HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATIONS TECHNOLOGY

FINAL REPORT

COURSE: FUNDAMENTALS OF OPTIMIZATION

Topic 8: Class - Course - Teacher Assignment and TimeTabling

Class ID: 152625
Group 1:
Phạm Đức Duy - 20235588
Ngô Mạnh Hiếu - 20235590
Dương Ngô Hoàng Vũ - 20235629
Hoàng Quốc Huy 20235594

Supervisor: PhD Bui Quoc Trung

Table of Contents

I. Problem Description:

1. Problem Description

- There are T teachers, M subjects, and N classes.
- Each subject has a number of periods d(m).
- Each class has a list of subjects that they need to study.
- Each teacher t has a list of subjects that they can teach.
- There are 5 school days, each day is divided into 2 sessions (morning and afternoon), and each session is divided into 6 periods.
- The task is to create a teacher assignment plan along with a timetable (starting day/period) for each class- subjects pair such that:

  + Class-subjects pairs of the same class do not have overlapping timetables.

  + Class-subjects pairs assigned to the same teacher do not have overlapping timetables.

  + The total number of assigned class- subjects and teachers is maximized.

2. Input

- Line 1 : ***T*** (number of teachers)***, N*** (number of classes)***, M*** (number of subjects)
- Line i + 1 (i = 1, 2, … , N) : Enter the list of subjects that class ***i*** need to study (ends with 0)
- Line t + N + 1 (t = 1, 2, … , T) : Enter the list of subjects that teacher ***t*** can teach (ends with 0)
- Line N + T + 2 : Enter ***d(m)***, which is the number of periods of subjects ***m*** (m = 1, 2, … , M)

3. Output

- Line 1 : Print a positive integer ***K***
- Line k + 1 (k = 1, 2, … , K) : Print 4 positive integers ***x, y, u, v*** where class ***x*** – subject ***y*** is scheduled to start at period ***u*** and teacher is ***v***

II. Our methods to solve our problem:

1. Greedy Method

a) Introduction

The **Greedy** method is used in this problem to maximize the number of class-subject assignments. The goal is to assign teachers and schedule subjects optimally by handling each class-subject pair independently and selecting the best solution at each step.

b) Initialize required data structures

- **Class schedules** (schedule) and **teacher schedules** (teacher_schedules) to track the class timetable and teacher assignments.
- An array subject_periods stores the number of periods for each subject.

c) Iterate through each class and its subjects

- For each class, consider the subjects that the class needs to study.
- Check which teacher(s) can teach the subject. If a teacher is available, try to assign the subject in a suitable time slot.

d) Check for assignment feasibility

The function can_assign() is used to check if a subject can be assigned to a teacher at a specific time. The conditions are:

- There should be no time overlap with other class-subject assignments for the same class.
- The teacher should not have overlapping classes in their schedule at the same time.

e) Update the result

- If a valid time slot and teacher are found, assign the class-subject and update both the class and teacher schedules.
- Store the assignment information for each class-subject: (class, subject, start time, teacher).

  The Greedy approach works by assigning classes to teachers and time slots as we go, ensuring that at each step, the best available option is chosen. By doing so, the algorithm maximizes the number of class-subject assignments while respecting the constraints on teacher availability and class timetable conflicts.

f) Conclusion

The Greedy algorithm provides a simple yet effective approach to solving the teacher assignment and scheduling problem. By iterating through each class-subject pair and choosing the best possible assignment at each step, the algorithm maximizes the number of class-subject assignments while satisfying the constraints. However, this approach may not always produce the globally optimal solution, but it provides a good approximation, especially in scenarios with large inputs.

2. Constraint Programming

We chose Constraint Programming because it is easy to implement the multiplication (check 4^{th} constraint) and "if" condition (check 5^{th} constraint)

a) Data Parsing

T : number of teachers

N : number of classes

M : number of subjects

class_subjects[c] : list contains subjects that class ***c*** need to study

subject_teachers[m] : list contains teachers that can teach subject ***m***

subject_periods[m] : number of periods of subject ***m***

TOTAL_PERIODS = 5 * 2 * 6 = ***60*** : number of periods in a week

b) Variables

for *c* in [*1*, *N*]:

for *m* in *class _ subjects*[*c*]:

for *h* in *range*(*TOTAL _ PERIODS*):

$$x[(c,m,h)] = \begin{cases} True = 1, & \text{if class } c \text{ study subject } m \text{ at period } h \\ False = 0, & \text{otherwise} \end{cases}$$

for *c* in [*1*, *N*]:

for *m* in *class _ subjects*[*c*]:

for *t* in *subject _ teachers*[*m*]:

$$y[(c,m,t)] = \begin{cases} True = 1, & \text{if class-subject } c\text{-}m \text{ is assigned to teacher } t \\ False = 0, & \text{otherwise} \end{cases}$$

c) Some important sums:

