Abstract

Tackling the thesis - teacher - jury problem by trying different algorithms namely, some exact algorithms (Back- tracking, branch & bound, integer or constraint programming), some approximate algorithms (greedy, meta-heuristics).

1 Introduction

There are N theses and M teachers need to be divided into K juries.

- t(i) is the teacher who instructs thesis i.
- The similarity between theses i and j is s(i,j).
- The similarity between theses i and teacher j is g(i,j).

Some constraints to the problem:

- The number of theses of each jury has to be greater than a given number **a** and smaller than a given number **b**.
- The number of teachers of each jury has to be greater than a given number **c** and smaller than a given number **d**.
- Teachers are not allowed to be in the same jury with the student they instruct.
- The similarity between theses in each jury is greater than a given number **e**.
- The similarity between theses and teachers in each jury is greater than a given number **g**.

2 Backtracking

We demonstrate the process of backtracking in the image below. It is a very straight forward approach, generate all possible solution and calculate the maximum value of the objective function.

3 Branch & Bound

We start our algorithms by finding all possible way to assign theses to jury like the previous backtracking algorithm. The theses assigned to the jury should satisfied the constraints involved in the number of theses per jury.

Initially, we set our maximum objective function as 0. When assigning the i^{th} teacher to the jury we calculate the total similarity of teachers and theses that has been assigned before (from 1^{st} to $(\text{i-1})^{\text{th}}$ teachers). If i^{th} is not the last teacher to be assigned, we approximate the maximum similarity of teachers and theses (the objective function) for the rest (K - (i+1)) teachers. If the approximation is smaller than the current maximum objective function value, we remove the current solution and iterate to the next solution.

If i^{th} is the last teacher, we update the maximum objective function value if it is larger than the current one.

3.1 Conclusion

The Backtracking algorithm systematically explored all possible assignments of theses to juries, adhering to the specified constraints. The algorithm's performance do well on very small test case, but with larger test cases, it can not give solution in a short amount of time. The Branch & Bound approach use a more sophisticated objective function, considering the total similarity of teachers and theses assigned to juries. It optimized the objective function by iteratively assigning teachers to juries and evaluating the potential for better solutions. This led to a faster, more efficient algorithms. Despite the fact that it is faster than the backtracking approach, there is no significant improve in performance for large test cases.

