

A Framework for School Timetabling Problem

Kimmo Nurmi, Jari Kyngäs

Satakunta University of Applied Sciences, Finland, Tiedepuisto 3, FIN-28600 Pori

{kimmo.nurmi, jari.kyngas}@samk.fi

Abstract: This paper introduces a framework for a highly constrained school timetabling problem, which was modeled from the requirements of various Finnish school levels. We present a successful algorithm to solve real-world problems as well as artificial test problems. Moreover, we find the best configuration for this algorithm using brute force and statistical analyses. Finally, we propose a set of benchmark problems that we hope the researchers of the timetabling problems would adopt.

Keywords: Evolutionary Algorithms, Heuristic Search, Real World Scheduling, Timetabling.

1. Introduction

In recent years many solution approaches for different timetabling problems have been introduced. Most of the work has concentrated on examination timetabling, university timetabling and course timetabling [1,2,3,4,5,6]. Timetabling researchers have obtained very promising and practicable results in these problem areas. However, school timetabling has not been as extensively studied, and widely usable results are not yet available. The school timetabling problem [7] consists of assigning lectures to periods in such a way that no teacher, class or room is involved in more than one lecture at a time and other constraints (hard and soft) are satisfied.

The main focus of this paper is to introduce a successful algorithm to tackle highly constrained school timetabling problems [8,9]. The algorithm is controlled with nine different parameters. For seven of these parameters we searched the best values using both brute force and statistical analyses. Two different methods were mainly used because we wanted to confirm that the chosen parameter values were the best ones. Two parameters were excluded because they have a great impact on the running time and therefore make the results incomparable (see Chapters 2-4).

Another point of this paper is to propose a set of benchmark instances and publish them as well as our results on the web. We wanted to lay out the foundation of comparable results (see Chapter 5). This idea is presented (by Schaerf and Di Gaspero) in [10].

2. Problem Description

We describe a school timetabling problem that arises in various Finnish school levels: secondary school, high school and college. This problem description is representative of many timetabling scenarios within the area of school timetabling. The timetabling is based on the following conditions:

- The timetable frame consists of n weeks; each week has m days and each day t periods (timeslots).
- The lecture is a predefined combination of a teacher, a class (or course), a room and the length of the lecture in periods.
- The set of teachers, classes/courses and rooms is fixed.
- More than one teacher can teach a particular class at the same time.
- Each class should have at least a given number of periods in a day, but should not have more than another given number of periods.
- The classes/courses can have common students.
- Some classes/courses must precede other classes/courses
- The rooms can be classified to certain room types.

The *school timetabling problem* consists of scheduling the predefined lectures in such a way that the solution is feasible and mostly acceptable to both school staff and teachers. A feasible solution satisfies the following *hard constraints*:

1. No teacher, class/course or room is scheduled to the same period more than once (basic model).
2. No class/course is scheduled to the same period, if they have common students.
3. The given lectures are scheduled to the same period.
4. The given lectures are scheduled for predetermined periods.
5. For each class the daily minimum and maximum number of periods is respected.
6. A class/course c_1 is scheduled to earlier in the week than class/course c_2 , if c_1 must precede c_2 .
7. A lecture cannot be scheduled to periods where the teacher, the class or the room is unavailable.

A solution is most acceptable if it satisfies the following *soft constraints*:

1. The timetable of each class should have as few idle (leap) periods as possible.
2. The school day of some classes should not start in the first period.
3. The school day of some classes should end as early as possible.
4. The timetable of some teachers should have as few idle periods as possible.
5. For some teachers the preferred daily minimum and maximum number of periods is given.
6. Some teachers prefer not to be scheduled in certain time periods.
7. Some teachers should be scheduled only on a limited number of days.
8. The lessons of a subject should be on different days.

The above formulation covers school timetabling problems occurring in Finnish secondary schools, high schools and colleges. In lower school levels the problem reminds the basic school timetabling problem and in higher levels it is more similar to a combination of the school timetabling problem and the course timetabling problem [11].

