Automated Reasoning Project

Ikrame Ounadi

July 2024

1 Introduction

The following report relates to the course project of "Automated Reasoning". This project focuses on optimizing patient scheduling in a medical facility to minimize the total number of days patients visit and their total waiting time, while adhering to various constraints such as visit durations... The problem as given by the professor will be explained in the next section.

2 Problem

7. Hospital. An hospital is ready to provide k types of visits in different labs (e.g., radiography, TAC, NMR, blood sample, ...). Each visit takes some time, let's use as unit time 10 minutes, so you have visits at 8.00, 8.10, 8.20, 8.30, 8.40, 8.50, 9.00, 9.20, . . . , 17.50. Some visits might take more than 1 unit time (for instance NMR takes 30 minutes). Part of the input (that depends on the hospital) is the k types of visits and their duration (invent the data here, I am not a doctor). The other part of the inputs are the patients. You have a list of patients wach of them with a set of required visits (e.g. patient 1 needs a blood sample and a TAC). Also here invent the data.

Reason on a time window of 5 days (work week). The aim is to schedule patients to labs in order to minimize (high penalty) the fact of non returning in two separate days and to minimize (smaller penalty) the sum of the time spent for waiting the next visit in the hospital. Blood sample ends at 11.00, the other exams end at 18.00. After blood sample patients have to stay seated for 30 minutes (3 times unit) before entering another visit or leaving the hospital. Of course, the same patient can give at most one visit at a given time.

3 Method

To model the hospital scheduling problem, the following assumptions and constraints were established:

3.1 Input Data

- **Patients and Visits**: Each patient has a predefined set of required visits. These visits (eg: Radiography, TAC, NMR, and Blood Sample..), with durations (eg: 10, 20, 30, 10 minutes respectively) .
- **Time Slots**: The hospital's operating hours (8:00 to 18:00) are divided into 60 discrete time slots per day, each representing 10 minutes. (10 hours of work a day each hour has 6 slots)
- **Time window**: The scheduling is done over a 5-day work week, from Monday to Friday.

3.2 Constraints

- **Non-Overlapping Visits**: Patients cannot have overlapping visits. and a patient cannot have 2 visits in the same time.
- **Waiting Periods**: After a blood sample, patients must stay seated for 30 minutes (3 time units) before attending another visit or leaving the hospital.
- **Daily Scheduling Constraints**: Visits must be scheduled between Monday and Friday(5 working days), and blood samples must end before 11:00 AM (time slot 18).

3.3 Optimization Objectives

- **Minimize Total Days of Visits**: Minimize the number of days each patient needs to visit the hospital.(high penalty)
- **Minimize Waiting Time**: Minimize the total waiting time between visits for each patient.(smaller penalty)

4 Tools Used

The following tools were used to carry out the project:

- **The MiniZinc IDE Version 2.8.5**
- **MiniZinc to FlatZinc converter, version 2.8.5**
- **Clingo 5.7.1**
- **Python 3.12.4**:
- **Visual Studio Code**

5 Models

5.1 input data

5.1.1 Minizinc data sample

The following data is used for the MiniZinc model:

```
num_days = 5;
num_time_slots = 60;
num_patients = 5;
num_visits = 3;
duration = [1, 1, 2];
lab_names = ["Radiography", "Blood Sample", "TAC"];
required = [| 0, 1, 1 | 0, 1, 1 | 1, 1, 0 | 0, 0, 1 | 1, 0, 1 |];
```

Figure 1: MiniZinc Data

- **num_days = 5;** - Specifies the number of days in the scheduling period.
- **num_time_slots = 60;** - Specifies the number of time slots available each day.
- **num_patients = 5;** - Specifies the number of patients.
- **num_visits = 3;** - Specifies the number of visit types.
- **duration = [1, 1, 2];** - Specifies the durations of each visit type: "Radiography" and "Blood Sample" each take 1 time slot, while "TAC" takes 2 time slots.
- **lab_names = ["Radiography", "Blood Sample", "TAC"];** - Specifies the names of the visit types.
- **required** - Specifies which visits are required for each patient in a 2D array. Each row represents a patient, and each column represents a visit type. A value of 1 means the visit is required, and 0 means it is not. For example, the first row [0, 1, 1] means patient 1 requires a "Blood Sample" and "TAC" visit.

5.1.2 ASP data sample

The following data is used for the ASP (Answer Set Programming) model:

- **day(1..5).** - Defines the days in the scheduling period, from 1 to 5.
- **time_slot(1..60).** - Defines the time slots available in a day, from 1 to 60.
- **patient(1..5).** - Defines the patients, numbered from 1 to 5.
- **visit(1, "Radiography").** - Defines the visit type(eg: 1 as "Radiography".)
- **duration(1, 1).** - Defines the duration of "Radiography" as 1 time slot.
- **required(P, V).** - Specifies which visits are required for each patient. For example, **required(1, 4)** means patient 1 requires a "Blood Sample" visit.

```
day(1..5).
time_slot(1..60).
patient(1..5).
visit(1, "Radiography").
duration(1, 1).
visit(4, "Blood Sample").
duration(4, 1).
visit(2, "TAC").
duration(2, 2).
required(1, 4).
required(1, 2).
required(2, 4).
required(2, 2).
required(3, 1).
required(3, 4).
required(4, 2).
required(5, 1).
required(5, 2).
```

Figure 2: ASP Data

5.2 MiniZinc Model

1 - Ensure Each Required Visit is Scheduled Exactly Once

This constraint ensures that each required visit for a patient is scheduled exactly once.

