Piano Major Practice Room Scheduling Project

Julia Schraml, Joyce Qiu, Youwei Jiang, Mia Yuan

21-393 Fall 2024

Introduction

Effective resource allocation in educational settings is critical to improving the learning experience and maximizing utility. In a professional environment such as a conservatory or university, managing instrument rooms and ensuring their effectiveness is a common logistical challenge. In this project, we specifically focused on optimizing the allocation of piano-equipped instrument rooms for piano majors at CMU. This aims to ensure that each student receives adequate practice time while making the best use of resources.

The idea for this project came from the personal experience of two of the team members, who are currently taking piano lessons at CMU, where the existing system of using the piano practice rooms is based on a first-come, first-served basis so that if the room is empty, it can be used without a reservation. However, our team members reported that when they needed to use the piano rooms, they were usually not available, leading to significant frustration and inefficiency. This situation prompted us to formulate a scheduling solution to solve this problem by allocating practice time more efficiently and equitably.

Since students usually have different class schedules and other activities, their practice times can vary greatly. Based on such variations and the limited number of piano rooms, schools need an effective scheduling program that ensures equity while increasing the efficiency of piano room use. Specifically, we will consider a situation where the room is used from 8 a.m. to 8 p.m. in 24 slots, and each student needs at least one hour of practice time per day, preferably in consecutive slots.

This scheduling problem can be solved from different perspectives, including mathematical optimization and heuristic techniques. In this project, we explored the integer programming model, which provides the optimal solution, and the greedy heuristic algorithm, which provides a faster alternative, although potentially non-optimal. The insights gained from solving this problem can provide valuable guidance on similar distributional issues in educational settings and other areas in the future.

By comparing these approaches, we aim to understand the trade-offs between optimality and computational efficiency in solving this type of scheduling problem. The insights gained from this project can provide valuable guidance for similar allocation issues in educational settings and beyond.

Assumptions

To model the problem effectively, we made the following assumptions:

1. There are 15 students majoring in piano performance, and 10 rooms with pianos available.
2. All rooms are identical; students do not have any preference for specific rooms.
3. Rooms are available from 8 am to 8 pm, divided into 24 half-hour time slots to accommodate varying availability, such as classes ending at half-past the hour.
4. These rooms are not used for any other purpose during the available hours, so they are always open for student practice.
5. Each student needs at least 1 hour (two consecutive half-hour slots) to practice daily.
6. Students can only practice during their available time slots, which vary based on their schedules.
7. Each student can be assigned to at most one room per day, and one student can be assigned to each room at any given time slot.

Initial Program

Variables

- $x_{i,j,t,d} \in \{0, 1\}$: 1 if student i is assigned to room j at time t on day d, otherwise 0.
- $y_{i,j,d} \in \{0, 1\}$: 1 if student i is assigned to room j on day d, otherwise 0.
- $T \in \{0, 23\}$: half-hour slots from 8 a.m. to 8 p.m.
- $D \in \{0, 5\}$: days of the week from Monday to Friday

Objective

Maximize the total assigned hours across all students:

$$\text{maximize} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \sum_{t=0}^{T-1} \sum_{d=0}^{D-1} x_{i,j,t,d}$$

Constraints

- Each room can only have one student assigned at any time slot on any day:

$$\sum_{i=0}^{n-1} x_{i,j,t,d} \leq 1 \quad \forall j \in \{0, \ldots, m-1\}, \forall t \in \{0, \ldots, T-1\}, \forall d \in \{0, \ldots, D-1\}$$

- Students can only be assigned during their available time slots:

$$x_{i,j,t,d} = 0 \quad \text{if } t \notin \text{availability}_{i,d} \quad \forall i \in \{0, \ldots, n-1\}, \forall j \in \{0, \ldots, m-1\}, \forall t \in \{0, \ldots, T-1\}, \forall d \in \{0, \ldots, D-1\}$$

- If a student i is assigned to any time slot in room j on day d, then $y_{i,j,d} = 1$:

$$\sum_{t=0}^{T-1} x_{i,j,t,d} \leq T \cdot y_{i,j,d} \quad \forall i \in \{0, \ldots, n-1\}, \forall j \in \{0, \ldots, m-1\}, \forall d \in \{0, \ldots, D-1\}$$

- Each student can be assigned to at most one room per day:

$$\sum_{j=0}^{m-1} y_{i,j,d} \leq 1 \quad \forall i \in \{0, \ldots, n-1\}, \forall d \in \{0, \ldots, D-1\}$$

