New approaches for the school timetabling problem *

Ana Cerdeira-Pena[1] Luisa Carpente[2] Antonio Fariña[1] Diego Seco[1]
[1]Database Laboratory [2]Department of Mathematics
University of A Coruña
Campus de Elviña s/n. A Coruña, 15071. Spain
{acerdeira, luisacar, fari, dseco}@udc.es

Abstract

School timetabling is a hard task that educational centers have to perform regularly and which implies a large waste of time and human efforts. For such reason designing techniques for the automatic generation of timetables is still of interest. Even though many contributions exist, the characteristics of the problem vary depending on the school policies, the country (laws), and other particular variables.

The complexity of this problem makes it difficult to find an optimal solution, so approximated techniques are traditionally used in practice. In this paper, we focus in the Spanish school timetabling problem and present several approaches to deal with it. The first technique proposed is based on the random non ascendent method (RNA). Then we provide several genetic algorithms which differ on the policies used for selecting how the next generation is created (including elitism) as well as on the levels of mutation considered. Finally, we study how to combine the two previous approaches. We run experiments both on synthetic and real scenarios in order to compare all the proposals. Even though the RNA and some of the pure genetic algorithms obtain good results in practice, we show that by joining RNA with genetic algorithms we gain stability in the results.

1. Introduction

Timetabling problems arising from educational institutions are related to the task of distributing teachers, periods of time, lessons and available resources (classrooms, labs,...) in such a way that some particular requirements are satisfied. Traditionally, depending on the institution these problems can be classified in three main groups: university, school, and exam timetabling problems.

*Funded in part by MEC (SEJ2005-07637-C02-02) and Xunta de Galicia (PGIDIT06PXIC207038PN) for the second author, and by MEC (TIN2006-15071-C03-03) and AECI (A/8065/07) for the first group.

In this paper, we focus in the school timetabling problem. Particularly, we are interested in all those cases where it is necessary to match fixed periods of time with lessons that are given by certain teachers. Under this context, different constraints have to be considered. Some of them, referred as *hard constraints* have to be satisfied for a solution to be considered valid (for example, a teacher cannot teach two lessons at the same time). Others, referred as *soft constraints*, reflect the preferences given by the teachers or by the policies of the school (for example, teachers usually want to minimize the holes in their schedule, and a school policy could aim at avoiding practising sports after lunch). Therefore, the accomplishment of hard constraints indicates the feasibility of a solution, whereas the soft constraints permit to quantify its goodness.

The timetabling problem we consider (including all the constraints) admits a mathematical programming representation [7], so an exact solution can be obtained by applying well-known techniques in this field [9]. However, it is known to be a NP-complete problem [4, 6, 10]. In practice, the high dimensionality of the problem makes it impossible to find an exact solution, and approximated methods are needed to tackle it. In most cases, they lead to *good quality* solutions in a reasonable amount of time, even at the expense of not guaranteeing that the *best solution* is reached.

We can find in the timetabling literature many works employing these kind of techniques. In [1] they give a survey oriented to the university timetabling problem. A graph coloring-based algorithm is shown in [3] also for the same scope. For the school timetabling field, we found solutions that are based on the use of genetic algorithms [2], but unfortunately they meet neither the requirements nor the structure of our problem. An interesting paper from Schaerf [7] discusses how to adapt both local search and tabu algorithms to this context.

In this work, we study a local search technique known as *Random Non Ascendent Method* (RNA) as well as two variants of genetic algorithms adapted to deal with our problem. The later differ basically in how new populations are ob-

tained, depending on whether parents having descendence can be chosen again or not. Moreover, we have obtained different variations of both genetic algorithms by applying distinct criteria for mutation and elitist policies. We complete our work with experimental results obtained for different synthetic scenarios. We first compare these techniques separately, and then we also study the effects of combining RNA with the others. Finally, we also test a reduced set of our best algorithms in a real case scenario.

In small scenarios, the three original algorithms manage hard constraints quite effectively, but RNA is usually the best choice for dealing with the soft constraints in most cases, with the unique exception of a variation of a genetic algorithm that uses an elitist policy. In larger scenarios, RNA does not behave so well, and some variants of the genetic algorithms outperform it.

The paper is organized as follows. In Section 2 we define the framework of the school timetabling problem we deal with. Then in Section 3 we describe the techniques developed. Section 4 shows the experimental results obtained for the synthetic scenarios. Section 5 analyze the behavior of some of the best alternatives in a real case. Finally, the conclusions of our work are discussed in Section 6.

