Solving University Timetabling Problems Using Advanced Genetic Algorithms

Spyros Kazarlis
Technological Educational Institute of Serres
Serres 621 24, Greece
kazarlis@teiser.gr

Vassilios Petridis and Paulina Fragkou
Aristotle University of Thessaloniki
Thessaloniki 540 06, Greece
petridis@eng.auth.gr, fragou@egnatia.ee.auth.gr

***Abstract* – Timetabling problems together with scheduling ones constitute a class of difficult to solve combinatorial optimization problems that lack analytical solution methods. As such, these problems have attracted researchers from a number of disciplines, like Operations Research and Artificial Intelligence, who have proposed a number of methods for solving them. In this paper we present a method based on Genetic Algorithms (GAs), to solve university course timetabling problems. This method incorporates GAs using an indirect representation based on event priorities, MicroGAs and heuristic local search operators in order to tackle a real world timetabling problem. The problem on which the method is applied and tested is a real case and comes from a Technological Educational Institute of Greece. The GA solution is compared to the man-made one produced by the institute's staff and the comparative results are discussed.**

Index Terms – Timetabling, Genetic Algorithms, Micro Genetic Algorithms, Local Search Operators, Combinatorial Optimization.

I. Introduction

According to A. Wren [16]: "Timetabling is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives." Real timetabling problems have many forms like educational timetabling (course and exam), employee timetabling, timetabling of sports events, timetabling of transportation means, etc. Timetabling problems as well as scheduling problems, define a class of hard-to-solve constrained optimization problems of combinatorial nature, Such problems are mainly classified as constraint satisfaction problems [3], where the main goal is to satisfy all problem constraints, rather than optimizing a number of objectives. At present, science has no analytical solution method for all problem cases of this category, other than exhaustive search, which however cannot be applied but only to toy problems, due to the immense search spaces of real problem cases.

Automated timetabling, on the other hand, is a task of great importance as it can save a lot of man-hours work, to institutions and companies, and provide optimal solutions with constraint satisfaction within minutes, that can boost productivity, quality of education, quality of services and finally quality of life. However, large-scale timetables, such as university timetables, may need great effort and many hours of work spent by a qualified person or a team in order to produce high quality timetables with optimal constraint satisfaction and optimization of the timetable's objectives at the same time.

In this paper we focus on university timetabling problems that exist in two forms: course and exam timetabling. Within the scope of this work we focus only on course timetabling problems.

A large number of diverse methods have been already proposed in the literature for solving timetabling problems. These methods come from a number of scientific disciplines like Operations Research, Artificial Intelligence, and Computational Intelligence [1], [7], [11], [14] and can be divided into four categories:

1) Sequential Methods that treat timetabling problems as graph problems. Generally, they order the events using domain-specific heuristics and then assign the events sequentially into valid time slots in such a way that no constraints are violated for each time slot. [4].

2) Cluster Methods, in which the problem is divided into a number of event sets. Each set is defined so that it satisfies all hard constraints. Then, the sets are assigned to real time slots to satisfy the soft constraints as well [15].

3) Constraint Based Methods, according to which a timetabling problem is modeled as a set of variables (events) to which values (resources such as teachers and rooms) have to be assigned in order to satisfy a number of constraints [3], and

4) Meta-heuristic methods, such as genetic algorithms, simulated annealing, tabu search, and other heuristic approaches, that are mostly inspired from nature, and apply nature-like processes to solutions or populations of solutions, in order to evolve them towards optimality [1], [2], [6], [7], [11].

GAs have been used for solving timetabling problems since 1990 [6]. Since then, the literature has hosted a large number of papers presenting evolutionary methods and applications on such problems with significant success. [5].

In this paper we present an advanced GA based method for solving university course timetabling problems. This method uses an indirect representation based on event priorities that is fully described in the next section. It also uses a number of advanced local search operators, including the Micro-GA combinatorial hill-climbing operator, in order to avoid local optima, fulfill constraints and discover optimal solutions effectively.