- Each student must be assigned at least 1 hour (2 consecutive half-hour slots) across all days:

$$\sum_{j=0}^{m-1} \sum_{d=0}^{D-1} \sum_{t \in \text{availability}_{i,d}} x_{i,j,t,d} \geq 2 \quad \forall i \in \{0, \ldots, n-1\}$$

- Each room can have at most 24 half-hour slots assigned per day:

$$\sum_{i=0}^{n-1} \sum_{t=0}^{T-1} x_{i,j,t,d} \leq 24 \quad \forall j \in \{0, \ldots, m-1\}, \forall d \in \{0, \ldots, D-1\}$$

Non-Optimal Program

We used Greedy Algorithm as our heuristic to assign rooms to students by sequentially selecting students based on the least total available time and filling available rooms until constraints are violated. The Greedy Algorithm does not guarantee an optimal solution but can quickly produce a satisfactory allocation of rooms to students.

```python
def hour_range_to_slots(hour_range):
    return [i for hour in hour_range for i in range(hour * 2, (hour + 1) * 2) if i < 24]

def assign_rooms(rooms, days_of_week, availability):
    # Initialize room availability with None for 24 half-hour slots from 8am to 8pm
    room_availability = {room: {day: [None] * 24 for day in days_of_week} for room in rooms}

    for student_id, days_avail in availability.items():
        for day, time_ranges in days_avail.items():
            for time_range in time_ranges:
                slots = hour_range_to_slots(time_range)
                for index in range(len(slots) - 1):
                    slot = slots[index]
                    next_slot = slots[index + 1]
                    for room in rooms:
                        room_slots = room_availability[room][day]
                        if room_slots[slot] is None and room_slots[next_slot] is None:
                            room_slots[slot] = f"Student {student_id}"
                            room_slots[next_slot] = f"Student {student_id}"
                            print(f"Student {student_id} assigned to Room {room} on {day} from {slot//2 + 8}:{('30' if slot % 2 else '00')} to {(next_slot//2 + 9):00}")
                            break
                    else:
                        continue
                    break

    return room_availability


def print_schedule(room_availability):
    for room_id, days in room_availability.items():
        print(f"\nRoom {room_id} Schedule:")
        for day, slots in days.items():
            print(f"  {day}:")
            for i, student in enumerate(slots):
                status = 'Free' if student is None else student
                print(f"    {i//2 + 8}:{('30' if i % 2 else '00')} - {status}")

rooms = [1, 2, 3, 4, 5, 6, 7, 8]
days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']

availability = {
    0: {  # Student 0's availability
        "Monday": [range(0, 6), range(8, 10), range(15, 16), range(20, 21)],
        "Tuesday": [range(4, 9), range(19, 20)],
    },
    1: {  # Student 1's availability
        "Monday": [range(1, 3), range(8, 9)],
        "Tuesday": [range(0, 5), range(10, 12)],
    },
    2: {  # Student 2's availability
        "Monday": [range(2, 5), range(14, 18)],
        "Wednesday": [range(0, 6), range(7, 10)],
    },
    3: {  # Student 3's availability
        "Monday": [range(8, 14), range(14, 18)],
        "Tuesday": [range(0, 4), range(20, 24)],
        "Wednesday": [range(4, 6), range(16, 24)],
    },
    4: {  # Student 4's availability
        "Monday": [range(4, 10), range(14, 24)],
        "Tuesday": [range(6, 8), range(14, 18)],
    },
    5: {  # Student 5's availability
        "Monday": [range(12, 20)],
        "Tuesday": [range(0, 4), range(10, 18)],
    },
    6: {  # Student 5’s availability
        "Tuesday": [range(4, 6), range(16, 24)],
        "Wednesday": [range(2, 4), range(12, 16), range(20, 24)],
    },
    7: {  # Student 7’s availability
        "Monday": [range(4, 14), range(18, 22)],
        "Tuesday": [range(12, 24)],
    },
    8: {  # Student 8’s availability
        "Wednesday": [range(6, 8), range(12, 14), range(16, 24)],
        "Thursday": [range(0, 8), range(12, 24)],
        "Friday": [range(0, 3), range(6, 10), range(16, 24)]
    },
    9: {  # Student 9’s availability
        "Monday": [range(0, 2), range(6, 9), range(12, 24)],
        "Tuesday": [range(0, 4), range(6, 15), range(18, 24)],
        "Wednesday": [range(0, 2), range(6, 8), range(15, 24)]
    },
    10: {  # Student 10’s availability
        "Thursday": [range(0, 6), range(8, 15), range(18, 24)],
        "Friday": [range(0, 2), range(6, 10)],
    },
    11: {  # Student 11’s availability
        "Monday": [range(3, 15), range(18, 24)],
        "Tuesday": [range(0, 9), range(12, 15), range(18, 24)],
        "Wednesday": [range(3, 6), range(8, 24)],
        "Thursday": [range(0, 9), range(12, 15), range(18, 24)],
        "Friday": [range(3, 8), range(10, 12), range(16, 24)]
    },
    12: {  # Student 12’s availability
        "Monday": [range(0, 4), range(8, 9), range(15, 24)],
        "Tuesday": [range(3, 9), range(15,24)],
        "Wednesday": [range(0, 4), range(8, 9), range(15, 24)],
        "Thursday": [range(3, 9), range(15,24)],
        "Friday": [range(0, 2), range(8, 24)]
    },
    13: {  # Student 13’s availability
        "Monday": [range(0, 4), range(12, 15), range(18, 22)],
        "Tuesday": [range(0, 9), range(12,24)],
        "Wednesday": [range(0, 4), range(12, 15), range(18, 22)],
        "Thursday": [range(0, 9), range(12,24)],
        "Friday": [range(0, 2), range(4, 6), range(9, 12), range(14, 24)]
    },
    14: {  # Student 14’s availability
        "Monday": [range(0, 9), range(18, 22)],
        "Tuesday": [range(0, 3), range(9,24)],
        "Wednesday": [range(0, 9), range(18, 22)],
        "Thursday": [range(0, 3), range(9,24)],
        "Friday": [range(0, 4), range(6, 9), range(12, 24)]
    },
    15: {  # Student 15’s availability
        "Monday": [range(0, 3), range(6, 8), range(10, 12), range(18, 24)],
        "Tuesday": [range(0, 6), range(9,12), range(15, 24)],
        "Wednesday": [range(0, 3), range(6, 8), range(10, 12), range(18, 24)],
        "Thursday": [range(0, 6), range(9,12), range(15, 24)],
        "Friday": [range(0, 14), range(16, 24)]
    },
}

room_availability = assign_rooms(rooms, days_of_week, availability)

print_schedule(room_availability)
```

