LIÈGE université
Sciences Appliquées

MATH0462-1: Room Optimization

Romain Lambermont[1] and **Arthur Louis**[2]
[1] *romain.lambermont@student.uliege.be (s190931)*
[2] *alouis@student.uliege.be (s191230)*

Course given by [1]**Quentin Louveaux**
[1] *q.louveaux@uliege.be*

ABSTRACT:

The allocation of rooms for university courses is a crucial factor in students' learning experience. However, at the University of Liège, the dispersed nature of available rooms often leads to students traveling long distances throughout the day, causing inconvenience and wasting time and energy.

This project aims to centralize room allocation to minimize students' travel distance. We will formulate the problem as a Mixed Integer Programming (MIP) model, ensuring that courses in the same room do not overlap, each course has sufficient seating capacity, each course is assigned to a room and overall student travel distance is minimized.

Additionally, a heuristic algorithm will be developed to provide a practical solution. This algorithm will iteratively improve the initial solution through local search techniques, providing a near-optimal room assignment in a reasonable amount of time.

CONTENTS

I. MIP

In this section, we will describe the implementation details of the Mixed Integer Programming (MIP) model, the development steps, findings and limitations. The implementation of the model followed these main steps :

A. Reading the Data

The first step to undertake when starting this project was to read the data provided. The `.csv` and `.xlsx` files were read using the package `pandas`, providing a simple and easy to use matrix style representation. The `.json`, on the other hand were read using the `json` package.

To make it easy to handle the different courses and especially their starting and ending times, the `Start` and `End` columns of the `Events` file were translated to the type `datetime` using the built-in function `pandas.to_datetime()`.

B. Creation of Helpful Structures

Unfortunately, simply working with the given data would make it difficult to create the different constraints to use in our model. These different data structures were created to help us in the design of the model :

- `distances`: we simply transformed the given upper-triangle matrix into a basic square matrix to help us compute the distances between classrooms more easily.
- `distance_between_classrooms`: using the `distances` matrix computed earlier, we computed the distance between each and every pair of classrooms in the University to avoid computing them at the time of creation of the model.
- `conflictsNN`: Once again to avoid computing this when creating the model, we created this matrix to report the courses that could not take place in the same classroom as they had a nonempty intersection between their schedules.
- `enrolledNN`: This data structure was created to help us when dealing with the room capacities when designing the model and reports the attendance of each course.
- `courses_per_studentNN`: This data structure simply "reverts" the `studentsNN.json` file giving every student schedule for each day.
- `flowsNN`: Making it easy to compute the objective function, this data structure gathers the flow of student going from a course to another by using the structure created just before and sorting it by the start time of the student's courses.

The data structures ended by `NN` were computed for each and every day.

C. Creation of the Model

The creation of the model followed these steps :

1. Decision Variable

For this model, only one variable was needed. We called it x_{ij}:

$$x_{ij} = \begin{cases} 1 \text{ if course } i \text{ takes place in classroom } j \\ 0 \text{ otherwise} \end{cases}$$

2. Objective Function

The objective function to optimize in this project is to minimize the total travelling minutes of students between courses along the day. The formula can be computed with the previously expressed data structures by denoting:

e_i a course $\in \mathcal{E}$ the set of courses

c_i a classroom $\in \mathcal{C}$ the set of classrooms

$$x_{e_i c_i} = \begin{cases} 1 \text{ if course } e_i \text{ takes place in classroom } c_i \\ 0 \text{ otherwise} \end{cases}$$

$f_{e_i e_j}$ the flow of students going from course e_i to course e_j

$d_{c_i c_j}$ the distance between classroom c_i and c_j

The final objective function can be written as :

$$\min \sum_{e_i \in \mathcal{E}} \sum_{e_j \in \mathcal{E}} \sum_{c_i \in \mathcal{C}} \sum_{c_j \in \mathcal{C}} x_{e_i c_i} \times x_{e_j c_j} \times f_{e_i e_j} \times d_{c_i c_j} \quad (1)$$

3. Constraints

To respect the project requirements we must satisfy at least three constraints:

- Each and every course must be organized in only one classroom :

$$\sum_j x_{ij} = 1 \qquad \forall i$$

- Each and every course must have a classroom big enough to accommodate the students enrolled (with E_i and K_i representing respectively the number of students enrolled to course i and the capacity of the classroom j):

$$x_{ij} \times E_i \leq K_j \qquad \forall i, j \quad (2)$$

- Two courses organized in the same classroom cannot have a non empty intersection of their schedule (implemented using the structure `conflictsNN`):

$$x_{ik} + x_{jk} \leq 1 \quad \forall k \quad \text{if their schedule intersect}$$

4. Results

Unfortunately when optimizing this model using `gurobipy`, the optimizer freezes and can not use the objective function formulated in the equation 1. We thus decided to optimize the constant value 0 to see if the optimizer could find a feasible solution but a relaxation was needed in the equation 2 to help it find a feasible solution (factor of 1.5 for days 18 and 20 and 2 for days 19 and 21. The amount of travelling minutes needed by students is not satisfying and we have tried improving on it in the Heuristic part of the project.

