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: UPDATE THIS*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

TODO: UPDATE THISThe 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.

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. Project Implementation

The implemented project consists in program developed using Python3, accessible with via a command-line interface. In order to run the program, the following syntax is used (may be ran using Python3 or PyPy):

python3 optimization.py <input_file>

The input file assumes the following strucutre:

- A line containing the number of events, the number of rooms, the number of features and the number of students
- For each room, a line containing the room's size
- For each student, for each existing event, a line containing a 0 (meaning the student does not attend the event) or a 1 (meaning the student attends the event)
- For each room, for each existing feature, a line containing a 0 (meaning the room does not satisfy the feature) or a 1 (meaning the room satisfies the feature)
- For each event, for each existing feature, a line containing a 0 (meaning the event does not need the feature) or a 1 (meaning the event needs the feature)

The program offers a set of menus, allowing the user to choose which algorithm they want to solve the input problem, presenting the obtained solution and the execution time.

When Genetic Algorithm is chosen, the maximum number of desired generations is specified by the user. In each iteration, the current best solution in the population is presented and a verification is made is order to test if the optimal solution was reached. If so, the algorithm stops. After the optimal solution is reached (or if the maximum number of generations is reached), the execution time and number of generated solutions are presented to user.

When the Hill Climbing or Simulated Annealing algorithms are chosen, the algorithm execution starts and the current solution is displayed in each iteration. After the optimal solution is reached or a local maximum is reached (and if the current temperature is equal to 0 in the simulated annealing algorithm), the execution time to reach the solution and the number of explored states are presented to the user.

In order to improve the algorithms performance, the initial solutions are being pre-computed instead of being randomly generated, as will be further expanded in section VII

VI. Optimization Algorithms

As determined beforehand, the following algorithms were implemented:

- Hill Climbing
- Simulated Annealing
- Genetic Algorithm

A. Hill Climbing Algorithm

The **Hill Climbing** algorithm is an iterative algorithm that, based on a given initial solution, analyzes all of the solution's neighbor states (the concept of neighbor state was detailed in subsection III-C), evaluating them. Then, it proceeds to choose the best neighbor state as the next state to be explored, until a maximum is reached (the algorithm is, for this reason, deterministic).

This algorithm has the problem of not always being able to find the optimal solution for a given input, since the termination condition is the non-existance of better neighbor states, that is, a local maximum.

B. Simulated Annealing Algorithm

The **Simulated Annealing** algorithm has a simillar behaviour to the Hill Climbing algortihm, starting with an initial solution and iteratively analyzing all of the solution's neighbor states, evaluating them. However, it does not always choose the best neighbor state in each iteration. Instead, there is a probability to choose a different solution than the best, in order to avoid determinism and stopping optimization in a local maximum. That probability is commonly called a *temperature*, that decreases in each iteration (*temperature annealing*). The algorithm optimization termination condition is after the temperature has reached 0 and a local maximum has been reached (there are no better neighbor states to explore).

Simillarly to the Hill Climbing algorithm, this algorithm does not guarantee an optimal solution for a given input. However, since the probability of not always choosing the best possible state removes the determinism factor, it is possible to find maximums other than the nearest local maximum.

C. Genetic Algorithm

The **Genetic Algorithm** is based on the species evolution theory. The algorithm groups a set of solutions (a *generation*), which represents a population of solutions in a given point in time. The optimization process consists in iteratively computing news generation (based on the current generation), using concepts such as *evaluation*, *selection*, *crossover* and *mutation*.

The *evaluation* process consists in computing how good (the *fitness*) of each of the generation's individuals, which is the variable that is being optimized and classifies how good a solution is.

The *selection* process consists in selecting a subset of the current generation (based on their fitness, prioritizing the most fit solution), which will be used to generate the following generation. This process is usually made by generating a set of random numbers (equal to the generation's population size), which dictate which solutions to select (the probability of a solution is chosen is proportional to its fitness).

The *crossover* process consists in selecting random solutions (from the set obtained in the selection process) and combining them, generating two new solutions (children solutions). There

are different ways of crossing over two solutions, such as *k-point* crossover and *uniform* crossover.

The *mutation* process consists in changing the solutions that resulted from the crossover process, based on a given mutation probability. This process allows to increase the variability of the new generation.

The algorithm may also use other concepts, such as *Elitism*, which preserves a number of best solutions from one generation to another, in order to avoid the loss of very fit solutions.