If *c*, *m* are given :

=> ***sum(y[(c, m, t)] for t in subject_teachers[m])*** is

the number of teachers that class-subject *c* - *m* is assigned to

If *c*, *h* are given :

=> ***sum(x[(c, m, h)] for m in class_subjects[c])*** is

the number of subjects that class *c* study at period *h*

If *c*, *m* are given :

=> ***sum(x[(c, m, h)] for h in range(TOTAL_PERIODS))*** is the number of periods that class-subject *c* – *m* was taught

d) Objective function

Maximize : ***sum(y[(c,m,t)]***

for ***c*** in ***[1, N]***

for ***m*** in ***class_subjects[c]***

for ***t*** in ***subject_teachers[m]***)

e) Constraints

- 1st: Each class-subject must be assigned to exactly one teacher

for *c* in [*1*, *N*]:

for *m* in *class _ subjects*[*c*]:

sum(y[(c,m,t)] for t in subject_teachers[m]) <= 1

- 2nd: Each class-subject must be scheduled exactly the required number of periods, and those periods should be taught by the assigned teacher

for *c* in [*1*, *N*]:

for *m* in *class _ subjects*[*c*]:

A = sum(x[(c,m,h)] for *h* in *range*(*TOTAL _ PERIODS*))

B = sum(y[(c,m,t)] for *t* in *subject _ teachers*[*m*])

A == B * subject_periods[m]

- 3rd: A class can only study one subject at a time

for *c* in [*1*, *N*]:

for *h* in *range*(*TOTAL _ PERIODS*):

sum(x[(c,m,h)] for m in class_subjects[c]) <= 1

- 4th: A teacher cannot teach multiple subjects or classes at a time

for t in $[1,\ T]$:

for h in $range(TOTAL\_PERIODS)$:

```
sum(x[(c,m,h)] * y[(c,m,t)]
    for c in [1, N]
    for m in class_subjects[c]) <= 1
```

- 5th: Enforce continuous periods for each subject in each class

for c in $[1,\ N]$:

for m in $class\_subjects[c]$:

$subject\_duration = subject\_periods[m]$

for $start\_period$ in $range(TOTAL\_PERIODS - subject\_duration)$:

```
if c-m start at start_period:
  for h in range(start_period,start_period+subject_duration)
    x[(c,m,h)]=True
```

3. Integer Programming

In our presentation session, we didn't have this section, but PhD Bui Quoc Trung suggested that we should try Integer Programming. That is why there is "Integer Programming" section in our Report but not in our Powerpoint

a) Introduction

We use the same approach with Constraint Programming, but we need to modify our model.

The variables are defined the same as Constraint Programming.

The "ideas" of the constraints stay the same, we just change how we implement the 4th and 5th constraints.

b) The 4th constraint

In the 4th constraint, we need to model the multiplication:

```
x[(c,m,h)] * y[(c,m,t)]
```

Note that ***x*** and ***y*** are binary variables. In order to represent the multiplication, we create a new binary variable ***z***

z have to satisfy the following conditions

```
z <= x[(c,m,h)]
z <= y[(c,m,t)]
x[(c,m,h)] + y[(c,m,t)] <= z + 1
```

⇨ ***z = x[(c,m,h)] * y[(c,m,t)]***

c) The 5th constraint

This is where the challenge really begins. We have to turn the "if" into mathematic calculations (addition, subtraction, etc)

- We create a variable ***z1***, where

```
z1 = x[(c,m,start_period)] - x[(c,m,start_period - 1)]
```

We realize that ***z1*** can take on the value of ***1*** of ***0*** of ***-1***, and value ***1*** corresponding to the case class-subject *c-m* start at *start_period*.