3. The Algorithm

The algorithm presented here is an extension of the *h-HCGA* algorithm presented in [12]. We have spent “quiet timetabling research life” since the development of the algorithm and returned to the problem only recently, when we were asked to schedule the timetables of one secondary school and one high school. First we spent quite a lot of time trying to improve our original algorithm with the ideas of Ant Algorithms [13, 14]. Unfortunately we were not able to find competing results. We moved into improving the parameter values of the *h-HCGA* algorithm. Table 1 summarizes the parameters and the values from which we were searching the best configuration. Because two of the parameters have a great effect on computational time, we fixed their values in order to keep the results comparable.

| Extended h-HCGA parameters | |
|--|--|
| F. Population size | 20 (<i>fixed</i>) |
| 1. Reproduction/selection | Random-average, Random-best, Marriage-best |
| F. Maximum move sequence | 10 (<i>fixed</i>) |
| 2. Tabulist size for the GHCM-operator | 0, 5, 10 |
| 3. Tabulist size for overall moves | 0x, 1.5x, 2x, 4x (maximum move sequence) |
| 4. Type of removal check | All, Hard, Hard or Soft |
| 5. Fitness calculation | Absolute, Weighted |
| 6. Update frequency of hard constraint weights | 5, 10, 20 |
| 7. Prevention of same lessons | No, Yes |

Table 1: Parameter summary of the extended h-HCGA algorithm.

The algorithm is a genetic algorithm [15,16] with one mutation operator and no recombination operators. The two most important features of the algorithm are *the greedy hill-climbing mutation (GHCM) operator*, which generates a new solution candidate from the current solution and *the*

adaptive genetic penalty method (ADAGEN), which is a multiobjective optimization method [17,18,19,20].

The GHCM operator is based on moving a lecture l_1 from its old period p_1 to a new period p_2 , moving another lecture l_2 from period p_2 to a new period p_3 and so on, ending up with a sequence of moves. The initial lecture selection is random. The new period for the lecture is selected considering all possible periods and selecting the one which causes the least increase in the cost function when considering the relocation cost only. Moreover, the new lecture from that period is again selected considering all the lectures in that period and picking the one for which the removal causes the most decrease in the cost function when considering the removal cost only. The operator stops if the last move causes an increase in the cost function value and if the value is larger than that of the previous non-improving move.

We noted in [12] that we can improve the GHCM operator by introducing a tabu list, which prevents reverse order moves in the same sequence of moves. We can also prevent only some of the moves (parameter 2). To further widen the use of the tabu list, we can prevent reverse order moves in consecutive move sequences (parameter 3), that is in consecutive applications of the GHCM operator.

The ADAGEN method is an adaptive penalty method for multiobjective optimization. A traditional penalty method assigns positive weights (penalties) to the soft constraints and sums the violation scores to the hard constraint values to get a single value to be optimized. The ADAGEN method assigns dynamic weights to the hard constraints. The weights are updated in every k^{th} (parameter 6) generation using a somewhat complicated formula [12].

The reproduction operation [21] of the (genetic) algorithm is based to a certain extent on the steady-state reproduction [22]. We select randomly a timetable from the population of timetables for single GHCM operation. The new timetable replaces the old one if it has better or equal fitness. We have three variations (parameter 1). In the first one the new timetable replaces also the least fit in the current population if it is better than the current population average. In the second one the least fit is replaced with the best one, when n better timetables have been found, where n is the size of the population. In the third one we use the second variation but select a timetable using a marriage selection [23].

When selecting a lecture to be removed from a period (parameter 4), we can consider either all the lectures in that period or only those that have at least one constraint violation with the lecture that was moved to that period previously. We can also consider only hard constraint violations. When calculating the best lecture to be removed and the best period to move to (parameter 5), we can use either absolute number of constraint violations or the weighted penalty value of the ADAGEN method.

We also tried (parameter 7) to programmatically prevent lessons of a subject to be on the same day (soft constraint 8). Another possibility would have been to change it to a hard constraint.