Non-Optimal Results

Student	Monday	Tuesday	Wednesday	Thursday	Friday
0	Room 1: 0–2, 16–18	Room 1: 8–10			
1	Room 1: 2–4, Room 2: 16–18	Room 1: 0–2, 20–22			
2	Room 1: 4–6		Room 1: 0–2, 14–16		
3	Room 3: 16–18	Room 2: 0–2	Room 1: 8–10		
4	Room 1: 8–10	Room 1: 12–14			
5		Room 3: 0–2, Room 2: 20–22			
6		Room 2: 8–10	Room 1: 4–6		
7	Room 2: 8–10				
8			Room 1: 12–14	Room 1: 0–2	Room 1: 0–2, 12–14
9	Room 2: 0–2, Room 1: 12–14	Room 4: 0–2, Room 2: 12–14	Room 2: 0–2, 12–14		
10				Room 2: 0–2, Room 1: 16–18	Room 2: 0–2, 12–14
11	Room 1: 6–8	Room 5: 0–2	Room 1: 6–8, 16–18	Room 3: 0–2	Room 1: 6–8, 20–22
12	Room 3: 0–2, Room 4: 16–18	Room 1: 6–8	Room 3: 0–2, Room 2: 16–18	Room 1: 6–8	Room 3: 0–2, Room 1: 16–18
13	Room 4: 0–2	Room 6: 0–2	Room 4: 0–2	Room 4: 0–2	Room 4: 0–2, Room 1: 8–10, 16–18
14	Room 5: 0–2	Room 7: 0–2, Room 1: 16–18	Room 5: 0–2	Room 5: 0–2, Room 1: 16–18	Room 5: 0–2, Room 3: 12–14
15	Room 6: 0–2, Room 2: 12–14, Room 1: 20–22	Room 8: 0–2, Room 2: 16–18	Room 6: 0–2, Room 3: 12–14, Room 1: 20–22	Room 6: 0–2, Room 2: 16–18	Room 6: 0–2

