P1 : Timetable Management System for an Academic Institution

Review 2 Report

Team 7 - DB4C

Aathish Sivasubrahmanian [3122 21 5001 001]

Harshida S P, [3122 21 5001 031]

Ayshwarya B, [3122 21 5001 017]

18.06.2022

FUNDAMENTALS OF PROGRAMMING AND SOFTWARE DEVELOPMENT

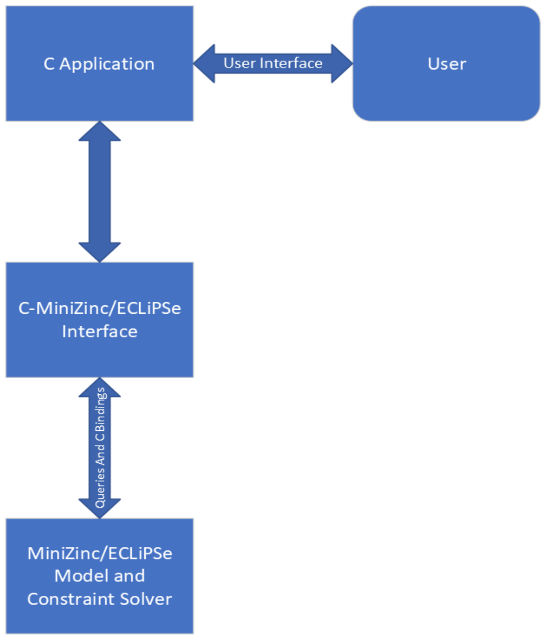
INTRODUCTION

Solving the timetabling problem involves arranging a university course timetable for all lectures in a week for a specific programme, where each lecture is assigned to a classroom and a time-slot. Students in indivisible groups who attend lectures at the same time are classified as homogeneous units called "sections". Each lecture contains one or more sections, as well as one or more faculty members assigned to it. The scheduler's job is to make sure that the teachers are available for the lecture at the scheduled time and that there are no scheduling conflicts between the various allocated lectures.

The method of solving the timetabling problem chosen by the proponents involves a Constraint Logic Programming approach implemented in a Constrainted Logic Programming language such as MiniZinc or ECLiPSe. This report will discuss the specific details, such as architectural design, modules involved, and societal impact of such a system.

ARCHITECTURAL DESIGN

The application will feature a front-end built in C that will allow the user to generate a timetable while passing data and arguments to the MiniZinc/ECLiPSe solver. The solver runs the query and returns the results in a file or a file stream, which can then be read by the C application and finally be displayed to the end user. The C application may also format the data returned into a more convenient format for display and transmission, such as a CSV file. The internals of the problem solver are hidden from the end user.


C Application
User Interface
User
C-MiniZinc/ECLiPSe Interface
Queries And CBindings
MiniZinc/ECLiPSe Model and Constraint Solver

There are three main resources that need to be considered by the MiniZinc/ECLiPSe solver in the timetabling problem - the sections, the faculty and the classroom. Professors and Sections are already assigned to each other, and the system's only purpose is to assign a classroom to a particular section and professor pair for a particular time-slot. The solver will need to ensure that none of the hard constraints are broken and also maximise the number of soft constraints satisfied.

Predetermined data (such as available faculty and assigned sections) can be fed to the system by means of a .dzn file, where the data is formatted in a similar manner to defining variables in MiniZinc, or by directly reading a suitable data file (such as CSV) from ECLiPSe. For the purpose of this report, code samples are given in MiniZinc.

ALGORITHM/CODE SAMPLES:

A) Coherence Constraints :

Module 1 :

INDIVIDUALITY CONSTRAINT : Faculty and Sections are only given courses assigned to them.

Input : Class list, section list, teacher list, number of slots per day,

```
constraint

forall (courseid in coursesarray)(

   forall(facultyindex in faculty_index_arr)(

       if sum(class in classes_per_week_for_single_fac_index_arr)

       (

           if facultydetails[facultyindex,class,3] == courseid

           then 1

           else 0

           endif

       ) < 1

       then forall(day in daysPerWeek, slot in slotsPerDay)(

           if facultyTimetable[facultyindex,day,slot] != 0

           then facultyTimetable[facultyindex,day,slot] != courseid
```

```
            endif

        )

        endif

    )

);
```

```
constraint

forall(sectionindex in sections_index_arr, day in daysPerWeek, slot in slotsPerDay)(

   if sectionTimetable[sectionindex, day, slot] > 0

   then sum(facultyindex in faculty_index_arr, class in
classes_per_week_for_single_fac_index_arr)(

       if facultydetails[facultyindex, class, 3] == sectionTimetable[sectionindex,
day, slot] /\

           facultydetails[facultyindex, class, 2] == sectionids_array[sectionindex]
/\

           facultyTimetable[facultyindex, day, slot] ==
sectionTimetable[sectionindex, day, slot]

       then 1

       else 0

       endif

   ) > 0

   endif

);
```

Module 2 :

ROOM-LIMITATION CONSTRAINT : A classroom is assigned just one lecture at a time.

The number of teachers in the array of teachers that are teaching during any given slot must be lesser than or equal to the total number of rooms.

Input : Class list, section list, number of slots per day

```
constraint

forall(day in daysPerWeek, slot in slotsPerDay)(

   sum(facultyindex in faculty_index_arr)(

       if facultyTimetable[facultyindex,day,slot]!=0

       then 1

       else 0

       endif

   ) <= num_rooms

);
```

Module 3 :

COURSE-ENTITY CONSTRAINT : The sections and teachers involved in a particular class must share the same time slot. Two constraints are used to implement this. The first constraint ensures that if a teacher has a class scheduled on a particular day and slot, one of the student sections also has a class scheduled at that time. The second constraint ensures that if a student section has a class scheduled on a particular day and slot, a teacher also has a class scheduled at that time.

