ALGORITHMIC METHODS FOR MATHEMATICAL MODELS

Marc Díaz Calderón
Pau Adell Raventós

Index

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Integer linear model

Input Variables

- $N \in \mathbb{N}^+$: The number of faculty members.
- $D \in \mathbb{N}^+$: The number of departments in the faculty.
- A vector n of size D, where, $n[d] \in [1, N]$ represents the number of members of the department $d \in [1, D]$ that must attend the committee. The sum of members of n can not be grater than the total faculty members N.
- A vector d of size N, where $d[i] \in [1, D]$ indicates the department of individual $i, \forall i \in [1, N]$.
- A symmetric compatibility matrix m of size $N \times N$ where the coefficients $m_{ij} \in [0, 1] \subset \mathbb{R}$ of the matrix represent the compatibility between the individual i and j. A higher value indicates better compatibility.

```
int                  D = ...;
int             n[1..D] = ...;
int                  N = ...;
int             d[1..N] = ...;
float     m[1..N][1..N] = ...;
```

Decision Variables

- x_i: Variable will be True if the individual i is selected for the committee for $1 \leq i \leq N$.
- y_{ij}: Variable that will be True if the individual i AND the individual j are selected for $1 \leq i, j \leq N$.
- z: Variable that will contain the average compatibility of a given committee. The goal is to maximize it.

```
dvar boolean x[1..N];
dvar boolean y[1..N][1..N];
dvar float+ z;
```

Integer linear model

Auxiliary Variables

- T: This variable represents the total number of individuals that will form the committee. It is computed as:

$$T = \sum_{d \in D} n_d$$

- G: This variable represents the total number of unique pairwise comparisons that can be made from a set of T individuals. It is computed using the Gauss sum formula:

$$G = \frac{(T-1) \cdot T}{2}$$

```
int T = sum(dep in 1..D) n[dep];
float G = (T - 1) * T / 2;
```

Objective Function

We want to maximize the average compatibility between all the members of the committee. As stated in the decision variables then, the objective function is expressed as:

$$maximize\,(z) \tag{5}$$

```
maximize z;
```

And this z will be upper bounded by the average calculation. Hence, by maximizing Z, we are rising the upper bound as much as possible. Then to compute z, we will just iterate without repetition, as the compatibility that individual i has with individual j is symmetrical, and divide by the total given by the Gauss sum. Then, z is defined as:

$$z \leq \frac{\sum_{i=0}^{N-1} \sum_{j=i+1}^{N-1} m_{ij} \cdot y_{ij}}{G} \tag{6}$$

```
z <= (sum(i in 1..N, j in i+1..N)
     y[i][j] * m[i][j])/ G;
```


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Integer linear model - Constraints

1. **Department Participation:** Each department $d \in [1, D]$ must have exactly n_d participants, as specified by the input vector n. This can be expressed as:

$$\forall d \in [1, D], \quad \sum_{i=1}^{N} (\delta_{d[i],d} \cdot x_i) = n_d \quad \text{where } \delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j. \end{cases} \tag{1}$$

```
forall(dep in 1..D)
       sum(p in 1..N: d[p] == dep)
           x[p] == n[dep];
```

2. **Incompatibility Constraint:** No two individuals i and j whose compatibility $m_{ij} = 0$ can be selected simultaneously:

$$x_i + x_j \leq 1, \quad \forall i, j \in [1, N] \text{ with } m[i][j] = 0 \tag{2}$$

```
forall(i in 1..N,
       j in i+1..N: m[i][j] == 0)
       x[i] + x[j] <= 1;
```

3. **Middleman Friend Constraint:** We also need to ensure that in the case the committee is formed by two individuals whose compatibility is less than 0.15, there must be also in the committee a "middleman friend" that has compatibility higher than 0.85 with both members. That is:

$$\sum_{k \in [1,N] \setminus \{i,j\}} \delta_{m_{ik} > 0.85} \cdot \delta_{m_{kj} > 0.85} \cdot x_k \geq x_i + x_j - 1, \quad \forall i, j \in [1, N] \text{ with } 0 < m_{ij} < 0.15 \tag{3}$$

```
forall(i in 1..N
       j in i+1..N:
       0 < m[i][j] < 0.15) {
       sum(k in 1..N: k != i &&
       k != j && m[i][k] > 0.85 &&
       m[k][j] > 0.85)
       x[k] >= x[i] + x[j] - 1;
}
```

4. **Pairwise Selection Constraint:** For each pair of individuals i and j, the variable y_{ij} must be True only if both individuals are selected:

$$y_{ij} = 1 \Leftrightarrow x_i = 1 \land x_j = 1, \quad \forall i, j \in [1, N] \tag{4}$$

```
forall(i in 1..N, j in i+1..N) {
       y[i][j] <= x[i];
       y[i][j] <= x[j];
       y[i][j] >= x[i] + x[j] - 1;
```

