Solving the School Timetabling Problem with Genetic Algorithm

Riccardo Scilla
University of Trento
riccardo.scilla@gmail.com

***Abstract*—The objective of this work is to evaluate a genetic algorithm that uses an direct representation to solve the school timetabling problem. The problem is designed as a multi-objective optimization with constraints.**

I. Introduction

The school timetabling problem is a common problem faced by many schools. It is a combinatorial NP-hard problem where the main goal is to allocate teachers in the slots of the week timetable and at the classes they are assigned to. In most schools, timetables are manually designed and require a significant time to be computed with results that often do not satisfy all the requirements. These requirements may be structural (like clashes), imposed by the law or tailored for the specific school we are working with. For example some teachers may teach in different schools and have only few days available that have to be considered when creating our timetable.

This work shows a solution for timetabling problem using a genetic algorithm, tuned from [1]. The requirements are designed as hard and soft constraints that need to be solved, minimizing the fitness function. An ad-hoc method to generate good initial individuals is described, together with specific genetic operators. Different tests are performed to evaluate the performance of the algorithm and the results are shown.

II. Genetic Algorithm

The genetic algorithm is a classical evolutionary algorithm, that draw inspiration from the principles of natural evolution. The main steps of the algorithm are:

1) Encoding: find an handy way to represent the timetables. This procedure is extremely problem-dependent.
2) Initial Population: find a number of individuals that are randomly spread in the search space. These will be evolved to find the best solution.
3) Fitness evaluation: find a way to evaluate a solution depending on the requirements we want to fulfill.
4) Selection: select the individuals that will be used to generate the new population.
5) Crossover and Mutation: how can we generate new solutions from the current ones.
6) Replacement: which individuals should survive.
7) Termination criteria: what is the condition that tells if we have found an optimal solution.

All these steps are designed for the School Timetabling Problem and described in the following subsections.

A. Encoding

Two main types of chromosome encoding techniques exists for timetabling problems: direct and indirect representations [2]. A direct representation can be a vector that directly encodes the solution, while an indirect representation encodes an abstract form of the solution that can be reconstructed following a set of instructions.

A matrix representation allows to directly represent a timetable which satisfy certain constraint violations and allows for an easy implementation of genetic operators e.g. swapping. For these reasons, the method used in this work is a direct encoding, where each individual in the population is a timetable that is represented using a two-dimensional matrix.

A school timetable is defined as a set of timeslots, called periods. Each period must be occupied by a tuple that consists of a combination of class-teacher. Each period is described with two integers: the first represents the day and the second the timeslot in that day. For example the first hour on monday is the period 11 and the third hour on friday is 53.

The matrix columns of the timetable represent the periods and each row represents a class. Each cell in the intersection of a row and a column, stores the information of the teacher that is assigned to that class in that period. An example of the encoding used is shown in table I. This representation allows to have the full timetable in a compact form which also simplify the process of generation of the initial population and the computation of the constraints violation.

B. Initial Population

Given that the problem requires a complex encoding to be fully represented, the definition of good individuals for the initial population is fundamental. The methodology used to form the initial population of N individuals exploits as many information as possible to generate individuals that minimize the feasibility constraints.

The information used to generate each individual is given by the user in file that specify the $prof\_data$, i.e. a set of [teacher, class, subject]. Each of these tuple is then used to create the $place\_data$, that is a more informative set used to place the teachers in the new timetable individual.

Every tuple of the $prof\_data$ is split in meetings in a predefined way, depending on the specific subject and the number of meetings in the week that it involves. Each meeting is described with a $type$, $possibilities$ set, $saturation$ level and a $fixed$ flag.

	11	12	13	14	15	16	21	22	...	56
Class1	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher
Class2	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher
Class3	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher	Teacher

TABLE I: Encoding of a timetable individual

- The $type$ can be: 0 for single-hour meeting, 1 for double-hours meeting and 2 for test-meeting, that is double-hours meeting with the constrain of being placed in the first two periods of a day (used for tests).
- The $possibilities$ set stores all the possible slots in which that meeting can be placed and the $saturation$ level is simply the count of all the $possibilities$.
- The $fixed$ flag states if the meeting should be fixed in all the timetables generated.

