Automated Time Table Generation Using Multiple Context Reasoning for University Modules

Dipti Srinivasan Tian Hou Seow Jian Xin Xu
Department of Electrical & Computer Engineering
National University of Singapore
10 Kent Ridge Crescent, Singapore 119260

Abstract— **Finding a feasible lecture/tutorial timetable in a large university department is a challenging problem faced continually in educational establishments. This paper presents an evolutionary algorithm (EA) based approach to solving a heavily constrained university timetabling problem. The approach uses a problem-specific chromosome representation. Heuristics and context-based reasoning have been used for obtaining feasible timetables in a reasonable computing time. An intelligent adaptive mutation scheme has been employed for speeding up the convergence. The comprehensive course timetabling system presented in this paper has been validated, tested and discussed using real world data from a large university.**

Index Terms—evolutionary algorithm, timetabling, hybrid intelligent systems, scheduling

I. INTRODUCTION

THE time-tabling problem (TTP) is basically the scheduling and assignment of the lessons into appropriate time-slots and resources respectively, without causing time clashes for the students and the teachers, as well as the resource clashes [1]. The drawing up of university timetables is a slow, laborious task, performed by people working on the strength of their knowledge of resources and constraints of a specific institution.

The construction of course timetables for academic institutions is a very difficult problem with a lot of constraints that have to be respected and a huge search space to be explored, even if the size of the problem input is not significantly large, due to the exponential number of the possible feasible timetables. On the other hand, the problem itself does not have a widely approved definition, since different variations of it are faced by different departments. This problem has therefore proven to be a very complex and time-consuming problem. Timetables are considered feasible provided the so-called hard constraints are respected. However, to obtain high-quality timetabling solutions, soft constraints, which impose satisfaction of a set of desirable conditions for classes and teachers should be satisfied.

Over the past few years, a wide array of techniques have been proposed for solving the course timetabling problem and its' variants. Several techniques have been developed for automated timetable generation [2, 3]. One of the popular technique is based on graph coloring algorithms [3, 4]. The main disadvantage of this approach is the difficulty of incorporating application specific constraints into the problem formulation. Other methods include artificial neural networks [5], simulated annealing [6], and tabu search [7]. Constraint Based Programming [8] has become an interesting approach for solving timetabling problems recently. This approach combines the declarativity of logic programming with the efficiency of constraint solving.

Recently, a number of researchers have applied evolutionary computation techniques to timetabling problems [9-18]. The problem was formulated using fuzzy logic in [9], and heuristics were used to improve the generalization capabilities. Ref. [10] represented the timetabling problem as a graph, and then manipulated it as a Graph-Colouring Problem, taking into account the presence of solved cliques in the mating parents. A method of decomposing large timetabling problems into smaller components, each of which was of a size that the EA could effectively handle was presented in [11]. A problem -specific chromosome representation and knowledge-augmented genetic operators were developed in [13] to intelligently avoid building illegal timetables. In [15], GA was combined with a heuristic specific greedy algorithm to take advantage of the global search of feasible solutions and specific technique efficiency in local solution optimization. This approach resulted in considerably smaller execution times. The methods described in [16] transform the large search space into one in which the proportion of feasible solutions is greatly increased. The chromosomes used are encoded instructions on how to build a timetable in a way that leads to the above-mentioned search space transformation. These approaches have established the effectiveness of genetic-based approaches for finding feasible solutions to this highly constrained problem. Hybrid approaches have also been proposed to further improve the quality of the solutions. Each implementation, however, is specific to the academic institution for which it is developed, and addresses a different problem with diverse constraints.

Although, the timetabling problem is distinctive for each institution, there exist a set of entities and constraints which are common to every possible instantiation of the timetabling problem. In this paper, we present a model of this common

0-7803-7282-4/02/$10.00 ©2002 IEEE

core in terms of a knowledge-augmented evolutionary algorithm approach which may be extended to cover the needs of a specific academic unit. The algorithm incorporates two distinct objectives that concern precisely the minimization of the violations of both types of constraints, hard and soft. The problem is modeled as a bi-objective problem to construct feasible assignments of course modules to lecture rooms on specified timeslots. A new representation for the timetabling problem is presented.