The proposed method is applied to a real-world university course timetabling problem that comes from the Technological Educational Institute of Serres, Greece, for

This work was partially supported by the Research Committee of the Technological Educational Institute of Serres, Greece, under the research program Archimedes II (Αρχιμήδης II) funded by the Greek Ministry of Education and the E.U.

which a man made solution was also available, and has been used to compare and evaluate the results of the GA method.

The paper is organized as follows: Section II analyzes university course timetabling and the specific real case that is used as a benchmark. Section III presents the advanced GA method and all advanced operators used by it. Section IV presents the simulation results and finally conclusions are presented in Section V.

II. University Course Timetabling

In general a university course timetabling problem consists in finding the exact time allocation within a limited time period (e.g. a week), of a number of events (courses-lectures) and also assign to them a number of resources (a teacher, a room, etc.) in such a way that a number of constraints are satisfied. Usually courses are organized in a number of semesters (e.g. 8). The constraints that have to be satisfied by a timetable are usually divided into two categories: hard constraints and soft ones.

Hard constraints are those constraints that must be rigidly fulfilled. Examples of such constraints are:

- No resource (teacher, student, room, etc.) may be assigned to different events at the same time.
- Events of the same semester must not be assigned at the same time slot (in order for the students of the semester to be able to attend all semester lessons).
- Assigned resources to an event (e.g. teachers) must belong to the set of valid resources for that event (e.g. only specific teachers can teach a specific course).

On the other hand, soft constraints are those that it is desirable to be fulfilled to the possible extent, but are not fully essential for a valid solution. Therefore, soft constraints can also be seen as optimization objectives for the search algorithm. Examples of such constraints are:

- Schedule an event within a particular "window" of the whole period (e.g. on evenings).
- Minimize time gaps or travel times between adjacent lectures of the same teacher.

The specific problem we used in this work comes from the Technological Educational Institute of Serres, Greece, and involves the weekly scheduling of all courses of the Department of Informatics & Communications of this Institute. The specifications of this problem are shown in Table I.

TABLE I
Timetabling Problem Specifications

No.	Description	Quantity
1	No of courses	71
2	No of different lectures	187
3	No of scheduled events	192
4	No of semesters	7
5	Lecture types (theory / lab practice)	2
6	No of teachers	82
7	No of rooms	17
8	No of days	5
9	No of time slots within a day	13

In Table I the value of "13" for the field "time slots within a day" just denotes the possible "starting periods" of each class (from 8:00 am to 20:00 pm) and not complete time slots that can accommodate equal number of consequent classes. As different lectures have different durations (2..4 hours), the real number of consequent classes that can be scheduled within a day depends on the specific set of classes chosen and their durations.

The hard constraints considered for this problem are shown in Table II.

TABLE II
Hard Constraints Of the Timetabling Problem

No	Hard Constraint
1	No resource (teacher or room) may be assigned to different events at the same time
2	Events of the same semester must not be assigned at the same time slot when both events are of type "theory" or when one event is "theory" and one event is "lab practice". Same-semester events can run concurrently only if they are both of type "lab practice", as for each course 4..6 "lab practice" classes are scheduled within the week, each attended by a different group of students.
3	There is a maximum number of time slots per day (13), that may not be exceeded
4	Each lecture may be held in a room belonging to a specific set of valid rooms for the lecture
5	Each room may have its own availability schedule
6	Each lecture may be assigned to a teacher that belongs to a specific set of teachers that can deliver the lecture.
7	Specific lectures must be rigidly assigned to specific teachers.
8	Classes of type "theory" need one teacher while classes of type "lab practice" need 2 teachers.

The soft constraints considered for this problem are shown in Table III.

