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*—TODO: Abstract**

***Index Terms*—Artificial Intelligence, Scheduling Problems, Optimization Problems, Optimization algorithms, Hill climbing, Simulated annealing, Genetic Algorithms, Python3, Pypy**

I. Introduction

TODO: Introduction

II. Problem Description

The *Timetabling Problem* that is subject of study in this project was designd 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 timeslots, over the course of 5 days with 9 hours of active time each (totaling 45 timeslots). 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 as respected 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 value equal to infinity is assigned to the solution, being them the following:

- Only one event can take place in a room at any given timeslot
- A student can only attend one event at the same time
- The room must be big enough for all the student 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, being them the following:

- A student attends an event in the last timeslot 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 (or to maximize the symmetrical of the penalty).

A. Solution Representation

The optimization problem will be solved using different algorithms, such as Hill climbing and Genetic algorithms. To do so, the solution representation should allow an efficient way to be evaluated and manipulated (mutated and crossed 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 **timeslot identifier**. This number consists in the sum between the room number (between 0 and the number of available rooms - 1) and the timeslot number (between 0 and 44) multiplied by the number of available rooms. Example:

Assuming that there are 2 days with 3 timeslots each (6 timeslots 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 timeslot identifier 8. The timeslot 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 timeslot identifier 25. The timeslot 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 timeslot identifier 16. The timeslot 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 may be easily evaluted and may also be easily 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 ***INSERIR REF***), an object-oriented approach is being used, being that all of the input entities (Events, Rooms and Students) are represented by different classes.

An event is composed by 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 1 (true) if the event needs that features and 0 otherwise.
- A boolean array of attending students, which has size equal to the total number of students. Each index is assigned the value of 1 (true) if the student is attending that event.
- 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)

1) Events:

An event is composed by the following attributes:

- The event's id number
- A boolean array of required features, which has size equal to the total number of existing features. Each index is assigned the value of 1 (true) if the event needs that feature and 0 otherwise.
- A boolean array of attending students, which has size equal to the total number of students. Each index is assigned the value of 1 (true) if the student is attending that event.
- 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 1 (true) if the room possesses that feature and 0 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 1 (true) if the student is attending that event and 0 otherwise.
- The room's size

The usage of the boolean array data structure to represent the described attribute allows for data access in constant time, which makes the constraints verification and penalty calculation 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 timeslot identifier (if more than one event's timeslot 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 timeslot 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 timeslot 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 ocurring in a solution, the tournament size (given the fact that the tournament is being applied in the selection process of the algorithm) 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 generating a new generation assures that the population size remais constant between generations. Given that fact, in order to generate a solution, two parents are selected from the current generation using the tournament method (that is, a random subset with a given size is extracted from the population and evaluated, choosing the best individual of that subset). The parents are then crossed over in order to generate a child solution, that has a probability of being mutated. This process is repeated until the number of individuals in the new generation is equal to the previous one (note that if elites are being preserved, this process may occur a smaller number of times).

IV. Related Work

TODO: Related Work

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 existant). 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] ”Unblock me FREE.”. Google Play. February 28, 2019.https://play.google.com/store/apps/details?id=com.kiragames.unblockmefree.

[2] Fogleman, Michael. ”Solving Rush Hour, the Puzzle.”. July, 2018. https://www.michaelfogleman.com/rush/.

[3] ”Bitboard.”. Wikipedia - The free Encyclopedia. December 6, 2018. https://en.wikipedia.org/wiki/Bitboard.

[4] Littman, Michael. ”Programming Assignment P1 - What A* Rush”. Priceton. 2012. https://www.cs.princeton.edu/courses/archive/fall12/cos402/assignments/programs/rushhour/.

[5] Findling, Rainhard. ”The RushHour Puzzle an Artificial Intelligence Toy Problem.”. April 4, 2012. http://geekoverdose2.rssing.com/browser.php?indx=39804402&item=1.