An example of split in meetings is shown in table II: in this case the tuple $[ProfA, Class1, Italian]$ is split in 5 weekly meetings (1 test, 1 double, 3 singles) for a total of 7 weekly hours. As said previously, this split is done in a predefined way: this means that every teacher that teaches $Italian$ in a class, will have the same split in meetings. The way in which a subject is split follows the rule that a teacher should meet a class as many times as possible.

Teacher	Class	Type	Possibilities	Saturation	Fixed
ProfA	Class1	2	[11, 12,...,56]	30	0
ProfA	Class1	1	[11, 12,...,56]	30	0
ProfA	Class1	0	[11, 12,...,56]	30	0
ProfA	Class1	0	[11, 12,...,56]	30	0
ProfA	Class1	0	[11, 12,...,56]	30	0

TABLE II: Example of split in meetings

Once all the meetings are computed, the fixed meetings are given as user input and taken into account to modify the set of possibilities, so that the only possibility left for that meeting is the one shown in the $fixed\_data$. For example, if the test meeting for $ProfA$ in $Class1$ must be placed in the first two hours of wednesday (i.e. periods 31 and 32), then the meeting in $place\_data$ becomes as shown in table III.

Teacher	Class	Type	Possibilities	Saturation	Fixed
ProfA	Class1	2	[31,32]	2	1

TABLE III: Example of fixed

The resulting data is then used as input for the creation of the initial individuals, using the **Sequential Construction Method (SCM)**. This method first sorts the $place\_data$ by priority, considering first the fixed meetings, then the test-meeting and last the meetings with lowest $saturation$. Second, it tries to allocate the teacher in the timetable considering the class and the periods in the $possibilities$ set: if a good period is found, then it is removed from all the other teachers in that class and from all the classes with that teacher; moreover, all the other periods in the day of the period chosen are removed. It may be the case that a good period cannot be found because all the possibilities are removed by other meetings: in this situation, a random free period is chosen and a constraint violation occurs.

C. Fitness Evaluation

The fitness is computed considering a set of hard and soft constraints. The hard constraints need to be solved in order to consider the timetable as feasible; on the other hand, the soft constraints are the one that should be solved in order to improve the quality of the timetable.
The hard constraints considered in this work are:

- HC1: No class teacher clashes.
- HC2: Lessons must be evenly distributed in the week.
- HC3: Teacher must not teach all six hours in a day.
- HC4: Teacher preferences in $fixed\_data$ should be met.

The soft constraint considered in this work is:

- SC1: Teacher timetables should be compact.

The fitness is the weighted sum of these hard and soft constraints and a lower fitness value represent a fitter individual.

$$Fitness(i) = \sum_{k=1}^{4} c_k HC_k + c_s SC_1$$

A better description on how hard and soft constraints are computed is reported below:

1) HC1: Using the matrix representation, the teacher clashes can be easily computed by counting the number of duplicates in each column.

2) HC2: The lessons distribution throughout the week is the one considered in the split in meetings to get the $place\_data$: because of random placements in the SCM or mutation operations (described later), the predefined split may not be found in the timetable. The correct split is described in patterns that are matched with the one in the timetable: the violation of the HC2 is met if the match is not found.

3) HC3: By law, a teacher should teach at most 5 periods in a day and if this does not occur, the violation is met.

4) HC4: This constraint allow to preserve the meetings required in the $fixed\_data$ that may be moved because of mutation operations.

5) SC1: Timetable compactness is computed counting the number of holes in each teacher timetable. Holes can be present in each teacher timetable but a large number characterize a poor quality timetable: a requirement for quality is that the max number of holes is 5. SC1 value is computed as the sum of the max and the mean number of holes.

D. Genetic Operators

*1) **Selection**:* Individuals are selected as parents based on their fitness function using modified version of the tournament selection. In the standard tournament selection, a group of individuals are randomly chosen from the population and the fittest is selected as parent. Because of the complexity of the timetable structure, the population size is set at tens individuals and the standard tournament selection leads to a lost of diversity. For this reason, a less elitist variation of the standard tournament selection is used.
In this variant, the selected parent is not always the best individual but weaker individuals in the tournament also have a chance to be selected. A random individual is selected from the population as champion, and it is placed in a match against other contenders: the winner is either the fitter individual or it is randomly chosen from the two individuals regardless of their fitness. The pseudocode is shown in algorithm 1.