Input : Class list, section list, teachers list, number of slots per day

```
constraint
forall(facultyindex in faculty_index_arr,class in
classes_per_week_for_single_fac_index_arr)(
   forall(sec in sections_index_arr,day in daysPerWeek, slot in slotsPerDay)(
       if facultydetails[facultyindex,class,2] == sectionids_array[sec] /\
facultyTimetable[facultyindex,day,slot]!=0
       then sectionTimetable[sec,day,slot] = facultyTimetable[facultyindex,day,slot]
       endif
   )
);
```

B) Feature Constraints :

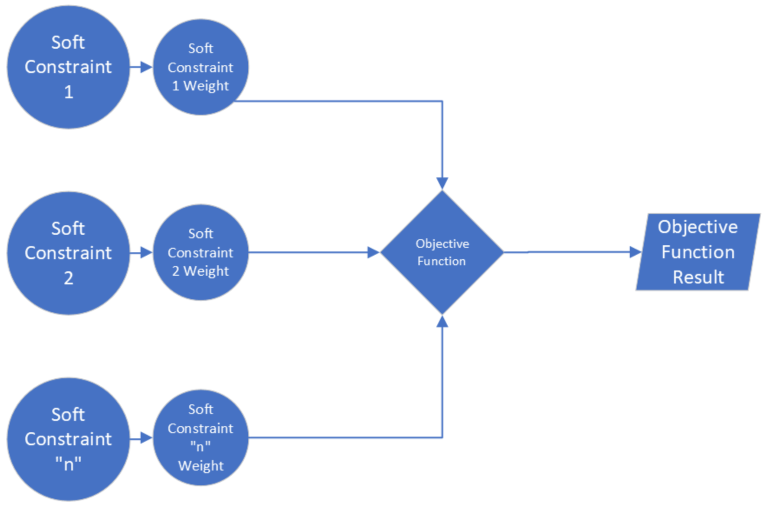
Module 4 :

NON-REPEATING LECTURE CONSTRAINT :

The lecture of the same course should not be held multiple times on the same day. This feature can be built into the "Individuality Constraint". The number of classes of a particular course on a particular day should be limited to one. The constraint below ensures that faculty do not have consecutive occupied slots unless the slot is occupied by a lab session, which may cover multiple slots.

```
constraint

forall(facultyindex in faculty_index_arr,day in daysPerWeek)(

   % Ensure faculty have free slot after every occupied slot unless

   % it is for a lab session.

   forall(slot in slotsPerDay)(

       if slot < slots

       then

           if facultyTimetable[facultyindex,day,slot] > 0 /\ not
courseid_is_lab(facultyTimetable[facultyindex,day,slot])

           then facultyTimetable[facultyindex,day,slot+1] = 0

           endif

       endif /\

       if slot > 1

       then

           if facultyTimetable[facultyindex,day,slot] > 0 /\ not
courseid_is_lab(facultyTimetable[facultyindex,day,slot])

           then facultyTimetable[facultyindex,day,slot-1] = 0

           endif

       endif

   )

);
```

C) General Soft Constraint Implementation



In order to implement recognition of soft constraints, an Objective Function can be used. Such a function would allocate equal or different weights to each soft constraint specified, and the solver would attempt to maximise the total return value of the objective function. The solution with the greatest objective function return value would then be returned as the ideal solution. The user may also choose which among the preference/soft constraints are included for the objective function. This choice is represented as a list of integers, representing both the priority of the soft constraint that the solver must use in order to calculate the return value of the objective function.

Further real-time analysis must be done in order to determine the weights that must be given to each soft constraint. It may also be necessary for the end user to be able to modify these weights as they require, or at least re-order the priority of the constraints, as each end user may have different soft constraint priorities in mind. For example, one user may find it more important to have classes alternate between Core and non-Core subjects, while another user may find it more important to avoid allocating core subjects in the last slot. To account for these differences, the end user must be allowed to edit the priority (if not the weight) of every soft constraint that can be taken into account.

IMPACT ANALYSIS

Due to the many disparate variables involved, class scheduling is a complicated and highly constrained task that has traditionally been solved manually. However, the complexity of the problem ensures that it is difficult for a human to find an optimal solution in a reasonable amount of time.

Many AI and Non-AI based approaches have been proposed and each of them have their own merits and demerits. Since both Manual and Non-AI based approaches tend to fall short of the goals achievable by AI based approaches, researchers often focus on the AI based approach.

The method described by the proponents involves a solver independent model where additional data is passed into the model as parameters. Previous research implementing similar methods have been able to achieve quite a high degree of accuracy and efficiency in implementing purely hard constraints.

It is expected that using the methods described in this report, both hard and soft constraints can be implemented with similar efficiency and accuracy.

By automating the process of timetable setting, both human error and wilful malice are greatly reduced, and hence the ethical and societal impacts are not insignificant.

By increasing the efficiency and accuracy of the solving system, the environmental impact is significant, as the power needed to run such a system will ultimately be lesser, resulting in lower net carbon emissions.

TIMELINE

Activities (in weeks)	1 R.1	2	3	4 R.2	5	6 R.3	7	8	9	10	11 R.4
Formulating problem statement	■										
Designing system architecture		■	■								
Implementing basic input parsing and unconstrained timetable generation			▨	▨							
Implementing hard constraint recognition				▨	▨		▨	▨			
Implementing soft constraint recognition					▨		▨	▨			
Review of soft and hard constraint recognition feasibility				▨	▨	▨			▨		
Finishing implementation details									▨	▨	▨