (m[member][other] ≥ 0.15 or $\exists k \in$ partial_solution, m[member][k] $> 0.85 \wedge m[k]$[other] > 0.85) **then**
            partial_solution $\leftarrow$ partial_solution $\cup$ {member}
            dep_participantN[dept] $\leftarrow$ dep_participantN[dept] $+1$
    candidates $\leftarrow$ candidates $-$ {selected_member}
if | partial_solution |< $\sum n$ **then**
    return INFEASIBLE
else
    return partial_solution

4 Tuning of Parameters and Instance Generation

In this section, the implementation of instance generator and tuning the α parameter of the GRASP constructive phase are presented.

4.1 Instance Generator

The purpose of instance generator is creating randomized instances for problem-solving purposes. It generates structured instances representing a configuration of departments, members, and their compatibilities. These instances are used as input to solve our problem statement. The generator is configurable. Config class is used to adjust parameters such as the range of the number of departments and members, as well as output settings. These parameters allow for flexibility in defining the scope and complexity of the generated instances. In this way, the instance generator is adaptable to various experimental scenarios. Moreover, the validation mechanism is used in order to ensure the correctness of the configurations. This validation mechanism checks the presence and validity of all required parameters.

4.2 Tuning the α parameter

The parameter α determines the level of randomness. The range of α value is between 0 and 1. The larger α value adds more randomness by considering a wider range of candidates while the smaller α value makes the selection more predictable. The GRASP constructive phase 6 includes a configurable alpha parameter. This parameter can be adjusted via a configuration file. It can either run with a fixed default α value or be tuned by incrementally testing values within a specific range. If tuning process is selected, the optimal α value can be found using a set of randomly generated instances of large enough size.

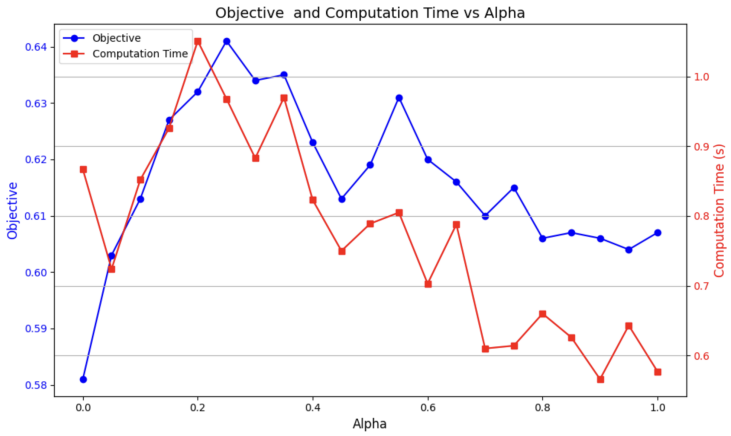

Figure 1: Objectibe and Computation Time vs Alpha

The graph 1 illustrates the best objective values (blue line) and computation times (red line) for each alpha value. It highlights how the objective improves with different alpha values, peaking at $\alpha = 0.25$. The computation time varies slightly but remains relatively stable across all alpha values. The data was generated using the `project.tune.dat` input file, and the results were saved in the `output.grasp.tune.sol` file. As the number of faculty members increases, the computation time of CPLEX becomes excessive. Because of this, a comparison between CPLEX and other methods is not possible when number of faculty member is greater than seventy. Therefore, we select instances which are less than seventy. When using these inputs, there is no significant difference between the alpha values. As a result, we select another large-size randomly generated data instance to calculate the alpha value. This allowed for a better observation of the effect of alpha's change on the objective and computation time

5 Performance

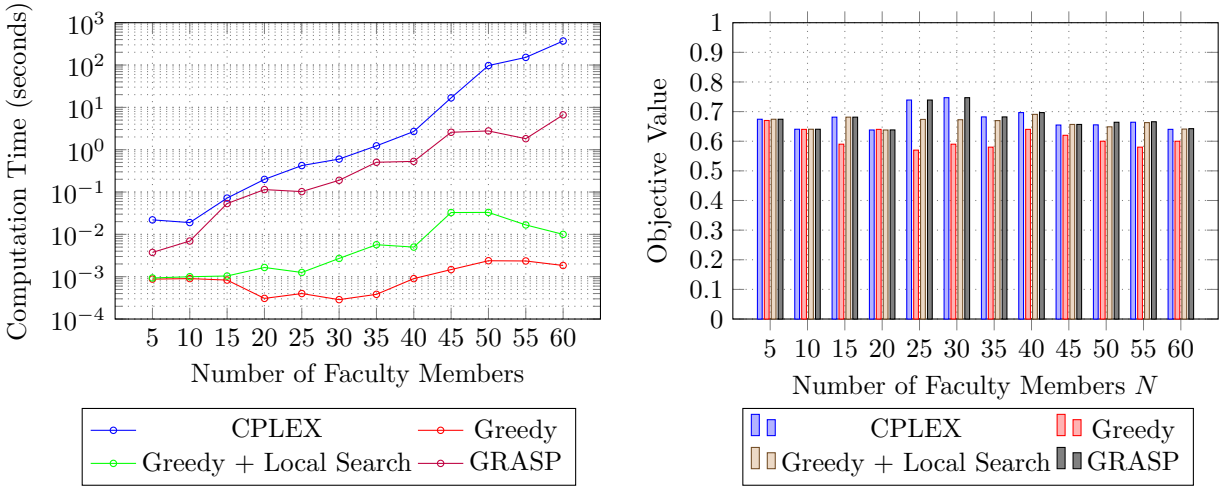

Figure 2: Performance and Objective Result Comparison

This figure 2 show a comparison of four methods based on their computation time and the objective values of their solutions for different numbers of faculty members (N). These methods are CPLEX, Greedy, Greedy + Local Search, and GRASP.

The first graph illustrates computation time. CPLEX takes much longer as the number of faculty members increases since CPLEX solves the problem by searching for the global minimum path, it must examine all potential solutions within its search space. On the other hand, Greedy Algorithm and Greedy + Local Search Algorithm are faster than other methods so it computes more efficiently. GRASP gives a better performance than CPLEX, but it is not as fast as Greedy Algorithm and Greedy + Local Search. GRASP performs worse than greedy because its iterative exploration and randomized construction increase computational overhead compared to Greedy's direct optimization.

The second graph shows the objective values based on the number of faculty members.Unlike the first graph, which focuses on computation time, this graph illustrates the objective values produced by each method. CPLEX and GRASP generates highest objection values when comparing to other methods. In addition, CPLEX, configured with an optimality gap of **epgap=0.01** for the achieving faster runtimes. The Greedy method performs the worst in terms of objective values, showing lower results compared to the other methods. However, when combined with Local Search, Greedy achieves significantly better results, indicating that the addition of Local Search helps improve its performance.

In conclusion, CPLEX guarantees the best solution but becomes too slow for larger problems. Greedy + Local Search is the fastest method and gives nearly the same quality of results. GRASP is a good balance between speed and solution quality. These results show that the best method depends on whether you need fast computation or the best possible solution.

6 References

- Lecture slides.
- Course Labs.
- `https://github.com/Cann23/MATH-Project`

List of Algorithms