Academic Timetable Optimization Using Constraint Satisfaction Problem

Sistemi Intelligenti

Problems to solve

- Too much problems with the timetable because I have subjects from differents degrees and they usually overlap.
- Fixed schedules
- Need to attend as many classes as possible

Algorithm to obtain the best combination of classes?

Goals

- Maximize attendance without overlaps.
- Minimize travel between buildings.
- Compact schedules and reduce downtime.
- Evaluate all possible combinations.

What is a CSP?

Constraint Satisfaction Problem

Main features

★ Some variables can take different values.
★ There are constraints that define which combinations of values are valid.
★ The goal is to find a combination of values for the variables that satisfies all the constraints.

Variables and domains

```
subjects = {
    "Mobile App Dev": ["Mon 8-10", "Fri 14-16"],
    "Sistemi Intelligenti": ["Tue 10-12", "Thu 14-16"],
    "Sistemi Elettronici": ["Fri 8-10", "Thu 16-18"],
    "Elettronica Biomedica": ["Wed 10-12", "Wed 16-18"],
    "Applicazioni di Fisica": ["Mon 14-16", "Tue 14-16"],
    "Circuiti": ["Wed 8-10", "Thu 10-12"],
    "Data Science": ["Tue 8-10", "Thu 8-10"],
    "AI Fundamentals": ["Mon 10-12", "Wed 14-16"],
    "Embedded Systems": ["Tue 16-18", "Fri 10-12"],
    "Computer Networks": ["Mon 16-18", "Thu 16-18"],
    "Biomedical Signals": ["Tue 10-12", "Wed 16-18"],
    "Digital Control": ["Wed 14-16", "Fri 8-10"],
    "Physics Lab": ["Thu 8-10", "Fri 16-18"],
    "Digital Design": ["Tue 14-16", "Thu 10-12"],
    "Robotics": ["Wed 10-12", "Fri 14-16"]
}
```

Each subject is a variable and the available time slot is the domain

Restrictions between variables

```
def no_overlap(slot1, slot2):
    day1, start1, end1 = parse_time_slot(slot1)
    day2, start2, end2 = parse_time_slot(slot2)
    if day1 != day2:
        return True
    return end1 <= start2 or end2 <= start1
```

"Thu 14-18"

Day Start End

No overlap

↳ first class ends when second class starts

Restrictions between variables

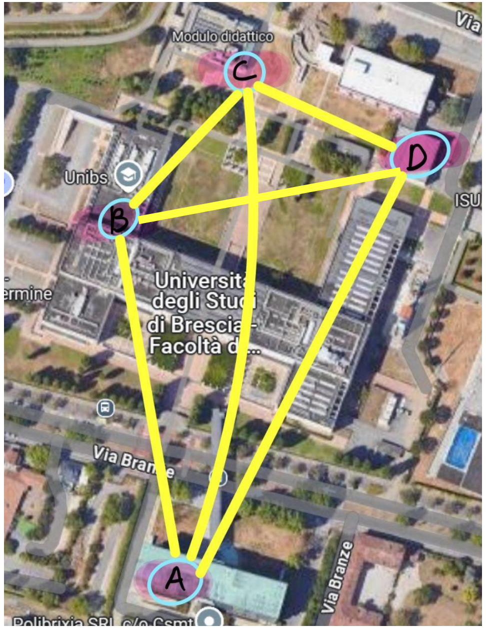
```
def no_travel_conflict(slot1, slot2, location_by_slot, graph):
    day1, start1, end1 = parse_time_slot(slot1)
    day2, start2, end2 = parse_time_slot(slot2)
    if day1 != day2:
        return True
    if end1 == start2 or end2 == start1:
        loc1 = location_by_slot.get(slot1)
        loc2 = location_by_slot.get(slot2)
        if shortest_path_time(loc1, loc2, graph) > 5:
            return False
    return True
```