2. The school timetabling problem

2.1. Overview

Considering the existence of groups of students, teachers who teach particular subjects to those groups, and a fixed scheduling of periods, the school timetabling problem consists in finding in which period of time a given teacher has to teach a certain subject to a specified group. Additionally, two different types of constraints have to be considered during this process: *hard* and *soft constraints*. The first ones refer to constraints that must be satisfied to obtain a feasible solution of the problem, whereas the second ones permit to measure the goodness of a solution. That is, a good valid timetable must satisfy the *hard constraints*, and it should also fulfill as many *soft constraints* as possible. Common examples of hard constraints are that a teacher can only teach a lesson to one group at the same time, and that a given group cannot receive more than one lesson at the same time (possibly by different teachers). Soft constraints usually depend on school policies or on teacher's preferences (i.e. minimizing the amount of periods with lessons, minimizing the *holes* in the timetable, avoiding some subjects at a specified time, etc).

2.2. Constraints involved

As an optimization problem, the school timetabling problem can be defined by a *mathematical formulation* that describes the feasible regions through a solution space; and an *objective function*, which permits to lead a search process towards an optimal solution. On the one hand, from a theoretical point of view, the constraints that define such space and must be satisfied by a solution are actually the *hard constraints*. On the other hand, each *soft constraint* has usually an assigned weight and contributes to the value of the objective function. Therefore, if such constraint does not hold, the value of the objective function will increase, and consequently, the quality of the solution worsens.

In practice, we are interested in managing both kind of constraints with a function called *cost function* that leads the search process for a solution of the problem. The algorithms we present in this paper use such cost function. Therefore, they are search algorithms that move not only through the solution space, but through a wider search space that includes both feasible and non feasible solutions. As a consequence, during the search of a good solution, they have to be able to distinguish between *feasible* and *non feasible* solutions, as well as considering the goodness of a given solution. It seems that it could worth to design search algorithms that could move only through the feasible space towards a good solution. However, in most problems, it would be very difficult to find an initial feasible solution to start with. This is the reason why we added the *distance to feasibility* to the objective function that permits us to determine the optimality of an obtained solution. Finally, it is important to point out that in order to lead the search process towards feasible regions, the hard constraints are given a higher weight than that of the soft constraints, but both of them are included in the evaluation of the *cost function*.

Before describing the constraints considered by our algorithms, let us define the term *period*, shown as the unit of time in which a lesson is taught. So the weekly schedule is divided into periods, excepting breaks and free hours that are not considered periods. We define a *class* (referred to different concepts in the literature) as the association between a subject (such as Maths, Foreign language, etc.) and one or more *groups* of students that are previously defined by the educational institution. Groups are named depending on the level (1^{st}, 2^{nd},... course) and on how all the students in the same educational level are partitioned (A, B, C,...). For example, as the Mandatory Secondary studies are known as *E.S.O.* in Spain, we will refer as *2A E.S.O.* to the first partition of students (A) in the second level of E.S.O. Therefore, we will have classes such as:

- 1A E.S.O.-Maths: Maths in the group 1A E.S.O.
- 1A/B/C E.S.O.-Computer Science: an association of

students coming from different groups (in this case, from 1A E.S.O., 1B E.S.O. and 1C E.S.O.) who are taught Computer Science subject together.

2.2.1 Hard constraints & Soft constraints

We describe in detail the different constraints we take into account in our algorithms.

Hard constraints. Four constraints are used to measure the feasibility of a solution:

- *Overlaps*: They avoid the possibility of a class been taught by more than one teacher in the same period. It also avoids that classes sharing resources (ie. a classroom, lab, etc.), as well as, classes in which the same group is involved, could be assigned to the same period. The case of a teacher giving more than one lesson at the same time is not considered as our representation of the problem avoids this issue (see Section 3).
- *Simultaneity*: Two classes are defined as simultaneous classes if they are taught by different teachers at the same time. Sometimes a group must be divided into two or more subgroups that are taught different subjects (for example, the case of elective subjects) by different teachers in different classrooms simultaneously. We also consider classes that must be divided into two (or more) subclasses. That is, where the group associated to that class is divided into two or more subgroups that receive the same lesson possibly in a different place and with a different teacher. A good example would be a group of 50 students doing an individual exercise in two different rooms with 25 computers each (and two different teachers). In this case, the class might be divided into as many subclasses (simultaneous or not) as subgroups are needed. In the example above, if only one computers room were available, both subclasses would not occur simultaneously.
- *Unavailability*: It considers periods when a class can not be given or when a teacher cannot teach.
- *Consecutiveness*: This constraint checks whether a distribution of hours for a pair teacher-class is followed. For example, some practical lessons should be taught in two consecutive periods of time.