> **Logic:** If a visit is required (required[p, v] = 1), it must be scheduled exactly once across all days and time slots.

```
% Ensure each required visit is scheduled exactly once
constraint
  forall(p in 1..num_patients, v in 1..num_visits) (
    required[p, v] = 1 -> sum([bool2int(schedule[p, v, d, t]) | d in 1..num_days, t in 1..num_time_slots]) = 1
  );
```

Figure 3: Ensure Each Required Visit is Scheduled Exactly Once

2 - A Patient Cannot Have Two Visits at the Same Time

This constraint ensures that a patient cannot have overlapping visits.

Logic: For each patient, on each day, at each time slot, the sum of scheduled visits must be less than or equal to 1.

```
% A patient cannot have two visits at the same time
constraint
  forall(p in 1..num_patients, d in 1..num_days, t in 1..num_time_slots)
    sum([bool2int(schedule[p, v, d, t]) | v in 1..num_visits]) <= 1
  );

% Blood samples must end before 11:00 (time slot 18)
```

Figure 4: A Patient Cannot Have Two Visits at the Same Time

3 - Blood Samples Must End Before 11:00 (Time Slot 18)

This constraint ensures that blood samples are completed before 11:00 AM.

Logic: If the end time of a blood sample visit exceeds the 18th time slot, the visit cannot be scheduled.

```
% Blood samples must end before 11:00 (time slot 18)
constraint
  forall(p in 1..num_patients, d in 1..num_days, t in 1..num_time_slots, v in 1..num_visits) (
    (lab_names[v] == "Blood Sample") -> (t + duration[v] > 18 -> not schedule[p, v, d, t])
  );
```

Figure 5: Blood Samples Must End Before 11:00

4 - Patients Must Stay Seated for 30 Minutes (3 Time Units) After a Blood Sample

This constraint enforces a 30-minute waiting period after a blood sample.

Logic: If a blood sample is scheduled, no other visits can be scheduled for the next 3 time units.

```
% Patients must stay seated for 30 minutes (3 time units) after a blood sample
constraint
  forall(p in 1..num_patients, d in 1..num_days, t1 in 1..num_time_slots, v in 1..num_visits) (
    (lab_names[v] == "Blood Sample") -> (schedule[p, v, d, t1] ->
      forall(t2 in t1+1..min(num_time_slots, t1+3)) (
        sum([bool2int(schedule[p, v2, d, t2]) | v2 in 1..num_visits]) = 0
      )
    )
  );
```

Figure 6: Patients Must Stay Seated for 30 Minutes After a Blood Sample

Objective: Minimize the Number of Days Each Patient Visits and the Total Waiting Time

The objective function aims to minimize a weighted sum of the total number of days patients visit and their total waiting time.

Logic: The goal is to minimize the number of days patients visit the hospital and the total waiting time, with a higher priority on minimizing visit days.

```
% Objective: Minimize the number of days each patient visits and the total waiting time
var int: total_days = sum([bool2int(visit_day[p, d]) | p in 1..num_patients, d in 1..num_days]);
var int: total_waiting_time = sum(waiting_time);
solve minimize total_days * 100 + total_waiting_time;
```

Figure 7: Objective Function

5.3 ASP

1 - Ensure Each Required Visit is Scheduled Exactly Once

This constraint ensures that each required visit for a patient is scheduled exactly once.

> **Logic:** If a visit is required (required(P, V)), it must be scheduled exactly once across all days and time slots.

```
% Decision variables: Schedule visit V for patient P on day D at time T
1 { schedule(P, V, D, T) : day(D), time_slot(T) } 1 :- patient(P), required(P, V).
```

2 - A Patient Cannot Have Two Visits at the Same Time and Must Respect the Duration

This constraint ensures that a patient cannot have overlapping visits and respects the visit durations.

> **Logic:** For each patient, on each day, at each time slot, the sum of scheduled visits must be less than or equal to 1, taking into account the duration of each visit.

```
% A patient cannot have two visits at the same time and must respect the duration
:- schedule(P, V1, D, T1), schedule(P, V2, D, T2), V1 != V2,
   duration(V1, Dur1), duration(V2, Dur2),
   T1 <= T2, T2 < T1 + Dur1.
```

Figure 8: Patients can not have 2 visits in the same time

3 - Blood Samples Must End Before 11:00 (Time Slot 18)

This constraint ensures that blood samples are completed before 11:00 AM.