TABLE III
Soft Constraints Of the Timetabling Problem

No	Soft Constraint
1	Every teacher has it own availability schedule or submits a plan with desirable time periods that suit him best
2	Every teacher has a minimum limit and a maximum limit for man-hour work per week
3	If a class is broken in more than one non-contiguous lectures within a week, a specific number of days must be left between these lectures.
4	Minimize the travel time of teachers and students between rooms within the campus
5	Minimize the time gaps within the schedule of each teacher
6	Minimize the time gaps within the schedule of each room

There are three reasons for selecting the specific case as a benchmark. Firstly because the authors had access to the real constraints considered for developing the man-made solution to the problem, in order to set-up the timetabling problem on a realistic basis. Secondly because the authors had easy access to man-made solutions for this problem, and could easily make comparisons. The third reason was the fact that the specific problem is not an easy one to solve, and thus can serve as a demanding benchmark for developing an efficient optimization algorithm.

The difficulty of the chosen problem case is justified by the following facts:

a) The problem has two types of lectures ("theory" and "lab practice") with diverse characteristics and constraints. In fact nearly every course has a "theory" lesson and a "lab practice" one. Lessons of type "theory" demand one teacher, while lessons of type "lab practice" demand two teachers.

b) The timetable has too few rooms (only 17) that have to accommodate all taught lessons, a fact that makes the timetable very tight. Some of the rooms are laboratories designed for "lab practice" classes, and others are lecture rooms. In practice, all 9 laboratories are occupied by classes, for the full number of periods per day and all five days with only minor time-gaps.

c) Specific classes may be taught in specific rooms. Each "theory" class can be assigned to any of the 8 lecture rooms, but "lab" classes must be assigned to specific "lab" rooms.

d) There is a large number of teachers, each of which has its' own minimum and maximum hour limits per week, and the ability to teach in a limited set of classes.

III. The Advanced GA implementation

In order to solve the timetabling problem described in the previous section, we have developed an optimization method based on Genetic Algorithms (GAs) that incorporate a number of advanced techniques and domain specific local search operators.

The first thing to consider was the representation method to use in order to encode a timetable solution into an encoded form or chromosome, suitable for applying the genetic operators. According to the literature two different approaches are used: "direct" and "indirect" ones.

A "direct" representation [2] directly encodes all event attributes (day, time slot, teacher, room, etc.) for all events. Thus, in these cases the GA has to decide for all timetable parameters and deliver the complete and constraint free schedule. Following this principle results in a very large search space where solutions that satisfy all constraints seem like "needles in a haystack". Moreover, directly encoded solutions, that undergo the genetic operators, frequently result in invalid solutions, that have to be "repaired" or handled in some manner.

In "indirect" representations [11] the encoded solution (chromosome) usually represents an ordered list of events, which are placed into the timetable according to some pre-defined method, or "timetable builder". The timetable builder can use any combination of heuristics and local search to place events into the timetable, while observing the problem's constraints.

For the GA implementation of this work we have chosen an "indirect" representation that encodes 4 fields for each event into the chromosome:

a) Day to allocate the event
b) Teachers (1 or 2) to assign to the event
c) Room where the event will be held
d) Priority to allocate the event within the day.

Of course all fields are first encoded as integers and then entered into the chromosome as binary numbers.

When GA produces such a solution, it first decodes it to gain these four fields for every event in the schedule. Then it invokes a "timetable builder" routine called "timetabler" that works as follows:

1. It separates events into clusters, one for each day.
2. For every cluster, it sorts the events according to their "priority" values and in ascending order (small values mean high priority and are placed first).
3. It takes the first event in the cluster (higher priority), marks it as taken, and tries to place it into the schedule of the particular day.
4. Starting from time slot 1 it places the event and checks if any constraints are violated. If not the allocation is fixed and the algorithm moves on to the next event in the cluster.
5. If any constraints are violated, it tries to allocate the event into subsequent time slots, until all constraints are satisfied.
6. If there exists no time slot for which all constraints are satisfied, the event is marked to violate the "maximum time slots per day exceeded" constraint (constraint no 3 from Table II).
7. The algorithm continues with the next event in the list. When all events have been processed, the "timetabler" moves to the next cluster (day), and this is repeated for all days in the schedule.

The "timetabler" manages to satisfy hard constraints 1,2,5,7 and 8 of Table II, while all other constraints are left to be satisfied by the GA.

