Computer Science
and Mathematics
COC255
B827126

TIMETABLE CREATION USING ARTIFICIAL INTELLIGENCE

by

Aiden Nico Tempest

Supervisor: Dr. S. Fatima

Department of Computer Science
Loughborough University

August 2023

Abstract

lol placeholder text for abstract lorem ipsum etc test

Contents

Chapter 1

Literature Review

1.1 The Timetabling Problem

The timetabling problem is a constraint satisfying problem and it is known to be NP-Complete. The aim is to create a university timetable where several constraints are met. These constraints are either hard constraints or soft constraints. For a solution to be valid, all the hard constraints must be met. Not all (or in fact, any) of the soft constraints need to be met for the solution to be valid, however it is preferable for as many soft constraints to be met as possible. Examples of possible hard constraints include:

- At most only one session (i.e., a lecture or lab) is happening in a specific room in a specific period
- A student can only be attending at most one session in a specific period
- A teacher (e.g., a lecturer or lab helper) can only be attending at most one session in a specific period
- The size of a student group cannot exceed the capacity of the room
- The room must be appropriate for the type of session, e.g., a lecture

must be in a lecture theatre, and a lab session must be in a computer lab

- Part time teachers can only be assigned certain time slots, e.g., they may not work on Tuesdays, so sessions they teach cannot be scheduled for time slots on Tuesday.

Possible soft constraints include:

- Students do not have more than two consecutive hours scheduled
- The capacity of a room is well suited to the size of the student group, to make an efficient use of space e.g., a group of twenty students are not going to be in a room with a capacity of two hundred
- If a student or teacher has one session immediately after another, then the respective rooms are relatively close to each other

A solution is invalid if at least of one the hard constraints are met. For example, if a student is scheduled to be in two different sessions at the same time — this is known as a clash.

1.2 Methods

1.2.1 Genetic Algorithms

Genetic algorithms (GAs) made up a group of search metaheuristics, inspired by Darwin's theory of evolution. Here, the fittest members of a population survive and produce offspring, which inherit the characteristics of the parents. It is also possible for the offspring to have small mutations within their genetic code, which may or may not be beneficial towards the population's survival. This theory can be applied to search problems. In this case, the population represents the search space, which is a collection of candidate solutions to a problem, and the population of solutions evolves as the algorithm searches for

a desired solution.

There does not exist a rigorous definition of GAs, but most methods use these five phases:

1. Initial population — populations of chromosomes
2. Fitness function
3. Selection — according to fitness
4. Crossover — to produce new offspring
5. Mutation — random mutation of new offspring

The initial **population** is a set of **individuals**, where each individual represents a candidate solution to the problem. These solutions will almost definitely not satisfy the problem, as they are randomly generated, but that is not an issue.

An individual (and hence that candidate solution), is defined by its **chromosome**. The chromosome is often encoded as a string of binary characters; however, any alphabet can be used. These characters are called **alleles**, and a single character or group of adjacent characters that encode a particular element of the candidate solution are known as a **gene**.

Using the example of the university timetabling problem, several alleles may be used to encode a room, but together they are one gene.

The **fitness** function measures how well a candidate solution solves the problem. For example, in the instance of the university timetabling problem, the fitness of the solution could be measured by the number of times in the solution that a hard constraint is not met. This means that a solution with a score of 0 is a valid solution, as there are instances of a hard constraint not being met.

Once the fitness function has been used to calculate the fitness of each individual, the fittest individuals are chosen for reproduction in the **selection**

phase. There are several ways to select individuals to use for producing offspring. One way is to simply choose the two individuals with the best fitness. Another way is to use roulette-wheel sampling, where any individual could be chosen for producing offspring, but fitter individuals are more likely to be chosen. This second method introduces more variation into the offspring, to reduce the chance of convergence onto a local maximum.

The **crossover** phase, or reproduction phase, is where genetic material is exchanged between two parents. The crossover operator randomly chooses a locus (position) in a chromosome, in between alleles. The subsequent before and after parts of the chromosome are swapped between the two parents to produce two offspring. This is repeated multiple times to produce a population of the same size as the initial population. Whether this is by two parents reproducing multiple times using different loci or not, depends on the method used in the selection phase.

After reproduction, **mutations** can be introduced into the offspring. For example, if the chromosomes are encoded by binary strings, then some bits are randomly flipped. However, there is a very small chance of this occurring at each bit, a suggested probability is 0.1%. By introducing mutations into the offspring, the likelihood of reaching a local maximum is reduced.