Optimal Program

This program uses integer linear programming and Gurobi to find the optimal solution. It translates the scheduling problem into a mathematical model with an objective functions and multiple constraints.

```python
from gurobipy import Model, GRB, quicksum

n = 16  # number of students
m = 10  # number of rooms
T = 24  # number of half-hour slots in a day (0 to 23 for 8 am to 8 pm)
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]  # Days of the week

# Define availability for each student for each day
availability = {
    0: {  # Student 0's availability
        "Monday": [range(0, 6), range(8, 10), range(15, 16), range(20, 21)],
        "Tuesday": [range(4, 9), range(19, 20)],
    },
    1: {  # Student 1's availability
        "Monday": [range(1, 3), range(8, 9)],
        "Tuesday": [range(0, 5), range(10, 12)],
    },
    2: {  # Student 2's availability
        "Monday": [range(2, 5), range(14, 18)],
        "Wednesday": [range(0, 6), range(7, 10)],
    },
    3: {  # Student 3's availability
        "Monday": [range(8, 14), range(14, 18)],
        "Tuesday": [range(0, 4), range(20, 24)],
        "Wednesday": [range(4, 6), range(16, 24)],
    },
    4: {  # Student 4's availability
        "Monday": [range(4, 10), range(14, 24)],
        "Tuesday": [range(6, 8), range(14, 18)],
    },
    5: {  # Student 5's availability
        "Monday": [range(12, 20)],
        "Tuesday": [range(0, 4), range(10, 18)],
    },
    6: {  # Student 5's availability
        "Tuesday": [range(4, 6), range(16, 24)],
        "Wednesday": [range(2, 4), range(12, 16), range(20, 24)],
    },
    7: {  # Student 7's availability
        "Monday": [range(4, 14), range(18, 22)],
        "Tuesday": [range(12, 24)],
    },
    8: {  # Student 8's availability
        "Wednesday": [range(6, 8), range(12, 14), range(16, 24)],
        "Thursday": [range(0, 8), range(12, 24)],
        "Friday": [range(0, 3), range(6, 10), range(16, 24)]
    },
    9: {  # Student 9's availability
        "Monday": [range(0, 2), range(6, 9), range(12, 24)],
        "Tuesday": [range(0, 4), range(6, 15), range(18, 24)],
        "Wednesday": [range(0, 2), range(6, 8), range(15, 24)]
    },
    10: {  # Student 10's availability
        "Thursday": [range(0, 6), range(8, 15), range(18, 24)],
        "Friday": [range(0, 2), range(6, 10)],
    },
    11: {  # Student 11's availability
        "Monday": [range(3, 15), range(18, 24)],
        "Tuesday": [range(0, 9), range(12, 15), range(18, 24)],
        "Wednesday": [range(3, 6), range(8, 24)],
        "Thursday": [range(0, 9), range(12, 15), range(18, 24)],
        "Friday": [range(3, 8), range(10, 12), range(16, 24)]
    },
    12: {  # Student 12's availability
        "Monday": [range(0, 4), range(8, 9), range(15, 24)],
        "Tuesday": [range(3, 9), range(15,24)],
        "Wednesday": [range(0, 4), range(8, 9), range(15, 24)],
        "Thursday": [range(3, 9), range(15,24)],
        "Friday": [range(0, 2), range(8, 24)]
    },
    13: {  # Student 13's availability
        "Monday": [range(0, 4), range(12, 15), range(18, 22)],
        "Tuesday": [range(0, 9), range(12,24)],
        "Wednesday": [range(0, 4), range(12, 15), range(18, 22)],
        "Thursday": [range(0, 9), range(12,24)],
        "Friday": [range(0, 2), range(4, 6), range(9, 12), range(14, 24)]
    },
    14: {  # Student 14's availability
        "Monday": [range(0, 9), range(18, 22)],
        "Tuesday": [range(0, 3), range(9,24)],
        "Wednesday": [range(0, 9), range(18, 22)],
        "Thursday": [range(0, 3), range(9,24)],
        "Friday": [range(0, 4), range(6, 9), range(12, 24)]
    },
    15: {  # Student 15's availability
        "Monday": [range(0, 3), range(6, 8), range(10, 12), range(18, 24)],
        "Tuesday": [range(0, 6), range(9,12), range(15, 24)],
        "Wednesday": [range(0, 3), range(6, 8), range(10, 12), range(18, 24)],
        "Thursday": [range(0, 6), range(9,12), range(15, 24)],
        "Friday": [range(0, 14), range(16, 24)]
    },
}

# Create a model
model = Model("Piano Practice Room Assignment")

# Define decision variables: x[i, j, t, d] -> 1 if student i is assigned to room j at time t on day d
x = model.addVars(
    n, m, T, len(days), vtype=GRB.BINARY, name="x"
)

# Set objective: Maximize total assignment hours for all students
model.setObjective(
    quicksum(x[i, j, t, d] for i in range(n) for j in range(m) for t in range(T) for d in range(len(days))),
    GRB.MAXIMIZE,
)

# Constraints

# Each room j has a maximum of 12 half-hour slots available per day
for j in range(m):
    for d in range(len(days)):
        model.addConstr(
            quicksum(x[i, j, t, d] for i in range(n) for t in range(T)) <= 24,
            f"RoomCapacity_{j}_Day{d}",
        )

# Each student i needs at least 1 hour (2 consecutive half-hour slots) across all days they are available
for i in range(n):
    model.addConstr(
        quicksum(
            x[i, j, t, d]
            for j in range(m)
            for d in range(len(days))
            for day_range in availability.get(i, {}).get(days[d], [])
            for t in day_range
        ) >= 2,
        f"MinPractice_{i}",
    )

# Room capacity constraint: only one student per room at each time t on each day
for j in range(m):
    for t in range(T):
        for d in range(len(days)):
            model.addConstr(
                quicksum(x[i, j, t, d] for i in range(n)) <= 1,
                f"RoomLimit_{j}_Time{t}_Day{d}",
            )

