# An approach to the (High) School Timetabling Problem based on Genetic Algorithms

Daniel Gomes and Rui Andrade[1]

Planning and Scheduling Methodologies
Faculty of Engineering, University of Porto
Rua Dr Roberto Frias S/N, 4200-465 Porto, Portugal
{up201306839, ei12010}@fe.up.pt

**Abstract.** The (High) School timetabling is an topic of most interest for both its practical importance and hardness in to solve. In this paper this problem is addressed through the XHSTT XML file format used on the third International Timetabling Competition and two slight different approaches with Genetic Algorithms. For test and benchmarking of these algorithms, datasets made available by this competition website were used.

**Keywords:** genetic algorithms, high school timetabling, XHSTT file format, International Timetabling Competition

## 1 Introduction

The (High) School Timetabling Problem **(HSTTP)**, in its simplest form, consists on the production of a schedule, such that no teacher or class attends to more than one lesson in the same time slot, being referred as the Class×Teacher Timetabling Problem. Yet, another type of constraints may also be considered, e.g, students or teachers preferences, as well other type of resources can be included e.g, classrooms. Furthermore a particular timetabling problem specification may allow a lesson to be broken down into smaller lessons that possibly may happen in different rooms with different teachers.

Although it represents a time consuming task when performed manually, it still happens, as existing software is not always capable of handle all the requirements or find the optimal solutions. The HSTTP is for so a topic of most interest for both fields of Artificial Intelligence and Operation Research, currently being classified as a NP-Complete scheduling problem.

Something to be noticed by now is that HSTTP is vast and, therefore, when studying a possible solving method, testing under real world data, performing comparisons with other approaches and infer conclusions is essential. Nonetheless, this is not always a trivial task for this problem as timetabling data has not been abundant. To mitigate this problem, the XML format XHSTT was created in 2009 by a working group led by Gerhard Post. This xml-based file format that was latter used by the International Timetabling Competition **(ITC)** 2011

and aims to be a standard for both describe a concrete problem and store its solutions. Also, under the ITC website multiple archives are provided that could be used for test and benchmarking.

With this, the ground of this work is to present a possible approach to the High School Timetabling Problem when following the formulation model proposed by XHSTT (section 3). In this sense, two different variations of Genetic Algorithms will be explored (section 2). The first, mas use of an event as the gene while in the second, the resources and time slot required are considered to this purpose. Yet, in order to run and benchmark this algorithms certain other tasks, that could be implemented independently from the solving approach chosen, must be firstly handled. That will be discussed on section 5. Finally the results of the experiments ran with two datasets from the ITC archives will be analysed (section 6) and the final conclusions presented.

## 2 Solving approaches

These days, to solve Timetabling problems, two approaches are have been taken with moderate success.

The first, constraint programming **(CP)**, makes use of constraint propagation systems that continuously reduce the variables domains. These make also use of backtrack search techniques. As its main advantage, the constraints specification made in a declarative manner make the program simple to read, understand and consequently easy to adapt. Nevertheless, modeling soft constraints is not a trivial task with CP as also, to make it able it to improve a given initial solution.

The second approach, local search methods, encompasses algorithms such as Simulated Annealing, Tabu Search or Genetic Algorithms where constraints are represented by a cost function. Through the use of search heuristics, better neighbourhood solutions are found and the cost value minimized. These, in opposition to CP these tend to be good at improving already feasible solutions and having difficulties to find them. With this procedures, the modeling of hard constraints is achieved through the use of a much larger cost in comparison to the soft constraints associated cost.

Since with XHSTT file format the problem specification is written in the XML file, including the constraints to be applied, being able to tweak the solver isn't much of a priority. Also, XHSTT defines a limited set of possible constraint types and in one particular case each can be applied as soft or hard. Finally, XHSTT already includes the definition of a cost function for each solution. This way, an approach through local search algorithms seems more appropriate.