Algorithm 1 Variation Tournament Selection pseudocode

```
parents = []
for population size do
  champion = random individual in population
  for tournament size do
    contender = random individual in population
    choice = random in [1,2,3]
    if choice = 1 then
      champion remains the same
    else if choice = 2 then
      champion becomes contender
    else if choice = 3 and f(contender) < f(champion) then
      champion becomes contender
    end if
  end for
  add champion in parents
end for
return parents
```

*2) **Crossover**:* In the genetic algorithm, the crossover operator is not implemented, because it would require repair operators to remove duplicate tuples and add missing tuples. The operator is therefore not used in order to avoid the extra processing of the repair operators.

*3) **Mutation**:* The most suited mutation mechanism is the swapping of tuples between periods: this allow to preserve the teacher-class pair association done in SCM, while changing the periods to find better results.

Hill climb technique is used to explore the neighbours of an individual [3]. The algorithm performs many tries (called HCSteps) starting from the parent individual; for each of these try, a number of swaps is performed creating an offspring: if the constraint cost is reduced, then the swap is deemed successful. An offspring is rejected if its fitness is worse than the fitness of the parent.
The pseudocode of Hill Climb is shown in algorithm 2.

In this work a multiphase approach is used to address timetable quality only if the hard constraints are satisfied. Different types of swaps are used depending on the phase.

During the first phase, the mutation takes into account the timeslot in which a violation of an hard constraint occurs. To find these timeslots, a $cost$ timetable is considered: this has the same structure of the individual timetable but it is originally filled with 0. During the fitness computation, when an hard constraint violation is found, the intersection class−period is updated depending on the cost of the constraint. For example if $ProfA$ is placed in period 11 for both classes $Class1$ and $Class2$, then it violates $HC1$ and the $cost$ timetable is updated to value c_1 for the slots $Class1 - 11$ and $Class2 - 11$.
Once the $cost$ timetable is filled, the timeslot with higher cost is selected; the second timeslot used for the swap is randomly chosen from the same class. This method allows to prioritize the swap of violations, possibly removing them.
Another important point that should be considered is the presence of fixed timeslots that should never be taken into account as timeslots for swaps. For this reason, a $fixed$ timetable is considered: this has the same structure of the individual timetable but it is originally filled with 0. During the SCM, when a fixed meeting is found, the corresponding timeslot of the $fixed$ timetable is updated to 1. The randomly chosen timeslot used for swap is taken among those in the class that are not fixed (value 0 in $fixed$ timetable).

During the second phase, all the hard constraints are solved ($cost$ timetable have only 0s) and only the soft one remains. To improve the quality of the timetable, random swaps are performed in a given class row; also in this case, the swap do not consider fixed timeslots and the swap is preserved if it improves the fitness.

Algorithm 2 Hill Climb pseudocode

```
Require: parent individual
Ensure: offspring individual
  HCSteps = 0
  offspring = parent
  while f(parent) ≤ f(offspring) and HCSteps < 10 do
    offspring = parent
    for # swaps do
      prev_offspring = offspring
      if phase1 then
        offspring = cost_random_swap(offspring)
      else if phase2 then
        offspring = random_swap(offspring)
      end if
      if f(prev_offspring) < f(offspring) then
        offspring = prev_offspring
      end if
    end for
    HCSteps += 1
  end while
  return offspring
```

*4) **Replacement**:* Survivor selection is implemented using an elitism strategy to decide which parent should survive after mutation: the best individual from the previous generation is maintained to prevent loss of best individuals.

III. Experimental Results

Different tests are done to evaluate the performance of the genetic algorithm. The data used to test the algorithm is taken from a real case scenario of middle school and consists of 9 classes and 14 teachers.

To improve the time required to create the initial population with the SCM, multiprocessing is used with 5 threads and the benefits of this are significant. Given that the algorithm do not use crossover, also the new population generation with Hill Climb is performed in multiprocessing with 5 threads.