A brief description of the problem is presented, followed by the evolutionary algorithm, using a standard fitness-sharing scheme improved with an elitist secondary population. This approach represents each timetabling solution with a matrix-type chromosome and is based on special-purpose genetic operators of crossover and mutation developed to act over the population. The paper concludes with a discussion of the favorable results obtained through an application of the algorithm to a real instance taken from a university establishment in Singapore.

II. The Time-Tabling Problem

The time tabling problem can be modeled as a constraint satisfaction problem with many parameters and loose constraints. These constraints have to be modeled in a format that can be handled efficiently by the scheduling algorithm. The scheduling involves allowing for a number of pairwise restrictions on which jobs can be done simultaneously. For instance, in attempting to schedule classes at a university, two courses taught by the same faculty member cannot be scheduled for the same time slot. Similarly, two course that are required by the same group of students also should not conflict. The constraints considered for this problem are:

The hard constraints: These are constraints that must be satisfied in order to obtain a feasible timetable. The hard constraints in this problem model can be viewed from 3 different perspectives as follows:

- Student's perspective – *"All my lessons must not have any time slot clashes!"*
- Resource manager's perspective – *"No two or more rooms could use a lecture theatre or tutorial room during a common time slot!"*
- Teacher's perspective – *"I cannot teach more than one lesson during any time slot!"*

The soft constraints: These are constraints of lower priorities to be satisfied. The violation of the soft constraints will not cause the timetable to lose its feasibility. The soft constraints in this problem model can be viewed from 2 different perspectives as follows:

- Student's perspective – *"I want my week to be short and I don't want to have too many empty slots in between lessons!"*
- Teacher's perspective – *"I cannot teach for more than 2 hours continuously!"*

The objective of the optimization algorithm is to obtain the best schedule while satisfying all these constraints.

III. The Problem Representation

This section shows how the problem was represented for assignment of course modules into appropriate time-slots and resources respectively, without causing time clashes for the students and the teachers, as well as the resource clashes.

The development of algorithm for timetabling in a large university department is illustrated in this paper by using second year electrical engineering (EE2) cohort as an example. There is only one large lecture theatre available for conducting the lectures, while a large number of smaller tutorial rooms is available.

Due to a very large number of students enrolled in the department, the entire cohort is divided into two groups. The number of modules to be taken by the students for each semester are classified into 3 categories as follows: Group A modules, which are taken by half the batch of the Year 2 students. Group B modules, which are taken by the remaining half batch of the Year 2 students. Group C modules, which are the common modules for the Year 2 students. Suppose in semester 1, the first half batch takes the Group A modules, then in semester 2, the first half batch will take the Group B modules and vice versa. Among the modules, the lessons conducted are in forms of 2-hours lectures, 1-hour tutorials and 3-hours laboratory sessions.

A. Chromosome Representation

The chromosome is made up of genes and each gene represents a unique class (i.e. a lecture or a tutorial). As only two laboratory sessions are required for each module, some 3-hour afternoons slots are allocated for labs.

The gene value represents the time slot, in which the lesson is conducted. Two genes are required to represent a lecture and one gene is required to represent a tutorial. As there is only one lecture theatre, the gene that is allotted to a particular lecture would be assigned to that one and only lecture theatre. As for genes, which represent tutorial classes, they will be assigned to any tutorial room, since tutorial rooms are assumed to be infinite. The actual chromosome representation is shown in Fig. 1. The gene value representation is shown is Fig. 2.

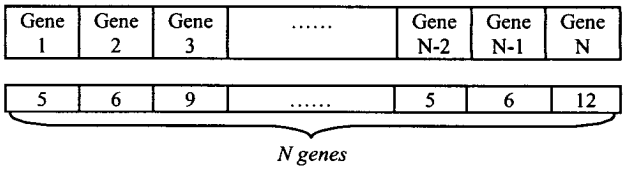

Fig. 1. An actual chromosome representation of N genes.
Each gene represents a unique lesson, say gene 3 represents EE2003 tutorial for class EE2. The gene value assigned to each gene represents the time slot in which the lesson is conducted.