## 3 Problem formulation

As before stated, in this work the XHSTT was used. This file format by it self defines a generic model for the problem formulation that allows differently HSTTP to be specified. Ahead the problem is presented as imposed by this framework.

### 3.1 Resources

Resources can be almost anything, but typically are classes, teachers or rooms. At the moment of specification a resource may or not be already assigned to an specific event. In the latter case, the resource type is provided, meaning that in the solving phase a resource of this type should be scheduled to this event. Each resource is associated to an event through a role allowing for multiple resources of the same type to be assigned to the same event, e.g., Resource: John Doe, Resource Type: Teacher, Role: Auxiliary Teacher.

### 3.2 Scheduling constraints

In XHSTT there are sixteen different types of constraints, five of them were taken into account in this work, allowing the test of the implemented solution against a set of artificial datasets provided by the XHSTT-2014A archive. In a particular timetable, multiple constraints of each type could be used.

Each specified constraint is applicable to groups of events or groups of resources.

1. Basic Scheduling Constraints
   **Assign Resource:** assign a resource to all roles with no resource defined yet of the applicable events
   **Assign Time:** assign a time slot to all applicable events
2. Event constraints
   **Prefer Resources:** assign resources from a specific set of resource groups to the applicable events
   **Prefer Times:** assign time slots of the defined time slot groups to the applicable events
3. Resources constraints
   **Avoid Clashes:** the applicable resources must not be assigned to different events in the same time slot

### 3.3 Objective function

The objective function $(\gamma)$ assigns to each solution a cost pair, being each value calculated in order of the sum of the disrespected requirements **(Deviation)** imposed by each soft and hard constraints **(c)** respectively. In addition to specify to which set of entities it is applicable, the constraint definition also establishes: a **Weight**, a **Cost Function** (Linear, Quadratic or Step) allowing the rescaling the constraint Deviation and if is it should be considered as hard **(HC)** or soft **(SC)** constraint.

To constraint an intermediate Cost is calculated as follows:

$$Cost(c) = Weight(c) * CostFunction(c)(Deviation(c)) \tag{1}$$

and objective function value is therefore:

$$\gamma = (\sum_{c \in HC} Cost(c), \sum_{c \in SC} Cost(c)) \tag{2}$$

## 4 Solving with Genetic Algorithms

Genetic Algorithms are based on the natural selection process and, in the field of Artificial Intelligence, represent an heuristic for the search of an optimal solution in a given problem domain. These belong to the class of Evolutionary Algorithms and, as in the Darwin's theory, include the four characteristics: selection, crossover, mutation and inheritance.

The algorithm start with an initial population i.e., a set of initial solutions where each different one is represented by a unique chromosome. A chromosome is a set of genes that represent the characteristics of a solution. This is typically is implemented through a vector of bits but could also by done with any other type of data structure.

To each solution a fitness function is associated, that serves as metric to the selection phase, where the best ones are chosen. This process may be stochastic – e.g, using the roulette method – where the probability of being chosen is proportional to the fitness value, elitist where the nth best solutions are selected or mixed.

The selected solutions are then paired and breed through crossover. Here, also different approaches exist such as, defining n crossing points and with a certain probability each chromosome segment is swapped with the other respective. Other possibility is to apply the same rational at the gene level – Uniform crossover.

Then, mutation is applied. With a small probability, new chromosomes' genes are mutated. This means, if using a binary vector swap one one bit else, for example, the the selected gene can be replaced by a random new one.

Once this steps are completed, a new population exists. By Holland's schema theorem it is guaranteed that at each iteration better solutions tend to appear. And, given the stochastic nature of this process it is guaranteed that it does not get stuck into local maximums and does, given enough iterations, it reaches the optimal solution.