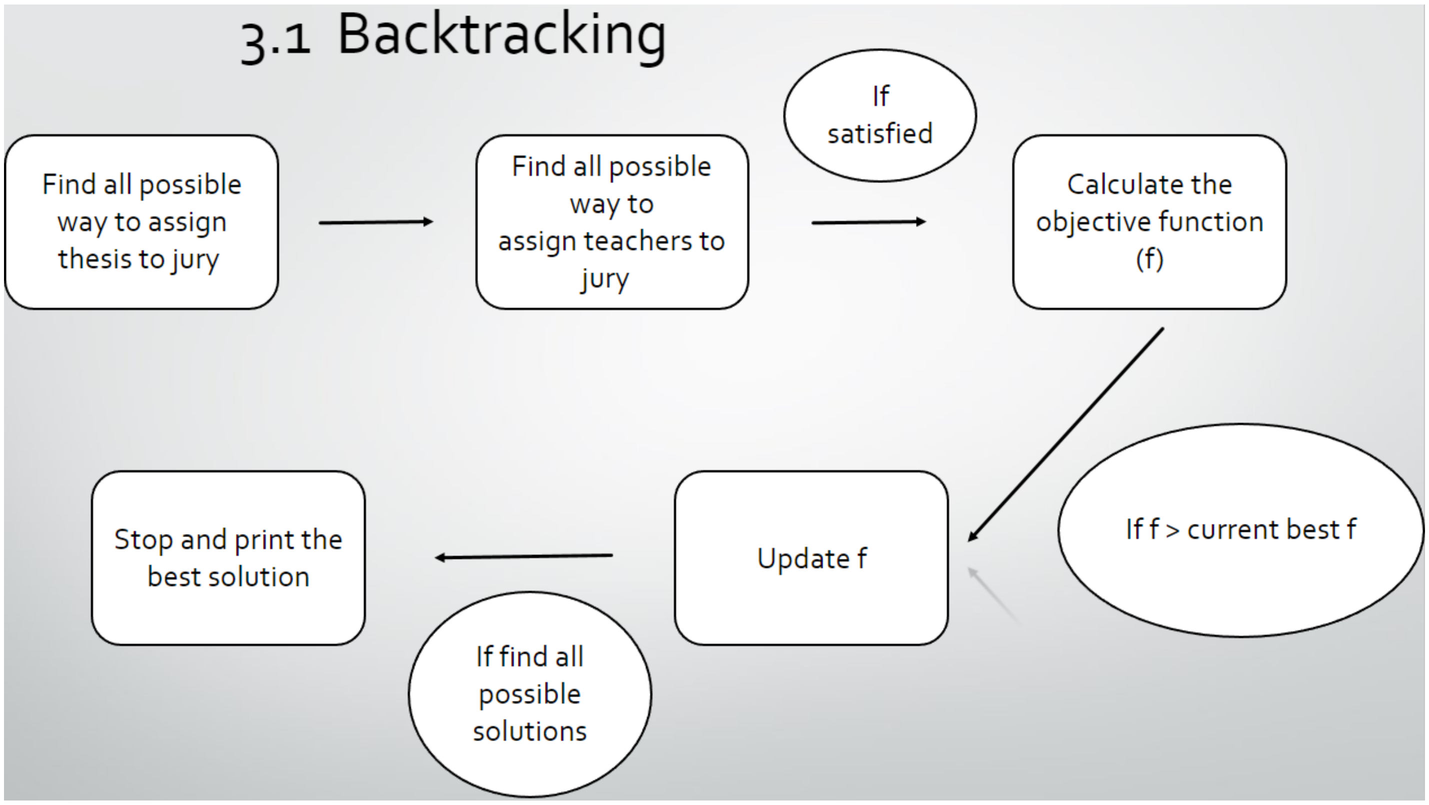
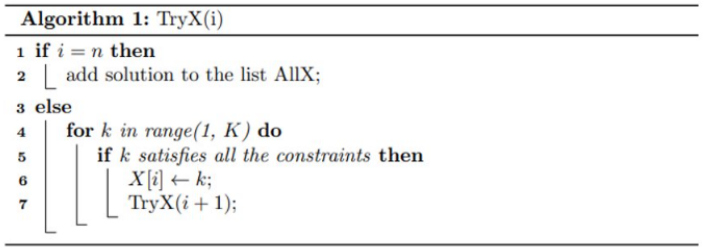
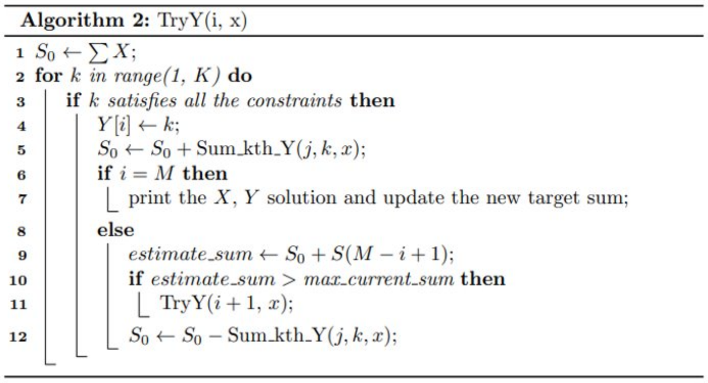

3.1 Backtracking
Find all possible way to assign thesis to jury
Find all possible way to assign teachers to jury
If satisfied
Calculate the objective function (f)
If f > current best f
Update f
If find all possible solutions
Stop and print the best solution