As the search space of our algorithms avoids some typical non-feasible scenarios, some constraints are not considered *hard*. For example, a class not being given all their lectures, or a lecture not taught at an unchangeable period.

Soft constraints. According to the preferences shown in Spanish schools, we show some interesting constraints that a schedule should try to fulfill:

- *Overuse*: It refers to the number of periods per day in which a teacher gives its lessons, over its specified maximum of periods per day.
- *Underuse*: When teachers have preferences on their minimum number of periods per day, it indicates the number of periods under such minimum.
- *Holes*: We consider the number of *empty* periods between two consecutive periods where a teacher is assigned a class. Breaks and free-time periods are not considered holes.
- *Splits*: They consider the number of periods between two non consecutive assignments to a same class in the same day.
- *Groups*: Assuming a specified maximum of periods per day for an association teacher-class, it considers the number of exceeding periods in such day. This constraint is only considered if a maximum number of consecutive periods (related with *consecutiveness*) for the pair class-teacher is not specified.
- *Undesired*: Assuming that there are periods in which a teacher would prefer not to teach, this constraint indicates the number of such periods where that teacher is assigned a class. This constraint is the *soft* version of the mandatory *unavailability* constraint (referring to teachers).

2.3. Cost function

The constraints described in the previous section are included into a function that associates a *cost* value to a given solution. Such value can be used to compare the goodness of different solutions. This function is defined as follows:

Definition 1 Let S be the search space; $s \in S$, a solution; N, the number of constraints considered and $f_i(s)$ the value of a specific function which scores the number of conflicts of the i^{th} constraint for the solution s. Then the cost function $F_c(s)$ is defined as: $F_c(s) = \sum_{i=1}^{N} f_i(s)$.

We can see that the lower the value $F_c(s)$ for a given solution s, the more quality of s. Therefore, the school timetabling problem can be shown as the search for the solution $s_i \in S$ such that $F_c(s_i) = \min_{s \in S} F_c(s)$.

Notice that we measure both the distance to feasibility, and the goodness of the solution. Therefore, by giving more weight to the hard constraints than to the soft ones, our algorithms lead the search process towards valid solutions. Additionally, by varying the weights of the soft constraints a given school can adjust the importance of the involved preferences.

3. Algorithms

In this section we first describe the representation of the problem used in our algorithms. Then we describe three different techniques to solve our timetabling problem (the first one is a RNA-based and others are based on genetic algorithms). Finally, we combine them obtaining two new hybrid algorithms. All of them are heuristic-based search methods which try to reach a good quality solution at a reasonable computational cost, even though it is not guaranteed its optimality or feasibility. This approach is commonly used in optimization problems which belong to the *NP-complete* class, as well as, in those ones that can be solved with an exact method, but whose solution is not computationally feasible.

3.1. Previous concepts

3.1.1 School timetabling problem representation

Many different representations have been chosen when tackling this problem, among them [5, 8]. In our case, we used a *teacher-oriented* representation. Hence, timetable is represented as a two-dimensional matrix containing in each cell (i, j), the class given by the i^{th} teacher at the j^{th} period. As shown before, this representation avoids implicitly the case of a teacher giving more than one class at a same time.

3.1.2 Underlying movements

Although our algorithms represent different approaches, all of them need to traverse the search space to find good solutions. Traversing such space is done through *moves*. We defined two kind of moves: *i) simple-moves*, that are obtained by swapping two distinct values in a given row of the timetable. Obviously, moves including either the swap of identical classes or the swap of empty cells are skipped; and *ii) double-moves* that are the combination of two simple-moves when the first move leads to an unfeasible scenario. Otherwise, it consists just in the first simple-move.

Notice that the search space remains connected and any solution s_i can be reached from any other s_j in a finite number of moves.

3.1.3 Initial solution

Our search algorithms need to start, at least, with an initial solution. Genetic algorithms need more than just one solution as they have to begin with a initial population of individuals. From that starting point, they navigate through the search space by means of moves. An initial solution can be provided a priori, for example, by starting with a previous timetable. However, in most cases, a random initial solution is generated. Actually a greedy procedure can be used to minimize the conflicts generated by some constraints such as *overlaps* and *unavailabilities*.

