Discrete Optimization Project

Examination Timetabling Problem

Mahsa Sedaghatinia

VR481427

1. Introduction

2. Basic Model

3. Advanced Model + Advanced Constraints

4. Results

Introduction

Introduction - Data

exams: $E = \{e1, e2, . . . , en\}$

students: $S = \{s1,s2, . . . ,sm\}$. Each student is enrolled in a non-empty subset of exams.

time slots: $Ts = \{t1 \leq t2 \leq . . . \leq tT \}$

$ne1,e2$ = number of students enrolled in both e1 and e2; if $ne1,e2 > 0$ then e1 and e2 are conflicting

Introduction - Constraints

each exam is scheduled exactly once during the examination period;

two exams $e1, e2 \in E$ are called conflicting if they have at least one student enrolled in both, i.e., if $n_{e1,e2} > 0$.

two conflicting exams are not scheduled in the same time-slot;

the total penalty resulting from the created timetable is minimized

given two exams $e1, e2 \in E$ scheduled at distance i of time-slots, with $1 \leq i \leq 5$, the relative penalty is $2^{(5-i)} \cdot (n_{e1,e2}) / |S|$.

Basic Model

Basic Model - Variables

$$X_{ij} = \text{student } s_i \text{ is enrolled in exam } e_j\ ,\ X_{ij} \in \{0, 1\}, \quad \exists\ s_i \in S, \quad e_j \in E$$

```
x = {}
for exam_id in exams:
    for time_slot in range(1, num_time_slots + 1):
        x[exam_id, time_slot] = model.addVar(vtype=GRB.BINARY) # 1 if exam e is scheduled in timeslot t, 0 otherwise
```

Basic Model - Variables

$$Y_{ij} = \text{exam } e_i \text{ takes place at timeslot } t_j \text{ , } Y_{ij} \in \{0, 1\}, \quad \exists\, e_i \in E, \quad t_j \in T$$

```
y = {}
penalty_distance = 5  # Maximum time slot difference for penalty calculation

for distance in range(1, penalty_distance + 1):
    y[exam1, exam2, distance] = model.addVar(vtype=GRB.BINARY)
```

Basic Model - Constraints

$\sum_{i=1}^{n}\sum_{j=1}^{n} Y_{ij} = 1$: each exam is scheduled exactly once during the examination period (= sum of all the timeslots).

```
# Basic Constraint: Ensure that each exam is scheduled exactly once
for exam_id in exams:
    model.addConstr(
        gp.quicksum(x[exam_id, t] for t in range(1, num_time_slots + 1)) == 1
    )
```

Basic Model - Constraints

$\sum_{i=m,p} Y_{ij} = 1$: conflicting exams e_m and e_p can't take place in the same time-slots.

```
# Basic Constraint: Ensure conflicting exam pairs are not scheduled together
for exam1, exam2 in conflicting_pairs:
    for time_slot in range(1, num_time_slots + 1):
        model.addConstr(
            x[exam1, time_slot] + x[exam2, time_slot] <= 1
        )
```

Basic Model - Constraints

By adding this additional constraint, the solutions found have lower penalty

```
# Basic Constraint: Ensure conflicting exam pairs are not scheduled together
for exam1, exam2 in conflicting_pairs:
    for time_slot in range(1, num_time_slots + 1):
        model.addConstr(
            x[exam1, time_slot] + x[exam2, time_slot] <= 1
        )
        y = {}
        penalty_distance = 5  # Maximum time slot difference for penalty calculation

        for distance in range(1, penalty_distance + 1):
            y[exam1, exam2, distance] = model.addVar(vtype=GRB.BINARY)

            # Add constraints to enforce the relationship between y and x variables
            for time_slot in range(1, num_time_slots - distance + 1):
                model.addConstr(
                    y[exam1, exam2, distance] >= x[exam1, time_slot] + x[exam2, time_slot + distance] - 1
                )
```

Basic Model - Objective Function

If 2 exams e_m and e_p are conflicting, we have:

$Y_{mj} = 1, \quad \exists\, m \in E$, for the first exam

$Y_{p,j+k} = 1 \quad \exists\, p \in E \; and \; p \neq m, \; \exists\, 1 \leq k < 5$, for the next exam and the

penalty is: $Z_{m,\,p,\,j,\,j+k} = 2^{5-k} \cdot \frac{n_{e_{m,p}}}{|S|}$

__**Objective function**__

$$\text{Min} \sum_{j=0}^{n} \sum_{i=0}^{n} Z_{m,p,j,\,j+k}$$

Basic Model - Objective Function

```
# Calculate the objective function to optimize
objective_expr = 0
for exam1 in exams:
    for exam2 in exams:
        if exam1 != exam2:
            shared_students = calculate_common_enrollment(exam1,exam2)

            if shared_students > 0:
                for t1 in range(1, num_time_slots + 1):
                    for i in range(1, 6):
                        t2 = (t1 + i) if (t1 + i) <= num_time_slots else (t1 + i - num_time_slots)
                        penalty = ((2 ** (5 - i)) * (shared_students)) / len(students)
                        #print("penalty: " + str(penalty))
                        objective_expr += penalty

# Set the objective to minimize the total penalty
model.setObjective(objective_expr, GRB.MINIMIZE)
```

Advanced Model
+
Advanced Constraints

Advanced Model - New Equity Measure

Maximum distance: This measure is the maximum number of time-slots between any two conflicting exams. A timetable with a high maximum distance is more equitable for students, as it means that they will have more time between exams.