	Mon	Tues	Wed	Thur	Fri
0800-0845	1	11	21	31	41
0900-0945	2	12	22	32	42
1000-1045	3	13	23	33	43
1100-1145	4	14	24	34	44
1200-1245	5	15	25	35	45
1300-1345	6	16	26	36	46
1400-1445	7	17	27	37	47
1500-1545	8	18	28	38	48
1600-1645	9	19	29	39	49
1700-1745	10	20	30	40	50

Fig. 2. Gene value representation.
The gene values range from 1 to 50. If a gene takes a value of say 25, it means that the lesson that the gene is representing will be conducted on Wednesday during the time slot of 1200hrs-1245hrs.

B. *Initialization*

The initialization of the algorithm is split into 2 parts: Initialization of the module data and the random generations of the initial chromosome population.

Initialization of module data: The module data that is required by the program are as follows:

- Total number of modules.
- Total number of classes (e.g. EE1-EE24)
- Total number of classes taking Group A modules (e.g. EE1-EE12)
- Total number of classes taking Group B modules (e.g. EE13-EE24)
- The module information (i.e. the number of timeslots needed for lecture and tutorial per week)
- Periods available for the lecture theatre and the tutorial rooms.

The above data stated are being initialized as follows:

- Assignment of gene ID to the lessons. (e.g. Module EE2004 tutorial for class EE3 is assigned with gene ID 4. It will mean that gene 4 in the the lecture and L2 represents the second slot of the chromosome will be representing the EE2004 tutorial class EE3.)
- Assignment of lesson type (T, L1, L2) to the genes. T represents tutorial, L1 represents the first slot of lecture. L1 and L2 are required because the lectures are conducted on a 2-hour basis.
- Assignment of group type (A, B, C) to the genes. A represents that the particular lesson belongs to group A modules, B represents group B modules and C represents group C modules.

A data template of the chromosome representation is then created based on the three assignments and is represented in Fig. 3. This template will be very useful during the second part of initialization, evaluation of cost function and mutation.

Gene 1	Gene 2	.....	Gene N-1	Gene N
L1	L2	.....	T	T
A	A	.....	B	C

Fig. 3. Data template of the chromosome representation.

C. *Generation of initial population*

The generation of the initial population is an *intelligent* randomized process. The randomized allocation of the gene values in the chromosome is based on the module data given on the resources available time slots. For each gene in the chromosome, the generator will check with the data template whether it is a lecture (L1) or a tutorial (T). If the gene is a lecture, it will check with the lecture theatre's available time slots and assign suitable time slots based on the available time slots randomly to the 2 lecture genes (L1 and L2). This is possible because the gene checking is done on a sequential basis from gene 1 to N, where N is the total number of lessons, and L1 gene will be detected first and when a random time slot is assigned to the gene, say time slot 2, the L2 gene will be automatically assigned with time slot 3 by the program. As for a tutorial gene, it will be done in the likewise manner. This *intelligent* process will provide the E.A process a jump-start, by producing timetables with fewer violations of the constraints than a completely random initialization would do, at roughly the same computational cost.

D. *Crossover*

After the initialization process, the next step of E.A would be the crossover operation. In the crossover operation, two chromosomes from the initial population will be randomly selected to produce a child chromosome. The first selected chromosome is viewed as the father chromosome and the second selected chromosome is viewed as the mother chromosome. A crossover point is randomly selected. The father chromosome will provide the genes of the child chromosome from the first gene up to the crossover point and the mother chromosome will provide the rest of the genes. The crossover process is illustrated in Fig. 4.

Suppose that the initial population consists on M chromosomes, M crossover operations will be required to generate M child chromosomes. The M child chromosomes will be added on to the initial population of M to make up a new population of 2M. Following the crossover process, the new population will undergo the mutation process.