### 4.1 Implementation to HSTTP

In the implemented versions of the algorithm a chromosome is a scheduling solution i.e., a set o event-solution. An event-solution is an specified event with the required resources and time slot assigned. The initial population is a set of random solutions. To calculate the fitness value the XHSTT cost function pair is used. First the cost is turned into a fitness by subtracting it to the maximum cost o the population. Then the pair is converted into an integer by shifting the hard constraint fitness value to the left **k** units and adding with the soft constraints value. Where **k** is the number of digits in the highest soft constraint fitness value. This ensures that hard constraints influence more cost and thus it is expected to be found a feasible solution first. Finally the fitness value is squared to better distinguish the solutions with different costs.

To choose the best solutions, a mix of elite and stochastic selection was used. Elite selected solutions transit directly into the following population. Crossover

is performed with Uniform Crossover. Mutation consists on the replace of a gene through a random new one.

Two slight different interpretations of what a the gene in this chromosome were made. In the first, a chromosome is event-solution. In the second, a chromosome is the time and the assigned resources of each event-solution.

## 5 High School Timetabling Engine

Although the focus of this work stays the on the methods to solve the HSTT problem, a set of other solving method independent functionalities such as import XHSTT file, evaluate a solution and produce statistical data are required to effectively put these algorithms into execution and analyse its results. For this reason it makes sense to think of a modular program that offers transparently such features and supports a separate implementation of, ideally, any kind of solver.

This idea is supported by Jeffrey H. Kingston with its KHE High School Timetabling Engine where out of the box its KHE14 solver is provided. Yet, in this it work was opted to design and develop a simpler, more modular engine. This allowed a better understating and control of all components. This makes possible, in future work, to study some of these in particular, such as the solution evaluator and possibly provide better methods, as this is also complex algorithm per se.

The High School Timetabling Engine, was designed with the main components:

1. **Input/Output:** has the capabilities of import an XHSTT schedule specification and save back obtained solutions
2. **Evaluator:** is capable of evaluate a given solution following the rules proposed by the file format.
3. **Data collector:** when studying these algorithms it is important not only to collect the final results but also intermediate ones. This module should be used by the solver to report its intermediate results. This could implement features such as real-time chart plotting.
4. **Solver:** this is the core module. Uses the functionalities@ provided by the remaining in order to implement a certain solving algorithm.

In this work, the implemented solver is parameterizable allowing to choose one of each variation of the proposed genetic algorithms. The the remaining modules were implemented in its minimum form allowing the import of XHSTT files, execution of the solver and collection, in console, of intermediate cost function results.

## 6 Tests and results

During the development and test phase, several characteristics and parameters of the algorithm were explored: the use of a mixed selected population, the square

of the fitness function, the size of the population, and the size of the elite group, number of iterations.

After several executions, a set of the most interesting values for each probabilistic parameter where chosen. The testes were run and the results are presented bellow.

The first testes used the ArtificialSudoku4x4 dataset from the XHSTT-2014A archive. This has sixteen events and four time slots, teachers, rooms and classes. Ten constraints are applied of the four types: Assign Resource, Assign Time, Prefer Resource and Avoid Clashes. For each test 10 executions were performed.

| **Gene** | **CP** | **MP** | **WC** | **SR** | **Min(NI)** | **Avg(NI)** | $\sigma$**(NI)** |
|---|---|---|---|---|---|---|---|
| **Event-solution** | 30% | 6% | 2.0 | 90% | 58 | 478,7 | 326,66 |
| | 15% | 6% | 3.0 | 50% | 95 | 636,7 | 405,59 |
| | 30% | 2% | 2.0 | 70% | 111 | 658 | 352,43 |
| **Time/resource** | 30% | 6% | 3.0 | 60% | 77 | 482 | 448,45 |
| | 15% | 6% | 3.0 | 40% | 92 | 662,4 | 437,51 |
| | 30% | 2% | 3.0 | 10% | 78 | 2636 | 3895,272 |

**Table 1.** Results of the ArtificialSudoku4x4 dataset tests. CP: Cross probability, MP: Mutation probability, WC: worst cost, SR: success ratio (is considered a success when one solution is found with zero cost is found), NI: number of iterations. The initial population best solution had on average 12 of cost with an standard deviation of 1,12.