After "timetabler" has produced the timetable, it is evaluated through a fitness function that analyzes the solution and calculates its overall fitness value as a sum of weighted scores and penalties for all constraints (hard and soft) as well as objectives. The fitness function used in this GA implementation is of the form:

$$F(x) = \sum_{i=1..6} w_i^s \times P_i^{soft}(x) + \sum_{i=3,4,6} w_i^h \times P_i^{hard}(x) \qquad (1)$$

where x is the timetable under evaluation, $P_i^{soft}(x)$ is a measure of violation of the i^{th} soft constraint, $P_i^{hard}(x)$ is a measure of violation of the i^{th} hard constraint, w_i^s is a weight factor for the i^{th} soft constraint, and w_i^h is a weight factor for the i^{th} hard constraint. This function must be minimized.

From the above it is clear that, some of the problem's constraints, are handled by the "timetabler" during the construction of the complete solution from the genetically

produced abstract solution. The rest of the constraints are handled using a penalty function that is composed as a weighted sum of penalty terms, each of which corresponds to a measure of violation of each constraint. Moreover, soft constraints could also be seen as optimization objectives that have to be optimized by the algorithm to the possible extent.

The next thing to consider was the blend of genetic operators to incorporate into the GA method, in order to achieve maximum optimization performance. To do this we first considered standard operators as well as general purpose combinatorial operators. The operators and their parameters considered for incorporation are shown in Table IV.

TABLE IV
STANDARD OPERATORS AND PARAMETERS CONSIDERED

No	Operator	Parameter
1.1	Crossover	20-point
1.2	Crossover	Uniform
2.1	Mutation	prob=0.005
2.2	Mutation	prob=0.01
3.1	Window Mutation operator	prob=0.1
3.2	Window Mutation operator	prob=0.3
4.1	Swap Chromosome operator	prob=0.1
4.2	Swap Chromosome operator	prob=0.3
5.1	Swap Bit operator	prob=0.1
5.2	Swap Bit operator	prob=0.3
6.1	Swap Window operator	prob=0.1
6.2	Swap Window operator	prob=0.3
7.1	Random Genotype operator	prob=0.1
7.2	Random Genotype operator	prob=0.3
8.1	Mutate Chromosome operator	prob=0.1
8.2	Mutate Chromosome operator	prob=0.3
9	Bit Swap-Mutate Hill Climbing oper.	prob=0.5
10	Window Swap Hill Climbing oper.	prob=0.5
11.1	Varying Fitness Function	Linear
11.2	Varying Fitness Function	Square
11.3	Varying Fitness Function	Exponential
12.1	GA Population	100
12.2	GA Population	200
13	MicroGA Combinatorial Hill Climb.	prob=1.0

Operators 3 (3.1) through 10 are described in [9]. The Varying Fitness Function technique is described in [12]. Operator 13 is described in [8], [10].

The standard GA setup employed Roulette Wheel Parent Selection, population of 50 solutions, standard 5-point crossover operator, bit mutation operator with a probability of 0.001 per bit, elitism, replacement of the whole population of parents with offspring, fitness scaling and a generation limit of 5000 generations. The operators (and their parameters) of Table IV where tested in order to decide whether to adopt them or not in the final algorithm. The corresponding simulation results are presented in section IV.

Due to the specific nature and difficulty of the problem we have also considered domain specific hill climbing operators that are applied only to the best solution of each generation. These operators are:

a) Change Day Hill Climbing Operator

This operator selects an event at random and changes its encoded day-of-allocation field, assigning to it all day values sequentially, except from the original day value. Every time the resulting timetable is evaluated and if it scores better than the original then the change is kept, otherwise the old day value is restored.

b) Fix Teacher Hill Climbing Operator

This operator finds all events with teacher-class constraint violations (constraint 6 of Table II) and selects one such event at random. Then it changes the encoded teacher-to-allocate field, assigning to it all valid teachers sequentially, except from the original one. Every time the resulting timetable is evaluated and if it evaluates better than the original, then the change is kept, otherwise the old teacher value is restored.