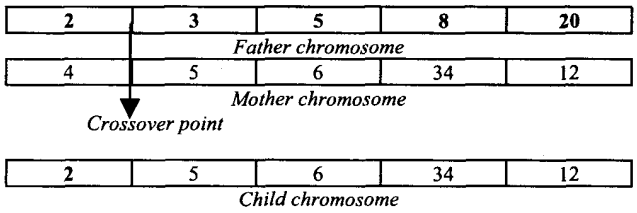

Fig. 4. Crossover process between 2 chromosomes of 5 genes.

E. Mutation

The mutation process is done with an intelligent and adaptive approach. The mutation operator selects a chromosome randomly out of the new population, and chooses a gene randomly from the chromosome for mutation. The gene value of the chosen gene of the selected chromosome will be changed randomly, but selectively, based on the gene data, whether it is a lecture or tutorial. The approach is similar to that as explained in the initialization process. This approach will lead to higher tendency for the mutated chromosome to become a fitter one. Besides having this intelligent gene mutation, the rate of mutation increases when the cost of the best chromosome of the population reduces to one-third of its initial best cost. The reason for the increase of the mutation rate will be discussed in Section III. The change in the rate of mutation has improved E.A performance significantly.

F. Evaluation

The evaluation process calculates the cost of all the chromosomes in the new population after the mutation process. The cost function of the evaluator is as follows:

- Cost = Summation (Penalty)
- Penalties assigned:
  - Time slot clash for tutorial classes = 1
  - Resource clash for lectures = 5

Non-paired lectures (L1 & L2) = 10

The assignment of the penalties is arbitrary, but it is based on certain logic. Non-paired lectures are considered the worst violation of the constraints, because paired lectures are the main criteria of the ECE department timetabling rules. Resource clash for lectures is ranked second because there is only one lecture theatre to be shared among the Groups A, B and C modules. Finally, the time slot clash for tutorial classes is ranked the lowest, because there is an infinite number of tutorial rooms and the time clashes could be resolved later when the first two constraints are satisfied. The assignment of the penalties will actually filter out chromosomes that violate the two higher priority constraints. The lowest constraint will be satisfied at the last.

G. Selection

After the evaluation process, every chromosome in the population is assigned with a cost. The selection operator will sort the chromosomes from the lowest to the highest cost. M chromosomes with the lowest costs will be selected for the next E.A generation. However, when it detects a chromosome with zero cost, it will stop the E.A operation and decodes the best chromosome of the zero cost into readable timetable.

H. Overview of the E.A

The overview of the E.A flow is as expressed in the following pseudo code:

- Initialization
- While (Cost of best chromosome is not zero) loop
  - Crossover
  - Mutation
  - Evaluation
  - Selection
- End loop
- Decode best chromosome into readable timetable.

IV. Experimental Results

This work has produced impressive results, which may provide future researchers an alternative approach to improve the performance of E.A. A automated Time-Tabling program, based on the EE Year 2 cohort, has been implemented in C++ using evolutionary algorithm and multiple context reasoning.

A. Genetic parameters used

The genetic parameters used in the program are as follows:

- Population Size, M = 500
- Mutation rate (normal), $p_{m,n} = 0.001$
- Mutation rate (boosted), $p_{m,b} = 0.05$
- Mutation rate boosting point = 1/3 of initial best cost

Population Size: The population size, M, is chosen to be 200 after some experimental testing by changing M from 100 to 500, in the increments of 100. The performance plots (Cost of best chromosome against the number of generations required) are shown in Fig. 5.

From the experimental results yielded, it is found that setting M to be 200-500 would yield roughly the same number of generations required. We choose M to be 500, after conducting some experiments with different population sizes.