Figure 1: Process of Backtracking



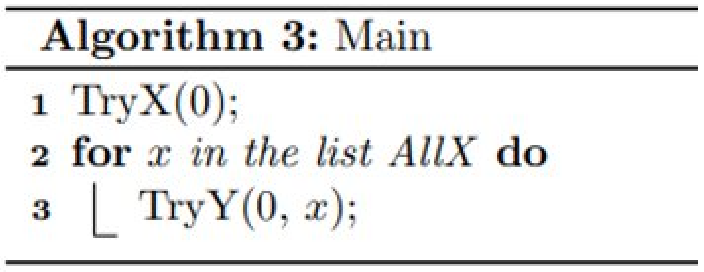
```
Algorithm 1: TryX(i)
if $i = n$ then
    add solution to the list AllX;
else
    for $k$ in range(1, $K$) do
        if $k$ satisfies all the constraints then
            $X[i] \leftarrow k$;
            TryX($i + 1$);
```

Figure 2: The pseudocode for the alogrithm



```
Algorithm 2: TryY(i, x)
$S_0 \leftarrow \sum X$;
for $k$ in range(1, $K$) do
    if $k$ satisfies all the constraints then
        $Y[i] \leftarrow k$;
        $S_0 \leftarrow S_0 + \text{Sum\_kth\_Y}(j, k, x)$;
        if $i = M$ then
            print the $X$, $Y$ solution and update the new target sum;
        else
            $estimate\_sum \leftarrow S_0 + S(M - i + 1)$;
            if $estimate\_sum > max\_current\_sum$ then
                TryY($i + 1$, $x$);
            $S_0 \leftarrow S_0 - \text{Sum\_kth\_Y}(j, k, x)$;
```

Figure 3: The pseudocode for the alogrithm



Algorithm 3: Main

```
1 TryX(0);
2 for $x$ in the list AllX do
3     TryY(0, $x$);
```

Figure 4: The pseudocode for the alogrithm

4 Or Tools (Integer Programming or Constraint Programming)

Here we will use Or Tools to model and solve this problem. In general, two models: Cp Model (used for constraint programming) and SCIP (used for Integer Programming) are basically the same, so here we will model its constraint and objective function by only one model which is Cp Model. This section consists of 3 parts:

1. Declaring Decision Variables
2. Define the Constraints
3. Define the Objective Function
4. The Testing Result

4.1 Decision variables

Here is the decision variables that we use:

- X[n][k] (n from 1 to N, k from 1 to K): to determine whether thesis n is in the jury k or not
- Y[m][k] (m from 1 to M, k from 1 to K): to determine whether teacher m is in the jury k or not
- Z[n1][n2][k] (n1 from 1 to N, n2 from 1 to N, k from 1 to K) is to determine if thesis n1 and n2 are in the same jury k or not
- T[n][m][k] (n from 1 to N, m from 1 to M, k from 1 to K) is to determine if thesis n and teacher m are in the same jury k or not

4.2 Define the constraints

1. For each jury k, the number of theses is greater or equal than a, least or equal than b, which means for each k from 1 to K, we have:
   - $\sum_{i=1}^{N} X[i,k] \geq a$
   - $\sum_{i=1}^{N} X[i,k] \leq b$
2. For each jury k, the number of teacher is greater or equal than c, least or equal than d, which means for each k from 1 to K, we have:
   - $\sum_{i=1}^{M} Y[i,k] \geq c$
   - $\sum_{i=1}^{M} Y[i,k] \leq d$
3. In each jury, the similarity between two thesis is larger or equal than e, the similarity between a thesis and a teacher is larger or equal than f. Since the similarity between two thesis is count 2 times, we need to have it larger or equal than 2*e. Therefore, for each jury k from 1 to K, we have:
   - $\sum_{i=1}^{N} \sum_{j=1}^{N} Z[i,j,k] \geq 2 * e$