II. HEURISTIC

In this section, we will describe the implementation details of the heuristic approach used to solve the room allocation problem. The heuristic aims to provide a practical solution by iteratively improving an initial random solution through local search techniques. The implementation consists of the following key steps:

A. Random Solution Generation

The first step in the heuristic implementation is to create a random solution. This is achieved by generating a schedule, which is a dictionary containing the assignment of each event to a room. For each event, a room is randomly selected, taking into consideration the capacity and availability constraints. However, to avoid getting stuck in an infinite loop due to infeasible room allocations, a restart mechanism is implemented. If an infeasible solution is encountered, the algorithm starts over with an empty schedule until a feasible solution is reached.

B. Pairing Dictionary Creation

Once a random solution is obtained, a pairing dictionary is created. The pairing dictionary maps each event to its assigned room. This pairing information is essential for further calculations and optimization steps in the heuristic.

C. Cost Matrix Calculation

To evaluate the quality of the current solution and identify potential improvements, a cost matrix is computed. The cost matrix is created based on the flow matrix, which represents the number of students moving from one event to another, and the pre-computed distances between rooms. For each pair of events (i, j), the cost is calculated as the product of the flow from event i to event j and the distance between the corresponding assigned rooms. The resulting cost matrix provides a measure of the total distance traveled by students in the current solution.

D. Local Search for Improvement

The local search process aims to improve the current solution by addressing costly trajectories. Initially, the 20 most expensive trajectories from the cost matrix are selected.

For each selected trajectory, the algorithm attempts to find a new room assignment for the destination event. It starts with the first trajectory and checks if there is a different room available that satisfies the capacity and availability constraints. If a suitable room is found, the algorithm stops and proceeds with the following steps:

- Update the pairing dictionary: Assign the new room to the destination event in the pairing dictionary.
- Recompute the cost matrix: Calculate a new cost matrix based on the updated pairing dictionary, flow matrix, and room distances.
- Update the schedule: Modify the schedule by updating the room assignment for the destination event.
- If no suitable room is found for the destination event, the algorithm proceeds to the next trajectory, attempting to find a new room assignment for its destination event. This process continues until a feasible room assignment is found or all trajectories have been considered.
- During each iteration of the local search, only one room change is made. If the algorithm exhausts all trajectories without finding a new room assignment, it indicates that further room swaps within a single iteration would not yield an improvement.

E. Result Generation

Once an improved solution is found, the heuristic implementation returns the updated pairing dictionary,

the modified schedule with new room assignments, and the updated cost matrix. These results provide valuable insights into the enhanced solution and can be used to analyze and compare the effectiveness of the heuristic approach.

The returned pairing dictionary captures the optimized room allocation for each event, while the updated schedule reflects the revised room assignments. Additionally, the updated cost matrix quantifies the total distance traveled by students in the improved solution.

By dividing the heuristic implementation into these two parts, the local search for improvement and result generation, we can clearly understand the process of refining the initial solution and the information obtained from the heuristic.

F. Cost Evolution

The cost, measured in minutes per student, was tracked throughout the iterations of the algorithm. The initial random solution had a cost of approximately 700,000 minutes, while the heuristic algorithm reduced it to around 300,000 minutes.

Figure 1 shows the graph of the cost evolution. The x-axis represents the iteration number, and the y-axis represents the average cost in minutes per traject.

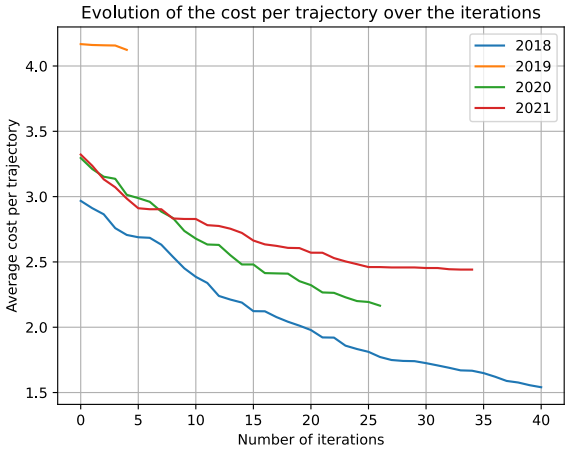

Figure 1. Cost Evolution (Minutes per Student) over Iterations

The graph demonstrates a gradual decrease in the cost over the iterations, indicating consistent improvement in the solution quality. The reduction in cost reflects the optimized room allocation achieved by the heuristic algorithm.

Overall, the heuristic approach successfully minimized the distance traveled by students, resulting in a more efficient room allocation for the university courses.

III. CONCLUSION

Comparing the approaches undertaken for this project, we can conclude that the heuristic worked much better than the MIP model. Indeed, due to the number of constraints and the complexity of the objective function, the MIP model was really hard to optimize (at least when testing on our computers who don't have unlimited computing power and memory to compute the constraints).

The heuristic proved to be much more efficient. Indeed, finding a random feasible solution did not take a lot of time and we could directly start making some swaps to try enhance our travelling times in a reasonable time compared to the MIP model.

In both implementations, days 19 and 21 proved to be the hardest to optimizes, potentially due to their bigger number of courses organized on the same day.