In preliminary experiments the extended h-HCGA algorithm found encouraging results. Some parameter values seemed to produce better results than others. Our next goal was to find the best configuration for the parameter values.

4. Finding the Configuration

We had two possible choices for the process of finding out the best configuration of parameters: brute force and statistical methods. The brute force approach was very attempting because we had the possibility to use up to 78 computers. We decided to use brute force but also analyze the runs using standard statistical methods. We also performed “what-if” analyses on the runs using the

Friedman test [24] in similar fashion to Birattari & al. [25]. We used the highly constrained benchmark instance C3 (see chapter 5) in our test runs.

For the runs to be comparable in computing time we had to fix two parameter values (see table 1). The rest of the parameter values were such that they did not influence the convergence time. The total number of configurations was 1296 but because the *tabulist size for overall moves* has no effect when *tabulist size for GHCM-operator* is zero, we were left with 972 configurations.

Our goal was to find a configuration which would minimize the hard constraints to zero. The configurations were ranked according to hard constraints. We decided to run every configuration five times and for two hours each, on an AMD Athlon 1700+ with 512Mb of memory. For every generation of the algorithm we saved the number of hard constraint violations.

At first we only considered those runs that had ended up with zero hard constraint violations — 218 out of 972 runs met this criterion.

4.1. Choosing the Configuration

We started analyzing the results by counting how many times each value of each parameter occurred in the best runs. Table 2 summarizes the results.

| Parameter | Values and occurrences |
|--|---|
| 1. Reproduction/selection | Random-average = 38% , Random-best = 40% , Marriage-best = 22% |
| 2. Tabulist size for GHCM-operator | 0 = 20%, 5 = 42% , 10 = 38% |
| 3. Tabulist size for overall moves | 0x = 18% , 1.5x = 25% , 2x = 26% , 4x = 31% |
| 4. Type of removal check | All = 29%, Hard = 63% , Hard or soft = 8% |
| 5. Fitness calculation | Absolute = 64% , Weighted = 36% |
| 6. Update frequency of hard constraint weights | 5 = 48% , 10 = 30%, 20 = 22% |
| 7. Prevention of same lessons | No = 77% , Yes = 23% |

Table 2: The chosen values (bolded) after brute force runs.

The best value for the last four parameters could easily be chosen (tested with the t-test). The first three, on the other hand, required further investigation.

Next we ran the chi-squared test [24] pairwise between all parameters. We were especially interested in those test cases which included the chosen parameter values. Unfortunately none of the chi-squared tests showed statistically significant dependencies among the parameter values. We enlarged the chi-squared test to include also those runs that had ended up with one hard constraint violation, but that did not change the test results. (These statistical tests can be obtained from us at request.)

At this phase we were still not able to find the values for the first three parameters. However, we were able to reduce the number of free parameters from 972 to 16 (the combination of chosen parameter values of each parameter). For the *reproduction/selection*, the *tabulist size for GHCM-operator* and the *tabulist size for overall moves* we were not able to choose one specific value. That strengthened our decision to use statistical methods.

4.2. Statistical Methods

Birattari & al. [25] have developed a process, F-Race, in which they start with a considerable large population and eliminate individuals from it with the help of statistical methods. The idea is to eliminate an individual from the population as soon as it can be statistically significantly proven that the individual is not performing good enough. The process can be thought of as a race where

every individual competes against each other. At specific intervals the performance of each individual is checked and the poorly performing ones are dropped out from the population. This process is well suited for problems with extensive number of parameters.

We tested this method to see how it would have worked in this case. However, we did not use the actual racing algorithm — we only tested the runs to see which ones would have dropped out after two hours of running. From the output of the runs we sampled ten evenly spaced observations. This meant that we had 972 treatments (configurations) and 10 blocks (samples) for the Friedman test.