- $\sum_{i=1}^{N}\sum_{j=1}^{M} T[i,j,k] \geq f$
-

4. $X[n][k] = 1 \Leftrightarrow x[n] = k$: This is to ensure that when thesis n is in the jury k then the decision variable X[n][k] = 1 and vice versa
    - $y[m] + CONST * (1 - Y[m,k]) \geq k$
    - $y[m] - CONST * (1 - Y[m,k]) \leq k$
5. $Y[m][k] = 1 \Leftrightarrow y[m] = k$. This is to ensure that when teacher m is in the jury k then the decision variable Y[m][k] = 1 and vice versa
$$X[n,k] + Y[t[n],k] \leq 1$$
6. For each teacher, the theses conducted can not be in the their jury which means if $X[n,k] = 1 \Rightarrow Y[t[n],k] = 0$ and vice versa. Therefore, for each n from 1 to N, k from 1 to K we have:
$$X[n,k] + Y[t[n],k] <= 1$$
7. Ensure the connection that if n1, n2 is in the same jury k then $X[n1,k] = X[n2,k] = Z[n1,n2,k] = 1$. In other words: $X[n1,k] = X[n2,k] = 1 \Leftrightarrow Z[n1,n2,k] = 1$
$$X[n1,k] + X[n2,k]) \leq Z[n1,n2,k] + 1$$
$$X[n1,k] + X[n2,k] \geq 3 * Z[n1,n2,k] - 1$$
8. Similarly, if teacher m and theses n is in the same jury k then X[n,k] = Y[m,k] = T[n,m,k] = 1 which means:
$$X[n,k] + Y[m,k]) \leq T[n,m,k] + 1$$
$$X[n,k] + Y[m,k] \geq 3 * T[n,m,k] - 1$$

4.3 Objective function

In the objective function, the similarity between 2 thesis is only count 1 time, so the objective function we need to maximize is multiplied by 2 (just to make it an integer, then we would divide it by 2 later):

$$\sum_{k=1}^{K}\sum_{n_1=1}^{N}\sum_{n_2=1}^{N} Z[n_1,n_2,k] * s[n_1,n_2] + \sum_{k=1}^{K}\sum_{n=1}^{N}\sum_{m=1}^{M} 2 * T[n,m,k] * g[n_1,n_2]$$

4.4 Implementation and testing result

We test all the test on our local computer with no GPU and set the time limit to 1 minute seconds. However, we can only find a sufficiently good solution for a test which has the size least or equal than 100; more than that will take really long.

Then to test the effectiveness of this model, we test it with several test cases and 3 time limit level: 20 seconds, 40 seconds and 60 seconds. This is the result we get:

Thi gian	M	N	K	a	b	c	d	e	f	Score
20s	6	4	2	2	4	1	3	1	1	66
40s	6	4	2	2	4	1	3	1	1	66
20s	50	15	5	1	12	1	5	1	1	1433
40s	50	15	5	1	12	1	5	1	1	1471
30s	100	30	10	1	12	1	5	1	1	2557
60s	100	30	10	1	12	1	5	5	5	2522
90s	100	30	10	1	10	1	3	10	10	2369

Table 1: The testing result

Algorithm 1 Generate X and Y

```
function GENERATEX($X$)
    for $i$ in $[0, N)$ do
        $options\_for\_thesis \leftarrow$ a list of rooms that the $i$-th thesis can occupy
        $room \leftarrow$ choose randomly from $options\_for\_thesis$
        $X.append(room)$
    end for
    return $X$
end function
function GENERATEY($Y$)
    for $i$ in $[0, M)$ do
        $options\_for\_teacher \leftarrow$ a list of rooms that the $i$-th teacher can occupy
        if all rooms have over $c$ teachers then
            choose the room that can increase the target sum the most from $options\_for\_teacher$
        else
            choose the room from $options\_for\_teacher$ with the lowest number of teachers
        end if
    end for
    return $Y$
end function
```

(a) The pseudocode for generate X and Y function

```
function MAIN
    $iterations \leftarrow an\ integer$
    $best\_solution \leftarrow \{\}$
    for $i$ in $[0, iterations)$ do
        $X \leftarrow$ GENERATEX([])
        $Y \leftarrow$ GENERATEY([])
        if $X, Y$ is better than the best solution we have right now then
            $best\_solution \leftarrow \{X, Y\}$
        end if
    end for
    return $best\_solution$
end function
```

