UNIVERSITY COURSE SCHEDULING USING PARALLEL MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS

[1]M. M. ALDASHT, [2]M. H. SAHEB, [3]I. NAJJAR, [4]M. H. TAMIMI, [5]T. O. TAKRURI

[1]Asstt Prof., Department of Information Technoloy, PPU, Hebron, Palestine

[2]Asstt Prof., Department of Information Technoloy, PPU, Hebron, Palestine

[3]IT student, Department of Information Technoloy, PPU, Hebron, Palestine

[4] IT student, Department of Information Technoloy, PPU, Hebron, Palestine

[5]PC Lab. Admin., Department of Information Technoloy, PPU, Hebron, Palestine

ABSTRACT

Evolutionary Algorithm (EA) provides a mechanism that can achieve efficient exploration for design spaces. Thus, it constitutes an efficient tool for identifying the best alternatives to implement the solution of a certain problem. In this work, EA is implemented to solve the university course scheduling problem and a real data from Palestine Polytechnic University (PPU) databases is used for testing. Sequential implementation of such a complex problem will suffer a long execution time to find sub-optimal solution. On the other hand, using single objective optimization model soft and hard constraints could not be well satisfied. In this work, we have implemented the EA using parallel programming techniques. This permits the execution of the program in a cluster computer to reduce the execution time. Also, many soft constraints can be considered along with the hard constraints in order to get better solutions. Results show that, after redrafting the algorithm to be multi-objective, soft cost could be reduced to the minimum when using enough individuals and iterations, at the same time, hard constraints are still satisfied. After distributing the algorithm on 7 machines with 11 processors the obtained speedup is around 6 on average and the quality of the obtained solution has improved considerably.

Keywords: *Parallel Evolutionary Algorithms, Multi-objective Optimization, University Course Scheduling.*

1. INTRODUCTION

University Course Scheduling (UCS) can be considered as an instant of what so called timetabling problem. In which time slots and teachers must be assigned to a set of courses in a way that satisfies a set of hard constraints and minimize the cost of another set of soft constraints [8]. This problem is considered to be a non-polynomial-time hard (NP-hard) problem, which means that the amount of computation required for finding solutions increases exponentially with problem size [14]. Evolutionary algorithms are good algorithms for solving complex and non-linear scheduling problems [2].

Scheduling problem, in general, can be defined as a problem of finding the optimal sequence for executing a finite set of operations under a set of certain constraints [10]. University course scheduling problem includes the process of assigning a set of classrooms and set of instructors to a given set of courses taking into account a set of constraints. The final solution is required to satisfy a set of constraints, these constraints are divided into hard constraints, which must be satisfied and soft constraints which should be satisfied.

Examples of hard constraints are: no person can be in more than one place at a time, and the total resources allocated to some time slot must be less than or equal to the resources that are available in that period, etc. Examples of soft constraints are: some lecturer should not have very late classes daily; a group of students should not have consecutive classes in different and large distant places, and other individual's preferences, etc. [4].

Various choices of objectives, such as a compact timetable, minimum number of consecutive classes of teachers and so on, lead the scheduling of a class timetable to a multi-objective optimization problem. However, a very limited number of researchers considered multiple objectives in the problem [12].

On the other hand, there are two types of algorithms could be used for solving the problem, sequential and parallel. Parallelization can improve considerably the quality of the solutions obtained, compared with sequential implementation of the algorithm [5]-[6].

The performance of all different evolutionary algorithms, most notably GAs, in solving the university course scheduling problem has been widely studied. Those algorithms are highly dependent on their special operators, i.e., mutation and crossover. Because the search space of the problem is very complex, a more general approach is needed to guarantee a robust exploration and to avoid the local minimum problem which is frequent in such search spaces. In this paper, our approach, built after [4] using parallel approach, while keeping its simple representation for the solution and a robust exploration for the search space.