Table 3 summarizes the results of the Friedman test for the case of the eliminated runs. When we compare this with table 2 we can see that these results are very compatible with each other. For example, table 2 tells us that in the 218 runs 63% of the configurations had parameter *type of removal check* set to value *Hard*. Table 3, on the other hand, tells us that only 7% of the eliminated runs included configurations where this parameter was set to value *Hard*. The information in the two tables supports each other very well.

| Parameter | Values and occurrences |
|--|---|
| 1. Reproduction/selection | Random-average = 21% , Random-best = 34%, Marriage-best = 45% |
| 2. Tabulist size for GHCM-operator | 0 = 45%, 5 = 24% , 10 = 31% |
| 3. Tabulist size for overall moves | 0x = 29%, 1.5x = 20%, 2x = 28%, 4x = 23% |
| 4. Type of removal check | All = 21%, Hard = 7%, Hard or soft = 72% |
| 5. Fitness calculation | Absolute = 14%, Weighted = 86% |
| 6. Update frequency of hard constraint weights | 5 = 16%, 10 = 28%, 20 = 56% |
| 7. Prevention of same lessons | No = 30%, Yes = 70% |

Table 3: The chosen values (bolded) in eliminated configurations.

In the brute force runs we were not able to choose the best values for the first three parameters. Here the best value for the *reproduction/selection* is *Random-average*. The difference between the values *Random-average* and *Random-best* is statistically significant at 0.001 level when measured with the t-test. Therefore we chose the value *Random-average* for parameter *reproduction/selection*.

The other two parameters show no statistical difference between the values. We chose value 5 for the parameter *tabulist size for GHCM-operator* because both tables indicate that this could be the best value (though not statistically significantly differing from value 10). The parameter *tabulist size for overall moves* shows no difference in goodness of the values. The value was chosen to be 2 because we *felt* it could be the best value.

5. Standard Benchmark Instances

In school timetabling we do not have a set of standard test problems, as is the case in examination timetabling [10]. We now introduce some test instances and hope that they will lay the foundation for the standard universal benchmark instances for school timetabling problems.

The first set of test problems consists of *all-N-problems* introduced in [12]. These artificial problems have a timeframe of N days with N periods in one day totaling N^2 periods. There are N teachers, N classes and N rooms. All possible combinations of a teacher, a class and a room are to be scheduled. This will give us N^3 lectures to be scheduled. The problem is quite difficult to solve since a feasible solution has no idle periods for any teacher, class or room.

The second set of test problems consists of *Abramson-N-problems* introduced in [26]. These artificial problems have a timeframe of 30 periods. The lectures are built by first choosing a random

teacher and class and room identifiers and then placing that lecture in the first possible period. If the lecture cannot be placed in an available period, then it is discarded and another one is generated. This process continues until all teacher, class and room identifiers have been used. The resulting timetable has again no idle periods and thus is tightly constrained.

Both *all-N-problems* and *Abramson-N-problems* are excellent test instances since

- a) every single school timetabling algorithm can tackle them because they are the simplest version of the problem (basic model) and
- b) we can easily increase the problem difficulty by increasing N .

| Problem identifier | B3 | S3 | H3 | C3 |
|---|-------------------|------------------|-------------|---------|
| School level | Artificial school | Secondary school | High school | College |
| #Weeks | 1 | 1 | 1 | 1 |
| #Days | 5 | 5 | 5 | 5 |
| #Periods per day | 4 | 7 | 7 | 8 |
| #Teachers | 21 | 25 | 18 | 46 |
| #Classes/courses | 11 | 14 | 50 | 41 |
| #Rooms/room classes | 3 | 25 | 13 | 34 |
| #Lectures | 169 | 280 | 219 | 387 |
| #Periods in lectures | 200 | 306 | 320 | 854 |
| #Clashes (hard constraint 1) | 3020 | 8590 | 6420 | 28574 |
| #Overlapping classes/courses (hard constraint 2) | 11 | 0 | 52 | 20 |
| #Unavailabilities (hard constraint 7) | 108 | 600 | 72 | 328 |
| Soft constraint 1 in use | Yes | Yes | No | Yes |
| Soft constraint 4 in use | Yes | Yes | Yes | Yes |
| Soft constraint 6 in use | No | Yes | Yes | No |
| Soft constraint 8 in use | Yes | Yes | Yes | Yes |

Table 5: Summary of one artificial and three real-world problems.

