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

Contents

1 Literature Review **3**

1.1 The Timetabling Problem 3

1.2 Methods 4

1.2.1 Genetic Algorithms 4

1.2.2 Binary Integer Programming 6

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 does not matter.

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 variables 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

$$x_{i,j,k,l,m,n} = \tag{1.1}$$