> **Logic:** If the end time of a blood sample visit exceeds the 18th time slot, the visit cannot be scheduled.

```
% Blood samples must end before 11:00 (time slot 18)
:- schedule(P, 4, D, T), T + 1 > 18.
```

Figure 9: no blood samples after 11:00

4 - Patients Must Stay Seated for 30 Minutes (3 Time Units) After a Blood Sample

This constraint enforces a 30-minute waiting period after a blood sample.

> **Logic:** If a blood sample is scheduled, no other visits can be scheduled for the next 3 time units.

Objective: Minimize the Number of Days Each Patient Visits and the Total Waiting Time

The objective function aims to minimize a weighted sum of the total number of days patients visit and their total waiting time.

```
% Patients must stay seated for 30 minutes (3 time units) after a blood sample
:- schedule(P, 4, D, T1), schedule(P, V, D, T2), V != 4, T1 + 4 > T2.
```

Figure 10: Patients Must Stay Seated for 30 Minutes After a Blood Sample

Logic: The goal is to minimize the number of days patients visit the hospital and the total waiting time, with a higher priority on minimizing visit days.

```
% Minimize the number of days each patient visits and the total waiting time
#minimize { 10*D,P : visit_day(P, D) }.
#minimize { T2 - (T1 + Dur1),P,D : waiting_time(P, D, T1, T2, Dur1) }.
```

Figure 11: objective function

6 Results

6.1 Minizinc results

```
Total days: 1
Total waiting time: 1
----------
Schedule:
Patient 1 has visit "Radiography" on day 1 at time 3
Patient 1 has visit "Blood Sample" on day 1 at time 4
Total days: 1
Total waiting time: 0
----------
==========
Finished in 52s 339msec.
```

Figure 12: minizinc results

Dimensions of the Matrix Rows: Represent the time slots in a day. Columns: Represent the different patients. Entries: Indicate the type of visit at that specific time slot for a particular patient.

6.2 ASP results

```
Answer: 101
schedule(2,4,1,1) schedule(2,1,1,5) schedule(2,3,1,6) schedule(3,4,1,1) schedule(3,3,1,5)
schedule(4,1,1,21) schedule(4,4,1,17) schedule(5,3,1,11) schedule(5,4,1,7) schedule(1,3,1,60)
Optimization: 66
OPTIMUM FOUND

Models       : 101
  Optimum    : yes
Optimization : 66
Calls        : 1
Time         : 5.112s (Solving: 3.10s 1st Model: 0.01s Unsat: 0.75s)
CPU Time     : 4.732s
```

Figure 13: ASP results

Each schedule(P,V,D,T) indicates that Patient P has Visit V on Day D at Time T.

6.3 Results comparaison

The following table presents a comparison of the MiniZinc and ASP (Answer Set Programming) approaches based on their execution times and the results obtained for different instances categorized as "easy", "average", and "hard".

MiniZinc: The MiniZinc solver took significantly longer to solve instances, particularly in the "average" and "hard" categories, where it timed out (exceeding the 5-minute limit).

ASP: The ASP solver demonstrated superior performance, solving even the "average" and "hard" instances much faster than MiniZinc, with times ranging from milliseconds to a few seconds.

Instance Category	Instance ID	MiniZinc Execution Time	MiniZinc Result	ASP Execution Time	ASP Result
Easy	0	8.94 seconds	Completed (2 days, 0 waiting time)	0.377 seconds	Completed (1 day, 0 waiting time)
Easy	1	23.62 seconds	Completed (2 days, 0 waiting time)	0.025 seconds	Completed (1 day, 0 waiting time)
Easy	2	3.51 seconds	Completed (1 day, 0 waiting time)	0.026 seconds	Completed (1 day, 0 waiting time)
Easy	3	3.55 seconds	Completed (1 day, 0 waiting time)	0.026 seconds	Completed (1 day, 0 waiting time)
Average	0	335.49 seconds (Timeout)	Unknown	5.112 seconds	Completed (1 day, 0 waiting time)
Average	1	370.27 seconds (Timeout)	Unknown	4.756 seconds	Completed (1 day, 0 waiting time)
Average	2	349.47 seconds (Timeout)	Unknown	2.205 seconds	Completed (1 day, 0 waiting time)
Average	3	353.05 seconds (Timeout)	Unknown	0.081 seconds	Completed (1 day, 0 waiting time)
Hard	0	363.45 seconds (Timeout)	Unknown	0.081 seconds	Completed (1 day, 0 waiting time)
Hard	1	382.49 seconds (Timeout)	Unknown	Timeout	Unknown

Table 1: Comparison of MiniZinc and ASP Approaches for Scheduling Problem

7 Conclusion

From the experimental results, it can be concluded that the Minizinc model performs well on smaller instances of the problem, providing optimal solutions within the time limit. However, as the problem size increases, the execution time also increases significantly. The ASP model, although slower, can provide solutions for larger instances within the time limit. Further optimizations and specialized solvers may improve performance