3.2. RNA Search

Following the RNA local search technique, we have developed an algorithm which, starting from an initial solution, iteratively moves from a solution to another doing *double-moves*. These are repeated until they make no improvements during a given number of iterations. It keeps track of the current best solution (s_b) at each stage, and by applying those moves generates a new solution s_i, trying to improve the value of $F_c(s_b)$. When $F_c(s_i) \leq F_c(s_b)$, s_i becomes the new best solution.

3.3. Genetic algorithms

Genetic algorithms are included into a branch of the *evolutionary computation* (EC) field called *evolutionary algorithms* (EAs), and base their operation on the evolution mechanism, in particular, on natural selection and inheritance features. Those algorithms do a simultaneous search in different regions of the search space. Starting with a *population*, a set of *individuals* or potential solutions, best candidates are selected based on their *fitness* value which measures the quality of each individual. These will be the *parents* of a new group of individuals obtained by modifying (recombining and mutating) the previous ones, which will compose a new population to be used in the next iteration of the algorithm. In all this process, both genetic operators and selection technique play an important role. The first ones determine how the genetic information is combined or modified. Among them, the most relevant are *crossover* and *mutation*, which allow passing information from parents to offspring, and jumping from a point to another of the search space, guaranteeing the genetic diversity, respectively. The second ones, decide what individuals will become parents of a new population. Many alternatives are known, but we will explain below those involved in our algorithms.

We consider a solution to our school timetable problem, that is, a timetable, as an *individual*, and obviously a set of them, as a *population*. The *fitness* value will be given by our *cost function* evaluation and we will be doing a *mutation* by means of a *simple-move*. We define the *one point crossover* that randomly selects a crossover point within the timetable, in such a way that all rows from the first one to that point are inherited from one parent and the rest ones, from the other. We have developed two main different approaches:

Tournament (GAT) Starting with a population randomly generated, two pairs of individuals are selected and then the best candidate of each one is chosen, according to its *fitness* value. In this way, the parents of the next generation

are obtained. New children result from applying the defined crossover operator and they will mutate at a given probability before being inserted into the population. The process is then repeated, updating the best solution found among the individuals up to now, and it stops when a certain number of cycles is completed. As can be noticed, some individuals can be selected as parents more than once, something that will not happen in the following approach.

Four Children Tournament (GAT4C) The main difference with the previous algorithm is that once two individuals are chosen to form a pair they are discarded, so they will never be chosen again. Hence, the size of a population must be multiple of *four* and also *four* children must be created in each iteration. The rest of the operation follows the same guidelines than the previous algorithm.

3.4. Hybrid algorithms

We can combine the previous techniques to study the performance of two different basic approaches: RNA and genetic algorithms. We check the usefulness of mutation as the way to avoid *local minima*. Thus, we have developed two new algorithms:

Tournament & RNA (GAT & RNA) In this approach, *Tournament* is alternated with *RNA* until a certain number of iterations is reached. First, a *Tournament* phase is performed. It is followed by a *RNA* phase which starts taking the best individual found among all the generations as the initial solution. After finishing the RNA stage, a new population of n individuals must be created as the input to a new genetic phase. To do this, during the RNA phase a list with the n best timetables is kept. Therefore, we guarantee that the best candidates known up to now are always included at the beginning of every *Tournament* phase, and consequently the best *genetic material*.

Four Children Tournament & RNA (GAT4C & RNA) This technique works exactly as the previous one. The only difference is the use of a *Four Children Tournament* phase instead of the *Tournament* one.

Apart from these techniques we have proposed some variants for our genetic-based algorithms. We include these strategies in our experiments. A brief description of such strategies is presented below:

v1) To increase dynamically the number of mutations after a given number of iterations without improvements.

v2) To apply a more elitist selection technique of the best candidates, so reducing its proportion to a given percentage.

v3) Not to eliminate the loser candidates when we work with a *Four Children Tournament* phase, in such a way, all the individuals could be parents at least once.