2. THE RESEARCH METHODOLOGY

The academic class timetabling problem is being studied for more than four decades, a general solution technique for it, considering different aspects of its variants, is yet to be formulated. Despite multiple criteria to be met simultaneously, the problem was generally tackled as single-objective optimization problem. Moreover, most of the earlier works were concentrated on school timetabling, and only a few on university class timetabling [1].

On the other hand, in many cases, the problem was over-simplified by skipping many complex class-structures. To overcome these shortages the multi-objective approach was identified. Multi-objective optimization (MOOP) methods introduce a new approach for optimization that is founded on compromises and trade-offs among the various objectives. The aim of MOOP methods is to discover a set of satisfactory compromises and, through them, the global optimal solution by optimizing numerous dependent properties simultaneously.

2.1 Problem Modelling

In this work, timetabling problem has been implemented on one of the four colleges of the university which is the College of Administrative Sciences and Informatics. Then, the methodology can be generalized on the rest of colleges. In this college there are four academic 4-years programs. Thus, courses are offered at the beginning of each semester for 4 student groups in each academic program. In total we have 16 groups (4 levels * 4 programs). On the other hand, we have five work days a week (Sunday to Thursday). On Sunday, Tuesday, and Thursday there are nine 60-minutes time slots a day, and on Monday and Wednesday there are six 90-minutes time slots a day.

To handle the problem, six different sets have been defined. Which are: student groups "S", instructors "I", course sections "C", class rooms "R", timeslots-lectures "L", and a set of constrains "O". Then, the problem is formulated as P = {S, I, C, R, L, O}. Where: S = {s_1, s_2, ..., s_i} is the set of student groups, each element in S is a vector (*m*, *y*, *std_slot*) where *m*: is the major (IT, IS, GM, or BA), *y*: is the group number which will be represented by academic year of that group. This vector determines a group of students of the same major and academic year. So, for the considered college, *m* ranges from 1 to 4, and *y* ranges from 1 to 4, and std_slot is an array of timeslots that indicates when the student groups are available. I = {i_1, i_2, ..., i_j} is the set of instructors in the college at the current semester, each element in I is a vector (inst_no, inst_slot) where inst_no: is the number of a specific instructor, inst_slot: is an array indicating to timeslots when the instructor can give a lecture. C = {c_1, c_2, ..., c_n} is the set of course sections offered for the student in the current semester, each element in C is a vector (s_i, co_no, section, cap, t_j , th, ph, ph_type, ph_inst) where s_i: is the group of students, which this course is offered for, co_no: is the course number, section: is the section number of this course, cap: is the maximum capacity of the section, t_j : is the instructor of this section, th: is the number of theoretical hours of this course, ph: is the number of practical hours of this course, ph type: is the type of the practical hour it is set to 0:if the course has no practical hour, 1: if this hour is lab, set to 2 if this hour is multimedia lab, set to 3 if this hour is photo lab, and set to 4 to indicate that this hour is workshop, ph_inst: the period that the teacher must be in the lab it is set to 0: no hour in the lab, 1: half of the lab period, 2: all the lab period.

R = {r_1, r_2, ..., r_m} is the set of classrooms, each element in R is a vector (cap, d, t) where cap: is the maximum capacity of this room (number of students can be assigned to this classroom), d: is a flag set to 1 if this room has data show, t: is a flag used to determine the type of this classroom, set to 0 to indicate that the classroom is a room, set to 1 to indicate that the classroom is PC lab, set to 2 to indicate that the classroom is a multimedia lab, set to 3 to indicate that the classroom is photo lab, and set to 4 to indicate that the classroom is workshop. L= {l_1, l_2, ..., l_k} is the set of time slots of the week, each element in L is a vector (n, d) where n: is the time slot number, and ranges from 1 to 9 for days 1, 3, and 5, and ranges from 1 to 6 for days 2 and 4, and d is the day ranges from 1 to 5. Thus, L = {(1,1), (2,1),..., (9,1), (1,2), (2,2), ..., (6,2), ..., (8,5), (9,5)} where (1,1) means: the first lecture in the first working day (Sunday), and so on. Each lecture (theoretical hour) on 1, 3, and 5 is 60 minutes, so the working hour is 3 timeslots (3 hours) in these days.