The parameters used for the tests are:

- pop size: variable in [10,20,40].
- tournament size: 20% of the pop size.
- swaps in hill climb: 10.

The costs used are 100 for all the hard constraints, while SC1 is considered as it is.

A first test is done modifying the pop size and using the variation of the tournament selection as selection method. The results over 3 simulations are shown in table IV. The fitness trend of a simulation with pop size = 20 is shown in figure 1. As we can see, incrementing the pop size require less generations to get the best individual but more time overall: this is because the pop size remains constant during the evolution and the time needed to mutate all the individuals is higher; on the other hand, having more individuals in the population increments the probability to have mutations that improve the solution, thus requires less generations to reach the best individual.
The fitness value is better as the generations increases: this is because more swaps are done, thus the average number of holes decreases (but not the max).

Pop size	avg Num Gen	avg Fitness	avg Time (min)
10	125	7.86	55
20	74	8.25	66
40	32	8.56	78

TABLE IV: Test with different pop size and VTS

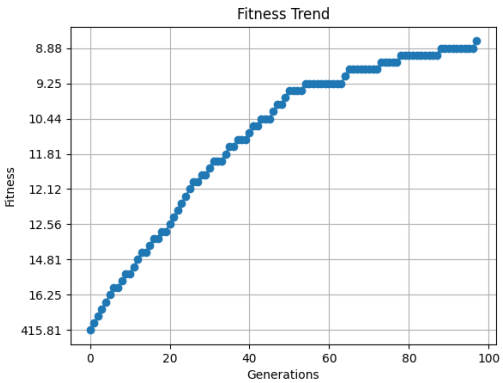

Fig. 1: Fitness trend pop size 20 VTS

An analysis is done to see the effect of the Variation Tournament Selection in the population diversity. Figure 2 shows the equality trend in the population along generations, that is the number of individuals that are the equals in the population. As we can see, the trend increases with the fitness. When there are low fitness improvements for many generations, the diversity does not stagnates but the Variation Tournament Selection allows to have also different individuals.

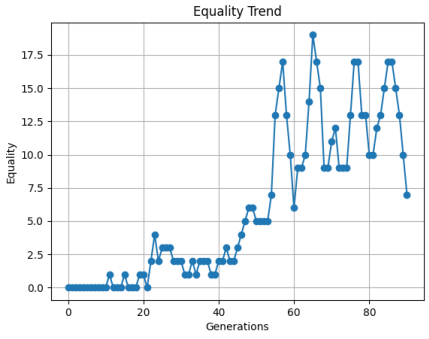

Fig. 2: Diversity trend pop size 20 VTS

In figure 3 we can see the result using the standard Tournament Selection: after few generations all the individuals are the same, resulting in a stagnation in the fitness values.

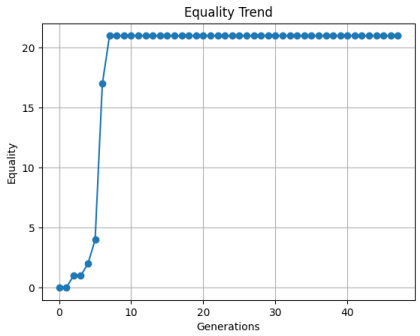

Fig. 3: Diversity trend pop size 20 TS

IV. Conclusions

In this work a genetic algorithm is designed to solve the School Timetabling Problem. A direct encoding suited for the problem is used and a heuristic SCM is applied to get good initial population. Particular genetic operators are used to select parents and mutate them in new offspring.
Some tests are performed to evaluate the algorithm changing pop size and selection method, with notable results in terms of time and fitness.

References

[1] Rushil Raghavjee. *A study of genetic algorithms for solving the school timetabling problem.* PhD thesis, 2013.

[2] M. Karova. Solving timetabling problems using genetic algorithms. In *27th International Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004.*, volume 1, pages 96–98 vol.1, 2004.

[3] C Stefano and Andrea Tettamanzi. An evolutionary algorithm for solving the school time-tabling problem. pages 452–462, 01 2001.