4. Experimental results on synthetic data

In this section we summarize our experimental results over two sets of synthetic test cases with different size and configurations. Each collection is composed of a given number of test files. Each file was created depending on: *i)* the number of unavailable and undesirable periods for each teacher; *ii)* the number of unavailable periods for each class; and *iii)* the distribution of classes along the whole week (at most 2 periods per day). We run experiments[1] with our algorithms using in all cases the same configuration of their parameters. Those parameters are: *i)* the weight associated to each *hard constraints* was set to 1000. *ii)* The weight of the *soft constraints* varies: the weight of *overuse*, *splits*, and *groups* is set to 6; that of *underuse* is 4; for *undesired* periods we used the value 3; and finally the weight of *holes* is 1. *iii)*For the genetic algorithms we fixed the population size to 32, and the probability of mutation to 0.4. And *iv)* in the hybrid algorithms, we fixed the maximum number of iteration of the RNA phase to 150.

Small test collection The first test collection is composed of 10 files. Each of them has 6 groups, 70 classes and a number of $15 - 16$ teachers who teach during 15 periods weekly. Table 1 shows the average value of unsatisfied constraints and the standard deviation from those values for 10 runs of each algorithm over the test data set. Each runs was limited in time to 30 minutes. Results show that most algorithms obtain good results and perform quite similarly, with the exception of a small group that perform very badly: variants $v1$ and $v1+v2$ of the genetic algorithms and variant $v3$. The best choices seem to be RNA and GAT4C.

Large scale scenario After discarding the algorithms that behave badly in the small scenario, we designed a huge file containing 27 groups, 333 classes and 71 teachers who teach during 15 periods weekly. Again, we run 4 times the different algorithms over the test file, limiting time to 5 hours each. In Table 2, results show that variant *v2* obtains very good results, especially version GAT & RNA, that reduces the variability. RNA and variant *v1+v2+v3* behave also well, but they lead to a high number of unsatisfied *soft constraints*.

5. A case study

In this section we test a selection of our best algorithms with real data obtained from a Secondary school (I.E.S.

[1] We used an Intel Core2Duo E6420@2.13Ghz with 2Gbytes of DDR2-800 memory. The computer ran Windows Xp OS. Prototypes were implemented in Java 1.6.

Table 1. Results for the small synthetic scenario.

Algorithm	constraints			
	Avg(F_c)		Std(F_c)	
	hard	**soft**	**hard**	**soft**
original				
RNA	0.16	0.60	0.09	0.39
GAT	0.22	1.45	0.22	0.32
GAT4C	0.31	2.05	0.29	0.68
GAT & RNA	0.44	2.30	0.50	1.35
GAT4C & RNA	0.38	2.46	0.33	0.93
var: v1				
GAT	6.94	23.57	3.81	3.88
GAT4C	7.07	23.32	3.99	3.84
GAT & RNA	0.34	2.74	0.29	0.96
GAT4C & RNA	0.28	2.74	0.25	1.20
var: v1+v2				
GAT	3.99	15.57	2.86	2.85
GAT4C	5.53	18.90	3.50	2.23
GAT & RNA	0.47	1.90	0.48	0.96
GAT4C & RNA	0.41	1.76	0.43	0.73
var: v2				
GAT	0.26	0.74	0.14	0.45
GAT4C	0.23	0.74	0.16	0.44
GAT & RNA	0.44	1.74	0.46	0.92
GAT4C & RNA	0.33	1.41	0.34	0.84
var: v1+v2+v3				
GAT4C & RNA	0.49	2.06	0.42	0.70
var: v1+v3				
GAT4C & RNA	0.43	4.33	0.42	1.55
var: v2+v3				
GAT4C	0.28	0.75	0.22	0.43
var: v3				
GAT4C	27.62	42.19	7.52	3.97

Menendez Pidal) in A Coruña-Spain. This school has 3 sections with different diplomas. Our data comes from one of them including: commerce, marketing, and international commerce studies. That scenario includes 11 teachers who can select 3 undesirable and 3 unavailable periods respectively. Teachers can be assigned classes from 1 to 5 periods per day. There are 33 classes, and 3 simultaneity relationships between classes.

In the top table in Figure 1 we can see that even though all the chosen algorithms reach good results, we can highlight those of RNA and GAT4C (which is the best choice). The bottom plot in that figure shows the evolution of the cost function in GAT4C with respect to execution time (1 hour). It is remarkable that during the first minutes the curve of the cost function is very steep, and that after 20 minutes

Table 2. Results for the large synthetic scenario.