On days 2 and 4 each lecture (theoretical hour) is 90 minutes, so the working hour is 2 timeslots (3 hours) in these days. For example if the course has 3 theoretical hours then it takes 3 timeslots in days 1, 3, and 5 or 2 timeslots in days 2 and 4. Except on 1 and 5 the fifth lecture is 2 timeslots because on 3 there is no fifth lecture.

O = {o1, o2, ..., oq} is the set of constraints, where oq: is the penalty weight (cost) of this constraint. Hard constraints will be assigned infinity cost, while soft constraints will assigned some constant cost to indicate the weight of violating each one. The set of solutions of such problem is a very large set. Each solution will be evaluated using an indicator to determine how much this solution is good (fitness). Our methodology will use an evolutionary algorithm to search for some optimal solution.

Parallel programming is used to implement the optimization algorithm, to get faster execution. Message-Passing Interface (MPI) library was the suitable choice for distributing such a problem on a distributed memory parallel machines, to get benefit from the available network computing resources which lead to faster execution and higher utilization.

In this work, domain decomposition is used to parallelize the algorithm. Population will be divided into subpopulations which then are distributed among various processes; each of these processes will produce a set of solutions. Then, a main process will choose the optimal solution form the results of each one. This mechanism permits finding the solution form a larger population, which means there is a chance for better solutions, and a reduction of execution time because each machine has small number of individuals [7].

To achieve the above mentioned goal the master-worker model is used where many worker processes should execute under the control of one master or root process. Each process should communicate to the root process to get the parameters, do processing and then return the result back to the root. Because the root should communicate with all processes, we will use global centralized communication technique. Also, dynamic load balancing is used because there is a need to change the population size in many cases during execution.

3. SOLUTION DEFINITION AND ALGORITHM

G= {g_1, g_2, ..., g_w}, represented in Figure 1, is the set of assigned classes "*genes*" which constitute the candidate solutions "individual".

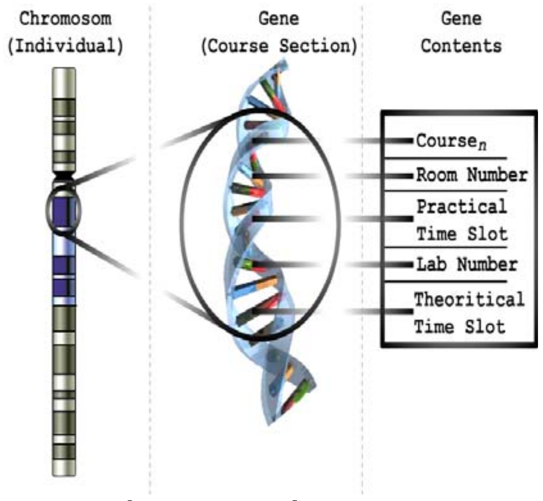


Figure 1: Chromosome and Gene representation

Every *gene* is a vector contains course number, room number, practical hour's time slot, lab number, and theoretical hour's time slot. See Figure 2.

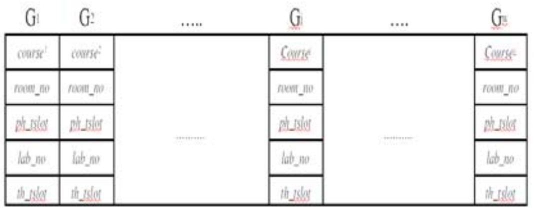
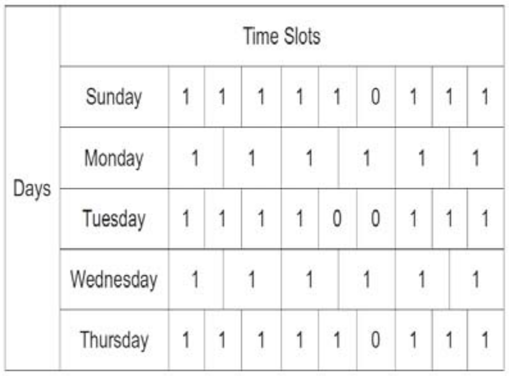