Greedy Algorithm

Algorithm 1: Greedy Algorithm

$S \leftarrow [-1, -1, \ldots, -1]$;
$C \leftarrow \{0, 1, \ldots, N-1\}$;
$count \leftarrow 0$;
while $count < sum(n)$ **do**
  $C \leftarrow \{\texttt{FeasibilityFunction}(C, n, S)\}$;
  if C *is empty* **then**
    break
  end
  $best\_candidate \leftarrow \arg\max_{c \in C} \texttt{CostFunction}(c,\ C,\ S,\ count)$;
  $S[count] \leftarrow best\_candidate$;
  $C \leftarrow C \setminus \{best\_candidate\}$;
  $n[d_{best\_candidate}] \leftarrow n[d_{best\_candidate}] - 1$;
  $count \leftarrow count + 1$;
end
return $< f(S), S >$;

Algorithm 2: Feasibility Function

Input : Set of candidates C, Current partial solution S
Output : C' – Set of feasible candidates

$C' \leftarrow \{\}$;
foreach $c \in C$ **do**
  if $n[d[c]] > 0$ ***and not*** $\texttt{CandidateIncompatible}(c, S)$ ***and not*** $\texttt{NeedsMiddlemanAndNotFound}(c, C, S)$ **then**
    $C' \leftarrow C' \cup \{c\}$
  end
end
return C'

Greedy Algorithm

Algorithm 3: Cost Function (GreedyCostFunction)

Input : Candidate to evaluate c, Set of candidates C, Current partial solution S and number of already assigned candidates *count*

Function `PenalizedAffinity`($value$):

    if $value < 0.15$ **then**

        return $-e^{2\times(0.15-value)}$ `// Strong penalty for poor affinity`

    else if $value \geq 0.85$ **then**

        return $value + e^{2\times(value-0.85)}$ `// Boost for strong affinity`

    return $value$

$sum\_solution \leftarrow \sum_{i\in S, i\neq -1}$ `PenalizedAffinity`$(m[candidate][i])$;

$sum\_candidates \leftarrow \sum_{i\in C, i\neq c}$ `PenalizedAffinity`$(m[candidate][i])$;

$progress\_ratio \leftarrow c \div sum(n)$;

$W_1 \leftarrow progress\_ratio$;

$W_2 \leftarrow 1 - progress\_ratio$;

return $W_1 \cdot sum\_solution + W_2 \cdot sum\_candidates$

Local Search

Algorithm 4: Local Search - Best-improving strategy

```
Input  : Initial Solution S_0
S ← S_0;
for it ← 1 to max_iterations do
    if time.now() − start_time ≥ max_time then
        break
    end
    N ← GenerateNeighbors(S);
    if N = ∅ then
        break
    end
    S′ ← arg max_{n∈N} f(n);
    if f(S′) > f(S) then
        S ← S′;
    end
    else
        break // No improvement, exit early
    end
end
return < f(S), S >;
```

Algorithm 5: Generate Neighbors

```
Input  : S – Current solution (list of assigned candidates)
Output : N – List of valid neighboring solutions
N ← {} ;
foreach i ∈ S do
    C ← {c | d[c] = d[i] and c ∉ S};
    foreach c ∈ C do
        neighbor ← S;
        neighbor.swap(i, c);
        if SolutionIsValid(neighbor) then
            N ← N ∪ {neighbor};
        end
    end
end
return N
```

Greedy Randomized Adaptive Search

Algorithm 6: GRASP Solver

```
S ← {};
for it ← 1 to max_iterations do
    if time.now() − start_time ≥ max_time then
        break
    end

    S′ ← DoConstructionPhase();
    if S′ = ∅ then
        continue
    end

    S″ ← DoLocalSearch(S′);

    if f(S″) > f(S) then
        S ← S″;
    end
end
return < f(S), S >;
```

Algorithm 7: Construction Phase

```
S ← [−1, −1, . . . , −1] ;
C ← {0, 1, . . . , N − 1};
count ← 0;
while count < sum(n) do
    C ← {FeasibilityFunction(C, n, S)};

    if C is empty then
        break
    end

    q_max ← max{CostFunction(c, C, S, count) | c ∈ C};
    q_min ← min{CostFunction(c, C, S, count) | c ∈ C};
    threshold ← q_max − α · (q_max − q_min);
    RCL_max ← {c ∈ C | CostFunction(c, C, S, count) ≥ threshold} ;
    Select candidate ∈ RCL_max at random;

    S[count] ← candidate;
    C ← C \ {candidate};
    n[d_candidate] ← n[d_candidate] − 1;
    count ← count + 1;
end
return S;
```

Instance Generation

```
#
N = 65
D = 2
#

----------------------------------

for i in range(N):
        old_bad_value = False
        for j in range(i, N):  # Only generate the upper triangle, including diagonal
            value = -1
            if i == j:
                value = 1.0  # Diagonal elements set to 1.0
            if(old_bad_value):
                value = round(random.randint(18, 20) * 0.05, 2) # [0.9 - 1.]
                old_bad_value = False
            else:
                value = round(random.randint(0, 20) * 0.05, 2) # [0. - 1.]
            if value < 0.15:
                old_bad_value = True

            m[i][j] = value
            m[j][i] = value

    return m
```