c) Fix Room Hill Climbing Operator

This operator finds all events with room-class constraint violations (constraint 4 of Table II) and selects one such event at random. Then it changes the encoded room-to-allocate field, assigning to it all valid rooms sequentially, except from the original one. Every time the resulting timetable is evaluated, and if it evaluates better than the original, then the change is kept, otherwise the old room value is restored.

d) Fix Day Hill Climbing Operator

This operator finds all events that are allocated beyond the maximum time-slot per day limit (constraint 3 of Table II), and selects one such event at random. Then it changes the encoded day-of-allocation field, assigning to it all day values sequentially, except from the original one. Every time the resulting timetable is evaluated, and if it evaluates better than the original, then the change is kept, otherwise the old day value is restored.

It is obvious that these four operators are specifically designed to give the GA the ability to fulfill all three hard constraints (constraints 3,4,6 of Table II) that are not satisfied automatically by the "timetabler". The effectiveness of these operators has been also tested and simulation results are reported in section IV.

IV. SIMULATION RESULTS

In order to decide which operators of Table IV to adopt, one should run simulations for all parameter combinations. However the number of combinations (nearly 420,000) is prohibitive for exhaustive evaluation. Thus, we have applied an "elitism-like" technique in order to reduce the number of simulations needed. This technique goes as follows:

First we conducted a simulation experiment for the standard GA setup described in the previous section. The experiment consisted of 10 independent runs. After the completion of the runs we calculated three statistical figures: the overall best solution quality achieved, the overall worst solution quality achieved and the average

solution quality achieved, throughout the 10 runs. Then, we added the first operator setup of Table IV to the standard GA setup and another simulation round of 10 runs was launched. The results were compared to those of the standard setup via the three statistical figures mentioned above. If the new setup was found to have better performance than the original, then the new setup was adopted as the "best-so-far" setup. Otherwise the tested setup was ignored. With this method only 25 simulations of 10 runs each are needed to evaluate the operators (and their parameters) of Table IV. Of course the validity of this method is based on the assumption that the operators are more or less independent of each other, a fact that is pretty much close to truth, and is also justified by experimental results.

The simulation results for the operators of Table IV are shown in Table V, where all adopted setups are marked with two asterisks "*" and are displayed in bold typeface.

TABLE V
SIMULATION RESULTS FOR STANDARD OPERATORS AND PARAMETERS

Setup	Mean Qual.	Best Qual.	Worst Qual.
Standard	74192	61087	97073
1.1	72387	52087	91080
*** 1.2 ***	**67980**	**60102**	**76077**
2.1	72482	57079	83072
2.2	81798	65113	93120
3.1	61486	50092	76090
*** 3.2 ***	**61596**	**44118**	**78093**
*** 4.1 ***	**66387**	**42095**	**83094**
4.2	65190	52094	76094
5.1	70192	56091	83101
5.2	65991	53101	83085
6.1	65992	54092	88094
6.2	65288	56094	76072
7.1	71793	61117	80096
7.2	76979	52102	89087
*** 8.1 ***	**53590**	**45093**	**65079**
8.2	59789	45097	77103
9	68581	46085	92080
10	58389	48098	71101
11.1	50370	39078	67089
*** 11.2 ***	**52359**	**37093**	**63036**
11.3	57796	47764	69626
12.1	53388	39084	88098
*** 12.2 ***	**42089**	**26104**	**59082**
*** 13 ***	**29882**	**23082**	**35078**

From Table V it is clear that the operators that exhibited best performance and were adopted in the GA scheme were the following:

1. Uniform Crossover
2. Window Mutation operator with a probability of 0.3
3. Swap Chromosome operator with a probability of 0.1
4. Mutate Chromosome operator with a probability of 0.1
5. Varying fitness Function with square increase.
6. GA population of 200 genotypes, and
7. Micro GA combinatorial hill climbing operator.

By adding these operators to the standard GA scheme we have managed to evolve the best overall solution from the value of 61087 for the standard setup, down to the value of 23082 for the advanced setup. A quality of 61087 means that the solution roughly violates 60 hard constraints, while at 23082 only 22 hard constraints are violated.