Figure 2: The individual

The search domain of the problem is a complex one, where, for each course section, suitable classroom and/or lab are chosen and time slots are assigned so that they are suitable for the student group and the lecturer. In order to reduce the search time and complexity, a heuristic is used to choose the suitable place and time for the course section so that the lecturer has no time conflicts. Then, the search is dedicated for optimizing the solution considering the hard and soft constraints for the students and only the soft constraints for the lecturers. The individual is constructed in a way that every individual from the set of solutions has different number of options when choosing timeslot and classroom; so that, course section in the individual G_1 has the maximum number of options available when selecting a classroom and a timeslot, course section in the individual G_2 has 1 option less than those were available for the individual G_1, and the last one, gene G_w has the least number of option.

For experimentation purposes, the initial parameter used as follows: 14 classrooms and 7 labs available. The seven days of the week, are divided into 41 theoretical classes (1 hour each) and 13 practical classes or labs (3 hours each). Which means, there are (41*14) = 574 theoretical classes, and (13*7) = 91 practical classes along the week as shown in Figure 3. Where 1 means time slot is available and 0 means time slot is not available.



Days	Time Slots									
	Sunday	1	1	1	1	1	0	1	1	1
	Monday	1	1	1	1	1	1			
	Tuesday	1	1	1	1	0	0	1	1	1
	Wednesday	1	1	1	1	1	1			
	Thursday	1	1	1	1	1	0	1	1	1

Figure 3: The timeslots

The final solution should satisfy a set of constraints. These constraints are divided into hard constraints, which must be satisfied (violation cost is infinity), and soft constraints which should be satisfied (violation cost is a constant relative to the constraint importance). Hard constraints are:

- A teacher must not have more than one class at any given time slot.
- At any given time slot no student group can have more than one class.
- An instructor can be assigned classes only in his available time, like part timers.
- A room must not assigned more than one class at any given time slot.
- The number of students in any lecture should be less than or equal the maximum capacity of the classroom.

Soft constraints are:

- Students should not have large number consecutive classes.
- Instructors should not have long free time between lectures.
- Students should not have long free time between lectures.
- Instructors should not have very late classes daily.
- A group of students should not have consecutive classes at different large distance locations.

3.1 Parallel Evolutionary Algorithm

There are two main reasons for parallelizing an evolutionary algorithm: the first is to minimize the execution time by distributing the computation. And the second is to get benefit from the parallel setting, in analogy with the natural parallel evolution of spatially distributed populations [13]. In the parallel evolutionary algorithms, computation and communication are the most time consuming factors. These two factors have an inverse relationship, so if the communication increases, the computation time deceases and vice-versa [8],[1]. In [4] they noticed that the main drawback of the sequential algorithm is the large execution time. This prevents continuing the search for better solutions regarding the willingness of students and staff.

Regarding our algorithm, the number of course sections is stored in a parameter called *numOfGenes* which determines how many *genes* will be in each *chromosome* or individual, and then a dynamic array is built for the class rooms, students groups, instructors, and courses sections. The number of individuals on each population is defined at the beginning of the program called *popSize*, also a variable called *numGener* is defined to determine the total number iterations for the evolutionary algorithm. The evolutionary algorithm and its functions are shown below:

```
Initialize_Population; // Initial population.
t←0;
while t < numGener - 1 do
Evaluate_Population; // Evaluate fitness
Do_Selection; // Select new population
Do_Mutation; // Mutate the new population.
t←t + 1;
end while
Evaluate_Population; //Evaluate fitness
```

The algorithm contains a number of functions:

First: the function Initialize_Population, creates the initial population of individuals based on the population size (*popSize*).