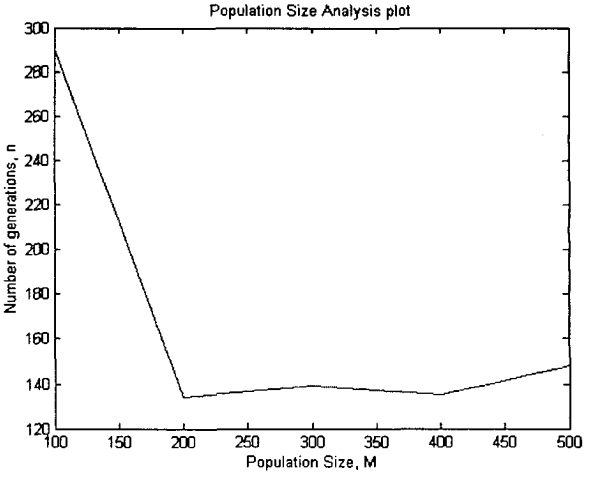

Fig. 5. Performance analysis on the population size.

Cross-over rate: There is no fixed crossover rate in this algorithm. The crossover operation stops when an additional N offsprings are generated and added to the original N chromosomes to make the total population size to 2N.

Mutation rate: The normal mutation rate, before the boost, is set to 0.001 arbitrarily. The boosted mutation rate is set to be 50 times higher to the normal rate, i.e. 0.05. The results are obtained after some experimental testing from 0.01 to 0.06, with the increments of 0.01. The results are shown in Fig. 6.

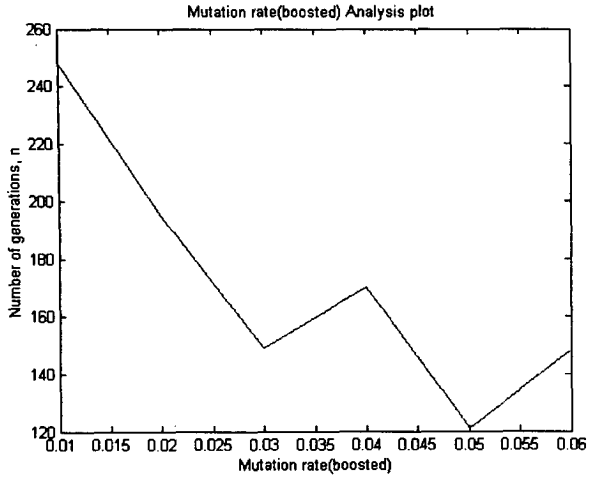

Fig. 6. Performance analysis on the boosted mutation rate.

From the experimental results yielded, the boosted mutation rate is set to be 0.05.

Mutation rate boosting point: The mutation rate boosting point is the point where the mutation rate is boosted. In the initial analysis on the performance plots (cost of best chromosome against number of generations) when mutation rate is kept constant at 0.001, it is found that the performance plot loses its linearity when the cost of the chromosome decreases to around 1/3 of its initial best cost. A sample result is shown in Fig. 7.

Analysis has also been done on the final population when the chromosome with cost zero is found. Results have shown that the chromosomes in the population have become very similar to each other, perhaps only a few genes different. This result has suggested that when the cost of the best chromosome has decreased to a certain amount, the population of the chromosomes has being generalized to a certain pattern by the E.A process and it will be highly dependent on the mutation process to improve the population, as crossover operations would have little impact on the improvement. For our work, we have chosen the mutation rate boosting point to be 1/3 of the initial best cost.

Results have shown that implementing the intelligent adaptive mutation operator have led a more than 10 times of improvement in the performance of E.A. The number of generations required to obtain a feasible timetable reduces from an average of 1200 to an average of 130.

B. Experimental results

Using the genetic parameters as mentioned earlier, the automated time-tabling program is run for at least 100 times on an AMD Athlon K7 550Mhz PC with 256MB of memory.

The following results are yielded based on the real data given by the ECE department of NUS:

- Number of modules = 9
- Number of unique lessons = 130
- Length of chromosome = number of unique lessons = 130
- Number of lecture slots available = 16
- Number of tutorial slots available = 22
- Number of generations required to obtain a feasible timetable = 130 (average)
- Time taken to generate a feasible timetable = 3 minutes (average)

A sample performance graph is shown in Fig. 8.