The next step was to test the effectiveness of the domain specific hill climbing operators described in the previous section. For this reason four (4) more simulation experiments were conducted. Each experiment incorporated one of the four domain specific operators. Again 10 runs were executed for each experiment and each time the results were compared to the best-so-far results. When an operator was found to enhance the performance of the GA optimizer, it was adopted. The simulation results for these operators are shown in Table VI.

TABLE VI
SIMULATION RESULTS FOR DOMAIN SPECIFIC OPERATORS

Operator	Mean Qual.	Best Qual.	Worst Qual.
Change Day	20978	14568	29047
Fix Teacher	18286	12533	29047
Fix Room	13462	9577	23035
Fix Day	70056	3107	11606

As it is clear from Table VI, each one of the four domain specific operators enhances the performance of the GA optimizer and thus, all four operators were adopted in the final scheme. The domain specific operators managed to evolve the best overall solution from the value of 23082 for the advanced setup, down to the value of 3107.

The optimal solution quality of 3107 can be analyzed into two parts:

1. Hard constraints violation part, which equals to 2000
2. Soft constraints violation part, which equals to 1107

The value of 2000 for the first part means that 2 hard constraints are violated at the optimal solution. The value of 1107 for the second part means that all soft constraints are fully satisfied and that gaps within the rooms' and teachers' schedules are adequately minimized.

The next step was to encode and evaluate a man-made solution for the same timetable problem that was already available. The man-made solution was evaluated through the same fitness function that was also used for the GA optimizer. The results of the evaluation of the man-made solution and the GA produced solution are shown in Table VII.

TABLE VII
MAN-MADE AND GA PRODUCED SOLUTIONS ANALYZED AND COMPARED

Feature	Man-made solution	GA produced solution
Fitness	1294	3107
Objective value	1294	1107
Penalty value	0	2000
Hard constr viol. #	0	2
Soft constr viol. #	0	0
Room hour gaps	94	5
Teacher hour gaps	102	1

where "objective value" is the part of the fitness value attributed to the violation of soft constraints (objectives), "penalty value" is the part of the fitness value attributed to hard constraints, "room hour gaps" is the total number of hours within the rooms' schedules during which the rooms are unoccupied, and "teacher hour gaps" is the total number of hours within each teacher's schedule during which the teacher does not have a class assignment.

As can be clearly seen from Table VII, the GA optimizer does not manage to satisfy all hard constraints, although it comes very close to achieving it, by producing a solution with only 2 violating constraints. On the other hand it is also evident that the GA produced solution satisfies soft constraints better than the man-made one. The GA solution scores an objective value of 1107 compared to 1294 of the man-made solution. The value of 1107 corresponds to only 5 "room hour gaps" and only 1 "teacher hour gap" compared to 94 and 102 of the man-made solution respectively. It seems like the man-made solution was the outcome of a focused effort to satisfy hard constraints, while soft constraints haven't enjoyed much attention. On the other hand the GA's solution is well-developed concerning soft constraints but fails to be 100% valid.

V. Conclusions

In this paper an advanced GA implementation has been presented for solving university course timetabling problems. The GA method proposed uses an indirect representation featuring event allocation priorities, and invokes a "timetable builder" routine for constructing the complete timetable. It also incorporates a number of standard and domain specific operators to enhance its search efficiency. The GA implementation has been applied on a real world university course timetabling problem, for which man-made solutions were also available. It has been shown through extensive simulation experiments, that the incorporation of certain combinatorial and domain specific operators can significantly enhance the search efficiency of the evolutionary algorithm.

Direct comparison of the GA-produced solutions with the man-made one shows that, although the evolutionary method does not manage to satisfy all hard constraints of the problem, it achieves a significantly better score in satisfying soft constraints and, therefore, its performance can be characterized as promising. GA's inability to satisfy all hard constraints may be attributed to the difficulty of the specific problem and to the limited resources (5000 generations) it used during the experimental simulations.