Second: the function Evaluate_Population, evaluates the fitness of the population. After constructing the population, fitness evaluation is achieved for each individual in the population. Population cost is measured as the sum of cost for each gene in the individual. The cost is determined by scanning the *genes* of the individual orderly for violations to hard or soft constraints; for example a hard constraint violation happens if, in some *gene*, a course section is assigned in an occupied time slot for the teacher, room, or a student group. On the other hand, a soft constraint violation happens if a course section is assigned in a time slot that makes many consecutive lectures for the teacher.
The fitness is calculated for the individual x by the simple function

$$f(x) = \frac{1}{\cos t(x) + 1}$$

Where *cost(x):* is the sum of the hard and soft costs of the individual *x*. The value of this function is important in determining whether this individual will pass to next step (new population) or not. The hard cost is multiplied by a very big constant "*infinity*" and added to a hard cost array for the individual, and the soft cost is represented through its priority by a smaller constant.

Third: the function Selection, selects a new parent population for the next generation based on the fitness of current individual the probability to select an individual is increased as its fitness increases

Fourth: the function Mutation, performs a random modification on the *genes* in the individual; this change can be done with a probability. Every individual in the population is ordered according to the gene's hard and soft costs, where genes with cost sum equal to zero are put at the beginning of the individual. Then the mutation starts from the first gene whose cost is greater than zero.

Load balancing is a very important issue for maximizing utilization and speedup. The load balancing between the processors is achieved using the algorithm called Exploitation of the Fastest Processor (EFP) from [3].

4. EXPERIMENTAL RESULTS

This section presents the testing environment in terms of software, hardware, and data, then obtained results is explained.

4.1 Testing Environment

The application is implemented using C programming on a workstation with dual processor Intel (R) Xeon^{TM} at 3 GH, 512 KB cache, 2 GB memory and runs under Linux. In our experiments real data available on the database of Palestine Polytechnic University is used. The data is coded into numbers and stored in text files which are used as inputs to the program, see Figure 4.

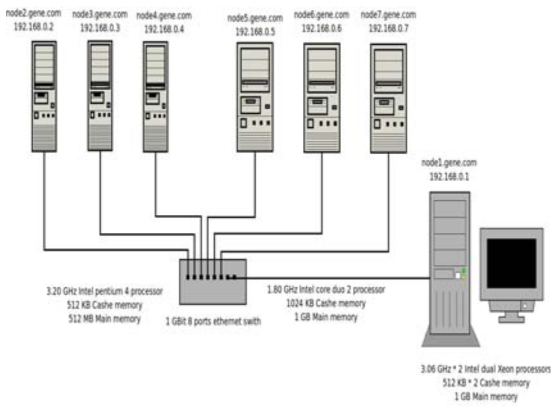


Figure 4: Testing environment

4.2 Results

A series of experiments is listed in this section to show the relationship between variables in diagrams, and the analysis of the results. Figure 5 shows that execution time of the parallel multi-objective algorithm decreases efficiently as the number of processors increases, where the population size is 300 and the number of generations is 20.

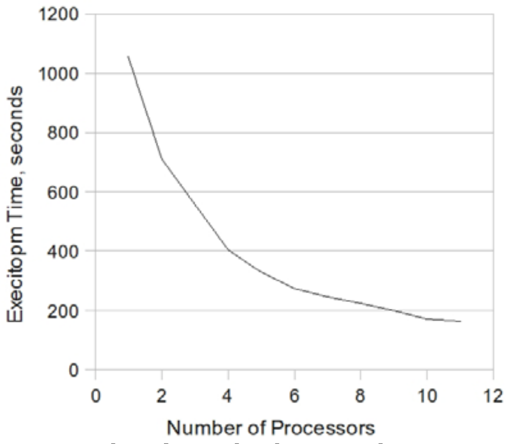


Figure 5: The relationship between the execution time and the number of processors.