(b) The pseudocode for main function

Figure 5: The pseudocode for the greedy approach

5 Greedy

5.1 The process of Greedy approach

1. Grouping and Sorting: Initially, we group the theses based on their guided teachers and sorts them in descending order of the number of theses each teacher guides.
2. Numerical Constraint Assignment: We then attempts to assign thesis to advisors based on a numerical constraint. The greedy algorithm tries to distribute theses evenly across juries while maximizing the number of theses each teacher guides.
3. Adjustment: The code adjusts the initial assignment to meet additional constraints. It handles cases where the number of advisors in a committee exceeds the specified limit and reassigns them to other committees to optimize the solution.
4. Final Result: We return the final assignment of theses and teachers to juries, ensuring that all constraints are satisfied.

5.2 Conclusion

The greedy approach implemented in the code provides a heuristic solution to the problem. While it may not guarantee an optimal solution, the approach efficiently handles constraints and maximizes the overall similarity between theses and teachers within each jury. The algorithm's simplicity allows for quick iterations and adjustments, making it suitable for practical use in scenarios where computational resources are limited.

6 Random-restart hill climbing

"Random-restart hill-climbing conducts a series of hill-climbing searches from randomly generated initial states, running each until it halts or makes no discernible progress" (Russell & Norvig, 2003). This enables comparison of many optimization trials, and finding a most optimal solution thus becomes a question of using sufficient iterations on the data.

6.1 Fitness function

There are many ways to evaluate this fitness function. We use the simplest way, which is calculate the number of thesis/teacher which violates the given constraints. Here is how we evaluate it:

- the number of teacher/thesis exceeded the given constraints in each jury

- the amount of similarity between 2 thesis/a thesis and a teacher which exceed the given constraint
- the number of student who is the same jury with the teacher conducted them

6.2 Initial solution

To initiate the initial solution, we will employ two main ideas, both aiming to satisfy constraints on the number of teachers/theses in a committee and the constraint that students cannot be in the same committee as their advising teacher:

1. First is the greedy approach, where we arrange teachers who supervise the most students first, as they are the teachers with the highest likelihood of violating the highest constraints. We strive to allocate students of that teacher to as few committees as possible.
2. Subsequently, we attempt to evenly distribute the number of teachers/theses across each committee to satisfy the constraint on the number of teachers/theses. Simultaneously, after each allocation of students of a teacher to committees, we immediately assign that teacher to the next committee (according to MODULO K).

6.3 Generating neighbour

The initialization method outlined above, while helping us satisfy many constraints right from the start, still has a weakness: There will exist committees with either too many or too few teachers. To address this issue, I will proceed as follows:

1. Identify teachers in incorrect committees and their corresponding advised students.
2. Transfer those teachers to committees without any of their advised students.
3. Simultaneously, to ensure that too many new teachers are not added to a committee, I will mark the committees that have been supplemented to ensure that each committee is only supplemented with a single new teacher (of course, checking whether the committee was already full before).

6.4 Random-restart

Now, to complete the algorithm, we need to decide how to start randomly. Here, I will randomly choose the first committee to assign students of the teacher who supervises the most theses, and then proceed in a circular fashion with a MODULO K pattern. Subsequently, depending on the test case, I will determine how many feasible solutions need to be obtained. However, since the initialization of the solution is already good, typically finding a single solution is sufficient to yield a very good solution.

Time	M	N	K	a	b	c	d	e	f	Score
0.0009970664978027344	6	4	2	2	4	1	3	1	1	66
0.0009987354278564453	50	15	5	1	12	1	5	1	1	1174
0.01997852325439453	100	30	10	1	12	1	5	1	1	2261
26.55797839164734	600	150	60	1	12	1	5	1	1	12514
67.54646545212152s	1000	200	100	1	12	1	5	1	1	19460

Table 2: The testing result

6.5 Conclusion

The Random-Restart strategy enhanced the greedy algorithm's effectiveness in finding optimal or near-optimal solutions. Across various problem sizes, the algorithm demonstrated good performance, providing feasible solutions within a reasonable time given. The Random-Restart Hill Climbing approach is a versatile and efficient optimization technique for the problem.