Advanced Model - Implementation

```python
# Add constraints for the maximum distance
for i in range(1, len(exams)):
    exam1 = exams[i]
    for j in range(i+1, len(exams)):
        exam2 = exams[j]
        shared_students = len(exam_to_students.get(exam1, set()) & exam_to_students.get(exam2, set()))
        if shared_students > 0:
            for t1 in range(1, num_time_slots + 1):
                for t2 in range(1, num_time_slots + 1):
                    model.addConstr(
                        z >= abs(t1 - t2) * x[exam1, t1] * x[exam2, t2]
                    )
```

Advanced Model - Objective Function

```python
# Set the objective to minimize the total penalty while maximizing the maximum distance
penalty_weight = 1  # Adjust this weight factor based on the importance of minimizing penalty
distance_weight = 1  # Adjust this weight factor based on the importance of maximizing distance
objective_expr = 0

for i in range(1, len(exams)):
    exam1 = exams[i]
    for j in range(i+1, len(exams)):
        if exam1 != exam2:
            shared_students = len(exam_to_students.get(exam1, set()) & exam_to_students.get(exam2, set()))
            if shared_students > 0:
                for t1 in range(1, num_time_slots + 1):
                    for t2 in range(1, num_time_slots + 1):
                        objective_expr += penalty_weight * (shared_students / len(students)) * x[exam1, t1] * x[exam2, t2]
                        objective_expr += distance_weight * abs(t1 - t2) * x[exam1, t1] * x[exam2, t2]  # Use + instead of -

# Set the objective in the model
model.setObjective(objective_expr, GRB.MINIMIZE)
```

Advanced Constraints - Introduction

At most 3 consecutive time slots can have conflicting exams.

If two consecutive time slots contain conflicting exams, then no conflicting exam can be scheduled in the next 3 time slots.

Include a bonus profit each time no conflicting exams are scheduled for 6 consecutive time slots.

Change the constraints that impose that no conflicting exams can be scheduled in the same time slot. Instead, impose that at most 3 conflicting pairs can be scheduled in the same time slot.

Advanced Constraints - Implementation

At most 3 consecutive time slots can have conflicting exams.

```
# Advanced Constraint: At most 3 consecutive time slots with conflicts
for time_slot in range(1, num_time_slots - 2):  # Update the range here
    model.addConstr(
        y[time_slot] + y[time_slot + 1] + y[time_slot + 2] <= 3
    )
```

Advanced Constraints - Implementation

If two consecutive time slots contain conflicting exams, then no conflicting exam can be scheduled in the next 3 time slots.

→ NOT IMPLEMENTED/DOESN’T WORK

Advanced Constraints - Implementation

Include a bonus profit each time no conflicting exams are scheduled for 6 consecutive time slots.

```
objective_expr = 0
# Advanced Constraint: Include a bonus profit each time no conflicting exams are scheduled for 6 consecutive time slots
bonus_reward = 100  # Adjust this value as needed
for time_slot in range(1, num_time_slots - 5):  # Update the range here
    model.addConstr(
        gp.quicksum(y[t] for t in range(time_slot, time_slot + 6)) <= 5  # At most one conflict in 6 consecutive time slots
    )
    objective_expr += bonus_reward * (1 - gp.quicksum(y[t] for t in range(time_slot, time_slot + 6)))
```

Advanced Constraints - Implementation

Change the constraints that impose that no conflicting exams can be scheduled in the same time slot. Instead, impose that at most 3 conflicting pairs can be scheduled in the same time slot.

```
# Advanced Constraint: At most 3 consecutive time slots with conflicts
for time_slot in range(1, num_time_slots - 2):  # Update the range here
    model.addConstr(
        y[time_slot] + y[time_slot + 1] + y[time_slot + 2] <= 3
    )
```

Results

Results

instances	basic model	advanced model
instance 01	187.702	186.216
instance 02	53.095	52.049
instance 03	59.575	53.20
instance 04	12.988	12.745
instance 05	27.211	33.738
instance 06	No feasible solution	No feasible solution
instance 07	18.902	18.902
instance 08	45.175	45.345
instance 09	22.009	21.599
instance 10	No feasible solution	No feasible solution
instance 11	No feasible solution	No feasible solution

No feasible solutions for all the instances when adding the additional constraints.