The algorithm allows the manipulation of a set of variables, such as the population size of each generation, the probability to mutate a solution and the number of elite solution that are preserved from the previous generation. The optimal values for these variables depends on the problem instance.

VII. Experiments and Results

In order to study the performance of the different optimization algorithms in the given problem, aswell as their optimal parameterizations, a set of experiments were done.

In a first iteration, the initial solutions (for all the different algorithms) were being randomly generated. However, as the problem input size increased, noone of the algorithms were able to generate valid solutions (that would satisfy all the problem's hard constraints), which resulted in solutions with no value. For that reason, in a second iteration, an algorithm to generate the initial solutions was implemented (the algorithm will be explained in subsection VII-B), which highly improved all of the algorithms performance.

In order to run the experiments under different input conditions, a set of 23 different input scenarios were generated (using a script that was developed for that purpose), which are grouped into 5 different "difficulty levels" according to the size of the input data (number of events, number of features, number of students, ...). Not all of the different input scenarios have an optimal solution (violating no soft constraints whatsoever).

A. Input Scenarios Categorization

The input scenarios (total of 23 different scenarios) were categorized into 5 different difficulties, based on the size of the input data, that is, the number of events that are taking place, the number of students attending the events, the number of features that exist (offered by rooms and required by events) and the number of available rooms and their size.

- *Very Easy* - About 5 events, 15 to 20 students, 3 to 5 available rooms and 3 to 5 existing features
- *Easy* - 12 to 18 events, 30 to 42 students, 3 to 6 available rooms and 5 to 8 features
- *Medium* - 20 to 25 events, 90 to 105 students, 6 to 8 available rooms and 5 to 10 features
- *Hard* - 24 to 26 events, 240 to 300 students, about 5 available rooms and 5 to 10 features
- *Very Hard* - 45 to 50 events, 335 to 410 students, 12 to 14 available rooms and 8 to 10 features

The different input sizes allows the studying of the algorithms' performance under several distinct scenarios.

Firsly, a study of the best parameterizations for the simulated annealing and genetic algorithms will be made (the hill climbing algorithm has no variable parameterizations).

Secondly, the three algorithms quality is going to be evaluated and compared, based on their resulting solutions.

B. Initial Solutions Generation

TODO - GUILHERME

C. Simulated Annealing Algorithm Parameterizations

The simulated annealing algorithm makes use of two different parameterization variables: the initial temperature and the annealing step. In order to understand which parameterizations achieve the best results, a set of 10 experiments with different temperatures and annealing steps were made with inputs of different difficulties:

Very Easy Input				
Initial Temperature	**Annealing Step**	**Penalty**	**Time (s)**	**Num. Explored States**
50	**0.5**	3.20	1.3286	55770
50	**1**	13.00	0.6552	28275
50	**2**	13.90	0.3832	16250
50	**3**	14.30	0.2799	11830
60	**0.5**	3.50	1.8073	78650
60	**1**	5.90	0.9054	38935
60	**2**	6.30	0.4607	20215
60	**3**	7.80	0.2900	12610
70	**0.5**	4.60	1.8778	81770
70	**1**	4.90	0.8836	38545
70	**2**	6.10	0.5150	21775
70	**3**	10.00	0.3853	16250
80	**0.5**	3.60	2.0703	86450
80	**1**	5.20	1.2076	52650
80	**2**	5.30	0.5157	22295
80	**3**	7.70	0.4242	18200

Fig. 1. Simmulated Annealing Results on very easy inputs

Easy Input				
Initial Temperature	**Annealing Step**	**Penalty**	**Time (s)**	**Num. Explored States**
50	**0.5**	4.1	18.3034	318150
50	**1**	5.3	9.7615	160650
50	**2**	17.5	5.1272	81900
50	**3**	18.3	3.4906	56700
60	**0.5**	3.4	22.9546	381150
60	**1**	6.1	11.5388	192150
60	**2**	6.8	6.1121	100800
60	**3**	17.2	5.5218	78750
70	**0.5**	4	58.9258	444150
70	**1**	4.1	13.686	223650
70	**2**	6.1	8.2993	113400
70	**3**	19.2	6.2968	78750
80	**0.5**	5.2	33.2494	507150
80	**1**	5.2	18.6444	255150
80	**2**	5.3	7.7928	129150
80	**3**	7.1	5.4041	88200