Algorithm	constraints			
	Avg(F_c)		Std(F_c)	
	hard	**soft**	**hard**	**soft**
original				
RNA	0.88	63.58	0.82	3.77
GAT	2.88	67.21	2.11	8.42
GAT4C	3.88	71.04	2.26	5.85
GAT & RNA	1.50	62.54	1.54	6.50
GAT4C & RNA	1.75	70.33	1.35	3.95
var: v1				
GAT & RNA	1.75	63.75	1.19	4.71
GAT4C & RNA	2.25	69.04	2.12	4.49
var: v2				
GAT	1.94	26.04	1.39	4.17
GAT4C	2.69	34.08	1.53	6.53
GAT & RNA	0.94	30.38	0.98	2.77
GAT4C & RNA	1.94	35.71	1.44	4.12
var: v1+v2+v3				
GAT4C & RNA	1.00	52.67	0.73	5.71
var: v2+v3				
GAT4C	2.75	49.88	1.71	4.64

the quality of the solution obtained is very good. Finally, after 40 minutes the improvements are almost negligible.

6. Conclusions and future work

We have developed different solutions for the secondary school timetabling problem focusing on the Spanish context. They are based on the use of RNA and genetic-based algorithms. A wide study was done by introducing some variations of the genetic-algorithms, as well as by creating hybrid versions. To test the behavior of the different alternatives developed we first applied them to synthetic scenarios (what allowed us to discard some of them). Finally, we studied how the best techniques performed over a large scenario and over a real case.

Our results showed that the RNA method behaves well in most cases. It is usually able to reduce drastically the number of unsatisfied hard constraints, and consequently the cost function. However, it does not manage soft constraints so effectively. This issue is improved by some of the genetic-algorithms variants. More precisely, that called GAT4C performed very well in small scenario. It obtained similar results to those of RNA for the hard constraints and improved values for the others. In our large synthetic scenario the best choice was a hybrid solution (GAT&RNA) using an elitist policy.

Algorithm	constraints			
	Avg(F_c)		Std(F_c)	
	hard	**soft**	**hard**	**soft**
original				
RNA	0.88	5.95	0.74	1.72
GAT	1.05	6.90	1.01	2.92
GAT4C	0.55	6.37	0.72	1.83
GAT & RNA	1.63	8.10	1.07	2.71
var: v2				
GAT4C & RNA	2.20	8.42	1.95	3.69

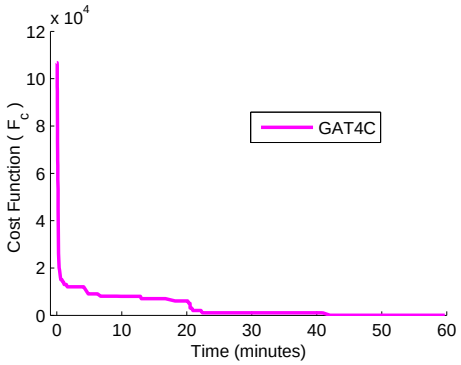

Figure 1. Results for a real case scenario.

As a future work, we are focusing in the generation of better initial solutions. The idea is to create initial timetables using a greedy procedure to avoid infeasibility. We also consider that more intelligent (although maybe heavier) moves can be used to skip bad quality timetables during the search process.

References

[1] M. W. Carter and G. Laporte. Recent developments in practical course timetabling. pages 3–19, 1998.

[2] A. Colorni, M. Dorigo, and V. Maniezzo. A genetic algorithm to solve the timetable problem. Technical Report 90-060, 1990.

[3] P. de Haan, R. Landman, G. Post, and H. Ruizenaar. A four-phase approach to a timetabling problem in secondary schools. In *Practice and Theory of Automated Timetabling (PATAT 2006)*, pages 423–425, 2006.

[4] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151–162, February 1985.

[5] S. Gyri, Z. Petres, and A. R. Vrkonyi-Kczy. Genetic algorithms in timetabling. a new approach.

[6] A. I. S. Even and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5:691–703, 1976.

[7] A. Schaerf. Local search techniques for large high-school timetabling problems. Technical Report 4, 1999.

[8] K. A. Smith, D. Abramson, and D. Duke. Hopfield neural networks for timetabling: formulations, methods, and comparative results. *Comput. Ind. Eng.*, 44(2):283–305, 2003.

[9] A. Tripathy. School timetabling–a case in large binary integer linear programming. *Management Science*, 30:1473–1489, 1984.

[10] R. Willemen. *School timetable construction: algorithms and complexity*. PhD thesis, Technische Universiteit Eindhoven, The Netherlands, 2002.