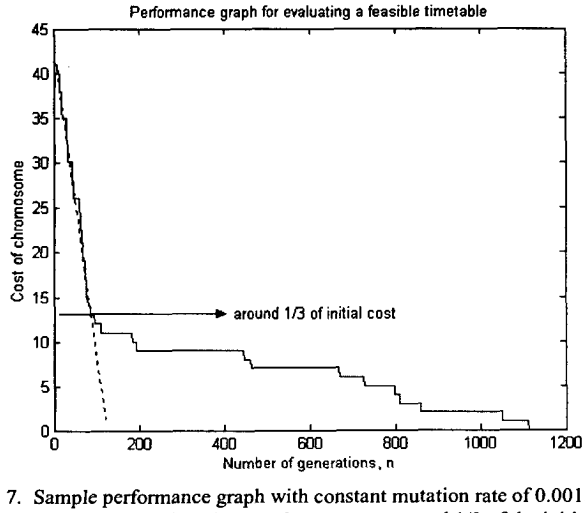

Fig. 7. Sample performance graph with constant mutation rate of 0.001. The plot loses its linearity when the cost decreases to around 1/3 of the initial best cost.

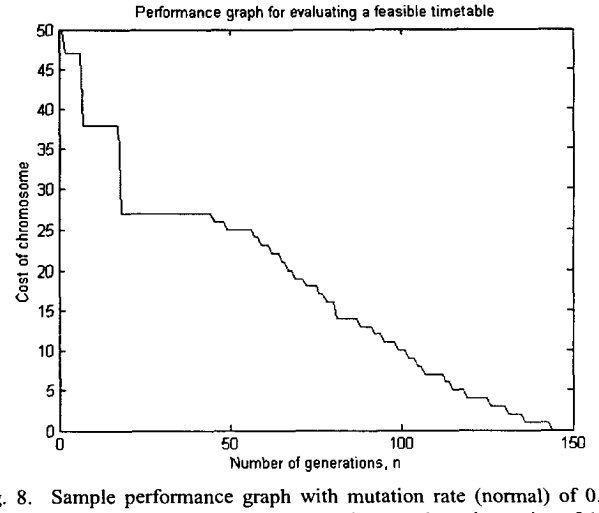

Fig. 8. Sample performance graph with mutation rate (normal) of 0.001, mutation rate (boosted) of 0.05 and mutation rate boosting point of 1/3 of initial cost.

The results have shown positively in the feasibility of applying E.A to solve the time-tabling problem. One important result that is being discovered on the behavior of E.A during the implementation of the work is the generalization of the chromosome population when its best chromosome reaches certain cost. This discovery has suggested two implications: First, there is a need to find out the mutation rate boosting point, in order to improve the performance of E.A. Second, it is more feasible to implement a dynamic mutation operator which will be activated at the mutation rate boosting point. The mutation rate (boosted) could be found by experiment. The mutation rate (boosted) and the mutation rate boosting point used in this work are found on an experimental trial and error basis, and they are not optimal in every situation.

The following heuristic has been used when implementing the dynamic mutation operator:

- Implement the E.A with constant mutation rate.
- Run the E.A for a number of times, say 100 times, and find the point when the performance plot loses its linearity for each run. (e.g. 1/2 of initial cost for run 1)
- Find the average of the points obtained from the runs.
- Define the average result as the Mutation rate boosting point.
- Implement the mutation rate (boosted) from the boosting point onwards.
- Find the optimal mutation rate (boosted) by experiments.

V. CONCLUSION

This paper addresses the Timetabling Problem (TTP), which covers a very broad range of real problems faced continually in educational institutions, and we describe how Evolutionary Algorithms (EAs) can be employed to effectively address arbitrary instances of automated timetabling problem. We present an algorithm combining constraint satisfaction and local search methods in order to solve a real-world university timetabling problem involving multiple contexts. The algorithm guides the construction of feasible timetables whereas local search attempts to resolve constraint violations which may still arise. An intelligent adaptive mutation scheme has been employed for speeding up the convergence.