Once there is a new population made up of the offspring, the process repeats until a solution is found. Each repetition is called a **generation**, and the entire set of generations is known as a **run**.

1.2.2 Binary Integer Programming

Binary integer programming is used to solve constraint satisfiability problems. Variables must either take a value of 1 or 0 (hence binary integer), as they are used to represent decisions, i.e., in the case of the university timetabling problem, a teaching session is happening in a specific room, at a specific time, on a specific day, with a specific teacher, with a specific teacher, with a specific

student group, about a specific module, or not. Then constraints are applied and used with the objective function to find what value each variable takes.

First, a mathematical model of the problem must be constructed. REF modelled the features of the university timetabling problem as a group of sets:

- $I = \{1, \ldots, n_i\}$: set of days in the week where courses are offered
- $J = \{1, \ldots, n_j\}$: set of time slots in a day
- $K = \{1, \ldots, n_k\}$: set of courses
- $L = \{1, \ldots, n_l\}$: set of student groups
- $M = \{1, \ldots, n_m\}$: set of teachers
- $N = \{1, \ldots, n_n\}$: set of classrooms

Next, the decision variables are defined. The basic variables $x_{i,j,k,l,m,n}$ are defined as
$\forall i \in I, \forall j \in J, \forall k \in K, \forall l \in L, \forall m \in M, \forall n \in N$

$$x_{i,j,k,l,m,n} = \begin{cases} 1, & \text{if a course } k \text{ taught by teacher } m \text{ for the group of} \\ & \text{students } l \text{ is assigned to the } j^{th} \text{ time slot of day } i \text{ in} \\ & \text{classroom } m \\ 0, & \text{otherwise} \end{cases}$$

In other words, the basic variables represent the decision of whether or a not a course is being taught by a specific teacher for a specific group of students is happening in a specific time slot of a specific day in a specific classroom. Then (Abdellahi and Eledum, 2017) defined their auxiliary variables as

$\forall i \in I, \forall j \in J, \forall l \in L$

$$y_{i,j,k,l} = \begin{cases} 1, & \text{if a course } k+s \text{ for group of students overlap with} \\ & \text{its prerequite } k \text{ for the same group } l \text{ in the } j^{th} \text{ time} \\ & \text{slot of day } i \\ 0, & \text{otherwise} \end{cases}$$

for $s \in 1, \ldots, n_k - 1, \text{where } k + s \leq n_k$

Finally, (people) defined two further sets of variables:

- z_{im}, which represents the existence of lectures for teacher m on day i
- z_{il}, which represents the existence of lectures for student l on day i

$\forall m \in M$

$$z_{im} = \begin{cases} 1, & \text{if } \sum_{j \in J} x_{i,j,k,l,m,n} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

$\forall l \in L$

$$z_{il} = \begin{cases} 1, & \text{if } \sum_{j \in J} x_{i,j,k,l,m,n} \neq 0 \\ 0, & \text{otherwise} \end{cases}$$

z_{im} and z_{il} are for use with the object function, later. The next thing to be modelled is the restraints. For the university modelled by (Abdellahi and Eledum, 2017), these were:

1. There is no overlapping for courses
2. Each teacher cannot be assigned to more than one course for any given period
3. Each classroom cannot hold more than one course for any given period
4. A student has some courses amount to 18 hours per week. Each course consists of 3 hours and taught in 2 periods (each period is 90 minutes long), which is expressed as the number of slots worked per day.

5. For a student, each course occupies only one slot per day
6. The lectures of each course must be distributed in such a way that there is one day off between them
7. All lectures of a given course in a week must be held in the same classroom
8. The overlap of course with prerequisites for the same group of undergrad students l is permitted (note: (Abdellahi and Eledum, 2017) refer to undergrad students as pre-graduated students)

As an example, the first restraint is modelled as:

$$\sum_{k \in K} \sum_{m \in M} \sum_{n \in N} x_{i,j,k,l,m,n} \leq 1, \forall i \in I, \forall j \in J, \forall l \in L$$

The objective function needs to be either minimised or maximised (dependent on how it is modelled), by changing the values assigned to the variables, whilst ensuring that the constraints are met. In this instance, the total dissatisfaction of teachers and students needs to be minimised, which is equivalent to maximising the number of lectures per day, which implies decreasing the waiting time between lectures (Abdellahi and Eledum, 2017).