The third set of problems consists of one artificial problem and three small to medium-sized real-world problems in Finnish schools, one from each school level. Table 5 summarizes these four problems. The data for these problems has been published on the web [27]. In all of these problems we are searching for a feasible solution which optimizes soft constraints 1, 4, 6 and 8. Other soft constraints are not in use. These four problems are good test instances since

- a) most of the school timetabling algorithms should be able to tackle them because they are not too complicated versions of the problem,
- b) they are quite moderate-sized and
- c) they are different from each other.

The C3 problem is the most constrained and thus the most interesting one. We solved six problems from the standard benchmark instances: All-11-problem, Abramson-15-problem, B3, S3, H3 and C3. We used the parameter values found in the statistical analysis. To keep the results fair we did no further fine-tuning to the extended h-HCGA algorithm. The test runs were performed using the same computers as in Chapter 4. Table 6 summarizes the results.

Furthermore, we decided to run the algorithm using a simulated annealing refinement introduced in [12]. When the GHCM operator selects a new period for a lecture l_i , we apply the following selection mechanism. Periods are examined in random order. A period is selected as the current best one if its fitness is less than the current best one or if the current annealing temperature [28] ac-

cepts their difference. The same mechanism is used in the GHCM operator while removing a lecture. Note that the GHCM operator does not accept upward (increasing cost) move sequences, even if a single move can be upward. Table 6 summarizes the results using the simulated annealing refinement. It also shows the best manual solution the staff of the schools were able to find. The best solutions we have found in all of our experiments can be found in [27].

| Problem identifier | Nbr of runs | Single run time | Best | Median | Worst | Best manual |
|--------------------|-------------|-----------------|------|--------|-------|-------------|
| Extended algorithm | | | | | | |
| All-11 | 8 | 8 hours | 0-0 | 0-0 | 0-0 | |
| Abramson-15 | 8 | 8 | 2-0 | 3-1 | 4-0 | |
| B3 | 8 | 8 | 0-3 | 1-0 | 1-6 | |
| S3 | 8 | 8 | 0-2 | 0-2 | 0-3 | |
| H3 | 8 | 8 | 0-3 | 0-3 | 0-3 | |
| C3 | 8 | 8 | 0-2 | 0-4 | 0-6 | |
| With SA refinement | | | | | | |
| All-11 | 8 | 24 hours | 0-0 | 0-0 | 0-0 | |
| Abramson-15 | | | 2-0 | 2-0 | 3-1 | |
| B3 | | | 0-2 | 0-5 | 1-1 | |
| S3 | | | 0-2 | 0-2 | 0-2 | 0-8 |
| H3 | | | 0-3 | 0-3 | 0-3 | 0-12 |
| C3 | | | 0-1 | 0-1 | 0-3 | 0-10 |

Table 6: Results for standard benchmark instances and the best manual solutions. For example, the value 1-6 stands for one hard constraint and six soft constraint violations.

6. Conclusions and further work

In this paper we considered a highly constrained school timetabling problem and presented a successful algorithm to solve real-world problems as well as artificial test problems. Our algorithm performs very well in the timetabling problems presented in this paper. In order to find out if statistical methods could be of any help in solving the parameter values we ran our algorithm with all possible configurations and analyzed the results using the Friedman test. The Friedman test clearly implied that it is able to extract poor configurations in favor of the good ones. In the near future, we will build the racing process into our program. Our research has shown that the chosen parameter values were very good for the problem C3, but we do not know if they are the best for all problems. Therefore it is best to start with as many configurations as possible and reduce them in time. We have developed a web page [27] that allows other researchers to download the problem description, the standard benchmark instances and the computational results.