The second used the Abramson15 dataset. This has four hundred and fifty events, thirty time slots and fifteen, teachers, rooms and classes. The two types of constraints are applird: Assign Time and Avoid Clashes. For the first three pairs o tests five runs were performed, four for the final two.

| **Gene** | **CP** | **MP** | **Avg(BC)** | **(C)** |
|---|---|---|---|---|
| **Event-solution** | 30% | 6% | 322 | 3,53 |
| | 15% | 6% | 328,8 | 2,58 |
| | 30% | 3% | 288,8 | 5,31 |
| | 30% | 2% | 278,5 | 2,08 |
| **Time/resource** | 30% | 6% | 320,6 | 2,70 |
| | 15% | 6% | 332,4 | 4,77 |

**Table 2.** Results of the Abramson15 dataset tests. CP: Cross probability, MP: Mutation probability, WC: worst cost, BC: obtained best cost. The inial population average cost is 466,25 with standard deviation of 10,25.

Something that can be noticed by these results is that, if for the first dataset the event-solution as gene tends to have better results, with the abramson the reverse is observed. This might be explained by the fact that, as the gene on the time/resource is smaller, the mixing happens more aggressively. In a smaller

problem as sudoku4x4, this may this may represent much accelerated search speed which is evidenced by the larger range of outcomes.

All these testes were executed on an Asus laptop with an Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 6GB of memory RAM with the Arch Linux distribution, Manjaro Daniella, as its operating system.

## 7 Conclusions

With this work, a complete solution for solving the (High) School Timetabling problem, where XHSTT had a key role, was delineated. This format defines a most generic conceptual model that allows the the specification of vast number of scenarios. This, allows the development of a solving algorithm independently from the particular problem characteristics. Also, with it, describing a problem becomes easier given the declaritivity of writing an XML file.

Regarding the proposed solving methods through Genetic Algorithms, these revealed unsatisfactory. When tests to find a solution with a problem closer to a real world one where performed, a feasible solution never was reached, and possible would take days or weeks to achieve it. Moreover, even to achieve such results, a large number of intermediate tests had to be performed to find the most promising values for the algorithm parameters. It also seems that this values are related with the concrete problem size and with the number of constraints involved, which implies that for different datasets different parameters values would should be used. This, in part, nullifies one of the XHSTT advantages: the approach / problem separability.

Some inspirational conclusions can be taken from this work, also. One of the major problems found with Genetic Algorithms is that as iterations pass by, its improving rate keeps decreasing. If in the first ones, each iteration could improve a dozen in the cost function, after about a hundred, only steps of one would be taken, at the most. This could be justified by the fact that the search space is covered in a stochastic manner, when having a good solution it is harder to find a better one. Although this seem a quite obvious consequence, it shouldn't be taken as inevitable as when looking to natural evolution processes our experience tell us that the exact opposite may happen. In order to surpass this, methods with learning capabilities such, as neural networks, could be integrated with the genetic algorithms that would provide an expected best value to each solution. Then, the fitness function would contemplate both the real fitness value as well the learned expected best value.

Methods such as the ones proposed above can now easly be implemented using the engine developed in this work.

## References

1. Wojciech Legierski: Search Strategy for Constraint-Based Class–Teacher Timetabling. Practice and Theory of Automated Timetabling IV, 247– 261 (2003)

2. George H.G. Fonseca, Samuel S. Brito, and Haroldo G. Santos: A Simulated Annealing Based Approach to the High School Timetabling Problem. Intelligent Data Engineering and Automated Learning - IDEAL 2012, 540–549 (2012)
3. Eugénio de Oliveira: Inteligência Artificial: Métodos de resolução de problemas, 43–91 [Powerpoint slides] (2015)
4. Jeffrey H. Kingston: KHE14: An Algorithm for High School Timetabling. (2014)