```

```python
# Assignments outside a student's available time are not allowed
for i in range(n):
    for j in range(m):
        for d in range(len(days)):
            for t in range(T):
                # Check if time t is not in any of the availability ranges for student i on day d
                if not any(t in day_range for day_range in availability.get(i, {}).get(days[d], [])):
                    model.addConstr(x[i, j, t, d] == 0, f"Availability_{i}_Room{j}_Time{t}_Day{d}")

# Define new binary variables: y[i, j, d] -> 1 if student i is assigned to room j on day d
y = model.addVars(n, m, len(days), vtype=GRB.BINARY, name="y")

# Link x[i, j, t, d] with y[i, j, d]: If student i is assigned to room j at any time, y[i, j, d] must be 1
for i in range(n):
    for j in range(m):
        for d in range(len(days)):
            model.addConstr(
                quicksum(x[i, j, t, d] for t in range(T)) <= T * y[i, j, d],
                f"Link_x_y_{i}_{j}_Day{d}",
            )

# Ensure each student is assigned to at most one room per day
for i in range(n):
    for d in range(len(days)):
        model.addConstr(
            quicksum(y[i, j, d] for j in range(m)) <= 1,
            f"OneRoomPerDay_{i}_Day{d}",
        )


# Solve the model
model.optimize()

# Function to find intervals of consecutive time slots
def get_intervals(assignments):
    intervals = []
    start = assignments[0]
    for i in range(1, len(assignments)):
        if assignments[i] != assignments[i - 1] + 1:
            intervals.append((start, assignments[i - 1]))
            start = assignments[i]
    intervals.append((start, assignments[-1]))
    return intervals

# Output the solution with intervals
print("\n--------------------Solution--------------------")
for i in range(n):
    for j in range(m):
        for d in range(len(days)):
            assigned_times = [
                t for t in range(T) if x[i, j, t, d].x > 0.5
            ]
            if assigned_times:
                intervals = get_intervals(assigned_times)
                # Filter out intervals where start == end
                intervals = [interval for interval in intervals if interval[0] != interval[1]]
                for interval in intervals:
                    print(f"Student {i} assigned to Room {j} from {interval[0]} to {interval[1]} on {days[d]}")
print(f"Optimal total assigned hours: {model.ObjVal}")
```