References

- [1] E.K. Burke and P. De Causmaecker (2003). *The Practice and Theory of Automated Timetabling IV: Revised Selected Papers from the 4th International conference, Gent 2002*, Springer Lecture Notes in Computer Science, vol. 2740, Springer.
- [2] E.K. Burke and M. Trick (2005). *The Practice and Theory of Automated Timetabling V: Revised Selected Papers from the 5th International conference, Pittsburgh 2004*, Springer Lecture Notes in Computer Science, vol. 3616, Springer.
- [3] E.K. Burke and Hana Rudová (2006). *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling*, Masaryk University.
- [4] A. Schaerf (1999), A Survey of Automated Timetabling, *Artificial Intelligence Review* **13**(2), 87 – 127.
- [5] E.K. Burke and S. Petrovic (2002). Recent Research Directions in Automated Timetabling, *European Journal of Operational Research* **140**(2), 266 – 280.

- [6] B. McCollum (2006). University Timetabling: Bridging the Gap between Research and Practice. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 15 – 35.
- [7] D de Werra D (1985). An Introduction to Timetabling. *European Journal of Operations Research* **19**, 151–162.
- [8] S. Even, A. Itai, and A. Shamir (1976). On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing* **5**, 691 – 703.
- [9] T.B. Cooper and J.H. Kingston (1996). The Complexity of Timetable Construction Problems, in the Practice and Theory of Automated Timetabling, ed. E.K. Burke and P. Ross, Springer-Verlag (Lecture Notes in Computer Science), 283 – 295.
- [10] A. Schaerf and L. Di Gaspero (2006). Measurability and Reproducibility in Timetabling Research: State-of-the-Art and Discussion. In *Proc. of the 6th Int. Conf. on the Practice and Theory of Automated Timetabling*, 53 – 62.
- [11] A. Schaerf (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review* **13**(2), 87 – 127.
- [12] K. Nurmi (1998). Genetic Algorithms for Timetabling and Traveling Salesman Problems. *Ph.D. dissertation*, University of Turku, Finland.
- [13] M. Dorigo, V. Maniezzo, and A. Coloni (1996). The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, **26**(1), 29 – 41.
- [14] M. Dorigo, G. Di Caro, and L. M. Gambardella (1999). Ant Algorithms for Discrete Optimization. *Artificial Life*, **5**(2), 137 – 172.
- [15] Goldberg, David E (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Kluwer Academic Publishers, Boston, MA.
- [16] Vose, Michael D (1999). *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, Cambridge, MA.
- [17] J.D. Landa Silva, E.K. Burke and S. Petrovic (2004). An Introduction to Multiobjective Metaheuristics for Scheduling and Timetabling, *MetaHeuristics for Multiobjective Optimisation* (edited by X.Gandibleux, M.Sevaux, K.Sorensen and V.T'Kindt), Springer Lecture Notes in Economics and Mathematical Systems Vol. 535, 91 – 129.
- [18] Fonseca C.M, Fleming P.J (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation* **3**(1), 1 – 16.
- [19] Smith A.E, Tate D.M (1993). Genetic Optimisation using a Penalty Function. In *Proceedings of the 5th International Conference on Genetic Algorithms*, 499 – 503.
- [20] Richardson J.T, Palmer M.R, Liepins G, Hilliard M (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 191 – 197.
- [21] D Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.
- [22] G Syswerda (1989). Uniform Crossover in Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 2 – 9.
- [23] P. Ross and G.H Ballinger (1993) PGA - Parallel Genetic Algorithm Testbed. Department of Artificial Intelligence. University of Edinburgh.
- [24] W.J. Conover (1999), *Practical Nonparametric Statistics*, John Wiley & Sons, New York.
- [25] M. Birattari & al. (2002). A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 11– 18.
- [26] D. Abramson, M. Krishnamoorthy M, H. Dang (1999). Simulated Annealing Cooling Schedules for the School Timetabling Problem. *Asia-Pacific Journal of Operational Research* **16**, 1 – 22.
- [27] K. Nurmi, J.Kyngäs (2007). School Timetabling Problem – Formulation, Instances and Computational Results. URL: www.samk.fi/sttp.
- [28] P.J.M van Laarhoven, E.H.L Aarts E.H.L (1987). *Simulated annealing: Theory and applications*. Kluwer Academic Publishers.