$$\begin{aligned}(\text{Total dissatisfaction}) = \text{Teacher}\{&\text{number of lectures per} \\ &\text{day}\} + \text{Regular student}\{\text{number of lectures per day}\} + \\ &\text{Predicted graduate student}\{\text{number of lectures per day}\}\end{aligned}$$

The model produced may not be tractable, meaning that the problem may not be able to be solved in a reasonable period. To make the problem easier to be solved, the model needs to be reduced. (Abdellahi and Eledum, 2017) achieved this by removing the index representing classrooms ($n \in N$) and by adding another constraint, that the number of courses cannot exceed the number of classrooms in a timeslot j in a given day i (constraint 9). To further simplify the model, the term corresponding to the overlapping of courses and their prerequisites has been removed from the objective function, with the related constraint 8.

The model is now:

$$\max\{1.5\sum_{j\in J}\sum_{k\in K}\sum_{l\in L}\sum_{m\in M}x_{i,j,k,l,m}+\sum_{m\in M}z_{im}+\sum_{l\in L}z_{il}\}(\text{objective function})$$

subject to

$$\sum_{k\in K}\sum_{m\in M}x_{i,j,k,l,m}\leq 1 \qquad \forall i\in I\forall j\in J,\forall l\in L \text{ (constraint 1)}$$

$$\sum_{l\in L}\sum_{k\in K}x_{i,j,k,l,m}\leq 1 \qquad \forall i\in I,\forall j\in J,\forall m\in M \text{ (constraint 2)}$$

$$\sum_{j\in J}\sum_{k\in K}\sum_{m\in M}x_{i,j,k,l,m}\leq n_j \qquad \forall i\in I,\forall l\in L \text{ (constraint 4)}$$

$$\sum_{j\in J}x_{i,j,k,l,m}\leq 1 \qquad \forall i\in I,\forall k\in K,\forall l\in L,\forall m\in M \text{ (constraint 5)}$$

$$\sum_{j\in J}(x_{i,j,k,l,m}+x_{i+2,j,k,l,m}=2), i+2\leq 5 \qquad \forall i\in I,\forall k\in K,\forall l\in L,\forall m\in M \text{ (constraint 6)}$$

$$\sum_{k\in K}\sum_{l\in L}\sum_{m\in M}x_{i,j,k,l,m}\leq n_n \qquad \forall i\in I,\forall j\in J \text{ (constraint 9)}$$

$$s\in\{1,\ldots,n_k-1\}, k+s\leq n_k$$

$$\sum_{m\in M}z_{im}\leq 1 \qquad \forall i\in I$$

$$\sum_{l\in L}z_{il}\leq n_l \qquad \forall i\in I$$

$$x_{i,j,k,l,m}, y_{i,j,k,l,m}, z_{im}, z_{il}\leq 0 \qquad \text{(so all variables are non-negative)}$$

The model can be reduced further into two models, one for students and one for teachers. The problem is solved with using an external penalty function method. Penalty functions convert constrained problems to those without constraints by introducing a penalty for violating the constraints. By using a penalty function, (Abdellahi and Eledum, 2017) found a real solution to the unrestrained problem, and then approximated it to a binary solution with an algorithm.

1.2.3 Tabu Search

Tabu search is a metaheuristic that guides a local search to explore the solution space outside of the local optimum, by using a **Tabu list**. A Tabu list is flexible memory structure that stores solutions that are not to be used. Tabu search is used to solve combinatorial optimisation problems, which have a finite solution set. The university timetabling problem has a finite solution set, as there are a finite number of teachers, student groups, time slots, rooms, courses, etc., so there is a finite number of different ways that the timetable can be created. Tabu search makes uses of three main strategies:

- Forbidding strategy: this controls what enters the Tabu list
- Freeing strategy: this controls what exits the Tabu list
- Short-term strategy: this is for managing interplay between the forbidding strategy and the freeing strategy to select trial solutions

Tabu search examines neighbouring solutions, which are solutions that are one "step" away from the current solution. Using the travelling salesman problem as an example, suppose the current solution is B C D A F E, where each letter represents a city. An example neighbouring solution is E C D A F B, where only one swap has occurred, between the positions of B and E. However, the solution E C D F A B is not in the neighbourhood of the current solution, as two swaps have occurred. If a solution has been used, then it is put into the Tabu list, until it meets an **aspiration criterion**. Aspiration criteria provide reasons for a solution to be freed from the Tabu table (freeing strategy). For example, if a using a Tabu move results in a solution better than any other so far, then it can come out of the Tabu table. Much like with genetic algorithms, a fitness function is used to measure how optimal a solution is.