Fig. 2. Simmulated Annealing Results on easy inputs

Medium Input				
Initial Temperature	**Annealing Step**	**Penalty**	**Time (s)**	**Num. Explored States**
50	**0.5**	156.5	147.9634	686800
50	**1**	224.2	63.8937	346800
50	**2**	269.8	36.6033	176800
50	**3**	333.1	24.0615	122400
60	**0.5**	42.6	155.876	822800
60	**1**	73.3	84.891	414800
60	**2**	85.2	38.859	210800
60	**3**	198.5	23.996	142800
70	**0.5**	30.5	153.6531	958800
70	**1**	74.3	94.41	482800
70	**2**	145.1	43.2957	244800
70	**3**	158.4	32.7321	170000
80	**0.5**	35.9	174.7215	1094800
80	**1**	84.9	95.8574	550800
80	**2**	87.5	46.2177	278800
80	**3**	98.1	32.3182	190400

Fig. 3. Simmulated Annealing Results on medium inputs

As expected, lower penalties (better solutions) were achieved with higher starting temperatures and lower annealing steps, since that results in more algorithm iterations. However, as the initial temperature increased and the annealing step decresed, the execution times and the number of explored states proportionally increased (as expected, since more algorithm iterations are being executed). These results were observed with inputs of all different difficulties.

D. Genetic Algorithm Parameterizations

The genetic algorithm makes use of three different parameterization variables: the generations population size, the mutation probability and the number of elite solutions preserved between consecutive generations. In order to understand which parameterizations achieve the best results, a set of 10 experiments with different parameterizations were made with inputs of different difficulties, with a maximum of 300 generations per experiment (may be less than this value if optimal is reached).

Firstly, the population size was manipulated (maintaining constant values for the other variables, namely a mutation probability of 15% and a number of preserved elite solutions equal to 10% of the population size). The obtained results were the following:

Medium Input			
Population Size	**Penalty**	**Time (s)**	**Num. Explored States**
30	194.3	2.2631	6030
40	187.9	2.8781	8040
50	176.2	3.5871	10050
60	168.8	4.2332	12060
70	169	4.9795	14070
80	170.2	5.5467	16080
90	158.3	6.4374	18090
100	154.6	7.3088	20100
110	164.9	8.0249	22110
120	169	8.607	24120
130	166.3	9.3345	26130
140	164.3	10.4152	28140
150	166.4	10.6161	30150

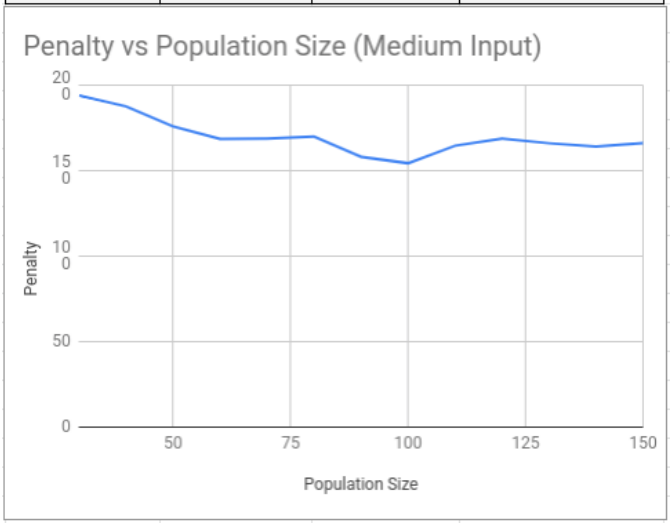

Fig. 4. Genetic Algorithm Population size influence on medium inputs

As expected, the solution penalty decreased as the population size increased (since more solutions were explored and the different generations had more variability). The best performance was achieved, in all tested input difficulties, when the population size was approximately 100. The number of explored solutions and the execution time increased proportionally to the population size.

Secondly, the mutation probability was manipulated (maintaining constant values for the other variables, namely a population size of 100 and a number of preserved elite solutions of 10). The obtained results were the following:

Insert Chart

Insert conclusion

Thirdly and finally, the number of preserved elite solutions between consecutive generations was manipulated (maintaining constant values for the other variables, namely a population size of 100 and a mutation probability of 12.5%). The obtained results were the following:

Insert Chart

Insert conclusion

E. Algorithms results and comparison

TODO

VIII. Conclusions and Future Work

TODO

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