7 Tabu Search

Tabu search (TS) is a metaheuristic search method employing local search methods used for mathematical optimization. Tabu search enhances the performance of local search by relaxing its basic rule. First, at each step worsening moves can be accepted if no improving move is available. In addition, prohibitions (hence the term tabu)

are introduced to discourage the search from coming back to previously-visited solutions. The implementation of tabu search uses memory structures that describe the visited solutions or user-provided sets of rules. If a potential solution has been previously visited within a certain short-term period or if it has violated a rule, it is marked as "tabu" (forbidden) so that the algorithm does not consider that possibility repeatedly.

7.1 Condition

We use it to check the solution and neighbors whether they satisfy.

1. Teachers can't be placed in the same juries with the students they instruct
2. The number of students and teachers in each jury exceeded the given constraints
3. The similarity between students and between students and teachers in each jury is under the given constraints

7.2 Generate Solution

We generate a solution by three main steps:

1. Firstly, we group the students who have the same teacher. Then we sort the groups based on the number of students.
2. Subsequently, we place the students who have the same teacher in a the same jury but if the number of student is more than b, we place them to the next jury. Because teachers can't be placed in juries that has the student they instruct. So after placing students, we mark the juries for each teacher.
3. We place the teacher randomly based on the marked juries and the limit of teachers in each jury.

7.3 Generate neighborhood

The solution above are quite accurate so we will change the positions between different teachers in different juries.

1. Find two different teacher in different juries and change their positions .
2. Then we change all the students instructed by those teachers into another juries.
3. We check the neighbor, if it satisfy the condition and not in tabu-list then add them into the neighbors' list.

7.4 Tabu search

We begin by generating the solution.

1. In each iteration, we generate neighbors for the current solution and calculate the sum of the similarity of them and find the best neighbor. If there is no best neighbor, stop the iterations.
2. The current solution will be the best neighbor and then we add the best neighbor into the tabu-list to avoid repetition.
3. Finally, we calculate the current solution's sum of similarity and compare it with the best solution's one. If it's bigger, then the current solution become the best solution. Else, we remain the best solution.

7.5 Conclusion

Tabu Search gives quite positive solutions in finding optimal or nearly optimal solutions. In this situation, the algorithm provided optimal solution in a short time but in a small size problem (less than 200 students). This may be cause by not good generating current solution. If we can combine it with Greedy algorithm, it can solve various problem sizes.

8 Summary

We have summarized our implementation to a table.

Table 3: Pros and Cons of Implemented Algorithms

	Pros	**Cons**
Branch & Bound, Backtracking	Optimized result, easy to implement	Large running time, impractical for large cases
Constraint programming & Integer programming	Optimized result, very consistent, faster than B&B and Backtracking	Large running time for huge cases
Greedy	Fast running time, sometimes gives near optimal results	Can be inconsistent for some cases
Local Search	Fast running time, sometimes gives near optimal results	Can be inconsistent due to randomness
Tabu Search	Fast running time, sometimes gives near optimal results	Memory usage according to tabu list

9 Work contribution

Each of the members in the group has contributed many efforts in this project.

- Nguyen Duc An : implemented the constraint programming or integer programming approach, greedy approach, random-restart hill-climbing approach, experimented all the algorithms with different cases, wrote technical report.
- Nguyen Duc Anh: implemented tabu-search approach,experimented all the algorithms with different cases, made slide representation, wrote technical report.
- Nguyen Viet Long: implemented backtracking, contribute to random-restart hill-climbing approach, experimented all the algorithms with different cases, wrote technical report, made slide presentation.
- Nguyen Tien Thanh: implemented constraint programming or integer programming approach, greedy approach, branch & bound approach.