Figure 6 shows that the average fitness is enhanced over generations, where the population size is 700 and the number of generations 700.

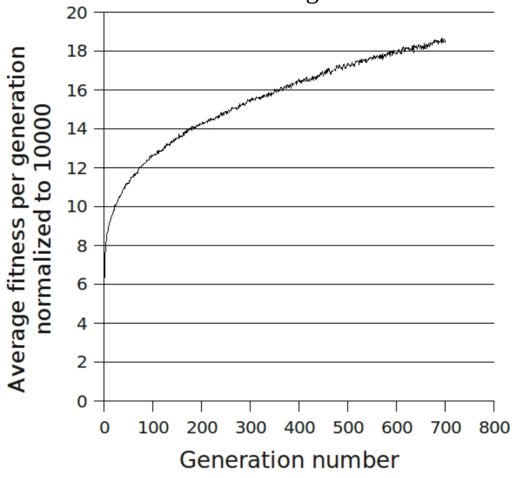


Figure 6: The average fitness evolution

Figure 7 shows that the fitness of the best individual is enhanced over generations.

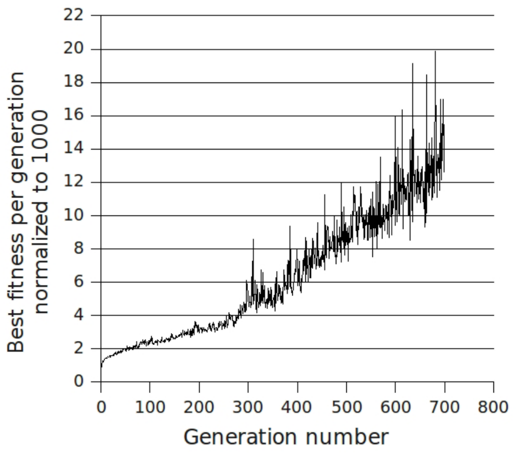


Figure 7: The best fitness evolution

Figure 8 shows the relation between number of generations and the best hard cost per generation, where the average hard cost is decreased efficiently.

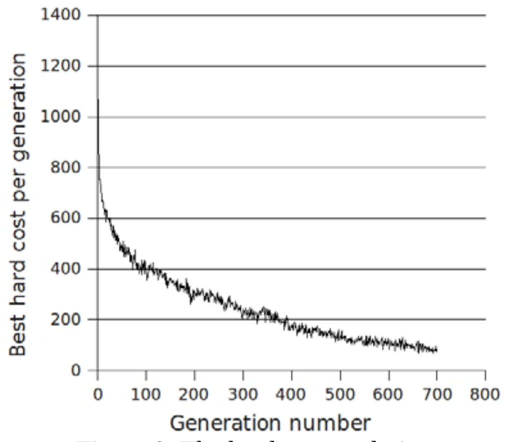


Figure 8: The hard cost evolution

In the same way, Figure 9 shows the relation between number of generations and the best soft cost per generation, where the average soft cost is decreased also efficiently.

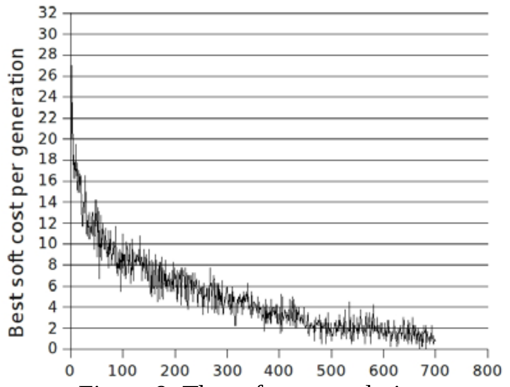


Figure 9: The soft cost evolution

As a result the multi-objective optimization is achieved, and both objectives: zero hard cost and minimum soft cost can be satisfied. The above figures show that execution time is reduced, in a way, allowing for more exploration to get better solutions in a very complex search space.