- We create a variable ***z2***, where

```
z2 = sum(x[(c,m,h)]
         for h in range(start_period,start_period+subject_duration))
```

So z2 is the sum of consecutive periods with the length of subject_durarion

- And this is how we model the "if":

```
z1 + z2 <= subject_duration + 1
-z1 + z2 + subject_duration * (1 - z1) >= subject_duration - 1
```

- Explaination:
  - If class-subject *c-m* starts at *start_period*, then ***z1 = 1***. In order to satisfy those 2 conditions, we found that ***z2 = subject_duration***.
  - If class-subject *c-m* start at *start_period*, then ***z1 = 0*** or ***z1 = -1***, and z2 can be any value.

d) Last words

We tried to run Integer Programming Method for 30 test cases, and we realized that its outputs and runtimes are not much different from Constraint Programming Method. It is not better, but also not worse. And since our time is limited, we decided not to run this method for all 100 test cases, and not to put it into the III. Result section. But you can find its code in our github link.

4. Hill Climbing Method

a) Introduction

Timetable scheduling is a complex combinatorial optimization problem that involves assigning classes, teachers, and time slots while satisfying various constraints. This report presents a solution using the **Hill Climbing** algorithm, a local search technique that iteratively improves the solution by exploring neighboring states.

b) Problem Definition

The objective of the timetable scheduling problem is to assign courses (subjects) to classrooms and teachers while considering the following constraints:

- Each subject has a fixed duration.
- Teachers can only be assigned to subjects they are qualified for.
- A class or a teacher cannot be scheduled for multiple subjects at the same time.

c) Solution Representation

The solution is represented as a mapping:

(Class, Subject) → (Start Slot, Teacher)

A timetable consists of **N** classes, **M** subjects, and **T** teachers.

A total of **60 slots** are available (5 days × 2 sessions per day × 6 slots per session).

d) Validity Check

A function is_valid_assignment(x, y, start_slot, v) is used to verify whether assigning subject y to class x at start_slot with teacher v satisfies all constraints. The key conditions include:

- The assigned slot does not exceed the maximum limit.
- The teacher is qualified for the subject.
- No conflicts exist between the class or teacher.

e) Scoring Function

The quality of a solution is measured by a scoring function:

Compute_score(solution) counts the number of successfully scheduled class-subject pairs.

The higher the score, the better the timetable arrangement.

f) Initial Solution Generation

The algorithm starts with a randomly generated solution:

- Create an empty timetable.
- Iterate through all (Class, Subject) pairs in random order.
- Assign the first available (Slot, Teacher) that satisfies the constraints.
- If no assignment is possible, mark it as unassigned.

g) Neighbor Generation

To explore new solutions, the algorithm generates neighboring solutions using small modifications (moves):

Move Slot: Change the start time for a scheduled subject.

Move Teacher: Assign a different teacher to a scheduled subject.

Assign Unscheduled: Try to assign an unassigned subject.

h) Hill Climbing Algorithm

The algorithm follows these steps:

- Initialize a random solution.
- Iteratively generate a neighboring solution.
- Accept the neighbor if it improves the score; otherwise, discard it.
- Repeat until reaching a predefined iteration limit or a solution with no further improvements.

i) Conclusion

The Hill Climbing algorithm is an effective heuristic for solving the timetable scheduling problem. While it does not guarantee a globally optimal solution, it efficiently finds a feasible schedule that satisfies most constraints. Further enhancements, such as **Simulated Annealing** or **Tabu Search**, can be applied to escape local optima and achieve better results.

5. Tabu Search

a) Introduction

Tabu Search is a metaheuristic optimization algorithm designed to solve complex scheduling problems like assigning teachers to classes and subjects. It works by iteratively improving a solution while avoiding revisiting recent solutions using a "tabu list." For this problem, the algorithm maximizes the number of valid teacher-class assignments while ensuring no schedule overlaps for classes or teachers. This approach effectively handles the problem's constraints and finds an efficient timetable.

b) Data parsing

T: number of teachers

N: number of classes

M: number of subjects

Class_subjects: The dictionary contains a list of subjects to be taught for each class.

Teacher_subjects: The dictionary contains a list of subjects that each teacher can teach.