Results - Alpha tuning

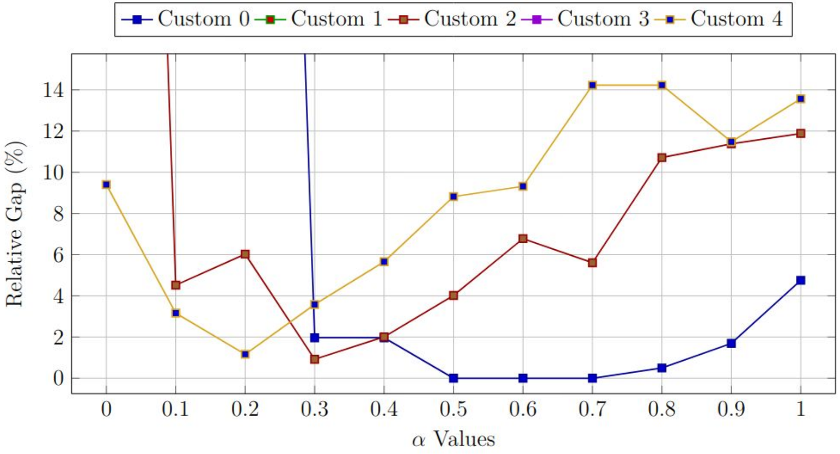


Figure 1: Relative Gap (%) vs different alpha values for different set of random instances of size N = 50

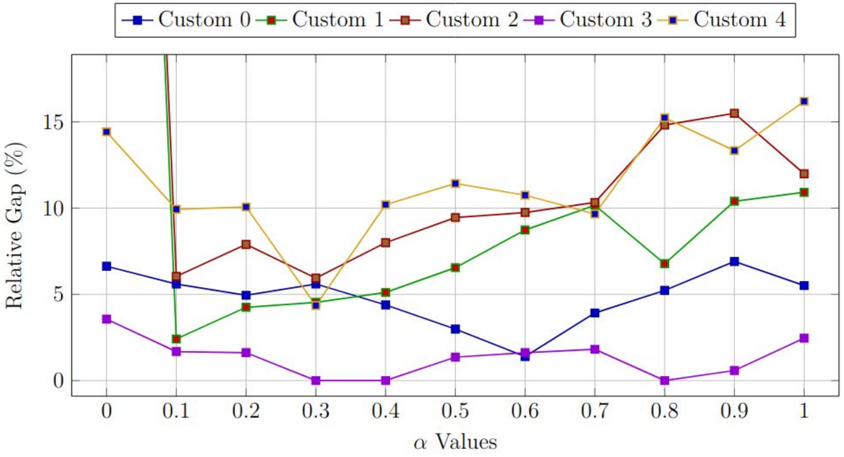
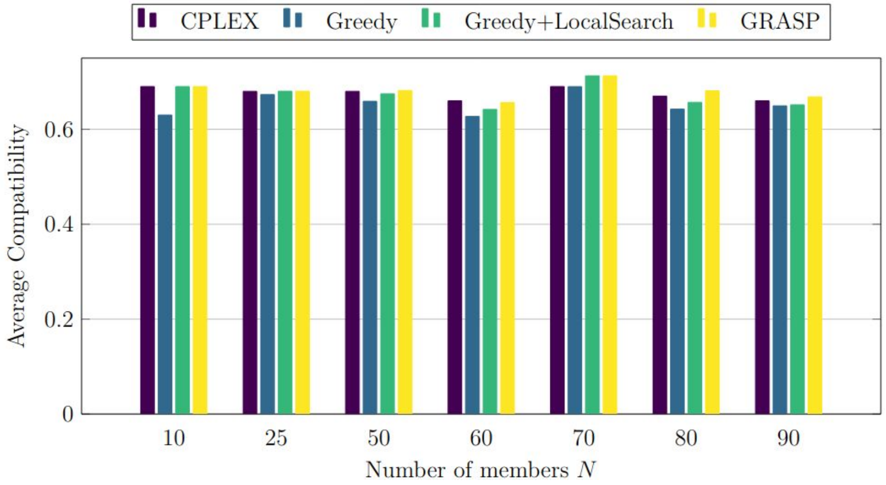


Figure 2: Relative Gap (%) vs different alpha values for different set of random instances of size N = 55

Results - CPLEX vs heuristics



Members	CPLEX	Greedy	Greedy+LocalSearch	GRASP
10	0.10	0.0000	0.00000	0.03501
25	0.21	0.00100	0.00100	0.32870
50	37.12	0.00400	0.01951	3.15251
60	429.00	0.00700	0.04852	5.87972
70	1,800.00	0.00800	0.04426	12.33155
80	1,803.00	0.01351	0.18565	19.59827
90	1,800.00	0.01551	0.11866	38.35215