5. CONCLUSION AND FUTURE WORK

In this research a new methodology for solving university course scheduling is proposed and implemented using real data set from the College of Administrative Sciences and Informatics at Palestine Polytechnic University. We have reduced the soft cost without affecting the hard cost by using multi-objective optimization. In addition, we have speeded up the algorithm by distributing it among a cluster of machines, overcoming the main problem in [4] regarding the large execution time.

Also, the communication process consumes small time compared with computational process, most of execution time consumed by mutation function. Furthermore, the load balancing algorithm becomes more complex under the random behaviour of the algorithm, because it is not possible to determine which node will reach the goal faster.

As a future work, it could be good idea to enhance the algorithm by trying other methods and evolutionary techniques such as particle swarm optimization "PSO" algorithm. Also, to consider more soft constraints to better satisfy the willingness of the students and teachers. Enhancement of the load balancing algorithm could be another contribution.

REFERENCES:

[1] Abraham, A., L. Jain, and R. Goldberg, 2005. Evolutionary multiobjective optimization: theoretical advances and applications. Springer Verlag London Limited, Printed in USA, ISBN 1852337877.

[2] Adeyemo, J.A., 2011. Reservoir operation using multi-objective evolutionary algorithms-a review. Asian J. Sci. Res., 4: 16-27.

[3] Aldasht, M.M., J. Ortega, and C.G. Puntonet, 2007. Dynamic load balancing in heterogeneous clusters-exploitation of the processing power," 2nd PICCIT, Hebron, Palestine.

[4] Aldasht, M.M., M.H. Saheb, S.I. Adi, and M.M. Qopita, 2009. University course scheduling using evolutionary algorithms, 4th ICCGI-IARIA, Cannes, France.

[5] Belkadi, K., M. Gourgand, M. Benyettou, and A. Aribi, 2006. Sequential and parallel genetic algorithms for the hybrid flow shop scheduling problem. J. Applied Sci., 6: 775-778.

[6] Benedict, S. and V. Vasudevan, 2008. Improving scheduling of scientific workflows using tabu search for computational grids. Inform. Technol. J., 7: 91-97.

[7] Datta, D., K. Deb, and C.M. Fonseca, 2006. Solving class timetabling problem of IIT-Kanpur using multi-objective evolutionary algorithm. Technical Report, Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur - 208 016, India.

[8] Ghaemi, S., M.T. Vakili, and A. Aghagolzadeh, 2007. Using a genetic algorithm optimizer tool to solve university timetable scheduling. 9th International Symposium on Signal Processing and Its Application. pp. 1-4, ISBN: 978-1-4244-0778-1

[9] Grama, A., and V. Kumar, 1993. A survey of parallel search algorithms for discrete optimization problems. ORSA JOURNAL ON COMPUTING

[10] Opera, M., 2006. Multi-agent system for university course timetable scheduling. The 1st International Conference on Virtual Learning, pp. 231-237.

[11] Park, H.H., A. Grings, M. Santos, and A. Soares, 2008. Parallel hybrid evolutionary computation: Automatic tuning of parameters for parallel gene expression programming. Applied Mathematics and Computation,Volume 201, Issues 1-2, 15 July 2008, Pages 108-120.

[12] Pongcharoen, P., W. Promtetn, P. Yenradee, and C. Hicks, 2008. Stochastic optimisation timetabling tool for university course scheduling. journal International Journal of Production Economics, Elsevier. Volume 112, Issue 2, pp. 903-918.

[13] Tomassini, M., 1999. Parallel and distributed evolutionary algorithms: A review. Institute or computer science, University of Lausanne, 1015 Lausanne.

[14] Zhang, L., and S. Lau, 2005. Constructing university timetable using constraint satisfaction programming approach. Proceedings of CIMCA-IAWTIC'06 - Volume 02, pp. 55-60. ISBN:0-7695-2504-0-02.