Subject_periods: The dictionary contains the number of periods required to teach each subject.

c) Initialization

- Start with an empty solution (no assignments).
- Define the total time slots available (5 days × 2 sessions × 6 periods).
- Initialize a tabu list to store recently explored solutions.

d) Neighbor Generation

- Add unassigned class-subject pairs to the schedule by assigning qualified teachers and feasible start times that do not cause conflicts.
- Remove existing assignments to explore alternative solutions.

e) Feasibility Check

- Ensure no overlapping schedules for classes or teachers.
- Verify that teachers are qualified for the assigned subjects.
- Ensure that the subject belongs to the class.

f) Evaluation and Update

- Evaluate solutions based on the number of valid assignments (**K**).
- Update the current solution to the best feasible neighbor not in the tabu list.
- Update the tabu list to avoid revisiting recent solutions.

g) Termination

The algorithm stops after a fixed number of iterations or when no significant improvement is observed.

h) Conclusion

The Tabu Search algorithm is a powerful tool for solving the teacher scheduling problem. By maximizing the number of valid assignments while adhering to all constraints, it demonstrates effectiveness and efficiency. Its iterative approach and exploration strategies make it a strong solution for handling complex scheduling challenges.

III. Result

1. Table of Output

Note :

- We generated 100 test cases, but we only show 40 of thems here because of the limited paper size. The full data can be found in the zip file or our github repo.

- We limited the running time of the program to 600 seconds.

Test Case	CP Output	Greedy Output	Hill Climbing Output	Tabu Output
1	37	37	37	37
2	170	165	163	120
3	287	287	285	41
4	148	138	140	54
5	12	12	12	12
6	200	198	200	21
7	227	226	227	15
8	318	315	318	34
11	37	37	37	37
20	30	30	30	30
21	186	186	186	186
22	127	123	125	125
25	181	172	173	17
26	149	149	149	149
35	343	343	334	343
37	327	324	305	271
41	357	355	342	353
42	151	551	546	434
43	307	307	307	307
44	445	479	439	258
45	473	507	489	230
46	402	415	414	247
47	387	390	370	107
51	325	307	296	232
52	218	205	178	83
53	344	320	295	94
54	404	392	351	191
55	335	311	307	54
57	369	348	325	86
58	321	309	274	96
60	268	245	223	45
61	67	67	67	67
73	7	7	7	0
74	24	24	24	0
75	31	31	31	7
79	18	18	18	5
80	18	18	18	3
86	270	276	253	237
97	69	66	61	57
98	111	106	104	106

2. Table of Runtime

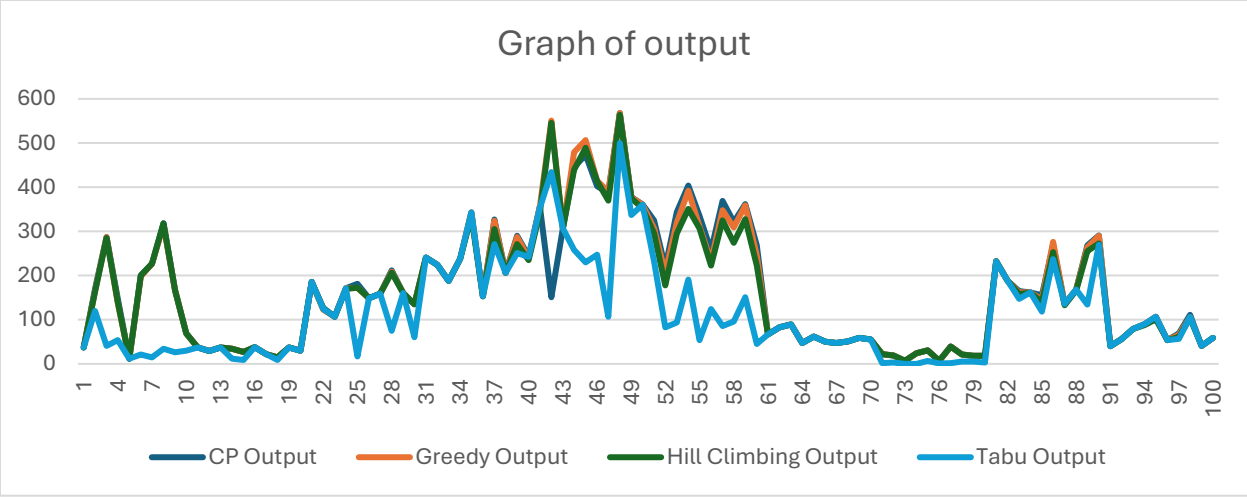
Note :

- We generated 100 test cases, but we only show 40 of thems here because of the limited paper size. The full data can be found in the zip file or our github repo.