A basic algorithm is:

1. Choose an initial solution i in the solution space S. Set $i' = i$ and $k = 0$, where k is the number of iterations.

2. Set $k = k + 1$ and generate a set of possible solutions $N(i, k)$, where $N(i, k)$ is the set of neighbouring solutions.

3. Choose a best $j \in N(i, k)$, where j is not Tabu (in the Tabu list). If j is Tabu, but meets an aspiration criterion, then choose j. Set $i = j$.

4. If $f(i) > f(i\prime)$ (where f is the fitness function), set $i' = i$. Note this is for the case when a higher fitness function is better. For the case when a lower fitness function is better, set $i' = i$ when $f(i) < f(i')$.

5. Put i into the Tabu table. Update aspiration criteria.

6. If a stopping condition is met, stop. Else, go to step 2.

There are several possible stopping criteria, such as:

- There are no more feasible solutions in the neighbourhood of the current solution
- k is larger than the maximum number of allowed iterations
- The number of iterations since the last improvement of i' is greater than a specified number —- meaning that convergence has been reached, and that there are diminishing returns on finding a more optimal solution
- An optimal solution has been obtained. There would need to be a method to find what the vicinity of the optimal solution, so that upper lower bounds can be set.

Tabu search has been used to solve the university timetabling problem (Awad et al, 2021). They defined four different neighbourhoods:

- Nb1: Randomly choose a particular course and progress to a feasible timeslot, which can produce the smallest cost
- Nb2: Randomly select a room. Also, randomly select two courses for that room. Next, swap timeslots.
- Nb3: Randomly select two times. Next, swap timeslots.

- Nb4: Randomly select a particular time and swap it with another time, in the range between 0 and 44, which can produce the smallest penalty cost.

In order to produce an initial solution which met all the hard constraints, a least saturation degree algorithm is used, where events that are more different to schedule are scheduled first. If that did not produce a feasible solution, then Nb1 is used for a specific number of repetitions, then Nb2 is used for a specific number of repetitions if Nb1 did not reach a feasible solution. Next an improvement algorithm was used with Nb3 and Nb4, specifically adaptive Tabu search. The penalty cost is checked every 1,000 iterations and if the penalty cost has not changed, then two solutions are removed from the Tabu List.

1.2.4 Answer Set Programming

Answer set programming (ASP) reduces problems to logic programs, and then answer set solvers are used to do the search. The logic programs are sets of rules that take the form:

$$a_0 \texttt{ :- } a_1, \ldots, a_m \texttt{ not } a_{m+1}, \ldots, \texttt{not } a_n$$

where every a_i is a propositional atom and `not` is default negation. If $n = 0$ then a rule is a fact. A rule is an integrity constraint if a_0 is omitted. A logic program induces a collection of answer sets, which are recognized models of the program determined by answer set semantics.

To make ASP better for real-world use, some extensions were developed. Rules with first-order variables are viewed as shorthand for the set of their ground instances. Additionally, there are conditional literals that take the form:

$$\texttt{a:} \quad b_1, \ldots, b_m$$

where `a` and b_i are possibly default-negated variables. There are also cardinality constraints which take the form:

$$\texttt{s}\ \{c_1, \ldots, c_n\}\ \texttt{t}$$

where each c_j is a conditional literal. `s` and `t` provide lower and upper bounds on the number of satisfied literals in the cardinality constraint. For example, `2{a(X):b(X)}4` is true when 2, 3 or 4 instances of `(X)` (subject to `b(X)`) are true. Also $N = c_1, \ldots c_n$ binds N to the number of satisfied conditional literals c_j. And objective functions that minimise the sum of weights w_j of conditional literals c_j are expressed as `#minimize{` $w_1 : c_1, \ldots, w_n : c_n$ `}`.

The logic programs can be written in a language called AnsProlog (Answer Set Programming in Logic), which can be used with answer set solvers such as smodels.

1.2.5 Bat Inspired Algorithm

1.3 Comparison of Methods

1.4 Chosen Method

Chapter 2

Requirements

Chapter 3

Design and Implementation

3.1 Program Overview

3.2 Genetic Algorithm

3.2.1 Initial Population

3.2.2 Fitness Function

3.2.3 Selection

3.2.4 Crossover

3.2.5 Mutation

Chapter 4

Testing

Chapter 5

Evaluation

Chapter 6

Conclusion