The heuristic as suggested is a general rule of thumb and there may be some empirical relationships between genetic parameters to obtain optimal solutions. Future work on this algorithm will explore this.

REFERENCES

[1]. *De Werra D.*, "An introduction to timetabling", European Journal of Operations Research", Vol. 19, 1985, pp. 151-162.

[2]. *Carter M. W., Laporte G.*, "Recent developments in practical course timetabling", Lecture Notes in Computer Science, Vol. LNCS1408, Springer-Verlag, 1998, pp. 3-19.

[3]. *Schaerf, A.*, "A survey of automated timetabling", Artificial Intelligence Review, No. 13, 1999, pp. 87-127.

[4]. *Neufield, G. A.; Tartar, J*, "Graph coloring conditions for existence of the solution to the timetabling problem", Communications of the ACM, No. 17, Vol. 8, 1974.

[5]. *Yu, T. L.*, "Time-table scheduling using neural network algorithms", IJCNN International Joint Conference on Neural Networks, 1990, Page(s): 279 -284 vol.1.

[6]. *Downsland K. A.*, "Simulated annealing solutions for multi-objective scheduling and timetabling", In Modern Heuristic Search Methods. Wiley. Chichester, England , 1996, pp. 155-166.

[7]. *Hertz A.*, "Tabu search for large scale timetabling problems", European Journal of Operations Research, Vol. 54, 1991, pp. 39-47.

[8]. *Abbas, A.M.; Tsang, E.P.K.,* " Constraint-based timetabling-a case study", ACS/IEEE International Conference on Computer Systems and Applications, 2001, Page(s): 67 –72.

[9]. *Calogero Di Stefano; Andrea G. B. Tettamanzi*, "An Evolutionary Algorithm for Solving the School Time-Tabling Problem", Lecture Notes on Computer Science, LNCS 2037, p. 452.

[10]. *Terashima-Marin, H.; Ross, P.; Valenzuela-Rendon, M.,* "Clique-based crossover for solving the timetabling problem with GAs", Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99, pp. 1999 -1206 Vol. 2.

[11]. *Burke, E.K.; Newall, J.P.* , "A multistage evolutionary algorithm for the timetable problem", IEEE Transactions on Evolutionary Computation, Volume: 3 Issue: 1 , April 1999, Page(s): 63 –74.

[12]. *Adamidis, P.; Arapakis, P.* , "Evolutionary algorithms in lecture timetabling", Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99 , 1999 -1151 Vol. 2.

[13]. *Erben, W., Keppler J.*, "A Genetic Algorithm Solving a Weekly Course-Timetabling Problem". Practice and Theory of Automated Timetabling, Selected Papers, Springer-Verlag, 1995.

[14]. *Caldeira JP and,Rosa AC*, "School Timetabling using Genetic Search", Proceedings of PATAT, 1997, pp 115-122.

[15]. *Arous, N.; Abdallah, S.B.; Ellouze, N.,* " Evolutionary potential timetables optimization by means of genetic and greedy algorithms", Proceedings of the International Conference on Information Intelligence and Systems, 1999, Page(s): 24 –31.

[16]. *Paechter, B.; Cumming, A.; Luchian, H.; Petriuc, M.,* " Two solutions to the general timetable problem using evolutionary methods", Proceedings of the First IEEE Conference on Computational Intelligence, World Congress on Computational Intelligence, 1994, Page(s): 300 -305 vol.1.

[17]. Colorni, M. Dorigo, and V. Maniezzo. *Genetic algorithms and highly constrained problems: the time-table case.* In H.-P. Schwefel and R. Manner, editors, Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1, LNCS 496, pages 55--59, Dortmund, Germany, 1-3 October 1991. Springer-Verlag.

[18]. *Sheung, J.; Fan, A.; Tang, A,* " Time tabling using genetic algorithm and simulated annealing", Proceedings of 1993 IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering.Proceedings. TENCON '93, Page(s): 448 -451 vol.1.