- We limited the running time of the program to 600 seconds.

Test Case	CP Runtime (s)	Greedy Runtime (s)	Hill Climbing Runtime (s)	Tabu Runtime (s)
1	5,251142263	0,078108549	0,228711843	1,210924625
2	27,78320813	0,06251955	0,402443647	5,655291796
3	50,56070209	0,078107834	0,589629412	1,682380676
4	34,93080854	0,078071356	0,417876005	2,013370514
5	1,854456902	0,072594643	0,156242847	0,386606455
6	13,34999037	0,079168081	0,463632822	0,937942982
7	20,99235773	0,062485456	0,527311087	0,802368879
8	22,45804906	0,093691587	0,605026007	1,48632884
11	4,654735327	0,062485218	0,214851379	2,216530085
20	3,696883678	0,062483549	0,194109678	1,276706219
21	20,56547379	0,078105688	0,421779871	24,26513839
22	12,94206834	0,121211052	0,542868137	16,99280667
25	68,62367344	0,078107834	0,45302248	1,054455519
26	16,98200274	0,078102589	0,357567787	16,35791111
35	80,3838048	0,156215429	0,948956966	169,2119172
37	611,3814485	0,125005245	0,84941864	107,8673644
41	94,89196897	0,140626431	0,859209538	260,6001241
42	698,5828869	0,343670368	1,702727079	600,7271011
43	120,1676693	0,140592813	0,546748638	252,6556048
44	630,3616905	0,234321356	1,405923843	183,9097478
45	632,4897728	0,249941111	1,399215937	191,7585514
46	631,9114125	0,281184435	1,431930065	278,4369586
47	622,7687268	0,187452078	1,046663046	113,6015992
51	609,2307379	0,109380722	0,882246256	63,02930069
52	606,0979669	0,093729496	0,718579292	3,827773809
53	610,4023204	0,124973059	1,015388489	5,182028294
54	612,7894485	0,156213284	1,343436241	29,62953186
55	608,824013	0,109387875	0,781088829	2,759793043
57	611,2311437	0,124969721	1,015424728	5,165403843
58	611,3170543	0,124971151	0,968519926	9,494735003
60	607,8987844	0,109383821	0,859196663	2,015125751
61	33,96372628	0,062489271	0,374949932	27,84221482
73	2,113224506	0,062523127	0,218734741	0,078073025
74	2,423325777	0,062483311	0,26672411	0,093726635
75	2,900397062	0,078106403	0,265561342	0,328047752
79	2,187021971	0,06248641	0,234319925	0,198573112
80	2,140156507	0,078140259	0,249939919	0,124967575
86	614,0216429	0,140588284	0,906072617	199,7836549
97	602,8857548	0,078105211	0,390533686	12,25394464
98	14,7201035	0,078104973	0,468637228	40,94082022

3. Graph of Output



4. Conclusion

- Exact Method (Constraint Programming) has the most optimized output, but its runtime can be very long and inconsistent (Sometimes it is fast enough, sometimes it is slow, even when the output is not really big. It depends on how the testcase was constructed).
- Greedy and Hill Climbing Method have slightly worse output than Exact Method, but they have much faster runtime and much more consistent.
- Tabu Search Method seems to be the worst one with the worst output and slow runtime. We believe this is due to our bad implementation.
- In conclusion, if you want to have the best output, and don't care how slow the program runs, use Exact Method. If you want to have a good enough output in a short period of time, use Greedy or Hill Climbing Method.

IV. Work assignment

Constraint Programming, Integer Programming	Phạm Đức Duy
Greedy code	Ngô Mạnh Hiếu
Hill Climbing code	Dương Ngô Hoàng Vũ
Tabu Search code	Hoàng Quốc Huy
Testing, Analyzing the output	Phạm Đức Duy
Presentation	
Slide	Duy, Hiếu, Vũ, Huy
Report	Duy, Hiếu, Vũ, Huy

V. Our source code and testcases

Our source code and testcases can be found in the link below

https://github.com/ImmortalZeus/Fundamentals-Of-Optimization