Differents days

Take the location

if takes less than 5 minutes
no travel conflict

Graph of the campus




```
campus_graph = {
    "A": [("B", 7), ("C", 15), ("D", 12)],
    "B": [("A", 7), ("C", 5), ("D", 10)],
    "C": [("A", 15), ("B", 5), ("D", 6)],
    "D": [("A", 12), ("B", 10), ("C", 6)]
```

graph where the edges are the travel time and the nodes are the buildings

CSP modeling

```python
problem = Problem()
for subject, slots in subjects.items():
    options = []
    if len(slots) > 1:
        options += [combo for combo in itertools.combinations(slots, 2) if no_overlap(*combo)]
    options += [(slot,) for slot in slots]
    problem.addVariable(subject, options)

subject_list = list(subjects.keys())
for i in range(len(subject_list)):
    for j in range(i + 1, len(subject_list)):
        s1, s2 = subject_list[i], subject_list[j]
        def constraint(a, b, s1=s1, s2=s2):
            for slot1 in a:
                for slot2 in b:
                    if not no_overlap(slot1, slot2) or not no_travel_conflict(slot1, slot2, location_by_slot, campus_graph):
                        return False
            return True
        problem.addConstraint(constraint, (s1, s2))

solutions = problem.getSolutions()
```

Is there only one valid solution?

- CSPs can generate many valid combinations.
-
- Each solution respects all constraints.
-
- Not all are equally efficient or practical.
-
- We use a heuristic to pick the best ones.

Heuristic evaluator

```python
def evaluate_schedule(solution):
    flat_solution = {subj: slot for subj, slots in solution.items() for slot in (slots if isinstance(slots, tuple) else [slots])}
    daily_slots = defaultdict(list)
    for subj, slot in flat_solution.items():
        day, start, end = parse_time_slot(slot)
        daily_slots[day].append((start, end, location_by_slot[slot]))

    total_dead_time = 0
    total_moves = 0
    total_classes_attended = sum(len(slots) if isinstance(slots, tuple) else 1 for slots in solution.values())
    for day, blocks in daily_slots.items():
        blocks.sort()
        for i in range(len(blocks) - 1):
            _, end1, loc1 = blocks[i]
            start2, _, loc2 = blocks[i + 1]
            dead_time = start2 - end1
            total_dead_time += max(0, dead_time)
            if loc1 != loc2:
                total_moves += 1

    days_with_class = len(daily_slots)
    has_friday_free = "Fri" not in daily_slots
```

Ranking

```
ranked = sorted(
    [evaluate_schedule(sol) for sol in solutions],
    key=lambda x: (-x['classes_attended'], x['dead_time'], x['moves'], x['days'], not x['bonus_friday_off'])
)
```

First Solution

```
Schedule #1:

Schedule Summary:

Mon:
  14-16 -> Applicazioni di Fisica
  16-18 -> Computer Networks

Tue:
  8-10 -> Data Science
  10-12 -> Biomedical Signals

Wed:
  8-10 -> Circuiti
  10-12 -> Robotics
  14-16 -> AI Fundamentals
  16-18 -> Elettronica Biomedica

Thu:
  8-10 -> Physics Lab
  10-12 -> Digital Design
  14-16 -> Sistemi Intelligenti
  16-18 -> Sistemi Elettronici

Fri:
  8-10 -> Digital Control
  10-12 -> Embedded Systems
  14-16 -> Mobile App Dev
```

