Timetabling Problem

Guilherme Silva (201603647)
IART
FEUP
Porto, Portugal
up201603647@fe.up.pt

Miguel Duarte (201606298)
IART
FEUP
Porto, Portugal
up201606298@fe.up.pt

Rui Alves (201606746)
IART
FEUP
Porto, Portugal
up201606746@fe.up.pt

***Abstract*—The aim of this paper is to implement and compare Artificial Intelligence algorithms for solving a *Timetabling Problem*, in which a set of events need to be scheduled into rooms at certain time slots, given certain restrictions. The objective is to achieve the best possible allocation efficiently, enforcing the *Hard Constraints* and maximizing the number of *Soft Constraints* verified. The problem will be described and then formulated as an *Optimization Problem*, together with some discussion on the reasoning behind some of the decisions taken during this process. Several algorithms will be investigated, along with testing them in different configurations of these types of problems, which have varying difficulty and complexity. Both the efficiency in reaching a solution and its optimality will be taken into account in the pursuit for the optimal algorithm.**

***Index Terms*—Artificial Intelligence, Scheduling Problems, Optimization Problems, Optimization Algorithms, Hill Climbing, Simulated Annealing, Genetic Algorithms, Python3, pypy**

I. Introduction

The problem that is the subject of study in this paper (a *Timetabling Problem*) will be approached with several different *Optimization Algorithms*, implemented in *Python3*, in order to conclude what is the best approach for solving it using *Artificial Intelligence Algorithms*.

Firstly, the problem will be described in further detail, ensuring that all non-formal aspects of it are well documented. Secondly, it will be formulated as an *Optimization Problem*, with the approach taken in doing so being illustrated in increased depth, providing reasoning for the decisions taken. Thirdly, a research on related work on this topic will be presented, giving some insight on the problem from other points of view.

Finally, a small section will cover the conclusions and future work, by expanding on the already developed tasks and elaborating on the planned approach.

II. Problem Description

The *Timetabling Problem* that is subject of study in this project was designed by Ben Paechter, a professor in the Edinburgh University and consists in a reduction of a typical university course timetabling problem.

In this problem, a given set of events has to be scheduled into one hour time slots, over the course of 5 days with 9 hours of active time each (totaling 45 time slots). The events take place in a given set of rooms, each with its own size (number of sets), and require certain features (which may or not be available in a room). The students attend a given set of these events.

The goal is to assign the events to the available rooms, in a such a way that the given **Hard Constraints** are respected and that the given **Soft Constraints** are verified as much as possible. A given proposed solution is assigned a certain penalty based on the constraints that are not being respected. The optimal solution (if existent for a given input) has a penalty of 0.

A. Hard Constraints

The hard constraints must be respected by any solution. If any of them are not respected, a penalty of Infinity is assigned to the solution.

They are the following:

- Only one event can take place in a room at any given time slot
- A student can only attend one event at the same time
- The room must be big enough for all the students that are attending the event
- The room must satisfy all the features that are required for the event

B. Soft Constraints

The soft constraints are used to evaluate the quality of a valid solution (a solution that respects all the hard constraints), assigning a penalty based on that quality. The assigned penalty is equal to the sum of the penalties of each soft constraint constraint violation.

They are the following:

- A student attends an event in the last time slot of the day. For each student that only attends an event in a given day, a penalty of value 1 is assigned.
- A student attends more than two events consecutively. For each consecutively attended event (above 2), a penalty of value 1 is assigned (e.g. 3 consecutive events result in a penalty of 1, 4 consecutive events result in a penalty of 2, an so on).
- A student has a single class on a day. For each student that attends only one event in a given day, a penalty of 1 is assigned.

III. Problem Formulation

The given scheduling problem can be formulated as an *Optimization Problem*, in which the goal is to minimize the penalty of the given solution.

A. Solution Representation

The optimization problem will be solved using different algorithms, such as *Hill Climbing* and *Genetic Algorithms*. Thus, the solution representation should provide efficient methods for evaluation and manipulation (mutations and cross-over).

For this reason, a solution consists in an array with size equal to the number of events to be allocated. The array's index represents the id of the event. The array value in that index contains a number representing the **time slot identifier**. This number consists in the sum between the room number (between 0 and the number of available rooms - 1) and the time slot number (between 0 and 44) multiplied by the number of available rooms. Example:

Assuming that there are 2 days with 3 time slots each (6 time slots total). There are 5 different rooms. There are 3 events to be allocated and a possible solution is represented by the array [8, 25, 16]. The solution is interpreted in the following way:

- Event 0 was assigned time slot identifier 8. The time slot number is equal to 8 divided by the number of available rooms (8 / 5 = 1). The room number is equal to the remainder of 8 divided by the number of available rooms (8 % 5 = 3).
- Event 1 was assigned time slot identifier 25. The time slot number is equal to 25 divided by the number of available rooms (25 / 5 = 5). The room number is equal to the remainder of 25 divided by the number of available rooms (25 % 5 = 0).
- Event 2 was assigned time slot identifier 16. The time slot number is equal to 16 divided by the number of available rooms (16 / 5 = 3). The room number is equal to the remainder of 16 divided by the number of available rooms (16 % 5 = 1).