Optimal Results

Student	Monday	Tuesday	Wednesday	Thursday	Friday
0	Room 4: 0–5, 8–9	Room 2: 4–8			
1	Room 2: 1–2	Room 3: 0–4, 10–11			
2	Room 9: 3–4, 14–17		Room 0: 0–5, 7–9		
3	Room 6: 8–17	Room 7: 0–3, 20–23	Room 1: 4–5, 16–23		
4	Room 8: 4–9, 14–23	Room 7: 6–7, 14–17			
5	Room 2: 12–19	Room 2: 0–3, 10–17			
6		Room 1: 4–5, 16–23	Room 7: 2–3, 12–15, 20–23		
7	Room 7: 4–13, 18–21	Room 3: 12–23			
8			Room 2: 6–7, 12–13, 16–23	Room 3: 0–7, 12–23	Room 5: 0–2, 6–9, 16–23
9	Room 5: 0–1, 6–8, 12–23	Room 6: 0–3, 6–14, 18–23	Room 4: 0–1, 6–7, 15–23		
10				Room 1: 0–5, 8–14, 18–23	Room 4: 0–1, 6–9
11	Room 0: 3–14, 18–23	Room 9: 0–8, 12–14, 18–23	Room 5: 3–5, 8–23	Room 7: 0–8, 12–14, 18–23	Room 1: 3–7, 10–11, 16–23
12	Room 3: 0–3, 15–23	Room 8: 3–8, 15–23	Room 3: 0–3, 15–23	Room 8: 3–8, 15–23	Room 8: 0–1, 8–23
13	Room 1: 0–3, 12–14, 18–21	Room 4: 0–8, 12–23	Room 8: 0–3, 12–14, 18–21	Room 9: 0–8, 12–23	Room 6: 0–1, 4–5, 9–11, 14–23
14	Room 6: 0–7, 18–21	Room 5: 0–2, 9–23	Room 9: 0–8, 18–21	Room 4: 0–2, 9–23	Room 7: 0–3, 6–8, 12–23
15	Room 9: 0–2, 6–7, 10–11, 18–23	Room 0: 0–5, 9–11, 15–23	Room 6: 0–2, 6–7, 10–11, 18–23	Room 0: 0–5, 9–11, 15–23	Room 0: 0–13, 16–23

Discussion of Results

As we can see from the two tables, there are some major differences between the two schedules.

- The non-optimal program has uneven assignment patterns; some students have very light schedules, while others are more heavily assigned. Meanwhile, the optimal program seems to provide a more balanced schedule, with most students having assignments on multiple days.
- The non-optimal program has significantly more blank cells, meaning students often go multiple days without assignments, whereas the optimal program reduces empty days by spreading assignments more evenly.
- The non-optimal program has more gaps (empty cells) in the schedule, suggesting fewer total hours assigned overall. The optimal solution appears more densely populated, indicating a higher total number of assigned hours for students.

The results of this project showed that both methods can effectively handle the diverse availability and requirements of students, ensuring equitable access to practice resources. The models minimized idle time slots and maximized resource utilization, proving their utility in addressing real-world scheduling problems.

Conclusion

This study addressed the piano room scheduling problem by formulating and implementing both mathematical optimization and heuristic methods. The proposed models aimed to allocate limited piano rooms to piano majors efficiently and equitably, addressing constraints such as individual availability, room capacity, and daily minimum practice requirements.

Through the use of integer programming, we achieved optimal solutions that maximize room utilization and student satisfaction. This approach demonstrated the effectiveness of mathematical optimization in resolving resource allocation challenges. However, it also highlighted limitations in scalability, as solving larger instances of the problem required significant computational resources.

To address these limitations, we developed a complementary greedy algorithm. While this heuristic method did not guarantee optimal solutions, it provided a practical and computationally efficient alternative, especially for larger datasets. The algorithm successfully assigned students to rooms in a way that respected key constraints and maintained fairness.

Limitations and Future Improvements

Since this research was conducted over a short period of time, there are improvements that could be made to this project, such as:

1. Allowing students to specify their preferred rooms (e.g., based on proximity, equipment quality, or personal comfort). This might be because a student prefers a specific piano room because it has a better instrument.

2. Allowing students to specify their preferred time slots (e.g., some students may prefer mornings or evenings). This might be because a student prefers to practice in the morning for better focus.

3. Ensuring fairness by evenly distributing room time across students, especially if room availability is limited. This would prevent a few students from monopolizing the most desirable slots or rooms.

4. Allowing rooms to have different capacities (e.g., one room can host 2 students at once for group practice). This is desirable because certain practice rooms can accommodate duets or group sessions.

5. Not splitting a student's practice time across non-consecutive slots or multiple rooms on the same day. This would encourage more efficient practice schedules.

6. Give priority to certain students (e.g., those preparing for an upcoming recital or competition). This would ensure that students with urgent needs are prioritized.