Of course more simulations are needed in order to develop a more efficient version of the proposed method that will be able to produce solutions with zero hard constraint violations. Moreover, in order to test the efficiency and robustness of the proposed method, it should be applied and tested on more real world timetabling problems. Another direction for further research is the adaptation of the method for solving university exam timetabling problems, or other timetabling and scheduling problems.

References

[1] D. Abramson, "Constructing school timetables using simulated annealing: sequential and parallel algorithms," *Management Science*, 37(1), January 1991, pp. 98-113

[2] P. Adamidis and P. Arapakis, "Evolutionary Algorithms in Lecture Timetabling," *Proceedings of the 1999 IEEE Congress on Evolutionary Computation (CEC '99)*, IEEE, 1999, pp. 1145-1151.

[3] S.C. Brailsford, C.N. Potts and B.M. Smith, "Constraint Satisfaction Problems: Algorithms and Applications," *European Journal of Operational Research*, vol 119, 1999, pp. 557-581.

[4] M.W. Carter, "A Survey of Practical Applications of Examination Timetabling Algorithms," *Operations Research* vol. 34, 1986, pp. 193-202.

[5] M.W. Carter and G. Laporte, "Recent Developments in Practical Course Timetabling," In: Burke, E., Carter, M. (Eds.), *The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd Int'l Conf. on the Practice and Theory of Automated Timetabling*, Springer Lecture Notes in Computer Science Series, Vol. 1408, 1998, pp. 3-19.

[6] A. Colorni, M. Dorigo, and V. Maniezzo. "Genetic algorithms - A new approach to the timetable problem," In *Lecture Notes in Computer Science - NATO ASI Series*, Vol. F 82, Combinatorial Optimization, (Akgul et al eds), Springer-Verlag, 1990, pp. 235-239.

[7] A. Hertz, "Tabu search for large scale timetabling problems," *European journal of operations research*, vol. 54, 1991, pp. 39-47.

[8] S.A. Kazarlis, "Micro-Genetic Algorithms As Generalized Hill-Climbing Operators for GA Optimization of Combinatorial Problems – Application to Power Systems Scheduling", *Proceedings of the the 4th Conference on Technology and Automation*, October 2002, Thessaloniki, Greece (Dept.of Automation, A.T.E.I. of Thessaloniki, Greece), pp. 300-305.

[9] S.A. Kazarlis, A.G. Bakirtzis, V. Petridis, "A Genetic Algorithm Solution to the Unit Commitment Problem," *IEEE Transactions on Power Systems*, Vol. 11, No. 1, February 1996, pp. 83-92.

[10] S.Kazarlis, S.Papadakis, J.Theocharis and V.Petridis, "Micro-Genetic Algorithms as Generalized Hill Climbing Operators for GA Optimization," *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 3, June 2001, pp. 204-217.

[11] B. Paechter, A. Cumming, M.G.Norman, and H. Luchian, "Extensions to a memetic timetabling system," In E.K. Burke and P.M. Ross, eds., *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, 1995.

[12] V. Petridis, S. Kazarlis and A. Bakirtzis, "Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 28, Part B, No. 5, October 1998, pp. 629-640.

[13] A. Schaerf, "A Survey of Automated Timetabling," *Artificial Intelligence Review*, vol 13 (2), 1999, 87-127.

[14] Arabinda Tripathy, "A lagrangian relaxation approach to course timetabling," *Journal of the Operational Research Society*, vol. 31, 1980, pp. 599-603

[15] G.M. White and P.W. Chan, "Towards the Construction of Optimal Examination Timetables," *INFOR 17*, 1979, p.p. 219-229.

[16] A. Wren, "Scheduling, Timetabling and Rostering – A Special Relationship?,", in *The Practice and Theory of Automated Timetabling: Selected Papers from the 1st Int'l Conf. on the Practice and Theory of Automated Timetabling*, Burke, E., Ross, P. (Eds.) Springer Lecture Notes in Computer Science Series, Vol. 1153, 1996, pp. 46-75.