This representation can be easily evaluted and manipulated (both mutated and crossed over).

B. Data Representation

In order to represent the problem's data (that is obtained by parsing a given input file, which follows a format as specified in the problem's specification website), an object-oriented approach is being used, being that all of the problem's entities (Events, Rooms and Students) are represented by different classes.

1) Events: An event is composed of the following attributes:

- The event's id number
- An boolean array of required features, which has size equal to the total number of existing features. Each index is assigned the value of true if the event needs that feature and false otherwise.
- A boolean array of attending students, which has size equal to the total number of students. Each index is assigned the value of true if the student is attending that event, being false otherwise.
- A numeric variable containing the number of students that are attending the event (in order to access the number of attendees in constant time instead of traversing the attendance array)

2) Rooms: A room is composed by the following attributes:

- The rooms's id number
- A boolean array of the features the room possesses, which has size equal to the total number of existing features. Each index is assigned the value of true if the room possesses that feature and false otherwise.
- The room's size

3) Students: A student is composed by the following attributes:

- The student's id number
- An boolean array of the events the student is attending, which has size equal to the total number of existing events. Each index is assigned the value of true if the student is attending that event and false otherwise.

The usage of the boolean array data structure to represent the described attributes allows data access in constant time, which makes the verification of constraints and calculation of penalties an efficient process.

C. Solution Neighbor States

In order to solve this optimization problem using the *Hill Climbing* and *Simulated Annealing* algorithms, the concept of a solution's (state) **Neighbor States** has to be defined.

Using the representation that was described in the previous subsection, a neighbor is a state where one of the event's (and only one) is assigned a different time slot identifier (if more than one event's time slot identifier is allowed to be changed, the evaluation process would decay into brute-force computation). For example:

Assuming that there are 2 events and 3 time slot identifiers (between 0 and 2. Only one room is available, only one day will be used with three active hours). If a given solution was [0, 0], the neighbor states would be [1, 0], [2, 0], [0, 1] and [0, 2]. Note that [2, 1], [1, 2] and [2, 2] are **not** valid neighbor states because more than one of the event's assigned time slot identifier needs to be changed to achieve such states.

D. Genetic Algorithm Approach

In order to solve this optimization problem using *Genetic Algorithms*, a set of input variables are manipulated: The population size of each of the generations, the probability of a mutation occurring in a solution and the number of elite solutions that are preserved between generations.

A generation consists in an array of solutions, with size equal to the generation's population size.

The process of producing a new generation ensures that the population size remains constant between generations.

Given that fact, in order to generate a new generation, all the population's solutions fitness is computed. Using that fitness, a probability of being chosen is assigned to each solution (for the selection process), being that probability proportional to how good the solution is. A set of numbers equal to the population's size is randomly generated in order to decide which solutions will be chosen to generate the next generation (according to the probability assigned to each solution). The chosen solutions are then randomly combined in pairs, generating two new children solutions that may or may not be mutated. This process is repeated until the new generation is complete.

IV. Related Work

Scheduling problems are extremely prevalent in computer science to help address problems in a variety of areas where some limited resource needs to be distributed over a number of users.

One such resource can be the access to airport docks and runways - This scheduling problem has the additional difficulty of having to provide real time updates. The paper by V. Ciesielski and P. Scerri [1] tries to solve this problem by employing a genetic algorithm that provides real time solutions and takes advantage of previously generated populations when the requirements of the solution change (a departure or a new request for landing changes the solution's requirements).

Although the problem tackled by this report is suitable for a genetic approach, the participants of the 1st International Timetabling Competition chose to use optimization algorithms, such as Tabu Search [3] or Simulated Annealing [2], in combination with a series of defined heuristics and an heuristics based method of generating the initial assignment.

V. Conclusions and Future Work

As of yet, the problem has been completely formalized - it has been formally described, and formulated as an optimization problem (a *Solution Representation* has been specified and the chosen data structures to store that input data have been stated).

The chosen programming language to developed the solving algorithms was Python3.

In order to study the best possible approach for solving the *Scheduling Problem*, the following algorithms will be the subject of various tests:

- Hill Climbing
- Simulated Annealing
- Genetic Algorithms

In order to evaluate the implemented algorithms, they will be assessed both in terms of the achieved solution's quality and in terms of the algorithm's performance, measuring the execution time to reach the optimal solution (if existent). When it comes to space efficiency, in the Hill climbing and Simulated annealing algorithms the number of evaluated states will be taken into account and in the Genetic Algorithm the number of generations will be analyzed.

References

[1] V. Ciesielski and P. Scerri, ”Real time genetic scheduling of aircraft landing times,” 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360), Anchorage, AK, USA, 1998, pp. 360-364.

[2] Philipp Kostuch, ”Timetabling Competition - SA-based Heuristic” in 1st International Timetabling Competition, January 2003.

[3] Halvard Arntzen, Arne Lokketangen, ”A local search heuristic for a university timetabling problem” in 1st International Timetabling Competition, January 2003