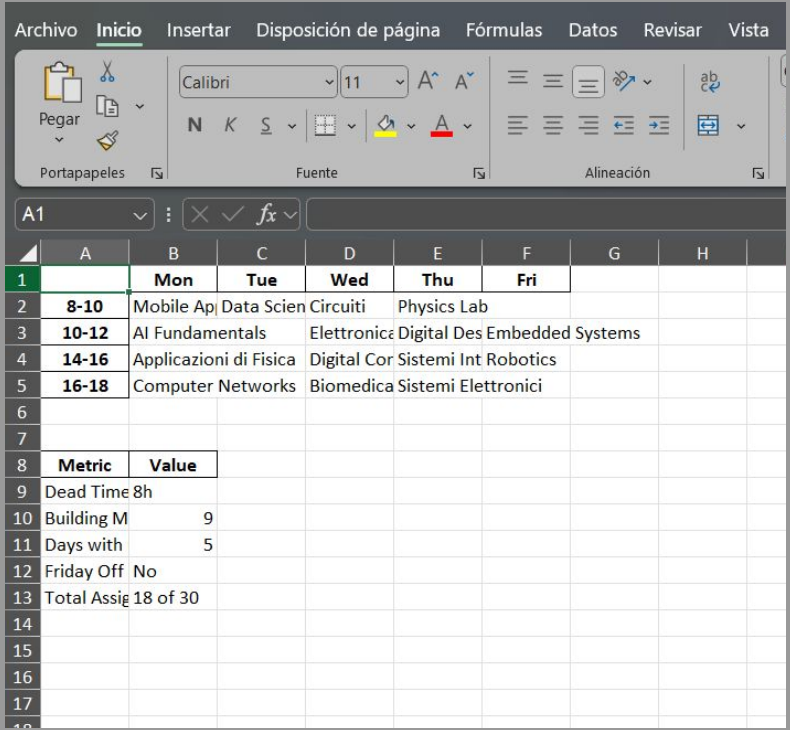
Second Solution

```
Schedule Summary:

Mon:
  8-10 -> Mobile App Dev
  10-12 -> AI Fundamentals
  14-16 -> Applicazioni di Fisi

Tue:
  8-10 -> Data Science

Wed:
  8-10 -> Circuiti
  10-12 -> Elettronica Biomedic
  14-16 -> Digital Control
  16-18 -> Biomedical Signals

Thu:
  8-10 -> Physics Lab
  10-12 -> Digital Design
  14-16 -> Sistemi Intelligenti
  16-18 -> Computer Networks

Fri:
  8-10 -> Sistemi Elettronici
  10-12 -> Embedded Systems
  14-16 -> Robotics
```

Third Solution

```
Schedule Summary:

Mon:
  8-10 -> Mobile App Dev
  10-12 -> AI Fundamentals
  14-16 -> Applicazioni di Fisica
  16-18 -> Computer Networks

Tue:
  8-10 -> Data Science

Wed:
  8-10 -> Circuiti
  10-12 -> Elettronica Biomedica
  14-16 -> Digital Control
  16-18 -> Biomedical Signals

Thu:
  8-10 -> Physics Lab
  10-12 -> Digital Design
  14-16 -> Sistemi Intelligenti
  16-18 -> Sistemi Elettronici

Fri:
  10-12 -> Embedded Systems
  14-16 -> Robotics
```

Excel



Archivo Inicio Insertar Disposición de página Fórmulas Datos Revisar Vista
Pegar
Calibri
11
N K S
Portapapeles
Fuente
Alineación
A1
A B C D E F G H
Mon Tue Wed Thu Fri
8-10 Mobile Ap Data Scien Circuiti Physics Lab
10-12 AI Fundamentals Elettronica Digital Des Embedded Systems
14-16 Applicazioni di Fisica Digital Cor Sistemi Int Robotics
16-18 Computer Networks Biomedica Sistemi Elettronici
Metric Value
Dead Time 8h
Building M 9
Days with 5
Friday Off No
Total Assig 18 of 30

Conclusions

- Schedules optimized using constraint programming.
- Criteria: idle time, campus moves, active days, Friday off.
- Results exported to Excel and shown in console.
- Improves academic planning and student experience.

Academic Timetable Optimization Using Constraint Satisfaction Problem

Lucía Blázquez Cintas, Erasmus University of Seville